

Design bots using Flowise

Last amended: 16th June, 2025

My folder: C:\Users\ashok\OneDrive\Documents\flowise

Flowise Book created by Community [at this link](#)

Vector stores working file: FAISS; Milvus, Milisearch But NOT chroma and Qdrant

Table of Contents

A. Simple demo	3
B. Translation bot:	3
C. Chat with llama:	4
D. Simple Conversational Chain	5
E. Using Conversational Agents	5
F. Export import chat flows:	6
i) Export chatflow	6
ii) Load chat flow	6
G. Simple RAG with single text file	7
H. RAG with chroma store and single text file	8
I. Prompt Chaining:.....	9
i) Prompt Chaining-I	9
ii) Prompt Chaining-II	11
J. Flowise Using Hugging Face Models.....	12
K. Document Stores-How to Upsert	12
L. Using Redis Backed Chat Memory.....	13
M. Langsmith for debugging.....	16
N. LLM Chains vs Conversational Agent vs Conversational Retrieval Agent vs Tool Agent.....	18
O. Example System message for Summarization	19
P. Multi-Prompt Retriever	20
Q. Multi-Retriever chatflow	21
i) With MultiRetriever node	21
ii) With Tool Agent node	22
R. Using Milisearch vector store.....	23

A. Simple demo

See [this link](#) on YouTube.

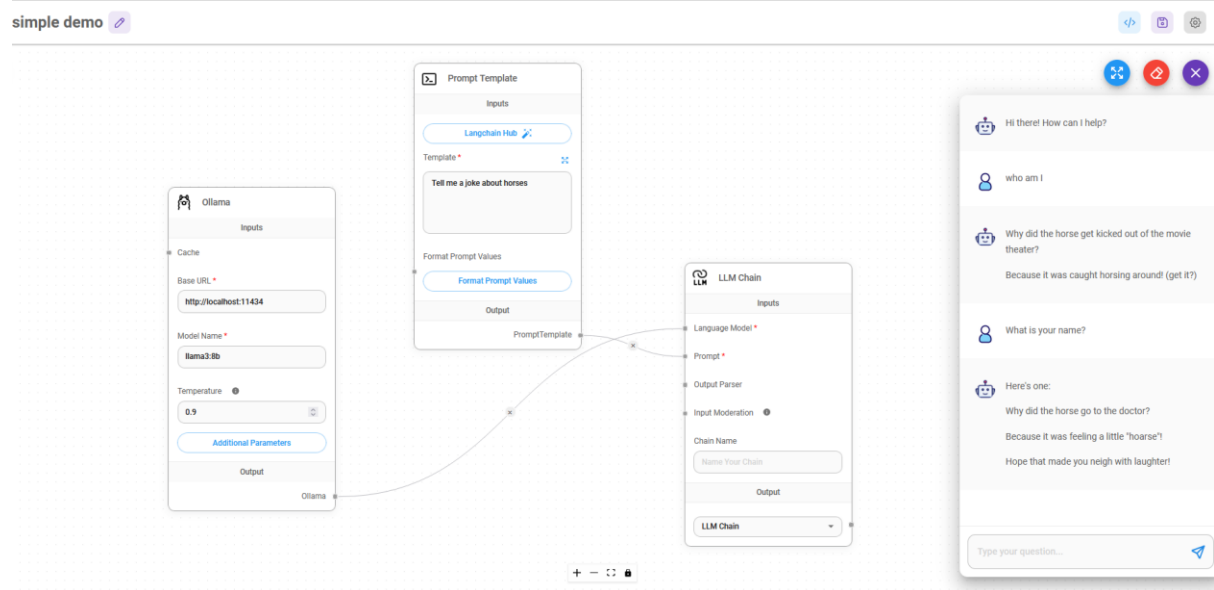


Figure 1: This bot will always answer your questions as a horse's joke. The only prompt is: **Tell me a joke about horses.**

B. Translation bot:

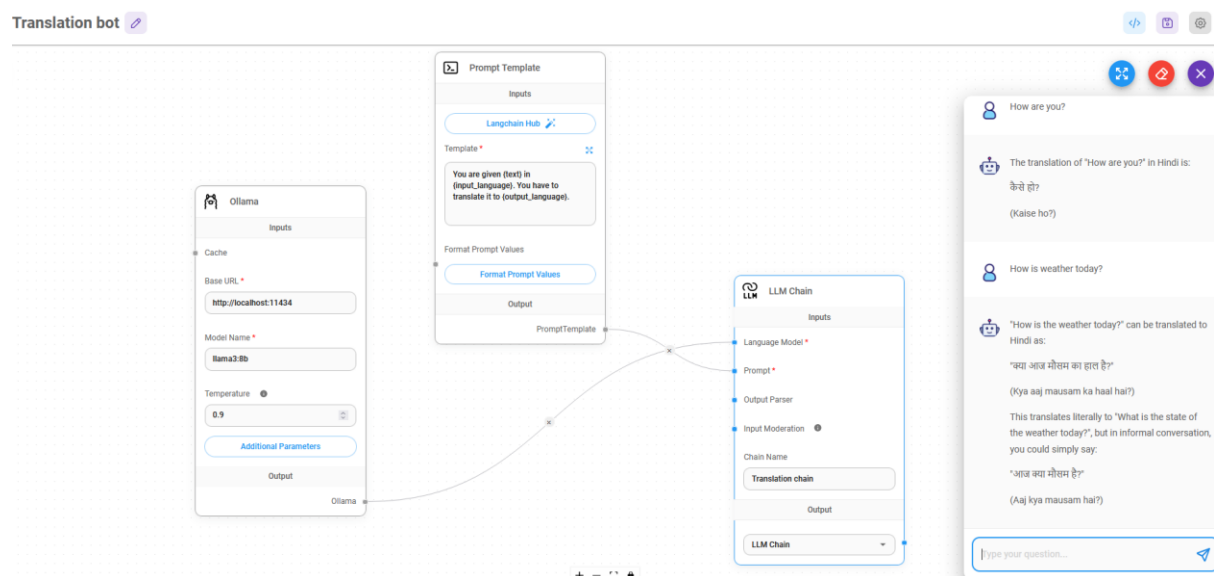


Figure 2: This bot translates all your questions into the desired language.

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.

C. Chat with llama:

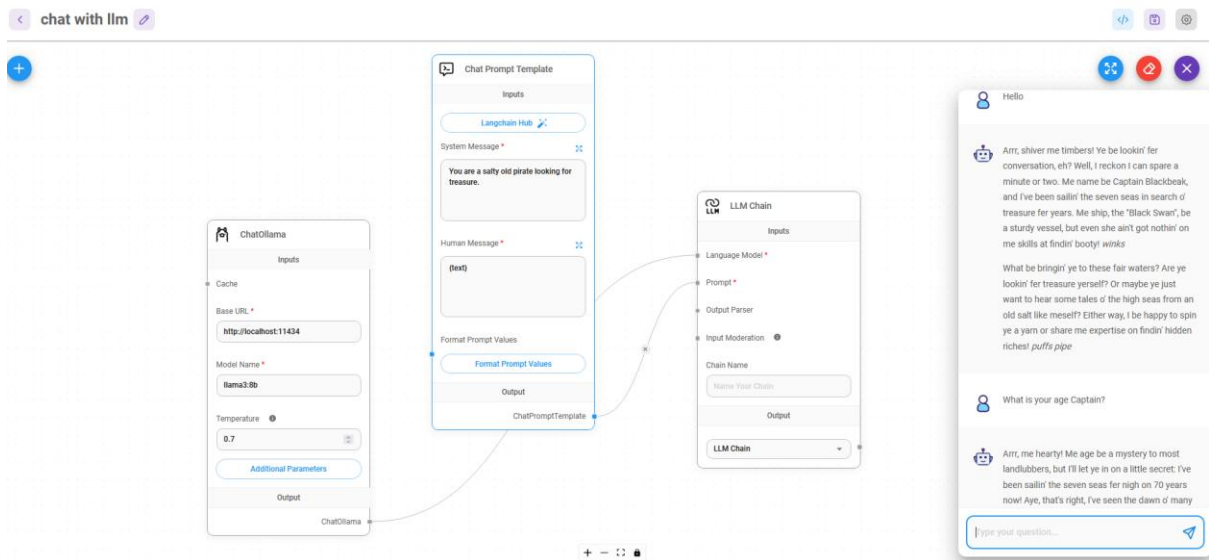
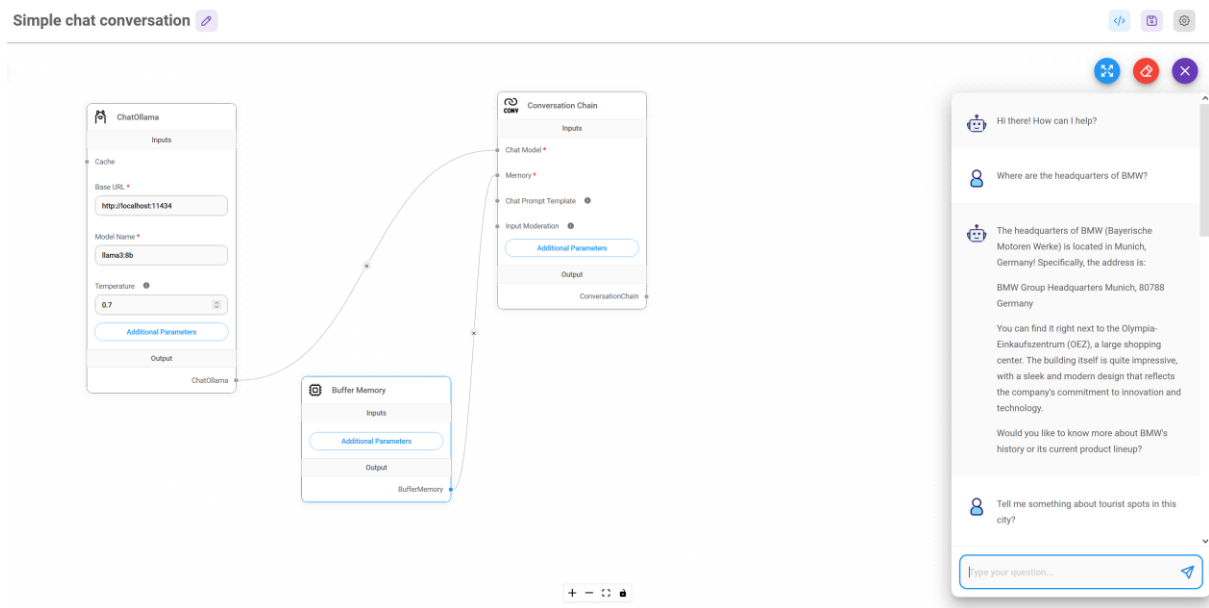


Figure 4: The 1st question is just **Hello** but the 11nd question asks more details about **Captain** referred to into the answer to **Hello**.

