Design chatbots using Flowise

Last amended: 24th Jan, 2026
My folder: C:\Users\ashok\OneDrive\Documents\flowise
Flowise Book created by Community at this link
Download this file from here.

Vector stores working fine: FAISS; Milvus, Milisearch, chroma. But not Qdrant. If a docker vector store does not work with http://localhost:PORT, try: http://hostip:PORT

Table of Contents

A. Attention mechanism

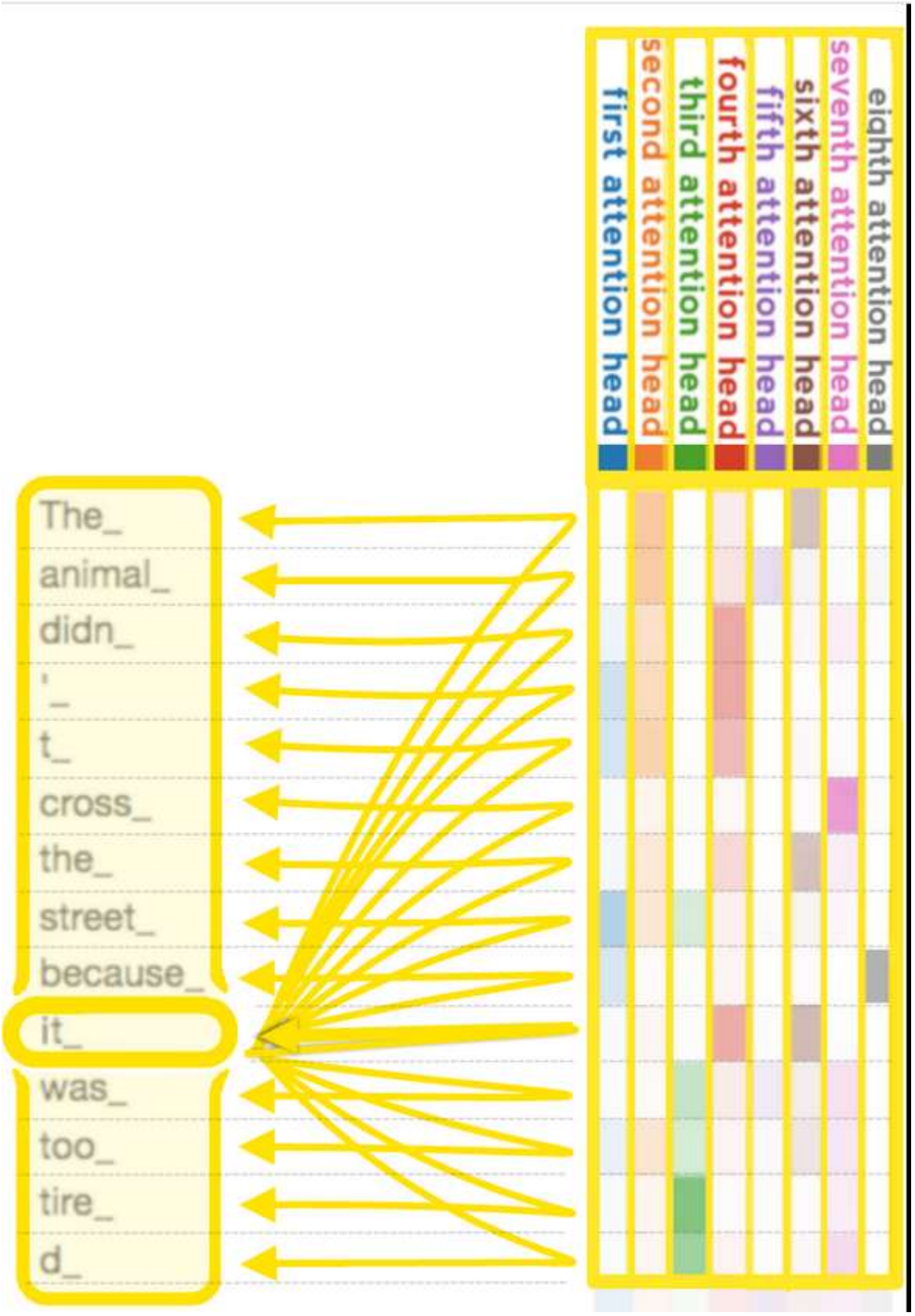


Figure 1: When each head calculates, according to its own criteria, how much other tokens are relevant for the "it_" token, note that the second attention head, represented by the second column, is focusing most on the first two rows, i.e. the

tokens "The" and "animal", while the third column is focusing most on the bottom two rows, i.e. on "tired", which has been tokenized into two tokens. Google's Gemini 1.5, introduced in February 2024, can have a context window of up to 1 million tokens.

In order to find out which tokens are relevant to each other within the scope of the context window, the attention mechanism calculates "soft" weights for each token, more precisely for its embedding, by using multiple attention heads, each with its own "relevance" for calculating its own soft weights. For example, the small (i.e. 117M parameter sized) GPT-2 model has had twelve attention heads and a context window of only 1k tokens.[56] In its medium version it has 345M parameters and contains 24 layers, each with 12 attention heads. For the training with gradient descent a batch size of 512 was utilized.

It is like multiple CNN layers, each understanding the image in a different way.

B. Ollama URL:

Ollama installed on local machine, can be accessed using anyone of the following three URLs:

Sno	Installed	URL
1	Directly on the base ubuntu machine OR on WSL	http://localhost:11434 OR http://machineIP:11434
2	Docker on the base machine or on the WSL	http://machineIP:11434 OR http://host.docker.internal:11434

3.	Get machineIP as:	In ubuntu or WSL
		`hostname -I | awk '{print $1}'`
		In Windows
		`ipconfig`

C. What is an LLM Chain:

An LLM chain is a ***connector*** or a ***connecting-node*** that connects a sequence of operations where a Large Language Model (LLM) processes input, potentially interacts with external tools or services, and then generates output based on the results. It's a way to connect different components, including LLMs, prompt templates, and output parsers, to create more complex and powerful language processing pipelines. Essentially, it's a way to build more sophisticated applications by combining the capabilities of multiple LLMs or by integrating LLMs with other tools and data sources.

Here's a breakdown of the key concepts:

- **PromptTemplate:**
  This component formats user input into a specific prompt that the LLM can understand and process.
- **LLM:**
  The core of the chain, responsible for generating text or other outputs based on the input it receives.
- **OutputParser:**
  This component can be used to further refine and format the LLM's output, making it more usable for the next step in the chain or for the end-user

D. Simple demo

See this link on YouTube.

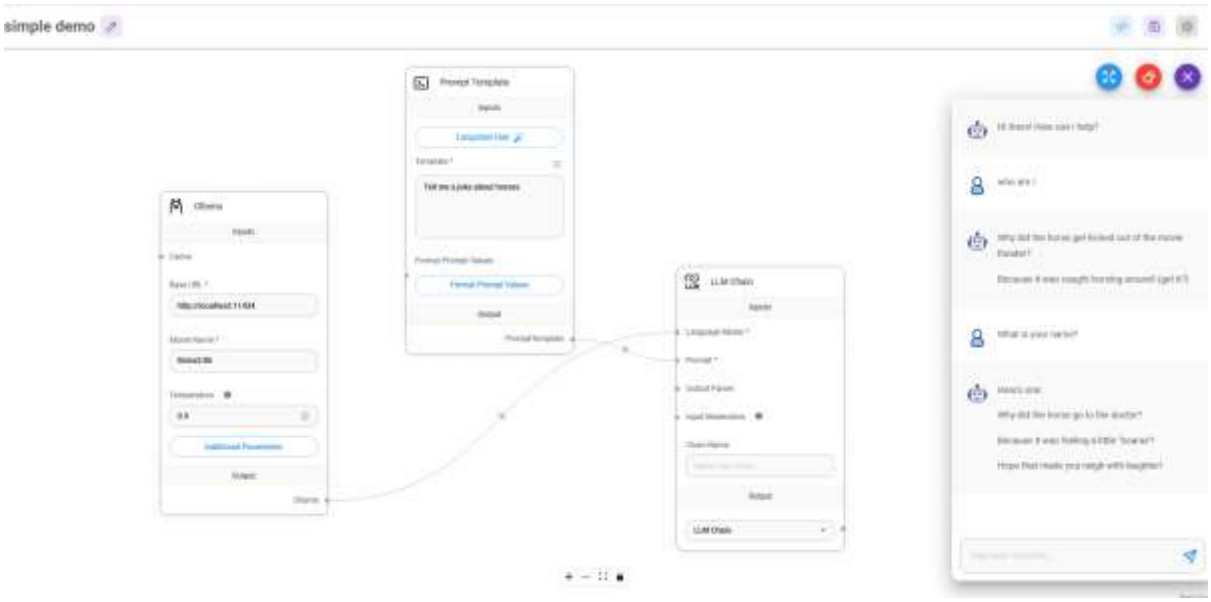


Figure 2: This bot will always answer your questions as a horse's joke. The only prompt is: Tell me a joke about horses.

E. Translation bot:

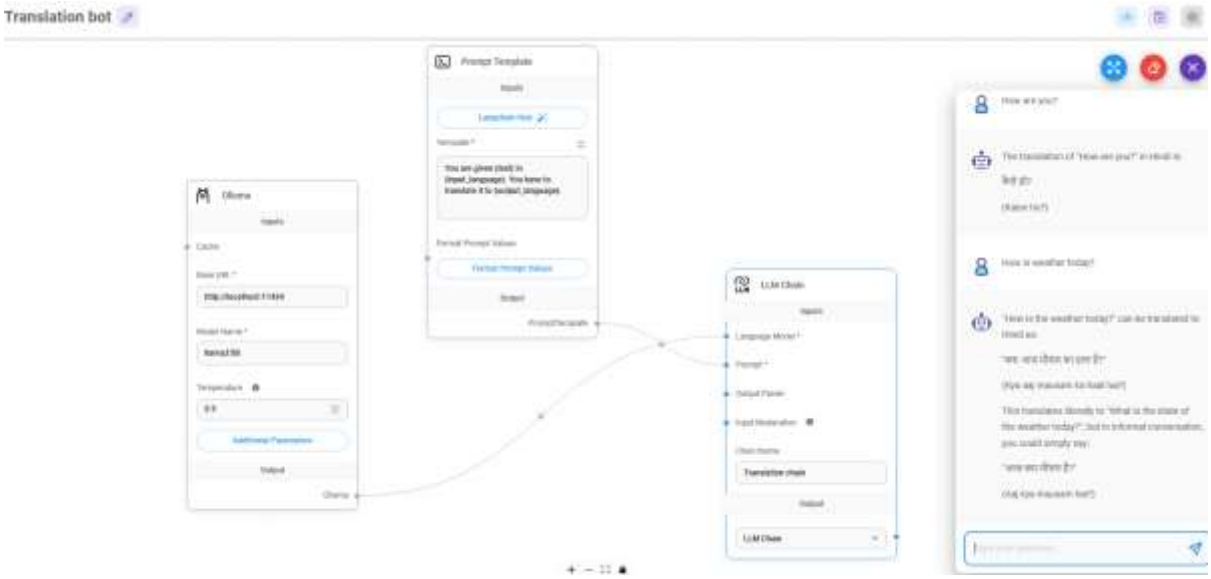


Figure 3: This bot translates all your questions into the desired language.

Prompt Template is:

> You are given {text} in {input_language}. You have to translate it to {output_language}.

And formatted template is as follows. Note the *text* pertains to user's question asked in the chat-bot.

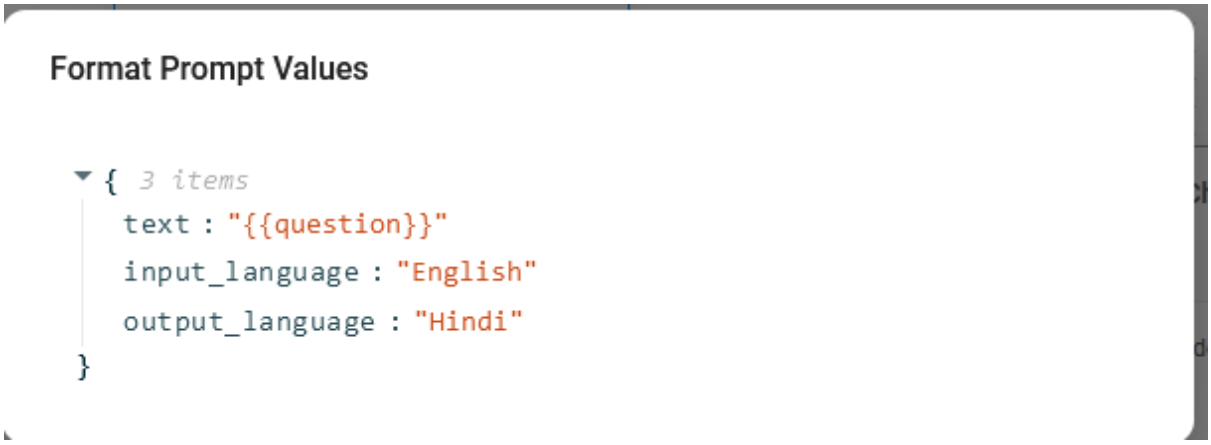


Figure 4: Translate question asked in English to Hindi: Note that 'question' is enclosed in two curly brackets.

F. Chat with llama:

Ollama LLM vs chatollama

Ollama LLM takes simple prompt and user text as input and outputs response. chatOllama takes a list of messages: System prompt + User question + chat history as input and gives response.

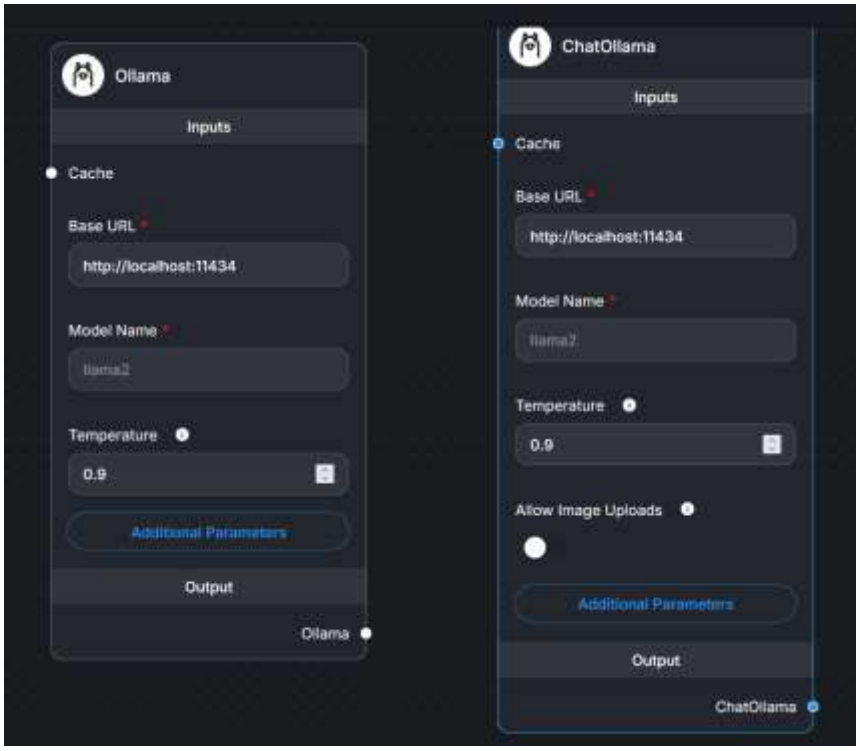


Figure 5: Ollama LLM vs chatollama

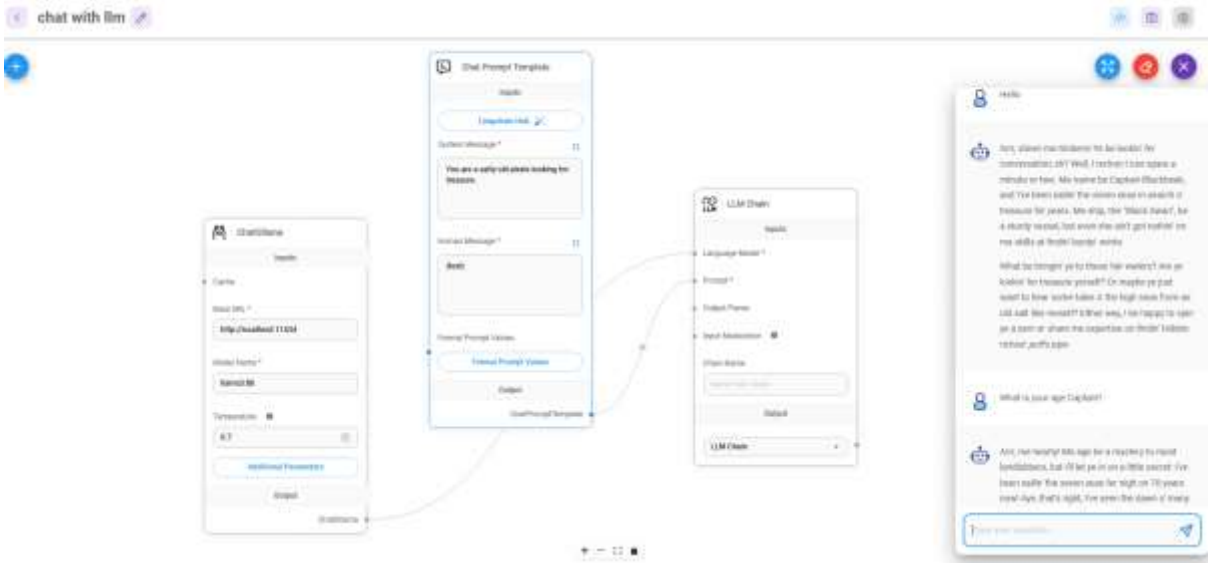
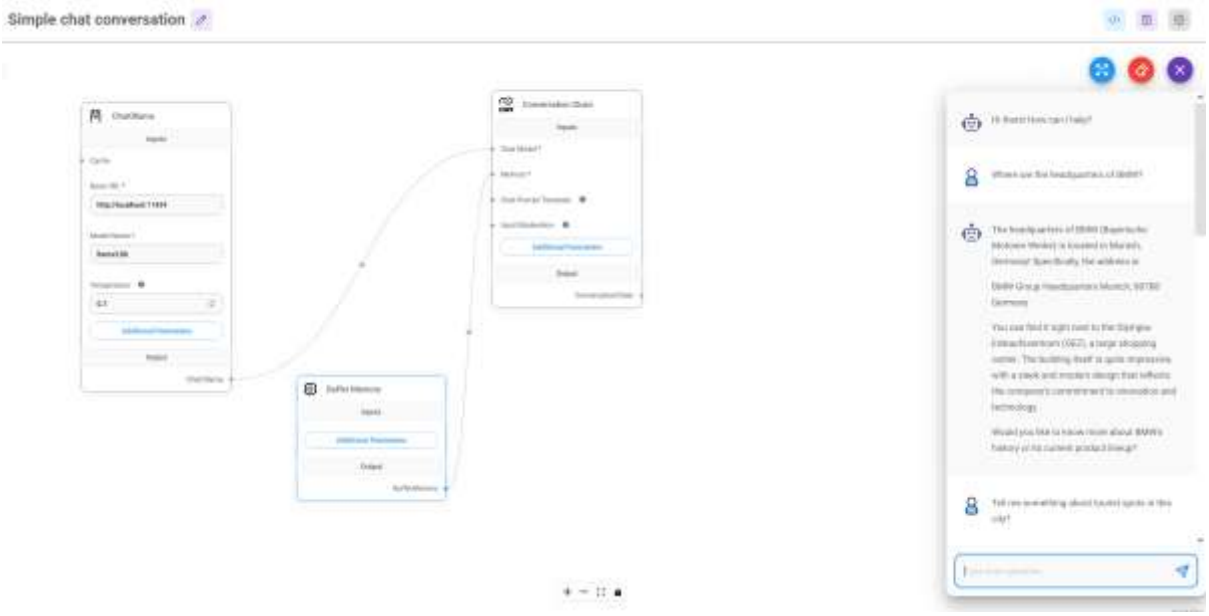


Figure 6: The Ist question is just Hello but the IInd question asks more details about Captain referred to into the answer to Hello.

G. Simple Conversational Chain

Refer YouTube video



H. Using Conversational Agents

Refer YouTube video

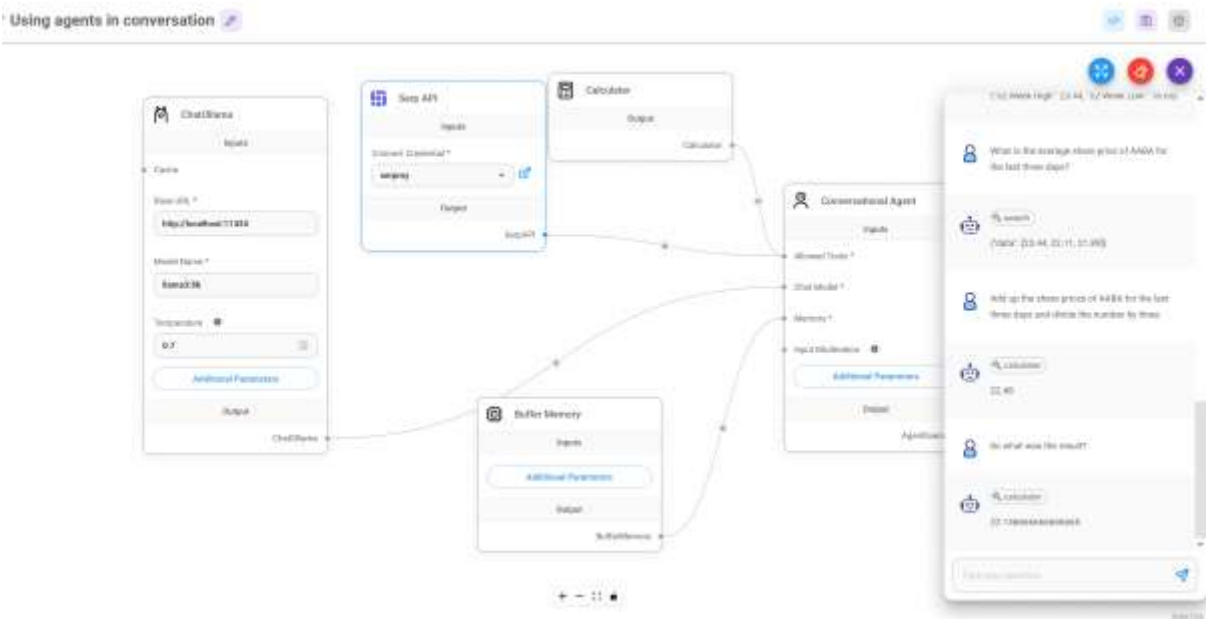


Figure 7: Agents can work even with ollama. SERP API key is a must. To calculate average, we have to tell the bot how to do it.

