DataStax and Google Cloud GenAI Workshop Silicon Valley

https://dtsx.io/ds-google-sv

Welcome!

We're looking forward to a fun few hours with you. Below are some resources and instructions you'll want to follow to get the most out of this workshop.

Resources

1. Sign Up for Astra Using This Link if you haven't already; work email addresses are best!
2. View the LangFlow Documentation
3. Please Star the LangFlow GitHub
4. Optionally, join the Zoom Meeting.
   NOTE: please ensure you are muted, video off, and if you are in person turn the volume on your speakers all the way down or wear headphones.

Sign up For Astra

Relevant Blog Entries

LangFlow Acquisition Announcement
Blog: What is a Vector Database?
Blog: What is RAG?

Getting Started

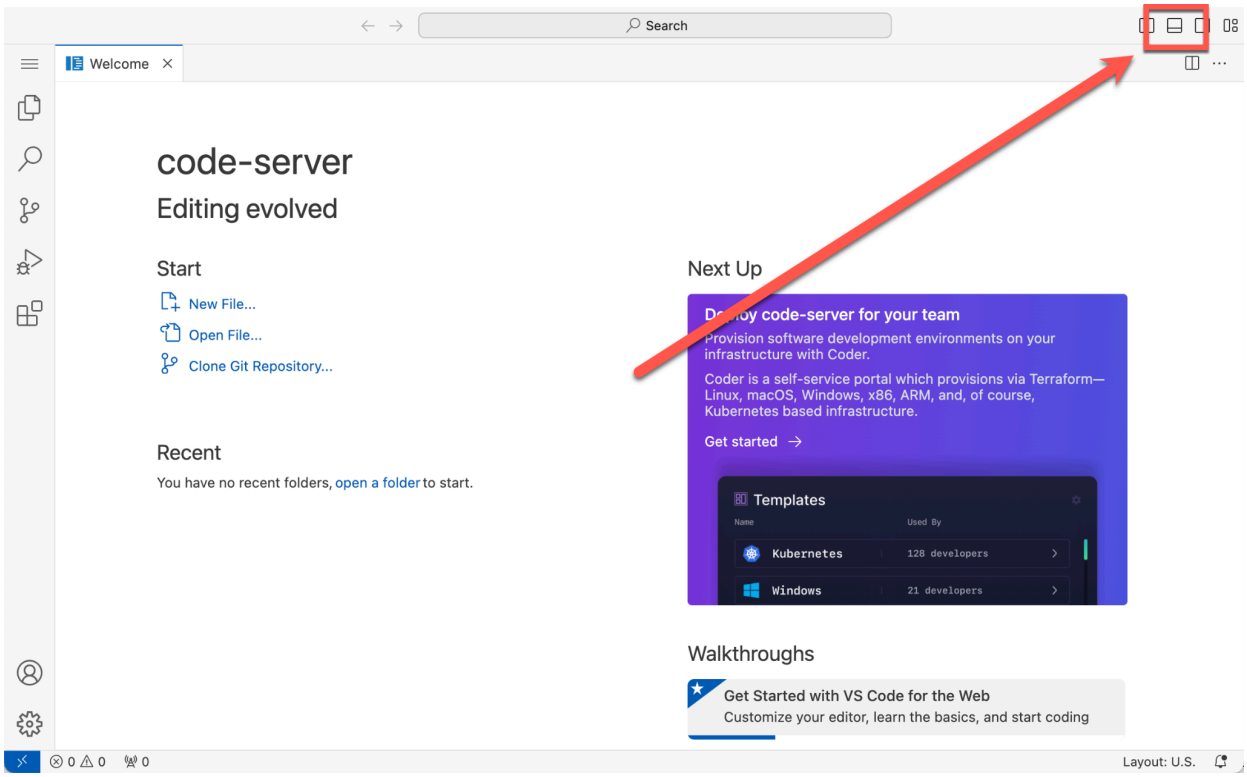
First, please use the Google Chrome browser for all parts of this workshop, if possible.

Step 1: Sign Up
Register for the workshop at https://events.oneblink.ai.
Your event code is **6122596**

Step 2: Verify Your Email
Check your inbox for a validation code and follow the instructions to confirm your email.
You will need to provide the verification code and the event code, **6122596**

Step 3: Access Your Workshop Environment
Enter the workshop by logging in with the credentials provided, giving you exclusive
access to a tailored sandbox environment for hands-on learning.

You'll be presented with a VSCode editor. Set up the editor however you prefer. Choose "Mark Done" to skip.
Once you're in the editor, click the button to split-open the lower half of the screen. This will give you a terminal.



Identify Your Google Project Name

This editor is running on a private Google Cloud project just for you. For some parts of this workshop you'll need to know the name of that project. Copy-Paste this command:

curl -H "Metadata-Flavor: Google" http://metadata.google.internal/computeMetadata/v1/project/project-id; echo

Note the output. If you misplace it you can re-run the command above in your code editor terminal.

Install Langflow

Next, we need a tool to manage our Python environments. We'll use Conda. Copy-paste these commands into the terminal:

```
mkdir -p ~/miniconda3
```

```
wget
https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x
86_64.sh -O ~/miniconda3/miniconda.sh
bash ~/miniconda3/miniconda.sh -b -u -p ~/miniconda3
rm -rf ~/miniconda3/miniconda.sh
~/miniconda3/bin/conda init bash
~/miniconda3/bin/conda init zsh
```

At this point we have to re-initialize the terminal so type:

```
exit
```

The terminal will close. Re-open it just like before. Now we'll create a Python environment, install some Google Cloud requirements, and then install langflow.

```
conda create --name langflow python=3.10
conda activate langflow
pip install google-cloud-aiplatform>=1.38.0
python -m pip install langflow --pre --force-reinstall
```

Now, start langflow:

```
langflow run --host 0.0.0.0
```

When you are prompted to open a browser window, cancel. This is trying to set up a proxy through VSCode and it will not work with Langflow. Note that we're running the latest pre-release version of Langflow. You're going to see some stack traces fly by your console but you can safely ignore them.

