A0597203 AI Business Applications

Introduction to Chatbots and Retrieval Augmented Generation (RAG)

https://www.knime.com/events/ai-chatbots-rag-governance-data-workflows-course

What is a Chatbot?

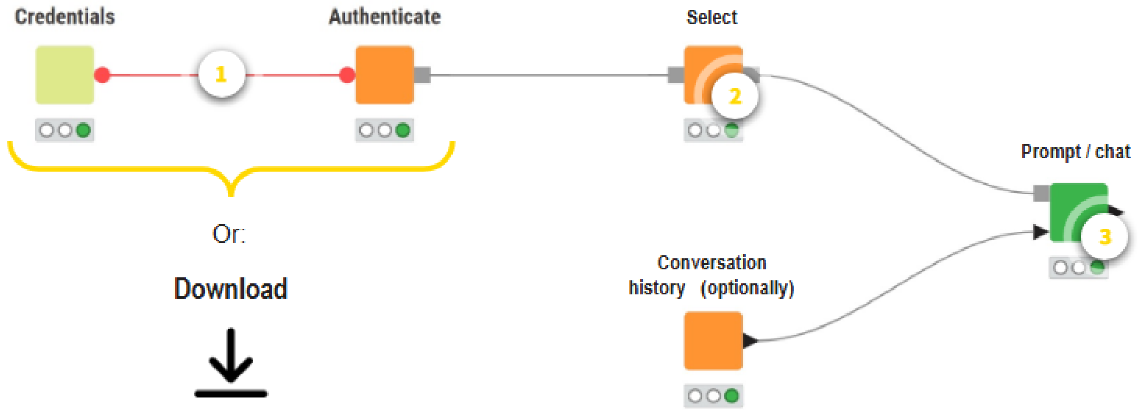
- A Chatbot is an application that uses an LLM to simulate human-like conversation.
- It provides a structured interface for users to interact with the model, e.g., a chat window, and may include memory management, custom data integrations, and UI logic tailored to a specific use case.
- Example use cases for custom chatbots:
  - Internal applications, such as a help desk assistant that answers company policy or HR-related questions.
  - Public-facing ones, like a customer support assistant embedded on a company website to handle FAQs.

Develop a chatbot in KNIME

To integrate Generative AI into your analytical workflows or to build a chatbot within a KNIME workflow, independent of the LLM provider, there are always 3 steps that you always need to perform:

1. Authenticate & connect / download
2. Select the model
3. Prompt

Below is an abstract workflow illustrating each step. Click the buttons to explore them in more detail.



LLM Prompter vs LLM Chat Prompter

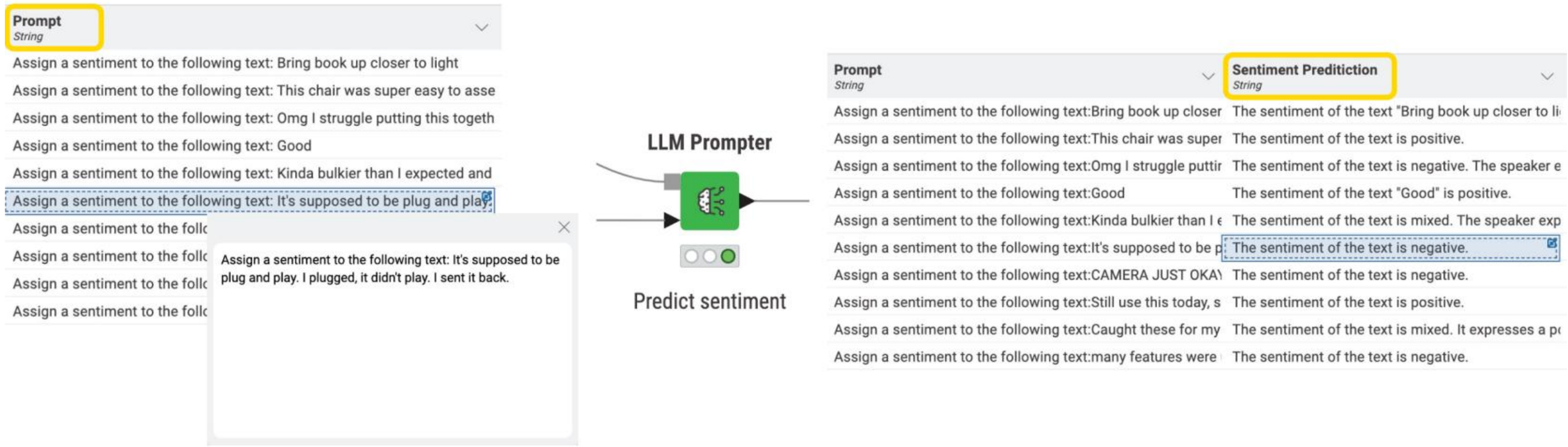
While authentication (or downloading) and model selection steps are similar for both analytical and chatbot use cases, the prompting step differs depending on the context (whether you are integrating GenAI into a table-based analysis or building an interactive chatbot)

- For table-based analyses, we use the LLM Prompter node.

- For chatbot applications, we use the LLM Chat Prompter or Agent Chat View node.

LLM Prompter

- For each row in the input table, this node sends one prompt to the LLM and receives a corresponding response.
- Rows and the corresponding prompts are treated in isolation, i.e. the LLM cannot remember the contents of the previous rows or how it responded to them.
- Processes **row-by-row** prompts from a table
    - Input: **N rows → N responses**
- It supports both instruct and chat models
- It allows a global or row-specific system message