D. Simple Conversational Chain

Refer [YouTube video](#)



E. Using Conversational Agents

Refer [YouTube video](#)

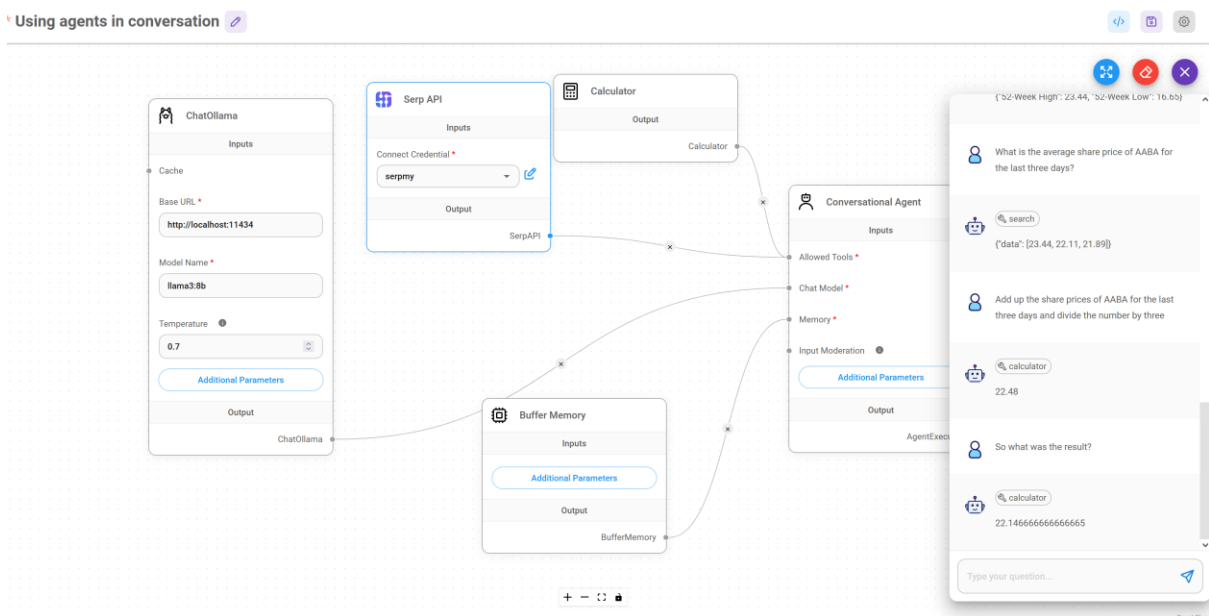


Figure 5: Agents can work even with ollama. SERP API key is a must. To calculate average, we have to tell the bot how to do it.

F. Export import chat flows:

i) Export chatflow

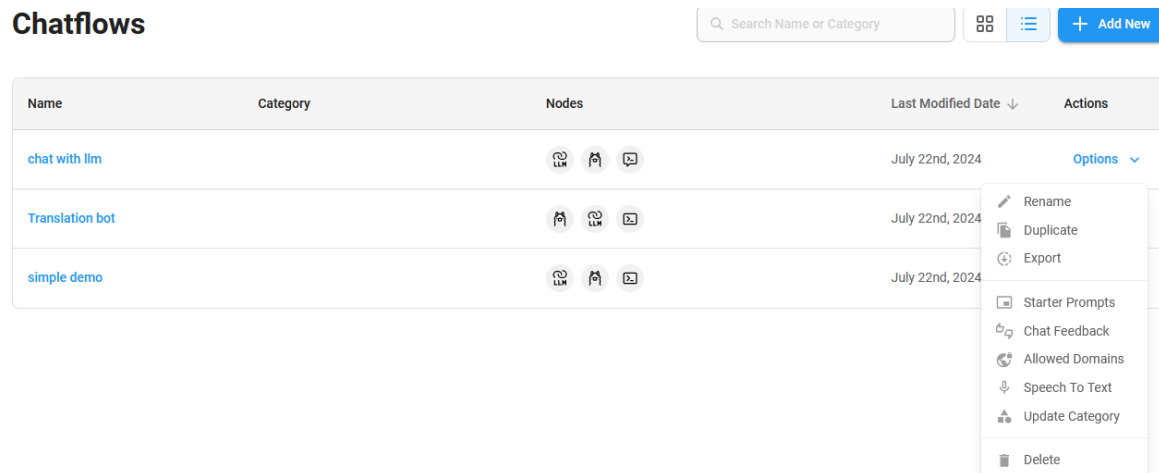


Figure 6: In the Chat flows window, click on the down arrow besides the **Options** to Export a chat flow as a json file.

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.

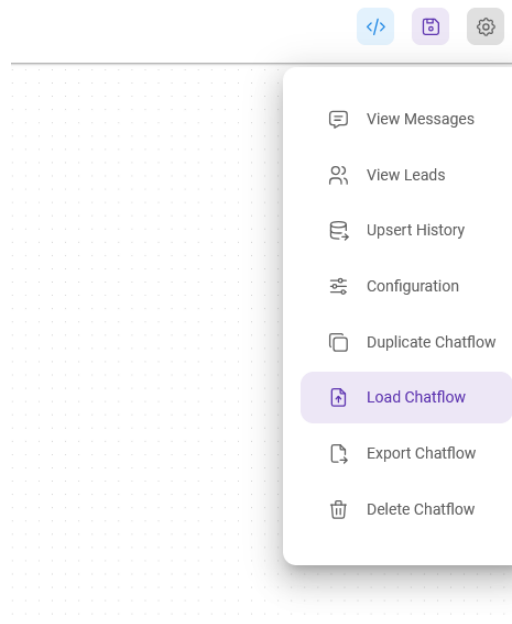


Figure 7: Click on Settings icon to import an exported chat flow (i.e. json file).

The following figure shows a chatflow loaded in Flowise:

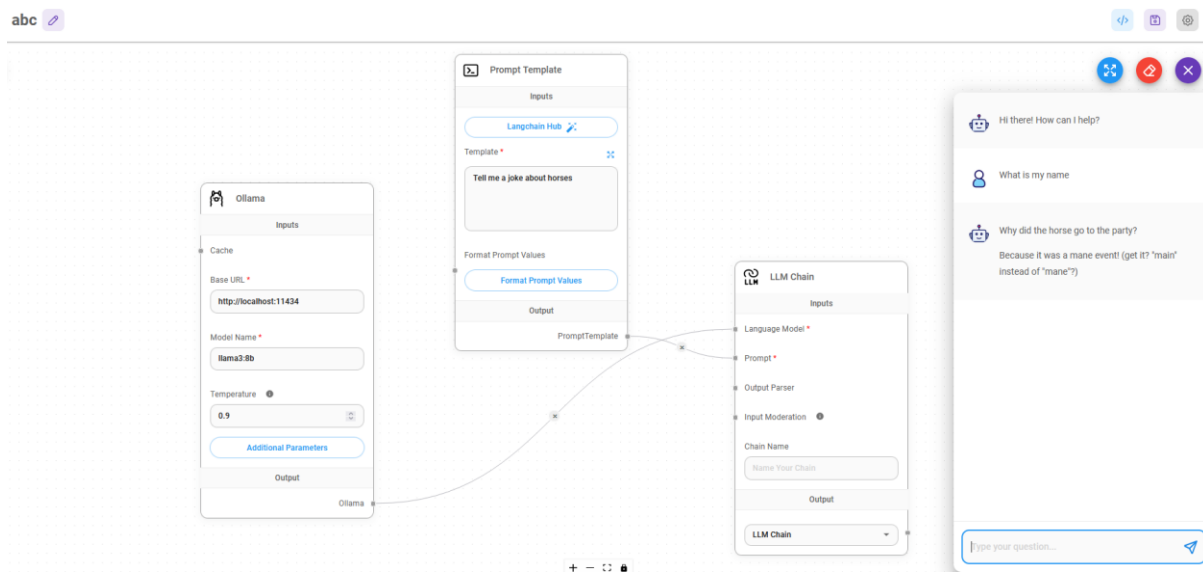


Figure 8: A chatflow loaded in a blank 'abc' chatflow canvas.

