BCT Internship Projects

Landon Zheng

What did I do? (Sequentially)

01

Data Migration

Moving API Data to an AWS S3 Bucket with Python

02

Flowise

What is Flowise, and my experience

03

Project 1

AI-Powered Employee Leave Database & Uploader

04

Project 2

AI-Powered Invoice Processor & Uploader

Data Migration

Moving API Data to an AWS S3 Bucket with Python

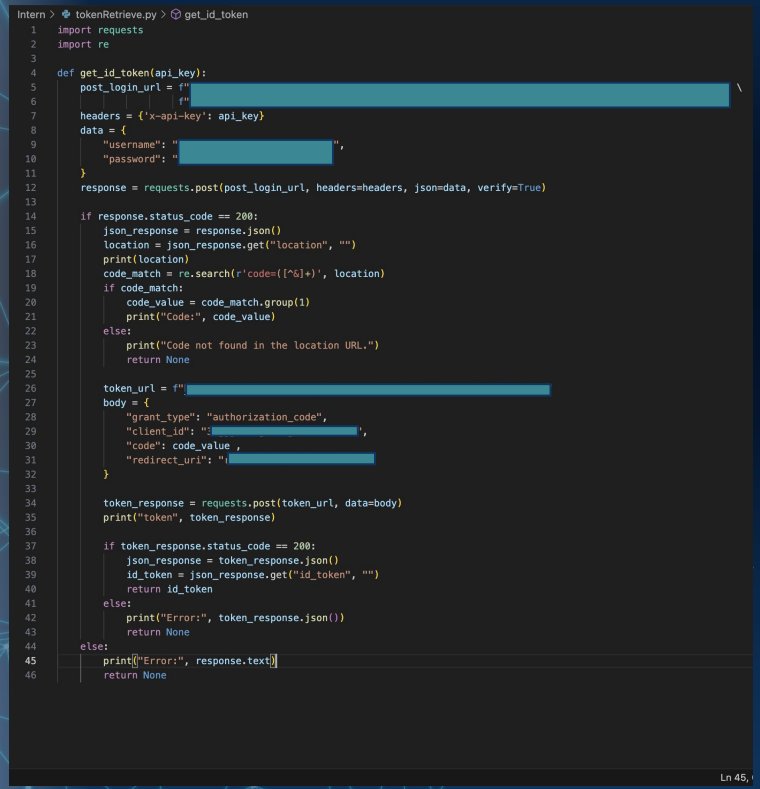
Migration Process

Retrieving ID Token

Obtain GPS Data

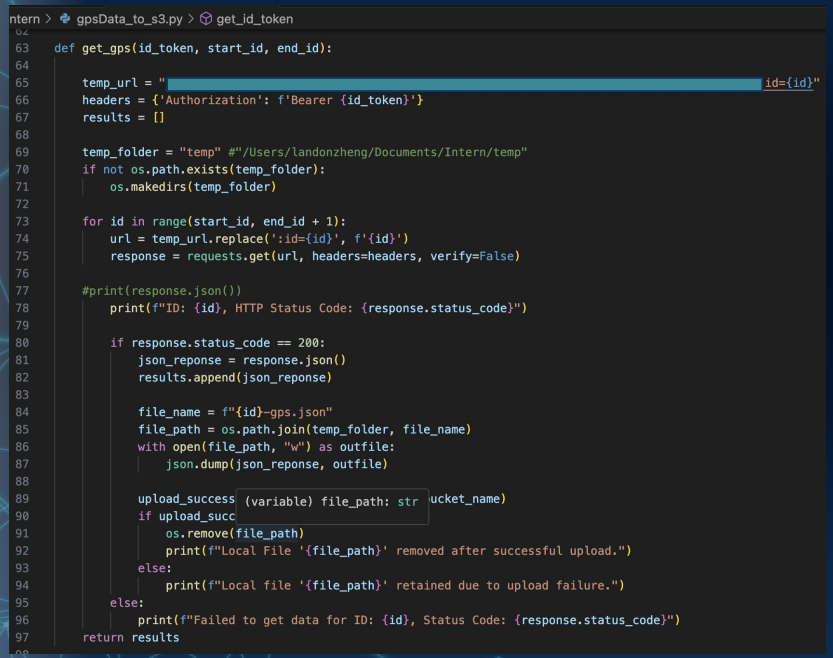
Upload Data to S3 Bucket

Retrieving ID Token



- Rest Post API
- Authentication: Username & Password and API Key
- Using requests.post() for posting & response json
- Two-Step Process: Retrieve the code, then use the code for the actual ID Token.

Obtaining GPS Data



- Rest Get API
- Use the ID Token retrieved as a Bearer Token for this API
- Multiple parts of GPS data, so retrieve each point with its unique ID
- Store all in a temporary file for upload later

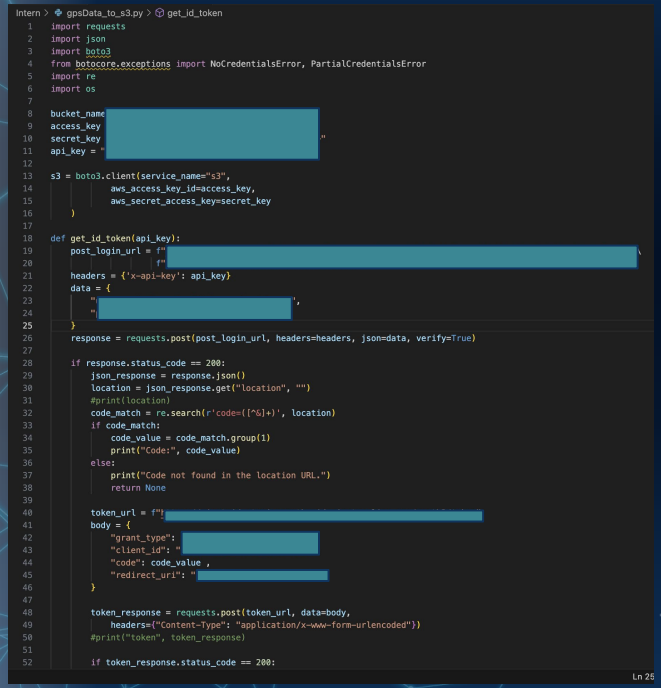
Uploading the Data

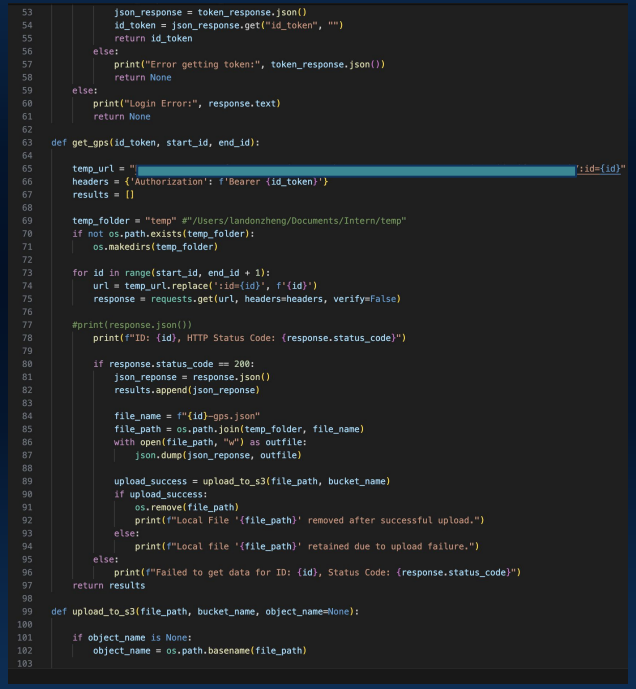
Intern > s3Upload.py > upload_file

```python
import logging
import boto3
from botocore.exceptions import ClientError
import os

def upload_file(file_path):

    bucket_name = 
    access_key = "
    secret_key = "                                                "

    s3 = boto3.client(service_name="s3",
            aws_access_key_id=access_key,
            aws_secret_access_key=secret_key
    )
    response = s3.upload_file(file_path, bucket_name, file_path)
    print(f"upload_log_to_aws response: {response}")
    if response is None:
        print("Success")
    else:
        print("Error")
```