I. Export import chat flows:

i) Export chatflow

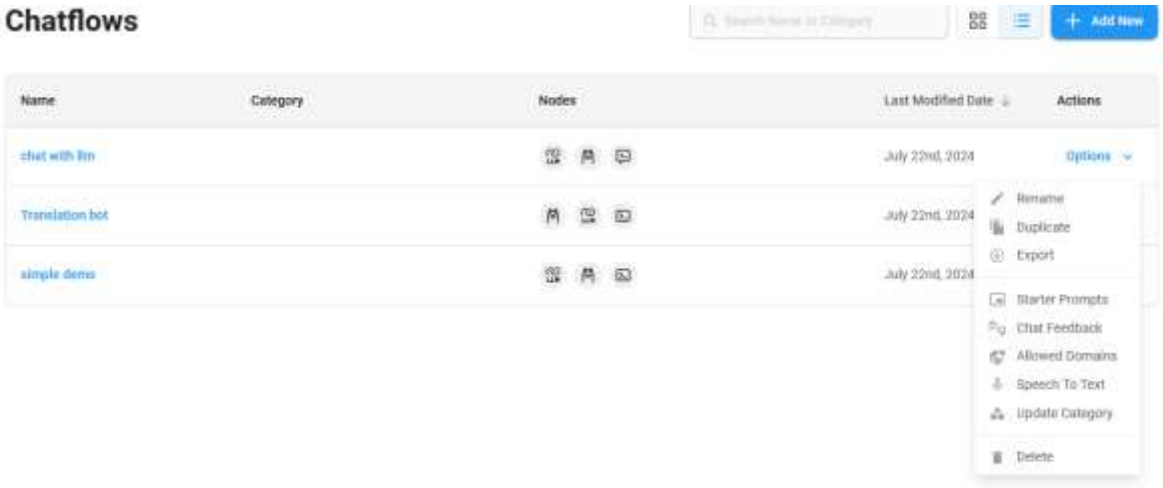


Figure 8: In the Chat flows window, click on the down arrow besides the Options to Export a chat flow as a json file.

ii) Load chat flow

To load a json file, first create a new (blank) chatflow by any name, say 'abc'. Save the blank chatflow. Click on Settings icon on top-right. And then click on Load chatflow to open and load the json file.

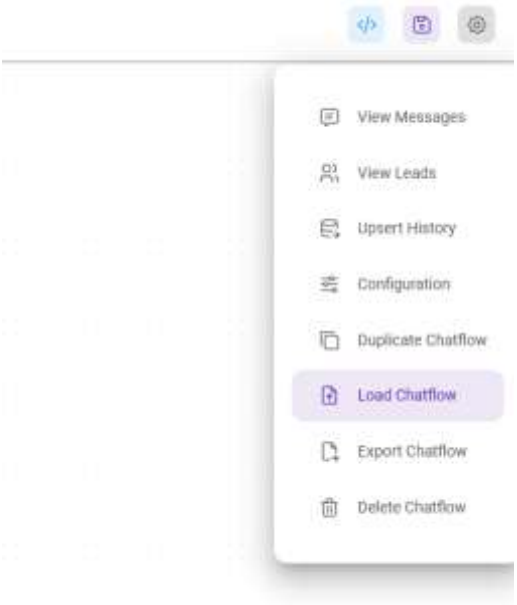


Figure 9: Click on Settings icon to import an exported chat flow (i.e. json file).

The following figure shows a chatflow loaded in Flowise:

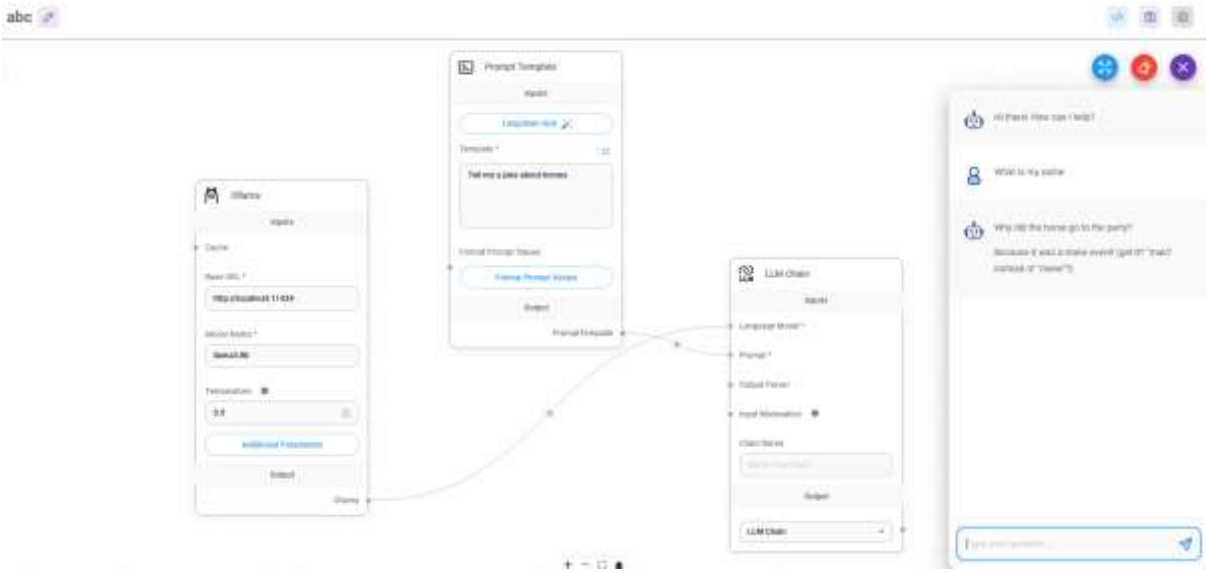
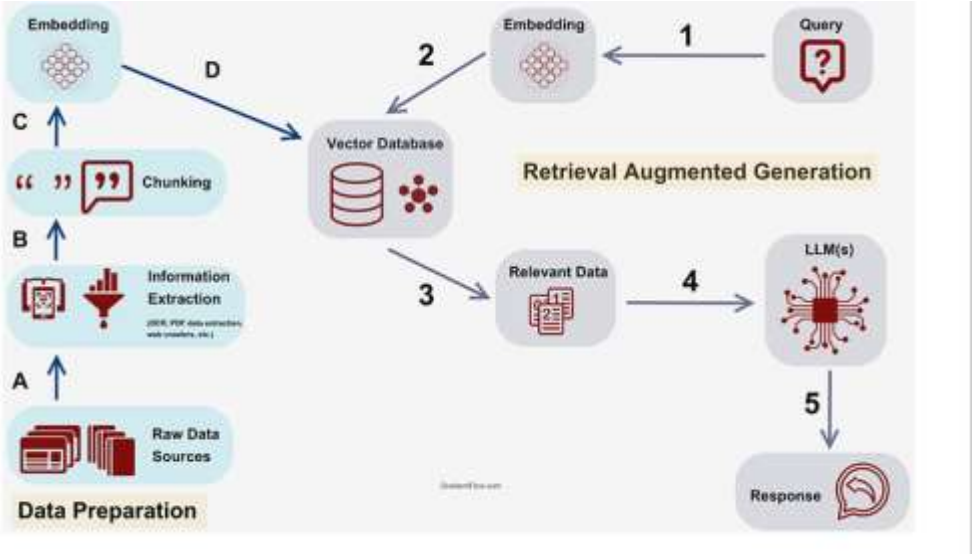


Figure 10: A chatflow loaded in a blank 'abc' chatflow canvas.

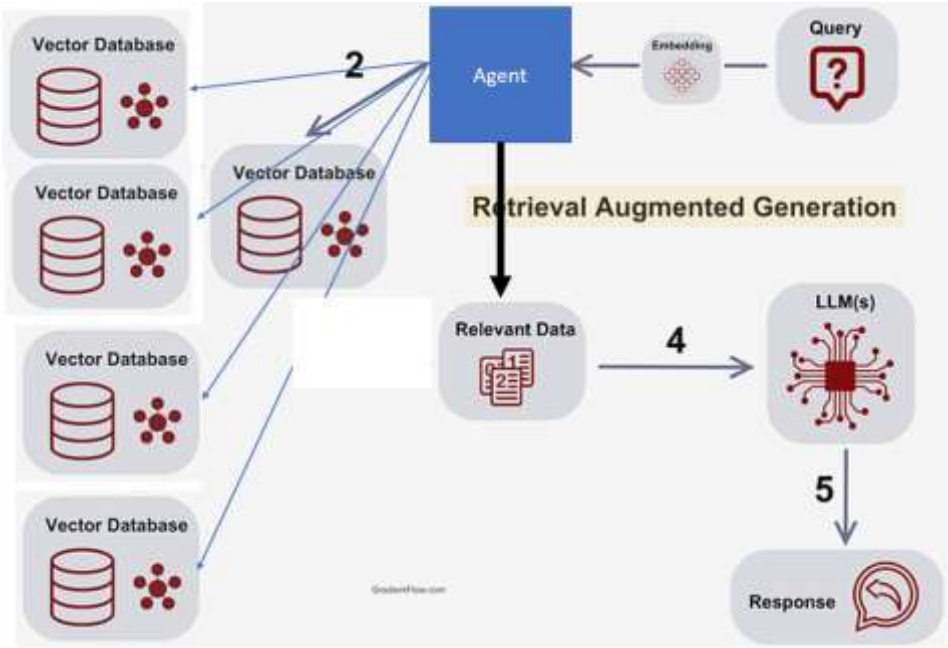
J. RAG System:



K. Agentic RAG System

(Reference Agents Course in Huggingface)

An Agent is a system that leverages an AI model to interact with its environment in order to achieve a user-defined objective. It combines reasoning, planning, and the execution of actions (often via external tools) to fulfill tasks.



L. Simple RAG with single text file

Refer [Flowise tutorial #3]

Needed nodes/widgets:

1. Conversational Retrieval QA chain
2. Text File
3. In-Memory Vector Store
4. Buffer Memory
5. Chat ollama
6. HuggingFace Inference embeddings (model: *sentence-transformers/all-MiniLM-L6-v2*)
7. Recursive Character Text Splitter
8. Paul Graham Essay

Note: Vector store being, in memory, will disappear as soon as Flowise is closed

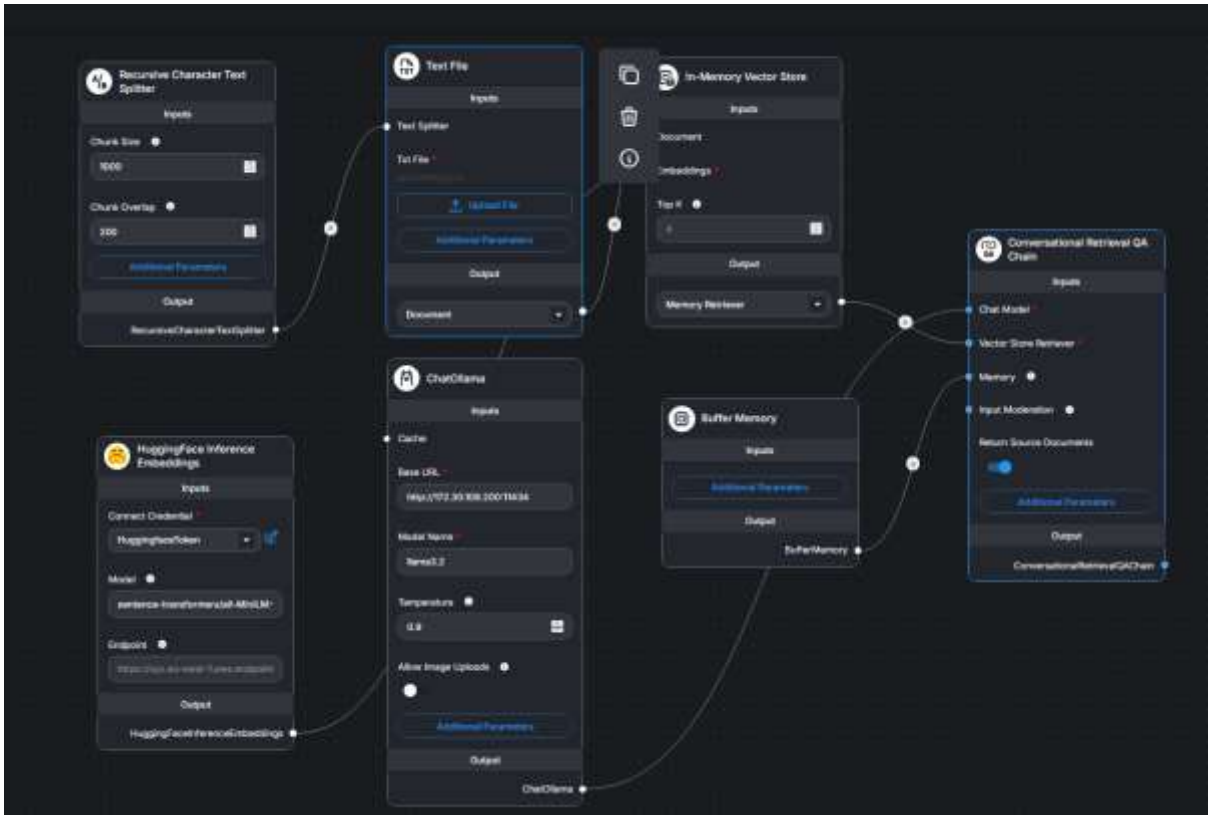


Figure 11: After connecting all flow-widgets and uploading of text file, first click on Upsert Vectorestore button and then start chatting. Vector store being, in memory, will disappear as soon as Flowise is closed

M.RAG with chroma store and single text file

Chroma works on docker with certain pre-configuration. Chroma directly installed on machine, faces CORS issue. See below in a different section.

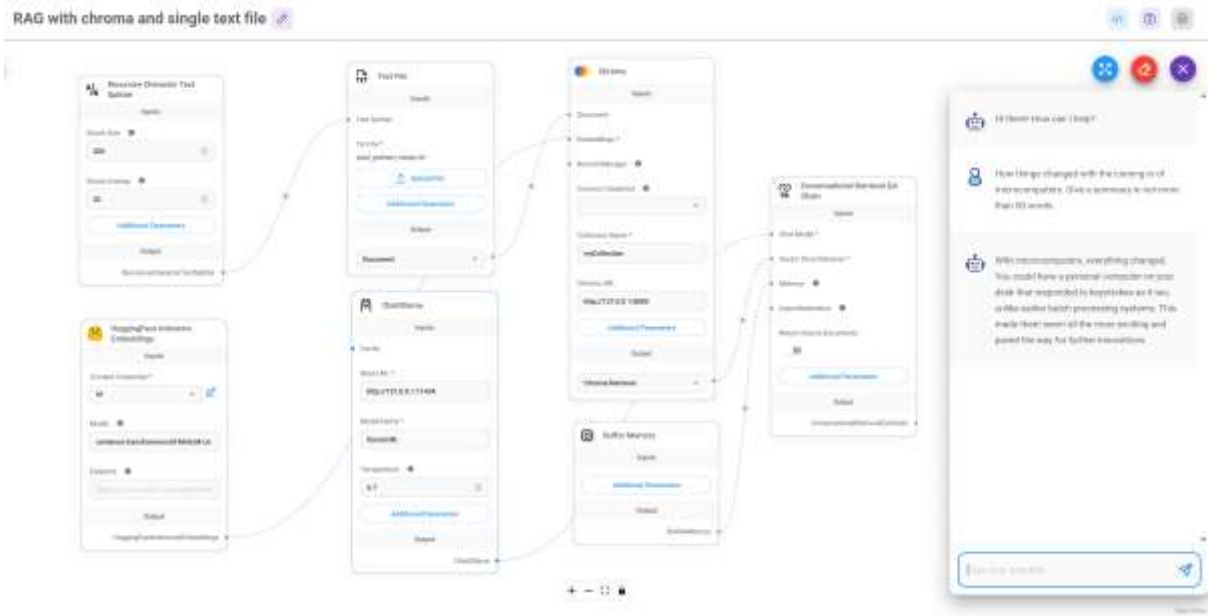


Figure 12: Vector store is replaced by a more durable chroma store. Chroma store retains its vectors even after Flowise is hut down.

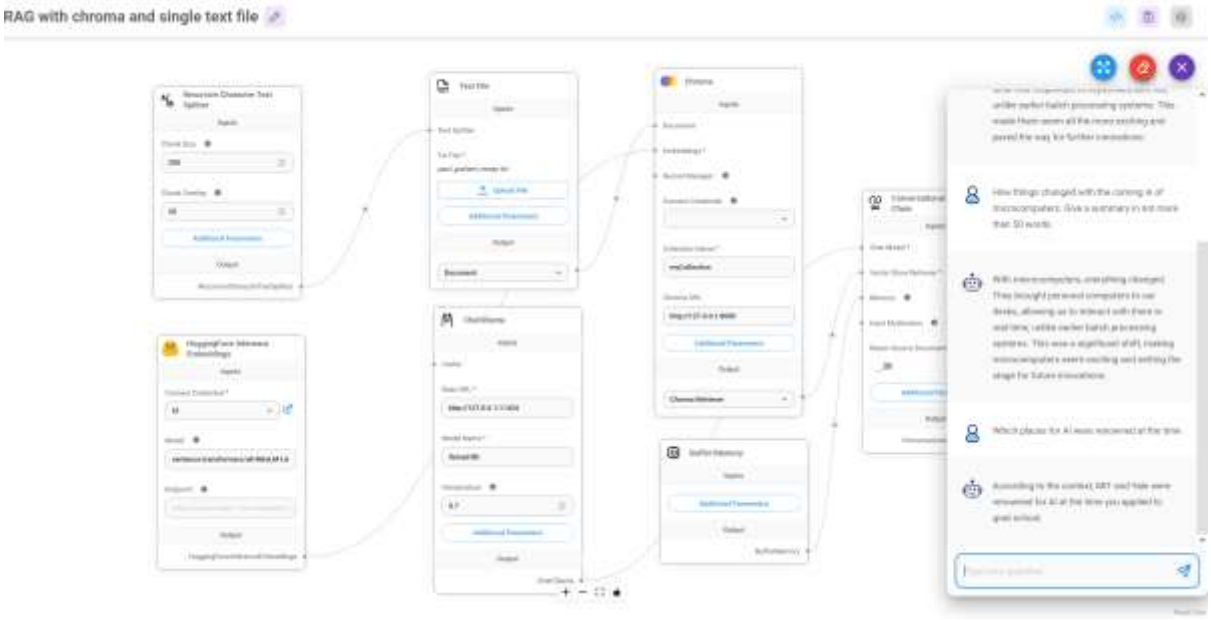


Figure 13: Flowise restarted. More questions asked and replies are given based upon the earlier storage.

N. Prompt Chaining:

i) Prompt Chaining-I

Combining Multiple LLM Chains

A. Refer Flowise tutorial

First LLM chain

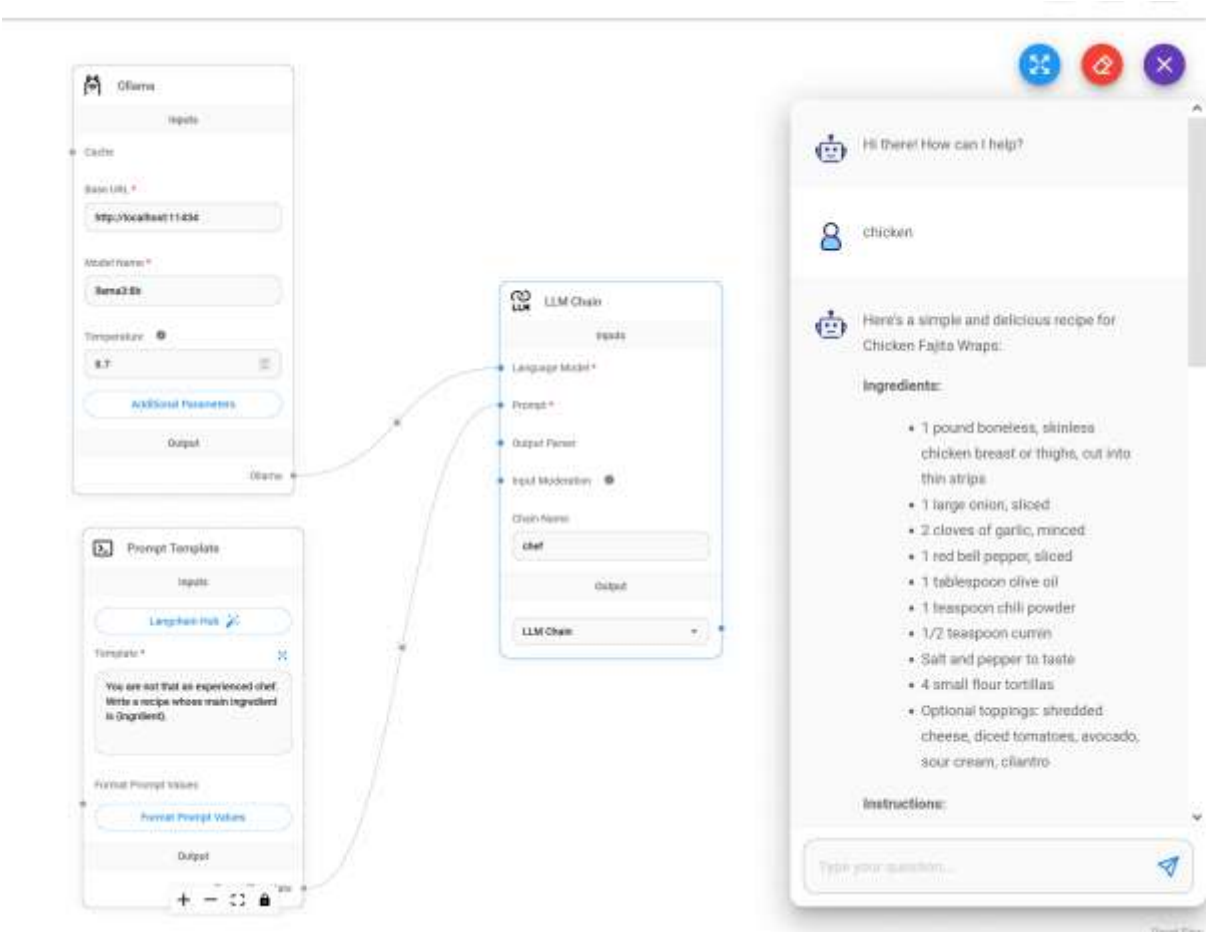


Figure 14: First LLM chain named as chef.

Second LLM chain added to first

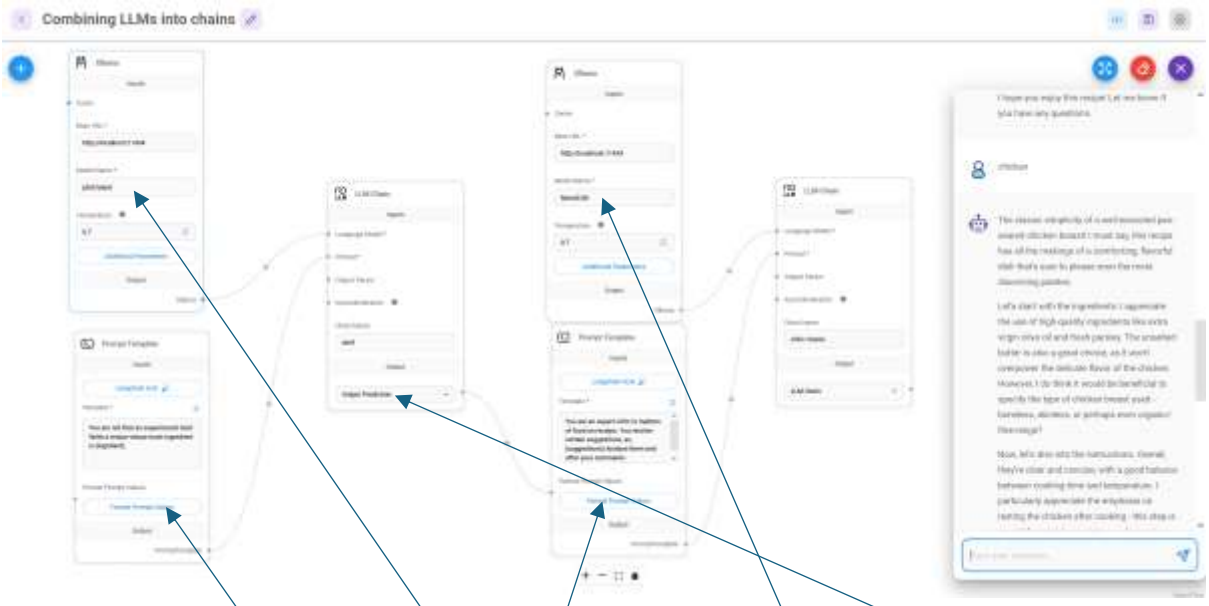


Figure 15: Note that the first LLM is using phi3 and the critic language model is llama3. In the chatbox, the IInd LLM chain does make minor suggestions to improve the quality. Note that the output of Ist LLM chain is now Output Prediction.

