# SNOWFLAKE CORTEX AI CHATBOT A robot touching a server AI-generated content may be incorrect.DOCUMENTATION

**Table of Contents**

[1. Introduction to Snowflake AI Chatbot 3](#_Toc210051389)

[2. Architecture of Snowflake Chatbot 3](#_Toc210051391)

[3. Step by step implementation 7](#_Toc210051392)

# **1. Introduction to Snowflake AI Chatbot**Snowflake Cortex is a set of AI tools built into Snowflake that uses advanced technology (like large language models) to help you understand and use your data better. One of these tools is the Cortex Agent, which acts like a coordinator. It can look at structured data (organized in tables) and unstructured data (like PDFs or text files) and use them together to give you answers or insights.

# To make a Cortex Agent work, you need to prepare your data, set up some tools it can use, and create an interface to interact with it. We will walk through every step, including building a Streamlit application for a user-friendly chat experience that directly references your data paths.

# 

# **2. Architecture of Snowflake Chatbot**

A diagram of a software flow

AI-generated content may be incorrect.

The diagram provides a high-level overview of the Snowflake AI chatbot's architecture, focusing on how user queries are processed from input to output. It highlights the flow through various layers, with numbered steps (1 through 6) indicating the sequence. This is based on the Snowflake ecosystem, incorporating tools like Cortex Analyst, Snowpark, and Cortex functions, as seen in typical implementations (e.g., your app.py code).

**1. Streamlit**

* **Role**: This is the frontend interface layer where users interact with the chatbot. It serves as the entry point for submitting queries and displaying responses.
* **How it Works**:
  + Users enter natural language questions (e.g., "What is the Total Burdened Cost?") via a chat input field (Step 1 in the diagram).
  + It handles user authentication (e.g., via Snowflake credentials) to ensure secure access.
  + The component manages session state, including chat history, model selection (e.g., mistral-large), and visualization options (e.g., charts via Plotly).
  + Once processed, responses (text summaries, data tables, or charts) are rendered back here (Step 6), making the experience conversational and user-friendly.
* **Integration**: Queries are forwarded to the Rest API for backend processing, and it receives formatted answers from downstream components. Streamlit acts as the "bookends" of the flow, encapsulating the user experience without handling core AI logic.

**2. YAML Semantic Model**

* **Role**: This is a configuration file that defines the structure, rules, and metadata for query handling, acting as the "brain" for understanding data relationships.
* **How it Works**:
  + It's a YAML file (e.g., GRANTS\_CHATBOT.yaml) stored in Snowflake Storage, containing definitions like table schemas, relationships, metrics, and business logic (e.g., for grants data in the GS schema).
  + It provides context for natural language processing, such as mapping user terms (e.g., "total cost") to database fields or calculations.
  + The model supports query classification (structured vs. unstructured) and ensures accurate SQL generation by referencing predefined entities, joins, and filters.
  + It's used throughout the flow to maintain consistency, e.g., applying filters like fiscal years or versions in budgeting queries.
* **Integration**: Connected to Question Understanding, SQL Generation, and Answer components, it feeds metadata to LLMs and tools, enabling precise routing and response generation without hardcoding logic.

**3. Rest API**

* **Role**: This serves as the communication gateway between the frontend (Streamlit) and the backend AI services.
* **How it Works**:
  + It receives the query from Streamlit (Step 2) and makes API calls to Snowflake's Cortex endpoint (e.g., /api/v2/cortex/agent:run).
  + The API handles authentication (using Snowflake tokens) and payload construction, including the query, semantic model reference, and tools (e.g., cortex\_analyst\_text\_to\_sql).
  + It supports timeouts and error handling to ensure reliable transmission.
  + Responses from downstream processing are parsed (e.g., SSE events) and routed back.
* **Integration**: Acts as a bridge to Question Understanding, forwarding queries for intent analysis. It's crucial for scalability, allowing integration with other frontends if needed.