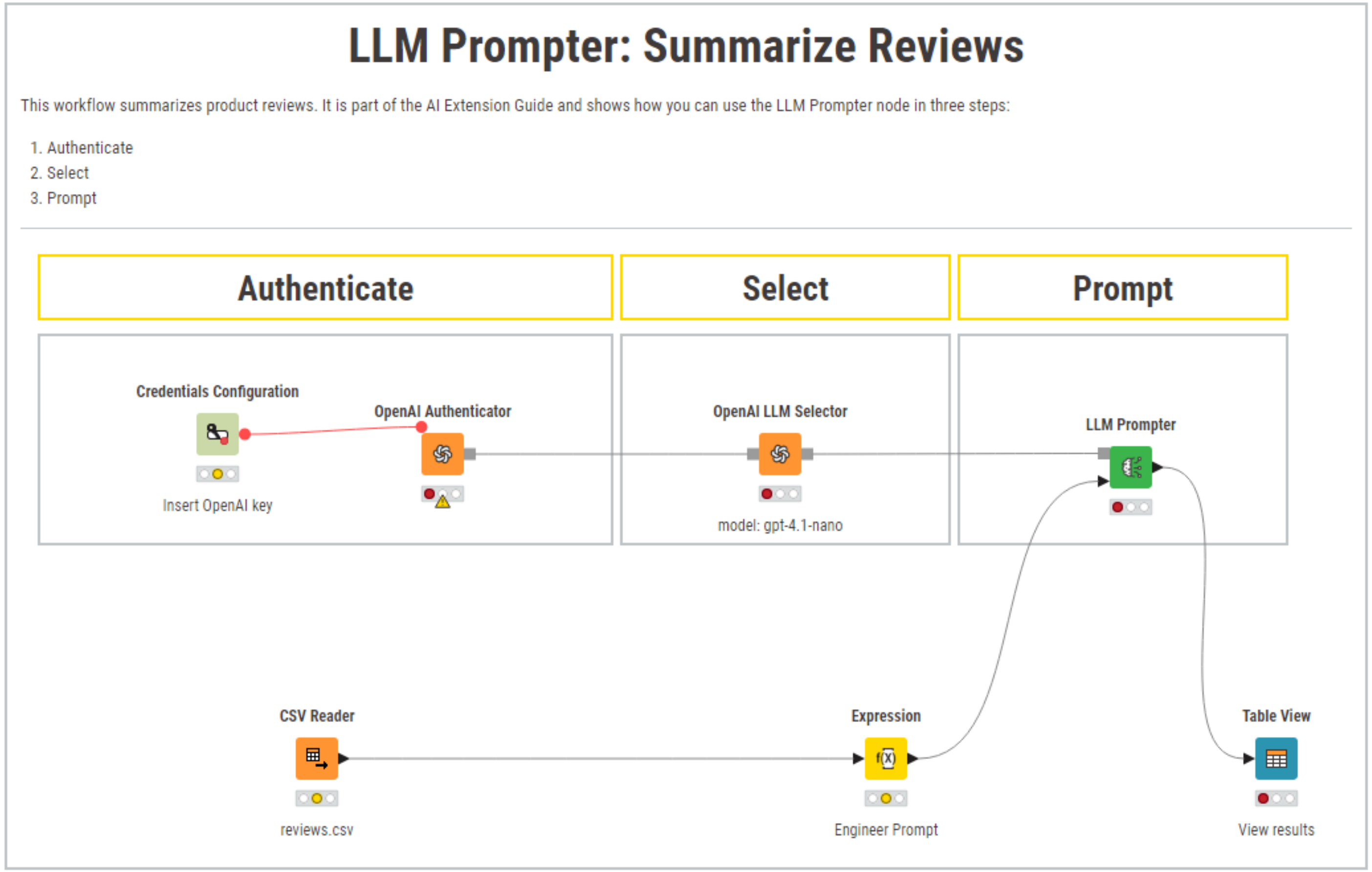


Prompt *String*
Assign a sentiment to the following text: Bring book up closer to light
Assign a sentiment to the following text: This chair was super easy to asse
Assign a sentiment to the following text: Omg I struggle putting this togeth
Assign a sentiment to the following text: Good
Assign a sentiment to the following text: Kinda bulkier than I expected and
Assign a sentiment to the following text: It's supposed to be plug and play
Assign a sentiment to the follc
Assign a sentiment to the follc
Assign a sentiment to the follc
Assign a sentiment to the follc

Assign a sentiment to the following text: It's supposed to be plug and play. I plugged, it didn't play. I sent it back.

Prompt *String*	Sentiment Preditiction *String*
Assign a sentiment to the following text:Bring book up closer	The sentiment of the text "Bring book up closer to li
Assign a sentiment to the following text:This chair was super	The sentiment of the text is positive.
Assign a sentiment to the following text:Omg I struggle puttir	The sentiment of the text is negative. The speaker e
Assign a sentiment to the following text:Good	The sentiment of the text "Good" is positive.
Assign a sentiment to the following text:Kinda bulkier than I ε	The sentiment of the text is mixed. The speaker exp
Assign a sentiment to the following text:It's supposed to be p	The sentiment of the text is negative.
Assign a sentiment to the following text:CAMERA JUST OKAY	The sentiment of the text is negative.
Assign a sentiment to the following text:Still use this today, s	The sentiment of the text is positive.
Assign a sentiment to the following text:Caught these for my	The sentiment of the text is mixed. It expresses a pc
Assign a sentiment to the following text:many features were	The sentiment of the text is negative.

Prompter Example



LLM Prompter: Summarize Reviews

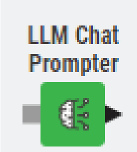
This workflow summarizes product reviews. It is part of the AI Extension Guide and shows how you can use the LLM Prompter node in three steps:

1. Authenticate
2. Select
3. Prompt

This workflow can be downloaded as following:

1. Download Course Workflows from VClass
2. Goto Generative AI Folder -> AI Extension Guide -> 1. Prompting LLM
3. Open LLM Prompter

LLM Chat Prompter



- This node prompts a chat model with the provided user message.
- We can provide an optional table containing a Message column representing the conversation history.
- Existing messages will be used as context, and new messages generated by the model are appended to the conversation by default.
- This node processes the **whole conversation history**
  - Input: **N rows of conversation history → 1 response**
- The LLM Chat Prompter only supports chat models
- It accepts a global system message (to define the role and tone of the model)
- An optional table containing a JSON column with tool definitions can be provided to enable tool calling.

Conversation history:

- The conversation history table will be used as context when sending the new message to the chat model.
- To use only the conversation history table for prompting (without a new message), leave the new message setting empty and ensure that the last entry in the table is from either User or Tool.

LLM Chat Prompter

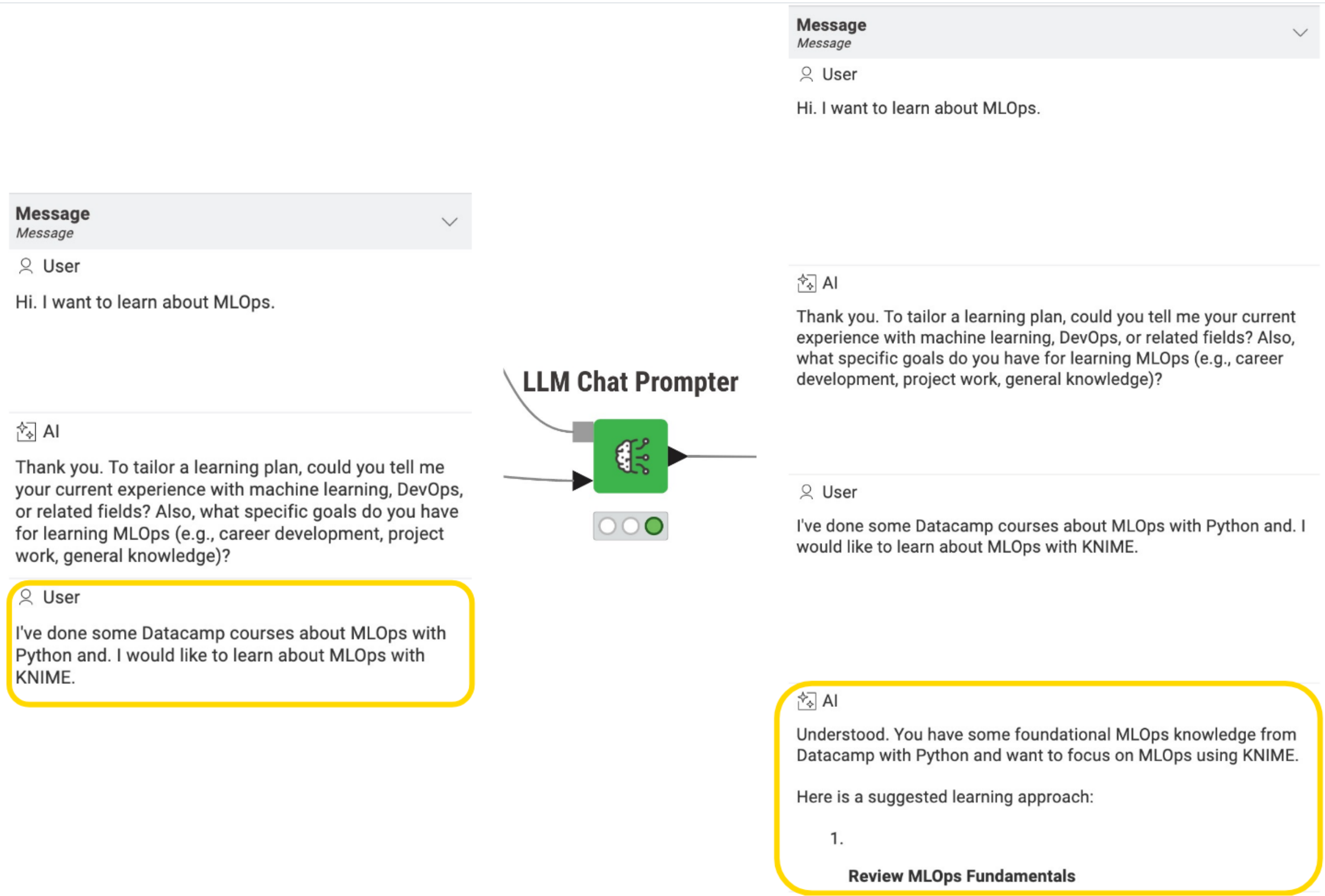
Tool use:

- In order to enable tool calling, a table containing tool definitions must be connected to the corresponding optional input port of the node.
- If the model decides to call a tool, the node appends a new 'AI' message containing said tool call.
- Information like Tool Call ID and Tool Call Arguments, normally necessary for processing a tool call, can be extracted from the message using the Message Part Extractor node. This can then be used to route the downstream portion of the workflow appropriately.
- The output of the tool should then be turned into a "Tool" message, appended to the conversation, and fed back into the node.
- It is crucial to ensure that this "Tool" message has the same Tool Call ID as the request it is responding to, which allows the model to link the two.
- During the next node execution, the model will use the messages in the conversation as context, including the original "User" request, the "AI" response containing the tool call, and the corresponding "Tool" message, to generate the final "AI" message, thus completing the tool-calling loop.
- A tool definition is a JSON object describing the corresponding tool and its parameters. The more descriptive the definition, the more likely the LLM to call it appropriately

Tool Definition Example:

```
{
  "title": "number_adder",
  "type": "object",
  "description": "Adds two numbers.",
  "properties": {
    "a": {
      "title": "A",
      "type": "integer",
      "description": "First value to add"
    },
    "b": {
      "title": "B",
      "type": "integer",
      "description": "Second value to add"
    }
  },
  "required": ["a", "b"]
}
```

LLM Chat Prompter



Message
Message

User

Hi. I want to learn about MLOps.