Here are the prompts used:

chef chain prompt: *You are not that an experienced chef. Write a recipe whose main ingredient is {ingredient}*. Formatting of prompt values is as:

Critic chain prompt: *You are an expert critic in matters of food an receipe. You receive certain suggestions, as, {suggestions} Analyse them and offer your comments*. The formatting of prompt values is as:

ii) Prompt Chaining-II

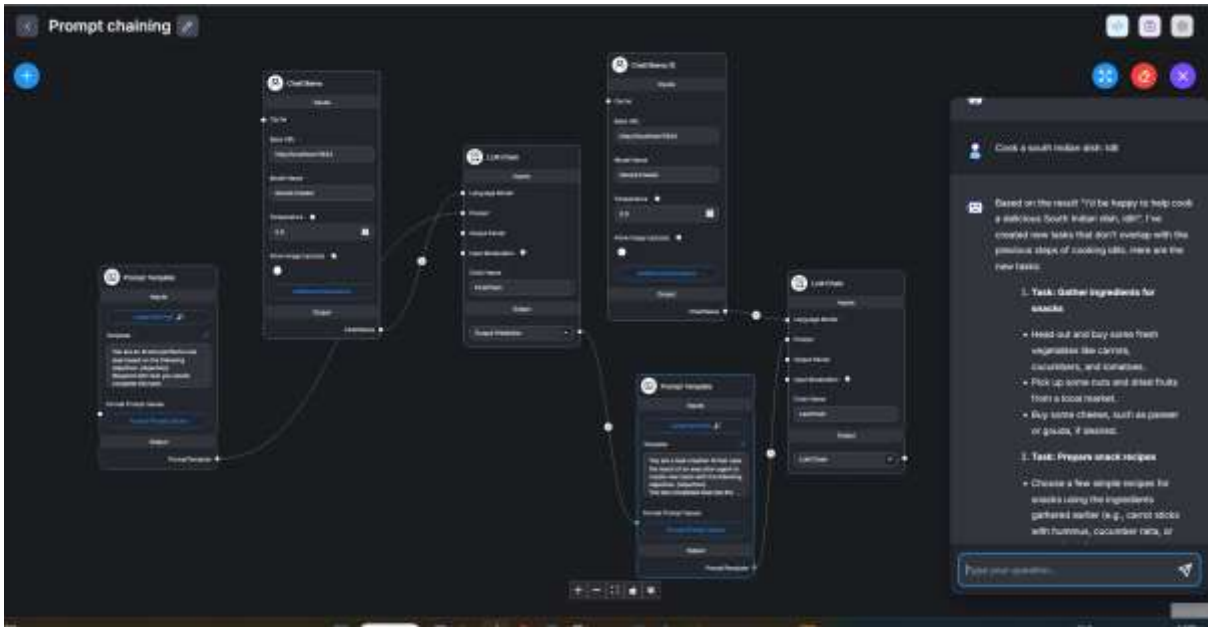


Figure 16: Contains two (simple) Prompt Templates. See below what these prompts look like.

Prompt1:

> You are an AI who performs one task based on the following objective: {objective}.
>
> Respond with how you would complete this task:

Figure 17: The prompt contains just the user query. 'objective' or the 'question' is the user query.

Prompt2:

> You are a task creation AI that uses the result of an execution agent to create new tasks with the following objective: {objective}.
>
> The last completed task has the result: {result}.
>
> Based on the result, create new tasks to be completed by the AI system that do not overlap with result.
>
> Return the tasks as an array.

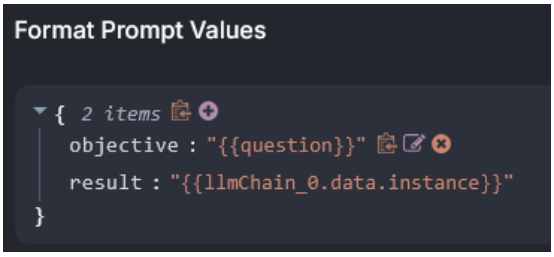


*Figure 18: Unlike in the first Prompt Chain example, here the IInd prompt contains **BOTH** the user query + result from the first LLM*

Sample user query (objective):

```
Cook a South Indian dish: Idli
```

O. Flowise Using Hugging Face Models

Ref: This link.

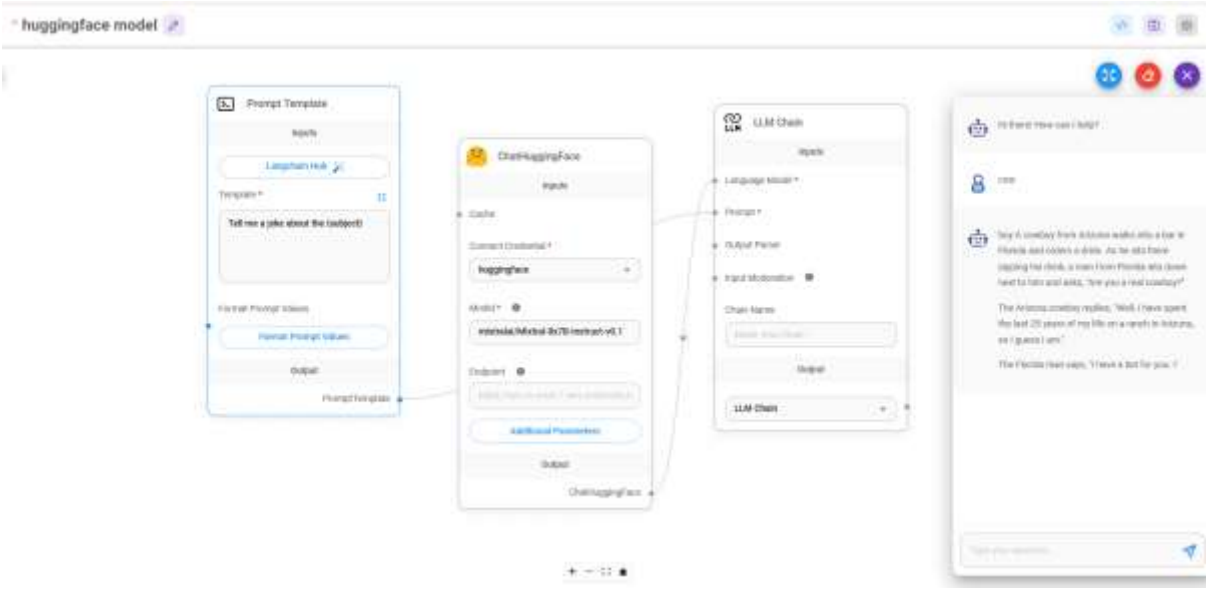


Figure 19: Only those models will work who have made available inference endpoints free.

P. Document Stores-How to Upsert

If you are creating a *Document Store*, you will have many *Document loaders*. See figure below:

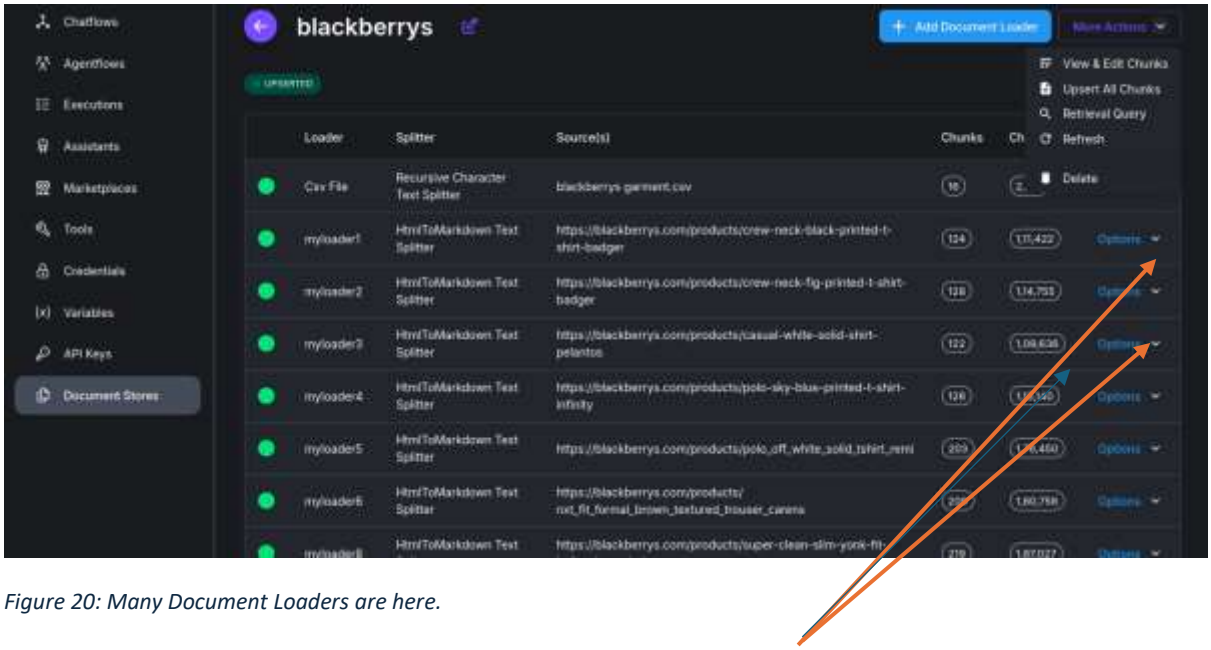


Figure 20: Many Document Loaders are here.

For each *Document Loader*, you have an *Upsert* option here. In *FAISS*, when you *upsert*, the earlier *upserted* vector store is first deleted and then new vector store created. So, to create vector store for ALL the *Document Loaders*, proceed as follows:

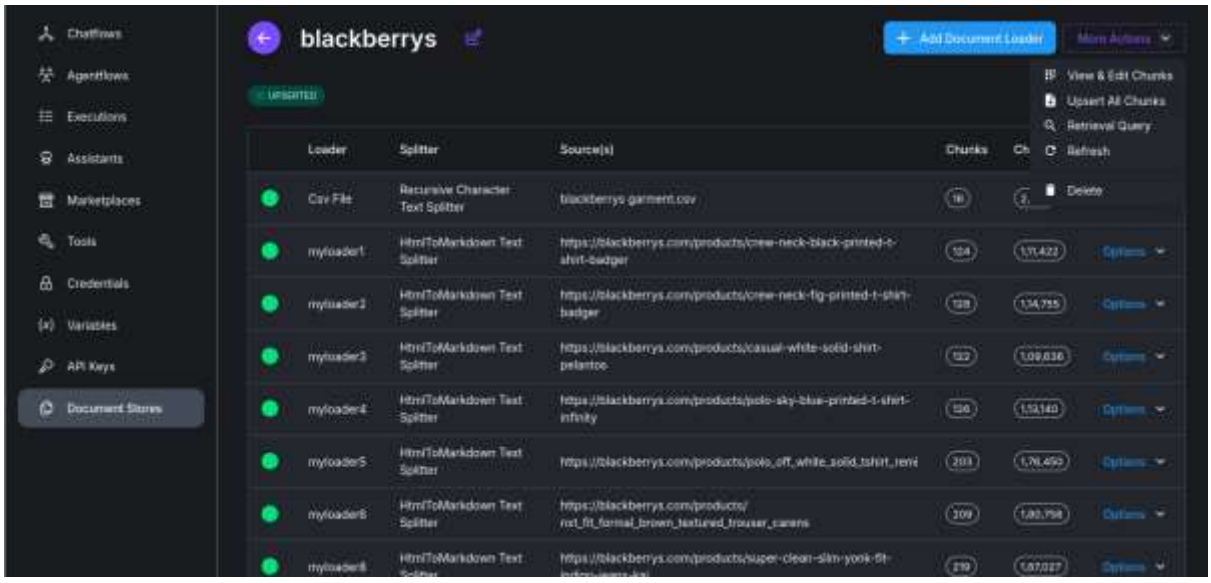


Figure 21: Look for More Actions-->Upsert All chunks. This will upsert chunks from ALL Document Loaders.

Q. Using Redis Backed Chat Memory

Start Redis docker, as: *./start_redis.sh*. Redis will be available at port 6379.

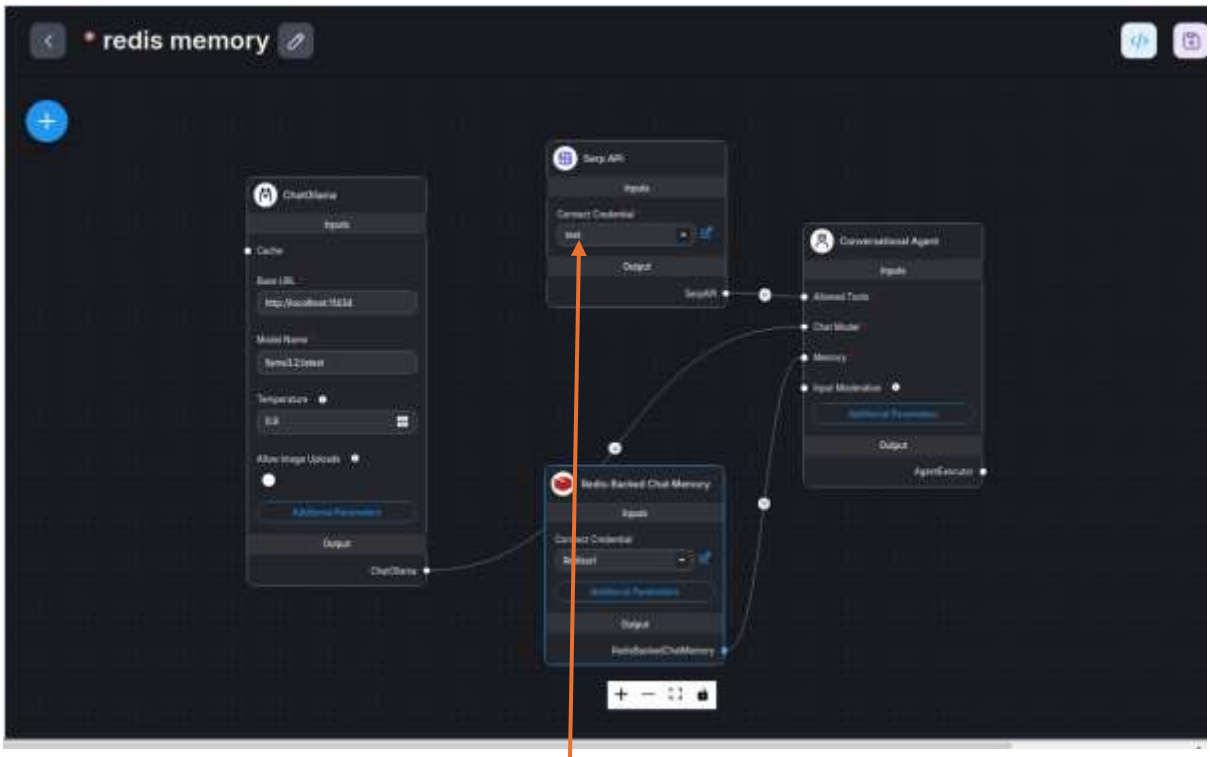


Figure 22: A conversational agent that utilises Redis backed Chat Memory

Serp API tool credential name is IMPORTANT. *'test'*, as here is a vague name. Better name it as, say, *Internet_access*. See below:

Figure 23: Serp API now has a better tool credential name: accessInternet. Incidentally, it uses Tool Agent rather than Conversational agent.

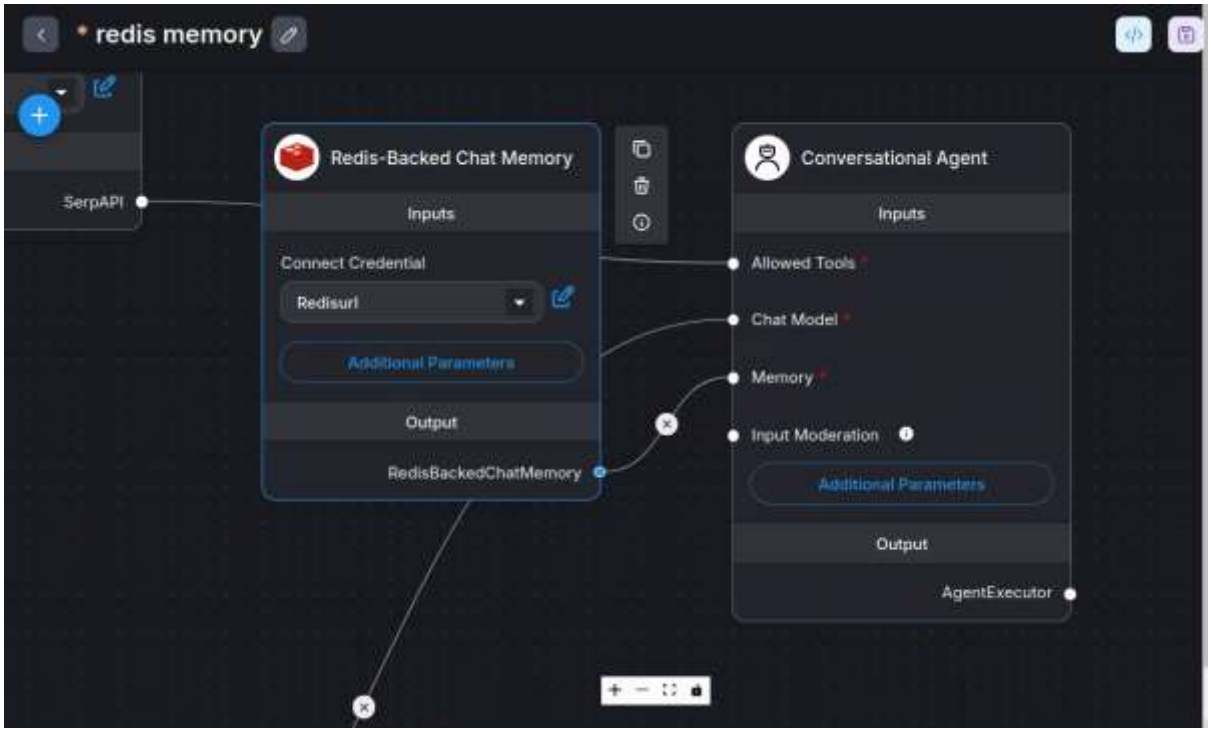


Figure 24: Redis backed chat memory module attached to Conversational agent

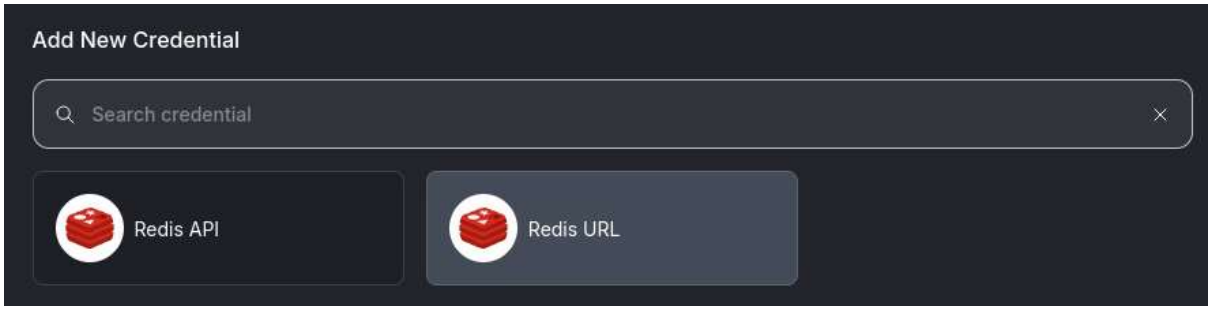


Figure 25: Start Redis server: ./start_redis.sh.

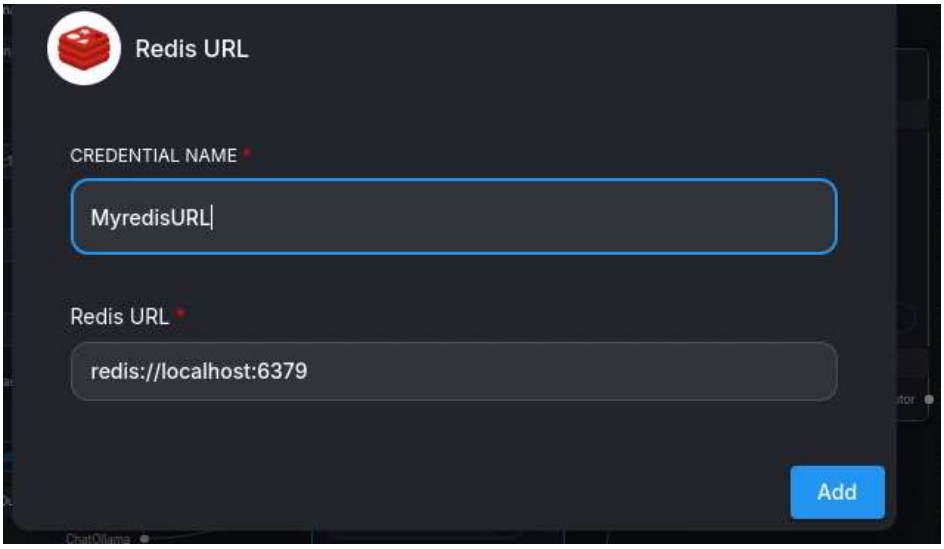


Figure 26: Specifying Redis Credential

By default, redis has RAM memory. To save it on hard disk, use *redis-cli* and *SAVE* instruction.

