Flowise Tool-Agent

Last amended: 22nd February, 2025
File: **using tool-agent Chatflow.json**
My folder: D:\Documents\OneDrive\Documents\flowise\agent\agents\tool-agent simple

Contents

LLM Chains vs Agents

See [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 Agent is now replaced by Tool Agent.

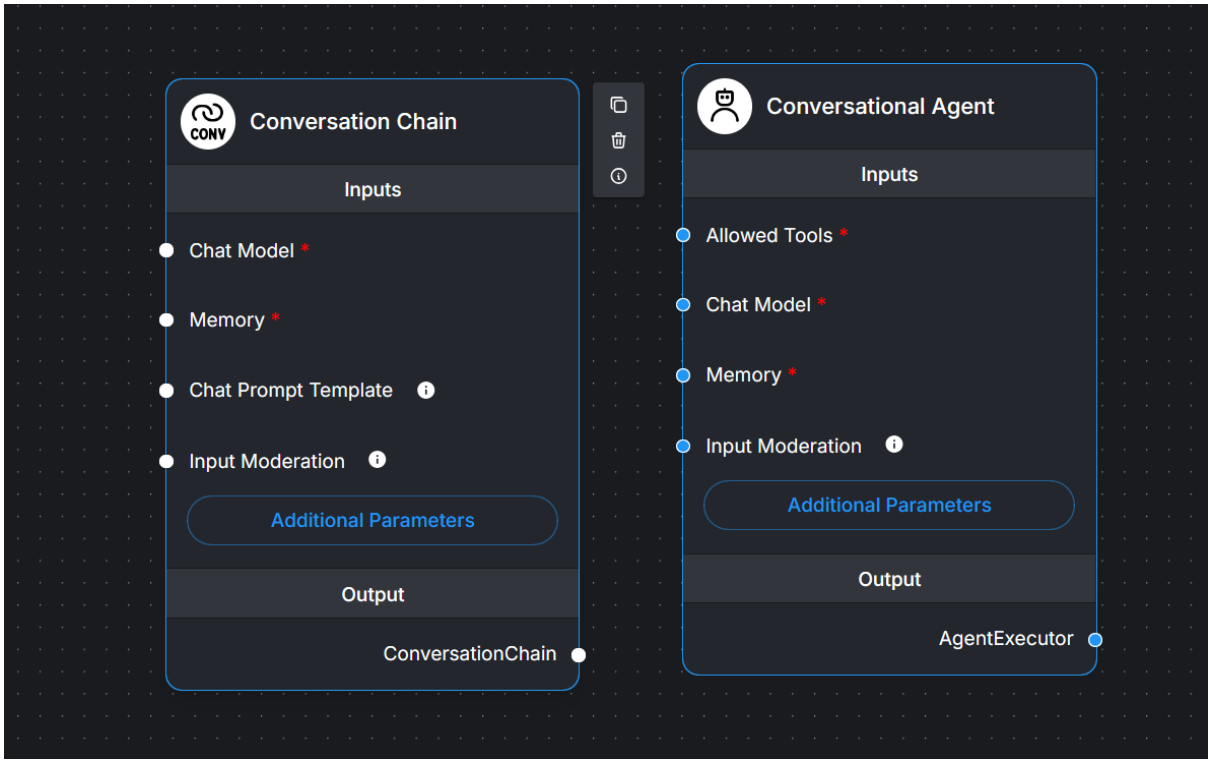


Figure 1: LLM chain vs Agent. Conversation Chain needs Mempry to keep track of earlier chats.

What is Tool Agent?

In Flowise, a "*Tool Agent*" is a type of AI agent that is specifically designed to utilize external tools and functions to complete tasks, allowing it to access and leverage various capabilities beyond just its own language model by calling specific tools when needed, effectively enhancing its functionality and range of actions.
Key points about Tool Agents in Flowise:

Function Calling:

The primary mechanism of a Tool Agent is to call functions or tools based on the situation, enabling it to interact with external data sources, APIs, or other services.

Flexibility:

You can integrate various tools with a Tool Agent, including search functions, calculators, data retrieval tools, and more, making it adaptable to different use cases.

Agent Design:

When building an AI agent in Flowise, you can choose to create a Tool Agent and then specify which tools it can access and how to use them depending on the task at hand.