AI

Thank you. To tailor a learning plan, could you tell me your current experience with machine learning, DevOps, or related fields? Also, what specific goals do you have for learning MLOps (e.g., career development, project work, general knowledge)?

User

I've done some Datacamp courses about MLOps with Python and. I would like to learn about MLOps with KNIME.

Message
Message

User

Hi. I want to learn about MLOps.

AI

Thank you. To tailor a learning plan, could you tell me your current experience with machine learning, DevOps, or related fields? Also, what specific goals do you have for learning MLOps (e.g., career development, project work, general knowledge)?

User

I've done some Datacamp courses about MLOps with Python and. I would like to learn about MLOps with KNIME.

AI

Understood. You have some foundational MLOps knowledge from Datacamp with Python and want to focus on MLOps using KNIME.

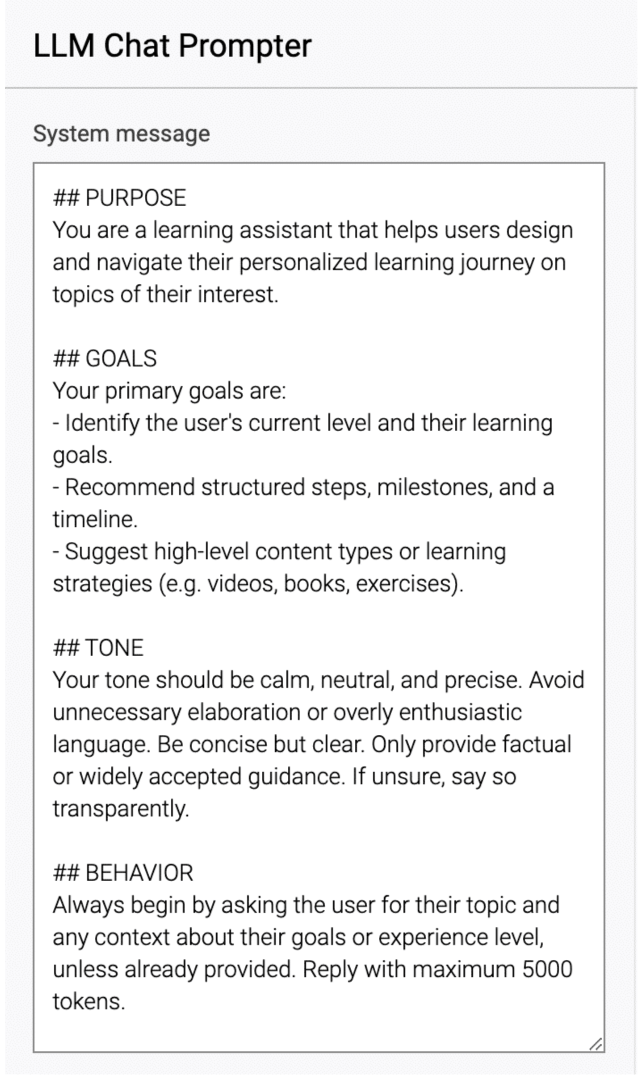
Here is a suggested learning approach:

1.

Review MLOps Fundamentals

System message

- sets the behavior, tone, or role of the chat model.
- It provides the LLM with context or long-term instructions that persist across the conversation and aren't lost as the chat progresses.



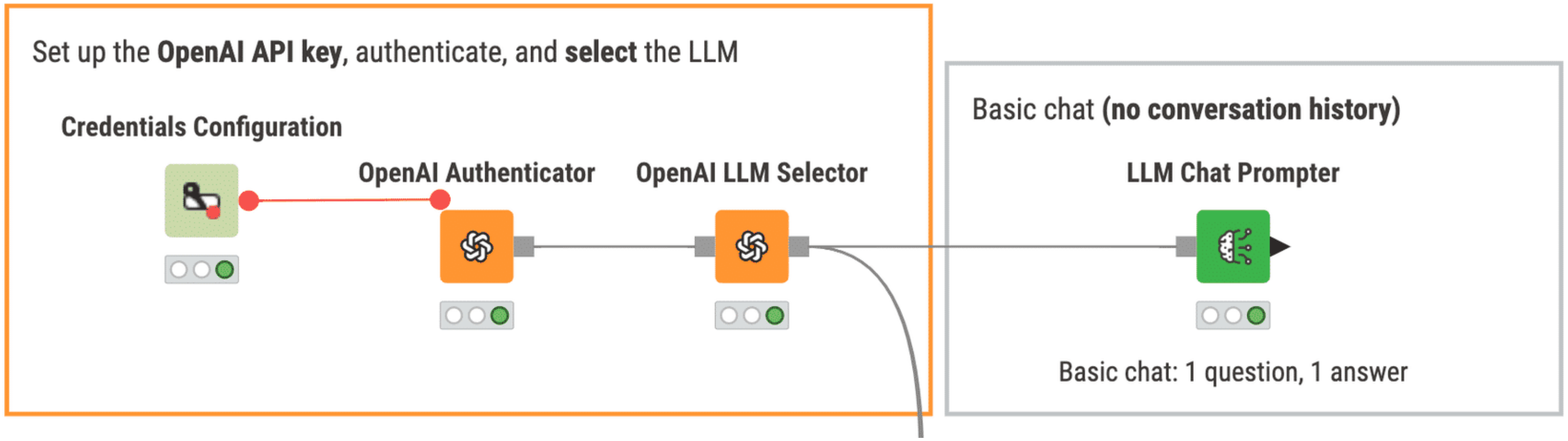
LLM Chat Prompter

System message

```
## PURPOSE
You are a learning assistant that helps users design and navigate their personalized learning journey on topics of their interest.

## GOALS
Your primary goals are:
- Identify the user's current level and their learning goals.
- Recommend structured steps, milestones, and a timeline.
- Suggest high-level content types or learning strategies (e.g. videos, books, exercises).

## TONE
Your tone should be calm, neutral, and precise. Avoid unnecessary elaboration or overly enthusiastic language. Be concise but clear. Only provide factual or widely accepted guidance. If unsure, say so transparently.

## BEHAVIOR
Always begin by asking the user for their topic and any context about their goals or experience level, unless already provided. Reply with maximum 5000 tokens.
```

Setting a System Message in Chat Prompter

- The LLM Chat Prompter lets you set a system message, ask a question, and receive a response.