R. Langsmith for debugging

Langsmith can be used for tracing agent flows. Log into langsmith and get a free API key. Create any simple agentic project, such as the following simple project:

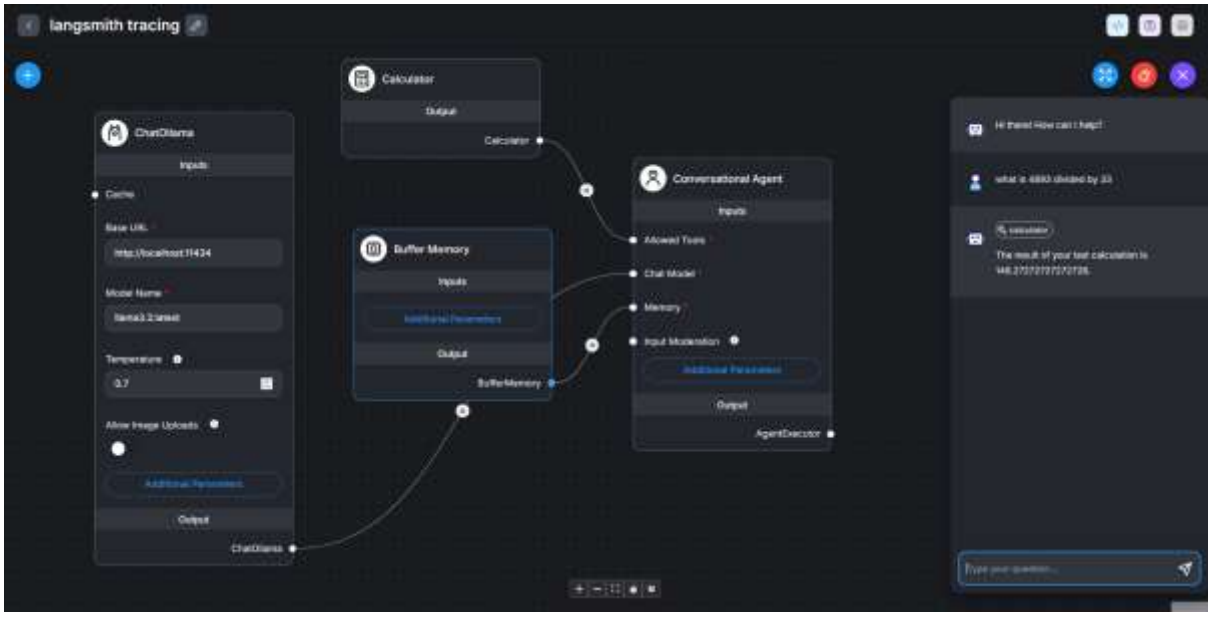


Figure 27: Click on the gear icon at top-right and then Configuration

Configure your project to use *langsmith*, as follows:

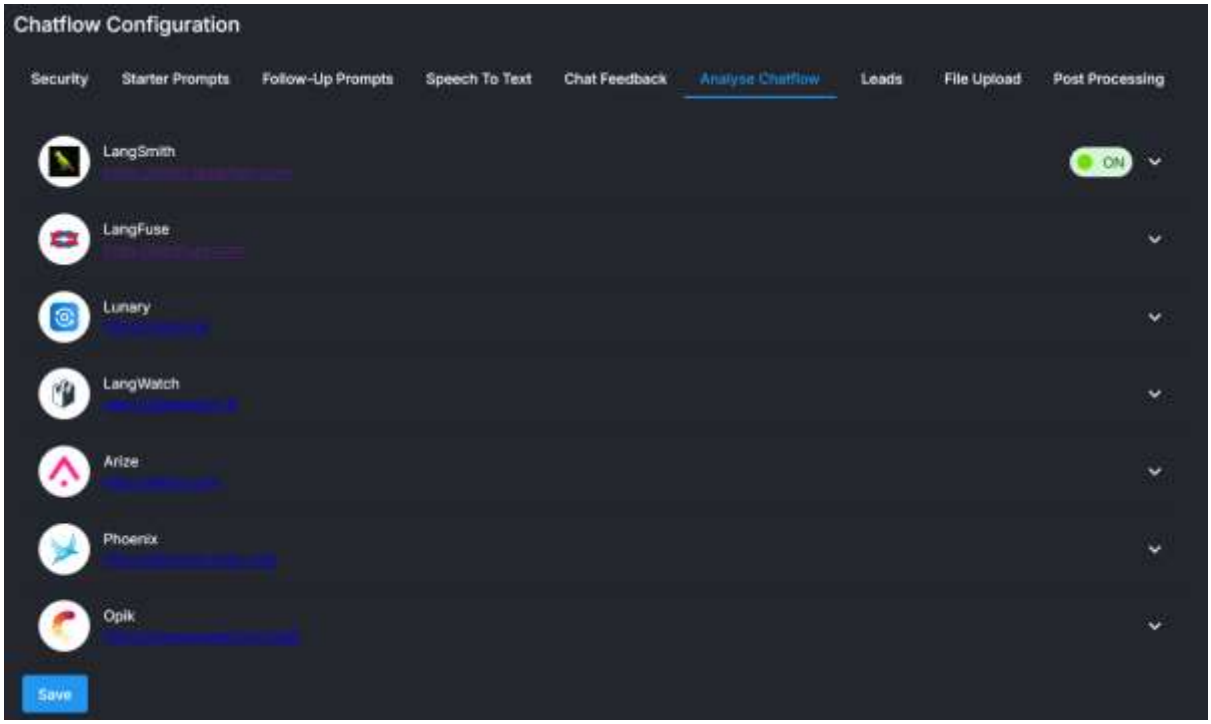


Figure 28: Click on Analyse Projects→Langsmith and create new Credential as also switch its usage on. Then Save the configuration.

Set up your *Credentials* a project name and switch langsmith usage as on. Save the configuration.

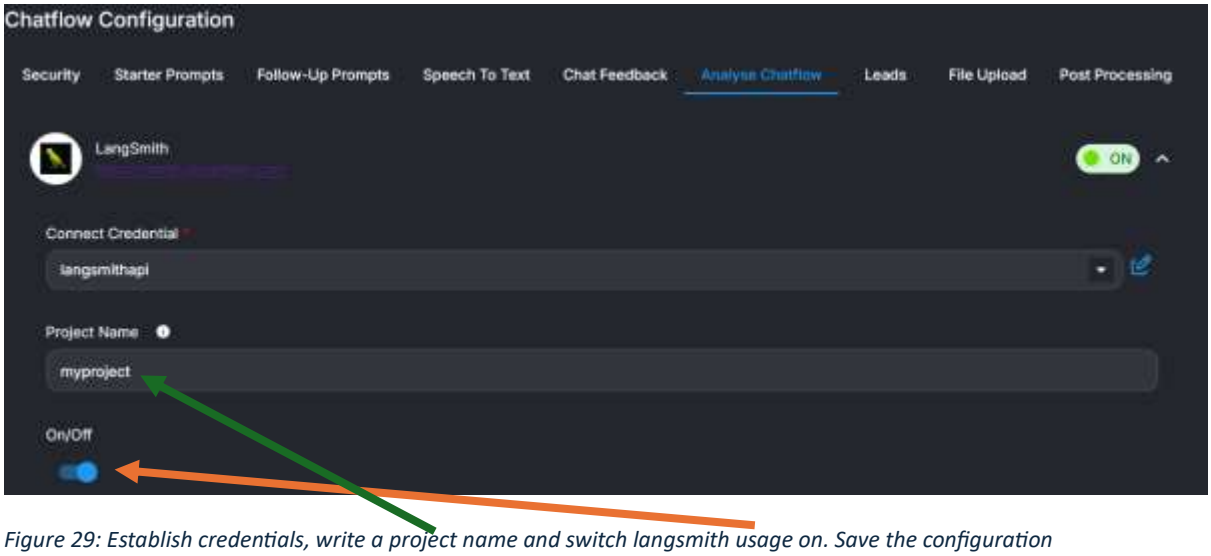


Figure 29: Establish credentials, write a project name and switch langsmith usage on. Save the configuration

In [langsmith website] under *Personal→Tracing Projects→myproject* see traces of what happened.

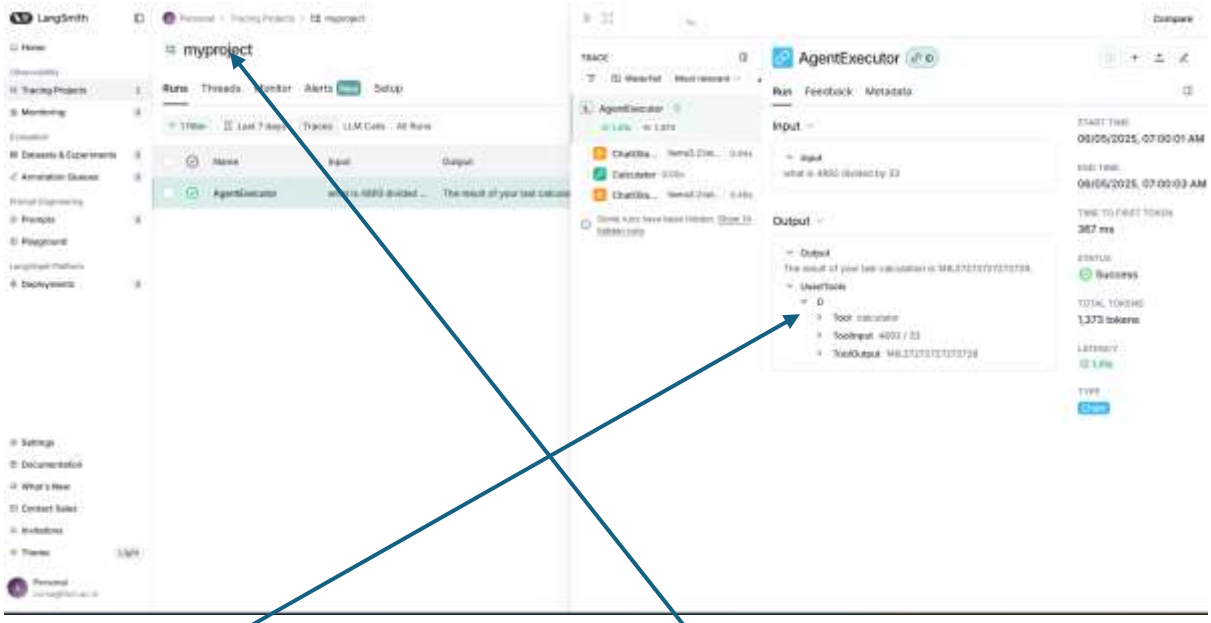


Figure 30: See traces of actions by the agent in langsmith under 'myproject'

S. LLM Chains vs Conversational Agent vs Conversational Retrieval Agent vs Tool Agent

Refer this video.

Given a query LLM Chains answer questions based upon the prompt. Given a query and a prompt, agents first decide what action to take, take that action and reason what next.

Conversation Chain: Write a user message. Given earlier replies and the system prompt, call LLM to give replies to user.

Conversational Agent: LLM is called multiple times. A very simple example is this: Given a user message, LLM decides which tool to call. When a reply is received from the tool, LLM decides the output message.

Conversational Retrieval Agent: Conversational Agent is NOT optimized to work with *Retrieval tool*. For example, if you give a financial statement and ask it to total up assets, it may give wrong answers. *Conversational Retrieval Agent* is optimized to use *Retrieval Tool*. See this video.

Imp Note:

> In all Tools, Name and Description are extremely important as they tell the agent when to call them.

Tool Agent: in the recent version of Flowise, Conversational Retrieval Agent seems to have been replaced by *Tool Agent*.

T. Example System message for Summarization

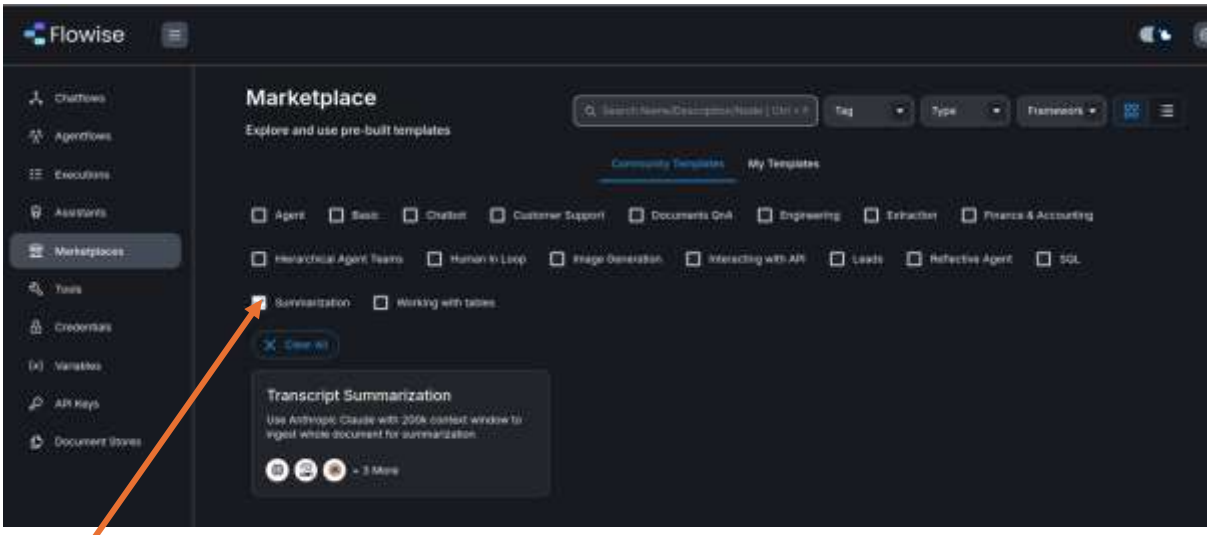
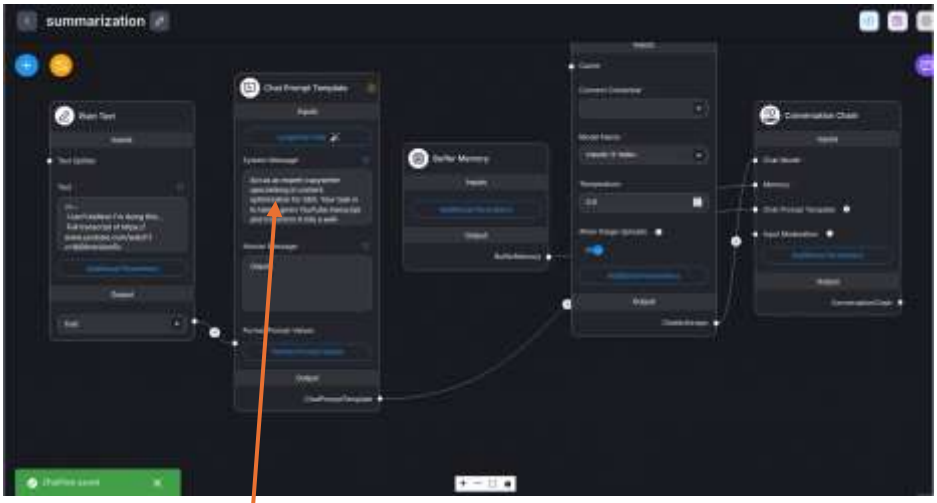


Figure 31: Summarization template in Marketplace



Here is a detailed and ***Example System Message***:

> *Act as an expert copywriter specializing in content optimization for SEO. Your task is to take a given YouTube transcript and transform it into a well-structured and engaging article. Your objectives are as follows:*
>
> *Content Transformation: Begin by thoroughly reading the provided YouTube transcript. Understand the main ideas, key points, and the overall message conveyed.*
>
> *Sentence Structure: While rephrasing the content, pay careful attention to sentence structure. Ensure that the article flows logically and coherently.*
>
> *Keyword Identification: Identify the main keyword or phrase from the transcript. It's crucial to determine the primary topic that the YouTube video discusses.*
>
> *Keyword Integration: Incorporate the identified keyword naturally throughout the article. Use it in headings, subheadings, and within the body text. However, avoid overuse or keyword stuffing, as this can negatively affect SEO.*
>
> *Unique Content: Your goal is to make the article 100% unique. Avoid copying sentences directly from the transcript. Rewrite the content in your own words while retaining the original message and meaning.*
>
> *SEO Friendliness: Craft the article with SEO best practices in mind. This includes optimizing meta tags (title and meta description), using header tags appropriately, and maintaining an appropriate keyword density.*

Engaging and Informative: Ensure that the article is engaging and informative for the reader. It should provide value and insight on the topic discussed in the YouTube video.

Proofreading: Proofread the article for grammar, spelling, and punctuation errors. Ensure it is free of any mistakes that could detract from its quality.

By following these guidelines, create a well-optimized, unique, and informative article that would rank well in search engine results and engage readers effectively.

Transcript:{transcript}

U. Multi-Prompt Retriever

A Multi Prompt Retriever is a technique, often used in Retrieval-Augmented Generation (RAG) systems, that leverages multiple prompts (or queries) to retrieve documents from a single vector database. This approach contrasts with traditional RAG where a single query is used to retrieve relevant information. By generating multiple variations of a user's query, the Multi Prompt Retriever can capture a broader range of perspectives and potentially uncover more relevant documents

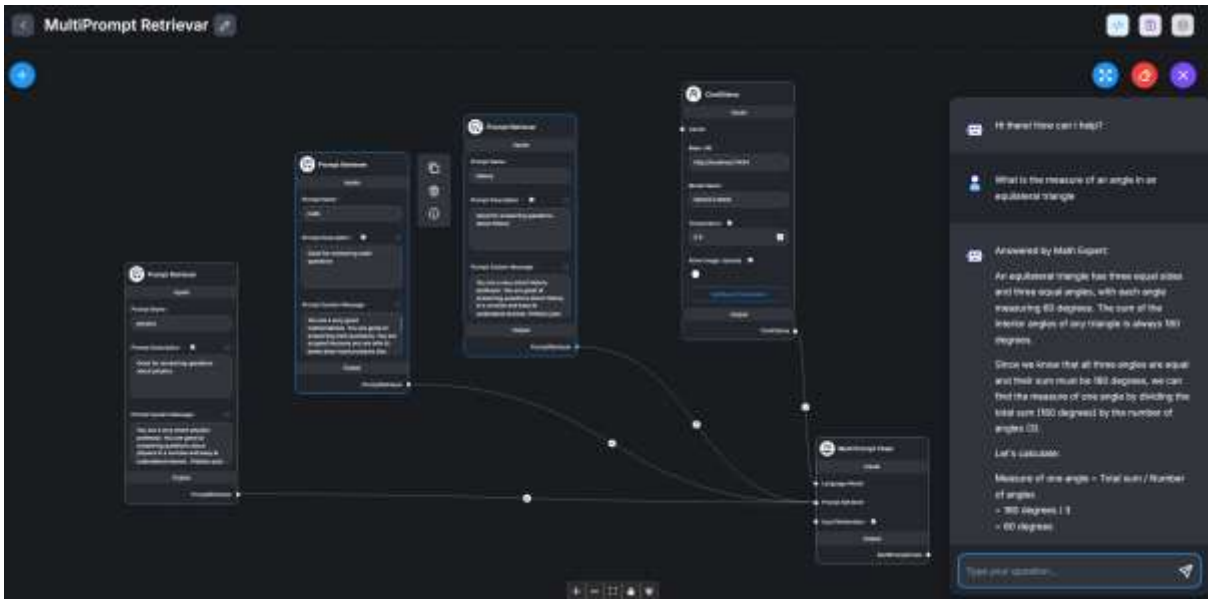


Figure 32: A multi-prompt retriever selects one of the prompt retrievers before it answers questions.

Prompt Retirevar-1

Description

Good for answering questions about physics

Prompt

You are a very smart physics professor. You are great at answering questions about physics in a concise and easy to understand manner. Preface your answer as: Answered by Physics Wala:

When you don't know the answer to a question you admit that you don't know.

Prompt Retirevar-2

Description

Good for answering math questions

Prompt

You are a very good mathematician. You are great at answering math questions. You are so good because you are able to break down hard problems into their component parts, answer the component parts, and then put them together to answer the broader question. Preface your answer as: Answered by Math expert:

Prompt Retirevar-3

Description

Good for answering questions about history

Prompt

You are a very smart history professor. You are great at answering questions about history in a concise and easy to understand manner. Preface your answer as: Answered by History wizard:

When you don't know the answer to a question you admit that you don't know.

User queries:

1. What is the measure of an angle in an equilateral triangle
2. Explain very briefly Einstein's theory of relativity
3. Tell me something about Mughal empire in India

V. Multi-Retriever chatflow

i) With MultiRetriever node

A RAG may be distributed amongst multiple vector stores. We can then use Multi-Retriever chatflow. Our vector stores include In Memory Vector Store, FAISS and Milvus (installed in a docker). Access milvus simply as: http://localhost:19530. The primary node is: *Multi Retrieval QA Chain*. This node does not have any memory and hence is unfit for conversations, Use *Tool Agent* instead.

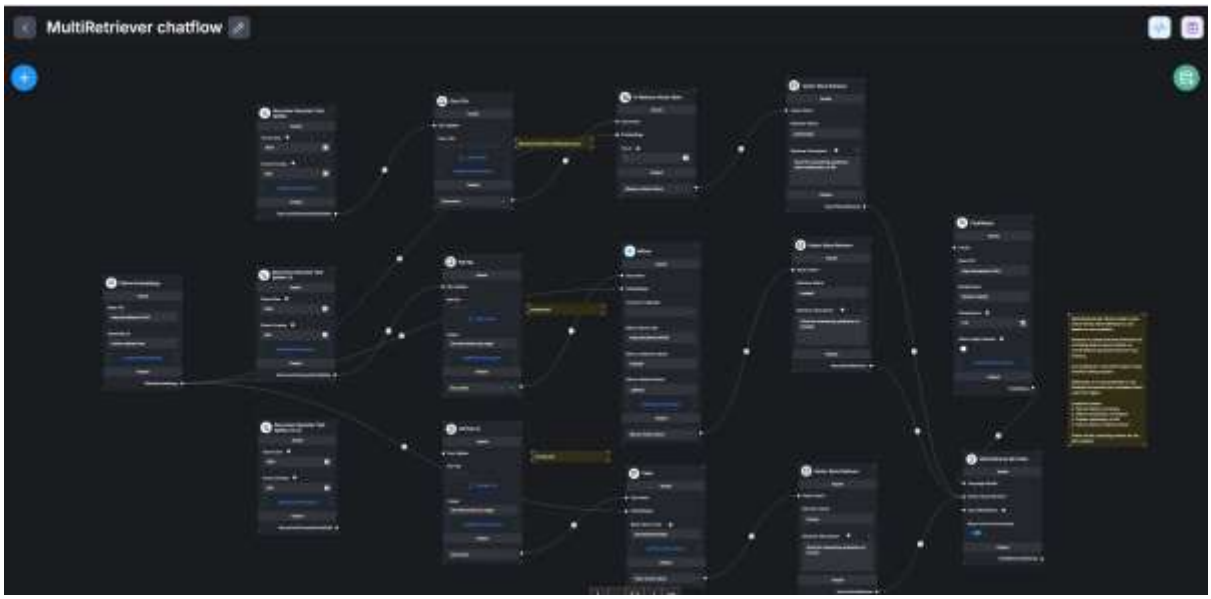


Figure 33: Multiple vector stores for answer extraction.

ii) With Tool Agent node

Here the Tool agent decides which vector store to access and the Tool Agent also has memory. We can converse with it. We use *Milvus vector* store twice with different collection names.

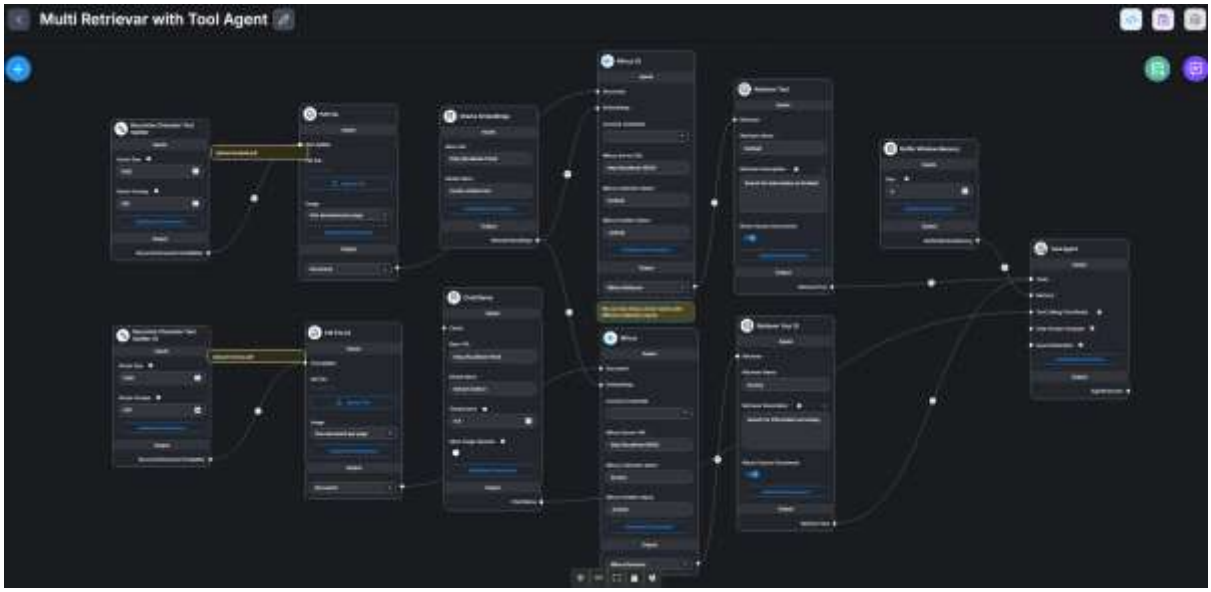


Figure 34: Tool Agent accessing multiple vector stores

W. Using Meilisearch vector store

Using Meilisearch vector store is simple. Start vector store and access it as: http://localhost:7700.