Example Use Case:

Imagine building a customer service agent using Flowise. A Tool Agent could be designed to access a knowledge base tool to retrieve relevant information about a customer's query, then use a natural language processing tool to generate a clear and concise response

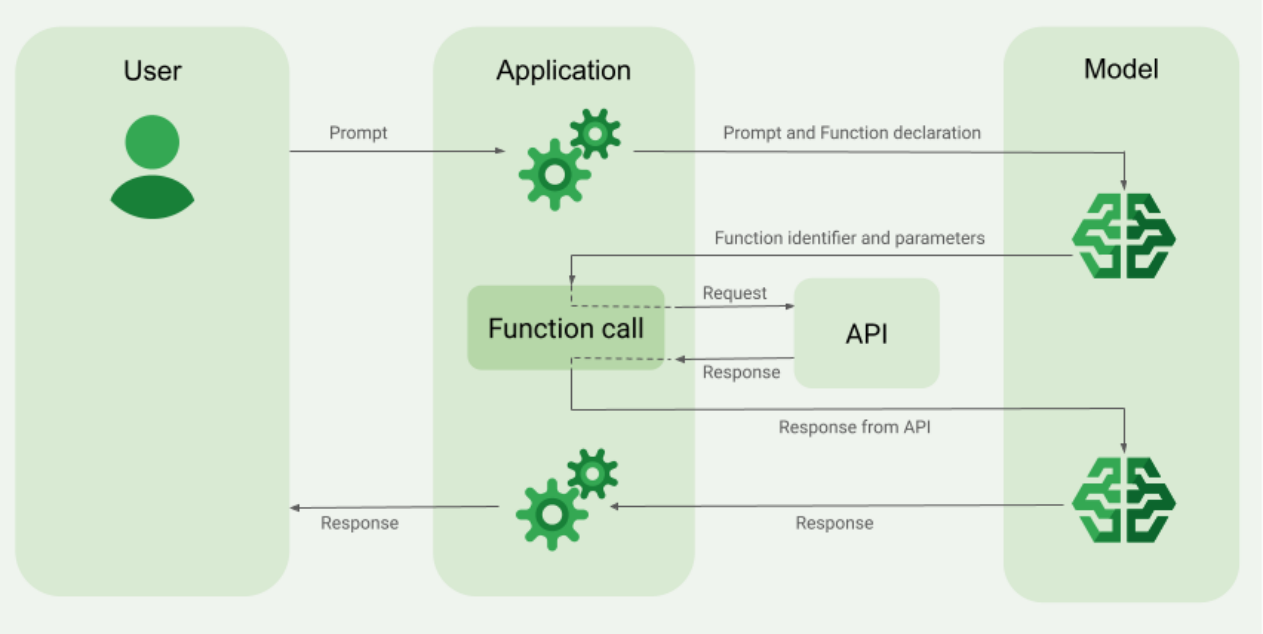


Figure 2: When user enters a prompt, the prompt + available function calls travel to the Tool-agent.

Steps:

1. User submits prompt to tool-agent
2. Tool-agent passes this prompt along with all function declarations to the Model.
3. The Model decides which of the functions to call and specifies its parameters.
4. The Tool-agent calls the function with necessary parameters and collects the function's response.
5. Tool-Agent passes the functions response to the Model. Model paraphrases the response as per the user's prompt and sends it back to the user.

Example use cases

1. **Interpret voice commands**: Create functions that correspond with in-vehicle tasks. For example, you can create functions that turn on the radio or activate the air conditioning. Send audio files of the user's voice commands to the model, and ask the model to convert the audio into text and identify the function that the user wants to call.
2. **Automate workflows based on environmental triggers**: Create functions to represent processes that can be automated. Provide the model with data from environmental sensors

and ask it to parse and process the data to determine whether one or more of the workflows should be activated. For example, a model could process temperature data in a warehouse and choose to activate a sprinkler function.

3. **Automate the assignment of support tickets**: Provide the model with support tickets, logs, and context-aware rules. Ask the model to process all of this information to determine who the ticket should be assigned to. Call a function to assign the ticket to the person suggested by the model.
4. **Retrieve information from a knowledge base**: Create functions that retrieve academic articles on a given subject and summarize them. Enable the model to answer questions about academic subjects and provide citations for its answers.
5. **Predictive maintenance:** Manufacturing companies use AI-driven automation for predictive maintenance. Traditional automation systems can schedule regular maintenance tasks, but AI enhances this by predicting when machinery is likely to fail based on historical data. AI can analyse equipment performance data and predict issues before they happen, enabling companies to perform maintenance only when needed. This minimizes downtime and extends the lifespan of equipment.
6. **Fraud detection in finance:** Financial institutions use AI to detect fraudulent transactions in real time. Automation can flag certain transactions based on predefined rules (e.g., transactions over a certain amount). However, AI goes beyond this by learning patterns of normal behavior and flagging transactions that deviate from those patterns. AI can detect fraud that would be missed by traditional rule-based systems, improving security and reducing financial risk.