G. Simple RAG with single text file

Refer [Flowise tutorial #3](#)

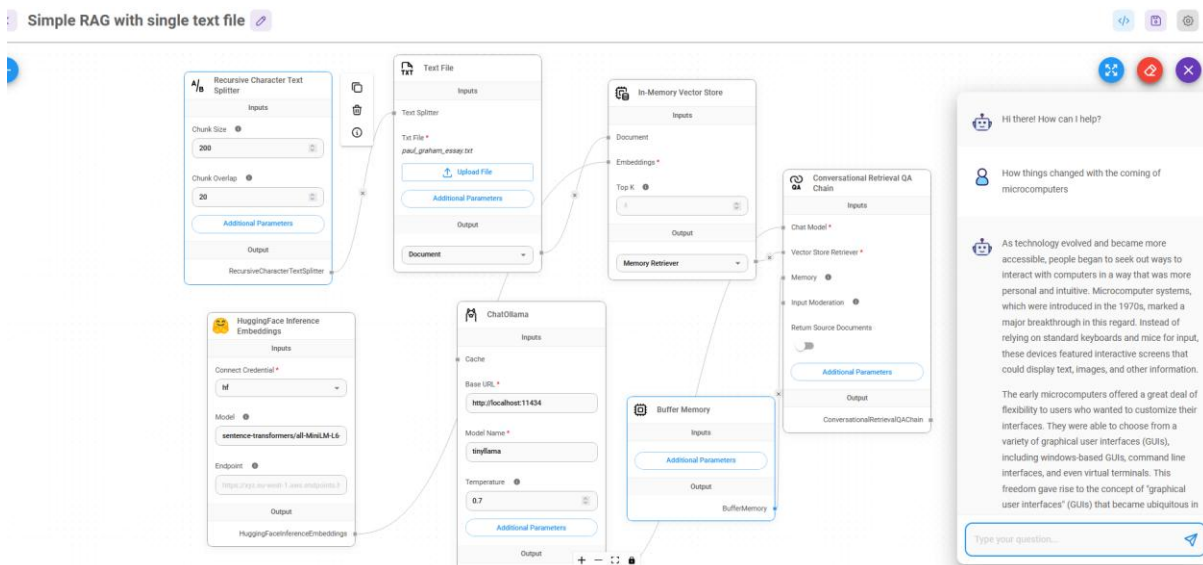


Figure 9: After connecting all flow-widgets and uploading of text file, first click on Upsert Vectors button and then start chatting.

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

H. RAG with chroma store and single text file

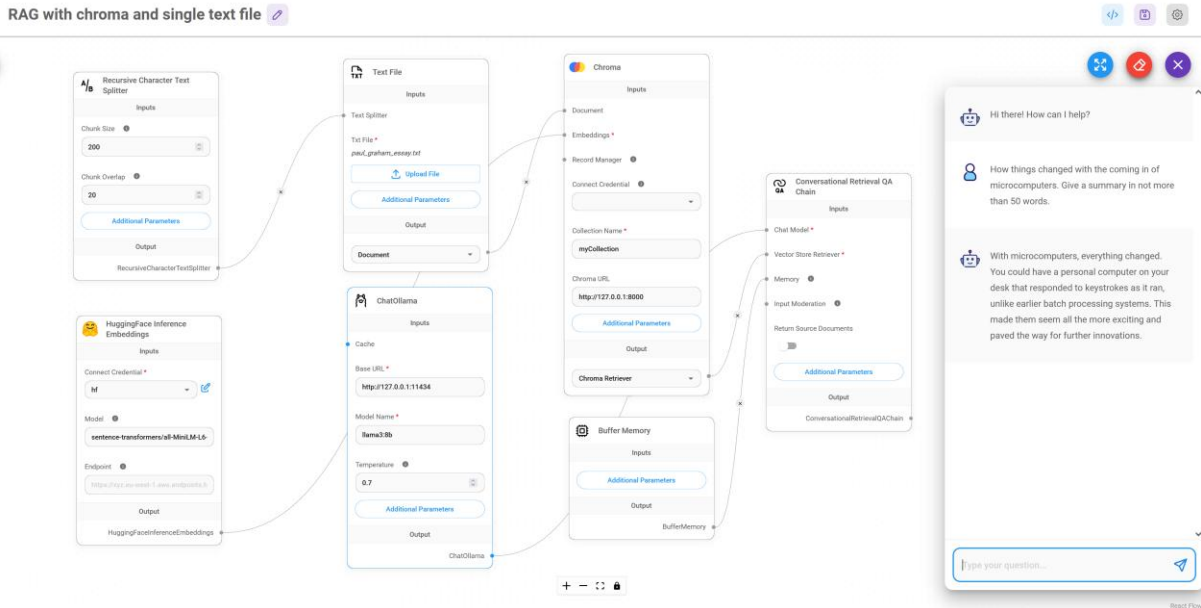


Figure 10: Vector store is replaced by a more durable chroma store. Chroma store retains its vectors even after Flowise is hut down.

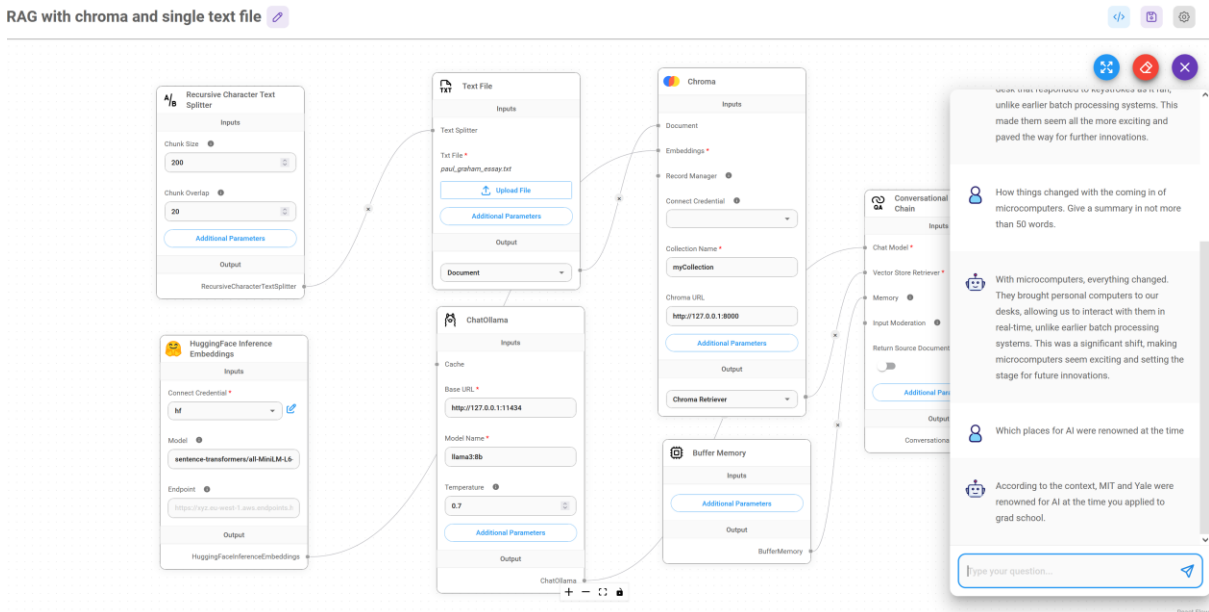


Figure 11: Flowise restarted. More questions asked and replies are given based upon the earlier storage.

I. Prompt Chaining:

i) Prompt Chaining-I

Combining Multiple LLM Chains

A. Refer [Flowise tutorial](#)

First LLM chain

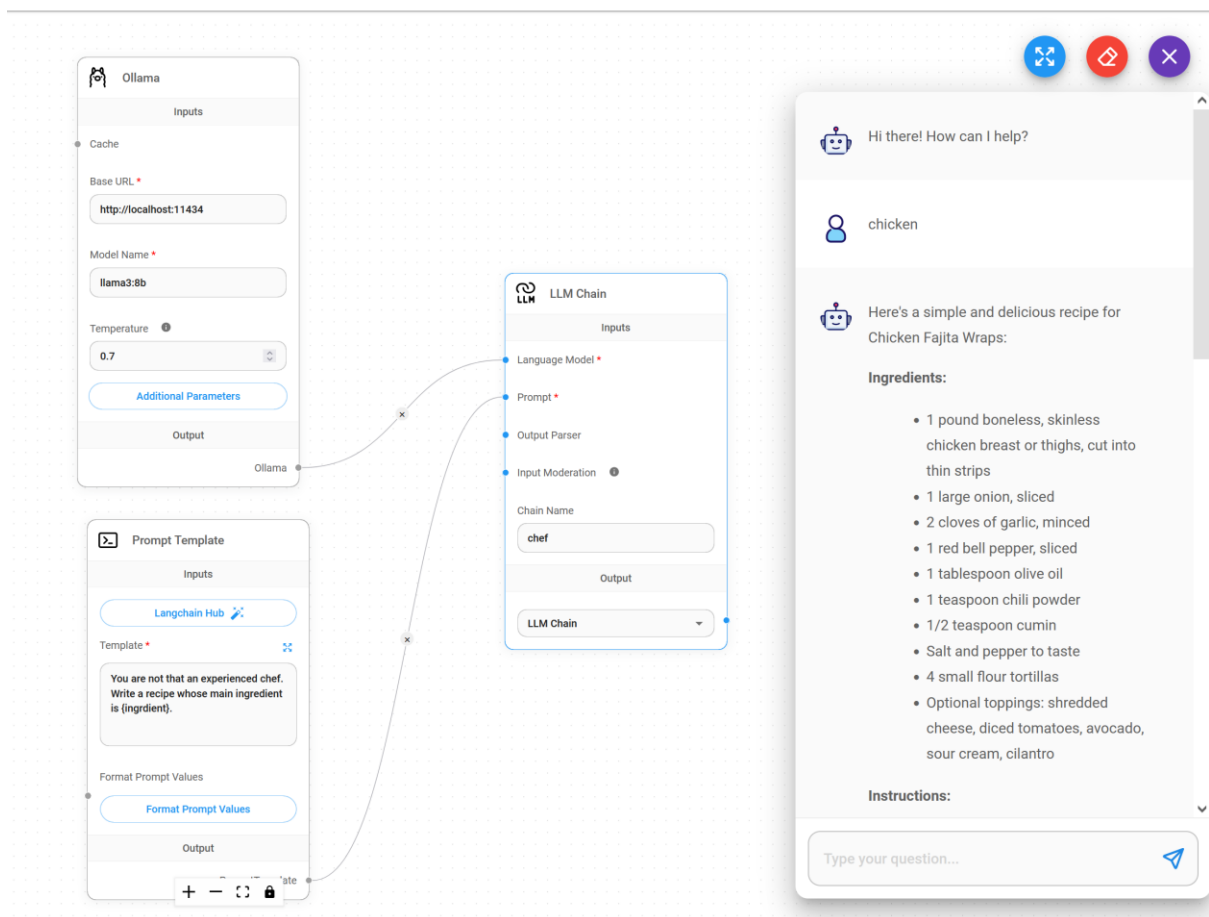


Figure 12: First LLM chain named as chef.