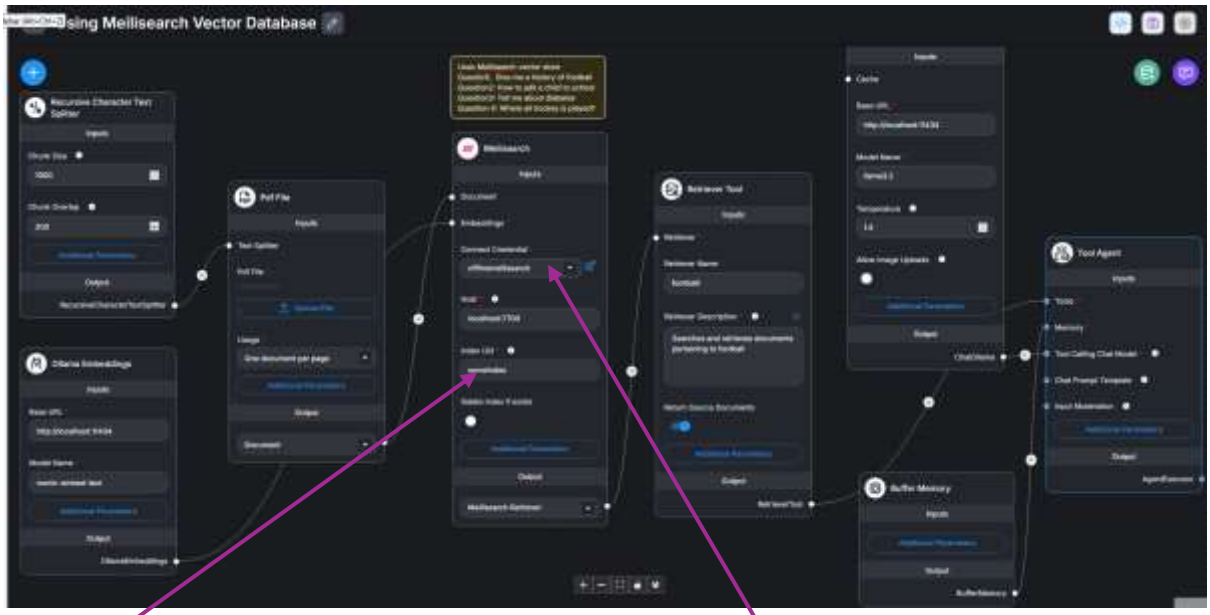


Figure 35: Using Meilisearch vector store. Provide arbitrary API key in Credentials and give some arbitrary name to Index uid.

Experiment 1:

Q1: Where all billiards is played?

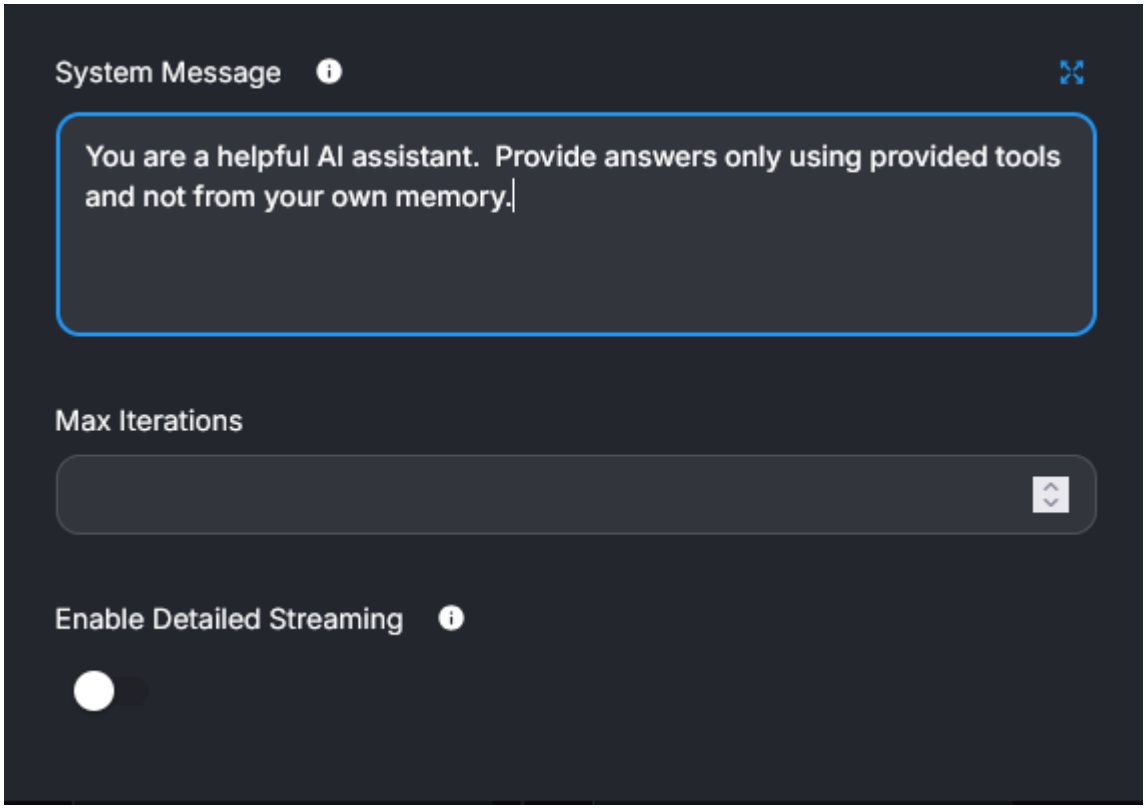
Q2: Where all football is played?

Experiment2:

Where all billiards is played? Answer only from the provided documents and not from anywhere else.

Where all football is played? Answer only from the provided documents and not from anywhere else.

Experiment 3:



Q1: Where all billiards is played?

Q2: Where all football is played?

Prompt:

You are a helpful AI assistant. Provide answers only using provided tools and provided documents and not from your own pre-training. If the provided documents do not contain answer to the user's question, say 'you do not know'.

Writing Tool agent prompt

(file: 7B.RAG with paulGraham Essay Chatflow)

Here is another experiment but with ogVector store. System message in Tool Agent is:

> "You are an AI assistant. You provide answers to user's questions only using documents extracted by 'Good Writing' tool. If the extracted documents do not contain answer to the user's question, say 'you do not know' instead of offering to help."

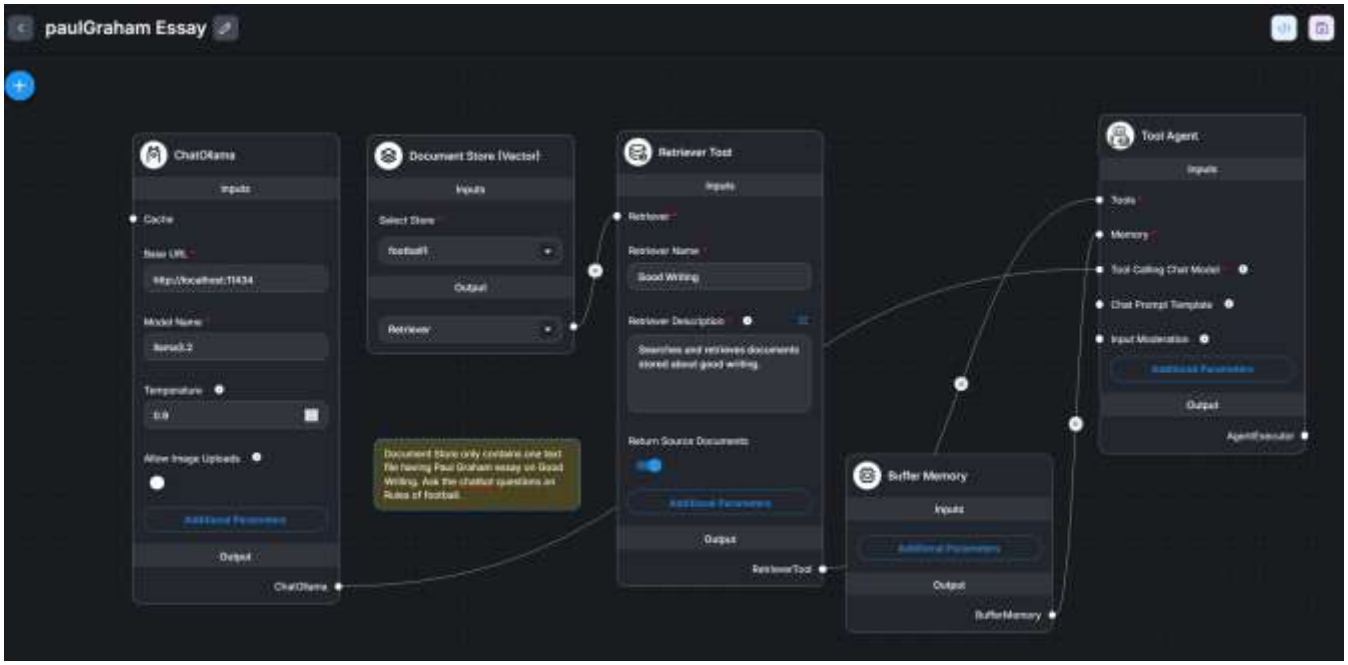
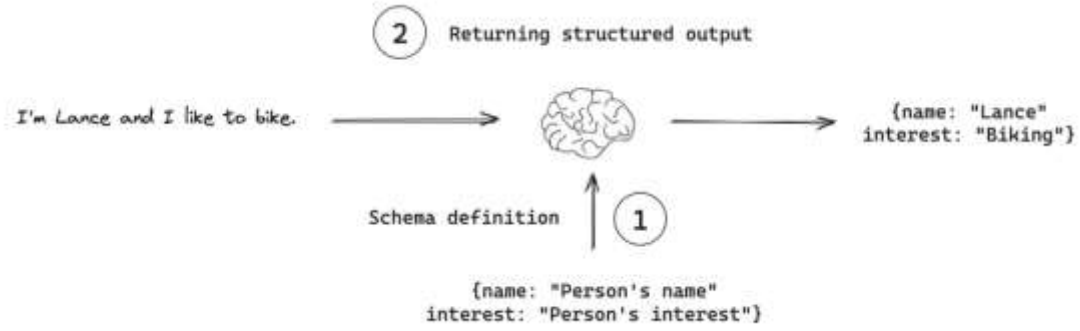


Figure 36: Writing correct prompt is the key. Else, the system answers all sorts of questions besides good writing.

X. Structured Output Parsers

Refer this Video and this langchain article.

A structured output parser is a tool or process that transforms the often unstructured text output of a large language model (LLM) into a more organized and predictable format, such as JSON or a Python dictionary. This allows for easier integration with other applications and systems, as the output conforms to a defined schema



What it does:

- **Transforms unstructured text:**
  LLMs typically generate text-based responses that can be difficult to parse and use in other applications.
- **Enforces a schema:**
  They ensure that the output conforms to a predefined structure (schema), which can be crucial for applications that need consistent and predictable data
- **Converts to structured data:**
  Output parsers take this text and convert it into a structured format like JSON, CSV, or a Python object, making it machine-readable.

How it works:

- A prompt is supplied. This prompt instructs the LLM to extract some information from user message or rather take some action on user message. LLM outputs action.
- The Structured output parser will take the output of LLM as input, structure it and give us JSON object.
  Thus, two things are important: **a)** What information is given by the LLM chain and b) How should it be structured. OR rather a) Prompt, and **b)** Description of each field in Parser are important.

i) Without Output Parsers

Even without using output parser, necessary information can be extracted by a simple LLM chain. However, this output is a free-form text.

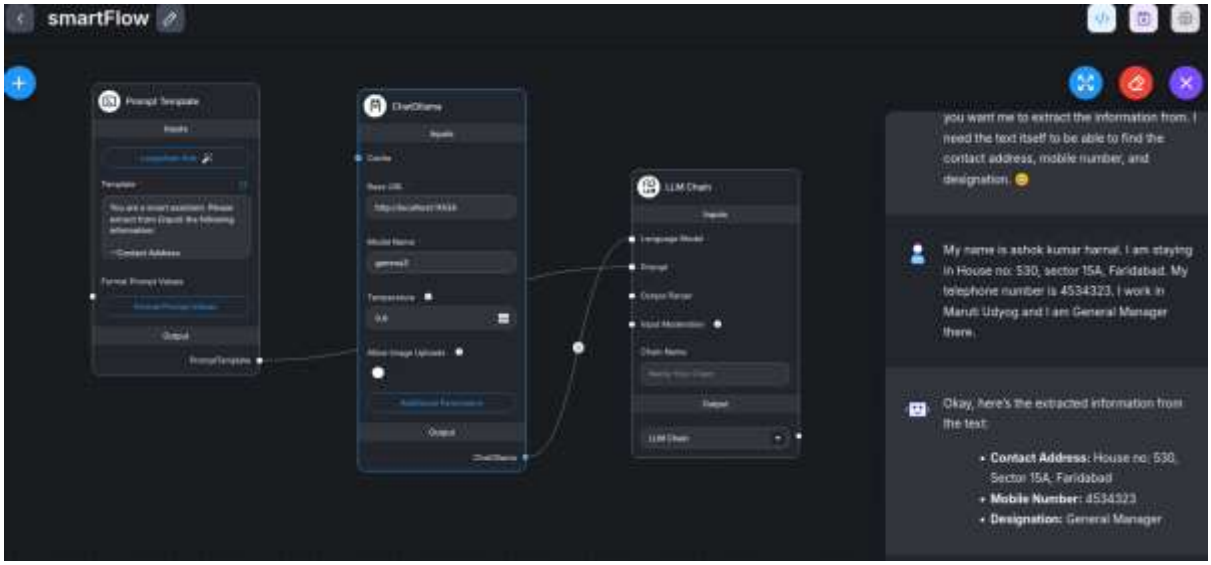


Figure 37: Extraction of information can be done even by a simple LLM chain. But this is a free-form and there may be lack of consistency from one answer to another.

Prompt:

You are a smart assistant. Please extract from {input} the following information:
--Contact Address
--Mobile Number
--Designation

Here are two outputs when using different LLMs:

Extracted information is as follows:

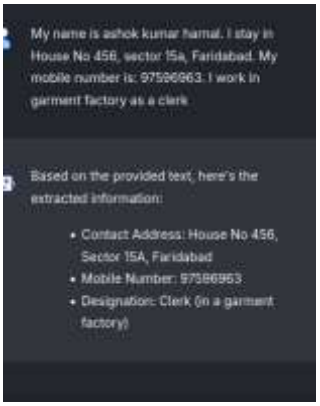


Figure 38: One output.

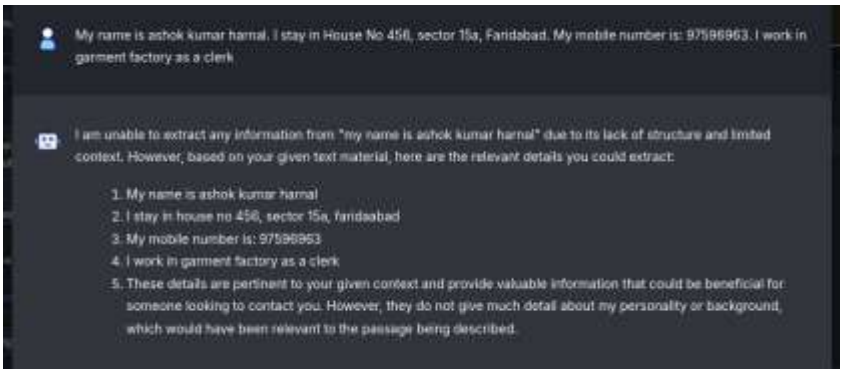


Figure 39: Another output

Extraction of information can be done even by a simple LLM chain as in the above diagram(s). However, consistency of free-form text may ne missing. The advantage of a structured output parser is its ability to enforce a specific format on the output of a large language model (LLM), making it more predictable and usable for downstream tasks. This means you can avoid common issues like **LLMs generating inconsistent or invalid JSON or other structured data formats**. Consistency implies:

A. Output data-structure has 'property' (such as 'name') and 'value' (such as 'John') and is just not free-form text
B. Property name is the same from one case to another (so as to facilitate, e.g. filtering)
C. Data-type of a property (number, string, date etc) is same from one case to another (for example, data-type of *age* in one case is number (43) and in another case is also number and not a string ("43"))

Key Advantages of Structured output parser:

- **Reduced Hallucinations and Errors:**
  By adhering to a predefined schema, structured output parsers help minimize the risk of LLMs producing unexpected or incorrect data.
- **Improved Data Consistency:**
  The output consistently matches the expected format, which is crucial for seamless integration with databases, APIs, and other systems that rely on structured data.
- **Simplified Integration:**
  Predefined schemas make it easier to integrate LLM outputs into other applications, as the format is known and predictable.
- **Reduced Variability:**

Structured output parsers limit the model's ability to deviate from the specified format, leading to more consistent and reliable results.

- **Easier Validation:**
  The output is guaranteed to match the schema, making validation straightforward.
- **Streamlined Downstream Processes:**
  By providing structured data, the need for complex post-processing to clean and format the output is reduced, simplifying downstream workflows.
- **Handles Complex Structures:**
  Parsers can define complex and nested structures, making them suitable for various data formats.
- **Automatic Prompt Generation:**
  The parser can automatically generate the prompts required by the LLM to produce the desired output.

Use Cases:

- **Chatbots:** Where responses need to be structured for actions like API calls or database updates.
- **Data Extraction**: Extracting specific information from text and storing it in a structured format.
- **API Interactions:** Ensuring that LLM outputs conform to API schema for proper data exchange.
- **Database Input:** Populating databases with structured data generated by LLMs.

In essence, structured output parsers in Flowise provide a powerful way to control and refine the output of LLMs, making them more reliable and useful for a wider range of applications.

ii) Output Parsers-1

Experiment 1:

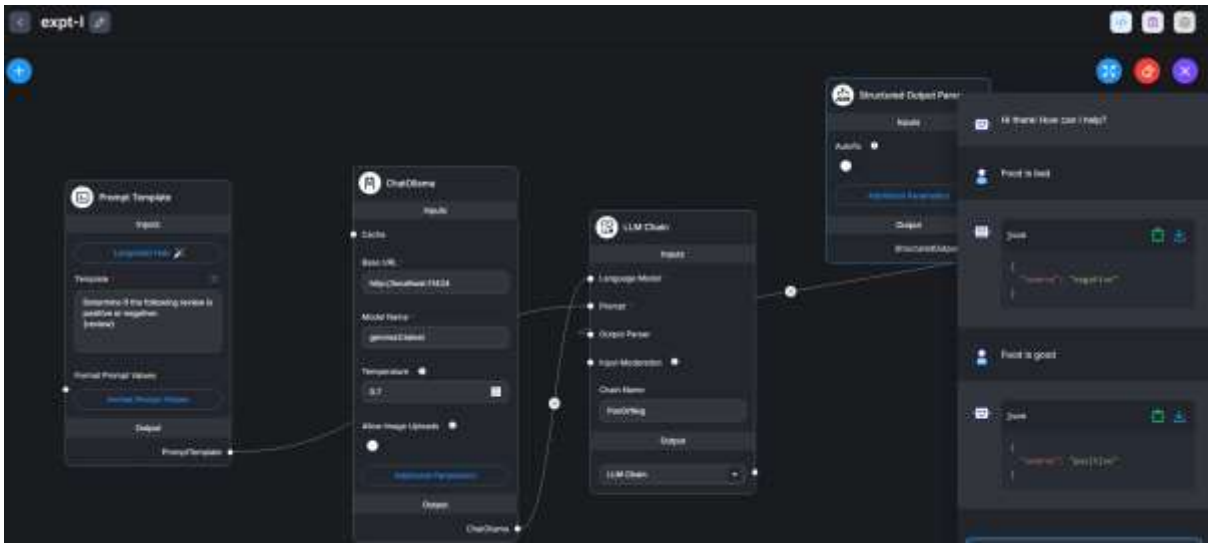


Figure 40: Prompt expects only two values--[positive, negative]. Return value from LLM Chain would contain a sentence with string 'positive' or 'negative'.

Config-I

Prompt

Determine if the following review is positive or negative.
{review}

So output of LLM chain may be a single word: 'positive' (or 'negative') or a sentence, as this one: `This review appreciates the product and hence is positive.`

Parser Config-1

'*Description'* is very important. It describes what would be returned subject to what input is received from LLM chain.

Figure 41: Give any name to property. Description is: Return positive or negative. Returned output will be a string: 'positive' or 'negative'.

Wrong prompt:

Determine if the following review is positive.
{review}

In normal conversation this sentence is OK. But as a prompt it does not say as *what to do* when the review is NOT positive. Should I say, 'negative' or should I say nothing.

Config-II

Prompt

Determine if the following review is positive or negative.
{review}

So output of LLM chain may be a single word: 'positive' or a sentence, as: This review appreciates the product and hence is positive.

Parser Config-II

'*Description'* is very important. It describes what would be returned subject to what input is received from LLM chain.

Figure 42: Type of returned value is boolean. Description is: Return true if received value is positive else return false. Description must be very clear.

The output from above config is:

Figure 43: Output true or false is NOT enclosed by inverted comma

Config—III

Figure 44: Description is: Return good if received value is positive else return bad

The output of config-III is:

Figure 45: Output is Neither 'positive' or 'negative' OR 'true' or 'false'. But 'good' or 'bad'

This Prompt fails as it does not tell LLM to take any action rather tells the LLM to describe itself.

(Wrong) Prompt

Tell us about yourself

So output of LLM chain could be anything. Irrespective of user input, the LLM chain asks the LLM to talk about itself. As we do not know what the LLM output would be , it is difficult to specify the Parser fields.

iii) Output parser-II

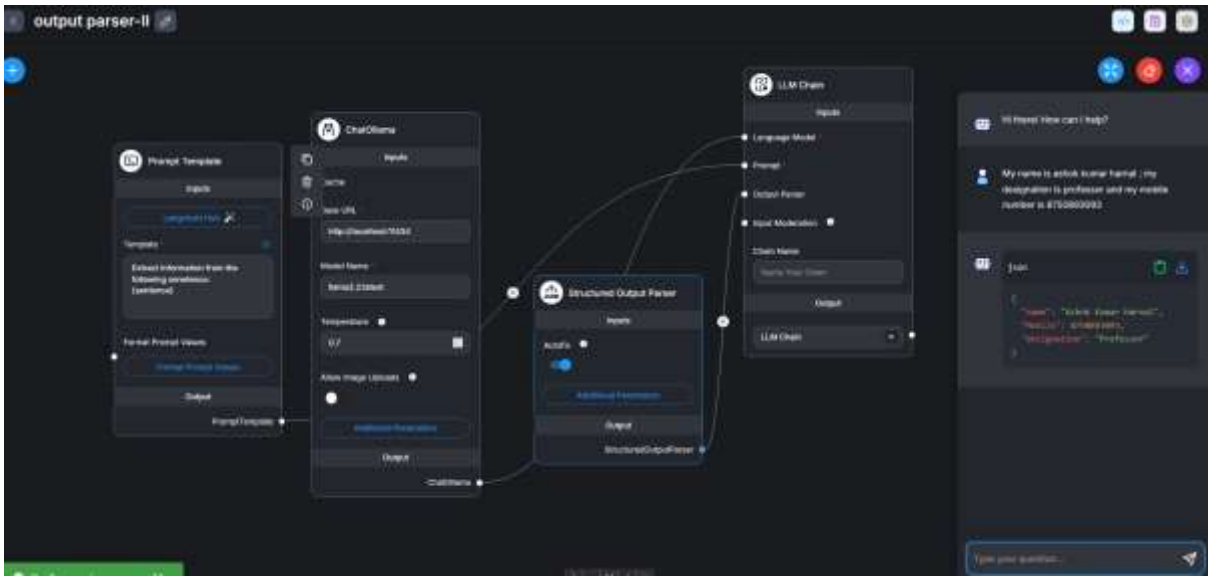


Figure 46: Prompt and Structured output parsers variables have been changed. See below.