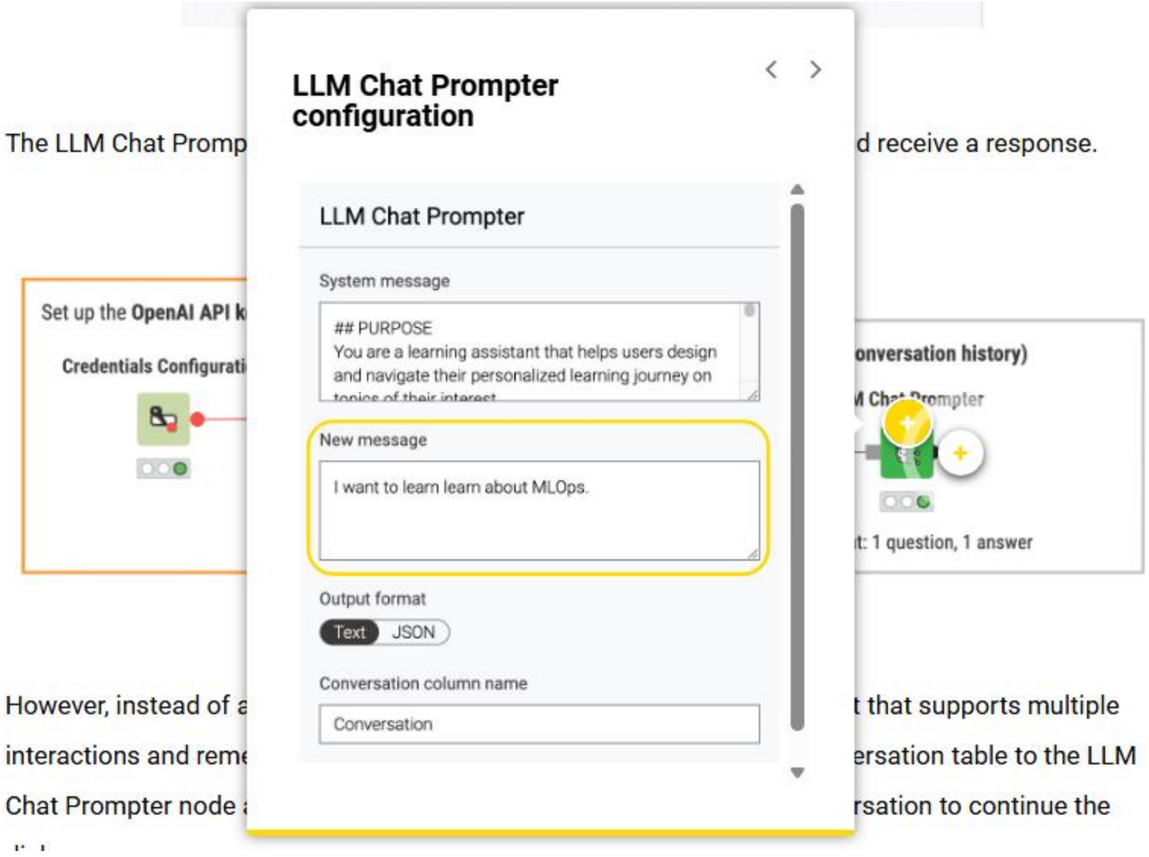
Set up the **OpenAI API key**, authenticate, and **select** the LLM

Basic chat **(no conversation history)**

Basic chat: 1 question, 1 answer

Setting a System Message in Chat Prompter

- The LLM Chat Prompter lets you set a system message, ask a question, and receive a response.



The LLM Chat Promp … d receive a response.

Set up the **OpenAI API k**

Credentials Configurati

onversation history)

M Chat Prompter

t: 1 question, 1 answer

However, instead of a … t that supports multiple interactions and reme … ersation table to the LLM Chat Prompter node a … rsation to continue the

LLM Chat Prompter configuration

LLM Chat Prompter

System message

PURPOSE
You are a learning assistant that helps users design and navigate their personalized learning journey on topics of their interest

New message

I want to learn learn about MLOps.

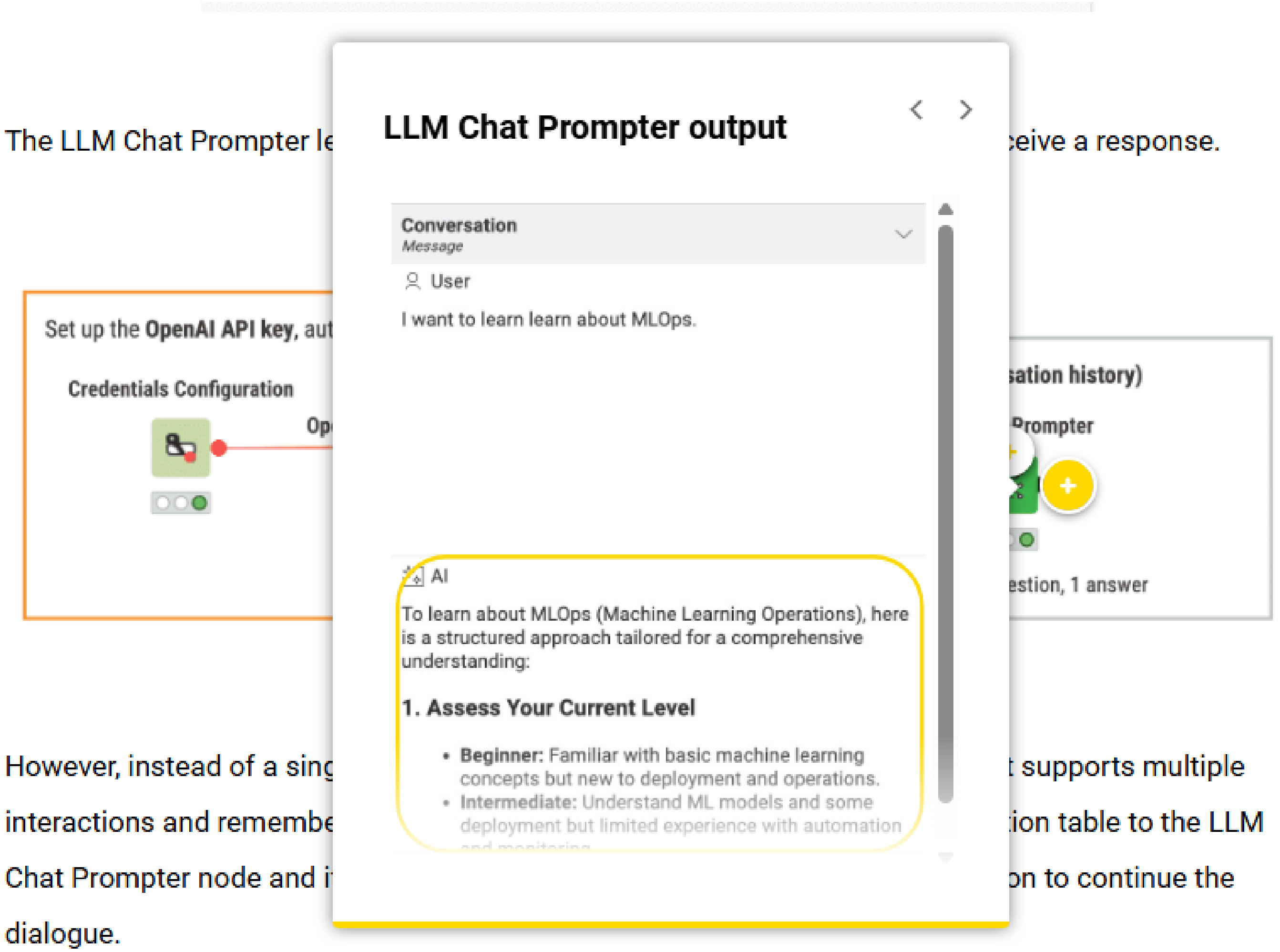
Output format

● Text ○ JSON

Conversation column name

Conversation

Responding to System Message and User Question



The LLM Chat Prompter le … ceive a response.

However, instead of a sing … t supports multiple interactions and remembe … ion table to the LLM Chat Prompter node and i … on to continue the dialogue.

LLM Chat Prompter output

Conversation
Message

User

I want to learn learn about MLOps.