**4. Question Understanding**

* **Role**: This component analyzes the incoming query to determine its type and intent, enabling intelligent routing.
* **How it Works**:
  + Powered by Cortex Analyst and LLMs, it parses natural language (Step 3) using patterns and semantic models to classify queries (e.g., structured like "Top 10 Projects by Cost" vs. unstructured like "Explain Total Burdened Cost").
  + It incorporates chat history for context (e.g., summarizing prior messages to refine the query).
  + For special cases (e.g., greetings or invalid queries), it triggers fallbacks like sample suggestions.
  + It detects keywords for routing: structured queries go to SQL paths; unstructured to Cortex functions.
* **Integration**: Outputs to SQL Generation for data queries or directly to Answer for non-SQL needs, using the semantic model for accuracy.

**5. SQL Generation**

* **Role**: Converts natural language queries into executable SQL for structured data retrieval.
* **How it Works**:
  + Using Cortex Analyst and LLMs (Step 4), it generates SQL based on the semantic model (e.g., translating "Top funding sources" to a SELECT query with GROUP BY and ORDER BY).
  + It validates and refines SQL to match database schema (e.g., tables in GRANTS.GS).
  + Snowpark is implicitly involved here for execution preparation, handling any custom logic or functions.
  + If the query is invalid, it falls back to suggestions or rephrasing.
* **Integration**: Feeds generated SQL to Data Tables for execution, with results flowing to the Answer component.

**6. Answer Using Data and Semantic Model**

* **Role**: Formats and enriches raw results into user-friendly responses.
* **How it Works**:
  + Combines data from queries (Step 5) with semantic context to create summaries (e.g., using Cortex COMPLETE for natural language explanations or SUMMARIZE for condensing results).
  + It generates visualizations (e.g., charts) and handles edge cases like empty results.
  + Incorporates business logic from the semantic model for meaningful insights (e.g., year-over-year calculations).
  + Ensures responses are concise and actionable, avoiding raw data dumps.
* **Integration**: Pulls from SQL Generation or Cortex functions, then sends polished output back via Rest API to Streamlit.

**7. Data Tables**

* **Role**: Represents the structured data storage and execution environment in Snowflake.
* **How it Works**:
  + This is the Snowflake database (e.g., GRANTS with GS schema), where SQL queries are run to fetch data (e.g., awards, budgets, costs).
  + It supports scalable querying on large datasets, with security features like role-based access.
  + Results are returned as dataframes for further processing (e.g., via Snowpark).
* **Integration**: Accessed after SQL Generation, providing raw data to the Answer component.

**8. LLM Models**

* **Role**: The foundational AI engines powering interpretation, generation, and summarization across the system.
* **How it Works**:
  + Specific models like Mistral Large (for general NLP), Arctic (Snowflake-native for efficiency), and Llama (for advanced reasoning) are selectable.
  + They handle tasks like query understanding, SQL creation, and response summarization via Cortex integrations.
  + Models process prompts with context (e.g., chat history) to ensure relevance.
* **Integration**: Underpins Question Understanding, SQL Generation, and Answer components, with the semantic model enhancing their accuracy.

# **3. Step by step implementation**

**Prerequisites:**

Before you start, make sure you have:

* A Snowflake account with the right permissions to create things like databases, tables, and functions.
* Some experience with Python (to write a function for splitting text and building the Streamlit app).
* A basic understanding of YAML (a simple file format we’ll use later).
* A Snowflake warehouse (a computing resource) ready to use.
* Streamlit installed (pip install streamlit) and a Python environment set up.
* Snowflake Python connector installed (pip install snowflake-connector-python) to handle Snowflake connections.

**Step 1 – Snowflake Login**

Log in to Snowflake with appropriate credentials.

**Step 2 – Creating a Database and Schema**

**A screenshot of a computer

AI-generated content may be incorrect.**



* Goto **Catalog -> Database Explorer**.
* Next step is to create a Database by clicking on the **“+ Database”** icon on the top right corner. Give a name to the Database.
* In a similar way, create a Schema in the Database by clicking on the newly created Database.

**Step 3 – Creating tables into the schema**

Once you are into the schema, we need to create tables and can insert the data using the various methods.

A screenshot of a computer

AI-generated content may be incorrect.



Once you click on the table button, you will see the following options:

A screenshot of a computer

AI-generated content may be incorrect.

You can choose the option you want on how you want to load the data.

Or you can create tables by writing SQL queries in Worksheets:

A screenshot of a computer

AI-generated content may be incorrect.