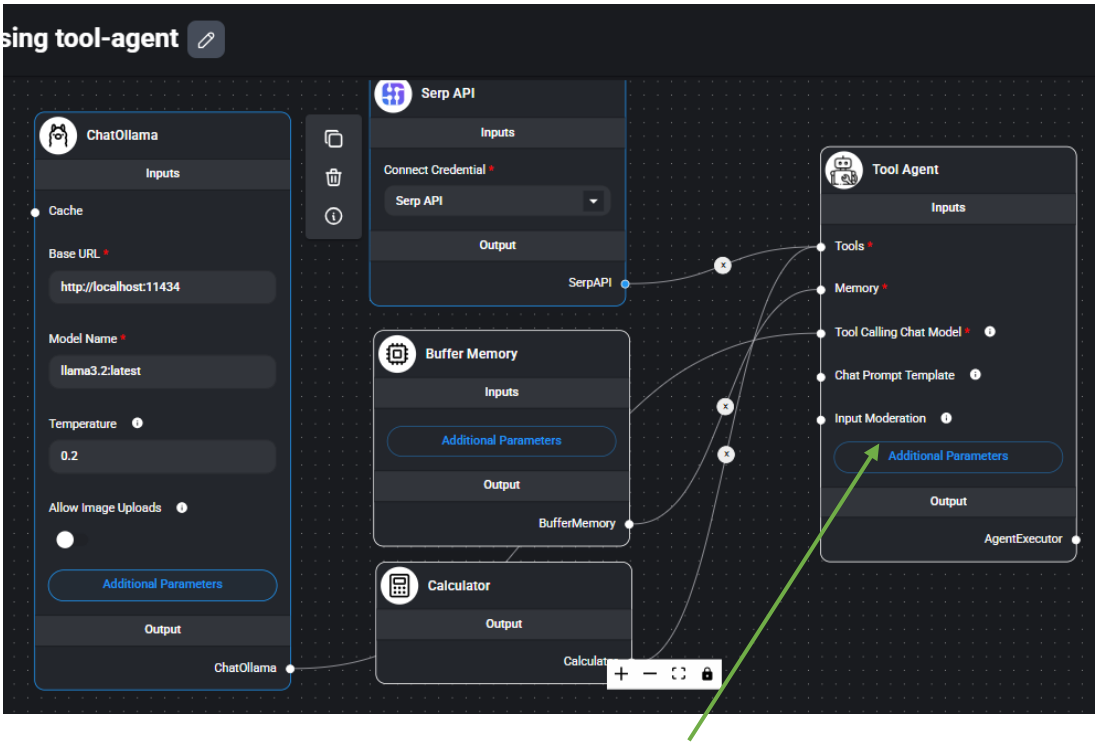


Figure 3: System message prompt is written within Additional Parameters of tool agent.

Folder: C:\Users\ashok\OneDrive\Documents\flowise\agents\tool-agent simple
File: Calculator_AI_agent_III Chatflow.json