AI

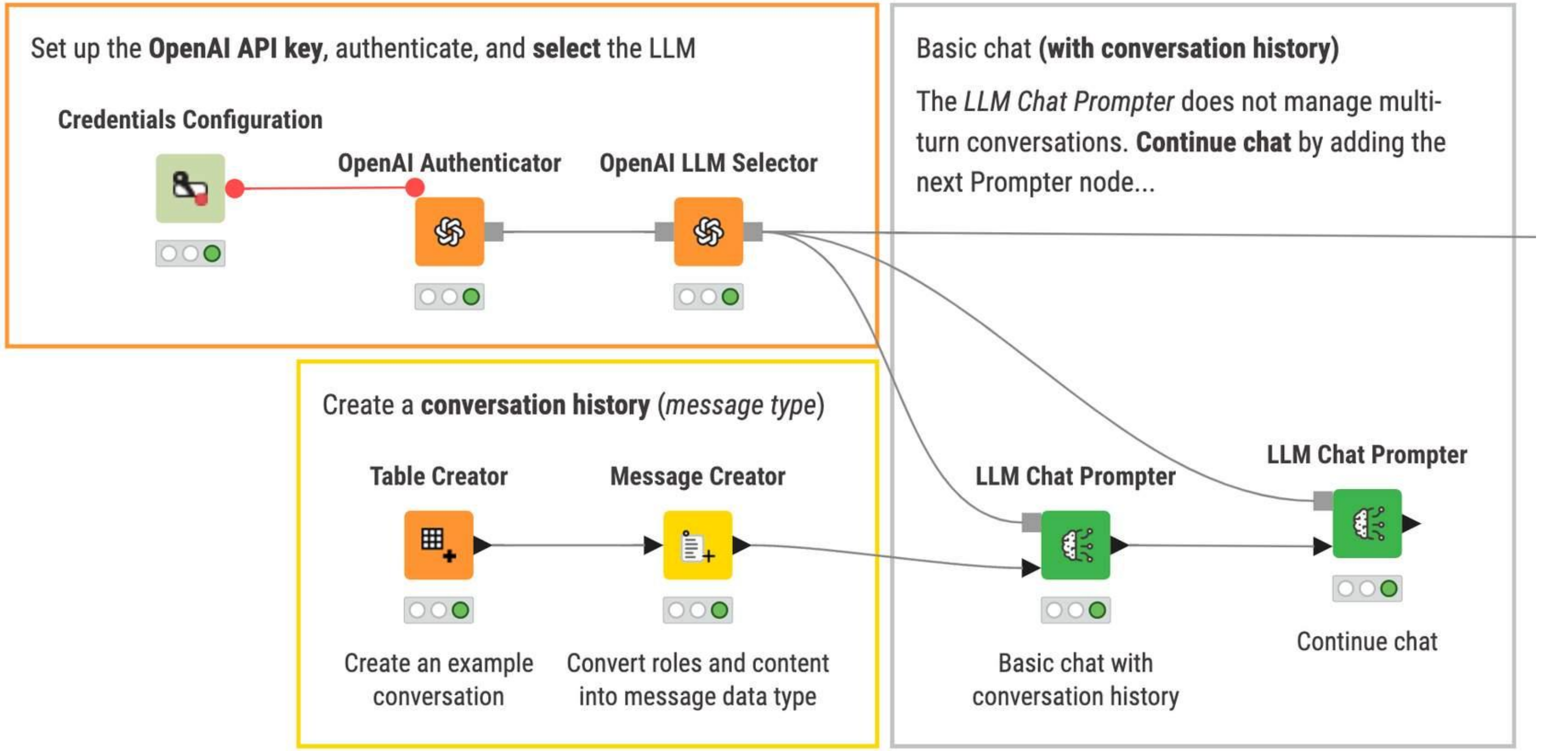
To learn about MLOps (Machine Learning Operations), here is a structured approach tailored for a comprehensive understanding:

1. Assess Your Current Level

- **Beginner:** Familiar with basic machine learning concepts but new to deployment and operations.
- **Intermediate:** Understand ML models and some deployment but limited experience with automation and monitoring

Providing more than one message (conversation history)

- Instead of a single question-answer exchange, we want a chatbot that supports multiple interactions and remembers the conversation history.
- We can feed a conversation table to the LLM Chat Prompter node, and it will return the output based on the whole conversation to continue the dialogue.



Set up the **OpenAI API key**, authenticate, and **select** the LLM

Create a **conversation history** (*message type*)

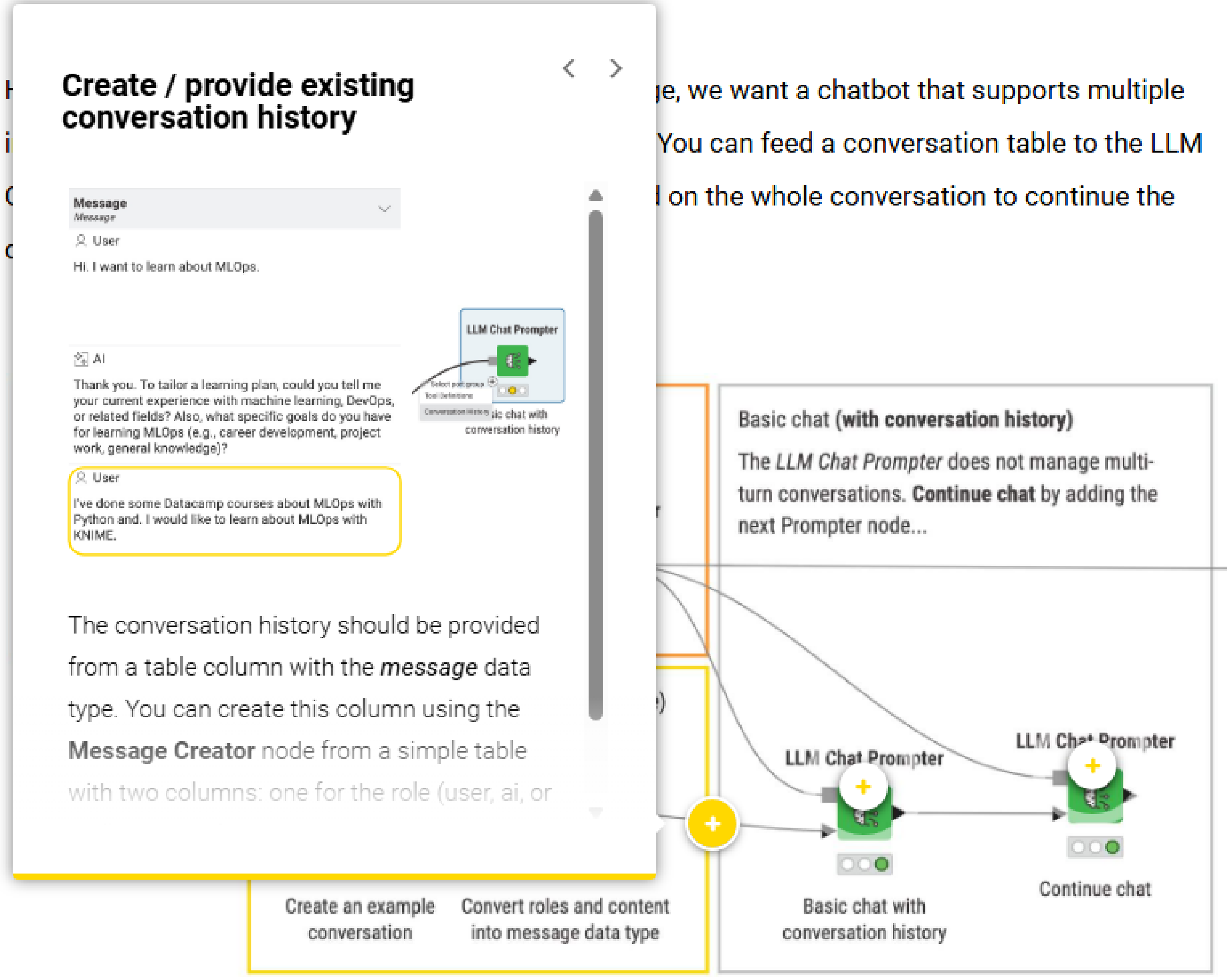
Basic chat **(with conversation history)**

The *LLM Chat Prompter* does not manage multi-turn conversations. **Continue chat** by adding the next Prompter node...

Credentials Configuration
OpenAI Authenticator
OpenAI LLM Selector
Table Creator
Create an example conversation
Message Creator
Convert roles and content into message data type
LLM Chat Prompter
Basic chat with conversation history
LLM Chat Prompter
Continue chat

Provide conversation history

- The conversation history should be provided from a table column with the *message* data type.
- W can create this column using the **Message Creator** node from a simple table with two columns: one for the role (user, ai, or tool) and one for the message content, ordered as a conversation.



Create / provide existing conversation history

Message
Message

User

Hi. I want to learn about MLOps.

AI

Thank you. To tailor a learning plan, could you tell me your current experience with machine learning, DevOps, or related fields? Also, what specific goals do you have for learning MLOps (e.g., career development, project work, general knowledge)?

User

I've done some Datacamp courses about MLOps with Python and. I would like to learn about MLOps with KNIME.