Second LLM chain added to first

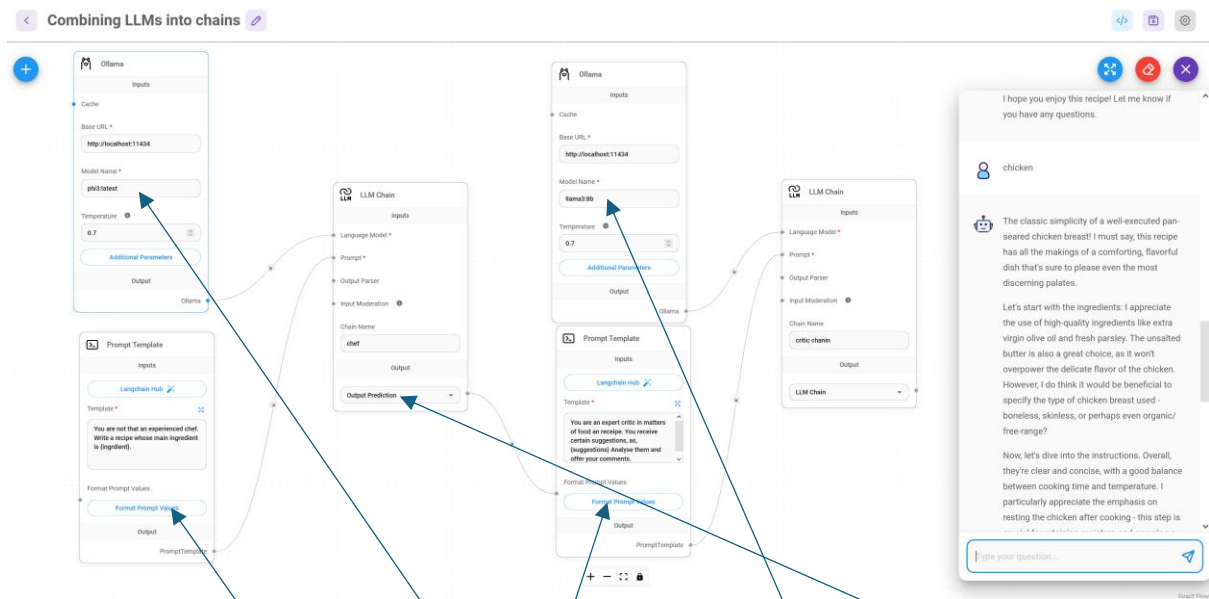


Figure 13: Note that the first LLM is using **phi3** and the critic language model is **llama3**. In the chatbox, the **llm** chain does make minor suggestions to improve the quality. Note that the output of 1st LLM chain is now **Output Prediction**.

