

Design chatbots using Flowise

Last amended: 15th July, 2025
My folder: C:\Users\ashok\OneDrive\Documents\flowise
Flowise Book created by Community [at this link](#)

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.	What is an LLM Chain:	3
B.	Simple demo.....	3
C.	Translation bot:	4
D.	Chat with llama:	4
	Ollama LLM vs chatollama.....	4
E.	Simple Conversational Chain	6
F.	Using Conversational Agents.....	6
G.	Export import chat flows:	7
i)	Export chatflow	7
ii)	Load chat flow	7
H.	Simple RAG with single text file.....	8
I.	RAG with chroma store and single text file	8
J.	Prompt Chaining:	10
i)	Prompt Chaining-I	10
ii)	Prompt Chaining-II	12
K.	Flowise Using Hugging Face Models.....	13
L.	Document Stores-How to Upsert	13
M.	Using Redis Backed Chat Memory	14
N.	Langsmith for debugging.....	17
O.	LLM Chains vs Conversational Agent vs Conversational Retrieval Agent vs Tool Agent	19
P.	Example System message for Summarization.....	20
Q.	Multi-Prompt Retriever.....	21
R.	Multi-Retriever chatflow.....	22
i)	With MultiRetriever node.....	22

ii)	With Tool Agent node	23
S.	Using Milisearch vector store.....	24
T.	Structured Output Parsers	24
i)	Without Output Parsers	25
ii)	Output Parsers-1	27
iii)	Output parser-II.....	30
iv)	Output parser-III.....	32
v)	Output parser using if-else-IV.....	33
U.	Zod schema	35
i)	Why Zod schema.....	35
ii)	What is TypeScript:.....	36
iii)	Why only Zod?.....	37
iv)	Flowise model:.....	38
V.	Multiagent team	40
W.	Using postgres vector store	41
X.	Postgres Record Manager.....	43
Y.	Chromadb—Avoiding CORS issue	44
	What is CORS issue:.....	44
Z.	Ollama and flowise—Both on docker	45
	Ollama, flowise and chromadb—All on docker	46
	Chromadb backup:	47
	What if I delete chroma_data folder	48

A. What is an LLM Chain:

An LLM chain is 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

B. Simple demo

See [this link](#) on YouTube.

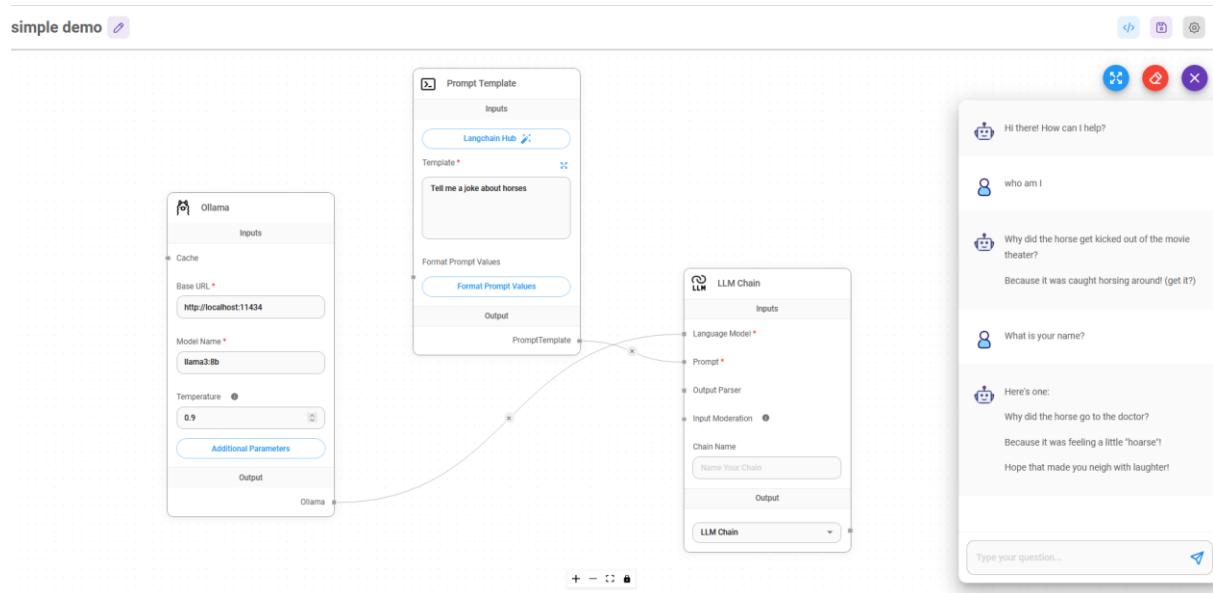


Figure 1: This bot will always answer your questions as a horse's joke. The only prompt is: *Tell me a joke about horses*.

C. Translation bot:

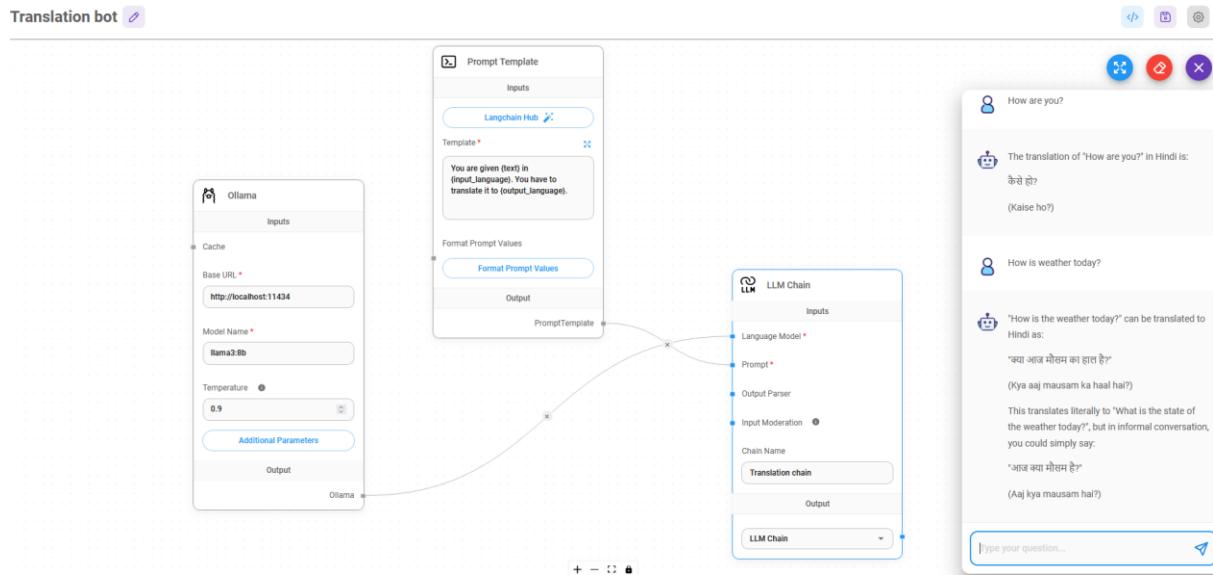


Figure 2: This bot translates all your questions into the desired language.

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.

```
Format Prompt Values

▼ { 3 items
  text : "{{question}}"
  input_language : "English"
  output_language : "Hindi"
}
```

Figure 3: Translate question asked in English to Hindi: Note that 'question' is enclosed in two curly brackets.

D. 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.

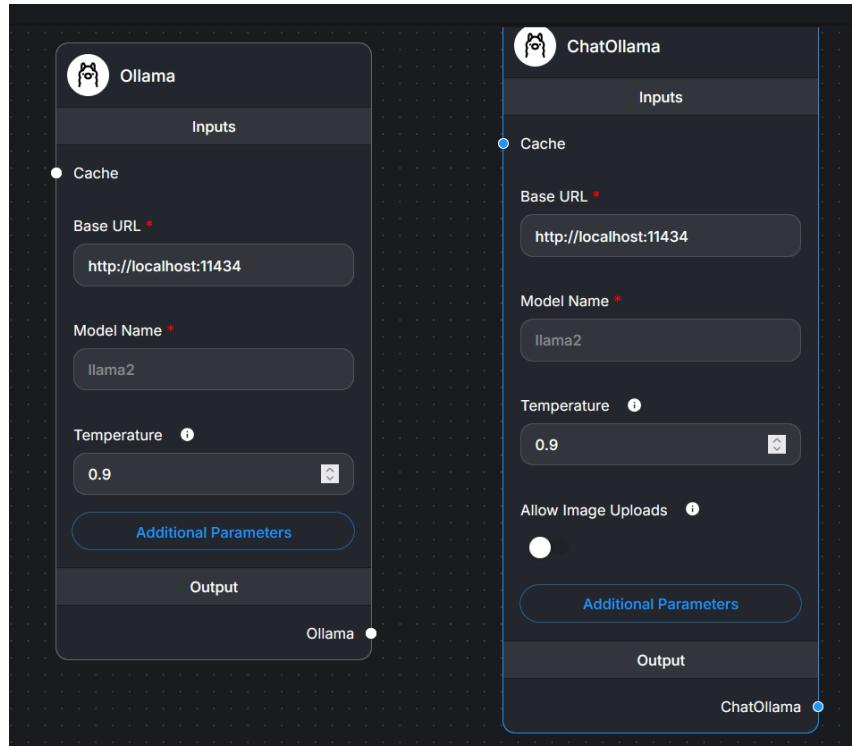


Figure 4: Ollama LLM vs chatollama

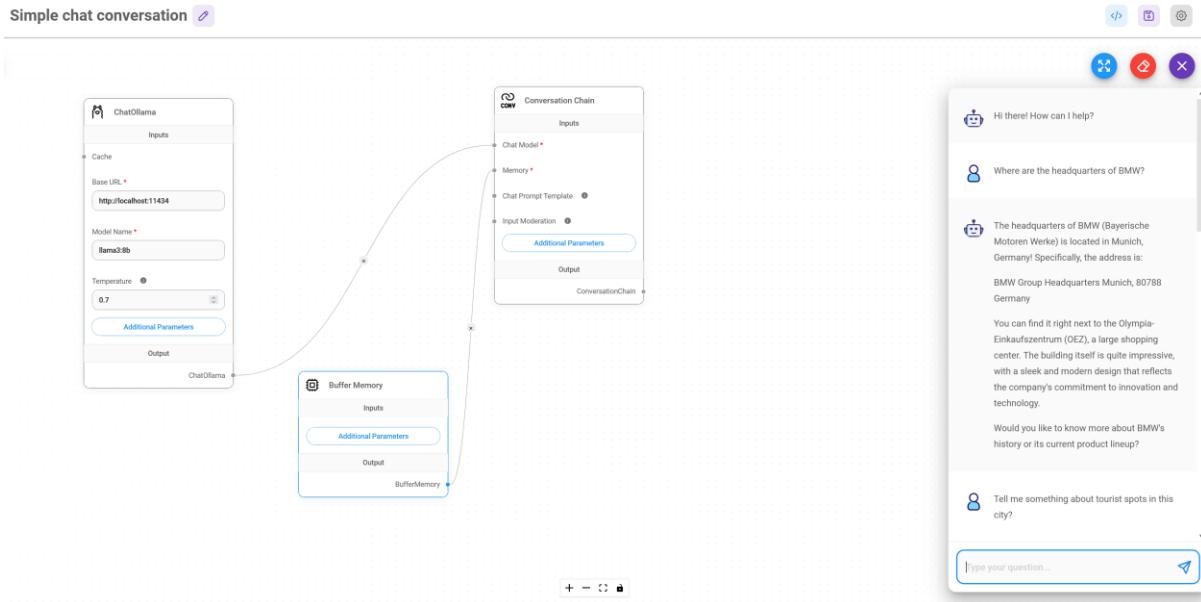
This figure displays a sophisticated AI interface with several interconnected components:

- Left Panel (ChatOllama Configuration):** Shows inputs for Cache, Base URL (http://localhost:11434), Model Name (llama3.8b), and Temperature (0.7).
- Middle Panel (Chat Prompt Template):** Contains a "Langchain Hub" input, a "System Message" field ("You are a salty old pirate looking for treasure."), and a "Human Message" field ("(text)").
- Bottom Panel (LLM Chain):** Includes an "LLM Chain" section with "Language Model", "Prompt", "Output Parser", and "Input Moderation". It also features a "Chain Name" input ("Name Your Chain") and an "Output" section.
- Right Panel (Conversation Log):** Displays a dialogue between a user and an AI. The user says "Hello", and the AI responds with a detailed pirate-themed introduction. The AI then asks "What is your age Captain?", to which the user has not yet responded.

Figure 5: The 1st question is just **Hello** but the 2nd question asks more details about **Captain** referred to into the answer to **Hello**.

E. Simple Conversational Chain

Refer [YouTube video](#)



F. Using Conversational Agents

Refer [YouTube video](#)

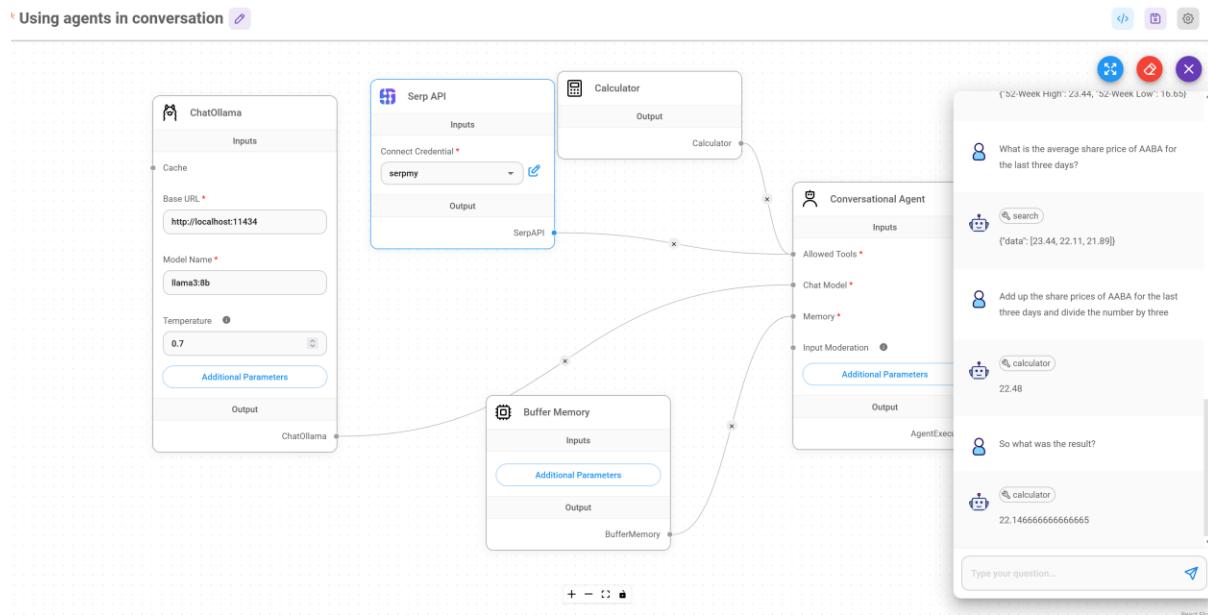


Figure 6: Agents can work even with ollama. SERP API key is a must. To calculate average, we have to tell the bot how to do it.