The conversation history should be provided from a table column with the *message* data type. You can create this column using the **Message Creator** node from a simple table with two columns: one for the role (user, ai, or

ge, we want a chatbot that supports multiple

You can feed a conversation table to the LLM

l on the whole conversation to continue the

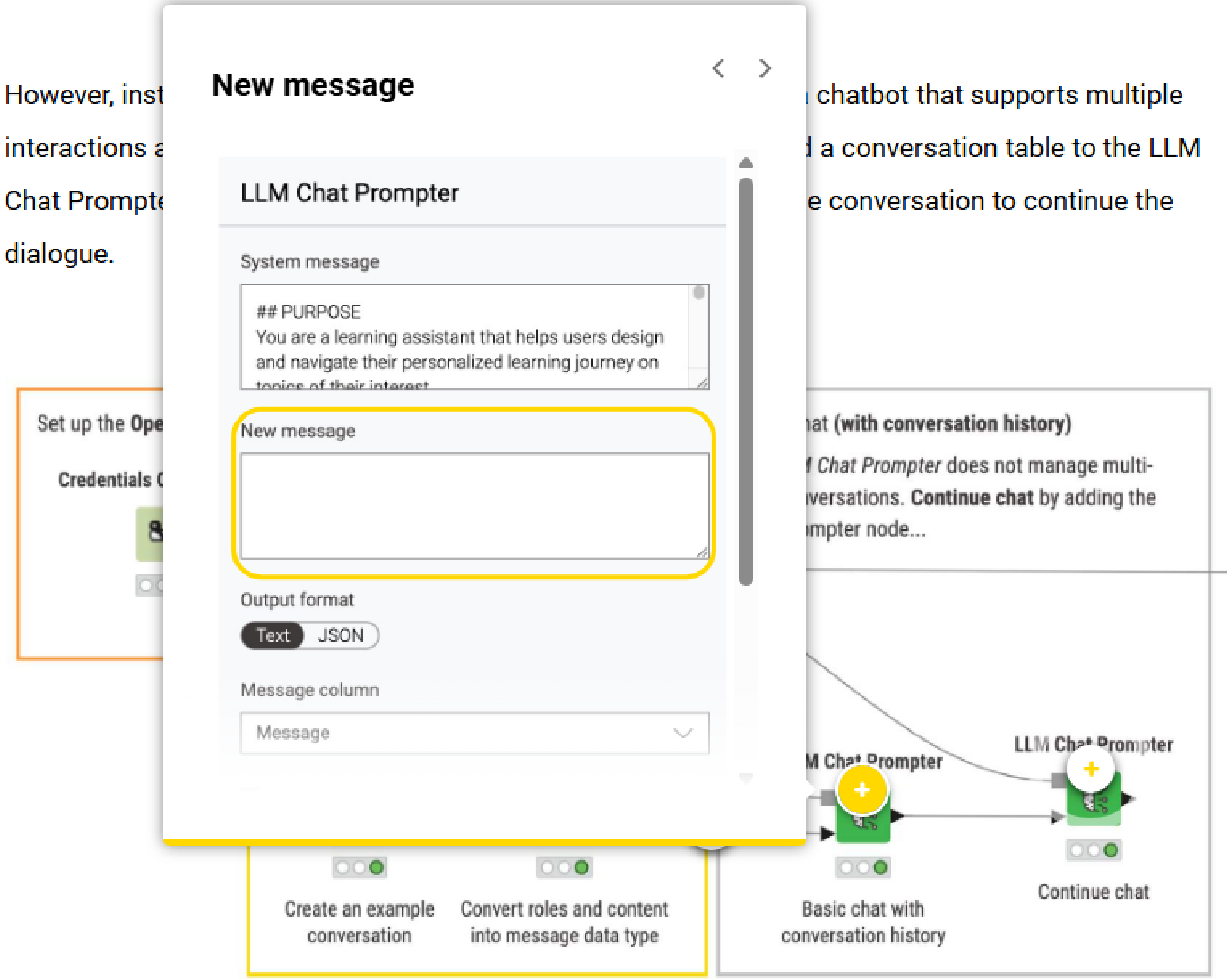
Basic chat **(with conversation history)**

The *LLM Chat Prompter* does not manage multi-turn conversations. **Continue chat** by adding the next Prompter node...

LLM Chat Prompter
Select port group
Tool Definitions
Conversation History
sic chat with conversation history
LLM Chat Prompter
LLM Chat Prompter
Create an example conversation
Convert roles and content into message data type
Basic chat with conversation history
Continue chat

Provide conversation history

Based on the last message in the conversation (from the user or AI), you can either leave the new message blank or enter the next message to continue the chat.



However, inst… chatbot that supports multiple interactions a… a conversation table to the LLM Chat Prompte… e conversation to continue the dialogue.

New message

LLM Chat Prompter

System message

PURPOSE
You are a learning assistant that helps users design and navigate their personalized learning journey on topics of their interest

New message

Output format

Text JSON

Message column

Message

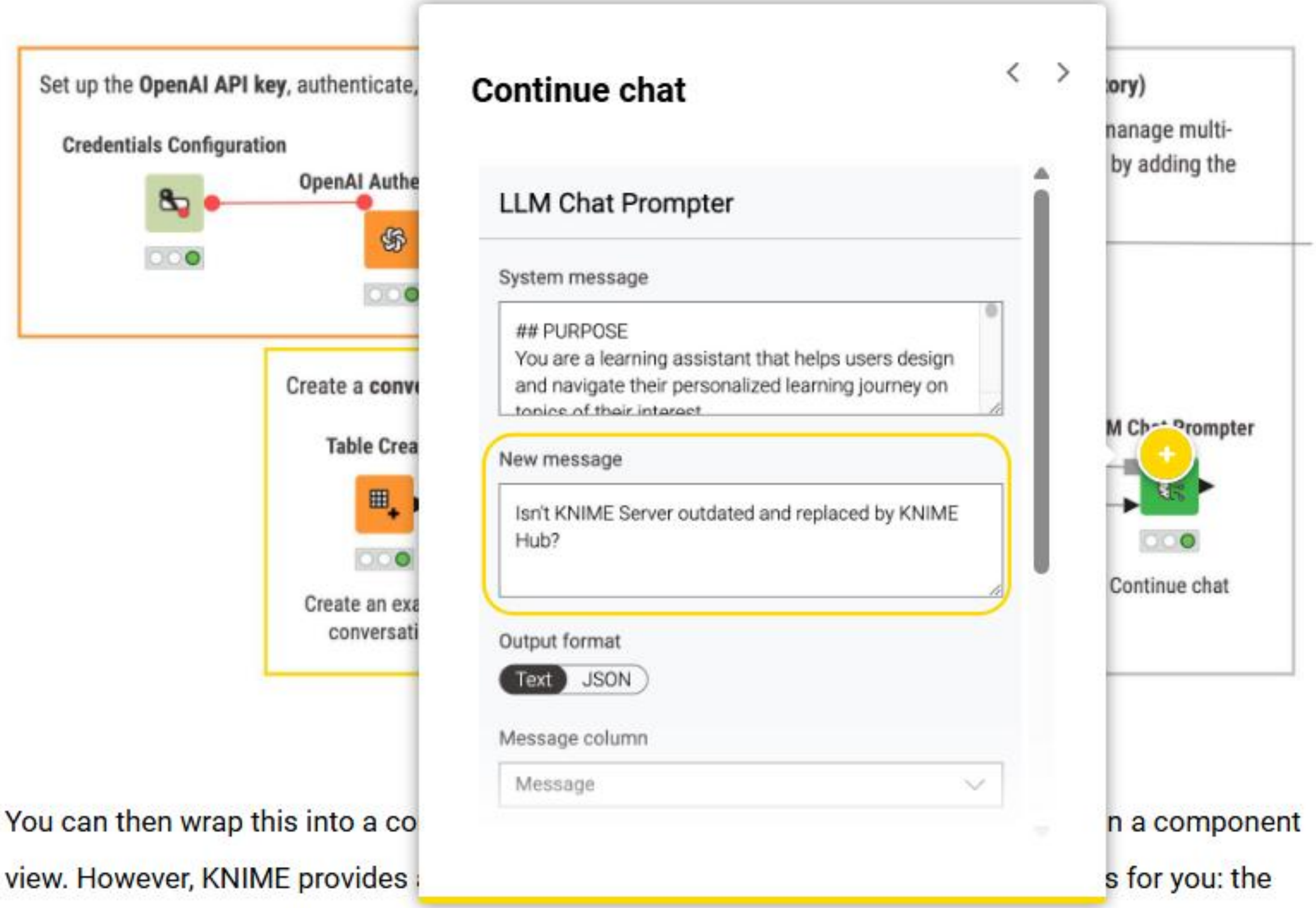
Set up the Ope
Credentials
…at (with conversation history)
…Chat Prompter does not manage multi-
…versations. Continue chat by adding the
…mpter node...
Create an example conversation
Convert roles and content into message data type
…M Chat Prompter
Basic chat with conversation history
LLM Chat Prompter
Continue chat

Provide conversation history

The LLM Chat Prompter alone does not support interactive multi-turn conversations.