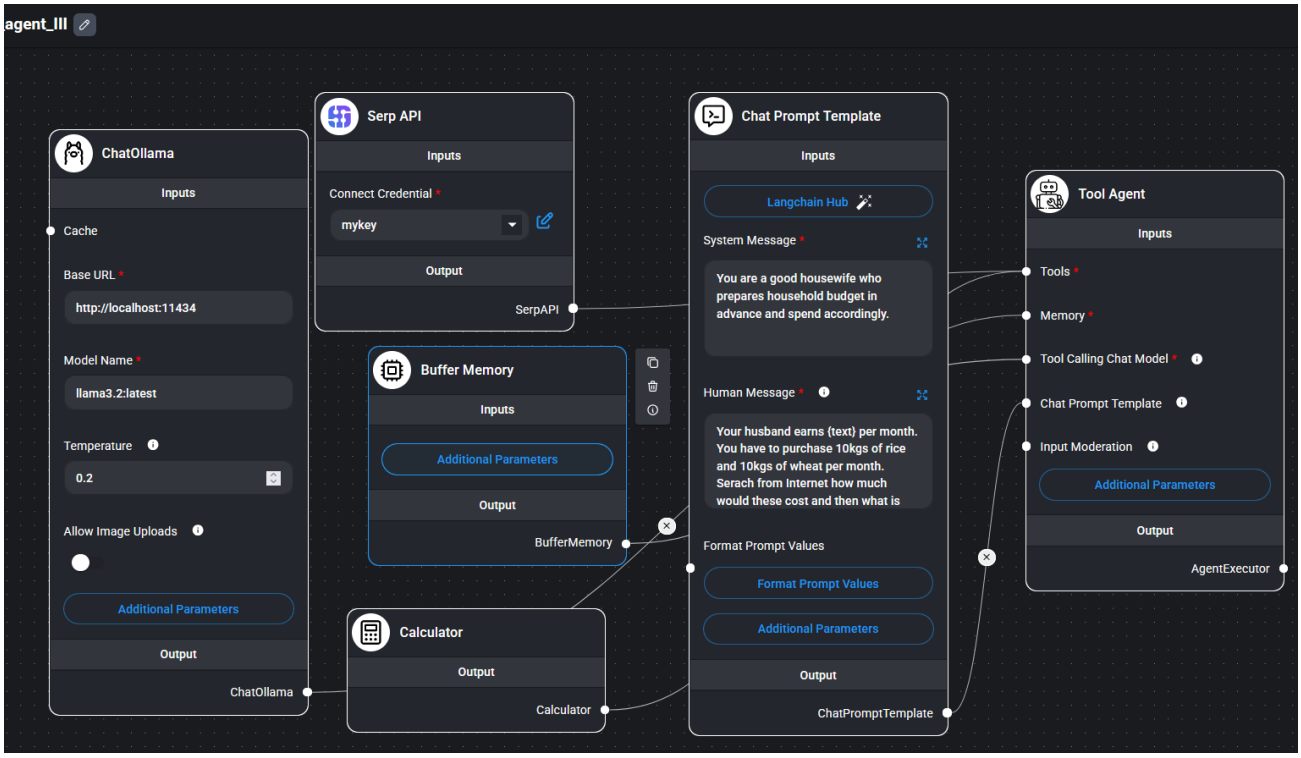
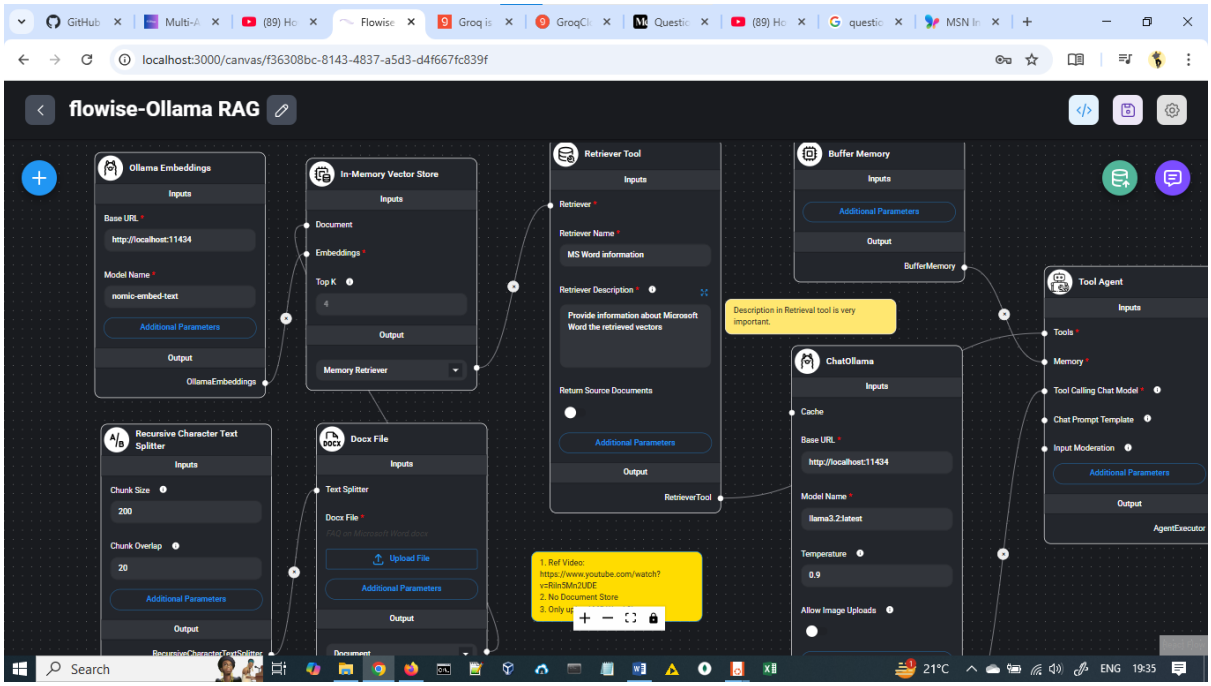


Figure 4: When Chat Prompt Template is used, System message and Human Message of Chat Prompt Template override System Prompt of Tool Agent

System message: You are a good housewife who prepares household budget in advance and spend accordingly.
Human message: Your husband earns {text} per month. You have to purchase 10kgs of rice and 10kgs of wheat per month. Search from Internet how much would these cost and then what is the balance amount left.