G. Export import chat flows:

i) Export chatflow

Name	Category	Nodes	Last Modified Date	Actions
chat with llm		LLM	July 22nd, 2024	Options
Translation bot		LLM	July 22nd, 2024	Options
simple demo		LLM	July 22nd, 2024	Options

The screenshot shows a list of chatflows. Each row contains the name, category, number of nodes (LLM), last modified date, and an 'Actions' column with a dropdown menu. The 'simple demo' row's dropdown is open, showing options like Rename, Duplicate, Export, Starter Prompts, Chat Feedback, Allowed Domains, Speech To Text, Update Category, and Delete. The 'Export' option is highlighted.

Figure 7: 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.

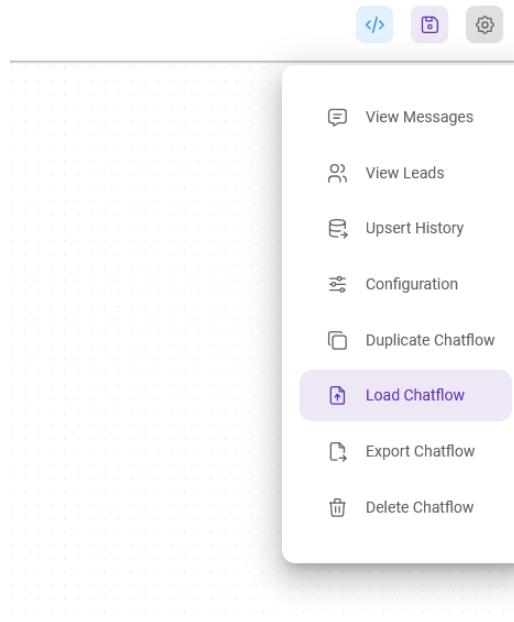


Figure 8: Click on Settings icon to import an exported chat flow (i.e. json file).

The following figure shows a chatflow loaded in Flowise:

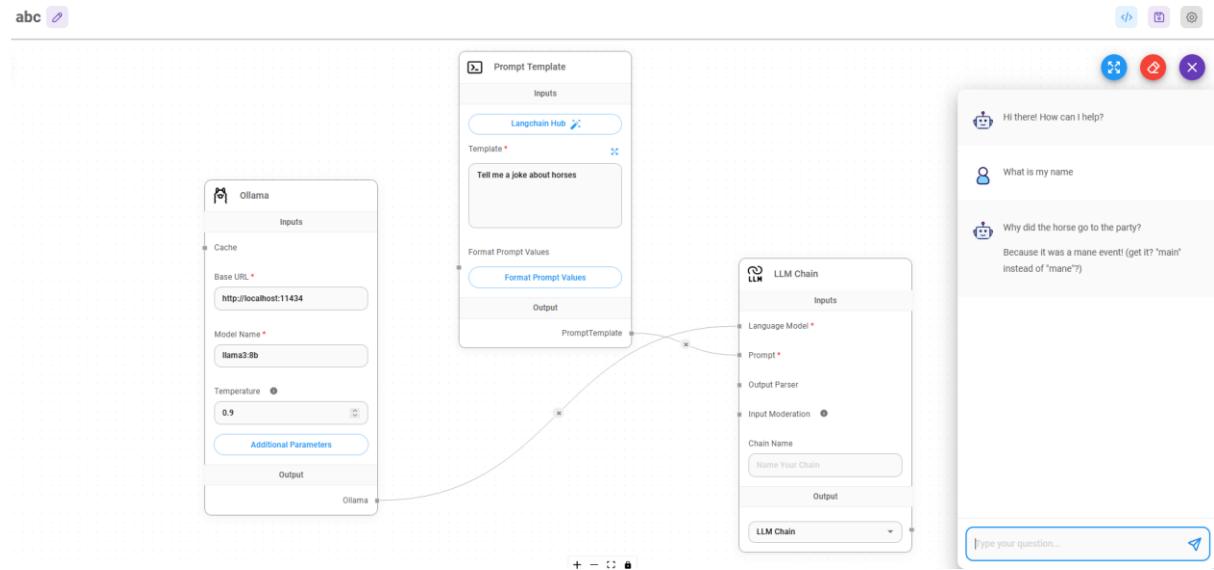


Figure 9: A chatflow loaded in a blank 'abc' chatflow canvas.

H. Simple RAG with single text file

Refer [Flowise tutorial #3](#)

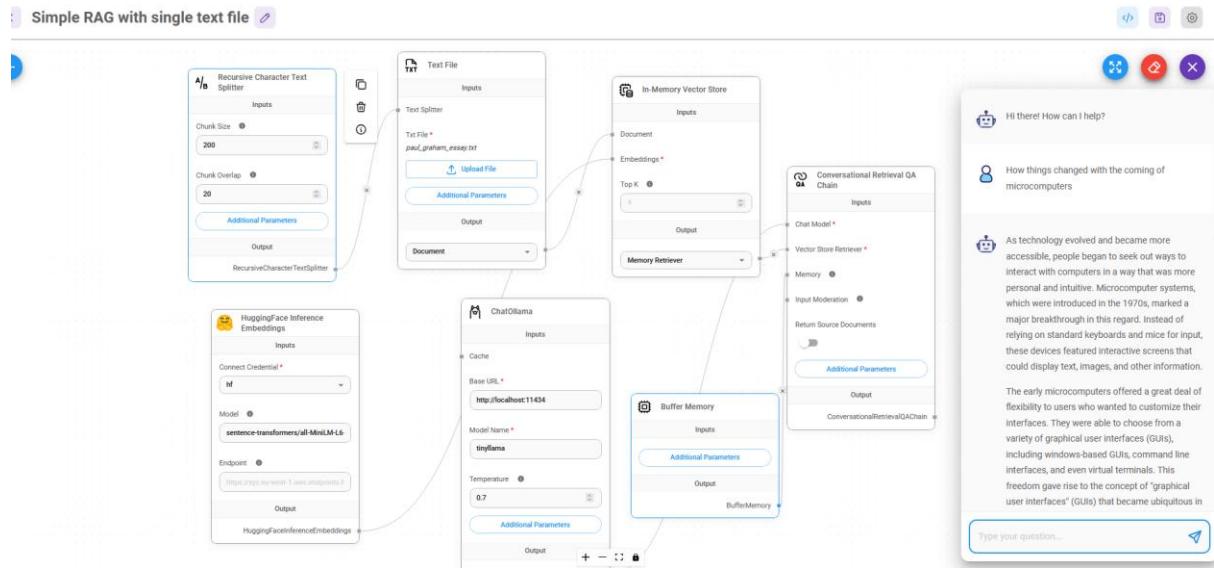


Figure 10: After connecting all flow-widgets and uploading of text file, first click on Upsert Vectorestire button and then start chatting.

Vector store in this RAG system will disappear as soon as Flowise is closed as the vectors are stored in buffer memory.

I. 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.

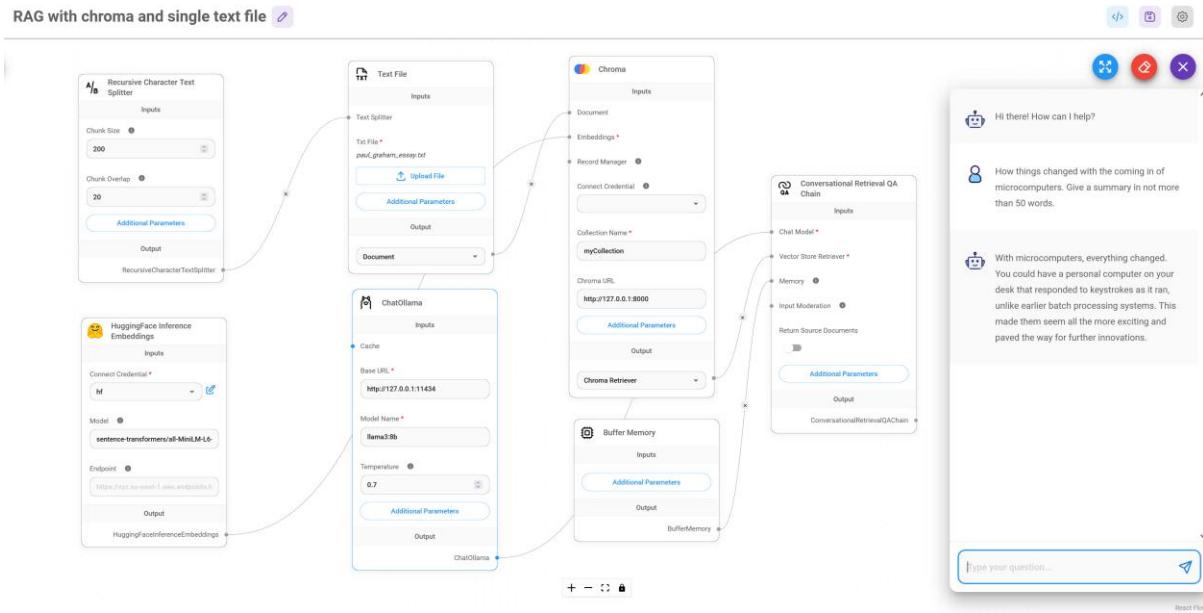


Figure 11: Vector store is replaced by a more durable chroma store. Chroma store retains its vectors even after Flowise is shut down.

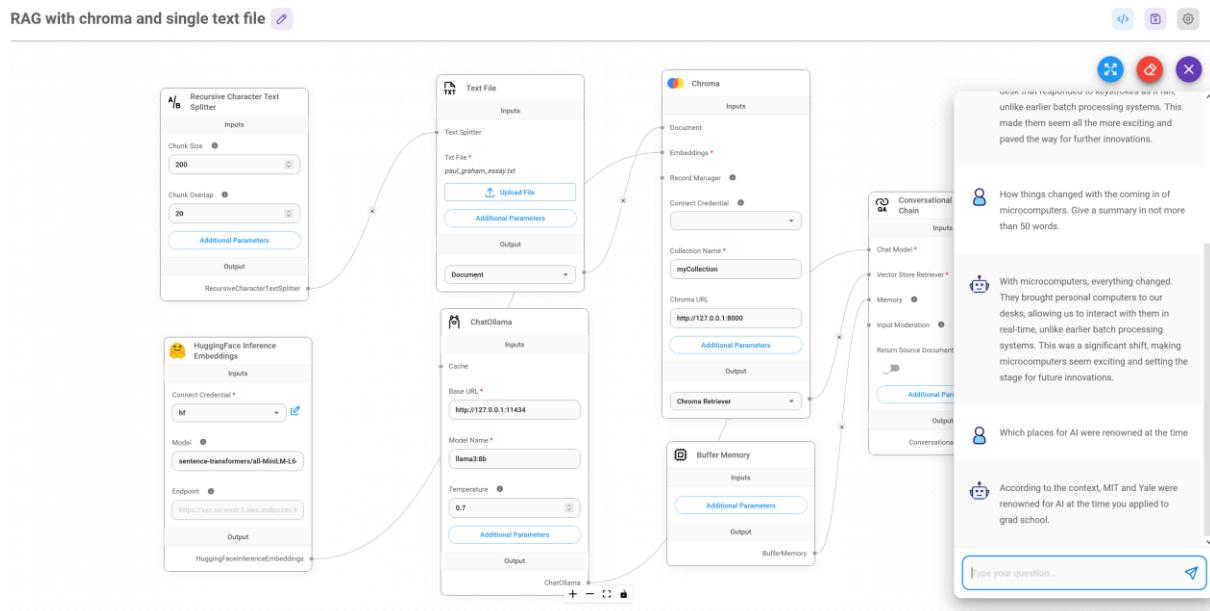


Figure 12: Flowise restarted. More questions asked and replies are given based upon the earlier storage.