In the worksheets you can write SQL queries and load the data.

**Step 4 – Creating a Stage**

To create a stage, we need to go to the schema we created. Click on the “**Create**” button on the top right corner and then click on the “**Stage**”.

A screenshot of a computer

AI-generated content may be incorrect.



After selecting “**Snowflake Managed**” you will see the following dialog box:

A screenshot of a computer

AI-generated content may be incorrect.



Assign a name to the **Stage** and then click on the “**Client-side encryption**”**.** The stage will be created.

**Step 4 – Creating a Semantic model**

A semantic model is a file that explains your table to Snowflake in a simple format called YAML. You can make it yourself or use a tool to help.

**Option 1:** **Make It Yourself**: Create a file called semantic\_model.yaml with this content:

name: Sales Data Model

description: Semantic model for sales data

tables:

- name: sales\_data

description: Table containing sales transactions

base\_table:

database: my\_database

schema: public

table: sales\_data

dimensions:

- name: product

synonyms: [item, merchandise]

description: The product sold

expr: product

data\_type: VARCHAR

- name: date

synonyms: [transaction\_date]

description: The date of the sale

expr: date

data\_type: DATE

measures:

- name: revenue

description: Total revenue from sales

expr: revenue

data\_type: NUMBER

- name: quantity

description: Quantity of products sold

expr: quantity

**What it does**: This file tells Snowflake what your table is about, including column names, what they mean, and other ways to say them (synonyms).

Once you are in the newly created **Stage** you can see the “**+Files**” button on the top right corner. Once you click on that button, it will pop up a dialog box:

A screenshot of a computer

AI-generated content may be incorrect.

You can browse the file or paste the path of the file.

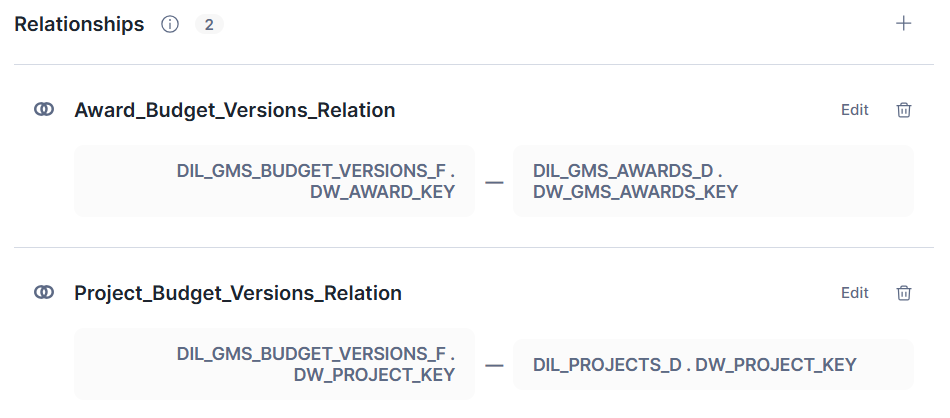
**Option 2:** **Use the Generator:** Snowflake has a tool in Snow sight:

o Go to **AI & ML - > Cortex Analyst -> Try -> Create new**.

A screenshot of a computer

AI-generated content may be incorrect.

Assign a name, Select the required tables and the required columns. The tool will make a draft YAML file for you to edit.



Provide the relationship between the tables and provide some verified queries that help in answering the questions.

Goto **Verified queries -> add new -> Add a question and write the SQL query -> create**.

Put the YAML file into a Snowflake stage.

**Step 5- Building a Streamlit Chat Interface**

To make the Cortex Agent easy to use, we’ll create a chat interface using Streamlit, a Python library for building web apps. This interface will let users type questions and get answers from the Cortex Agent by referencing the paths to structured data.