Copy the URL with your environment's IP address in it from the editor window, open a new tab, and paste into the address bar. Change the port from :8080 to :7860 and hit enter. You should see the Langflow project manager.

The Basic, Basics

Let's start by creating a new project.

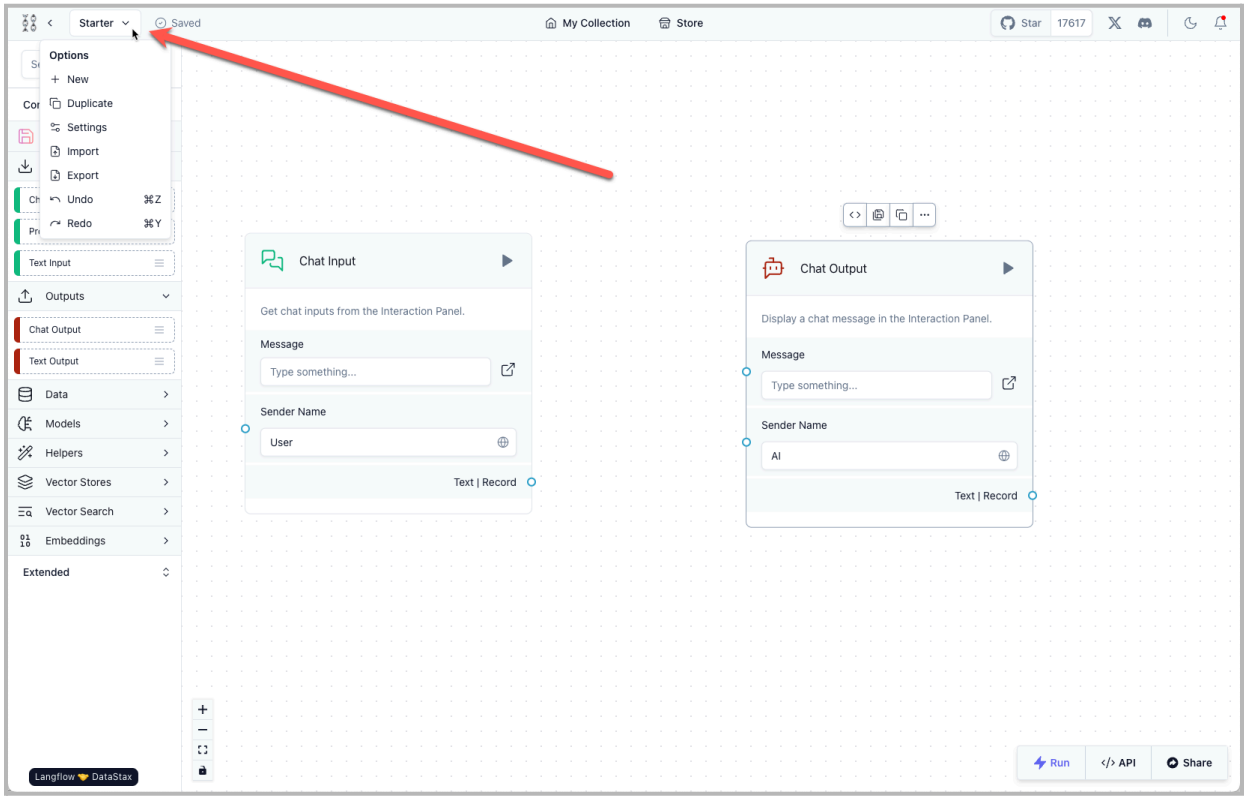
1. From the list of project templates, choose "Blank Flow"
2. Take a look through the list of Core and Extended components on the left.
3. Under "Inputs" drag the "Chat Input" component onto the canvas
4. Under "Outputs" drag the "Chat Output" component onto the canvas
5. Connect the "Text|Record" connector of the Chat Input component to the "Message" connector of the Chat Output component by dragging a line from one connector to the other.
6. Click the "Run" button in the lower left hand corner of the window. You should see a chat-style interface.
7. When you send a message, what do you think you'll see?
8. You should see whatever you typed echoed back. Input to Output.

Now let's add some intelligence to the middle...

Your own Chat UI

Download this file; note where it is saved.
We've modified the Vertex AI component to make it easier to use your Google Project. This file contains the modified VertexAI component.
Continuing from the previous exercise, in the top-left corner of the UI is a drop-down menu. Choose "Import" and then select the "Vertex AI LLM Component.json" file you downloaded above.

pip install langchain-google-vertexai



Now you should have a new component on your canvas. Click the line between your Chat Input and Chat Output components and then press the delete key on your keyboard. The connection should disappear.

- Connect the Chat Input “Text|Record” to the “Input” of the Vertex AI component
- Connect the “Text” output of the Vertex AI component to the “Message” input of the Chat Output component.
- Populate the “Project” field of the Vertex AI component with your Google Project Name
- Click the “Run” button in the lower right corner
- In the chat UI, type “Tell me a story.” You should get a story
- Ask a few other questions to demonstrate how Gemini Pro 1.5 can be.

At this point you have a playground interface to Gemini Pro 1.5.

Prompt Engineering

Langflow gives you some pretty simple ways to experiment with prompts to see which technique delivers the best results for your use case. Continuing with the canvas in the previous exercise,

- Delete the Chat Input component.
- Under “Inputs” drag the “Prompt” component onto the canvas
- Click the icon on the Prompt component to open the prompt editor
- We are going to start with a “Zero-Shot” prompt based on this example. Read this page and grab the text in the “Prompt” section. Paste that into the prompt editor
- Connect the “Text” of the Prompt component to the “Input” of the Vertex AI component
- Click the Run button. You’ll notice that the Chat UI has changed. Since we deleted the “Chat Input” component, we can’t input any text here. However, we can still run the flow as is by clicking the lightning bolt icon.
- Experiment with changing the text of the prompt or other parts of the instructions.