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
{ 1 item
  ingredient: "{{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
{ 1 item
  suggestions : "{{llmChain_0.data.instance}}"
}
```

ii) Prompt Chaining-II

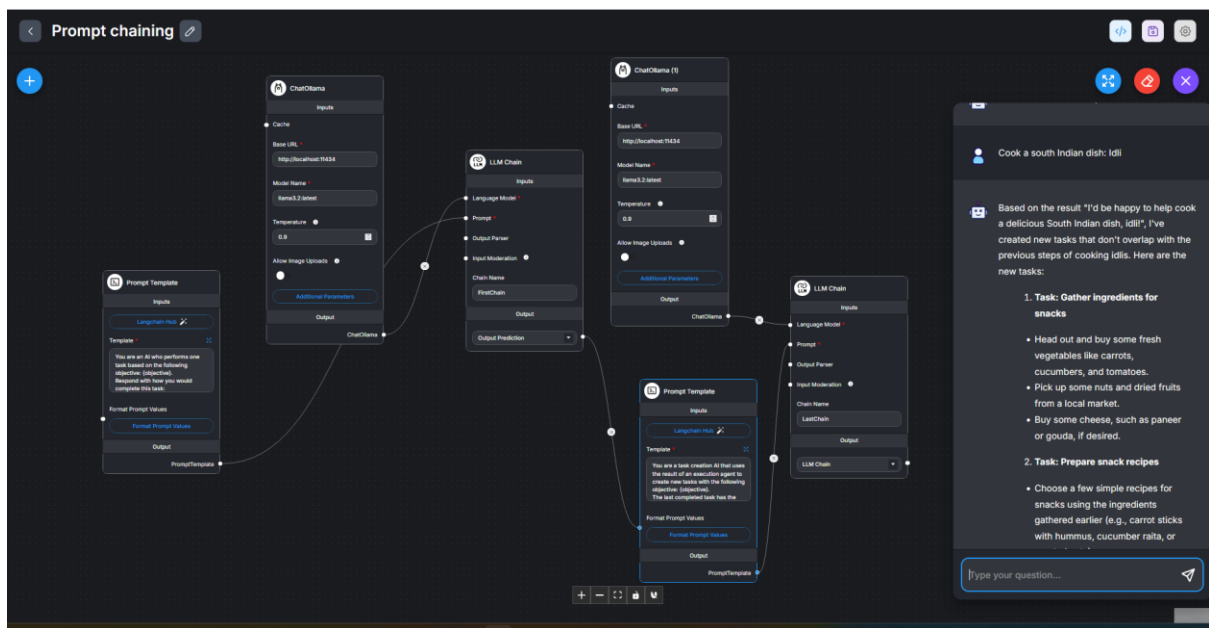


Figure 14: Contains two (simple) Prompt Templates. See below what these prompts look like.

Prompt1:

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

Respond with how you would complete this task:



Figure 15: 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 16: 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

J. Flowise Using Hugging Face Models

Ref: [This link](#).

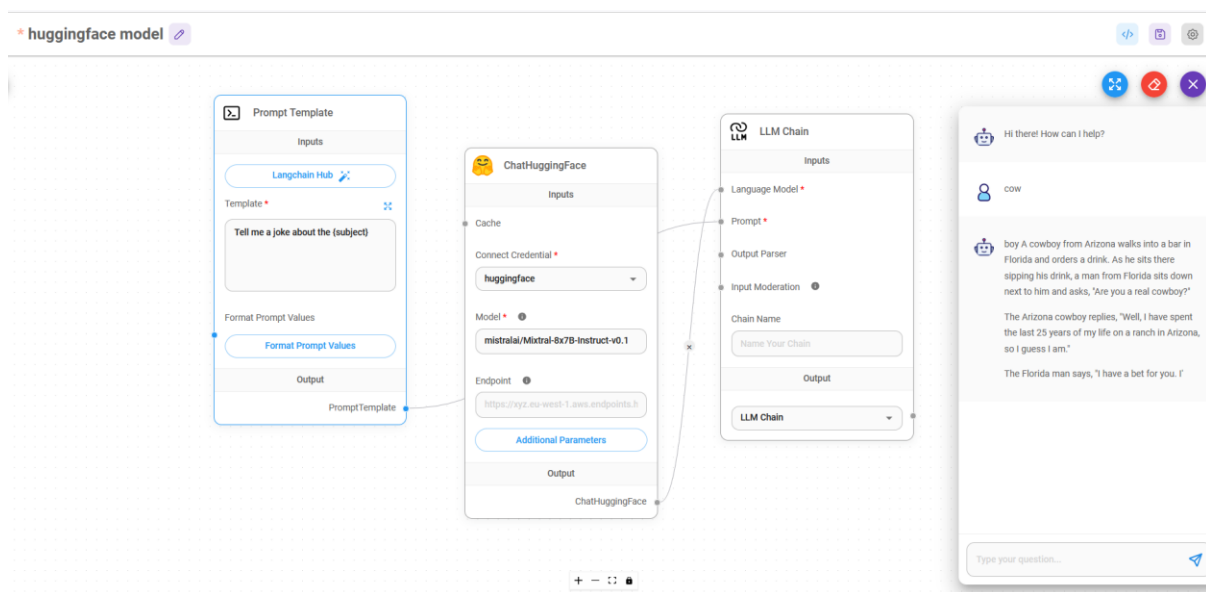


Figure 17: Only those models will work who have made available inference endpoints free.

K. Document Stores-How to Upsert

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

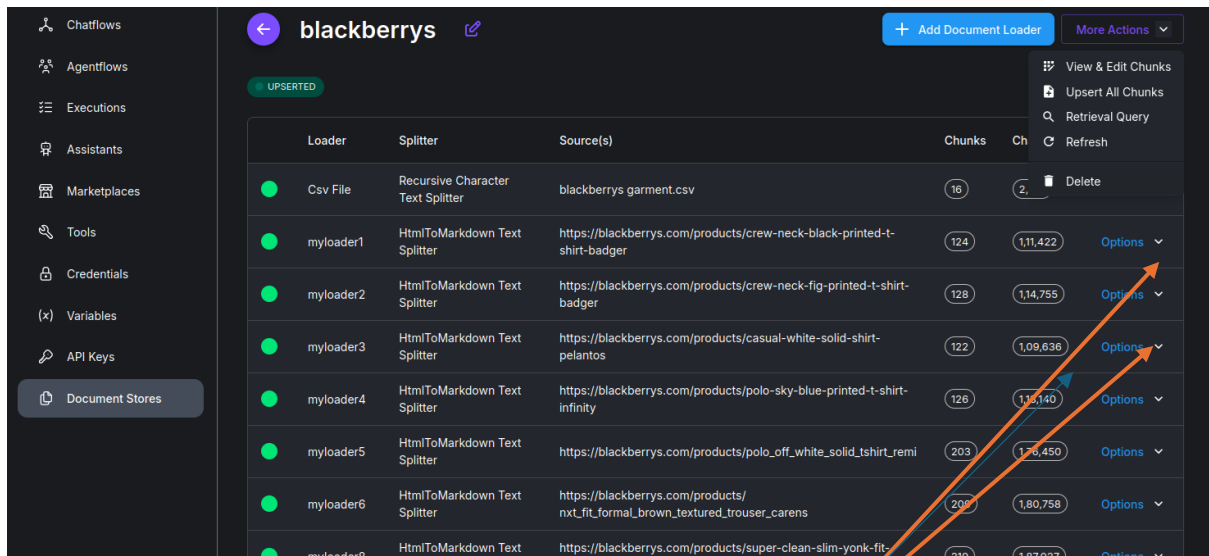


Figure 18: Many Document Loaders are here.

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:

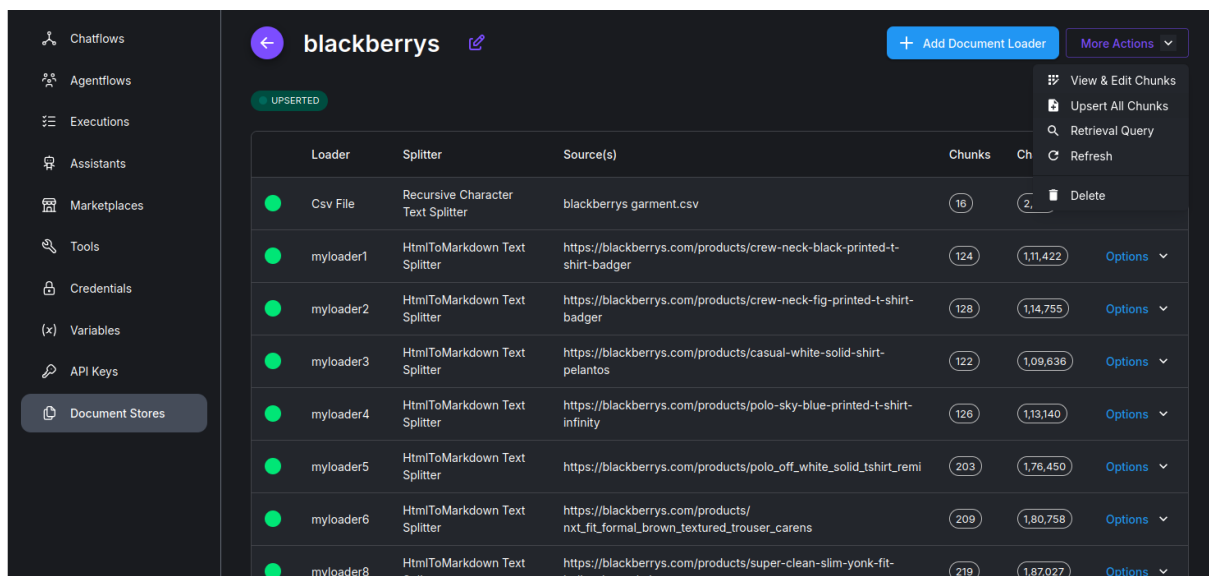


Figure 19: Look for More Actions-->Upsert All chunks. This will upsert chunks from ALL Document Loaders.

L. Using Redis Backed Chat Memory

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

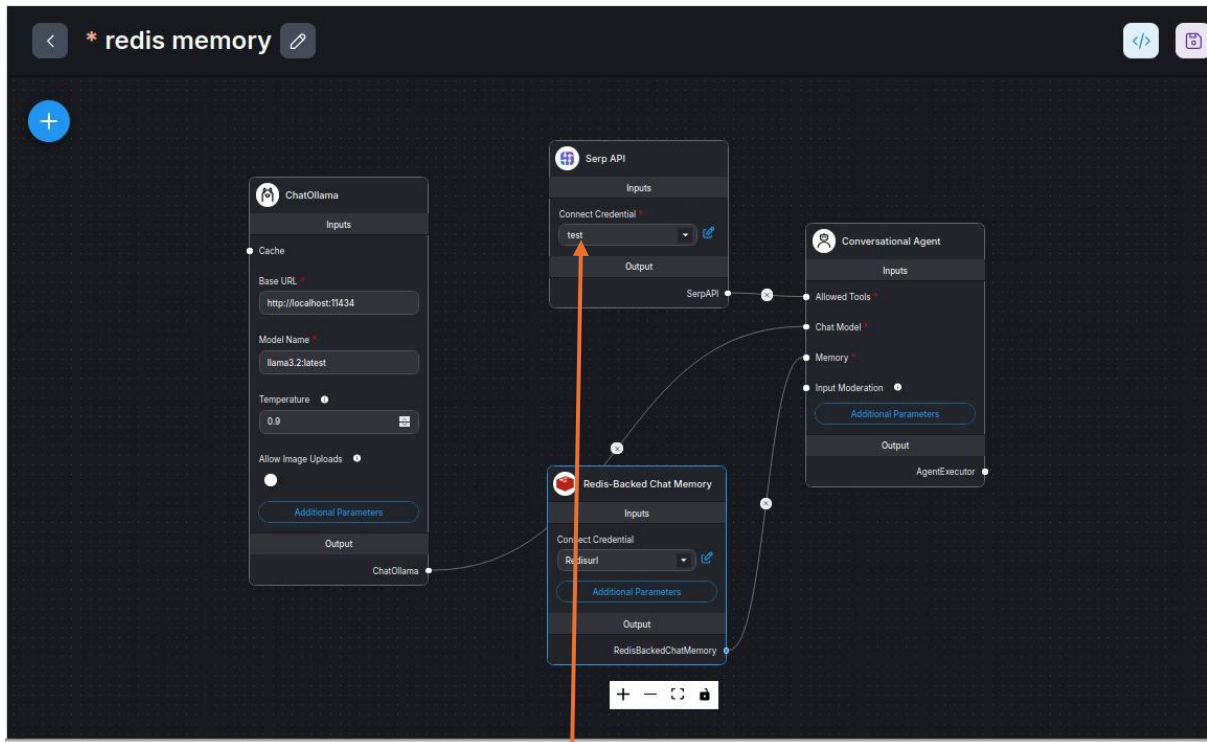


Figure 20: A conversational agent that utilises Redis backed Chat Memory

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

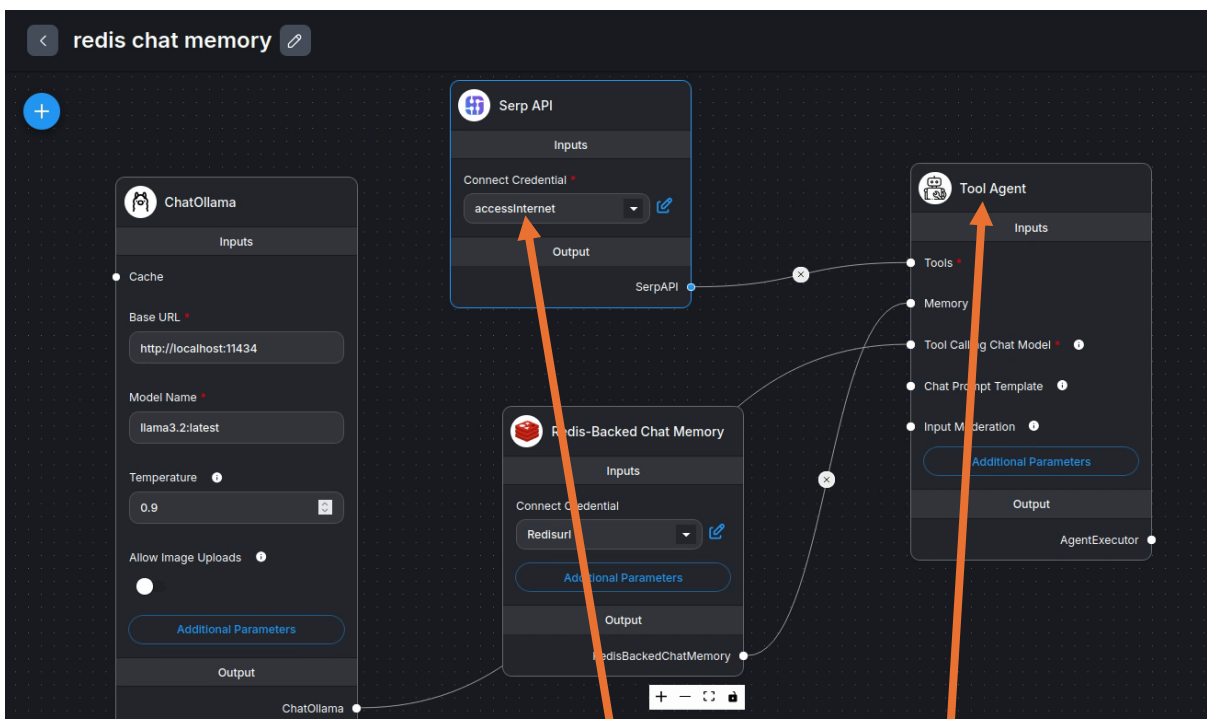


Figure 21: *Serp API* now has a better tool credential name: *accessInternet*. Incidentally, it uses *Tool Agent* rather than *Conversational agent*.

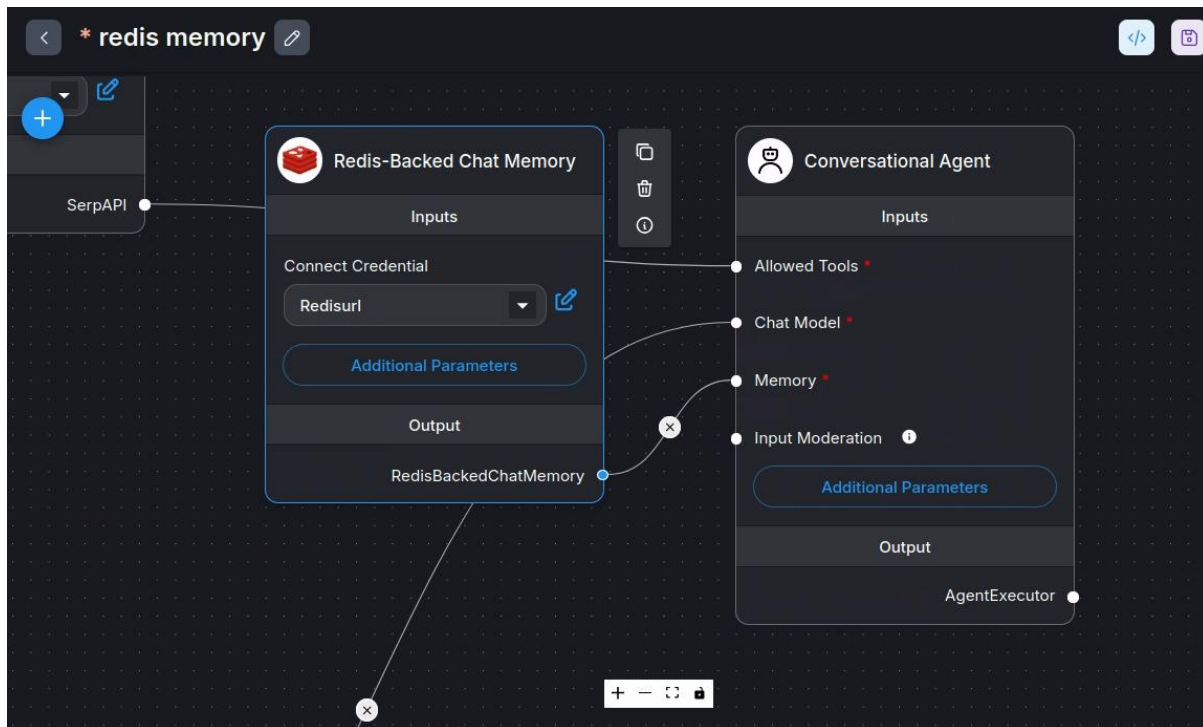


Figure 22: Redis backed chat memory module attached to Conversational agent

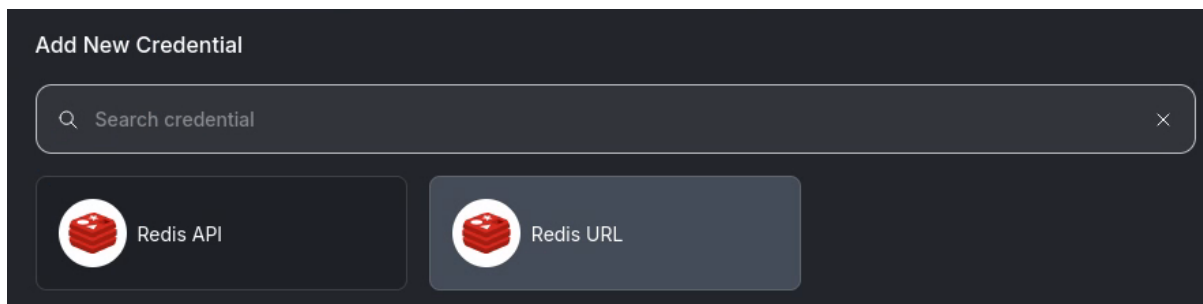


Figure 23: Start Redis server: `./start_redis.sh`.

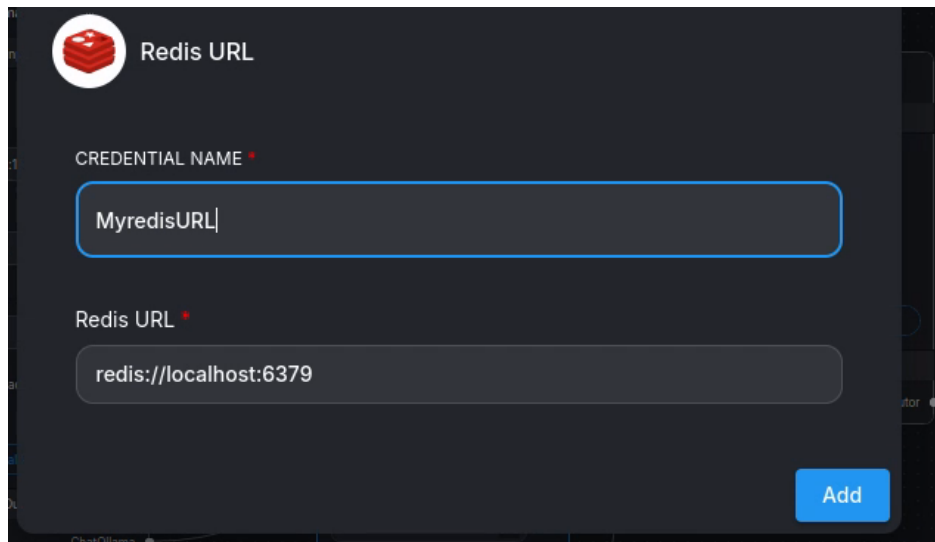


Figure 24: Specifying Redis Credential

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

M. Langsmith for debugging