- S3 Bucket: Database in AWS
- Using Boto3 Library for S3 Upload: https://boto3.amazonaws.com/v1/documentation/api/latest/guide/s3-uploading-files.html
- boto3.client().upload_file() method

Combining All Steps





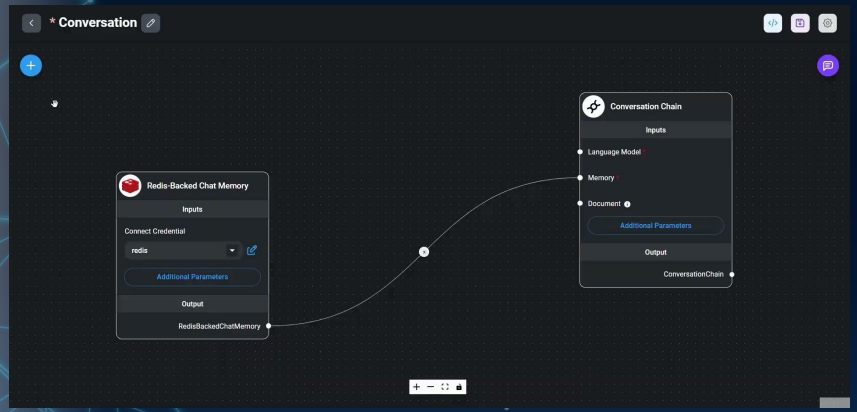
- All 3 steps in one class
- Changed some methods in order to function all together
- By running this python script, GPS data will be automatically uploaded to the S3 Bucket

02

Flowise

What is Flowise, and my experience

What is Flowise?



Credit: flowiseai.com

- Traditionally, connecting LLMs (Chat GPT) to serve personal projects is difficult
- Low-code tool that enables LLMs to be connected to more services through interactive chatbots
- Turns LangChain's complex code into simple drag-and-drops
- Able to build personalized AI assistants- a need for the future
- Download with npm or Docker, used on Google through local host

Learning Flowise

I learned Flowise with a 10-Part Tutorial, exploring most of Flowise's tools:

Chains & LLMs

A chain is a basic interactive chatbot. They are connected to a LLM (Large Language Model) such as Chat GPT and a prompt template for how to behave. There are many types of chains.

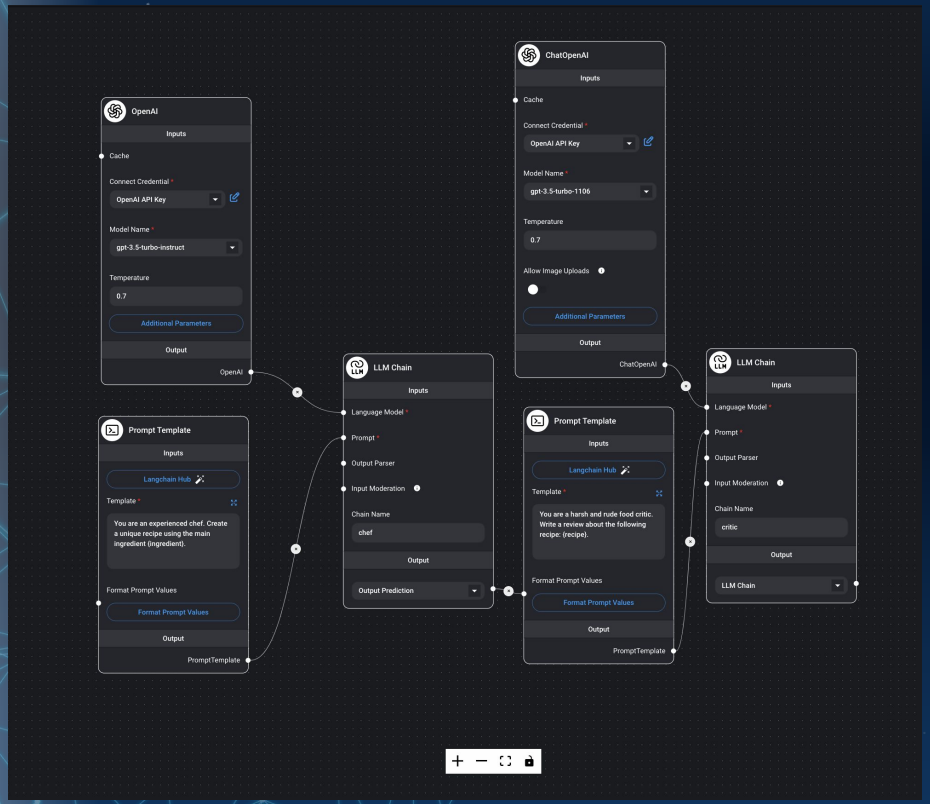
RAG & Vector Storage

RAG (Retrieval Augmented Generation) is a useful tool in AI, allowing them to obtain information off external sources. These sources are stored in vector storages, an easy way for LLMs to comprehend the info.

Agents

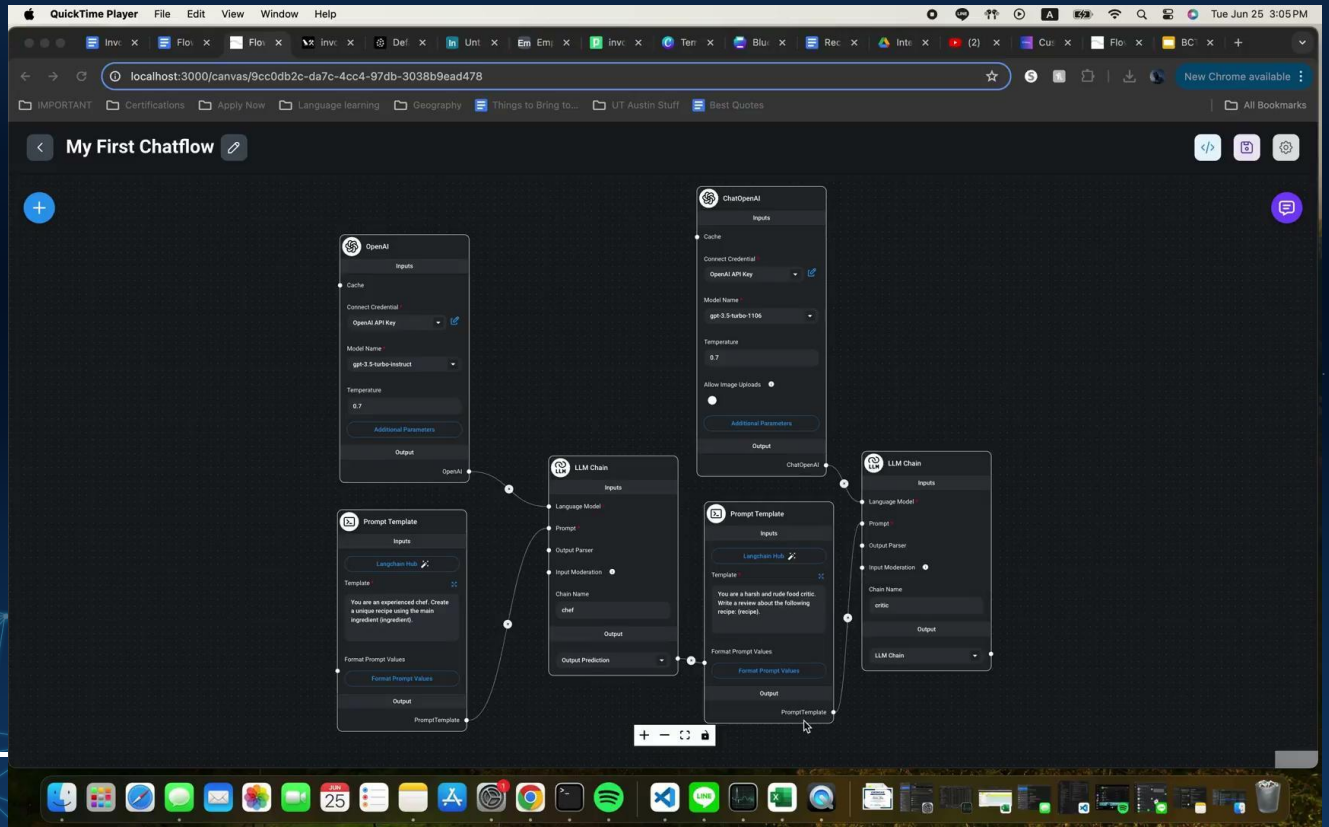
Agents are by far the most powerful chatbot in Flowise. They are equipped with memory for previous conversations, a LLM chat model, and multiple customizable tools. The agent consciously picks which tool it needs to complete a task.