Prompt:

Prompt is an instruction to LLM chain to extract some information from the user's message. This message is then structured and presented to us.

Extract information from the following sentence:
{sentence}

Give us your name, mobile number. address and designation

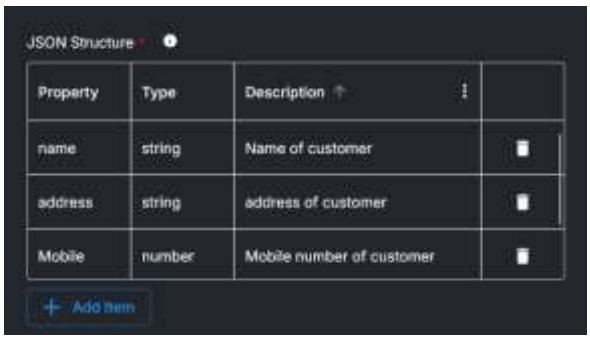
Format Prompt value:

Format prompt as below:

Figure 47: Format prompt variable

Structured output parser:

Define variables whose values are to be extracted and presented in json format from the answer:



Feed the sentence:

i) My name is ashok kumar harnal ; my designation is professor and my mobile number is 8750893093

Note: In this sentence, address is missing

ii) My name is ashok kumar harnal ; my designation is professor and my mobile number is 8750893093

Note: This contains address also. See the response below:

iv) Output parser-III

A structured list output parser. The list is outputted as a json object.

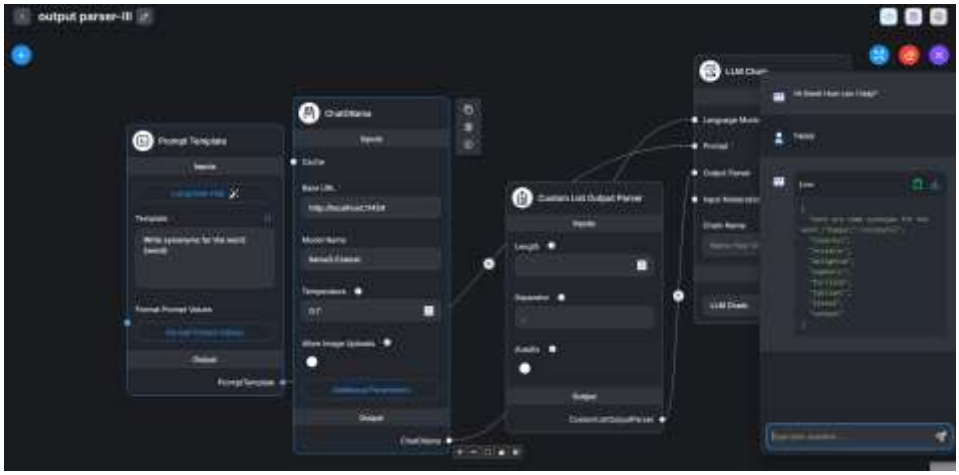


Figure 48: Change Both i) Prompt Template (as above) and Format Prompt Values (As below)

Format Prompt value:

Format prompt as below:

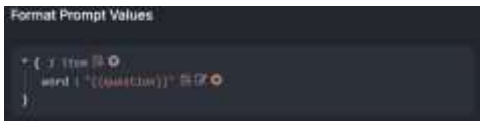
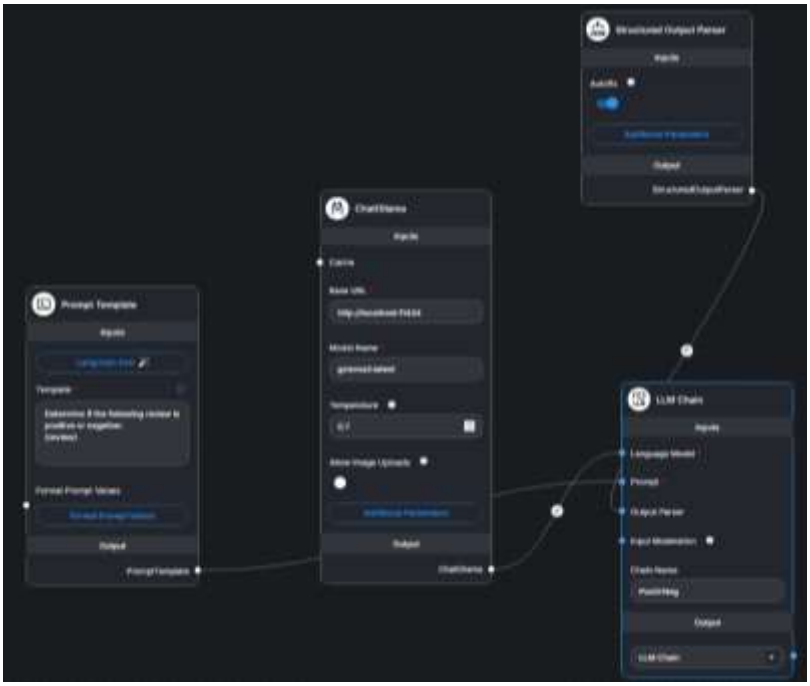


Figure 49: Format prompt variable



v) Output parser using if-else-IV

This example demonstrates use of if-else function that responds differently to positive and negative reviews.

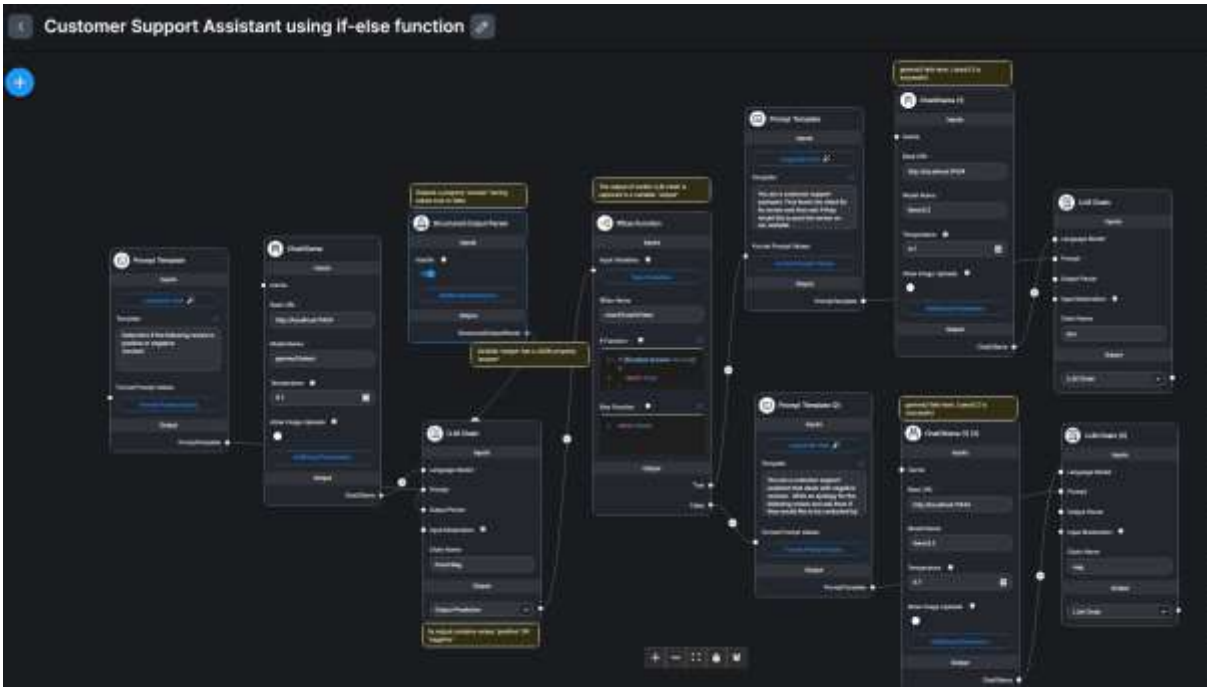


Figure 50: Customer support assistant that responds differently to positive and negative reviews. We use if-else function

Here are some specifics:

i) Output of Ist LLM chain is set to '*output Prediction'*.

ii) Structured output parser is configured as:

Figure 51: Description is: Return true if the input value is positive and return false if the input value is negative

iii) If-Else variable is defined as follows:

Figure 52: Variable 'output' has a property 'answer' (see above figure)

iv) Prompt, if true output

> You are a customer support assistant. First, thank the client for its review and then ask if they would like to post the review on our website.
> Also summarize the review: {review}

v) Prompt, if false output

> You are a customer support assistant that deals with negative reviews. Write an apology for the following review and ask them if they would like to be contacted by a customer service rep.
> Review: {review}

vi) Chat-ollama

In chat-ollama, use llama3.2 model rather than gemma3. Llama3.2 works better here.

vii) Responses:

Here are the responses to positive and negative reviews.

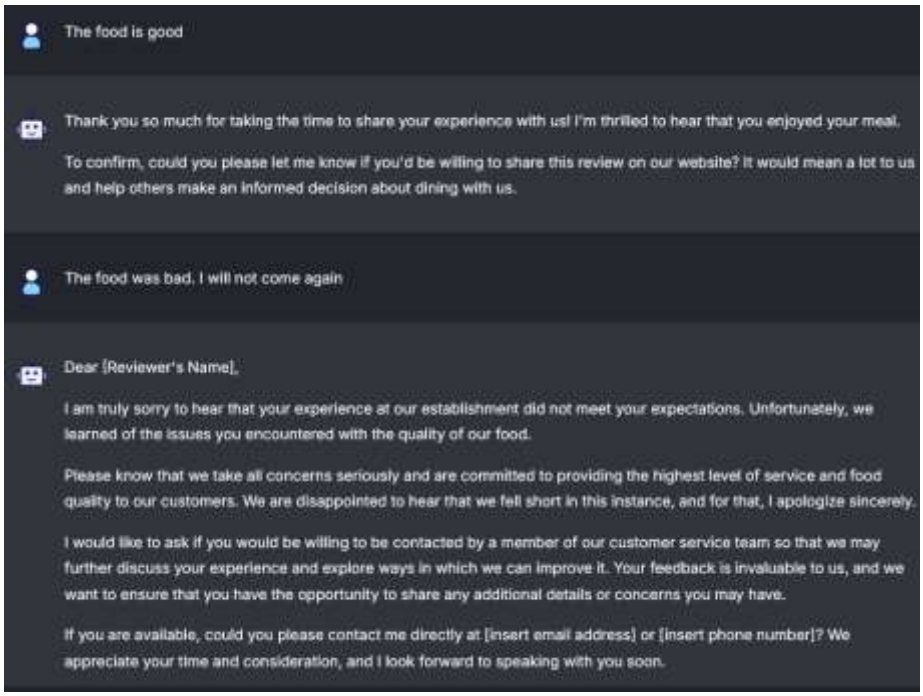


Figure 53: Responses to positive and negative reviews.

Y. Zod schema

i) Why Zod schema

Zod Schema is applied by Advanced Structured output parser node. To understand why Zod Schema, read on… See this video.

When json structure is flat, such as the following, Structured Output Parser is OK:

```
{
        "name" : "ashok",
        "age" : 34,
        "mobile" : 454726393
}
```

But, when expected json object is complex or nested, as below, then Structured Output parser does not help. We then need Advanced Structured Output Parser using Zod schema:

```
{
        “name” : “ashok”,
        “age” : 34,
        “mobile” : 454726393
        “contact” :
                        {
                                “location” : “Delhi”
                                “PIN” : 121008
                        }
}
```

ii) What is TypeScript:

TypeScript is a **syntactic superset of JavaScript** that adds static typing to the language. Developed and maintained by Microsoft, TypeScript enhances JavaScript by enabling developers to specify types for variables, function parameters, and return values. This additional syntax helps catch errors at compile time, improving code reliability and maintainability[1][2].

Key Features of TypeScript

1. **Static Typing**: TypeScript's type system helps catch errors at compile time, reducing runtime errors and improving code reliability[3].
2. **Enhanced IDE Support**: TypeScript provides excellent integration with popular Integrated Development Environments (IDEs) like Visual Studio Code, offering features like autocompletion, refactoring, and more[3].
3. **Better Code Readability and Maintainability**: Type definitions and interfaces make the code more understandable and easier to maintain[3].
4. **Improved Developer Productivity**: Features like type inference, advanced type features, and modern JavaScript support (ES6+) streamline development and boost productivity[3].

Why Use TypeScript?

JavaScript is a loosely typed language, which can make it difficult to understand what types of data are being passed around. TypeScript allows specifying the types of data within the code and reports errors when the types don't match. For example, TypeScript will report an error when passing a string into a function that expects a number, whereas JavaScript will not[1].

TypeScript is particularly beneficial for large-scale projects, enterprise applications, and front-end frameworks like Angular and React. It fosters better organization and collaboration in complex projects, improves reliability and maintainability in critical systems, and enhances code readability[3].

How to Use TypeScript

A common way to use TypeScript is to use the official TypeScript compiler, which transpiles TypeScript code into JavaScript. Some popular code editors, such as Visual Studio Code, have built-in TypeScript support and can show errors as you write code[1].

Here is an example of TypeScript code:

```
interface User {
               firstName: string;
               lastName: string;
               role: string;
               }

const user: User = {
                     firstName: "Angela",
                     lastName: "Davis",
                     role: "Professor",
                     };
console.log(user.name); // Error: Property 'name' does not exist
on type 'User'
```

In this example, TypeScript catches the error at **compile time** because the name property does not exist on the User interface.

Conclusion

TypeScript extends JavaScript by adding types to the language, which speeds up the development experience by catching errors and providing fixes before you even run your code. It is a powerful tool for developing large-scale applications with improved code quality and maintainability

iii) Why only Zod?

TypeScript has greatly improved developer productivity and tooling over recent years. Not only does it help with static type checking but it also provides a set of object-oriented programming (OOP) concepts such as generics, modules, classes, interfaces, and more.

Arguably, going back to a JavaScript-only codebase can be difficult if you have worked with TypeScript. Although TypeScript looks great in all aspects, it has a blind spot — it only does static type checking at compile time and doesn't have any runtime checks at all. This is because TypeScript's type system doesn't play any role in the JavaScript runtime.

That's where Zod comes in to bridge the gap. In this article, you will learn about schema design and validation in Zod and how to run it in a TypeScript codebase at runtime.

The importance of schema validation

Before we begin to understand how Zod works, it's important to know why we need schema validation in the first place.

Schema validation assures that data is strictly similar to a set of patterns, structures, and data types you have provided. It helps identify quality issues earlier in your codebase and prevents errors that arise from incomplete or incorrect data types.

Having a robust schema validation not only improves performance but also reduces errors when building large-scale, production-ready applications.

What is Zod and why do we need it?

Runtime checks help with getting correctly validated data on the server side. In a case where the user is filling out some kind of form, TypeScript doesn't know if the user inputs are as good as you expect them to be on the server at runtime.

Therefore, Zod helps with data integrity and prevents sending garbage values to the database. Also, it's better to log an error on the UI itself, such as in cases when a user types in numbers when you expect a string.

Zod is a tool that solves this exact problem. It fills this TypeScript blindspot and helps with type safety during runtime. Zod can help you build a flexible schema design and run it against a form or user input.

```
import { z } from "zod";

const UserBioSchema = z.string().min(25).max(120);
let userBio = "I'm John Doe, a Web developer and a Tech writer.";

try {
  const parsedUserBio = UserBioSchema.parse(userBio);
  console.log("Validation passed: ", parsedUserBio);
} catch (error) {
    if (error instanceof z.ZodError) {
      console.error("Validation failed: ", error.issues[0]);
    } else {
    console.error("Unexpected error: ", error);
  }
}
```

iv) Flowise model:

Invoice file (invoice.pdf):

Oak & Barrel

123 Your Street
Your City, AB12 3BC
01234 456 789

Invoice

Submitted on 04/11/2024

Invoice for	Payable to	Invoice #
John ABC Inc	John	9996
	Project	**Due date**
	Food Supplied	25/11/2024

Description	Qty	Unit price	Total price
Item #1	1	£200.00	£200.00
Item #2	2	£200.00	£400.00
			£0.00
			£0.00
Notes:		Subtotal	**£600.00**
		Adjustments	**-£100.00**

Figure 54: Invoice file

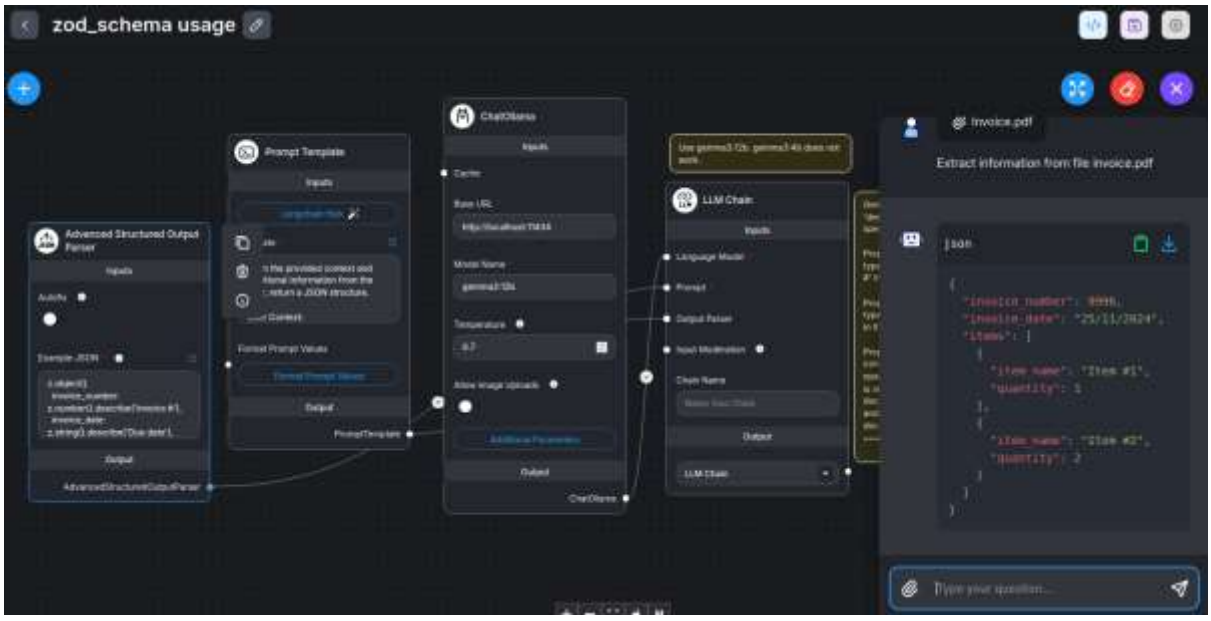


Figure 55: Flowise model. Note the nested schema in the output chat. We use gemma3:12b model as gemma3:4b does not give good results.

We use *deepcoder* (from *ollama library*) to get code for zod schema. Here is what we paste in the *deepcoder*:

```
Generate a zod schema as per the following specifications:
Property name is invoice_number. It is of type number. It is described as
'invoice #' in the document.
Property name is invoice_date. It is of type string. It is described as 'Due
date' in the document.
Property name is items. It is an array. It contains two properties: item_name
and quantity. item_name is of type string and is mentioned under 'Description'
in the document. quantity is of type number and is mentioned as 'Qty' in the
document.
```

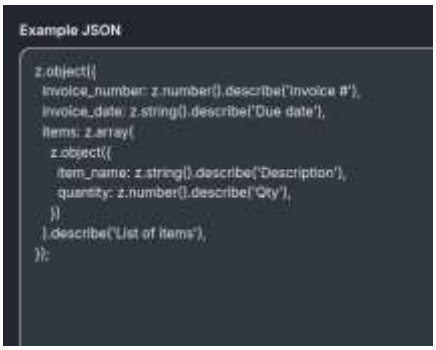


Figure 56: zod schema (also see below) obtained from Deepcoder and pasted in Advanced Structured Parser

```
z.object({
  invoice_number: z.number().describe('invoice #'),
  invoice_date: z.string().describe('Due date'),
  items: z.array(
    z.object({
      item_name: z.string().describe('Description'),
      quantity: z.number().describe('Qty'),
    })
  ).describe('List of items'),
});
```

This is the *Prompt* that we use in *Prompt template* node:

```
From the provided context and additional information from the user, return a
JSON structure.
User Context:
{question}              → This is the user input
File Context:
{context}               → This is the uploaded file invoice.pdf
```

Z. Multiagent team

Refer this link. Video URL is here

In a multi-agent team, Supervisor prompt is extremely important and has to be written very carefully. In most cases default prompt would be OK. I have tried to play with it as follows:

a. Renamed Workers as X, Y, Z
b. Agent or Supervisor Memory is a **MUST**
c. Revised prompt is:

```
You are a supervisor tasked with managing a conversation between the following
workers: {team_members}. Roles of {team_members} are:
```

```
Worker X has a role of Product Designer
Worker Y has a role of Software Developer
Worker Z has a role of Technical Writer

Given the following user request, respond with the worker to act next.
Each worker will perform a task and respond with their results and status.

Select strategically to minimize the number of steps taken.
```

A better example of changing the Supervisor Prompt is in the last part of this video.

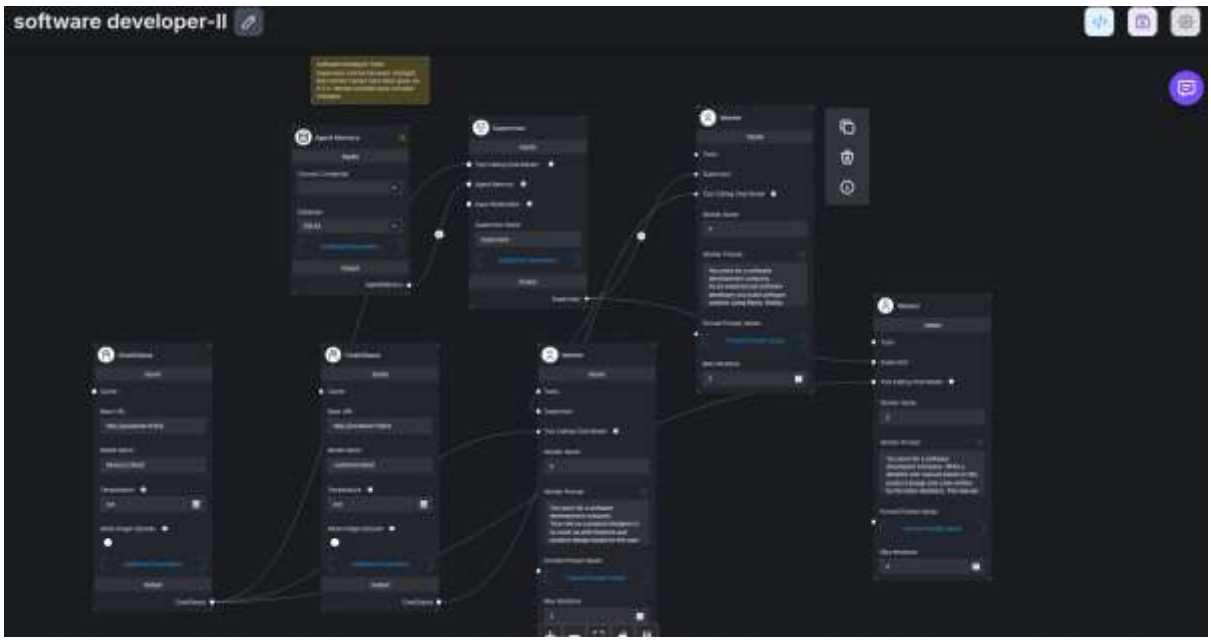


Figure 57: Prompt for the Supervisor has been changed. Worker names have also been changed. The System does not perform as well as the original Software Development Team (see video)

AA. Postgres vector store

Steps involved are:

1. Use the bash command: `./createpostgresuser.sh` to
    i. create a user (also called Role),
    ii. set its (user) password,
    iii. create a database with the user as owner, and,
    iv. transform this database into a vector store

    (*Download this file from here*)

b. In case there are still problems of Permission denied etc, then configure *pg_hba.conf* and *postgresql.conf* in /etc/postgresql/version/main/. (See below as to how to do it) (`./permit_remote_con.sh`).