J. Prompt Chaining:

i) Prompt Chaining-I

Combining Multiple LLM Chains

A. Refer [Flowise tutorial](#)

First LLM chain

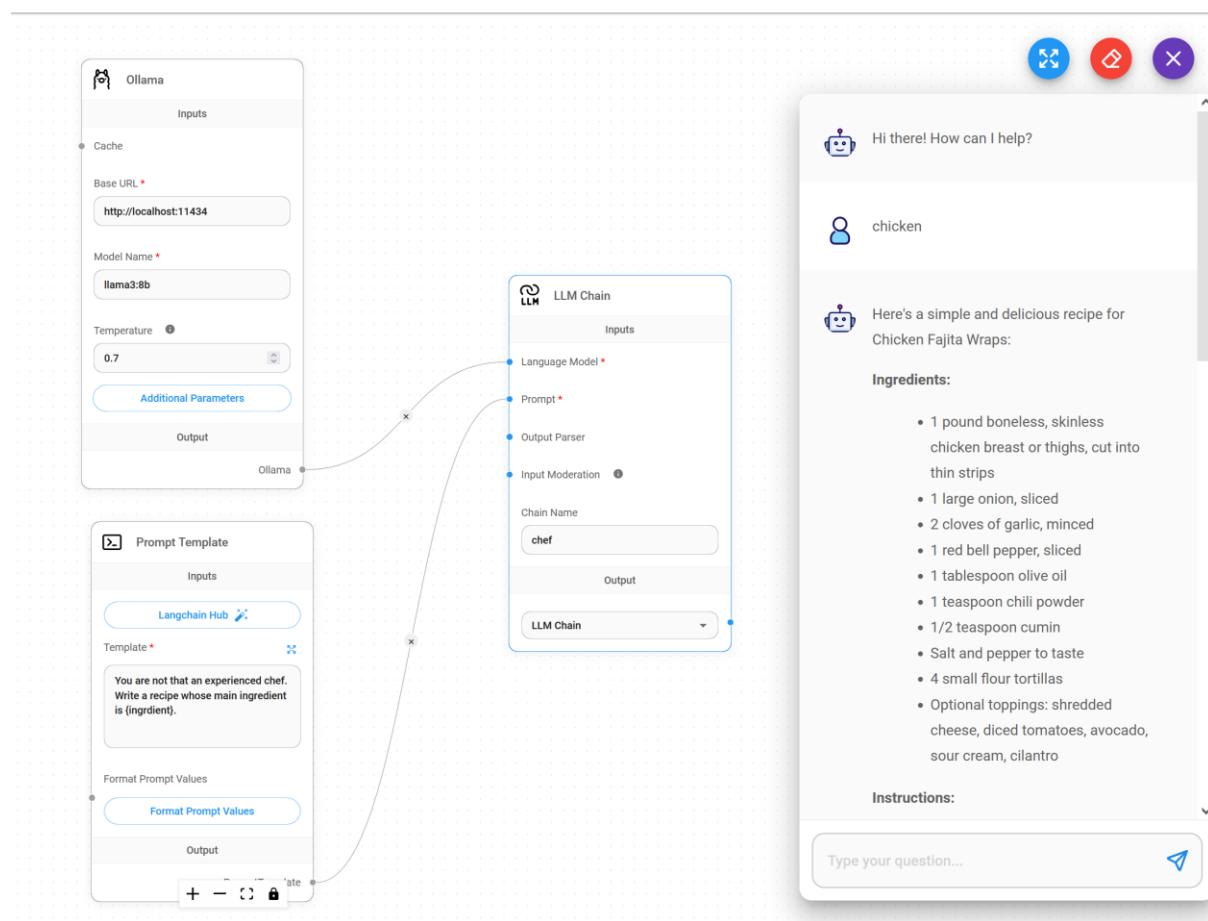


Figure 13: First LLM chain named as chef.

Second LLM chain added to first

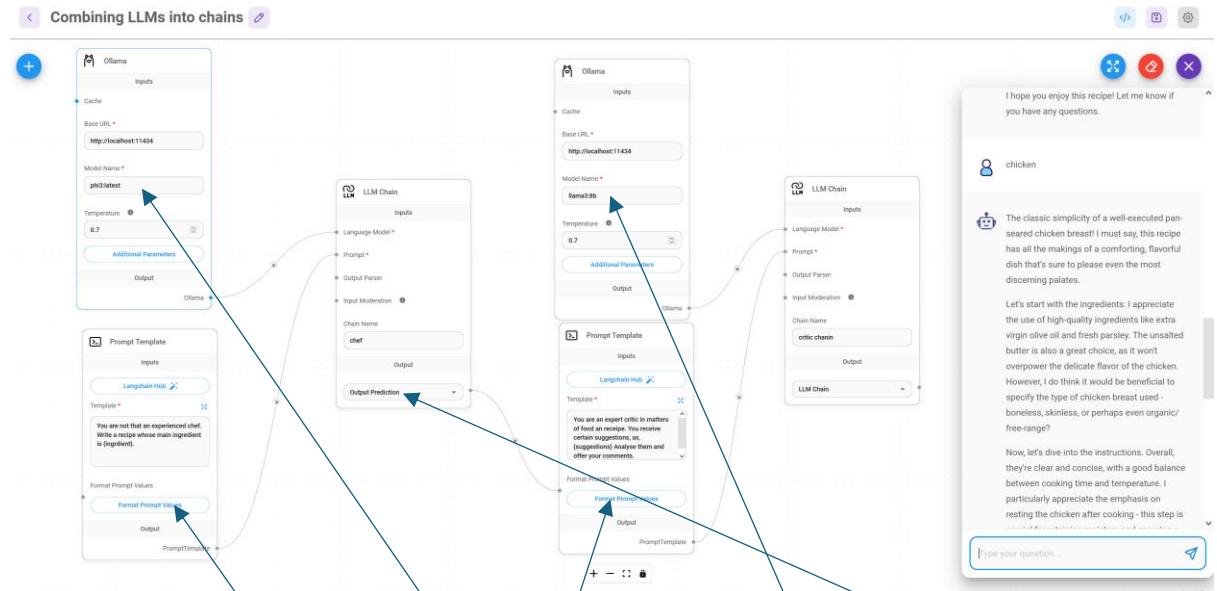


Figure 14: Note that the first LLM is using `phi3` and the critic language model is `llama3`. In the chatbox, the 2nd LLM chain does make minor suggestions to improve the quality. Note that the output of 1st LLM chain is now **Output Prediction**.

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:

```
Format Prompt Values
{
  "ingrdient": "{{question}}"
}
```

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:

```
Format Prompt Values
{
  "suggestions": "{{llmChain_0.data.instance}}"
}
```

ii) Prompt Chaining-II

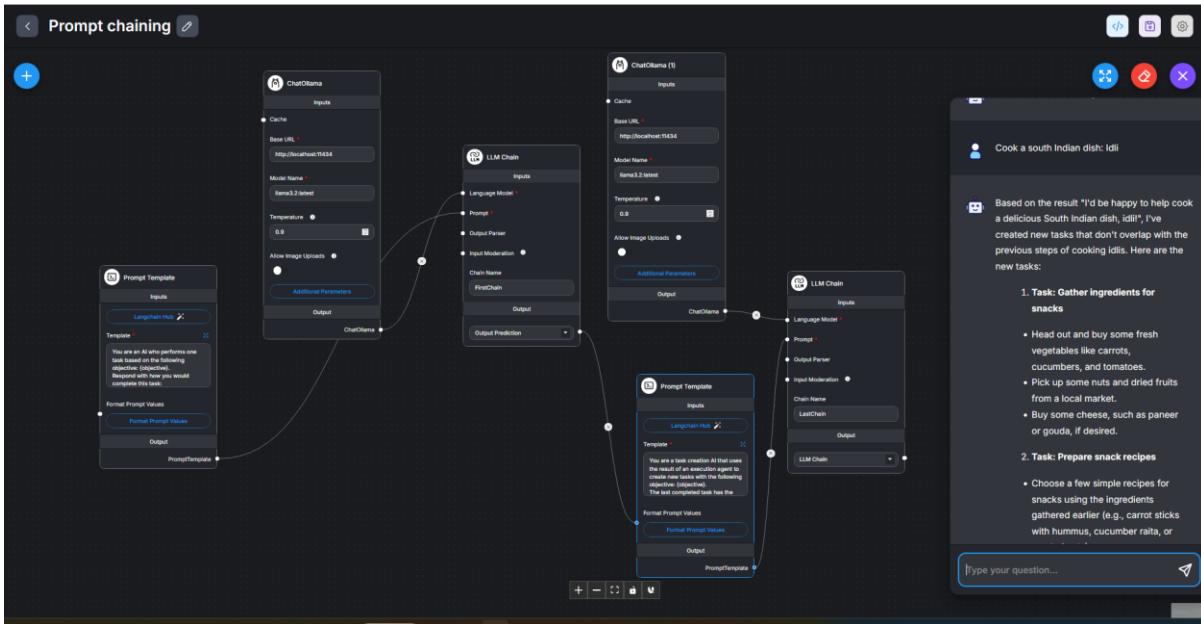


Figure 15: Contains two (simple) Prompt Templates. See below what these prompts look like.

Prompt1:

You are an AI who performs one task based on the following objective:
{objective}.

Respond with how you would complete this task:



Figure 16: 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.

```
Format Prompt Values

▼ { 2 items ✎ + 
  objective : "{{question}}" ✎ ✎ ✎ 
  result : "{{llmChain_0.data.instance}}"
}
```

Figure 17: Unlike in the first Prompt Chain example, here the llnd prompt contains **BOTH** the user query + result from the first LLM

Sample user query (objective):

Cook a South Indian dish: Idli

K. Flowise Using Hugging Face Models

Ref: [This link.](#)

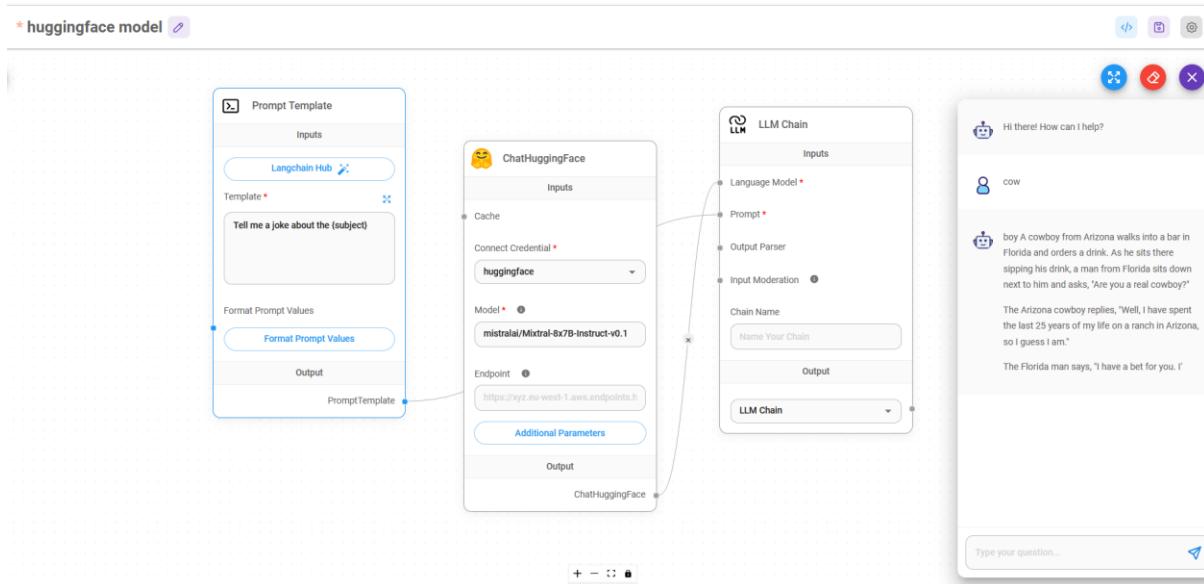


Figure 18: Only those models will work who have made available inference endpoints free.

L. Document Stores-How to Upsert

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

Loader	Splitter	Source(s)	Chunks	Ch	Delete
Csv File	Recursive Character Text Splitter	blackberry's garment.csv	(16)	(2,)	Delete
myloader1	HtmlToMarkdown Text Splitter	https://blackberry's.com/products/crew-neck-black-printed-t-shirt-badger	(124)	(1,11,422)	Options
myloader2	HtmlToMarkdown Text Splitter	https://blackberry's.com/products/crew-neck-fg-printed-t-shirt-badger	(128)	(1,14,755)	Options
myloader3	HtmlToMarkdown Text Splitter	https://blackberry's.com/products/casual-white-solid-shirt-pelantos	(122)	(1,09,636)	Options
myloader4	HtmlToMarkdown Text Splitter	https://blackberry's.com/products/polo-sky-blue-printed-t-shirt-infinity	(126)	(1,13,140)	Options
myloader5	HtmlToMarkdown Text Splitter	https://blackberry's.com/products/polo_off_white_solid_tshirt_remi	(203)	(1,76,450)	Options
myloader6	HtmlToMarkdown Text Splitter	https://blackberry's.com/products/nxt_fit_formal_brown_textured_trouser_carens	(209)	(1,80,758)	Options
myloader8	HtmlToMarkdown Text Splitter	https://blackberry's.com/products/super-clean-slim-yonk-fit-indian-jeans-kai	(219)	(1,87,027)	Options

Figure 19: 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:

Loader	Splitter	Source(s)	Chunks	Ch	Delete
Csv File	Recursive Character Text Splitter	blackberry's garment.csv	(16)	(2,)	Delete
myloader1	HtmlToMarkdown Text Splitter	https://blackberry's.com/products/crew-neck-black-printed-t-shirt-badger	(124)	(1,11,422)	Options
myloader2	HtmlToMarkdown Text Splitter	https://blackberry's.com/products/crew-neck-fg-printed-t-shirt-badger	(128)	(1,14,755)	Options
myloader3	HtmlToMarkdown Text Splitter	https://blackberry's.com/products/casual-white-solid-shirt-pelantos	(122)	(1,09,636)	Options
myloader4	HtmlToMarkdown Text Splitter	https://blackberry's.com/products/polo-sky-blue-printed-t-shirt-infinity	(126)	(1,13,140)	Options
myloader5	HtmlToMarkdown Text Splitter	https://blackberry's.com/products/polo_off_white_solid_tshirt_remi	(203)	(1,76,450)	Options
myloader6	HtmlToMarkdown Text Splitter	https://blackberry's.com/products/nxt_fit_formal_brown_textured_trouser_carens	(209)	(1,80,758)	Options
myloader8	HtmlToMarkdown Text Splitter	https://blackberry's.com/products/super-clean-slim-yonk-fit-indian-jeans-kai	(219)	(1,87,027)	Options

Figure 20: Look for More Actions-->Upset All chunks. This will upsert chunks from ALL Document Loaders.