Add Data from a URL to implement Prompt Chaining

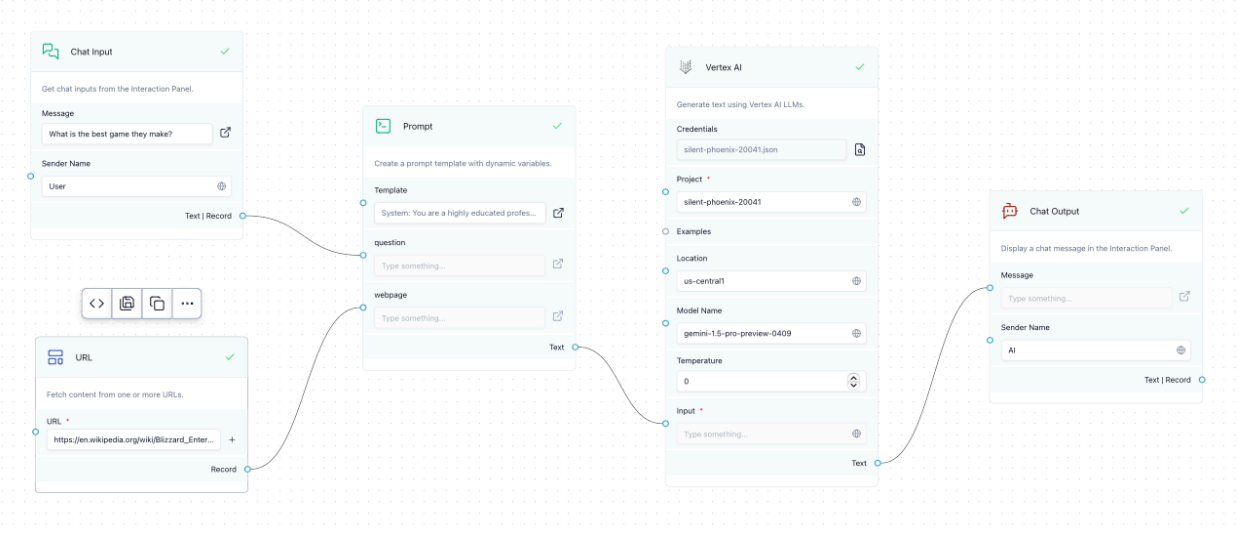
It is pretty common to want to provide specific data for the LLM to process so that we can ask questions about the data or otherwise use the LLM to operate on the data. To accomplish this, we just make a more sophisticated prompt. Let’s modify our prompt to make a placeholder for information from a webpage.

- Click the icon on the Prompt component to open the prompt editor
- Copy-Paste this simple prompt:

```
System: You are a highly educated professor.  You always respond with the most obscure technical words you know.  You give long answers.  Please answer the user’s question about the following webpage (delimited by ####).

####
```

```
{webpage}
####

User: {question}

Response:
```

- Click “Check & Save”
- Drag the “Chat Input” component back to the canvas
- Connect it’s output to the “question” input of the Prompt component
- Drag the “URL” component onto the canvas (find it in the “Data” category on the left)
- Connect its output to the “webpage” input of the Prompt component
- Pick a webpage, any webpage. The more content, the better. Wikipedia pages usually work well.
- Copy-Paste the URL for that page into the URL field of the URL component.
- You should end up with something that looks like this:



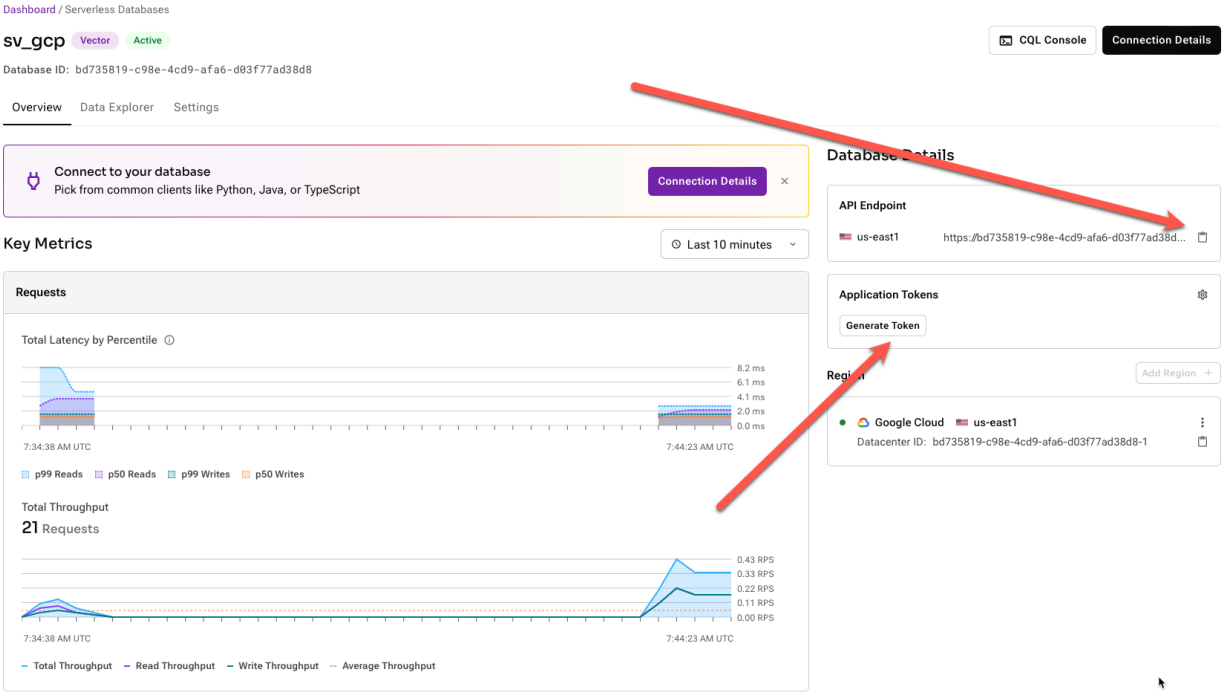
- Click the Run button and ask a question about the page.
- Experiment with the prompt. Change the persona or the way they talk and see how the responses change. Have a little fun with it. Here are some ideas.