Langsmith can be used for tracing agent flows. Log into [langsmith](https://smith.langchain.com/) and get a free API key. Create any simple agentic project, such as the following simple project:

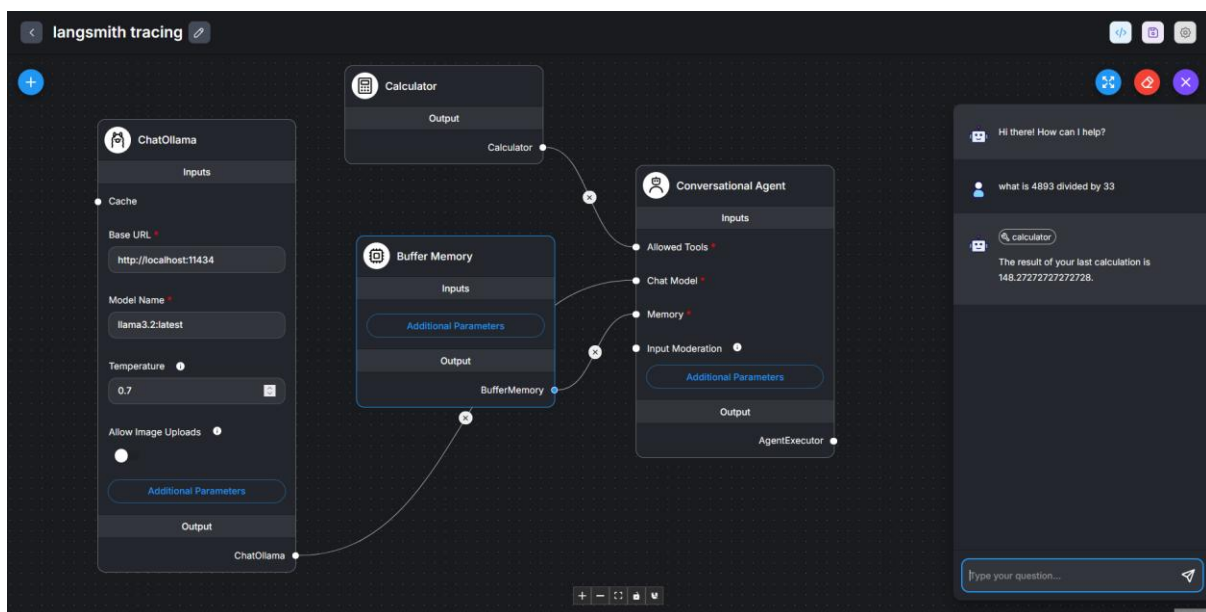


Figure 25: Click on the gear icon at top-right and then Configuration

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

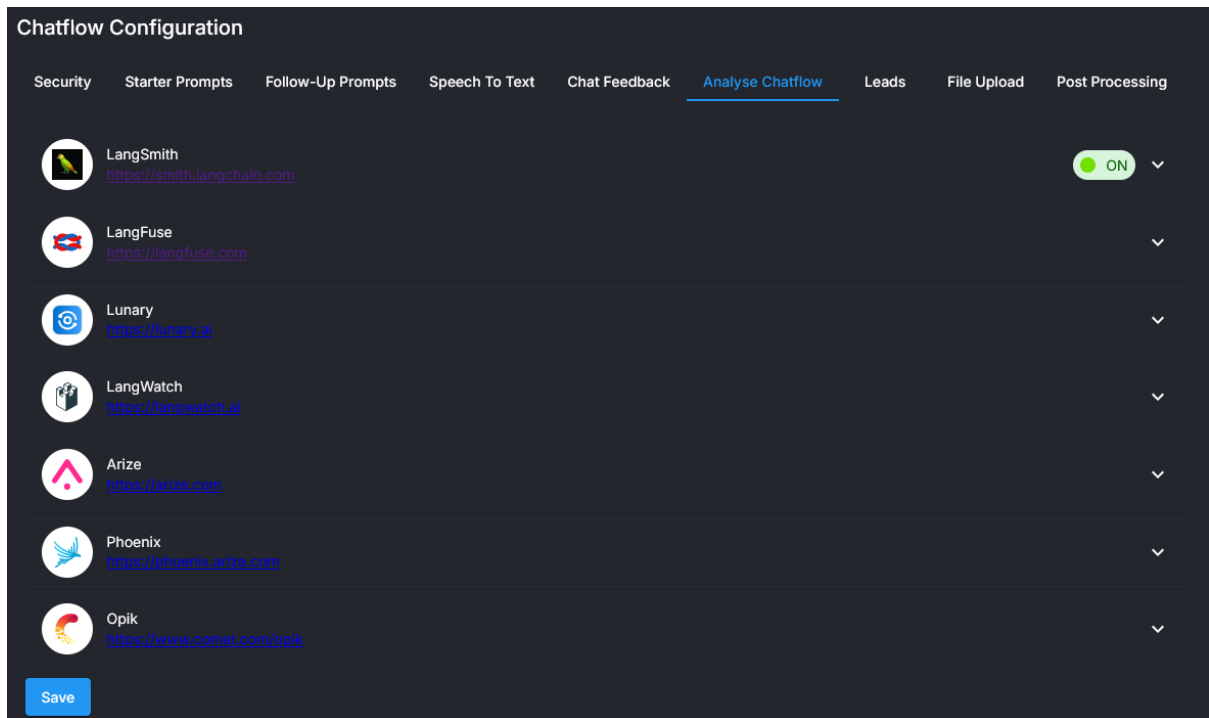


Figure 26: Click on Analyse Projects → Langsmith and create new Credential as also switch its usage on. Then Save the configuration.

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

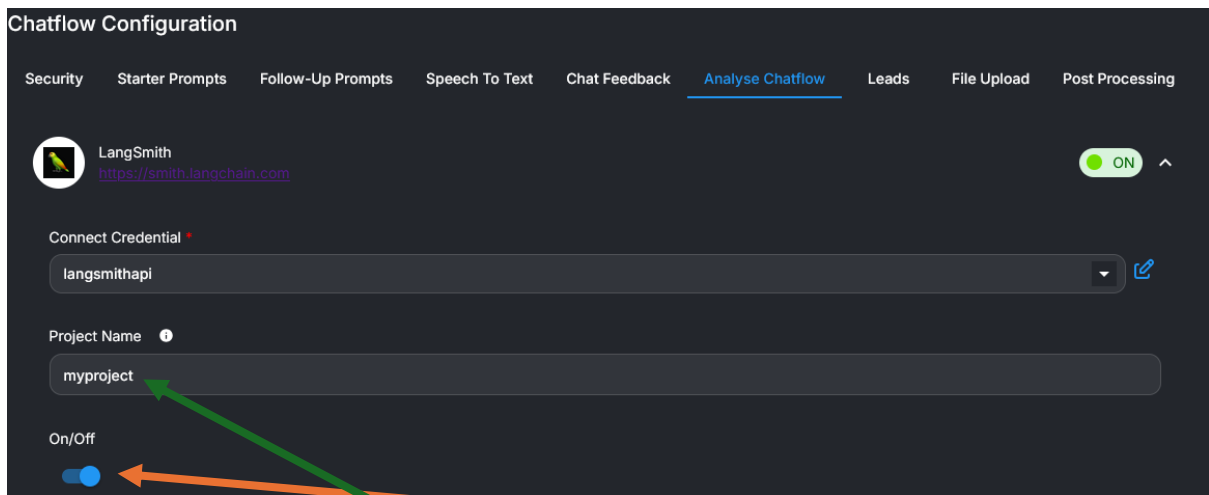


Figure 27: Establish credentials, write a project name and switch langsmith usage on. Save the configuration

In [langsmith website](https://smith.langchain.com) under *Personal* → *Tracing Projects* → *myproject* see traces of what happened.

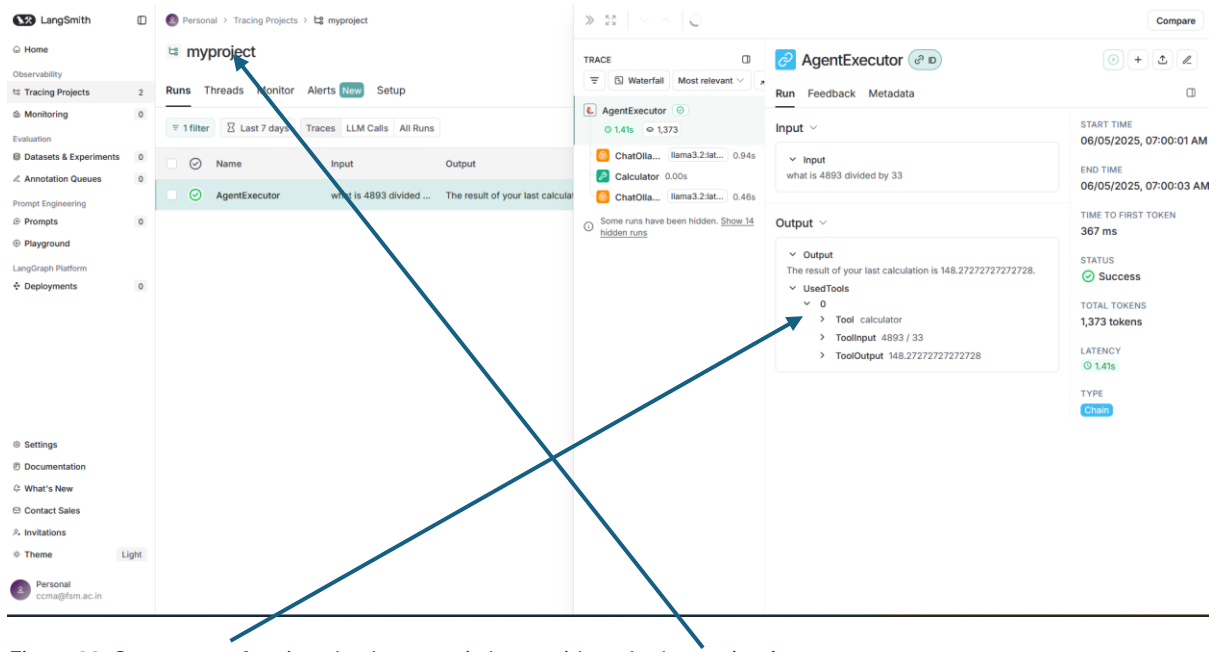


Figure 28: See traces of actions by the agent in langsmith under 'myproject'

N. 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*.