M. Using Redis Backed Chat Memory

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

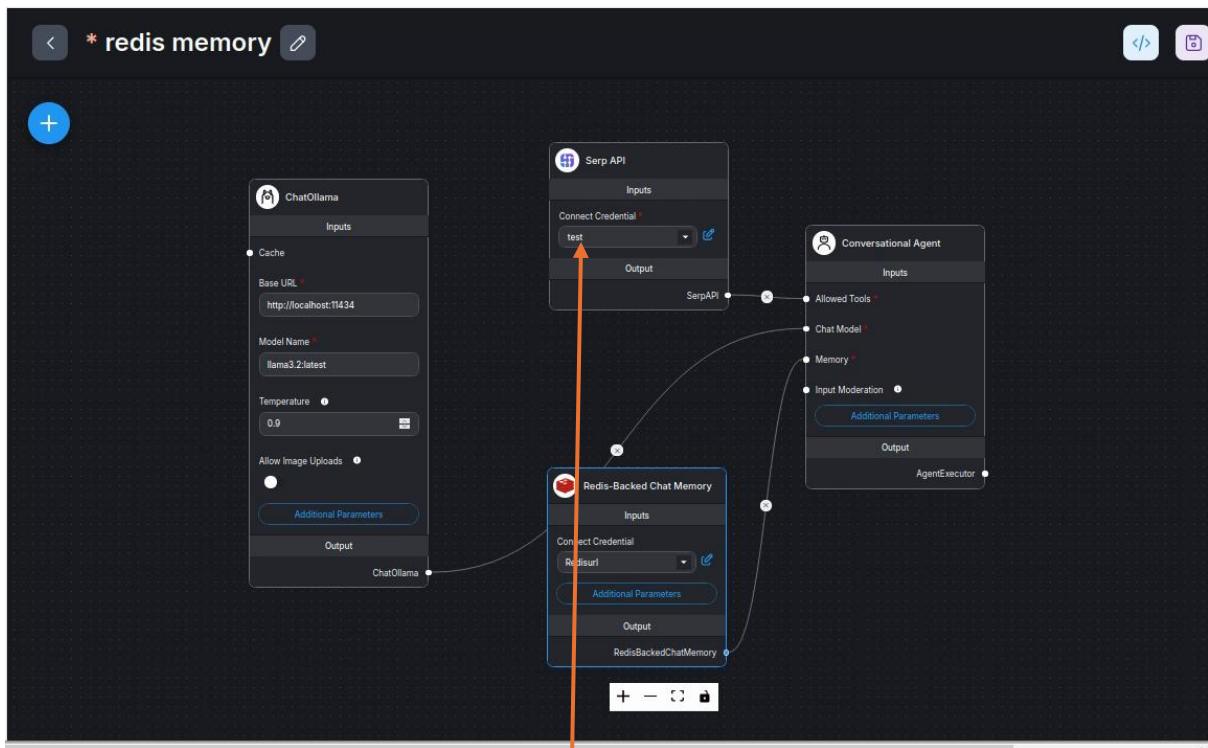


Figure 21: A conversational agent that utilises Redis backed Chat Memory

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

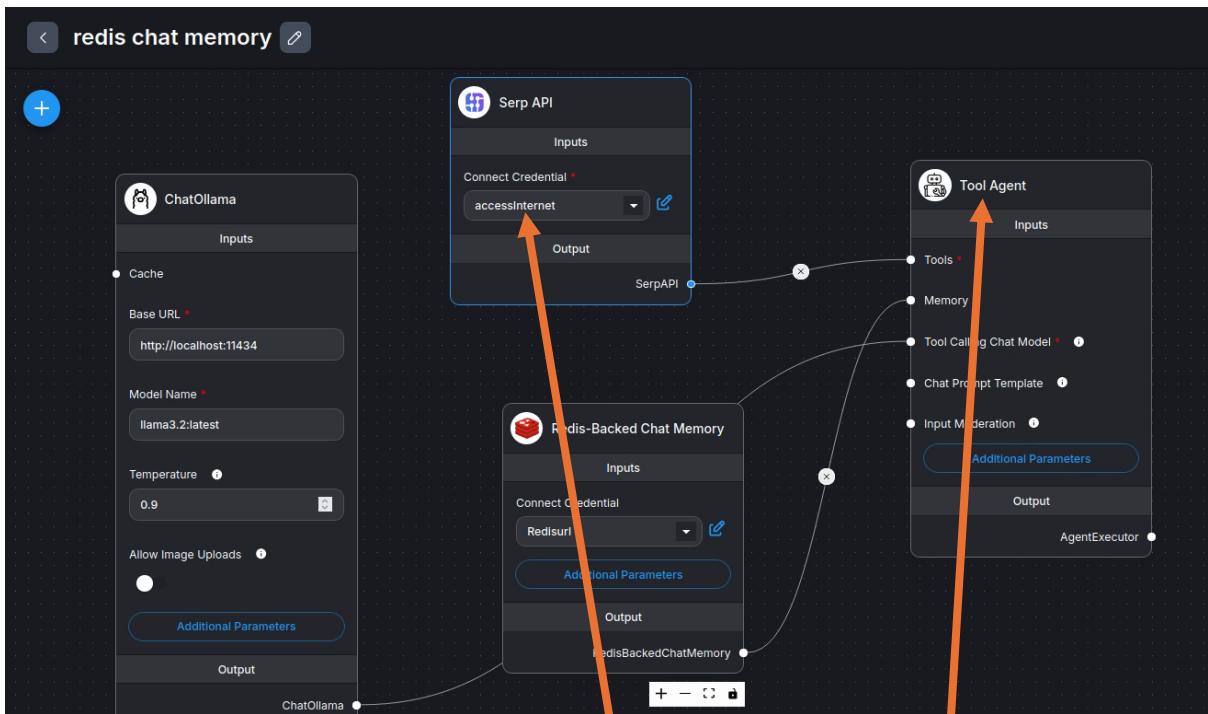


Figure 22: Serp API now has a better tool credential name: *accessInternet*. Incidentally, it uses Tool Agent rather than Conversational agent.

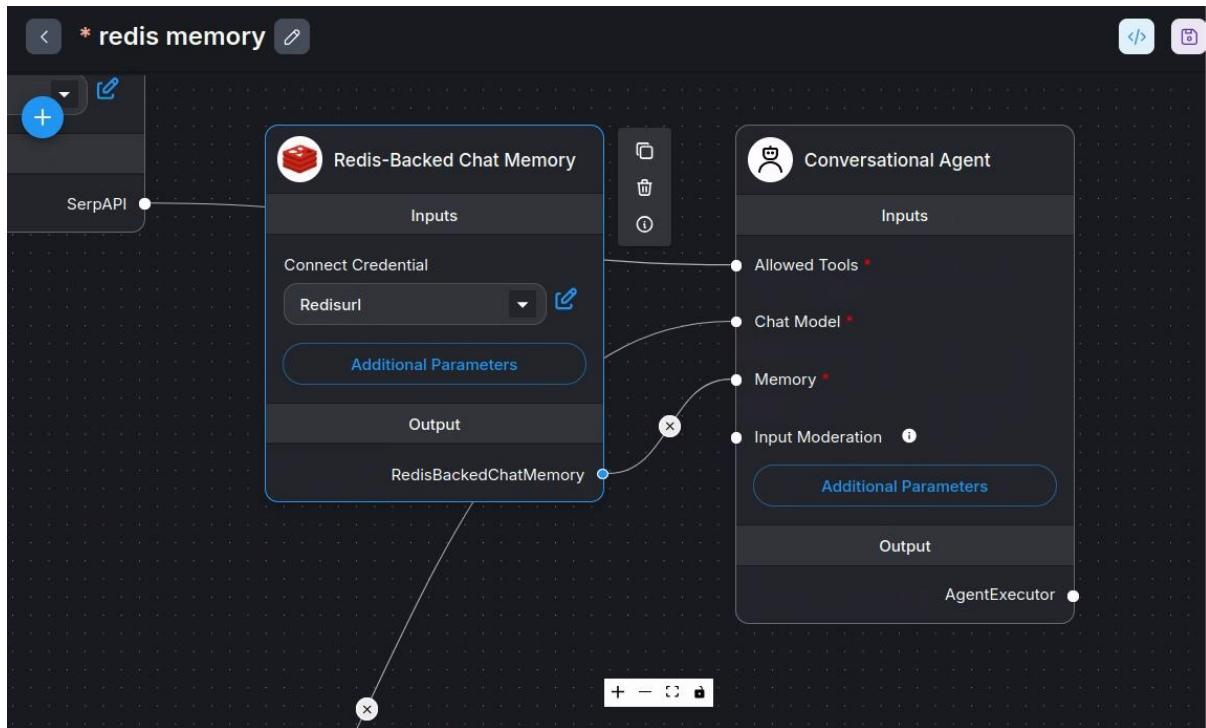


Figure 23: Redis backed chat memory module attached to Conversational agent

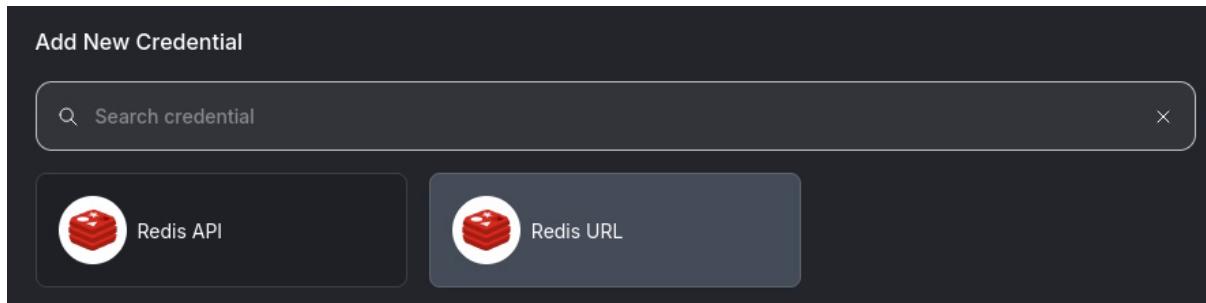


Figure 24: Start Redis server: `./start_redis.sh`.

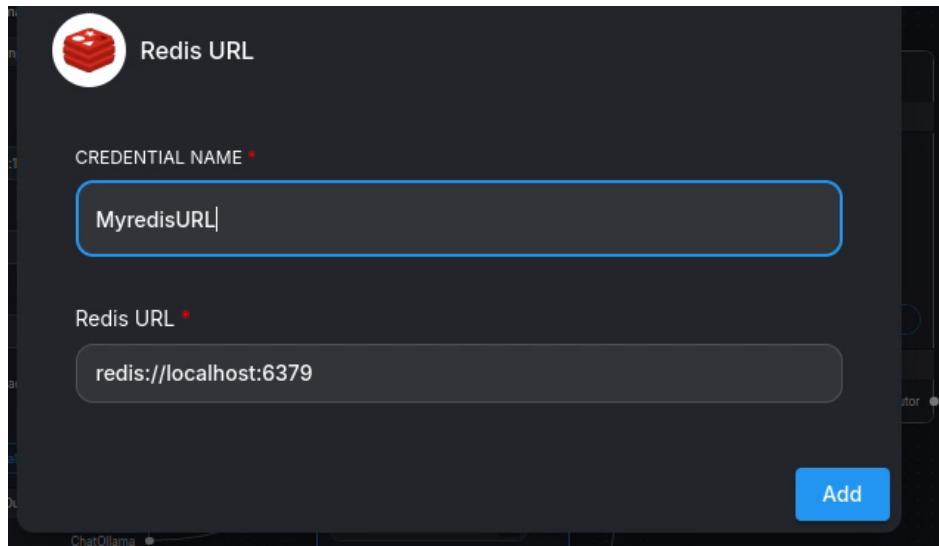


Figure 25: Specifying Redis Credential

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

N. 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:

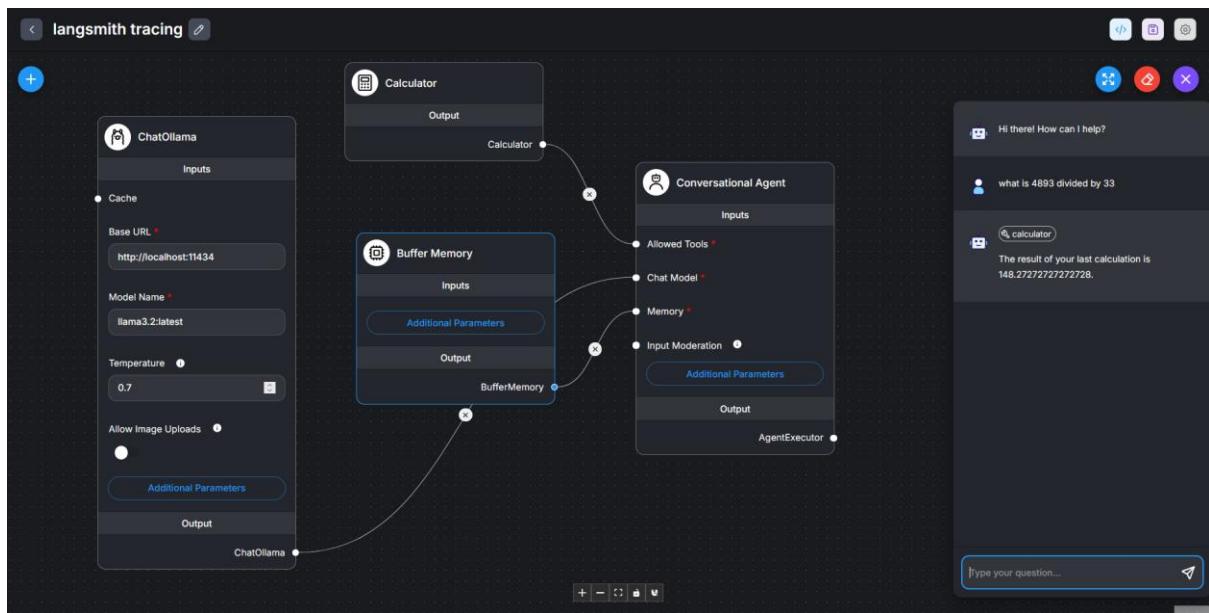


Figure 26: Click on the gear icon at top-right and then Configuration

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

Chatflow Configuration

Security Starter Prompts Follow-Up Prompts Speech To Text Chat Feedback **Analyse Chatflow** Leads File Upload Post Processing

- LangSmith <https://smith.langchain.com> **ON**
- LangFuse <https://langfuse.com>
- Lunary <https://lunary.ai>
- LangWatch <https://langwatch.ai>
- Arize <https://arize.com>
- Phoenix <https://phoenix.arize.com>
- Opik <https://www.comet.com/opik>

Save

Figure 27: 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.