Chains & LLMs

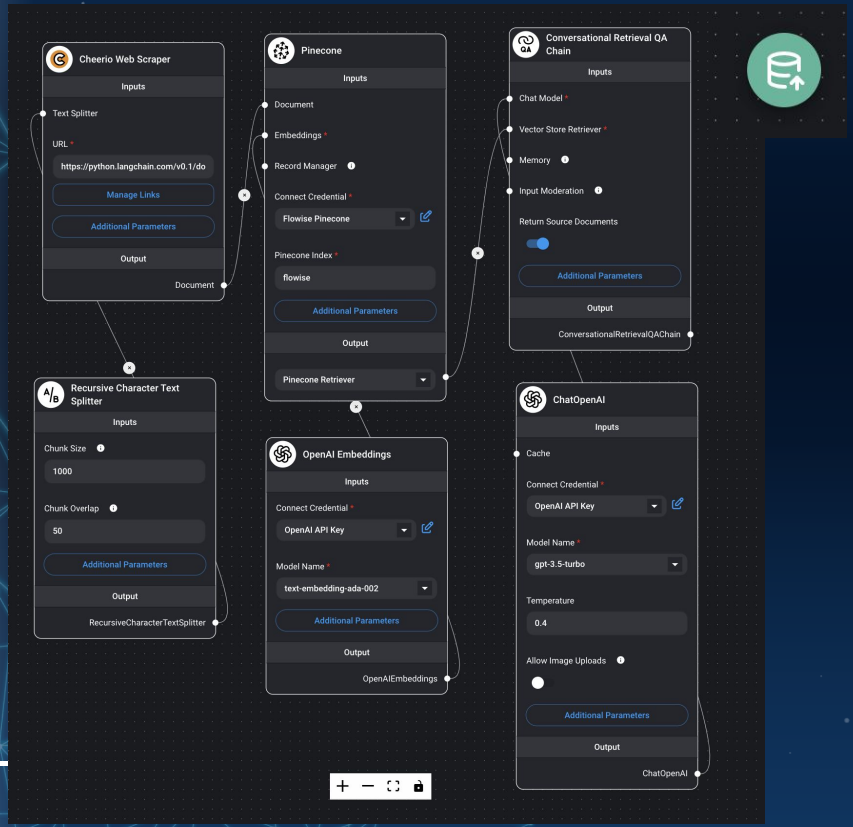


- This Chatflow has 2 LLM Chains- basic chains that connect Chat GPT to an interactive chatbot
- 1st LLM Chain (chef) has a prompt template, telling it to create a unique recipe when an ingredient is typed in
- That recipe is then passed to another prompt LLM chain with a prompt template to write a harsh review about the recipe
- So, when an ingredient is typed in, a criticism of the recipe created by the 1st LLM Chain is outputted