Challenge: Read this page about Prompt Chaining. How could you add additional components (or duplicate them) to implement this technique?

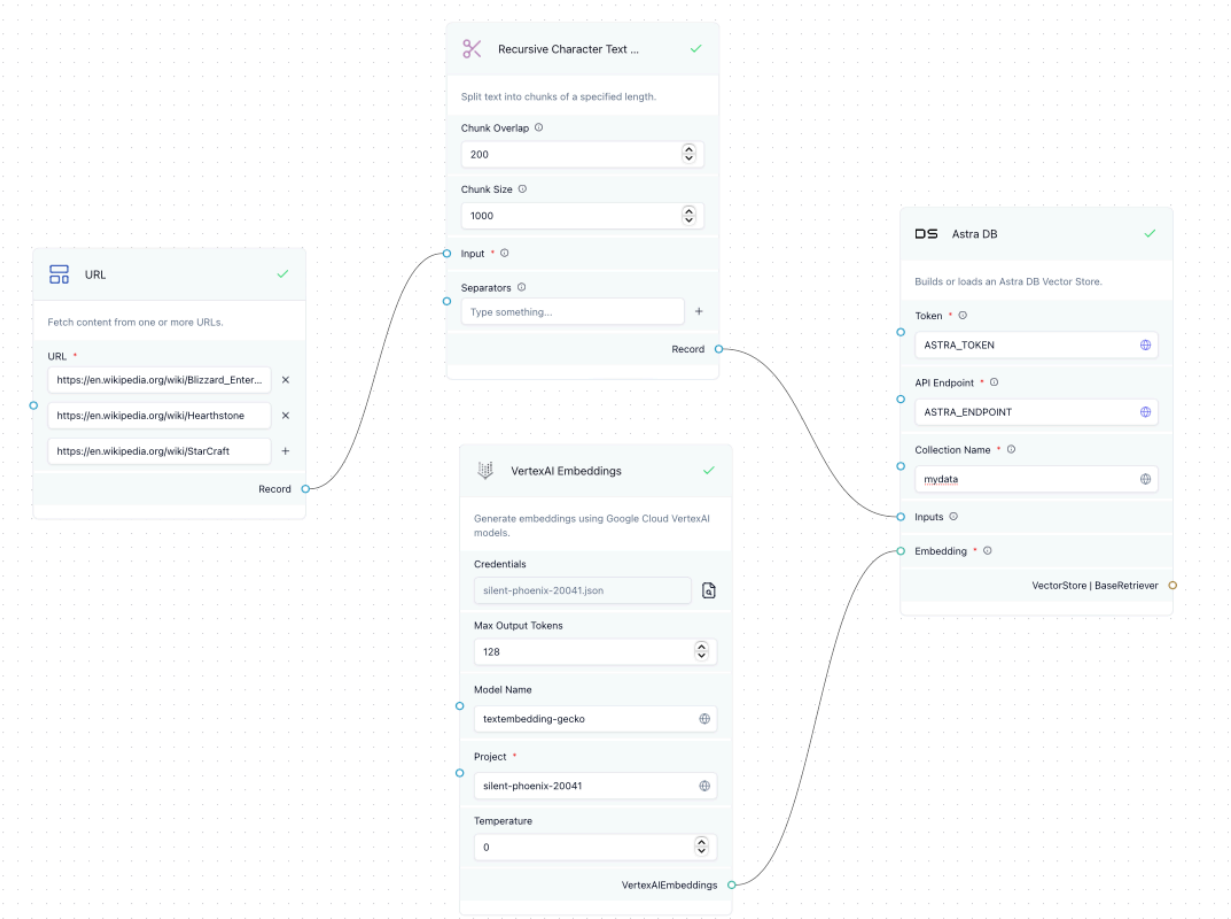
Build a data ingestion pipeline

We now have a convenient way to work with one or a few web pages at a time. Now we'd like to work with a database worth of information. We'll start with unstructured data and then work with some structured data.

- If you haven't created your Astra account, do so now. It's free. Here's the link to sign up.
- Start a new, blank Langflow project
- Download and then import this file to your canvas. This is a Vertex AI module for accessing an Google Cloud embedding model
- Hover over the top of the component and click the ... then Advanced
- Populate the Project field of this module with your Google Project Name; Save.
- Under the "Extended" > "Text Splitters" category, drag the "Recursive Character Text Splitter" component to the canvas.
- Under the "Data" category, drag the "URL" component to the canvas
    - Find a few web pages that contain recent news or any other information which the LLM wouldn't know about. Add the URL of each one to the URL component. Notice the "+" sign to add more URLs to the component.
- Under the "Vector Stores" category, drag the "Astra DB" component to the canvas
    - Log in to Astra
    - Click the button to "Create a Database"
    - Choose "Serverless (Vector)"
    - Give your database a name
    - Choose Google Cloud as the provider
    - Choose us-east1 as the Region (Note that the paid tier of Astra supports more regions)
    - Click "Create Database" and after a few minutes your database will be available. Maybe time for a coffee break?
    - After the database has provisioned, you'll collect the API Endpoint and Generate a security token.



- Copy the API Endpoint and paste it into the “API Endpoint” field of the “Astra DB” component
- Click “Generate Token” and copy this to the “Token” field of the “Astra DB” component
- Populate the “collection” field of the “Astra DB” component with a short name appropriate for the URLs you are going to ingest into the database
- Rearrange the components and wire them together like this:



NOTE: Your Vertex AI Embeddings component does not have a “credentials” field. That’s normal.

Before you’ve run a flow, you may have noticed that there is a ▶ icon in the top-right corner of each component. You can run these one at a time to verify that they are working and configured correctly. After the flow runs, the icon changes to a ✓ to indicate that it has run successfully or a red X if not. After a successful run, you can mouse over these icons to inspect the output of each component.

The first time you run a data ingestion flow that includes Astra DB, it will automatically create a collection to store the data for you. However, this takes an extra 10 seconds or so. Subsequent runs will go faster.