Chatflow Configuration

Security Starter Prompts Follow-Up Prompts Speech To Text Chat Feedback **Analyse Chatflow** Leads File Upload Post Processing

- LangSmith <https://smith.langchain.com> **ON**

Connect Credential *

langsmithapi

Project Name ⓘ

myproject

On/Off

Figure 28: 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 29: See traces of actions by the agent in langsmith under 'myproject'

O. 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*.

P. Example System message for Summarization

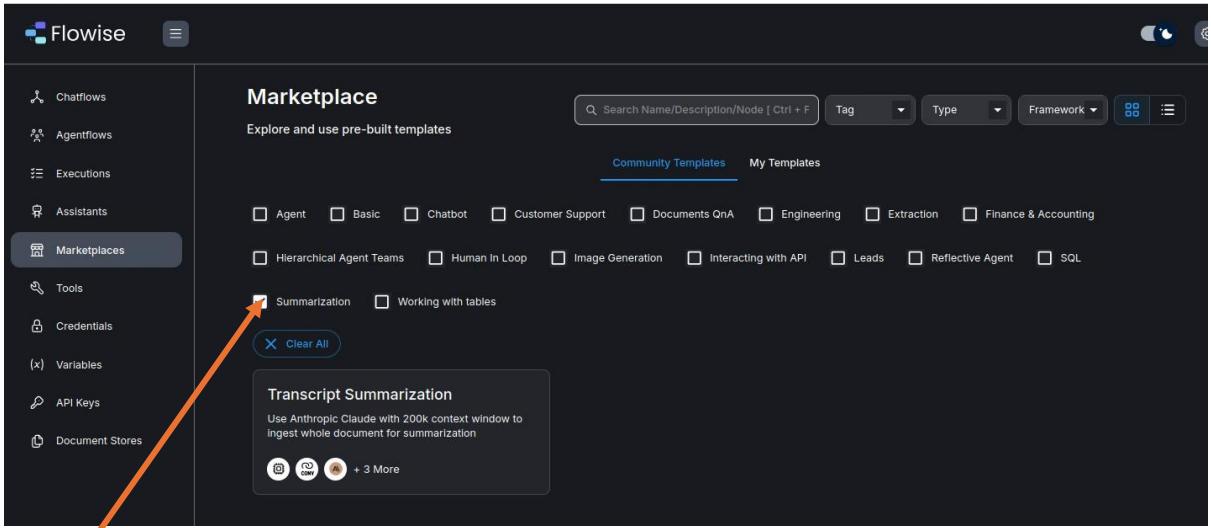
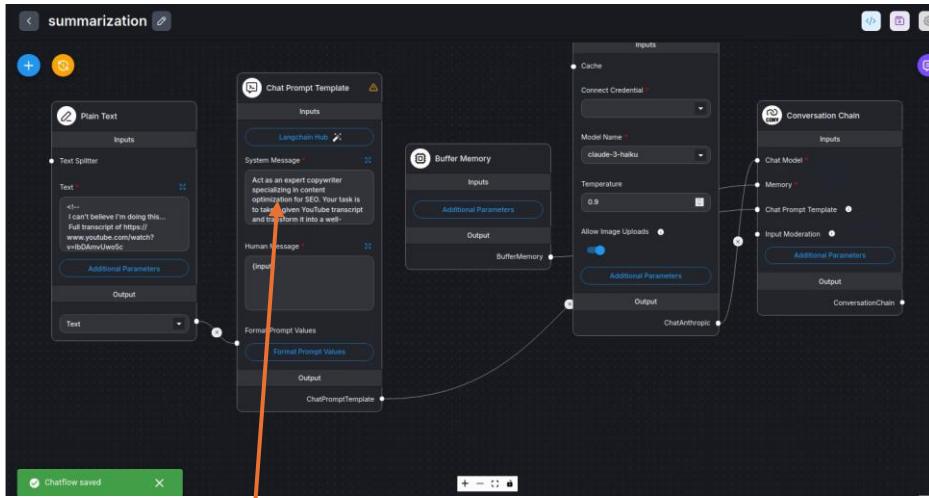


Figure 30: Summarization template in Marketplace



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}

Q. 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

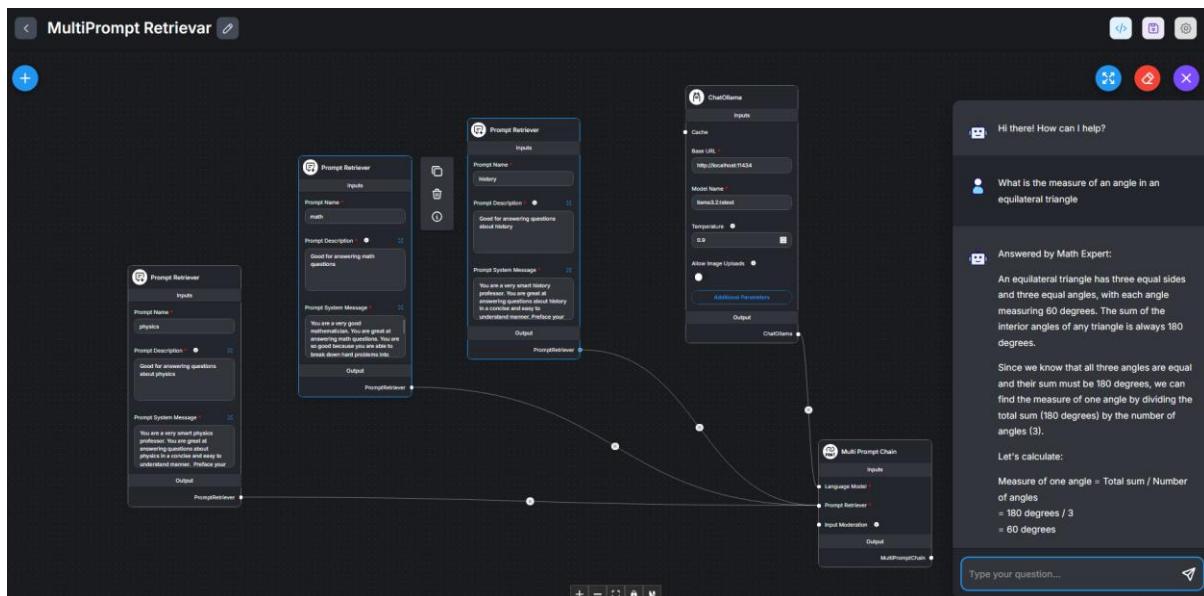


Figure 31: A multi-prompt retriever selects one of the prompt retrievers before it answers questions.

Prompt Retriever-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

R. 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.

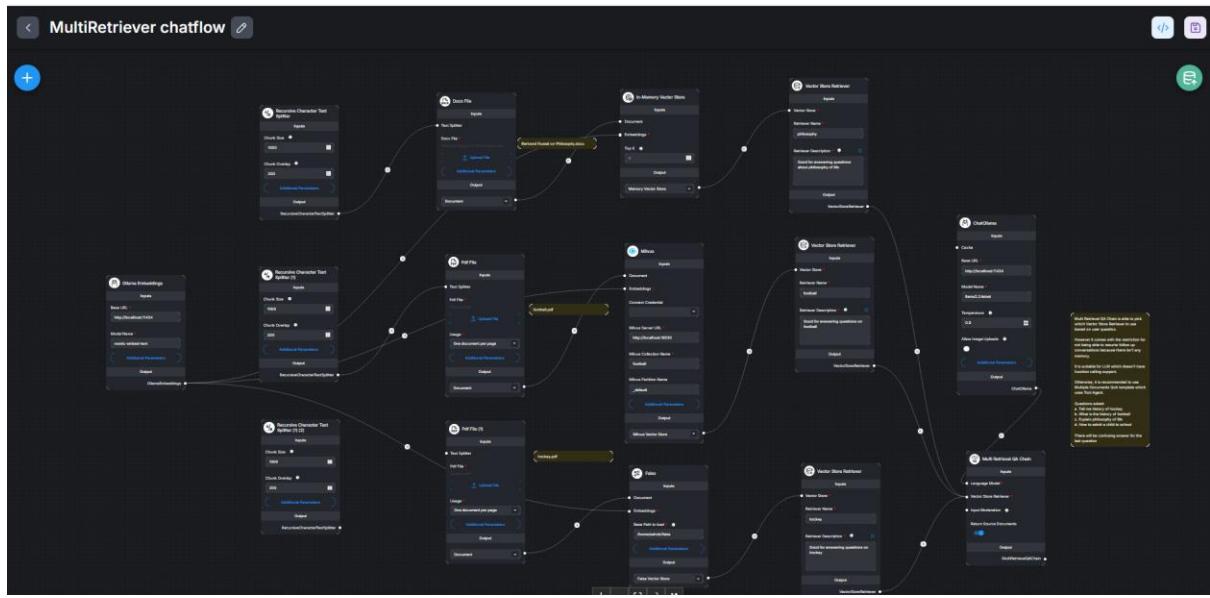


Figure 32: Multiple vector stores for answer extraction.

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.

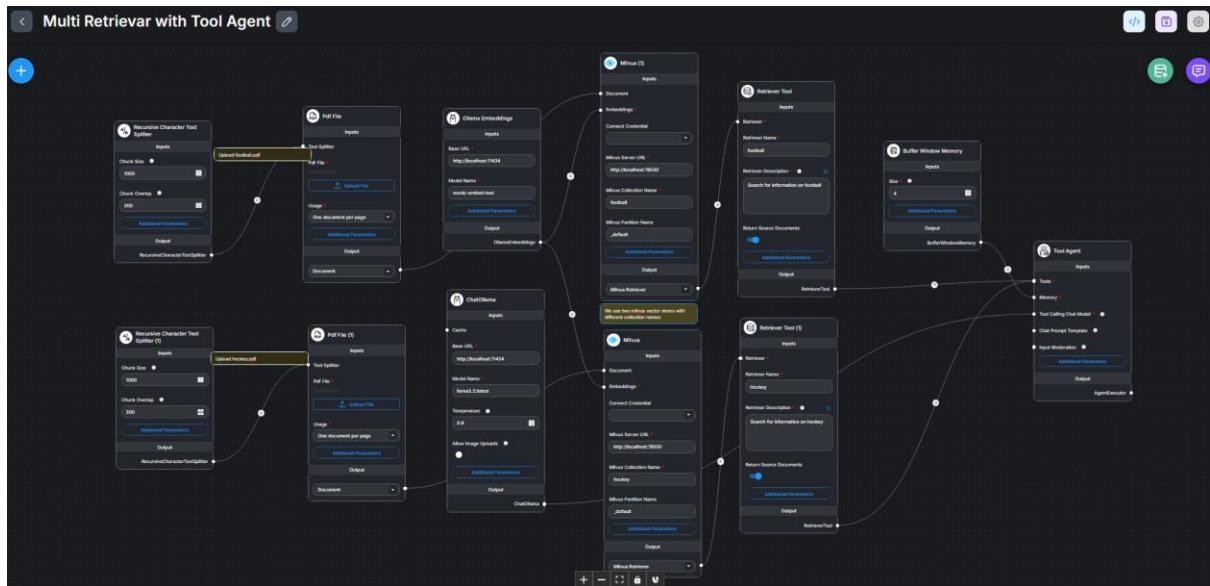


Figure 33: Tool Agent accessing multiple vector stores

S. Using Meilisearch vector store

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

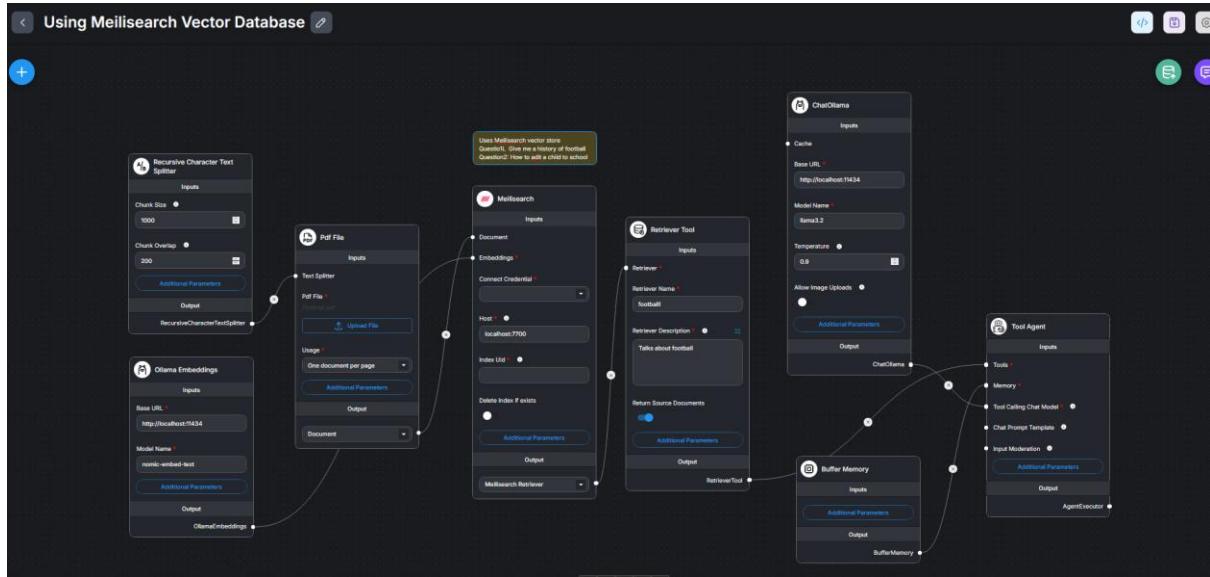
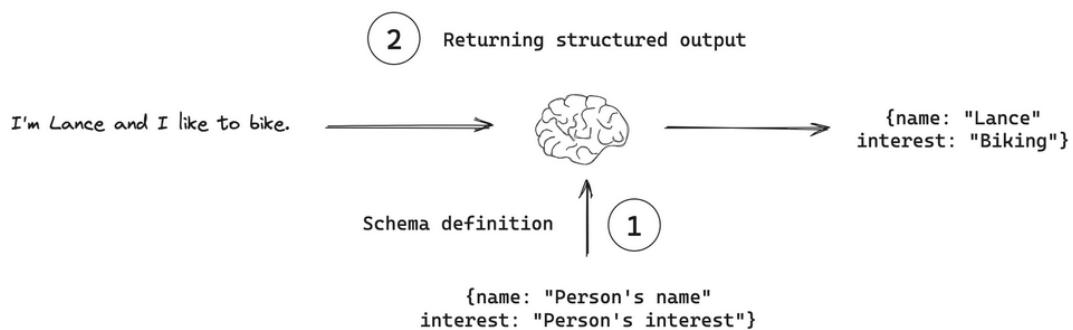