Chains & LLMs Video Demo

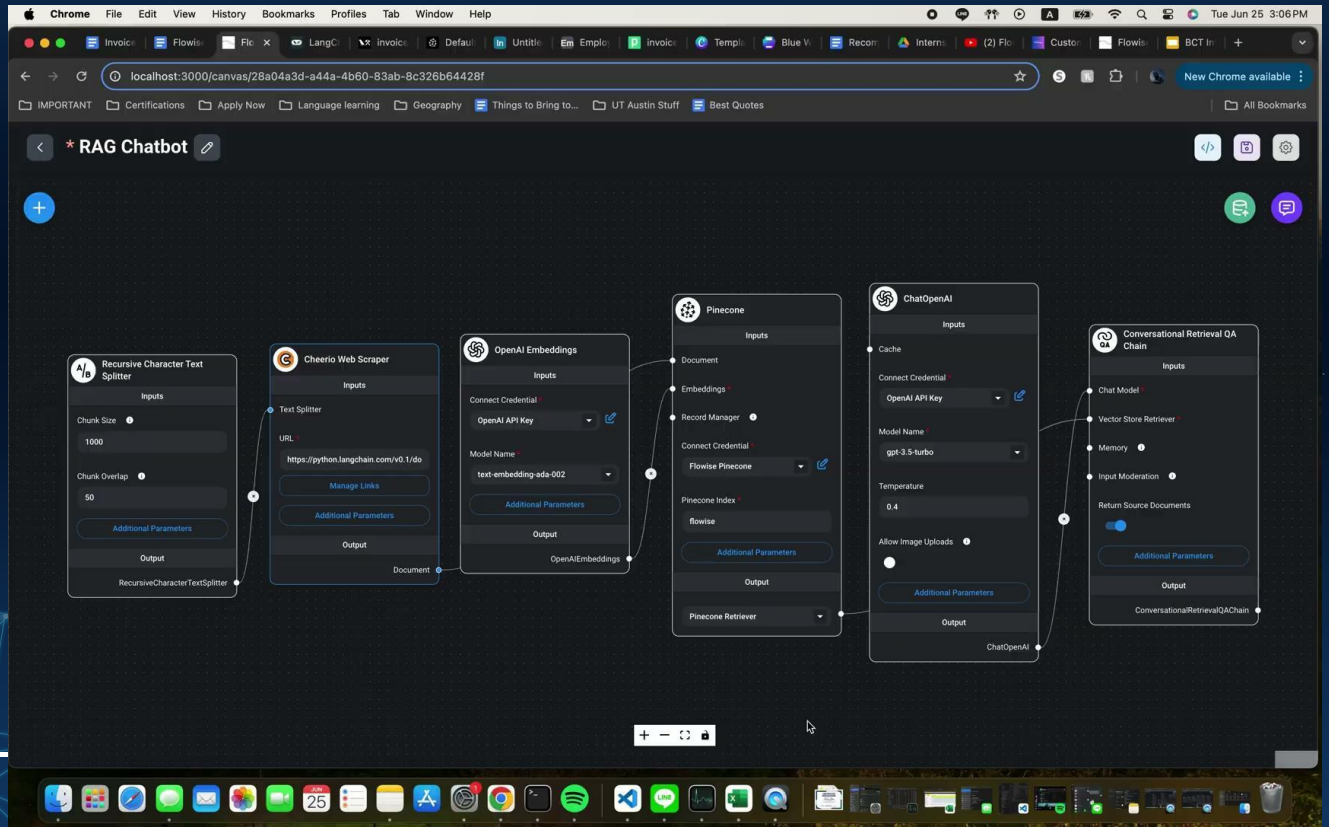


RAG & Vector Storage

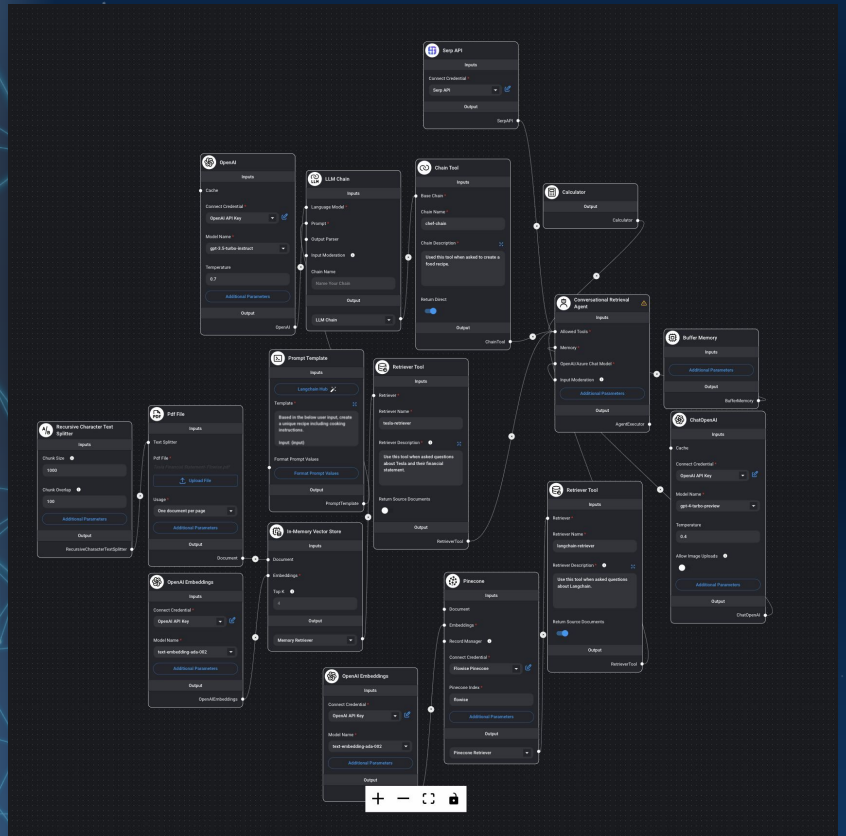


- A Conversational Retrieval QA Chain is used- a special chain that can retrieve data from vector stores
- Pinecone is a very common vector store; embeddings are used to translate the vectors into info
- Pinecone grabs information from the Cheerio Web Scraper, which scrapes a page about LCEL
- Text Splitters are used to chunk the text and make it easier to process
- MUST UPSERT INFO (Green Button on Top Right Corner) to Pinecone Vector Storage before testing AI Chatbot

RAG & Vector Storages Video Demo

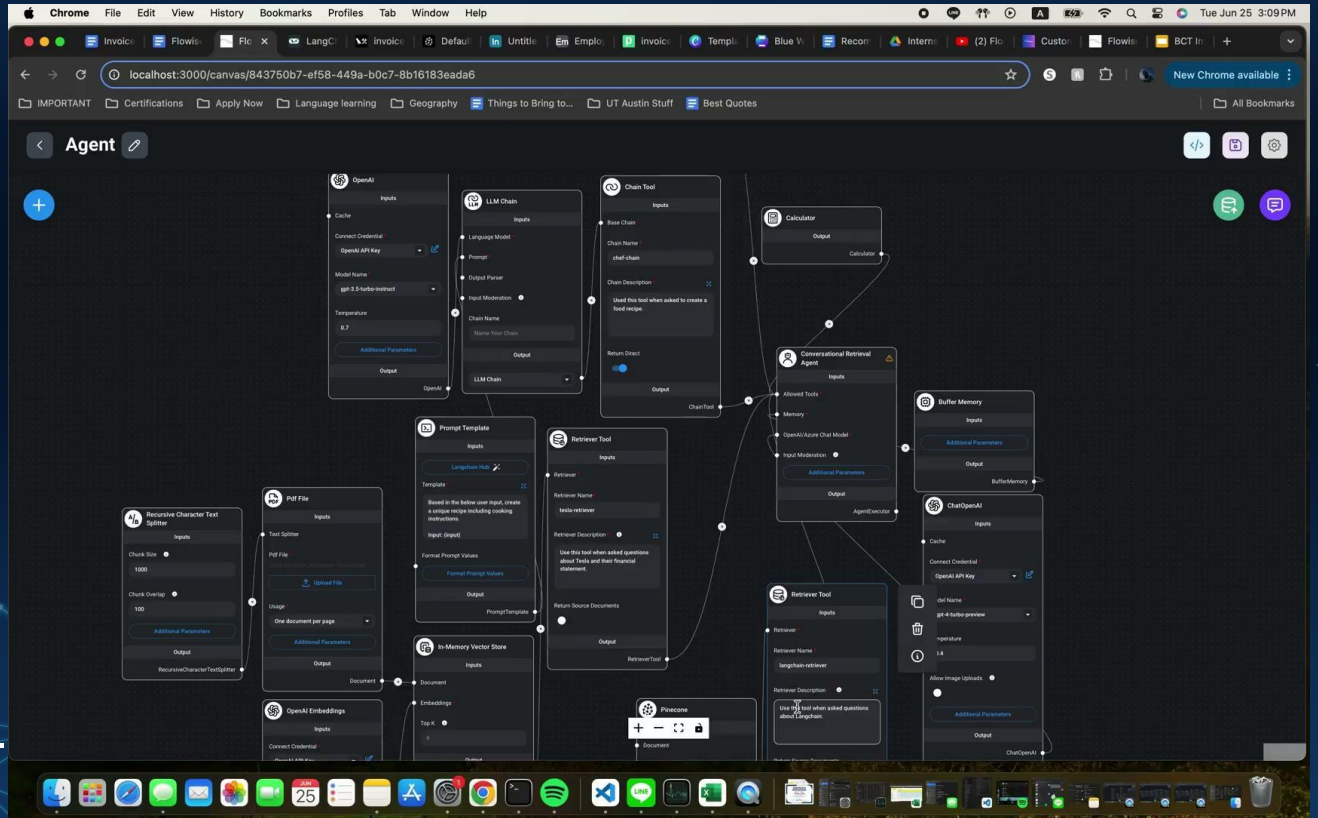


Agents



- An agent can be connected to many tools, and picks which one to use.
- A Conversational Retrieval Agent has 2 more capabilities- RAG and Function calling. Function calling is used to call custom tools (seen later)
- This agent is connected to the tools: Serp API (searches Google), Chef Chain Tool (creates a unique recipe off an ingredient). Retriever Tool 1 (Info about Tesla Financial Statement), Retrieval Tool 2 (Info about LCEL), and a Calculator (Math computation).
- These tools help the agent in specific questions.

Agents Video Demo



Project 1

AI-Powered Employee Leave Database
& Uploader

Problem to Solve

Current situation

Right now, BCT's Employee Leave system is an old website that's glitchy and hard-to-use. An improvement to the leave system is needed.