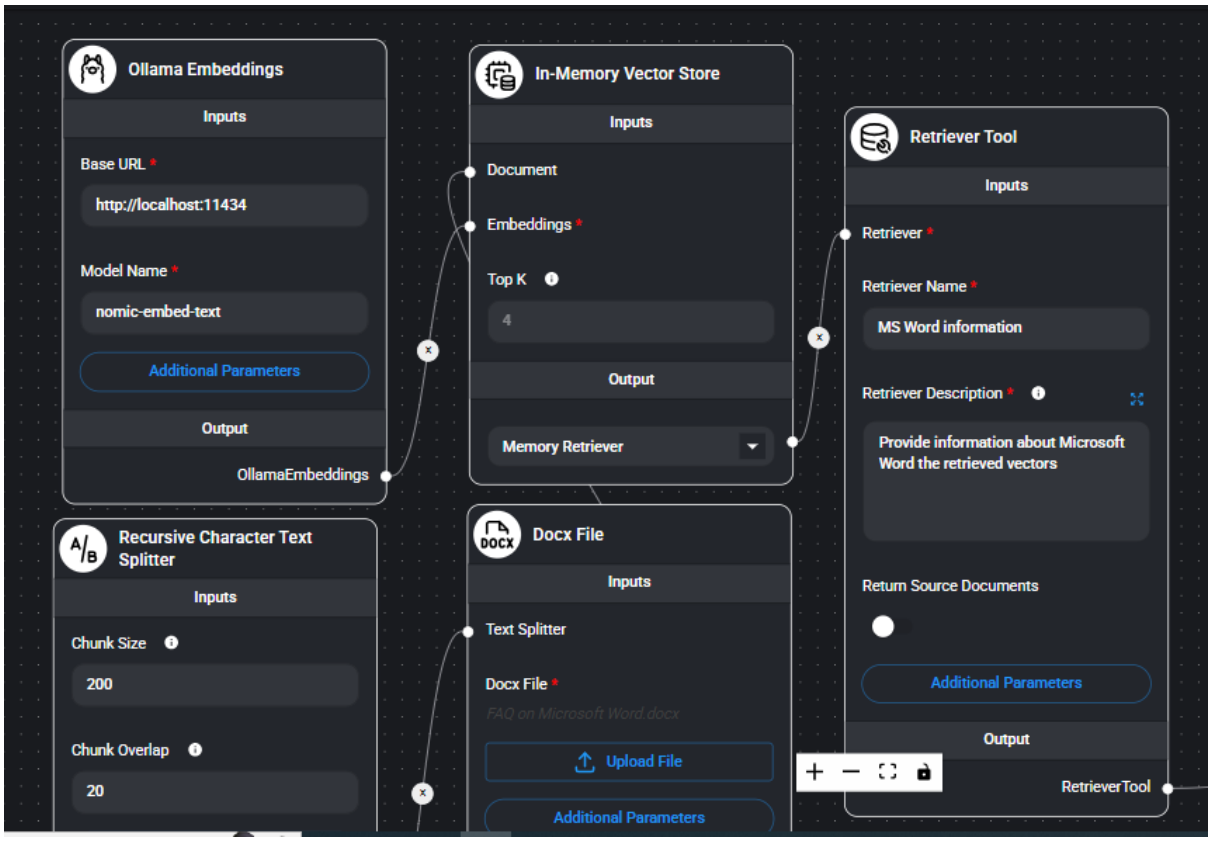
Retrieval Tool: In RAG **Retrieval Tool** name is extremely important so also its description.