After you've successfully run your ingestion pipeline, return to Astra, go to the database you created, then the "Data Explorer" tab and then click on the collection which was created for your data. Inspect the data. Take note of how the Recursive Text Splitter cut up the chunks of text and the embeddings which were calculated for each chunk of text.

Ingest a Product Database

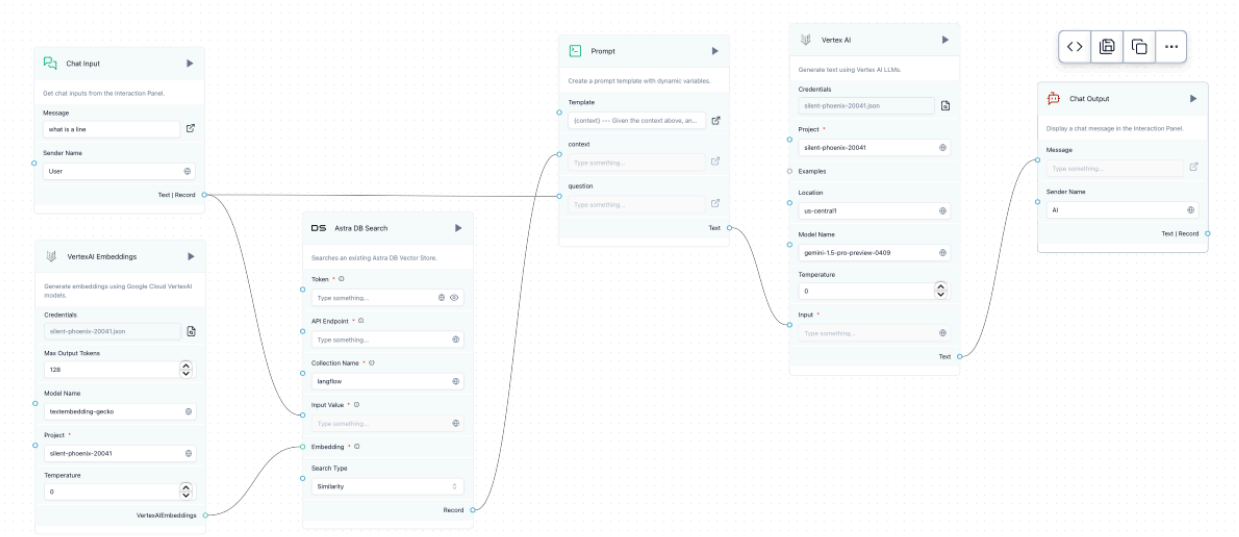
We don't need to make much of a change to import structured data.

- Download this small product database.
- Delete the URL component
- From the Data category, drag the "File" component to the canvas
- Connect this component to the Input of the Recursive Character Text Splitter component
- Click the  icon and select the product database you downloaded above.
- Change the "Collection Name" in the Astra DB component to "products"
- Run the ingestion flow
- Return to Astra and examine the imported data

Build a RAG Pipeline

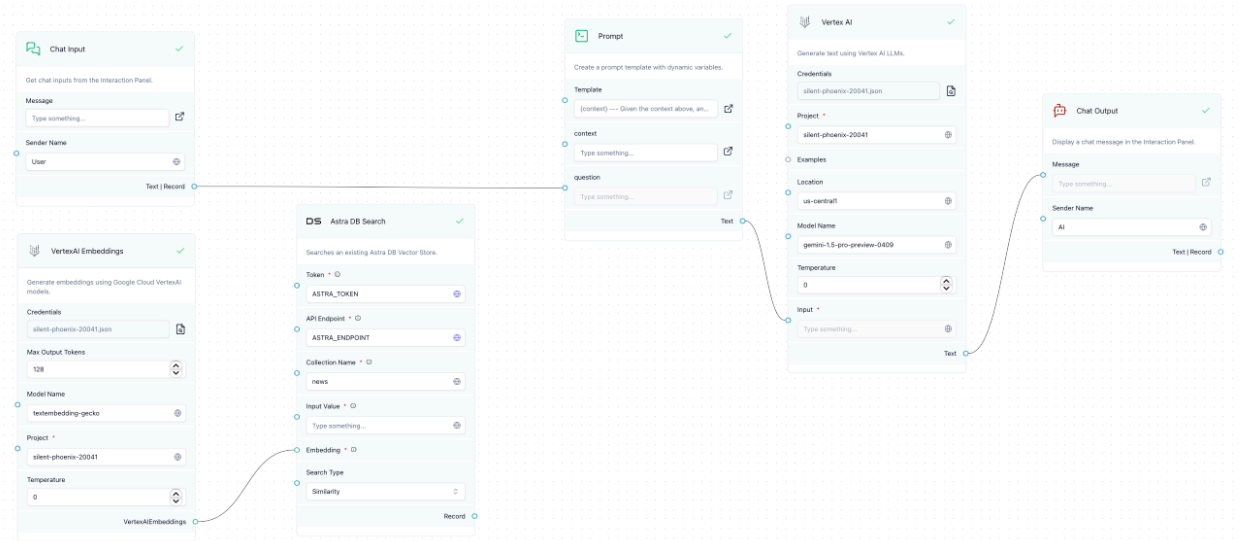
Retrieval Augmented Generation (RAG) is the process by which we use vector similarity search to find data most likely to answer the user's question and then add that to our prompt as context for the LLM. To do this, we have to embed the user's query in the same way as we embedded the data during ingestion.

For this part, rather than spell out each detail, use the following picture as a guide for the flow that you need to create. Note that you'll need to supply your information to the Vertex AI, Vertex AI Embeddings, and Astra DB Search components.