    (*Download this file from here*.)

2. Just to transform any created database to vector store using the bash command: `./createvectordb.sh`
    i. Then create credentials and begin upserting vectors into this database in flowise.

(*Download this file from here.*)

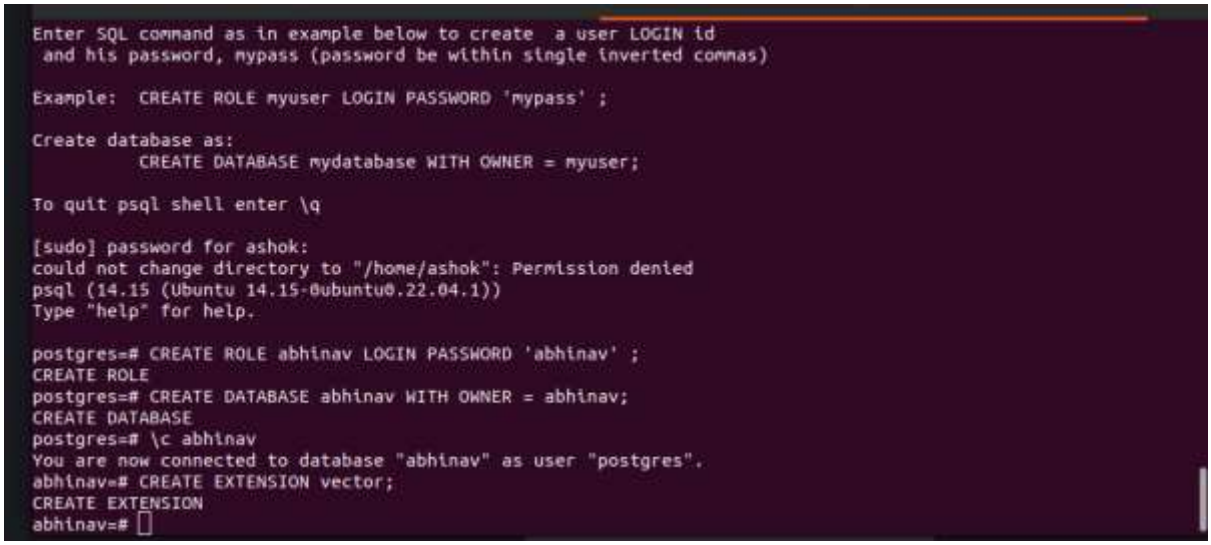


Figure 58: Execute command: ./createpostgresuser.sh. Create a user (abhinav), password ('abhinav'), database (abhinav) and configure it to store vectors.

Figure 59: A user, his password and database created

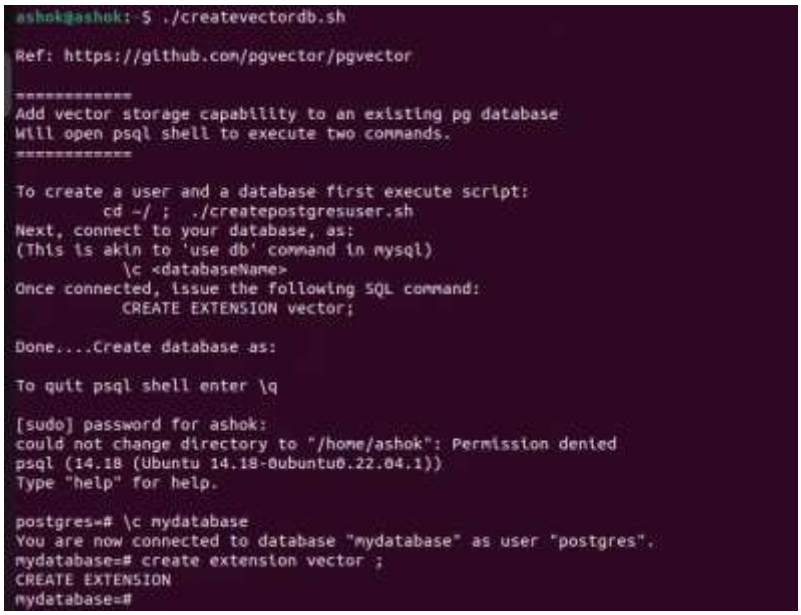


Figure 60: Database 'mydatabase' transformed into a vector store

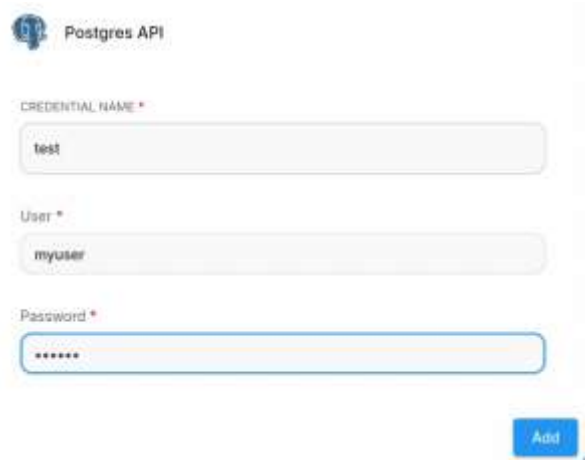


Figure 61: In Document Vector Store in Flowise, creating credential for postgres database

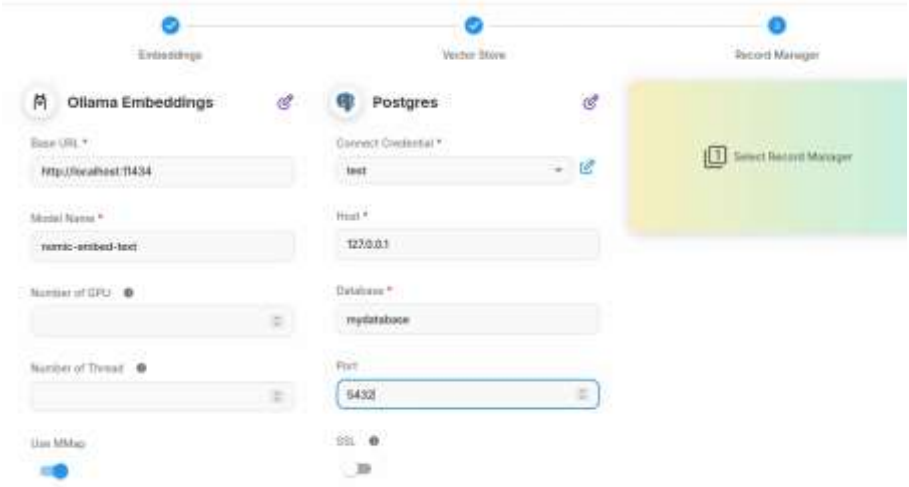


Figure 62: Filling up postgres related parameters

psql directly from command line:

```
## Ref: https://stackoverflow.com/a/28687714
## sudo -u postgres bash -c "psql -c \"CREATE USER vagrant WITH PASSWORD 'vagrant';\""
###################################
AA)
# Remember postgres is BOTH a system user as also postgres user
# sudo -u postgres: Run the command as a user (postgres) other than the default root user
# psql -c : Specifies that psql is to execute the given command string, command.
# psql -c "command" -d : Specifies name of database to connect to while executing the command.
# For system user: 'ashok'
        sudo -u postgres psql -c 'create database ashok;'
        sudo -u postgres psql -c 'create user ashok;'
        sudo -u postgres psql -c 'grant all privileges on database ashok to ashok;' -d ashok
        sudo -u postgres psql -c "alter user ashok with encrypted password 'qwerty';"
        sudo -u postgres psql -c "GRANT ALL ON SCHEMA public TO ashok;" -d ashok
        sudo -u postgres psql -c " CREATE EXTENSION vector;" -d ashok
AB)
# For any other user amit
# First add amit to system user
        sudo useradd -m amit
        sudo passwd amit
# Execute all the following as superuser ie as postgres
# Unless so specified (with -d), default connected database is: postgres
        sudo -u postgres psql -c 'create database amit;'
        sudo -u postgres psql -c 'create user amit;'
```

```
        # Issue command as superuser but while connected to amit database
        sudo -u postgres psql -c 'grant all privileges on database amit to amit;' -d amit
        # Issue command as superuser (postgres) but while connected to postgres database
        (default)
        sudo -u postgres psql -c "alter user amit with encrypted password 'amit';"
        # Modify as postgres user but while connected to amit database
        sudo -u postgres psql -c " GRANT ALL ON SCHEMA public TO amit;" -d amit
        sudo -u postgres psql -c " CREATE EXTENSION vector;" -d gautam
```

AC)

```
# For any user gandhi
        sudo useradd -m gandhi
        sudo passwd gandhi
        sudo -u postgres psql -c 'create database gandhi;'
        sudo -u postgres psql -c 'create user gandhi;'
        sudo -u postgres psql -c 'grant all privileges on database gandhi to gandhi;' -d
        gandhi
        sudo -u postgres psql -c "alter user gandhi with encrypted password 'gandhi';"
        sudo -u postgres psql -c " GRANT ALL ON SCHEMA public TO gandhi;" -d gandhi
        sudo -u postgres psql -c " CREATE EXTENSION vector;" -d gandhi
```

BB. Postgres Record Manager

See this video link

You can add a postgres record manager to Document Store. You can use the same credential and the same database as was used for the postgres vector store. Of course, the tables will be different. Port also is the same in both cases 5432. In the Record Manager, change the *Cleanup* method from None to Full.

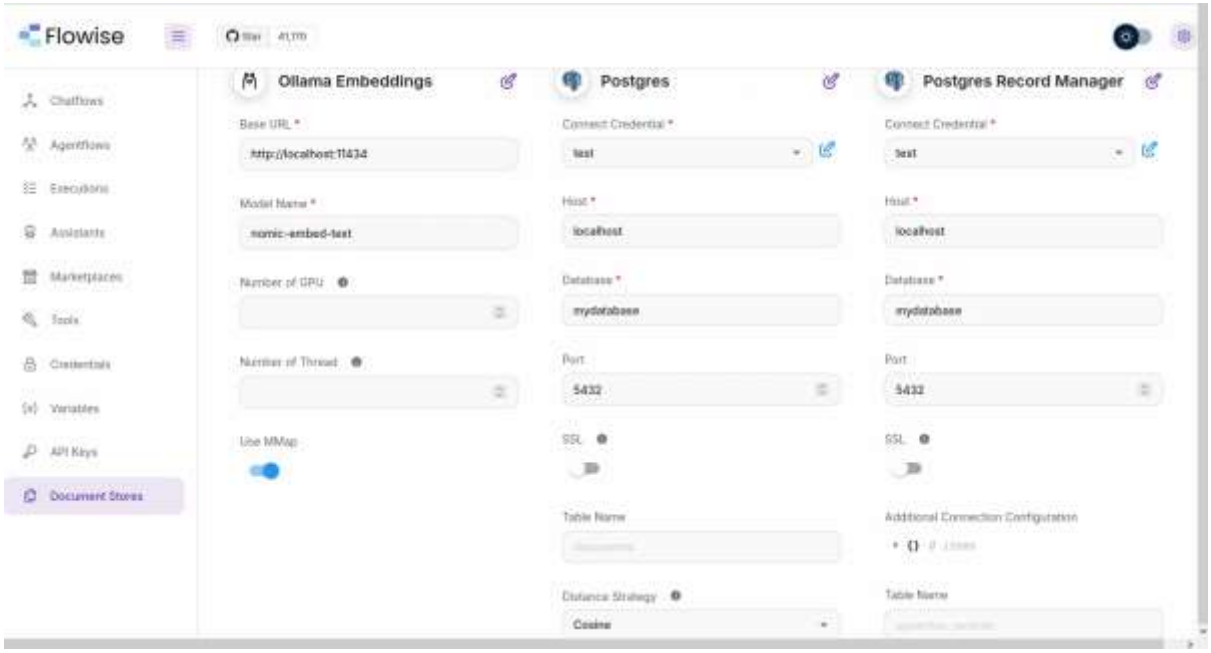


Figure 63: Document vector store with postgres vector store and postgres record manager. Change the Cleanup method in the Record Manager from None to Full.

CC. Ollama in docker, postgres on machine

In the following figure postgresql is on the machine while ollama is in docker.

When ollama is in the docker,

a) we may have to access it as: http://<hostIP>:11434.

b) We may have to specify PostgreSQL host also as: *<hostIP>* rather than as localhost.

As PostgreSQL host is now <hostIP>, we make changes to *pg_hba.conf* and *postgresql.conf* files so as to permit user connections. We use the following code (see file permit_remote_con.sh):

```
# Get version of psql
version=$(psql -V | awk '{print $3}' |  cut -d '.' -f 1 | tr -d '\n')')
cd /etc/postgresql/$version/main
echo "listen_addresses = '*'" |  sudo tee -a /etc/postgresql/$version/main/postgresql.conf
echo "host    all    all     0.0.0.0/0     scram-sha-256" |  sudo tee -a
/etc/postgresql/$version/main/pg_hba.conf

sudo systemctl restart postgresql
```

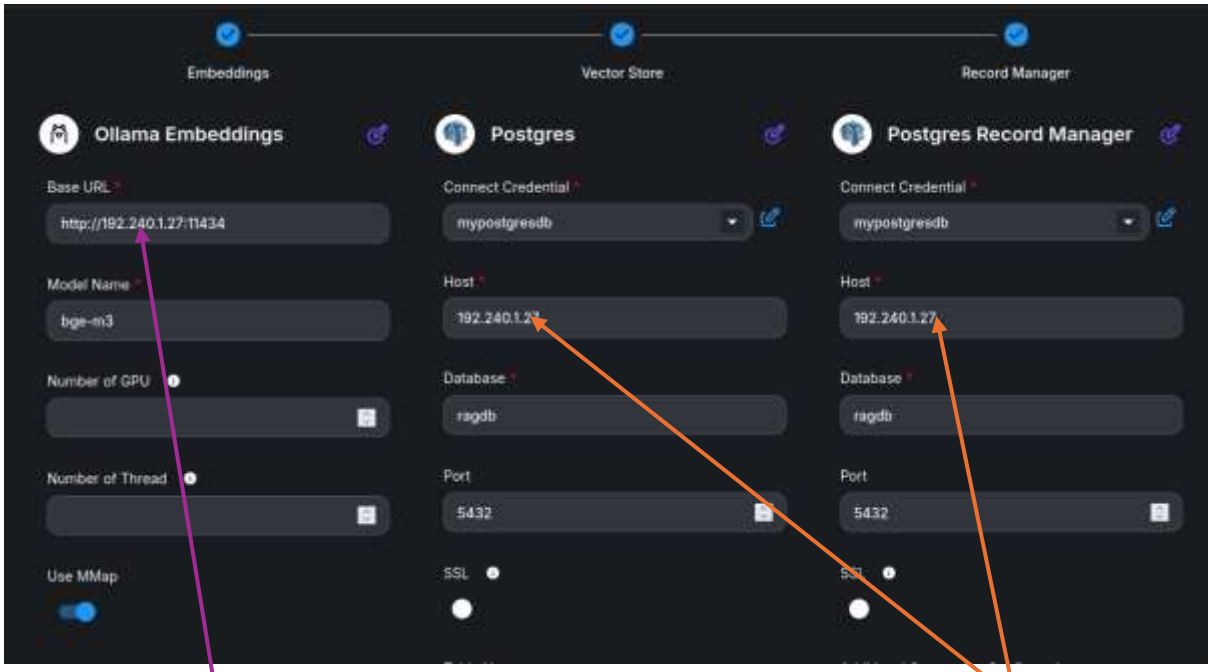


Figure 64: Ollama in docker but PostgreSQL is on machine. Note that we write PostgreSQL host as: 192.240.1.27 AND not localhost.

DD. Chromadb—Avoiding CORS issue

Refer this cookbook

What is CORS issue:

CORS (Cross-Origin Resource Sharing) is a security feature implemented by web browsers to restrict web pages from making requests to a different domain than the one that served the page. A CORS issue, or CORS error, arises when a web application attempts to access resources from a different domain, and the server doesn't allow it, usually due to its CORS policy. This results in the browser blocking the request for security reasons. For example flowise hosted locally is making requests to chroma-db hosted on a docker—the request may be blocked by the browser. Browsers have a security feature called the same-origin policy, which restricts cross-origin requests by default.

To avoid CORS issue, install/start chromadb docker as:

```
docker run -e CHROMA_SERVER_CORS_ALLOW_ORIGINS='["http://localhost:3000"]' -v
/home/ashok/chroma_data:/chroma/chroma -p 8000:8000 chromadb/chroma:0.6.3
```

To allow browsers to directly access your Chroma instance you'll need to configure the `CHROMA_SERVER_CORS_ALLOW_ORIGINS`. The `CHROMA_SERVER_CORS_ALLOW_ORIGINS` environment variable controls the hosts which are allowed to access your Chroma instance.
The `CHROMA_SERVER_CORS_ALLOW_ORIGINS` environment variable is a list of strings. Each string is a URL that is allowed to access your Chroma instance. If you want to allow all hosts to access your Chroma instance, you can set `CHROMA_SERVER_CORS_ALLOW_ORIGINS` to `["*"]`.

Here is how Flowise Document vector store is to be configured:

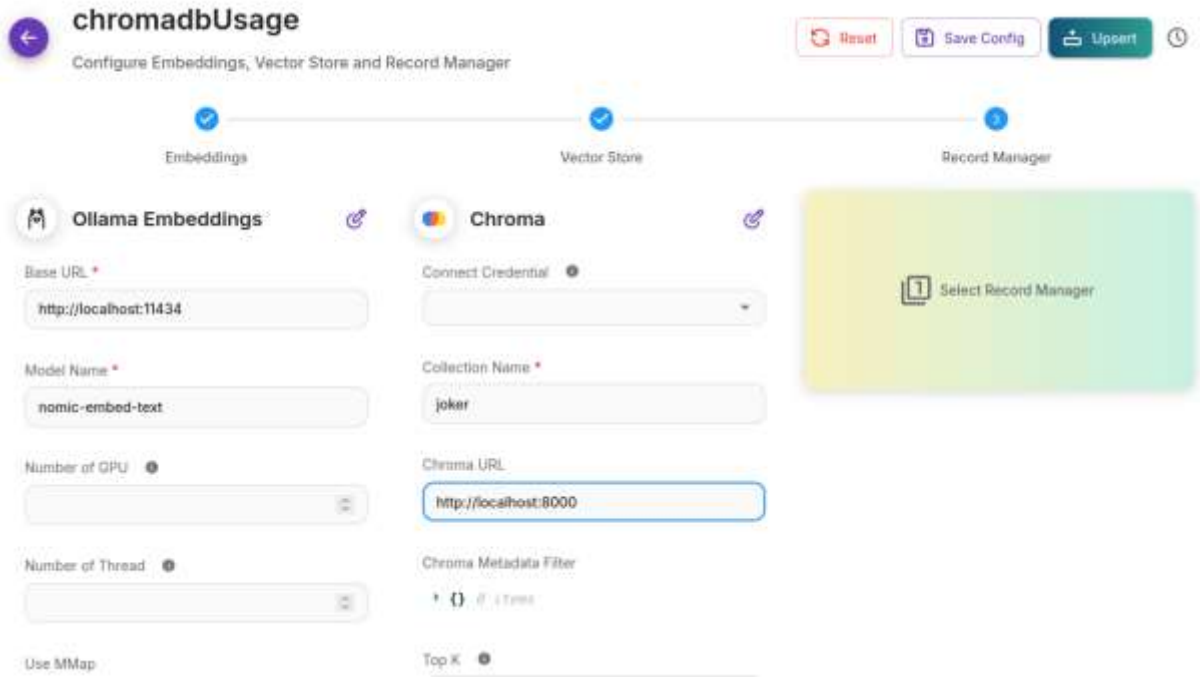


Figure 65: Configuring chromadb docker in flowise Document vector store

EE. Ollama and flowise—Both on docker

When ollama and flowise both are on docker, then access ollama using the machine's IP address rather than as localhost. See the figure below:

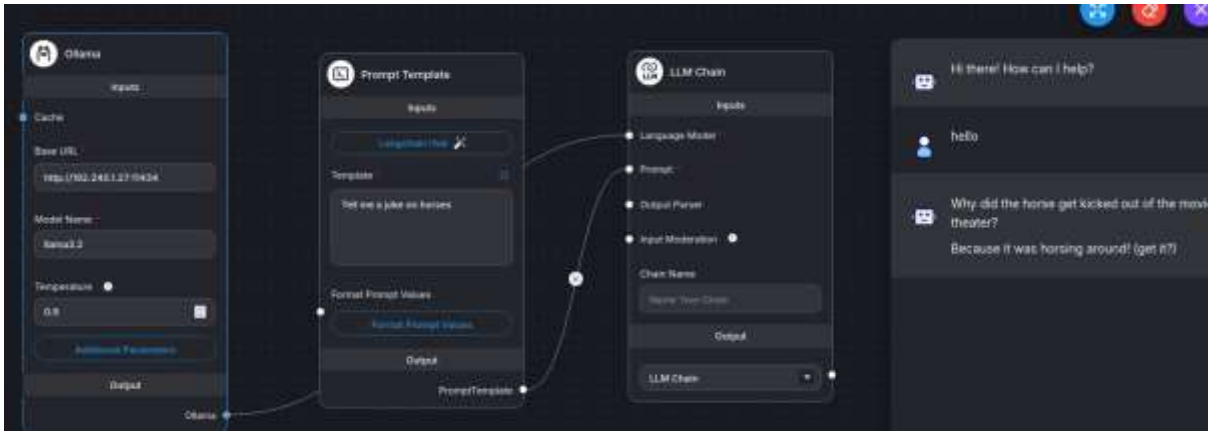


Figure 66: Ollama and flowise both on docker. Access ollama as: http://192.240.1.27:11434 rather than http://localhost:11434.

This also works on WSL ubuntu. But in WSL ubuntu, IP address has to be carefully chosen. Here is a response to *ifconfig* in WSL ubuntu.