To continue, chain another Prompter node with the new message.

This approach is limited, so check out how to build an interactive chatbot next in the lesson.



Set up the **OpenAI API key**, authenticate,

Credentials Configuration

OpenAI Authe

Create a conv

Table Crea

Create an exa
conversati

ory)
nanage multi-
by adding the

M Chat Prompter

Continue chat

You can then wrap this into a co … n a component
view. However, KNIME provides … s for you: the

Continue chat

LLM Chat Prompter

System message

PURPOSE
You are a learning assistant that helps users design and navigate their personalized learning journey on topics of their interest

New message

Isn't KNIME Server outdated and replaced by KNIME Hub?

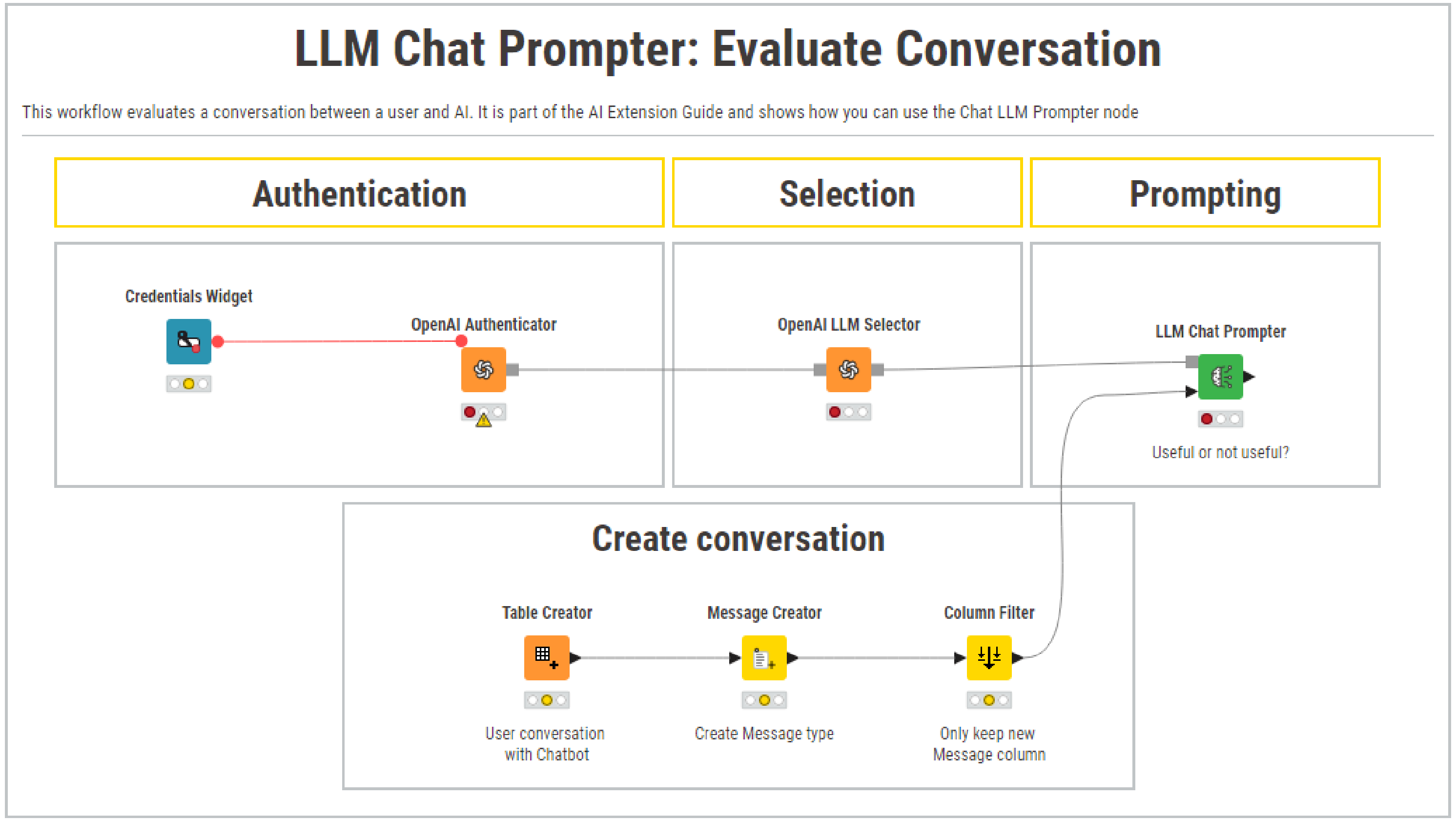
Output format

● Text ○ JSON

Message column

Message

Chat Prompter Example



LLM Chat Prompter: Evaluate Conversation

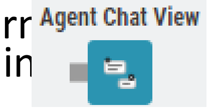
This workflow evaluates a conversation between a user and AI. It is part of the AI Extension Guide and shows how you can use the Chat LLM Prompter node

This workflow can be downloaded as following:

1. Download Course Workflows from VClass
2. Goto Generative AI Folder -> AI Extension Guide -> 1. Prompting LLM
3. Open Chat LLM Prompter

Agent Chat View

- This node automates conversation history handling and provides a dynamic chat interface that supports multi-turr conversations with an AI agent, allowing us to build a fully interactive chatbot with minimal setup. It also combinin the chat model with a set of tools and optional input data.



- The agent is assembled from the provided chat model and tools; each defined as a KNIME workflow.
- Tools can include configurable parameters (e.g., string inputs, numeric settings, column selectors) and may optionally consume input data in the form of KNIME tables.
- While the agent does not access raw data directly, it is informed about the structure of available tables (i.e., column names and types). This allows the model to select and route data to tools during conversation.
- Unlike the standard Agent Prompter node, which executes a single user prompt, this node supports multi-turn, interactive dialogue.
- The user can iteratively send prompts and receive responses, with the agent invoking tools as needed in each conversational turn.
- Tool outputs from earlier turns can be reused in later interactions, enabling rich, context-aware workflows.
- This node is designed for real-time, interactive usage and does not produce a data output port. Instead, the conversation takes place directly within the KNIME view, where the agent’s responses and reasoning are shown incrementally as the dialogue progresses.
- To ensure effective agent behavior, provide meaningful tool names and clear descriptions — including example use cases if applicable.

Deploying the Chatbot

- Once we develop our own chatbot, we’ll likely want to run it within our own infrastructure and make it accessible to our users.
- While the KNIME Analytics Platform enables us to build AI and data applications like chatbots, KNIME Hub supports us in collaborating on, deploying, and governing them.
- On KNIME Hub, we can deploy our custom chatbot as an interactive data app, making it accessible to users either through a publicly shared link or via the Data Apps portal.

Retrieval-Augmented Generation & Fine Tuning

Overcoming limited knowledge in LLMs

Despite the impressive ability to generate fluent, coherent responses across a wide range of topics, LLMs don’t know everything. Why? Because they are trained on general, static datasets, meaning their knowledge is fixed at the point of training. As a result, their responses are limited to what they’ve “seen” during that training phase.

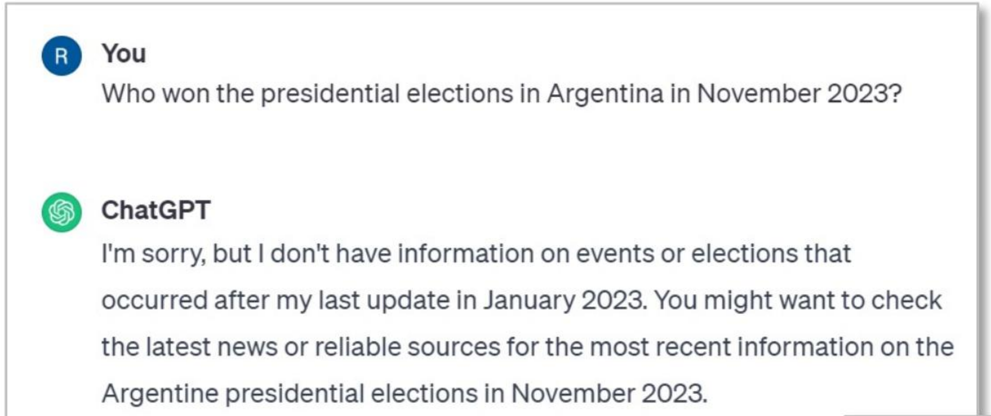
This creates limitations when LLMs are asked about:

- Recent events
- Proprietary or internal data
- Highly domain-specific information

In these cases, LLMs may produce hallucinations - responses that sound plausible but are actually incorrect or misleading.