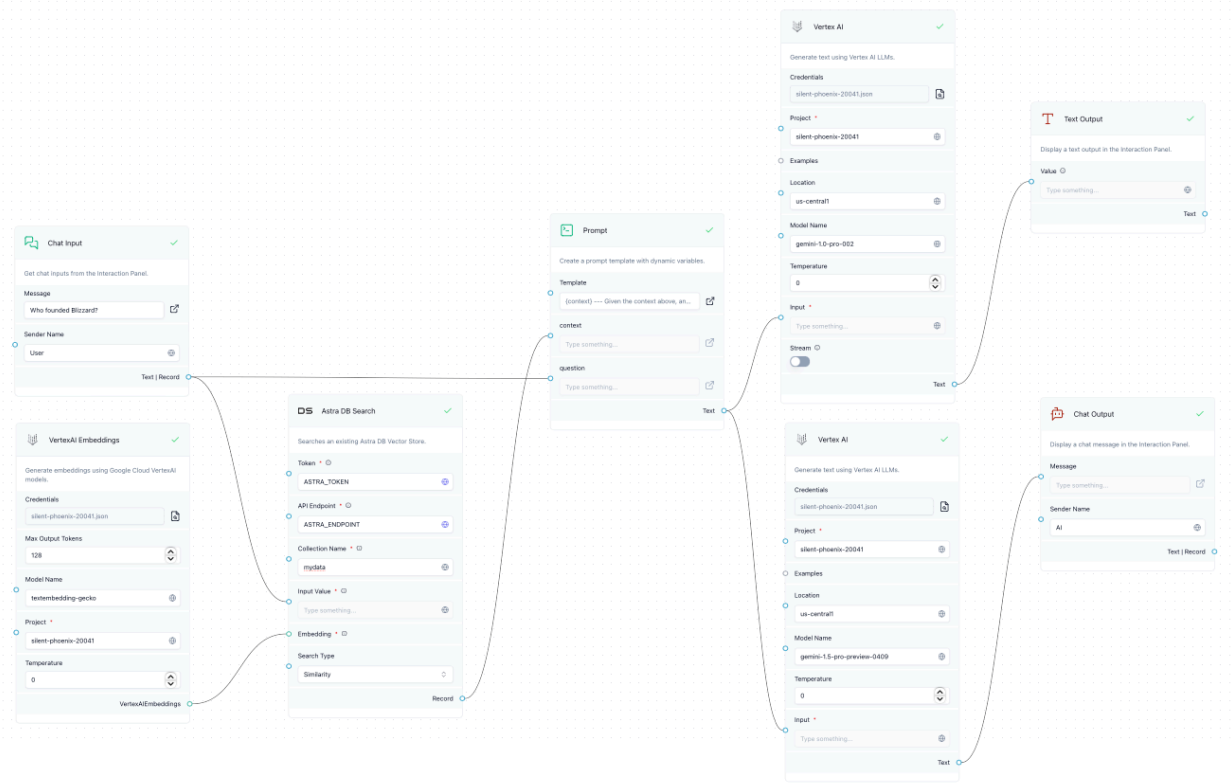
You can download [this file for the Vertex AI Embeddings component]
You can download [this file for the Vertex AI LLM component]
Import these onto your canvas after downloading.

Ask this pipeline something about the content you provided during the ingestion stage. You'll see that it has the answer. Now break the connections to the vector database like this and ask the same question:



By providing the LLM with additional information from a database, we are able to shore up some of the LLM's shortcomings, such as not having recent or private data.

After building and running your RAG flow, experiment with ways you can engineer the prompt to get better responses. You can also copy the Vertex AI component and try out two different models simultaneously! That flow would look something like this:



Make a RAG-based Recommendation Flow for Retail

In product data ingestion exercise, you should have created a collection in Astra for products. Now we're going to use that data as part of this RAG pipeline for recommendation products to shoppers based on natural-language queries.

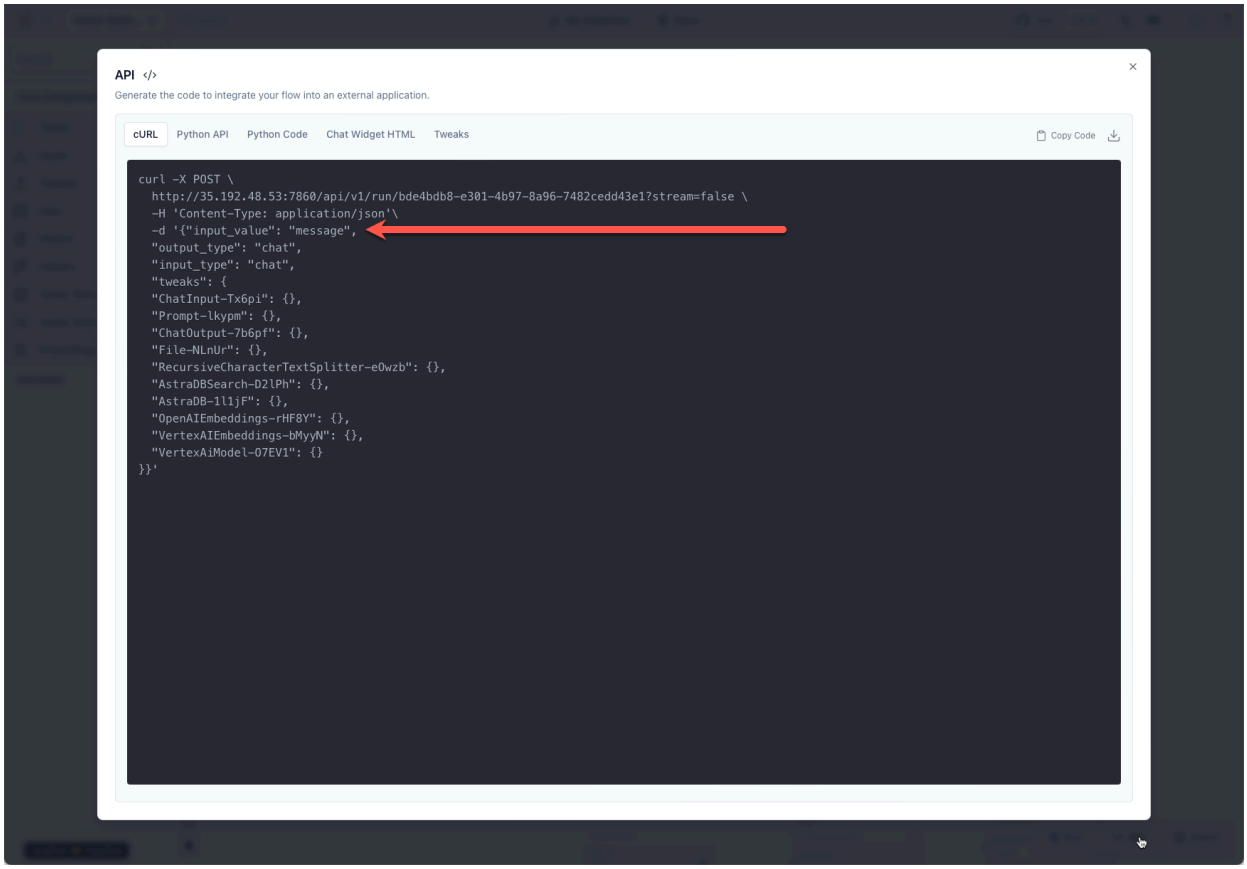
- Change the Astra DB Search Component to use the products collection
- Change the prompt to the following:

```
System: You are a personal shopping assistant.  Provide the user
with friendly guidance as to which products they may like.  If
none of the suggested products fit the user's request, respond,
"We don't have anything that fits your request."  Use the
following products as a starting place:

Products: {products}

User: {request}

Recommendation:
```