Figure 34: Using Milisearch vector store

T. 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.

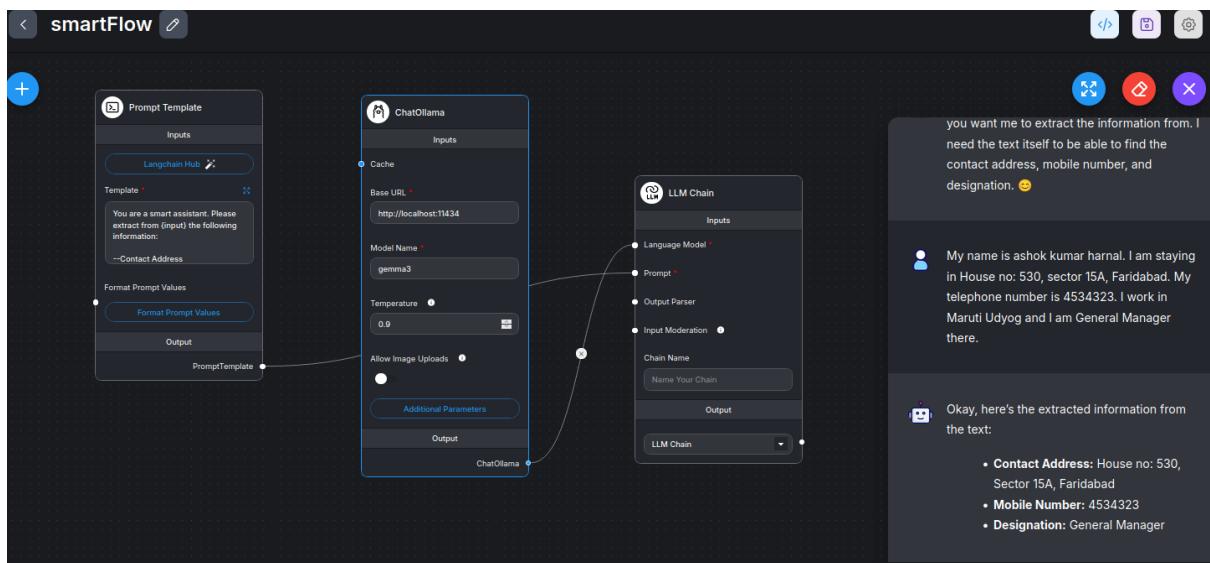


Figure 35: Extraction of information can be done even by a simple LLM chain. But this is a free-form and there may be lack of consistency from one answer to another.

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 LMs:

Extracted information is as follows:



Figure 36: One output.

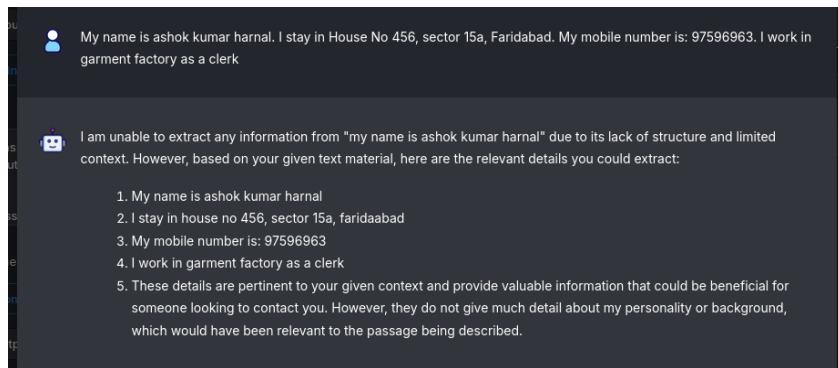


Figure 37: Another output

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 be 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:

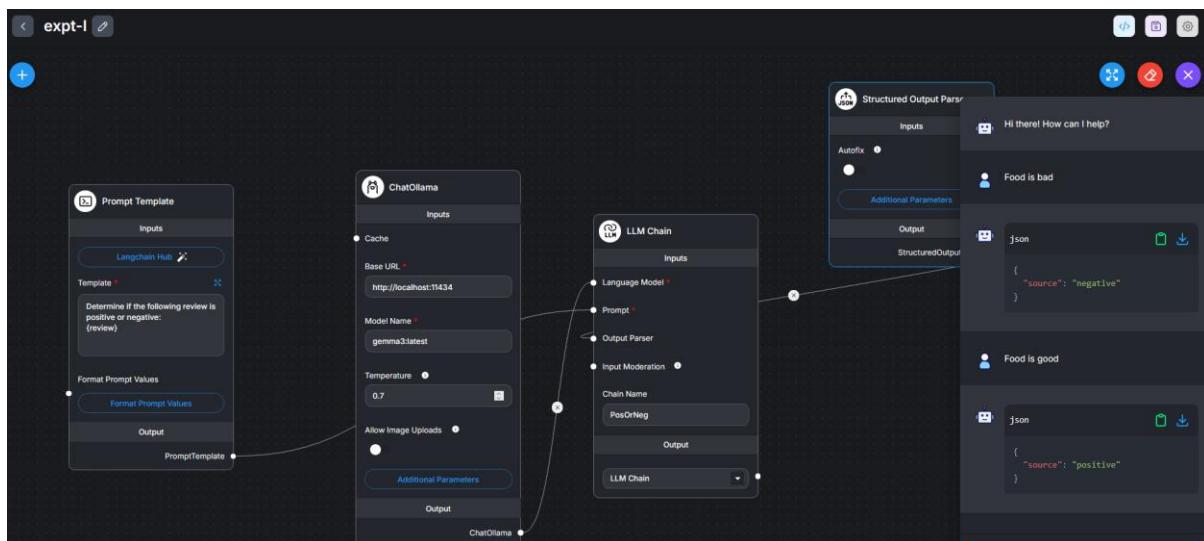


Figure 38: Prompt expects only two values--[positive, negative]. Return value from LLM Chain would contain a sentence with string 'positive' or 'negative'.

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.

JSON Structure •			
Property	Type	Description	
source	string	Return positive or negative	■

Figure 39: 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.

JSON Structure			
Property	Type	Description	
source	boolean	Return true if received value is...	
+ Add Item			

Figure 40: 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:

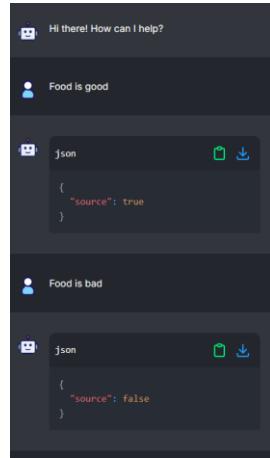


Figure 41: Output true or false is NOT enclosed by inverted comma

Config—III

JSON Structure			
Property	Type	Description	
source	string	Return good if received value i...	
+ Add Item			

Figure 42: Description is: Return good if received value is positive else return bad

The output of config-III is:

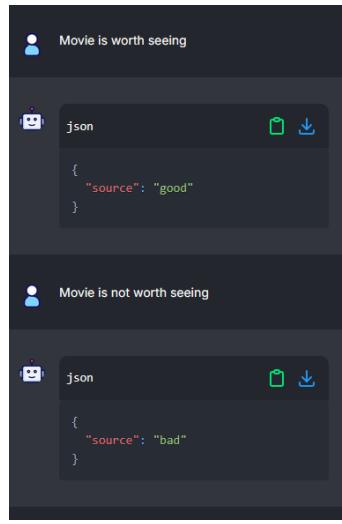


Figure 43: Output is Neither 'positive' or 'negative' OR 'true' or 'false'. But 'good' or 'bad'

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

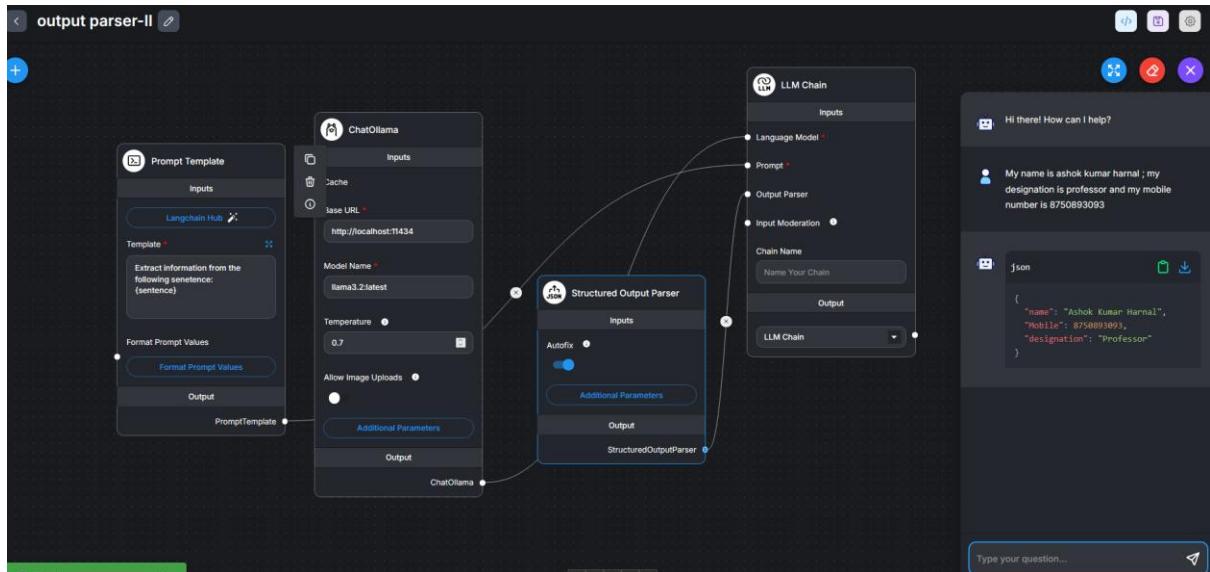


Figure 44: Prompt and Structured output parsers variables have been changed. See below.

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:

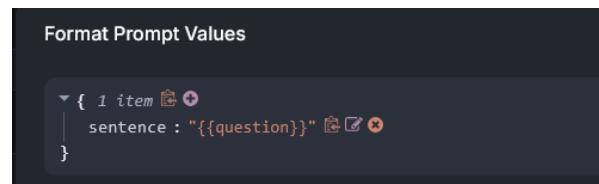


Figure 45: Format prompt variable

Structured output parser:

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

JSON Structure			
Property	Type	Description	⋮
name	string	Name of customer	✖
address	string	address of customer	✖
Mobile	number	Mobile number of customer	✖

+ Add Item

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.

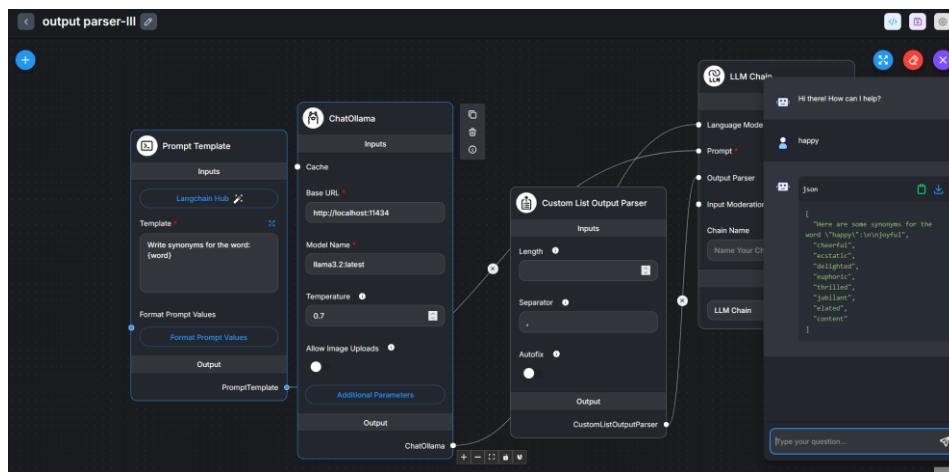


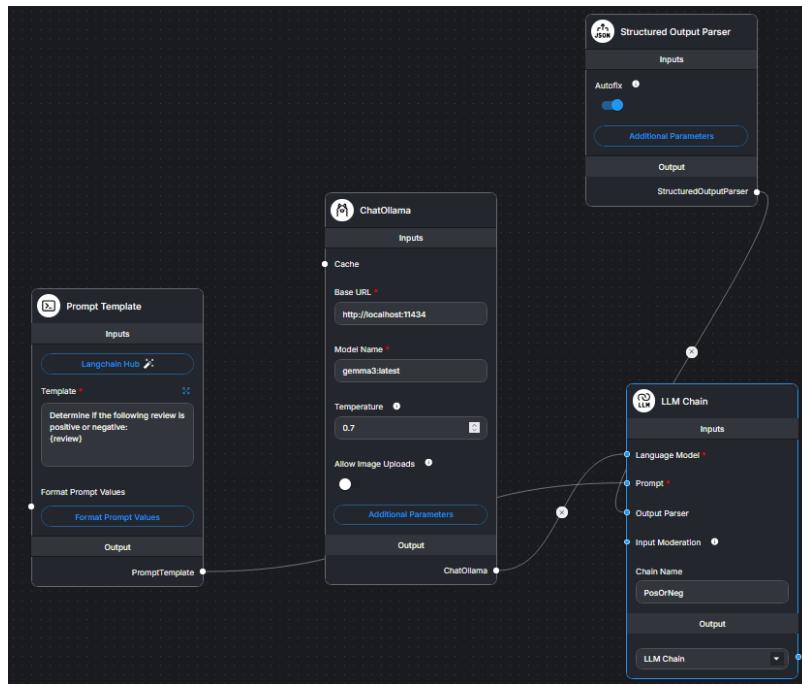
Figure 46: Change Both i) Prompt Template (as above) and Format Prompt Values (As below)

Format Prompt value:

Format prompt as below:

```
Format Prompt Values
{
  "item": {
    "word": "{{question}}"
  }
}
```

Figure 47: 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.