1

RAG

The AI has to understand BCT's Leave Policy

2

Retrieve

The AI has to retrieve info from the database

3

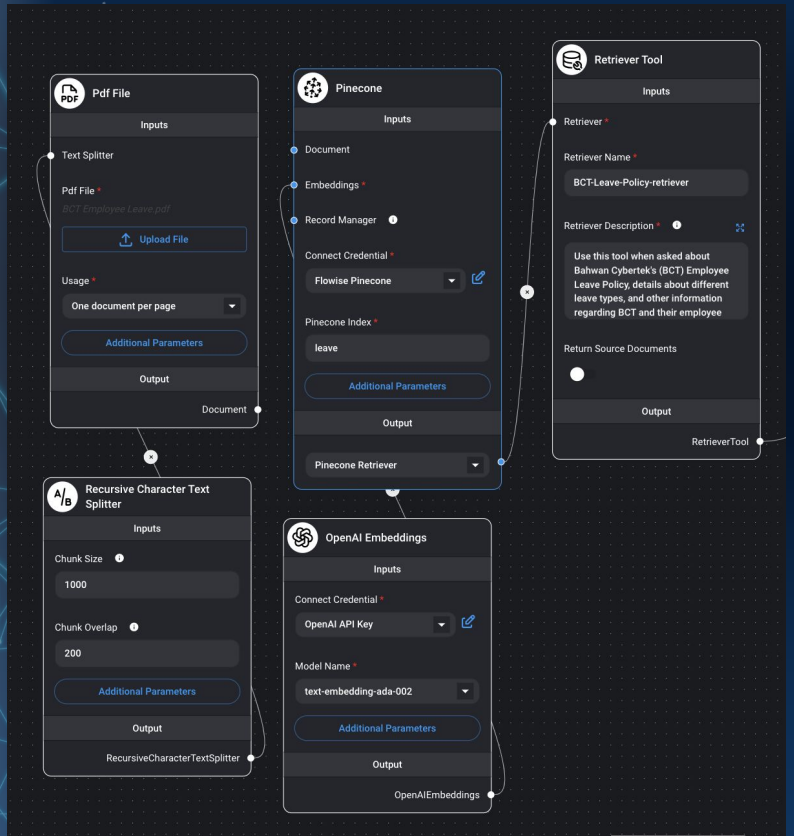
Upload

The AI has to upload leave info into the database

Solutions Needed

I concluded I need to use a Conversational Retrieval Agent with Chat GPT 4-Turbo-Preview for RAG & Custom Tools. Separate steps are fine.

Using RAG For BCT's Policy Sheet

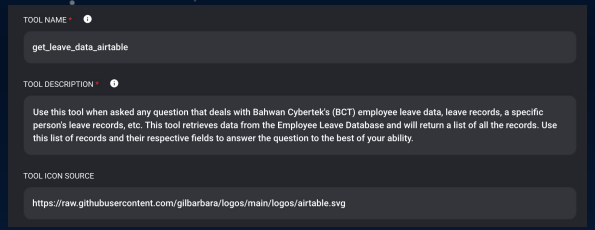


- With BCT's Policy Sheet, I used a simple RAG procedure
- Retriever Tool for the Agent, specified should be used when asked about BCT Leave Policy
- Pinecone as Vector Storage
- Uploaded the PDF of the Policy Sheet as the Document, split into chunks with a text splitter
- Upserted the PDF into Pinecone (and now can disconnect PDF File to Pinecone)

Retrieving Data with Custom Tool

Javascript Function:

```
const fetch = require('node-fetch');

const token = 'API_KEY';
const baseId = 'BASE_ID';
const tableId = 'TABLE_ID';
const url = `https://api.airtable.com/v0/${baseId}/${tableId}`;
const fields = ['id', 'name', 'leave_type', 'start_date', 'duration'];

const options = {
  method: 'GET',
  headers: {
    'Authorization': `Bearer ${token}`,
    'Content-Type': 'application/json'
  }
};

try {
  const response = await fetch(url, options);
  const text = await response.json();
  return text.records.map(record => ({
    id: record.fields.id,
    name: record.fields.name,
    leave_type: record.fields.leave_type,
    start_date: record.fields.start_date,
    duration: record.fields.duration
  }));
} catch (error) {
  console.error(error);
  return 'Error';
}
```

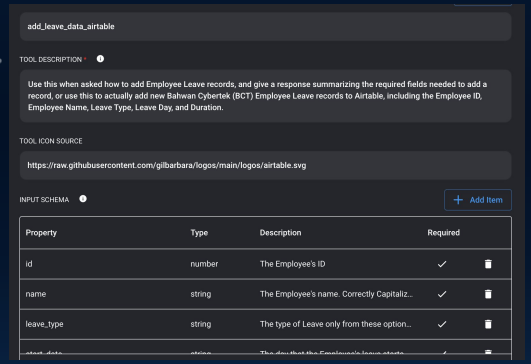


- For the Database, I chose Airtable because of its free API integration
- Custom Tools use OpenAI's Function Calling, so a Conversational Retrieval Agent or OpenAI Function Agent is needed
- The JS Function sends an Airtable Get Request to fetch the entire table of leave records so the LLM can interpret them and answer the user's question about the existing leave records
- Need a Bearer API Key, Base ID, and Table ID

Uploading Data with Custom Tool

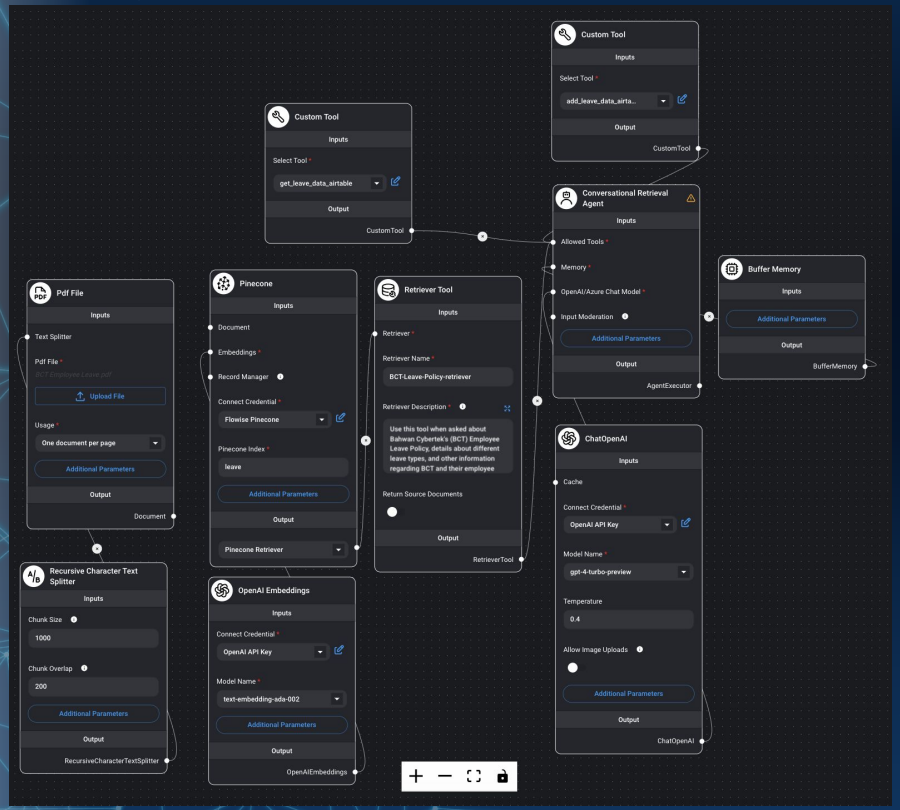
Javascript Function:

```
const fetch = require('node-fetch');
const baseId = 'BASE_ID';
const tableId = 'TABLE_ID';
const token = 'API_KEY';

const body = {
"records": [
{
"fields": {
"id": $id,
"name": $name,
"leave_type": $leave_type,
"start_date": $start_date,
"duration": $duration,
}
}
]
};

const options = {
        method: 'POST',
        headers: {
                'Authorization': `Bearer ${token}`,
                'Content-Type': 'application/json'
        },
        body: JSON.stringify(body)
};

const url = `https://api.airtable.com/v0/${baseId}/${tableId}`

try {
        const response = await fetch(url, options);
        const text = await response.text();
        return text;
} catch (error) {
        console.error(error);
        return 'Error';
}
```