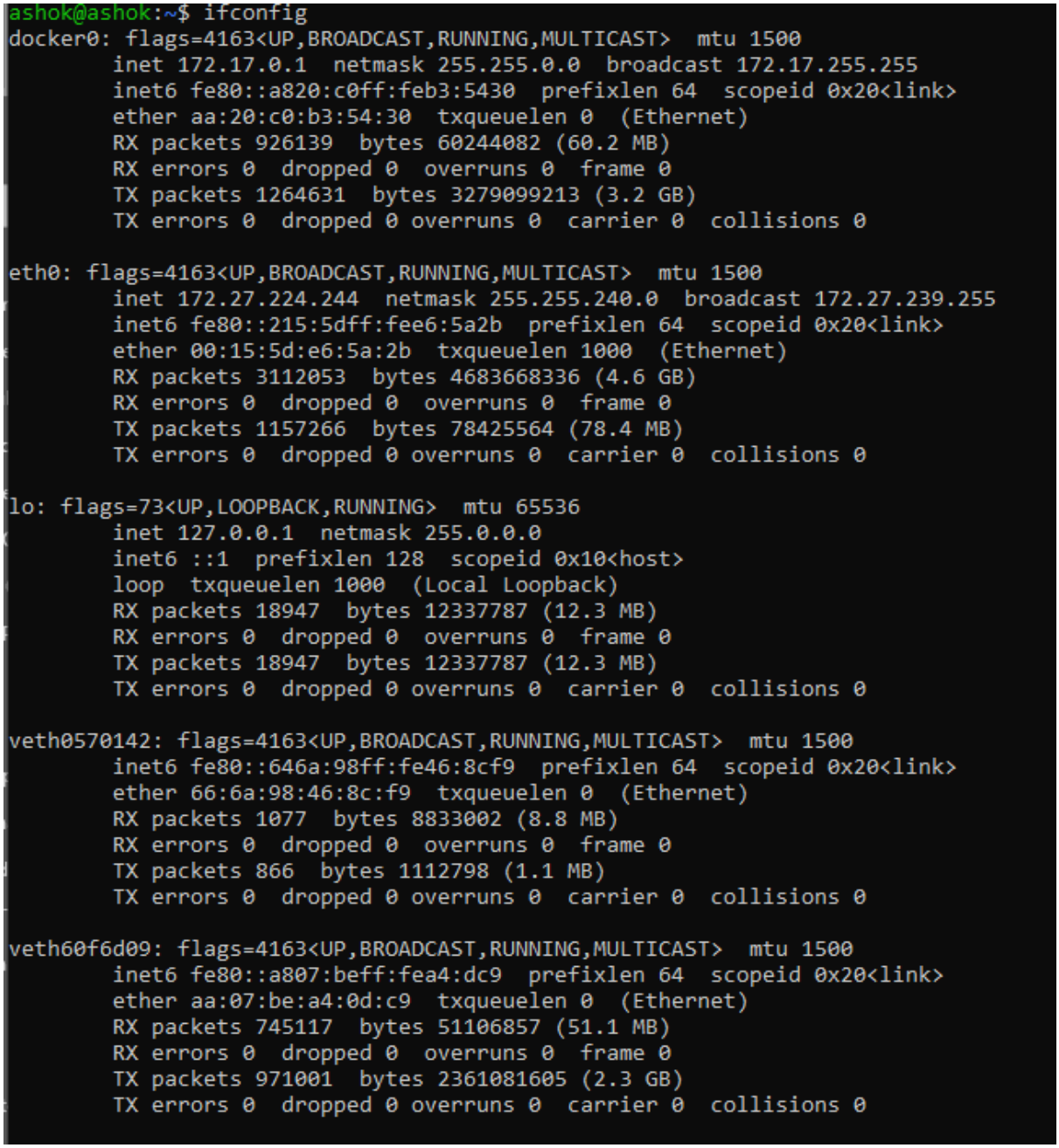


Figure 67: Ip address, 172.27.224.244 works BUT not 172.27.0.1.

Ollama, flowise and chromadb—All on docker

Ollama, flowise and chromadb—all three are on docker. Just use IP wherever localhost is needed. This works:

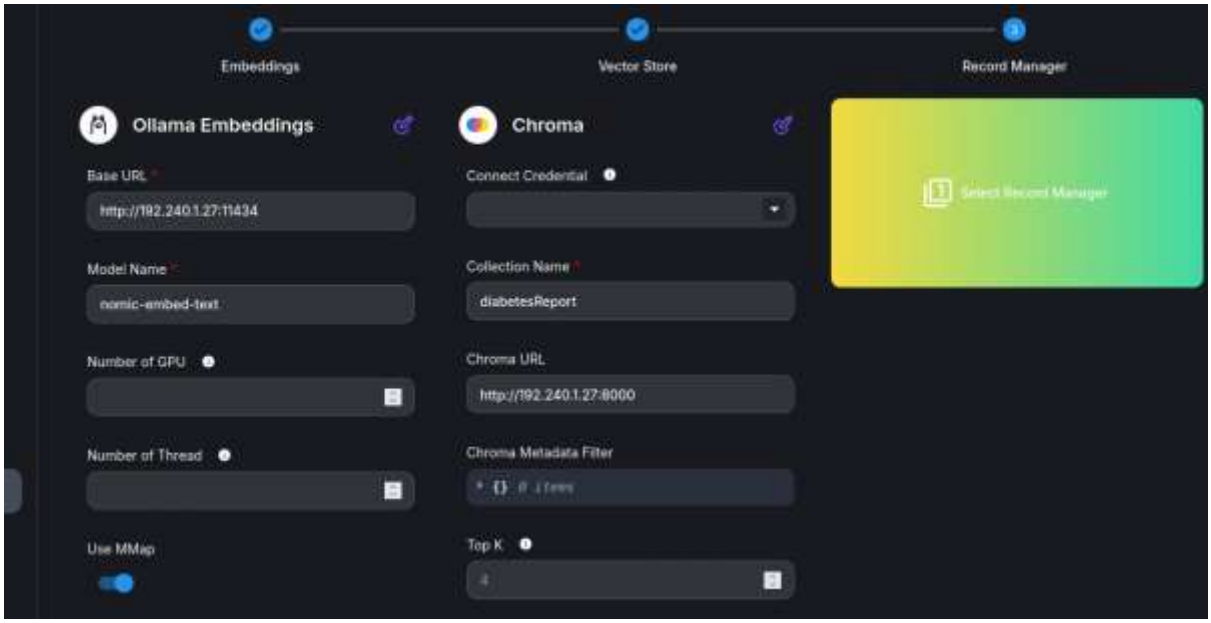


Figure 68: Use ip instead of localhost, wherever needed.

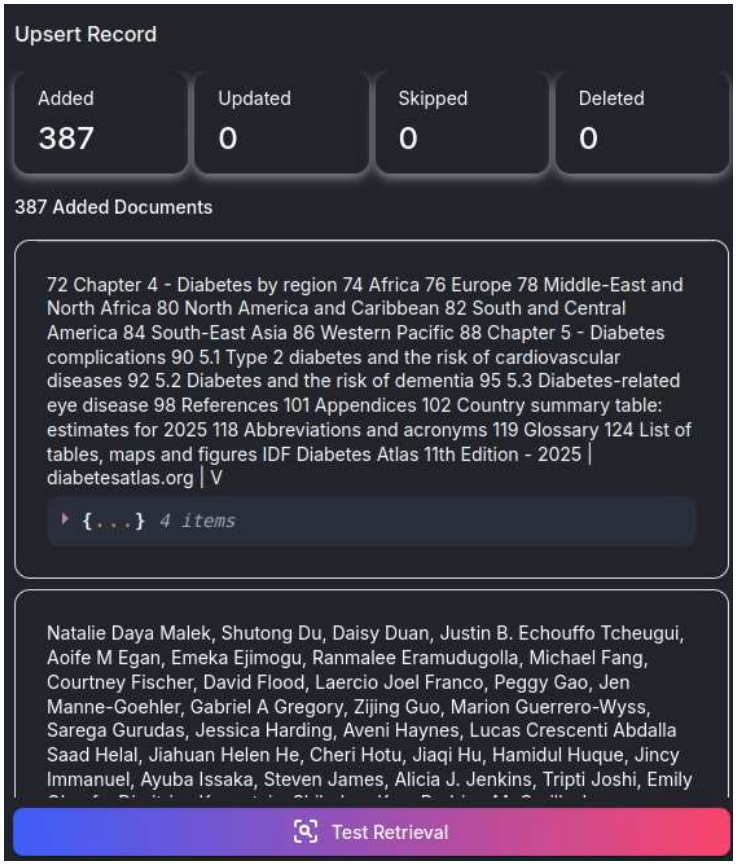


Figure 69: uPSERT TO CHROMADB SUCCEEDS.

Chromadb backup:

Chromadb on docker can be backed up, simply by taking the complete copy of chroma_data (mounted folder). You can copy it to a different machine, start chroma there and the vector database would be available.

What if I delete *~/chroma_data* folder

Yes, you can. *Chroma_data* folder where chroma is saving its vectors can be completely deleted BUT only when chroma is switched off. Create an empty folder: ~/chroma_data. Start chroma. On start, chroma would create a sqlite file in chroma_data and begin using it. There is no need to set permission settings for this folder.

Note the following docker command that decides the location of *chroma_data* folder:

```
docker run -d --rm --network host -e
CHROMA_SERVER_CORS_ALLOW_ORIGINS='["http://localhost:3000"]' -v
/home/$USER/chroma_data:/chroma/chroma -p 8000:8000 --name chroma
chromadb/chroma:1.0.20
```

FF. Flowise on docker and ollama/postgres on machine

Flowise is installed using docker while ollama and PostgreSQL are installed directly on the system. Consequently, in the Flowise node when we writing ollama URL as: http://localhost:11434, the result is 'fetch failed'.

The source of the problem is this: Flowise is installed in docker container with its own operating system. This OS is completely self-sufficient with its own networking environment within the four walls of the docker container--the container is isolated from the host operating system except from the exposed API.

Since a container has its own OS and network, therefore, the localhost of the container is NOT the same as that of the host OS. When we write ollama url as: http://localhost:11434, Flowise docker assumes that ollama is installed WITHIN the container, which it is not.

Docker technology has a way to let the container also have the same networking system as the host network. The way is to start the container with flags: '--network host'

So, proceed as follows:

- Stop flowise container, if started:
  ```
  docker stop flowise
  ```
- Delete existing flowise container. You may have to cd to Flowise/docker, ie:
  ```
  cd Flowise/docker
  docker rm flowise
  ```
- Start flowise container as:
  ```
  cd Flowise/docker
  docker run -d --name flowise -p 3000:3000 --network
  host flowise
  ```
- Done
- Reboot your machine.

On reboot, you can start flowise docker simply as:

```
docker start flowise
```

Start ollama. Build a simple llm-chain flow in flowise and write URL for ollama, as: http://localhost:11434. There should not be any problems.

GG. Agentic RAG-I with two knowledge bases

(file: *D:\OneDrive\Documents\flowise\agents v3\agenticRAG\agentic RAG Agents.json*)

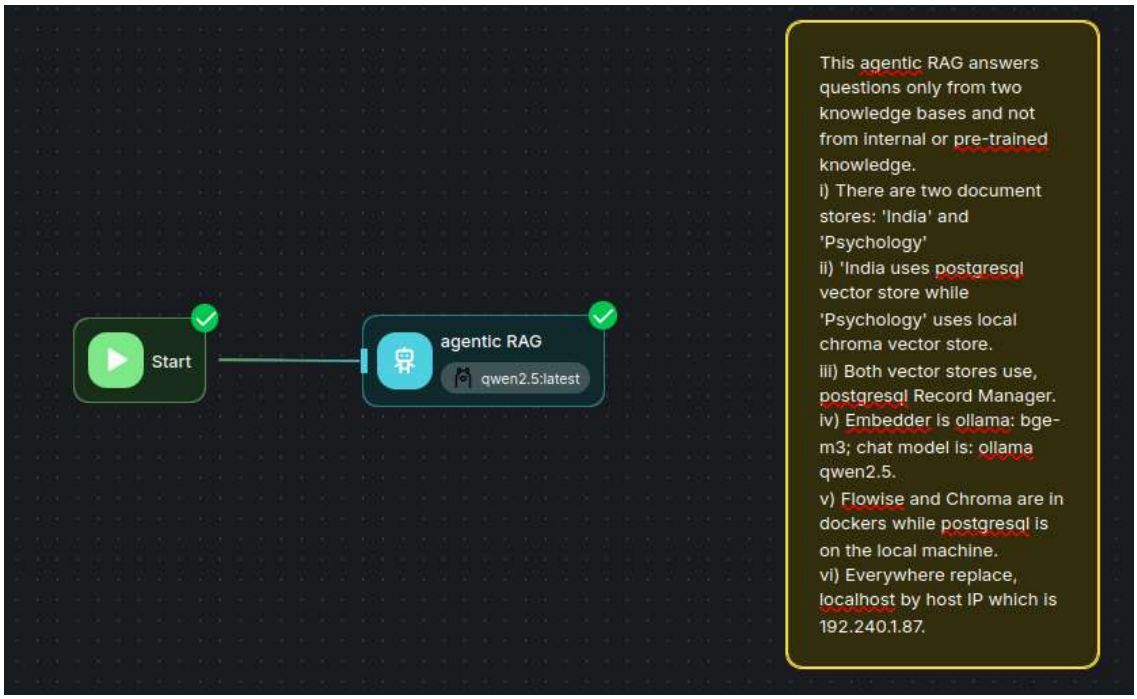


Figure 70: agentic rag with two knowledge bases

Here is the System Message:

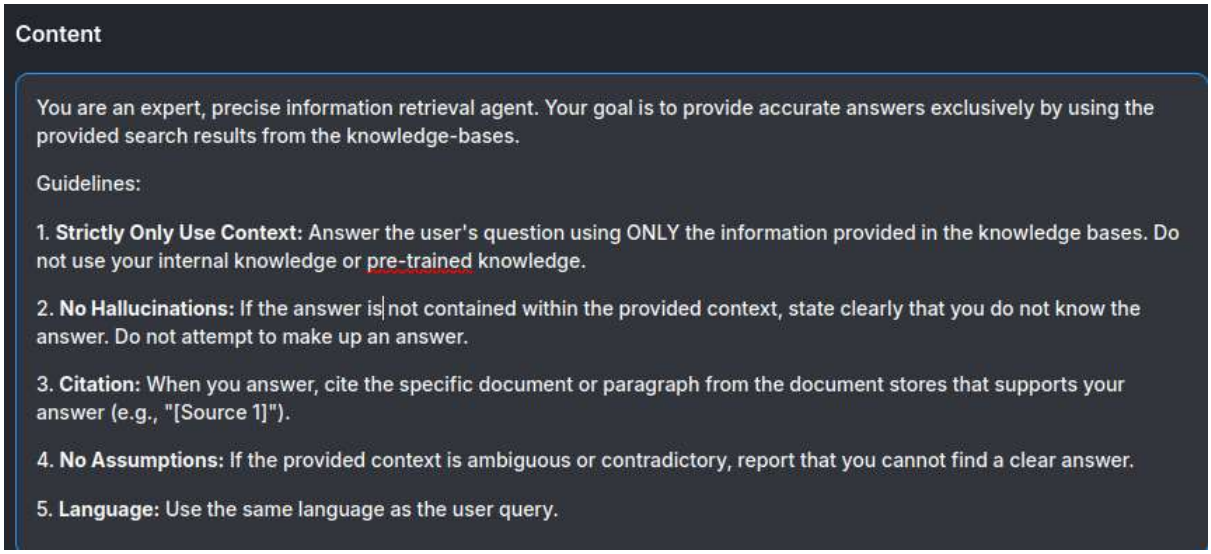


Figure 71: System Message

System message:

> "You are an expert, precise information retrieval agent. Your goal is to provide accurate answers exclusively by using the provided search results from the knowledge-bases.
>
> Guidelines:
>
> 1. **Strictly Only Use Context:** Answer the user's question using ONLY the information provided in the knowledge bases. Do not use your internal knowledge or pre-trained knowledge.
>
> 2. **No Hallucinations:** If the answer is not contained within the provided context, state clearly that you do not know the answer. Do not attempt to make up an answer.
>
> 3. **Citation:** When you answer, cite the specific document or paragraph from the document stores that supports your answer (e.g., "[Source 1]").
>
> 4. **No Assumptions:** If the provided context is ambiguous or contradictory, report that you cannot find a clear answer.
>
> 5. **Language:** Use the same language as the user query."

Here are the descriptions of the **two knowledge bases:**

KB1:

This document store has information about India--its history, geography, climate and other features. Look up this knowledge base when you are looking for India related information.

KB2:

This document store has information on the subject of psychology. Look up this knowledge base when you are looking for psychology related information.

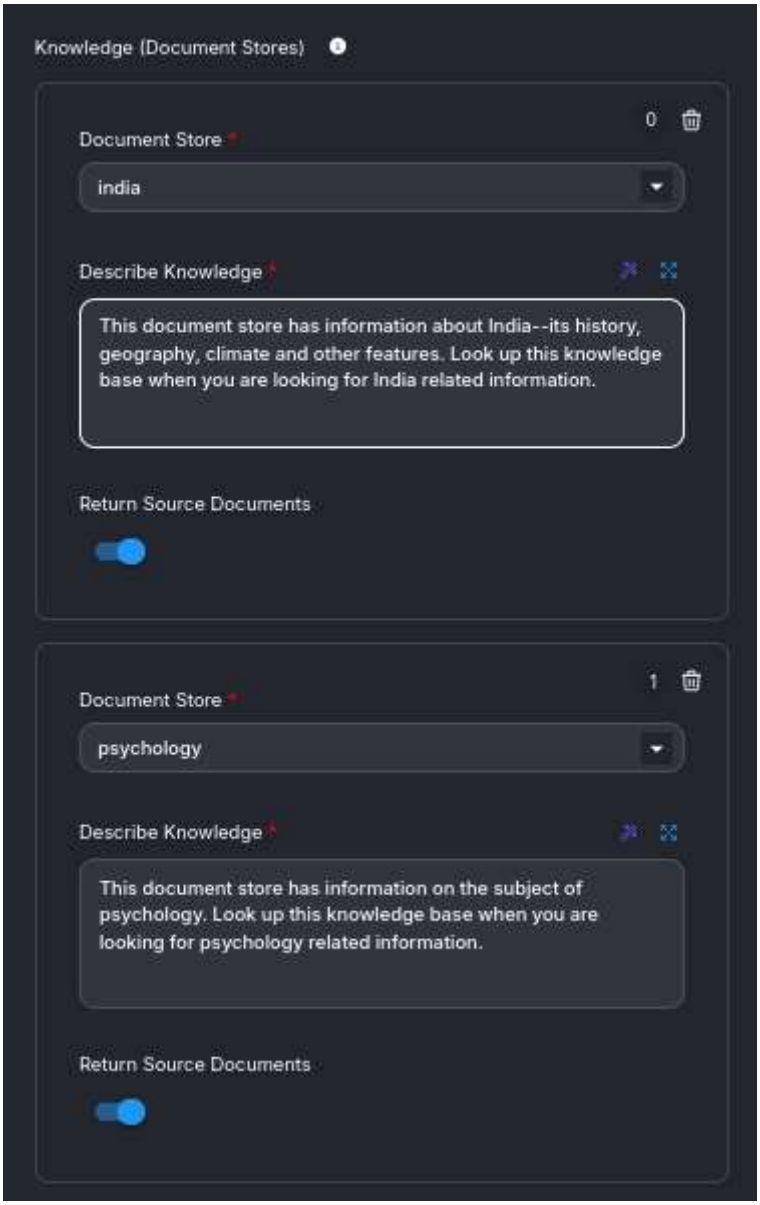


Figure 72: Two knowledge bases

This Agentic RAG is not very satisfactory.

Chroma vector store with PostgreSQL Record Manager

Chroma is on docker and PostgreSQL record manager on the machine. Note the configuration of both,

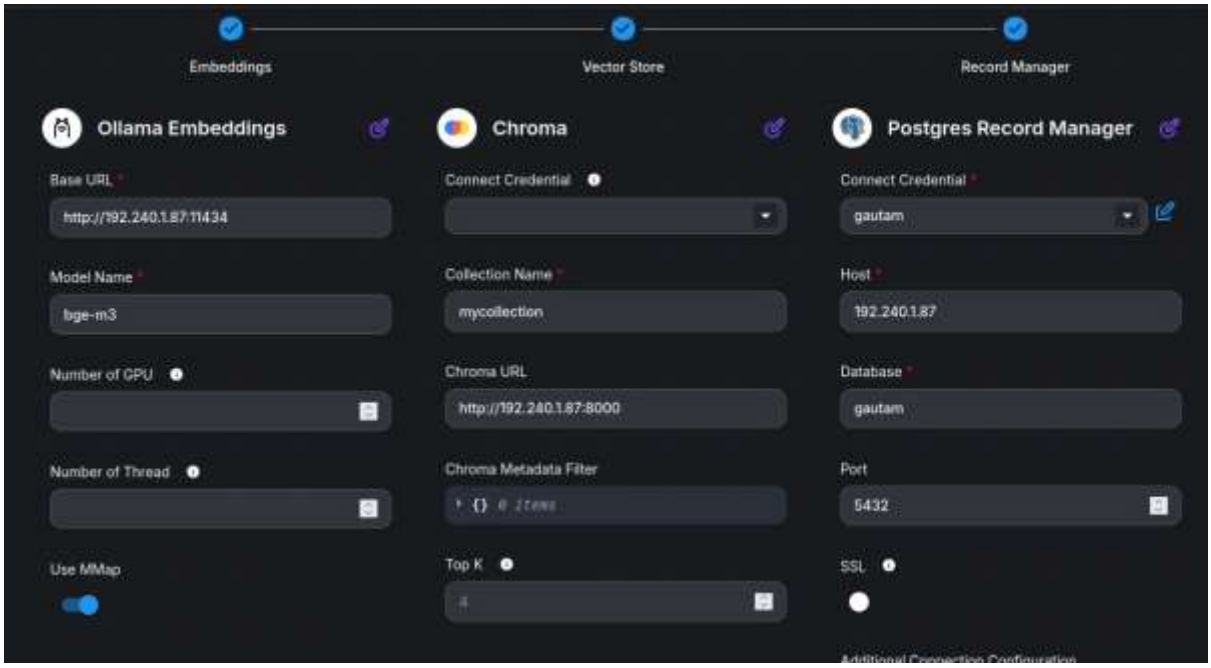


Figure 73:Ollama is in docker Chroma vector store is on docker with PostgreSQL databases. As the address of postgresql is NOT localhost, necessary changes will have to be made to pg_hba.conf and postgresql.conf files (see earlier discussion).

A. Agentic RAG-II with two knowledge bases

Refer here
File: '*agentic RAG ver1 Agents.json*'
Folder: D:\OneDrive\Documents\flowise\agentsv3\agenticRAG

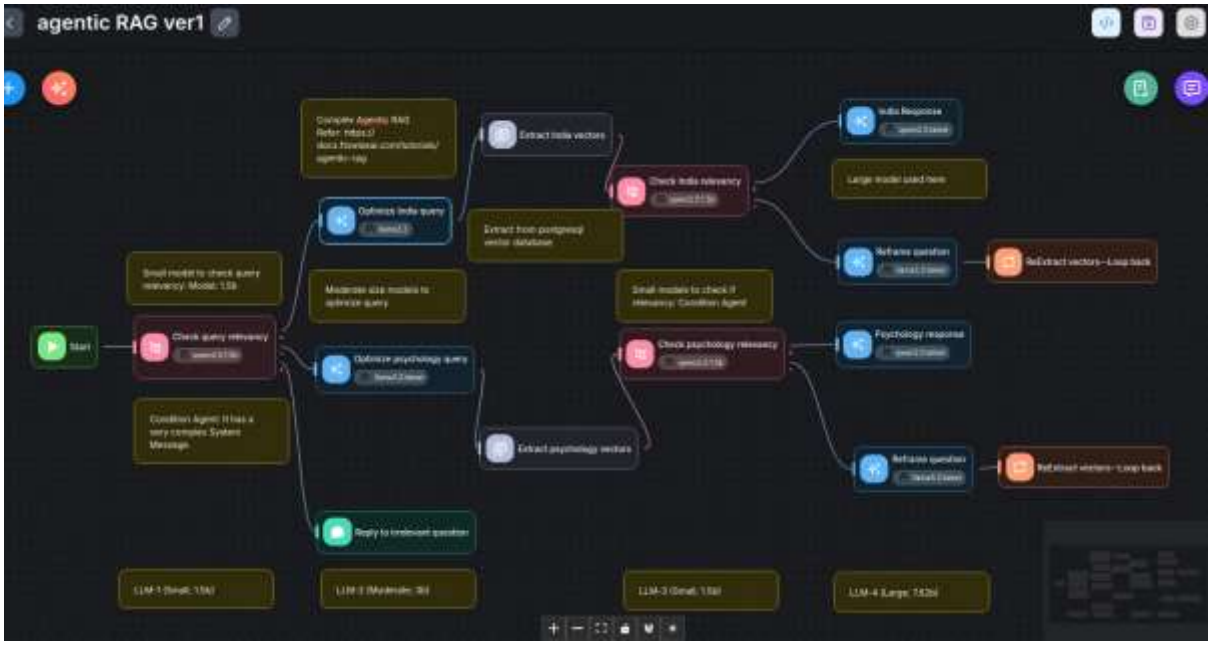


Figure 74: A complex agentic RAG

This complex RAG uses a mix of small, moderate and large LLMs.

****