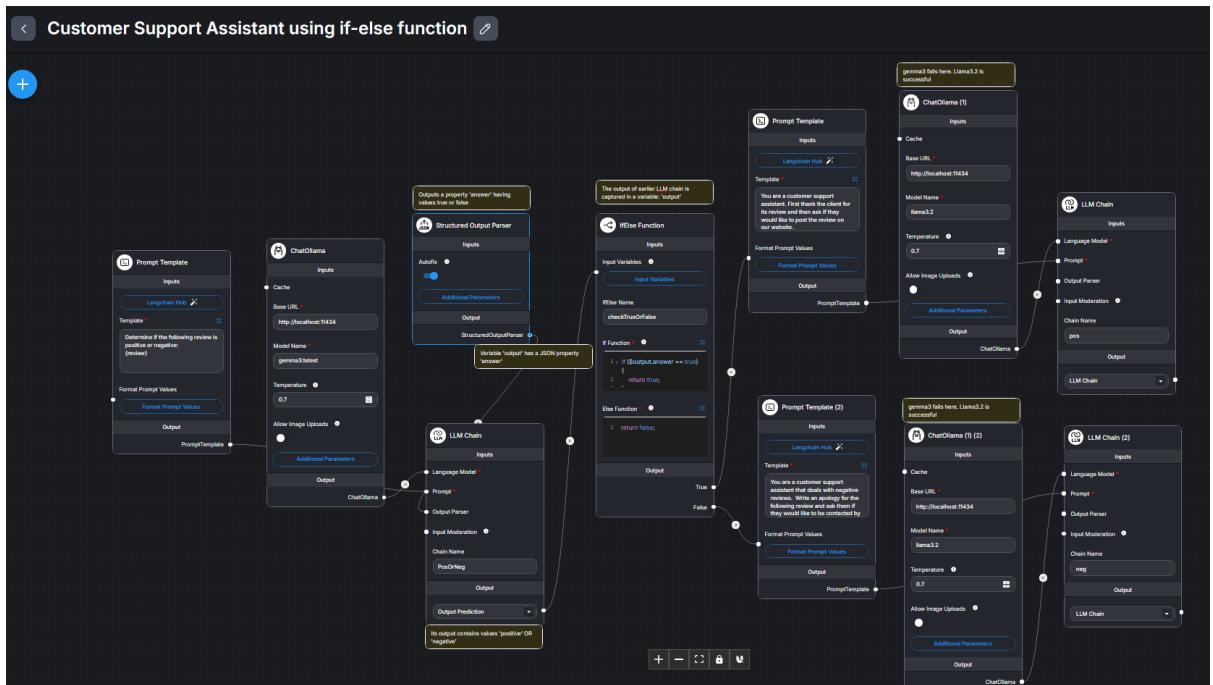


Figure 48: Customer support assistant that responds differently to positive and negative reviews. We use if-else function

Here are some specifics:

- i) Output of 1st LLM chain is set to ‘output Prediction’.
- ii) Structured output parser is configured as:

JSON Structure <small>• ⓘ</small>			
Property	Type	Description	
answer	boolean	Return true if the input value is... <small>⋮</small>	<small>trash</small>

Figure 49: 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:

Input Variables	
<small>▼ { 1 item</small>	<small>output : "{{llmChain_0.data.instance}}"</small>

Figure 50: 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.

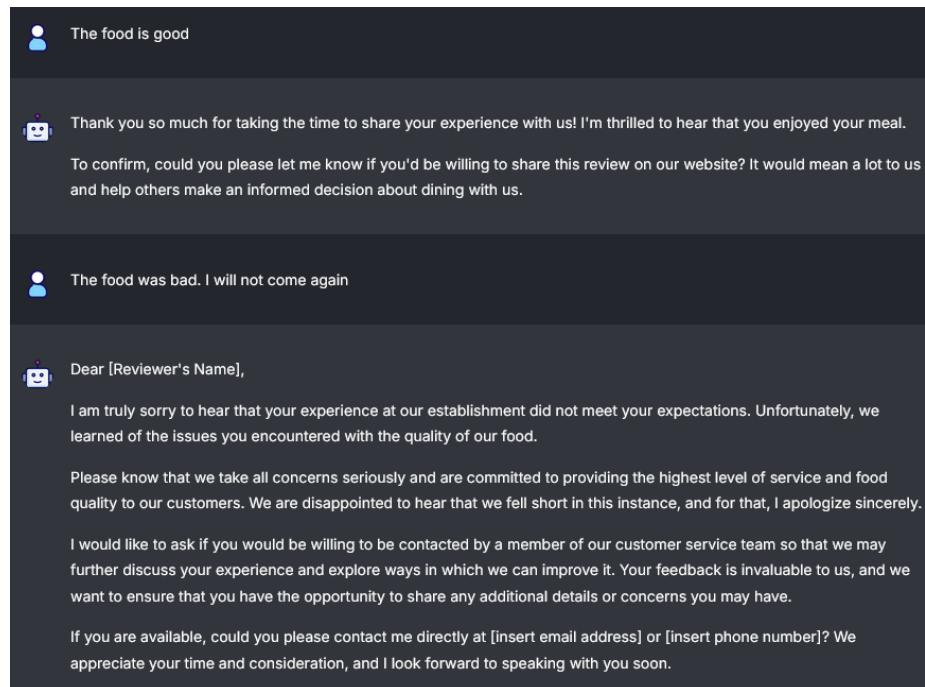


Figure 51: Responses to positive and negative reviews.

U. 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 maintainability¹².

Key Features of TypeScript

1. **Static Typing:** TypeScript's type system helps catch errors at compile time, reducing runtime errors and improving code reliability³.
2. **Enhanced IDE Support:** TypeScript provides excellent integration with popular Integrated Development Environments (IDEs) like Visual Studio Code, offering features like autocompletion, refactoring, and more³.
3. **Better Code Readability and Maintainability:** Type definitions and interfaces make the code more understandable and easier to maintain³.
4. **Improved Developer Productivity:** Features like type inference, advanced type features, and modern JavaScript support (ES6+) streamline development and boost productivity³.

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 not¹.

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 readability³.

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 code¹.

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	John	9996
ABC Inc		
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 52: Invoice file

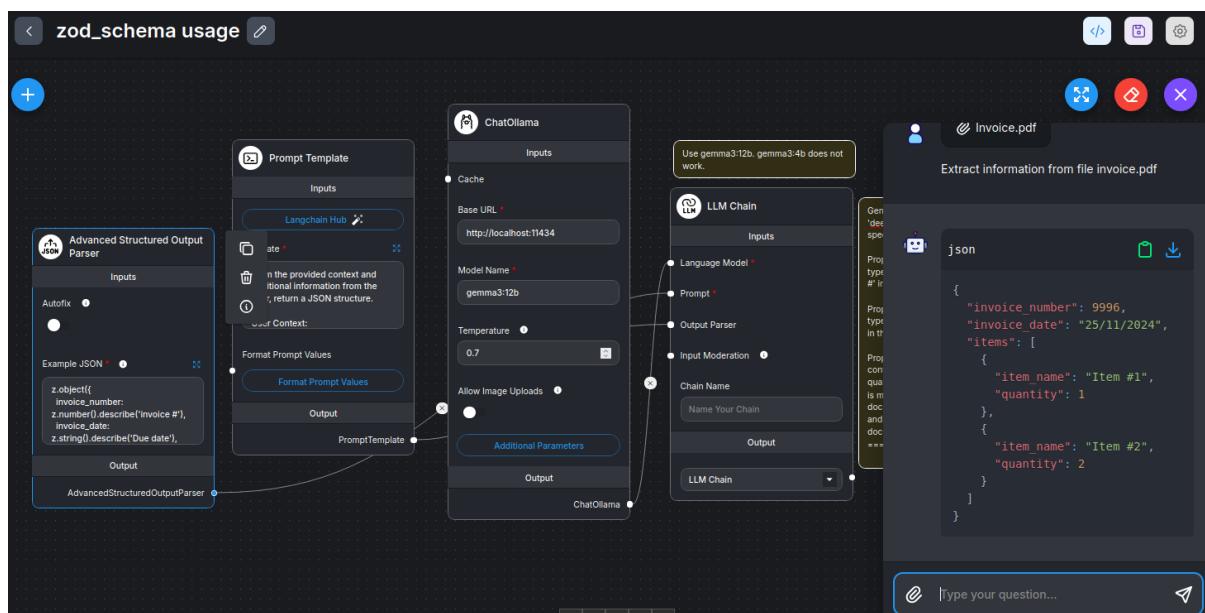


Figure 53: Flowise model. Note the nested schema in the output chat. We use gemma3:12b model as gemma3:4b does not give good results.

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.
```

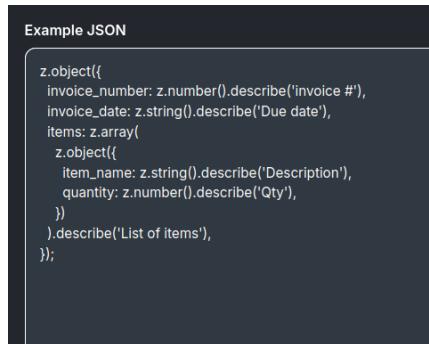


Figure 54: zod schema (also see below) obtained from Deepcoder and pasted in Advanced Structured Parser

```
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
```

V. 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](#).

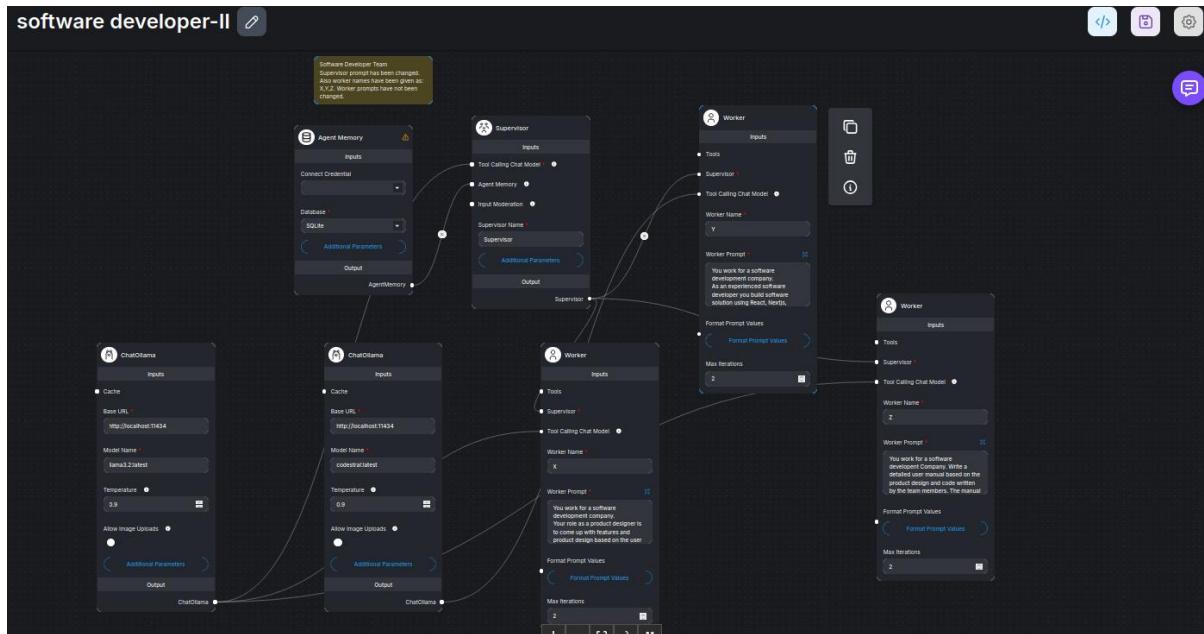


Figure 55: Prompt for the Supervisor has been changed. Worker names have also been changed. The System does not perform as well as the original Software Development Team (see video)

W. Using postgres vector store

Steps involved are:

1. Use the bash command: `./createpostgresuser.sh` to create a user (also called Role), its password and a database.
2. Transform the created database to vector store using the bash command: `./createvectordb.sh`
3. Then create credentials and begin upserting vectors into this database in flowise.