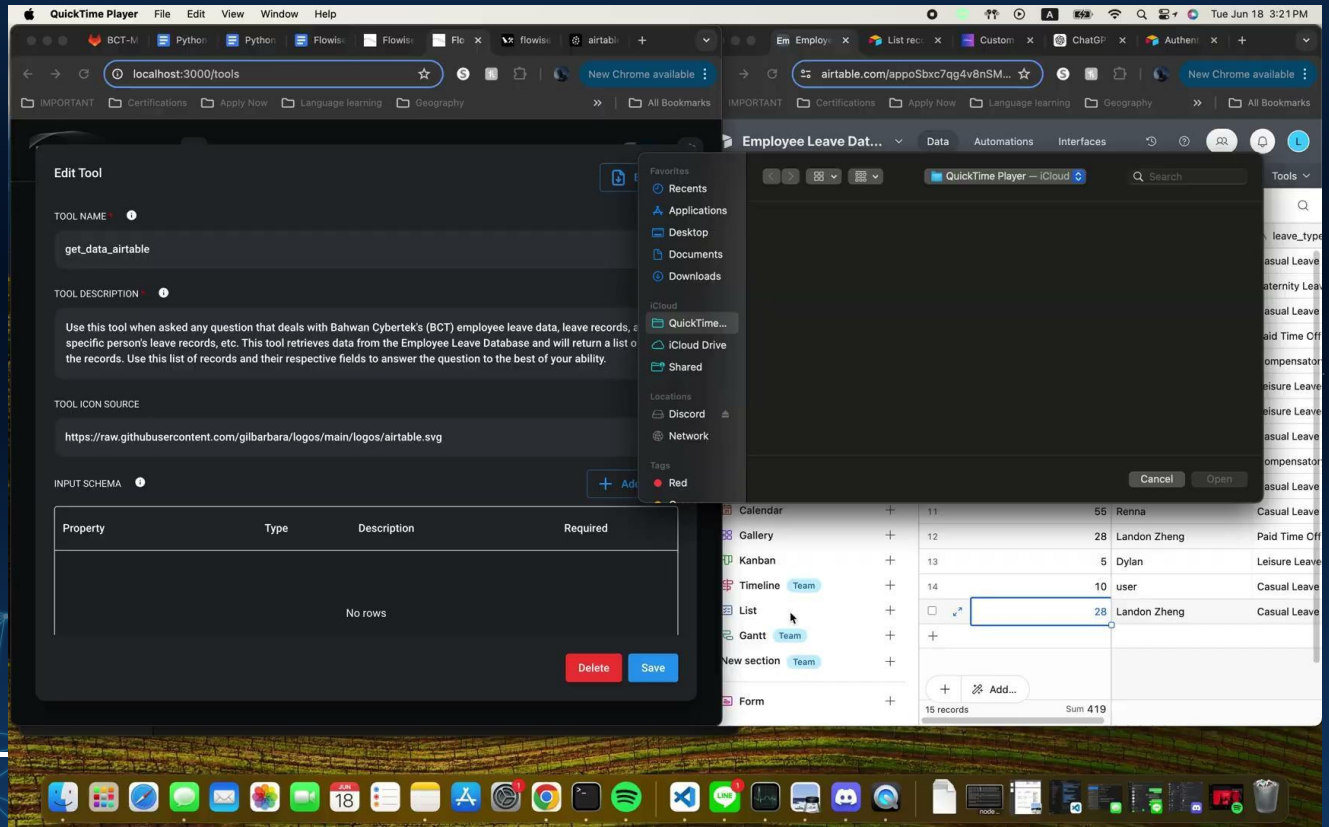
- The Airtable has 5 different fields needed to create a leave record: Employee ID, Employee Name, Leave Type, Start Date, and Duration
- Creating a custom tool to upload the records with a custom JS function is the most efficient way to upload a record
- The Javascript function sends a Post Request with the required fields in order to create a leave record

Final Chatflow

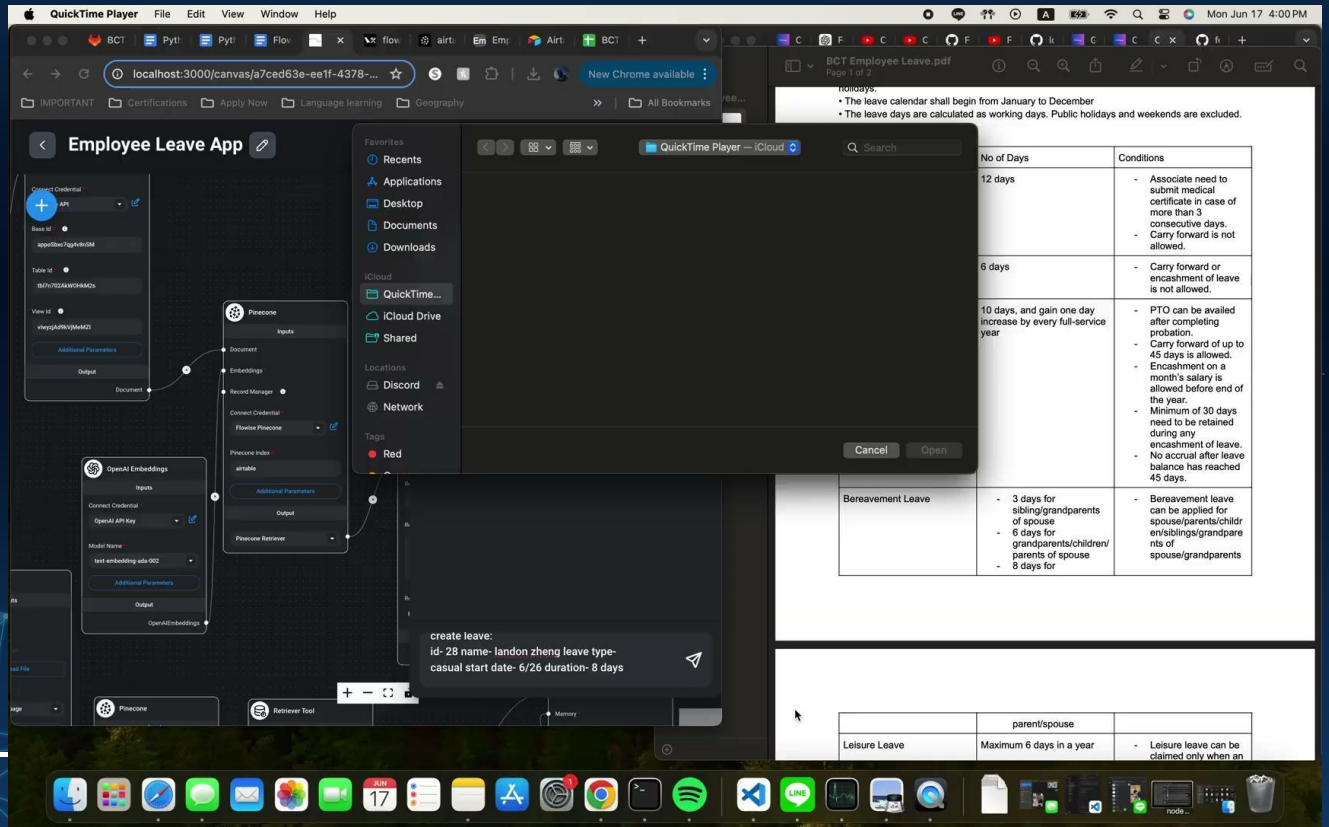


- Combining all the above parts, the final chatflow has a conversational retrieval agent with 2 Custom Tools to get and upload data from Airtable, and a Retrieval Tool to understand BCT Leave Policy Data
- The agent will ask for confirmation before uploading to make sure details are correct
- Possible change: Agent asks for user and password first before granting access

Getting Leave Records Video Demo



Uploading Leave Records Video Demo



04

Project 2

AI-Powered Invoice Processor & Uploader

Problem to Solve

Current situation

BCT, as any company, has many expenses. Some come in invoices, and the traditional method is to manually scan and upload them. This can be improved with AI chatbots.