Connect Astra DB Search to the Prompt component's products input and the Chat Input to the Prompt's request.
Run the flow and tell the system something about you and what you're looking for. Gemini is particularly good about having these kinds of interactive conversations.

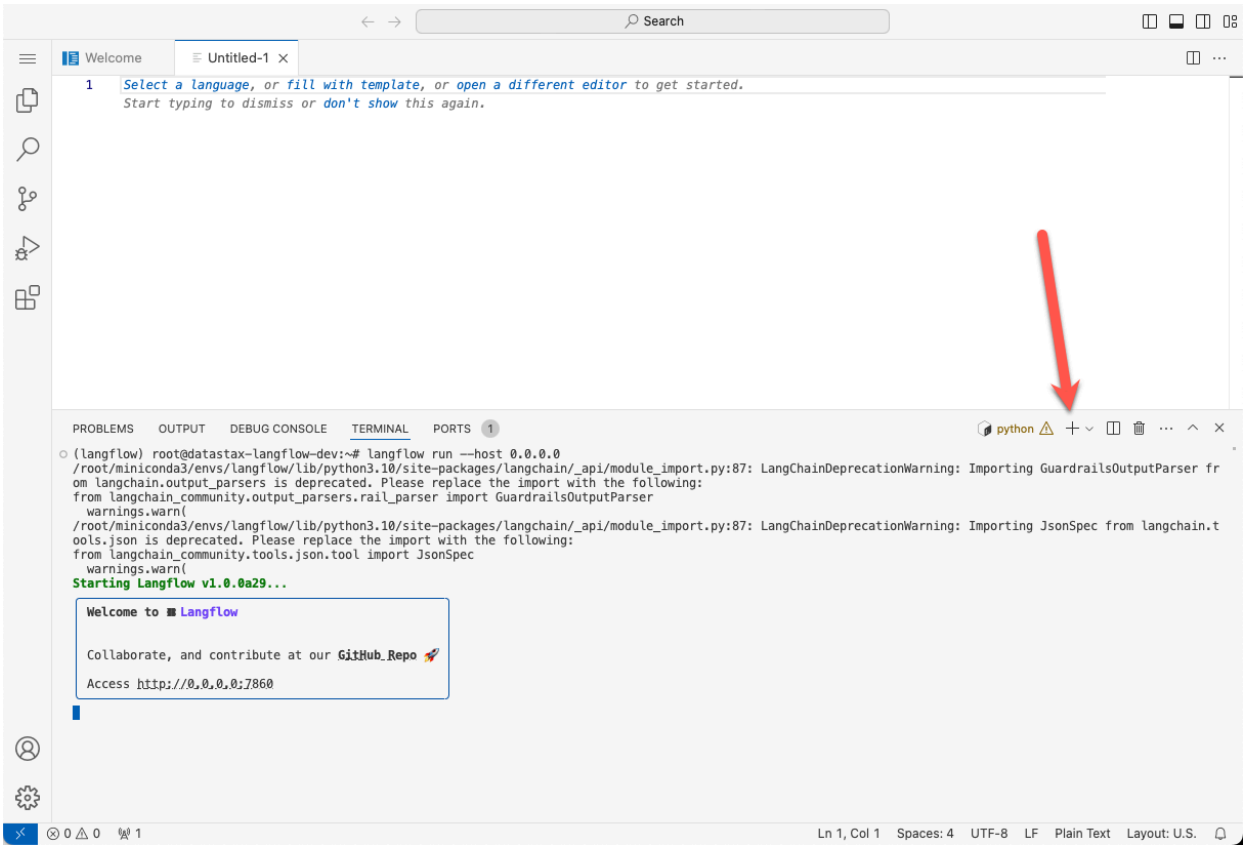
Build an API to Host the Chatbot

Langflow isn't just a development tool. It's also an API server. If you click on the "</> API" button, you can see several ways that you could access the API for a flow you've created. We'll start with using the command line:

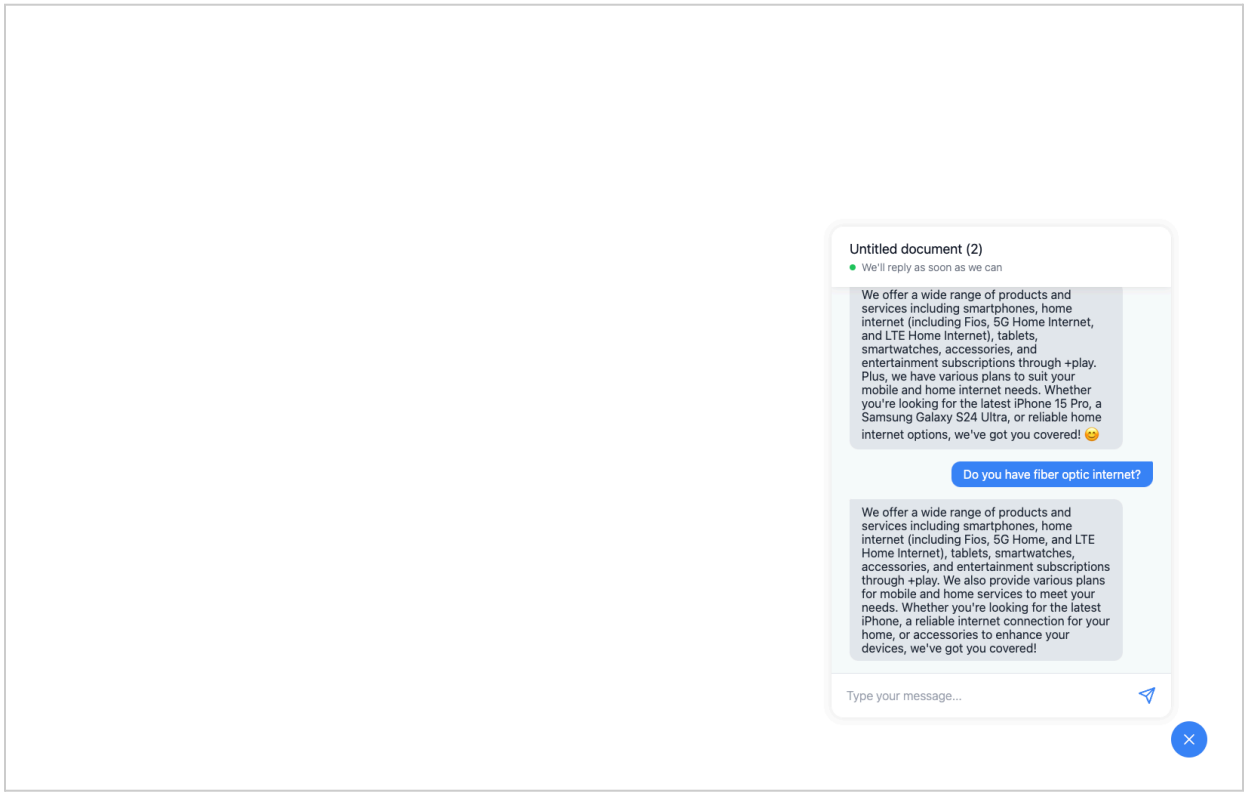


Copy-Paste the contents to an editor and change the “message” to whatever you would have typed in the chat UI. Now switch back to the tab where we started with VSCode. Click the button to create an additional terminal tab,

and then paste in your modified curl command.



Langflow also comes with a convenient chatbot UI that you can hook into your flows. In its most minimal form, here's what it looks like. In this example data was ingested from the Verizon homepage and then made available for chat.

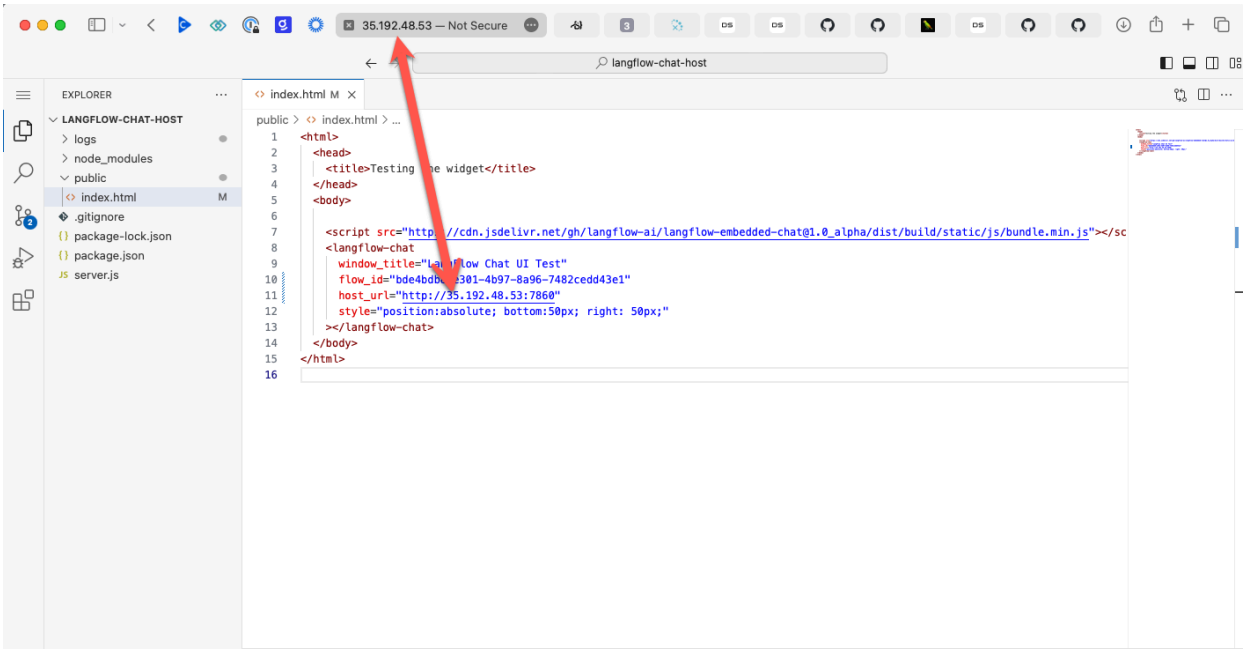


If you'd like to try this out on your own environment, follow these instructions:

In another terminal, paste these commands:

cd ~
apt-get install -y nodejs npm
git clone https://github.com/qzg/langflow-chat-host.git
cd langflow-chat-host
npm install express

Edit the index.html file in /root/langflow-chat-host/public. Change the flow_id to your flow_id and the host_url to match your IP address. The port number should stay 7860.



Save these changes and then on the command line run:

```
cd ~/langflow-chat-host
node server.js
```

VSCode will offer to open this in your browser; do that. This page will work correctly with the VSCode proxy.