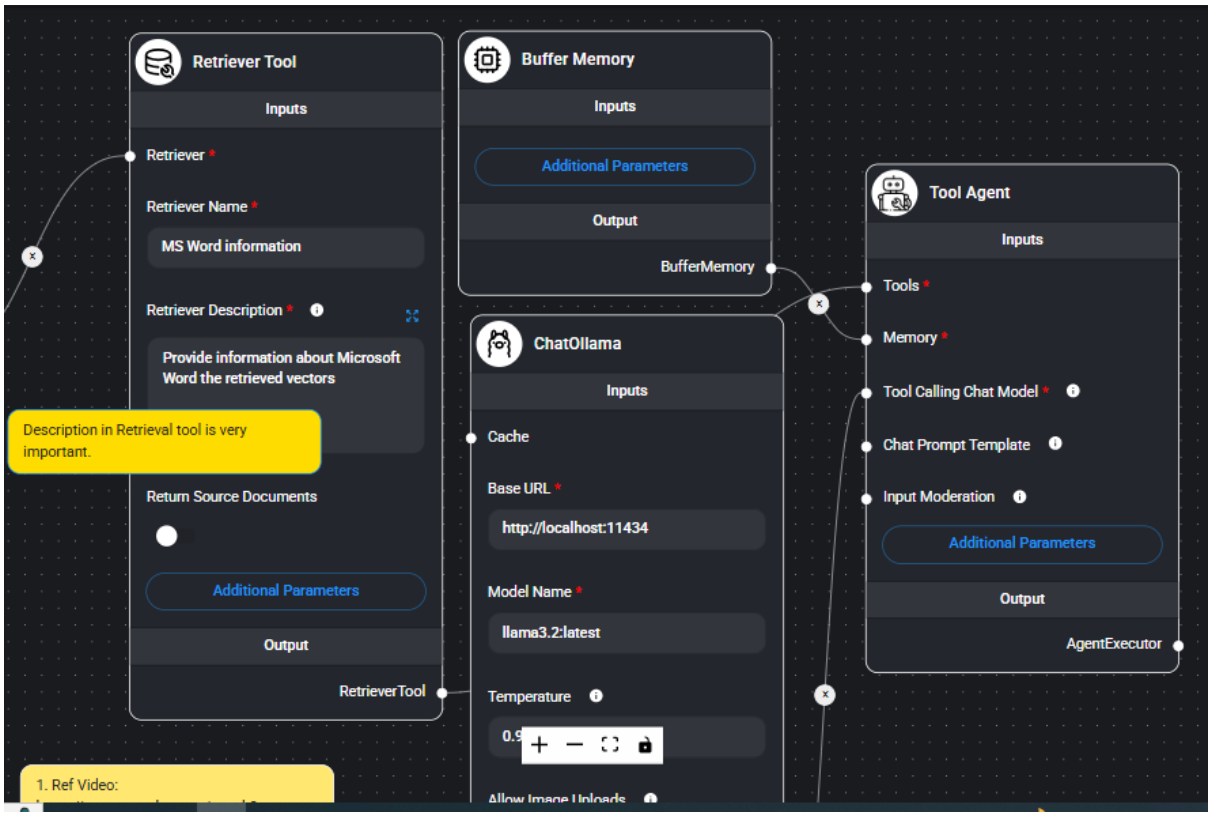


Used widgets:

1. Agents→Tool-agent
2. Vector Store→In Memory Vector store
3. Memory→Buffer Memory
4. Chat Models→Chat Ollama
5. Embeddings→Ollama Embeddings
6. Text Splitters→Recursive Character Tet Splitter (250,20)
7. Document Uploader→DocX file uploader
8. Tools→Retrieval Tool

Zoomed views are as below:





Experiments in RAG

(Refer file: `flowise/chatflows/7A.RAG Using Meilisearch Vector Database Chatflow.json`)

We use Meillisearch vector database (docker). It is easy to install and to use.

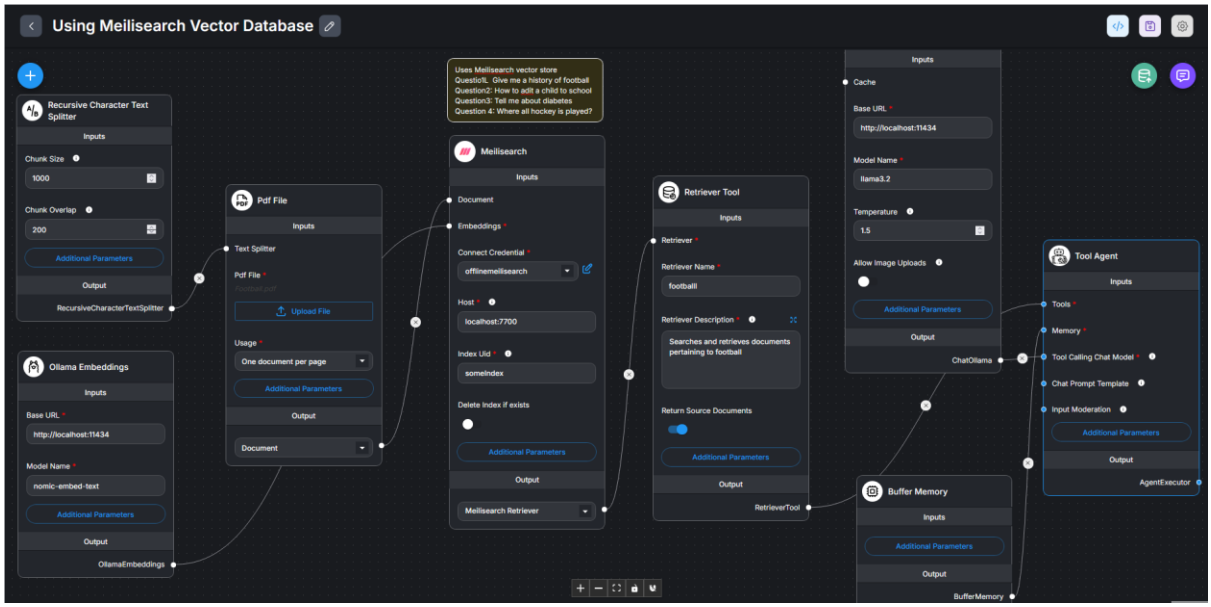


Figure 5: Tool Agent is connected to Retrieval tool and to meillisearch vector database. Prompts are important. Document is Wikipedia pdf file of [Football]

1.
Retrieval tool name: Football
Retriever description: Searches and retrieves documents pertaining to game of football

2.
System message in Tool Agent:

> You are a helpful AI assistant. You provide answers only to user questions that pertain to the game of football. Your answers use only the provided documents and nothing else. Do not answer from your own knowledge. If the provided documents do not contain answer to the user's question, say 'I do not know'.

3.
Sample Questions to ask:

> Questio1: Give me a history of football
> Question2: How to adit a child to school
> Question3: Tell me about diabetes
> Question 4: Where all hockey is played?

4.
Experiments:

> Try to have a Chat Prompt Template and experiment with it.