Create a Python file called **app.py**

“

import streamlit as st

import snowflake.connector

import json

# Streamlit page configuration

st.set\_page\_config(page\_title="Snowflake Cortex Agent Chat", layout="wide")

# Initialize session state for chat history

if "messages" not in st.session\_state:

st.session\_state.messages = []

# Snowflake connection (replace with your credentials)

def get\_snowflake\_connection():

conn = snowflake.connector.connect(

user="your\_snowflake\_user",

password="your\_snowflake\_password",

account="your\_snowflake\_account",

warehouse="my\_warehouse",

database="my\_database",

schema="public"

)

return conn

# Function to call Cortex Agent

def call\_cortex\_agent(query):

conn = get\_snowflake\_connection()

cursor = conn.cursor()

try:

# Construct the Cortex Agent query using data paths

agent\_query = f"""

SELECT CORTEX\_AGENT\_CHAT(

ARRAY\_CONSTRUCT(

OBJECT\_CONSTRUCT(

'tool\_spec',

OBJECT\_CONSTRUCT(

'name', 'sales\_data\_model',

'type', 'cortex\_analyst\_text\_to\_sql'

)

),

OBJECT\_CONSTRUCT(

'tool\_spec',

OBJECT\_CONSTRUCT(

'name', 'document\_search',

'type', 'cortex\_search'

)

)

),

OBJECT\_CONSTRUCT(

'sales\_data\_model',

OBJECT\_CONSTRUCT(

'semantic\_model\_file', '@my\_stage/semantic\_model.yaml'

),

'document\_search',

OBJECT\_CONSTRUCT(

'name', 'my\_database.public.my\_search\_service',

'max\_results', 5,

'title\_column', 'file\_name',

'id\_column', 'chunk\_id'

)

),

'{query}'

) AS response

"""

cursor.execute(agent\_query)

result = cursor.fetchone()[0]

return json.loads(result) if result else {"error": "No response received"}

except Exception as e:

return {"error": f"Error: {str(e)}"}

finally:

cursor.close()

conn.close()

# Streamlit UI

st.title("Snowflake Cortex Agent Chat")

st.write("Ask questions about sales data or documents, and the Cortex Agent will respond!")

# Display chat history

for message in st.session\_state.messages:

with st.chat\_message(message["role"]):

st.markdown(message["content"])

# Input box for user query

if prompt := st.chat\_input("Type your question here..."):

# Add user message to chat history

st.session\_state.messages.append({"role": "user", "content": prompt})

with st.chat\_message("user"):

st.markdown(prompt)

# Get response from Cortex Agent

with st.chat\_message("assistant"):

with st.spinner("Thinking..."):

response = call\_cortex\_agent(prompt)

if "error" in response:

st.error(response["error"])

else:

# Display the response (customize based on actual response structure)

answer = response.get("response", "No response provided")

st.markdown(answer)

st.session\_state.messages.append({"role": "assistant", "content": answer})

”

**Explanation of the Streamlit Code**

**Imports**: The code uses streamlit for the UI, snowflake.connector for Snowflake connectivity, and json for parsing responses.

**Snowflake Connection:** The get\_snowflake\_connection function establishes a connection to Snowflake using your credentials. Replace

your\_snowflake\_user, your\_snowflake\_password, your\_snowflake\_account, my\_warehouse, my\_database, and public with your actual Snowflake details.

**Cortex Agent Call:** The call\_cortex\_agent function executes a SQL query using CORTEX\_AGENT\_CHAT, a hypothetical Snowflake function that processes the agent request (since direct path-based API calls without authentication are assumed here for simplicity). It specifies:

o **Tools:** sales\_data\_model (for structured data) and document\_search (for unstructured data).

o **Resources:** Paths to the semantic model (@my\_stage/semantic\_model.yaml) and Cortex Search service (my\_database.public.my\_search\_service).

o **Query:** The user’s question.

**Streamlit UI:**

o Displays a title and subtitle.

o Maintains a chat history using st.session\_state.

o Shows past messages in a chat format.

o Users type questions in a chat input box.

o Submits the question to the Cortex Agent, shows a “Thinking...” spinner, and displays the response.

**Running the App:**

* Before you run the app, create a virtual environment and install all the dependencies.

**Creating a Virtual Environment:**

Make sure you have Python installed on your machine. [Note: Make sure your python version is a stable version and at least 2 versions lesser than the latest version.]

>python -m venv <env\_name>

**Activate your Virtual Environment:** >venv\Scripts\activate

>Install dependencies: pip install streamlit requests snowflake-connector-python snowflake-snowpark-python pandas plotly

A black screen with white text

AI-generated content may be incorrect.

**Execute the code:**  
>streamlit run app.py