```

Enter SQL command as in example below to create a user LOGIN id
and his password, mypass (password be within single inverted commas)

Example: CREATE ROLE myuser LOGIN PASSWORD 'mypass' ;

Create database as:
    CREATE DATABASE mydatabase WITH OWNER = myuser;

To quit psql shell enter \q

[sudo] password for ashok:
could not change directory to "/home/ashok": Permission denied
psql (14.15 (Ubuntu 14.15-0ubuntu0.22.04.1))
Type "help" for help.

postgres=# CREATE ROLE abhinav LOGIN PASSWORD 'abhinav' ;
CREATE ROLE
postgres=# CREATE DATABASE abhinav WITH OWNER = abhinav;
CREATE DATABASE
postgres=# \c abhinav
You are now connected to database "abhinav" as user "postgres".
abhinav=# CREATE EXTENSION vector;
CREATE EXTENSION
abhinav=#

```

Figure 56: Execute command: ./createpostgresuser.sh. Create a user, password, database and configure it, as below, to store vectors

```

postgres=# CREATE ROLE myuser LOGIN PASSWORD 'mypass' ;
CREATE ROLE
postgres=# CREATE DATABASE mydatabase WITH OWNER = myuser;
CREATE DATABASE
postgres=#

```

Figure 57: A user, his password and database created

```

ashok@ashok:~$ ./createvectordb.sh

Ref: https://github.com/pgvector/pgvector

=====
Add vector storage capability to an existing pg database
Will open psql shell to execute two commands.
=====

To create a user and a database first execute script:
    cd ~/ ; ./createpostgresuser.sh
Next, connect to your database, as:
(This is akin to 'use db' command in mysql)
    \c <databaseName>
Once connected, issue the following SQL command:
    CREATE EXTENSION vector;

Done....Create database as:

To quit psql shell enter \q

[sudo] password for ashok:
could not change directory to "/home/ashok": Permission denied
psql (14.18 (Ubuntu 14.18-0ubuntu0.22.04.1))
Type "help" for help.

postgres=# \c mydatabase
You are now connected to database "mydatabase" as user "postgres".
mydatabase=# create extension vector ;
CREATE EXTENSION
mydatabase=#

```

Figure 58: Database 'mydatabase' transformed into a vector store

CREDENTIAL NAME *

User *

Password *

Add

Figure 59: In Document Vector Store in Flowise, creating credential for postgres database

Embeddings

Vector Store

Record Manager

Ollama Embeddings

Postgres

Select Record Manager

Host * 127.0.0.1

Database * mydatabase

Port 5432

Figure 60: Filling up postgres related parameters

X. 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.

The screenshot shows the Flowise application interface. On the left, there's a sidebar with various options like Chatflows, Agentflows, Executions, Assistants, Marketplaces, Tools, Credentials, Variables, API Keys, and Document Stores (which is currently selected). The main area has three main sections: 'Ollama Embeddings', 'Postgres', and 'Postgres Record Manager'. The 'Ollama Embeddings' section includes fields for 'Base URL' (set to 'http://localhost:11434'), 'Model Name' (set to 'nomic-embed-text'), 'Number of GPU' (set to 1), 'Number of Thread' (set to 1), and 'Use MMap' (which is turned on). The 'Postgres' section includes fields for 'Connect Credential' (set to 'test'), 'Host' (set to 'localhost'), 'Database' (set to 'mydatabase'), 'Port' (set to 5432), and 'SSL' (which is turned off). The 'Postgres Record Manager' section includes fields for 'Connect Credential' (set to 'test'), 'Host' (set to 'localhost'), 'Database' (set to 'mydatabase'), 'Port' (set to 5432), and 'SSL' (which is turned off). There are also dropdowns for 'Table Name' (set to 'documents') and 'Distance Strategy' (set to 'Cosine'). At the bottom right, it says 'Additional Connection Configuration' with a note '0 items'.

Figure 61: Document vector store with postgres vector store and postgres record manager.

Y. 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:

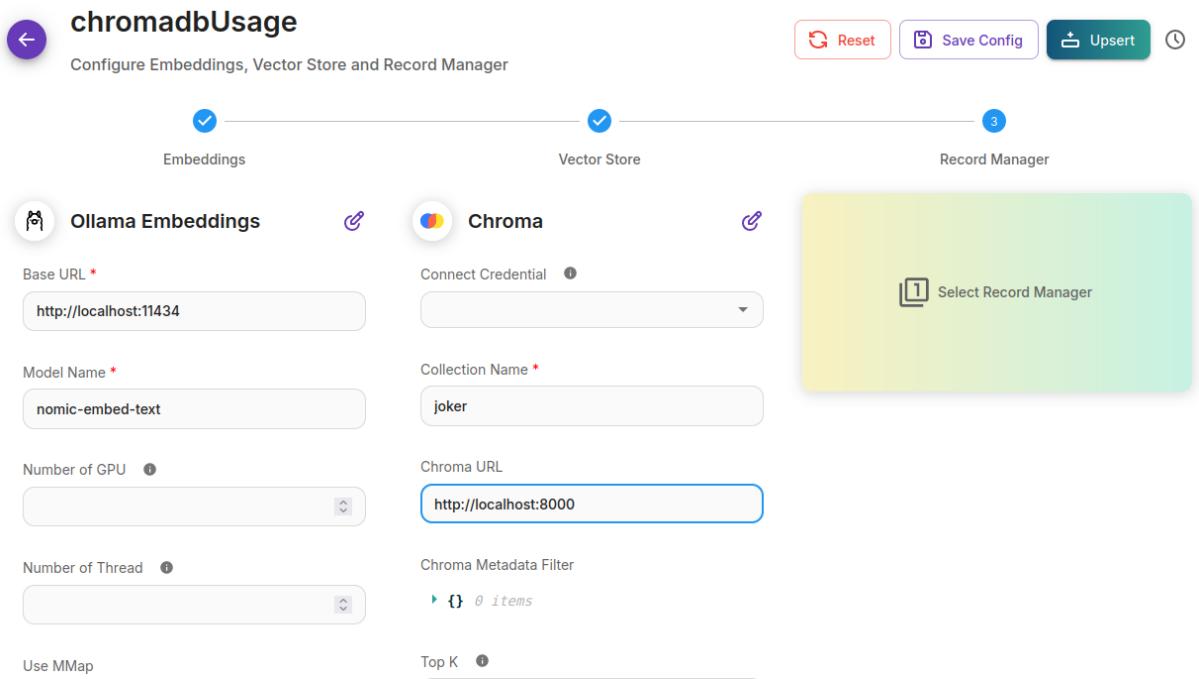


Figure 62: Configuring chromadb docker in flowise Document vector store

Z. 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:

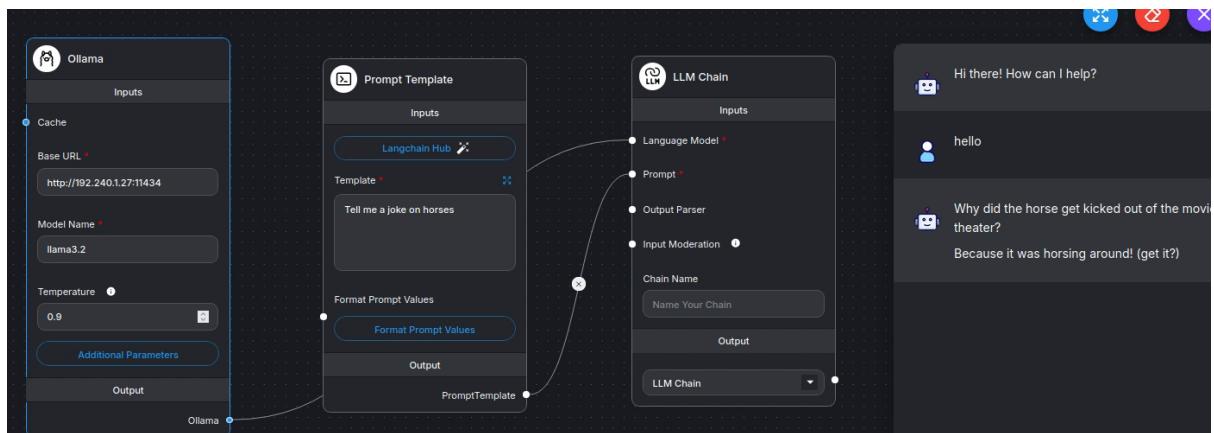


Figure 63: Ollama and flowise both on docker. Access ollama as: <http://192.240.1.27:11434> rather than <http://localhost:11434>.

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.

```

ashok@ashok:~$ ifconfig
docker0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 172.17.0.1 netmask 255.255.0.0 broadcast 172.17.255.255
        inet6 fe80::a820:c0ff:feb3:5430 prefixlen 64 scopeid 0x20<link>
          ether aa:20:c0:b3:54:30 txqueuelen 0 (Ethernet)
            RX packets 926139 bytes 60244082 (60.2 MB)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 1264631 bytes 3279099213 (3.2 GB)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 172.27.224.244 netmask 255.255.240.0 broadcast 172.27.239.255
        inet6 fe80::215:5dff:fee6:5a2b prefixlen 64 scopeid 0x20<link>
          ether 00:15:5d:e6:5a:2b txqueuelen 1000 (Ethernet)
            RX packets 3112053 bytes 4683668336 (4.6 GB)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 1157266 bytes 78425564 (78.4 MB)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
        inet 127.0.0.1 netmask 255.0.0.0
        inet6 ::1 prefixlen 128 scopeid 0x10<host>
          loop txqueuelen 1000 (Local Loopback)
            RX packets 18947 bytes 12337787 (12.3 MB)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 18947 bytes 12337787 (12.3 MB)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

veth0570142: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet6 fe80::646a:98ff:fe46:8cf9 prefixlen 64 scopeid 0x20<link>
          ether 66:6a:98:46:8c:f9 txqueuelen 0 (Ethernet)
            RX packets 1077 bytes 8833002 (8.8 MB)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 866 bytes 1112798 (1.1 MB)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

veth60f6d09: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet6 fe80::a807:beff:fea4:dc9 prefixlen 64 scopeid 0x20<link>
          ether aa:07:be:a4:0d:c9 txqueuelen 0 (Ethernet)
            RX packets 745117 bytes 51106857 (51.1 MB)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 971001 bytes 2361081605 (2.3 GB)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

```

Figure 64: 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:

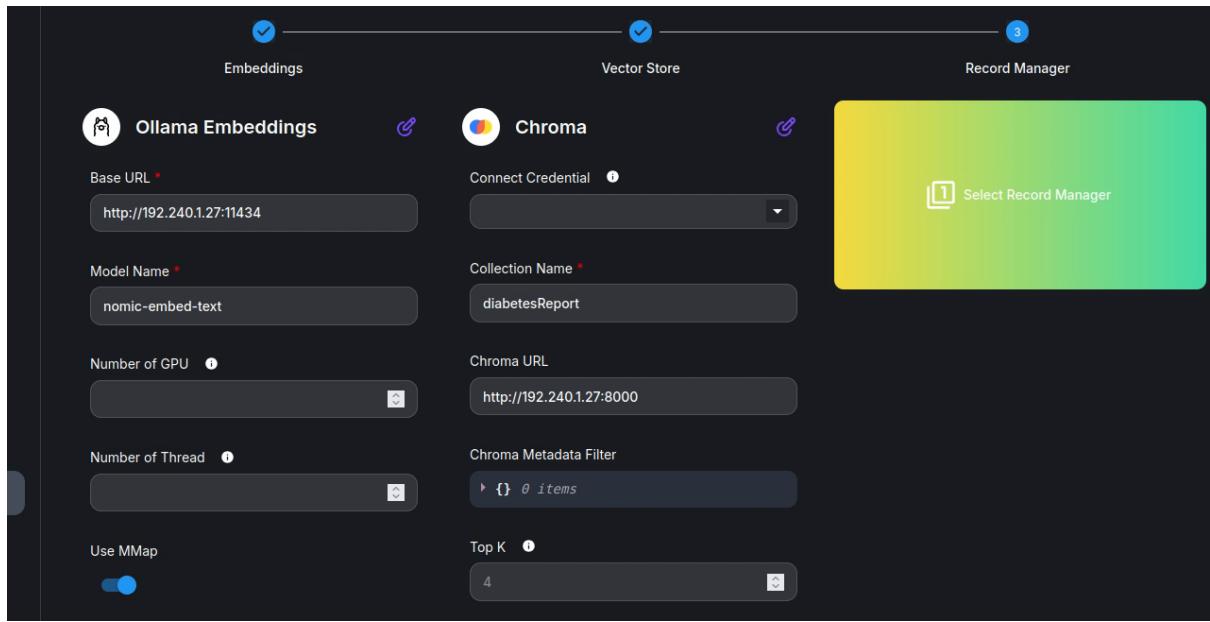


Figure 65: Use ip instead of localhost, wherever needed.

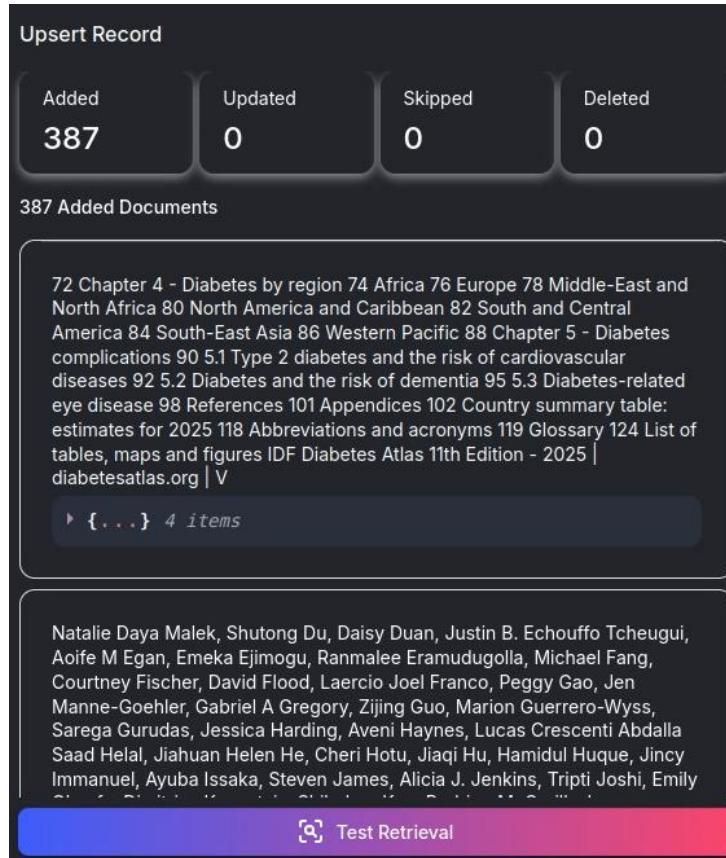


Figure 66: uPSERT TO CHROMADB SUCCEEDS.

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 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.