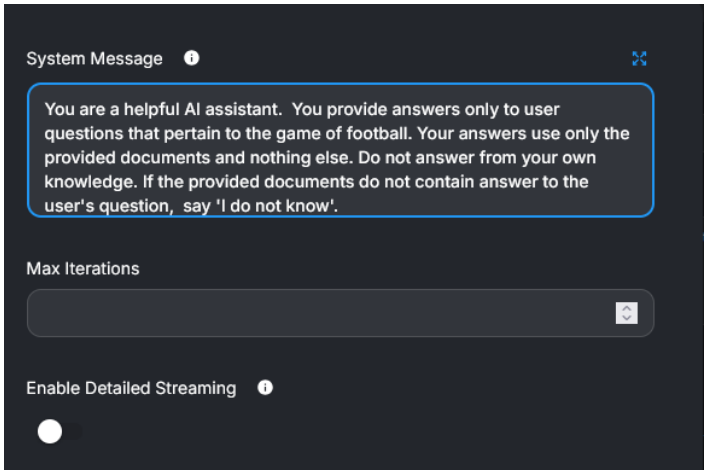


Figure 6: System message in Tool Agent

Using langsmith for tracing agent flows:

Create the following simple project:

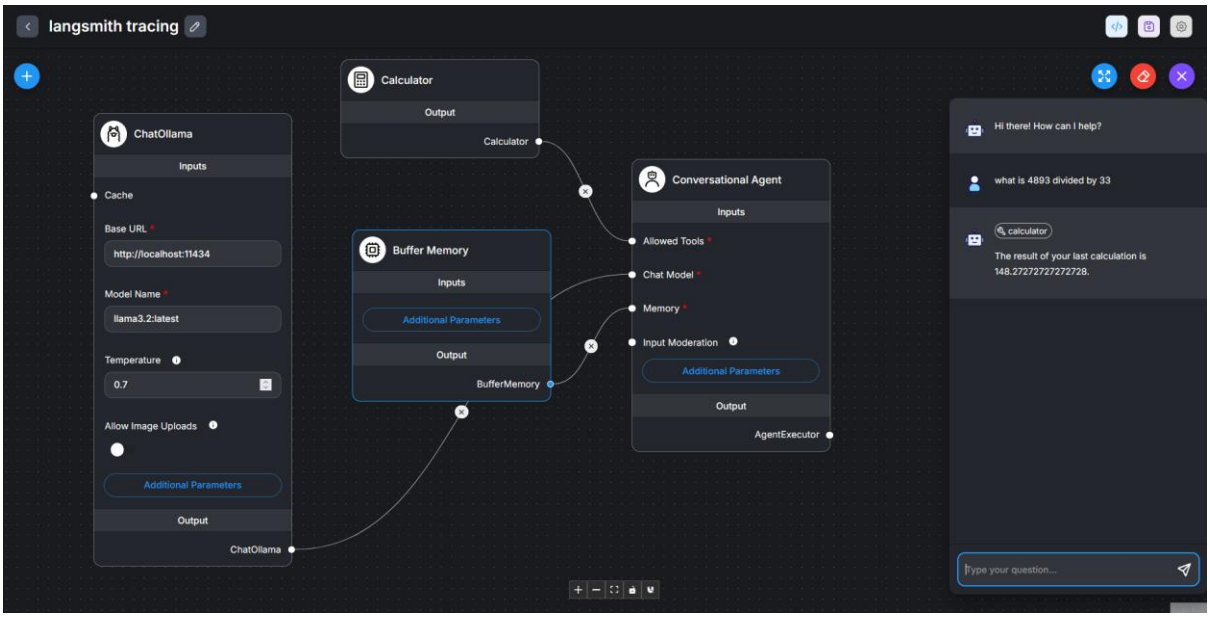


Figure 7: Click on gear icon for configuration

Log into langsmith and get a free API key. Configure your project to use langsmith, as:

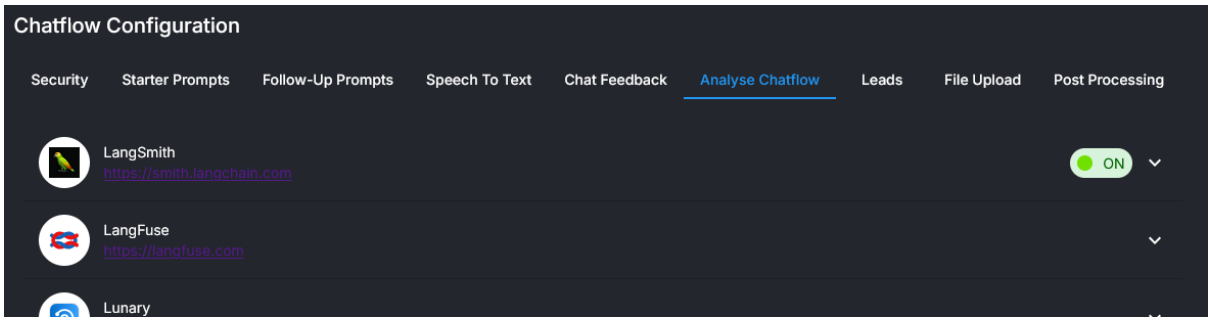


Figure 8: Click on Langsmith to open it

Set up your Credentials a project and switch langsmith usage as on. Save the confiuratuion.