1

Understand

The AI must understand invoice image uploads

2

Summarize

The AI has to summarize the needed fields

3

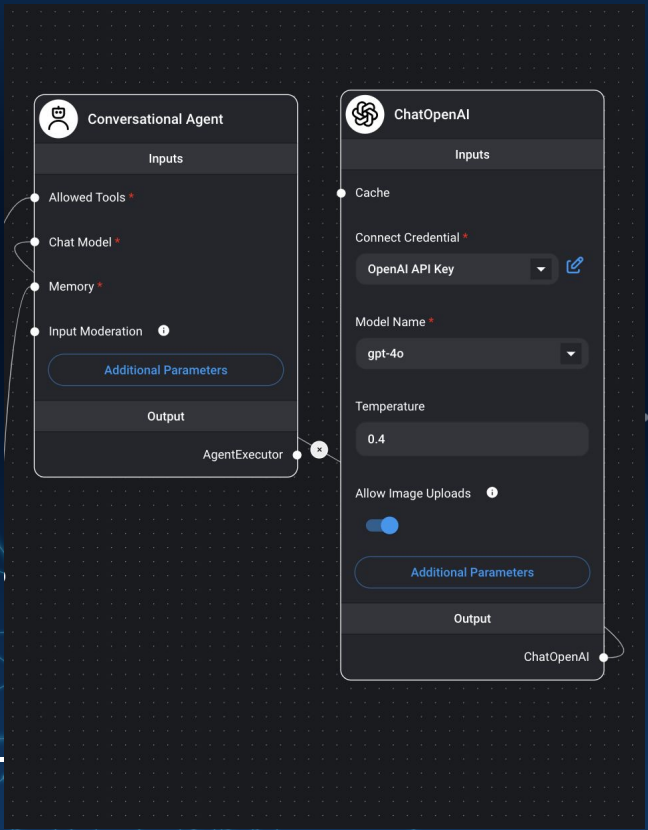
Upload

The AI has to upload invoice info into the database

Solutions Needed

I concluded I need to use a Conversational Agent with Chat GPT 4o for Image Understanding. Also, I wanted the invoice processing and uploading to happen ALL AT ONCE, rather than at separate steps like Employee Leave.

Understanding Image Uploads

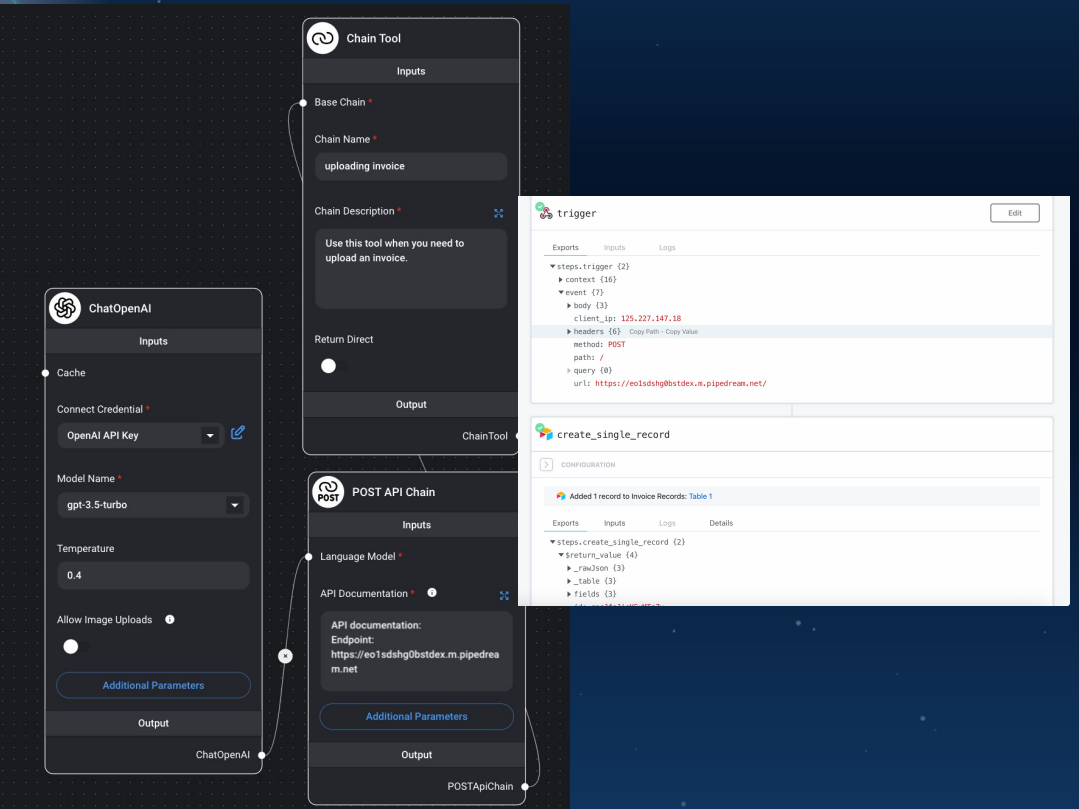


- OpenAI recently released Vision Capabilities, which means that some models can process images. GPT-4o is one of them, so I used this model.
- Since Flowise is still in development, only Conversational Agent & React Agent are compatible with Image Uploads. I picked Conversational Agent
- The Conversational Agent becomes a problem pretty soon.

Summarizing Invoice Images

- This is an example invoice. The required fields to obtain are the Name of the place purchased, the Price (總計), and the Buyer ID (買方). This example would be Miramar, 600, and 43942656
- Full Invoice Documentation can be seen here
- Vision Capabilities automatically allow Chat GPT to get these fields (although sometimes inaccurate), just needed to specify in the prompt template of the agent under “Additional Parameters”

Uploading Invoice Details



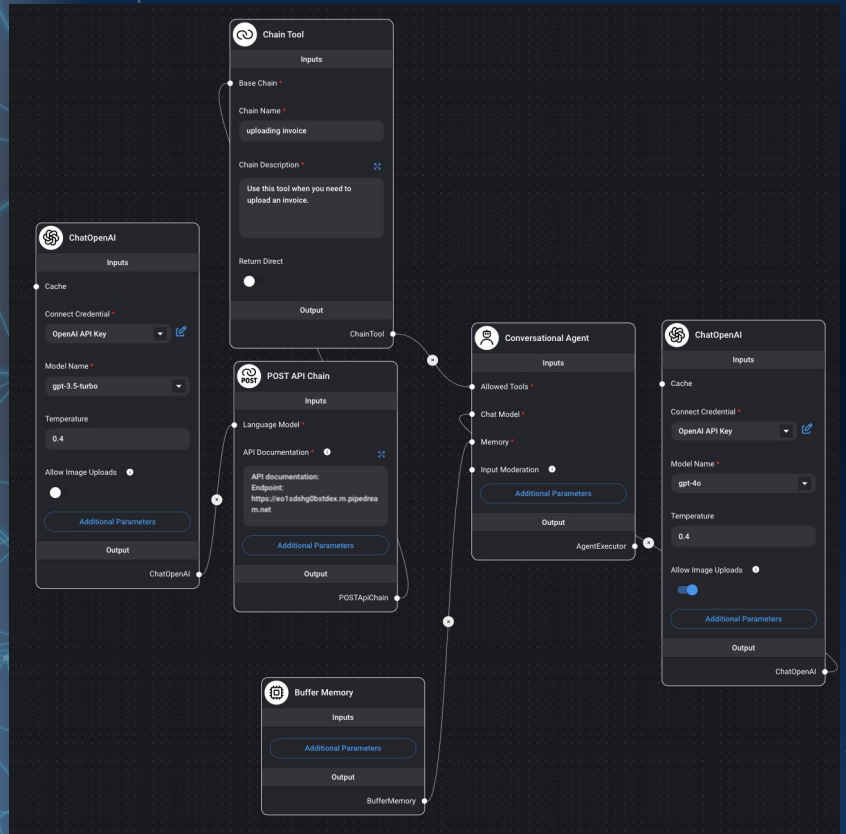
- Airtable was used as the database
- To upload the invoice, I used a chain tool that connected to a Post API Chain- executing Post Requests.
- API Documentation tells the LLM to put the name, price, and buyer fields in the body
- This chain posts to a Pipedream API, which triggers a process due to the HTML post. This then creates a record inside Airtable.
- This method costs $29/month because Pipedream isn't free

Why is Uploading so Complicated?