O. Example System message for Summarization

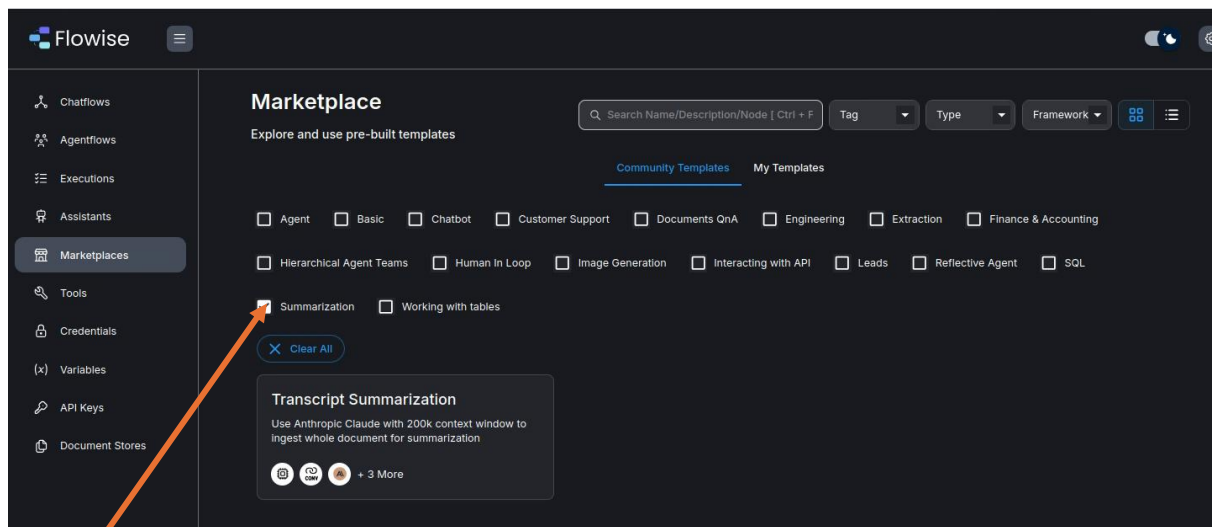
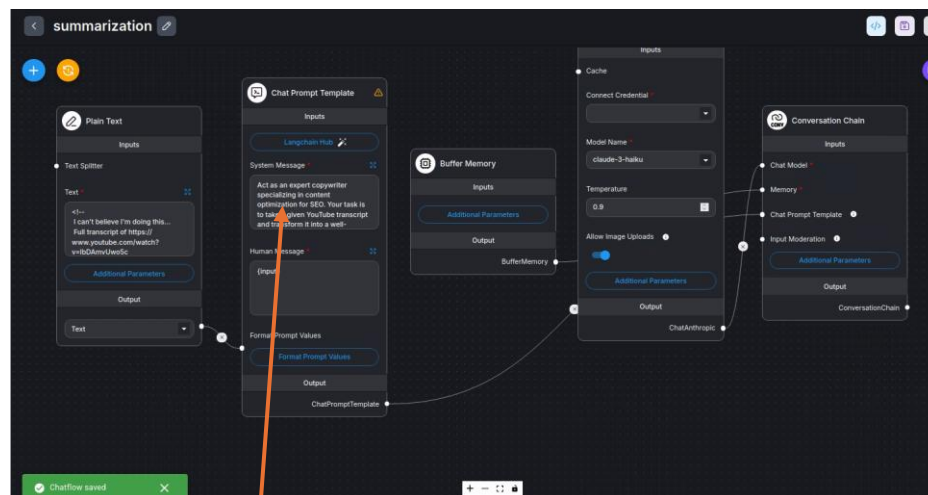


Figure 29: Summarization template in Marketplace



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}

P. 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

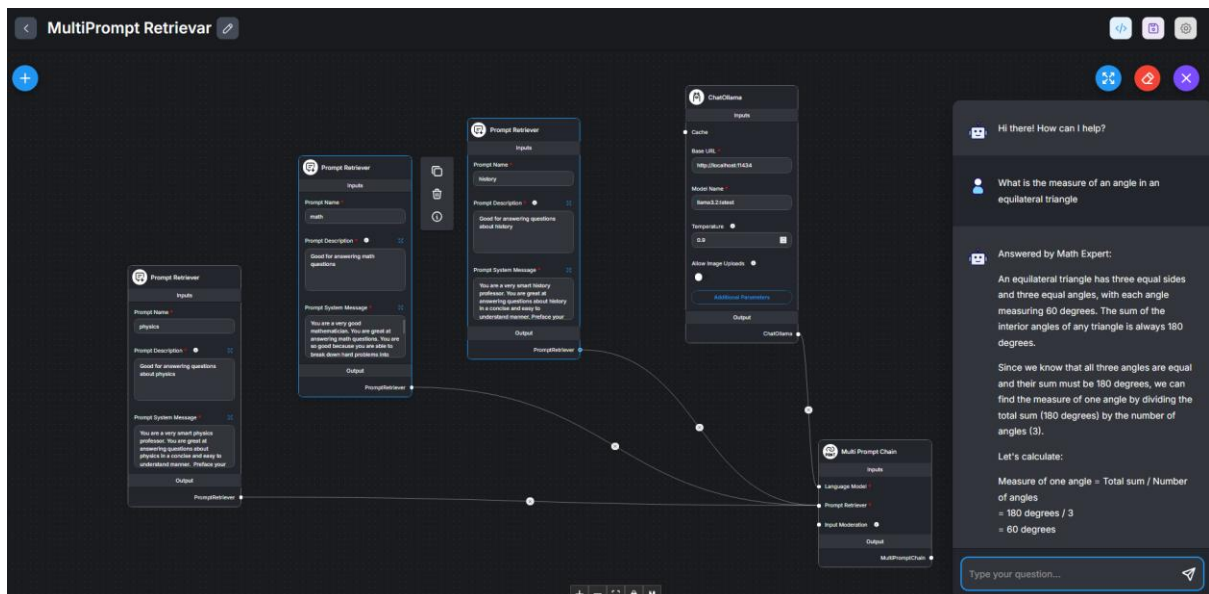


Figure 30: A multi-prompt retriever selects one of the prompt retrievers before it answers questions.

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

Q. 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.

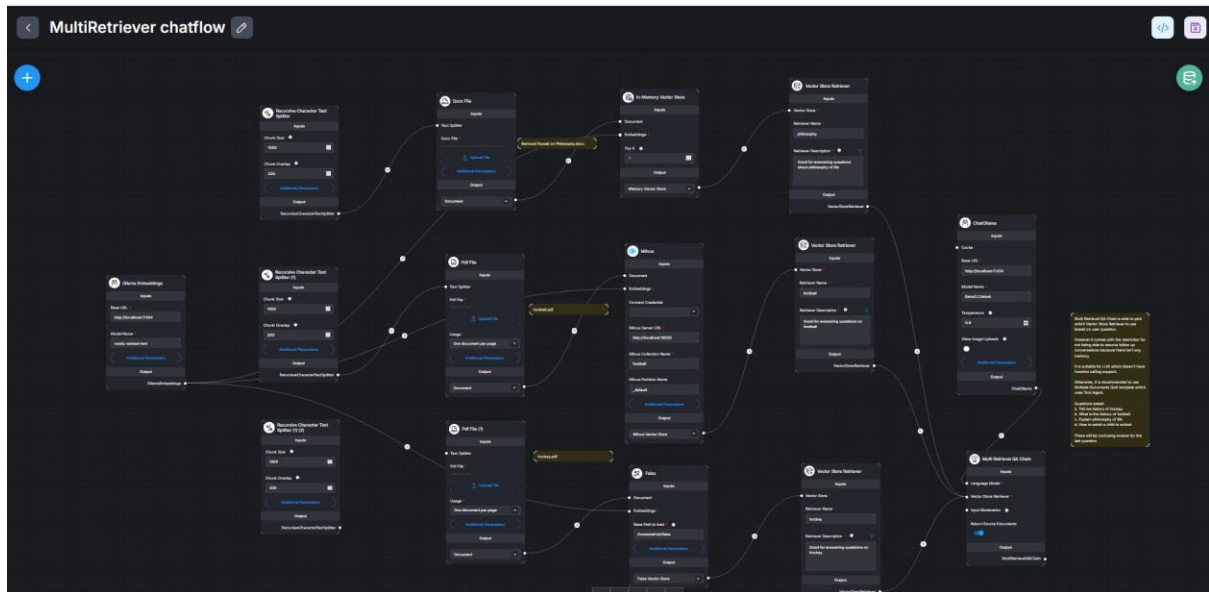


Figure 31: Multiple vector stores for answer extraction.

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.

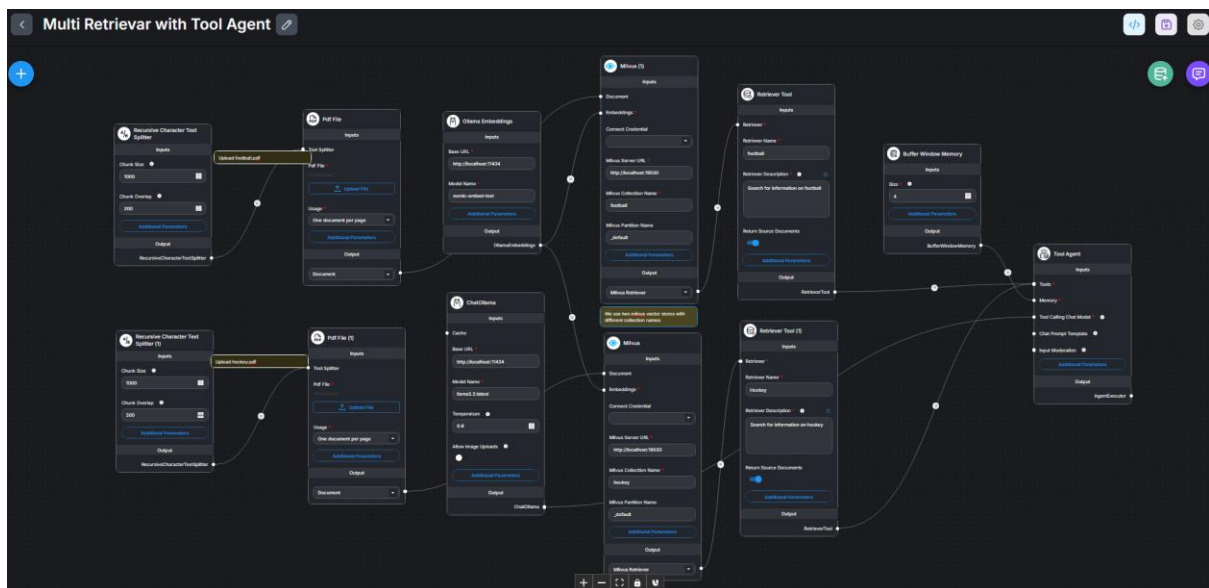


Figure 32: Tool Agent accessing multiple vector stores

R. Using Milisearch vector store

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

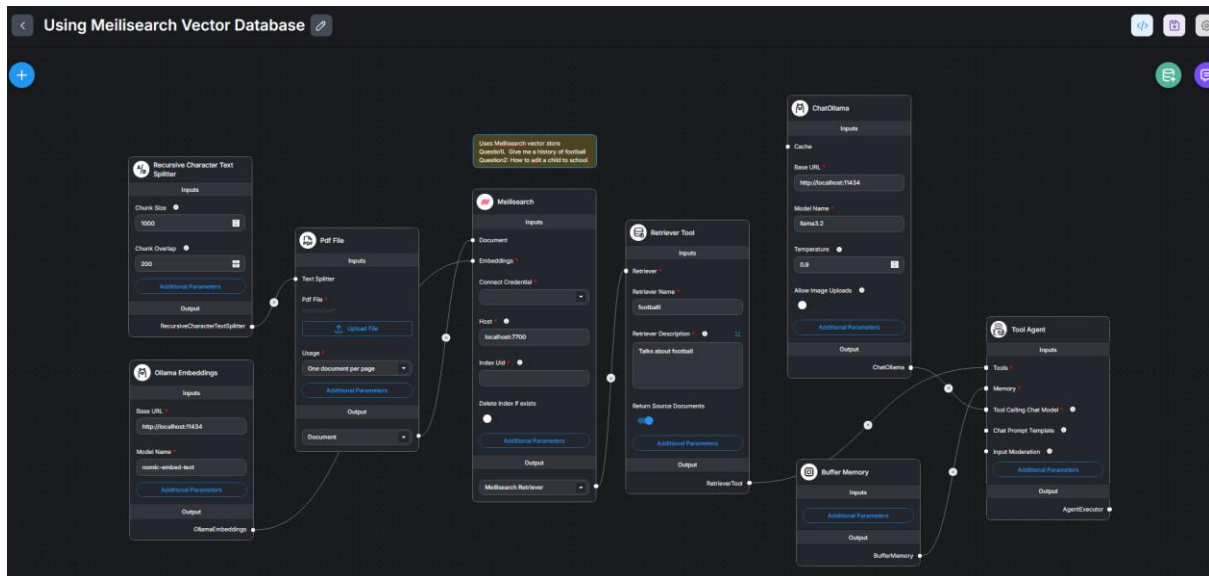


Figure 33: Using Milisearch vector store