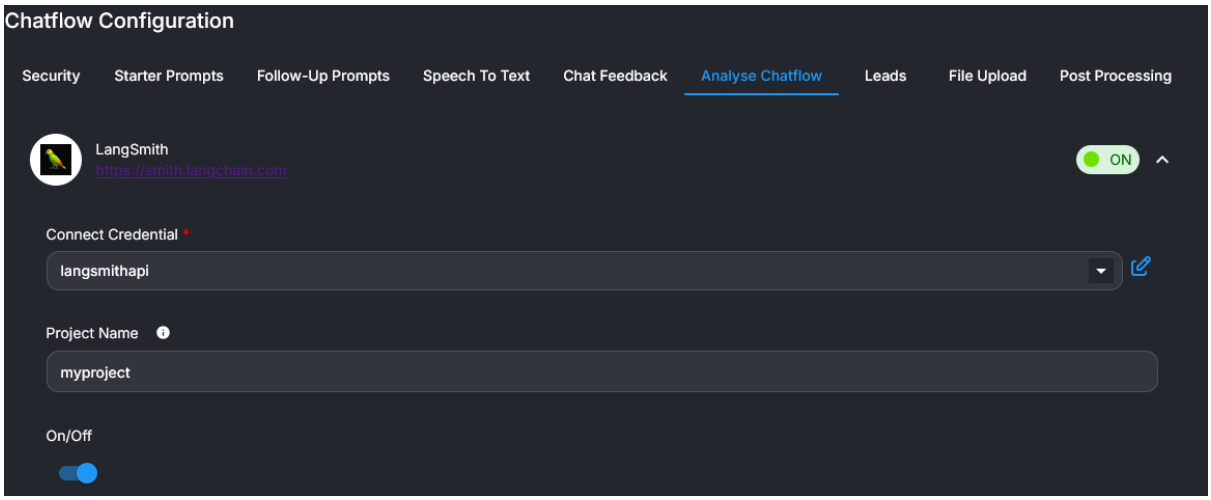


Figure 9: Creating credential for langsmith usage

In langsmith under Personal→Tracing Projects→myproject see traces of what happened.

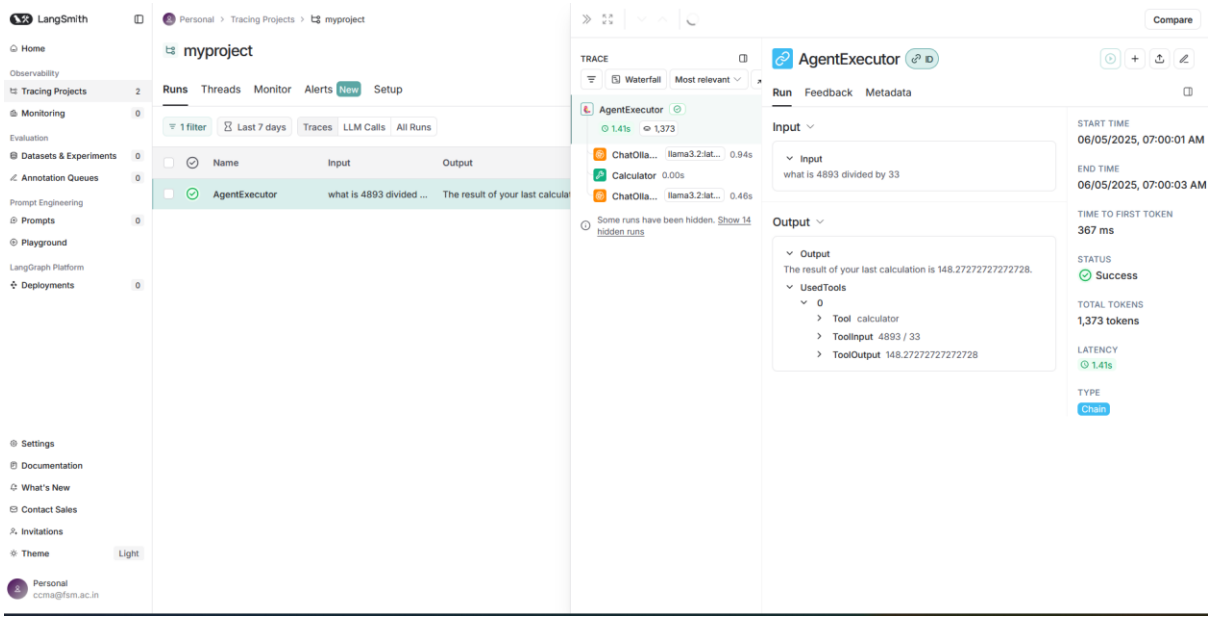


Figure 10: Tracing events in langsmith

****