As you can see, uploading invoices is quite complicated and expensive. Why can't I just use a Custom Tool like when uploading Employee Leave Records? This is because:

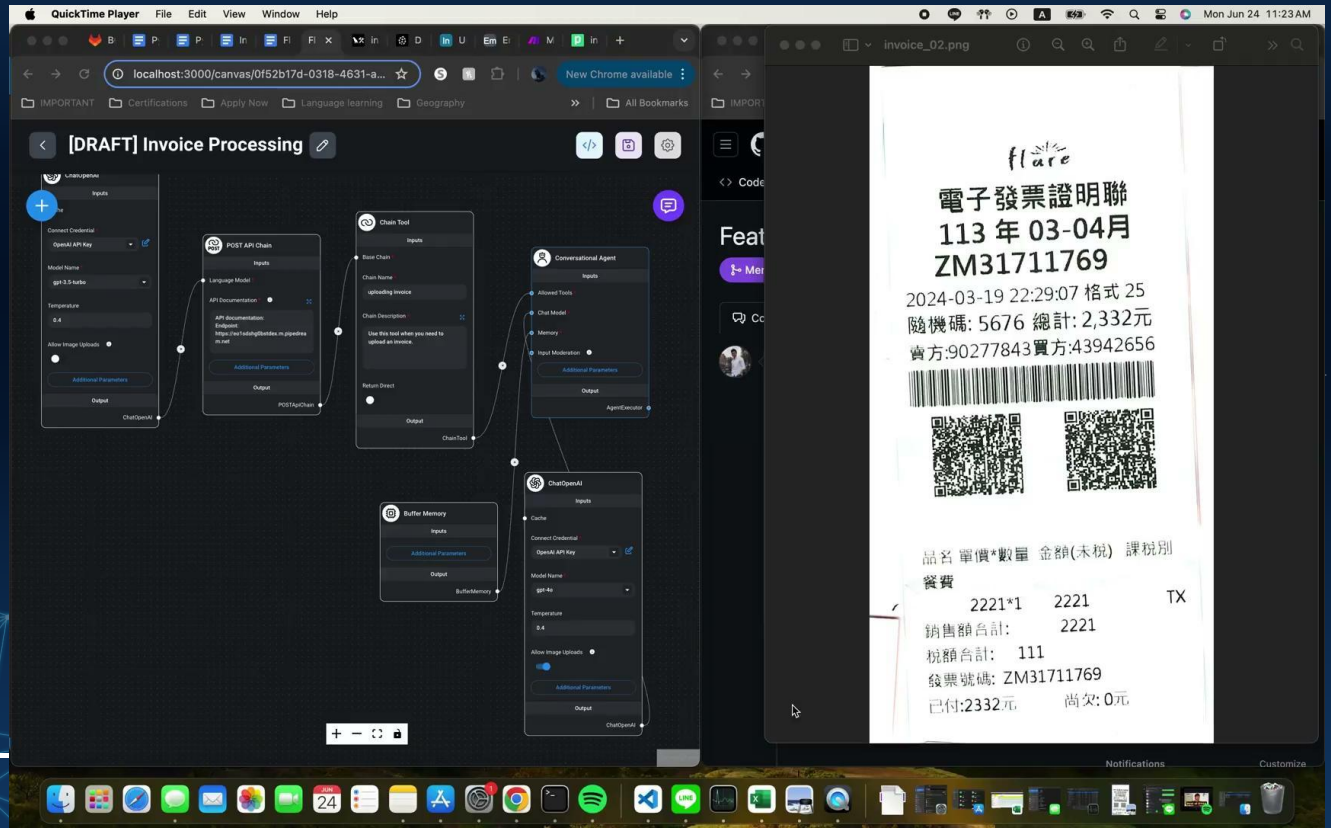
- Remember how I was restricted to Conversational Agents due to image processing compatibility? Conversational Agents don't have function calling, meaning they can't call a Custom Tool or another Chatflow. This rules out Custom tools completely.
- Additionally, Posting directly to Airtable with the Post API Chain (instead of Posting onto Pipedream which posts onto Airtable) is not possible, as Flowise only configured LLMs to post requests with JSON strings. Airtable API requires JSON objects, which thus makes anything posting directly to Airtable impossible.
- Thus, the best solution I can find right now is posting on a Pipedream API to trigger a post onto Airtable.

Final Chatflow

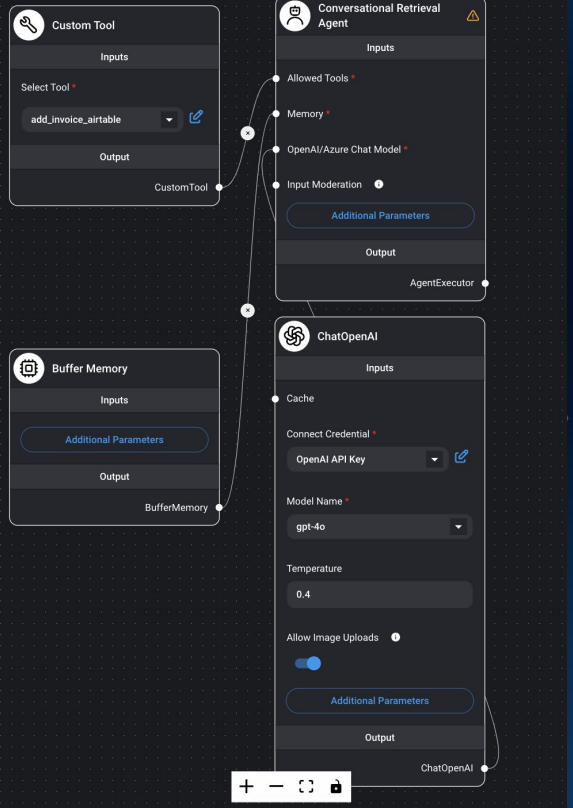


- Combining all the above parts, the final chatflow has a conversational agent with 1 Chain Tool, which uploads invoice details to Airtable using Pipedream as a middleman
- Keep in mind I configured this process to be automatic, so that when an invoice was uploaded the Agent would automatically process and summarize the invoice image details and then upload the details onto Airtable in one go.

Invoice Processing Video Demo



Potential Chatflow in the Future?



Javascript Function for Tool:

```
const fetch = require('node-fetch');
const baseId = 'BASE_ID';
const tableId = 'TABLE_ID';
const token = 'API_KEY';

const body = {
"records": [
{
"fields": {
"name": $name,
"price": $price,
 "buyer": $buyer
}
}
]
};

const options = {
        method: 'POST',
        headers: {
                'Authorization': `Bearer ${token}`,
                'Content-Type': 'application/json'
        },
        body: JSON.stringify(body)
};

const url = `https://api.airtable.com/v0/${baseId}/${tableId}`

try {
        const response = await fetch(url, options);
        const text = await response.text();
        return text;
} catch (error) {
        console.error(error);
        return 'Error';
}
```

- A Conversational Retrieval Agent would make this much easier. With function calling, I could just use a custom tool (already built) to upload directly to Airtable, just like in the Employee Leave Project.
- Conversational Retrieval Agent will be deprecated (updated) in the future, so if it has image upload compatibility, this is a much simpler and more efficient way.

Thank you for everything, BCT. It was amazing!

Special Thanks to Dylan for giving me this intern chance, Jeff for mentoring me through and through, Renna for the assistance during the migration process, and Ken & Alex for making the lunches so enjoyable.