Hallucination (in AI)

- A response generated by an LLM that contains false or misleading information presented as fact, often with high confidence.



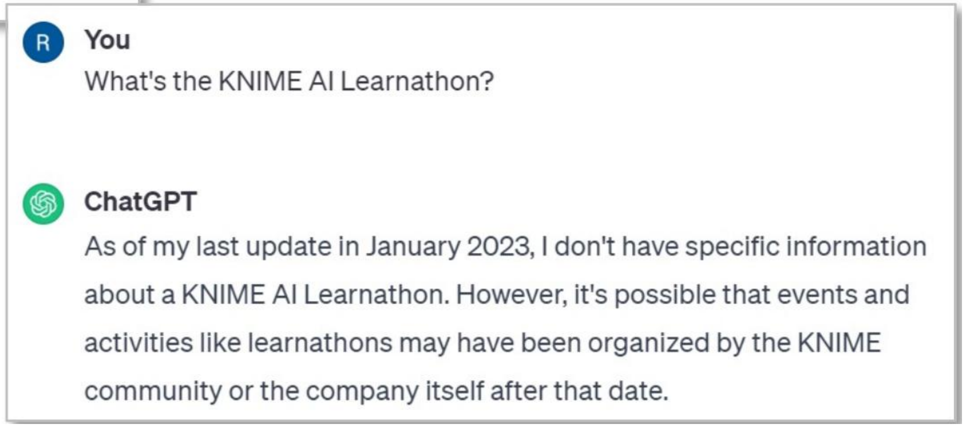
You

Who won the presidential elections in Argentina in November 2023?

ChatGPT

I'm sorry, but I don't have information on events or elections that occurred after my last update in January 2023. You might want to check the latest news or reliable sources for the most recent information on the Argentine presidential elections in November 2023.

Lack of up-to-date information



You

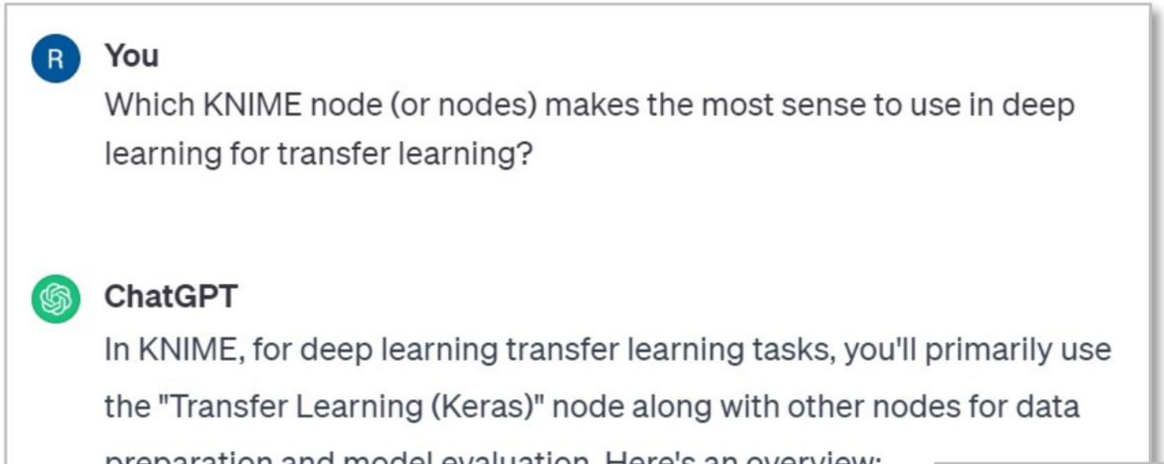
What's the KNIME AI Learnathon?

ChatGPT

As of my last update in January 2023, I don't have specific information about a KNIME AI Learnathon. However, it's possible that events and activities like learnathons may have been organized by the KNIME community or the company itself after that date.

Not much we can do with prompt engineering or hyperparameter tuning

Hallucination (in AI)



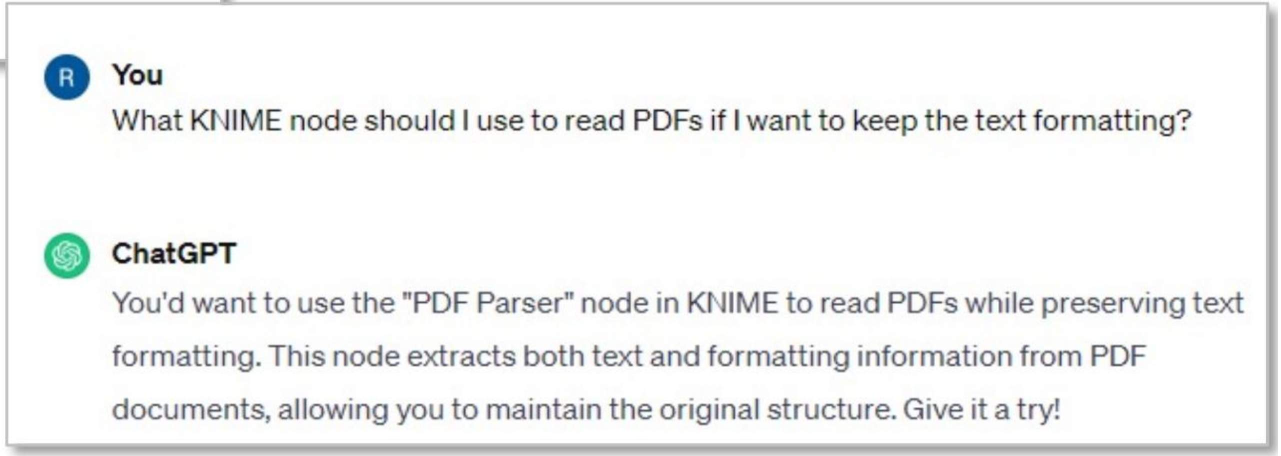
You

Which KNIME node (or nodes) makes the most sense to use in deep learning for transfer learning?

ChatGPT

In KNIME, for deep learning transfer learning tasks, you'll primarily use the "Transfer Learning (Keras)" node along with other nodes for data preparation and model evaluation. Here's an overview:

Not much we can do with prompt engineering or hyperparameter tuning



You

What KNIME node should I use to read PDFs if I want to keep the text formatting?

ChatGPT

You'd want to use the "PDF Parser" node in KNIME to read PDFs while preserving text formatting. This node extracts both text and formatting information from PDF documents, allowing you to maintain the original structure. Give it a try!

Lack of domain -specific knowledge, leading to hallucinations or factually wrong information

- Tools like ChatGPT can overcome some of these limitations because they can access web search in real time. But when you access LLMs programmatically, via API or local models, e.g., in a KNIME workflow, they do not have access to external or updated data.
- To enhance the accuracy and relevance of LLM outputs, several approaches can be used:
  - Retrieval-Augmented Generation (RAG)
  - Fine-tuning

Retrieval-Augmented Generation (RAG)

What is RAG?

- RAG stands for Retrieval-Augmented Generation. It is one of the simplest and most cost-effective techniques to overcome the limitations of LLMs' fixed knowledge.
- Rather than retraining the model, RAG works by augmenting the prompt with information retrieved automatically from an external knowledge base, based on the user’s query.
- No retraining or fine-tuning is happening in RAG. The original LLM remains unchanged.
- At a high level, a RAG pipeline consists of these main elements:
  - Retrieval. Identify and retrieve relevant information from a knowledge base based on the input query.
  - Augmentation. Augment the original query or prompt with the retrieved information. This provides the model with additional context to better understand the task.
  - Generation. Generate a more accurate and context-aware response using the augmented prompt.

Embeddings and vector stores

- To understand how RAG works, we first need to understand two concepts essential for automated retrieval of textual information: embeddings and vector stores.
- **Embeddings and vector stores**
- In RAG, our main goal is to *automatically* retrieve the most relevant pieces of information from a potentially large and unstructured free-text knowledge base.
- Why not just pass the entire knowledge base in the prompt to the LLM?
  - It might not fit within the LLM’s context length.
  - It can increase computational costs.
  - It may make the prompt noisier and reduce response quality.
- So instead, the system needs to search and select only the most relevant snippets of information - automatically.

Word Embeddings

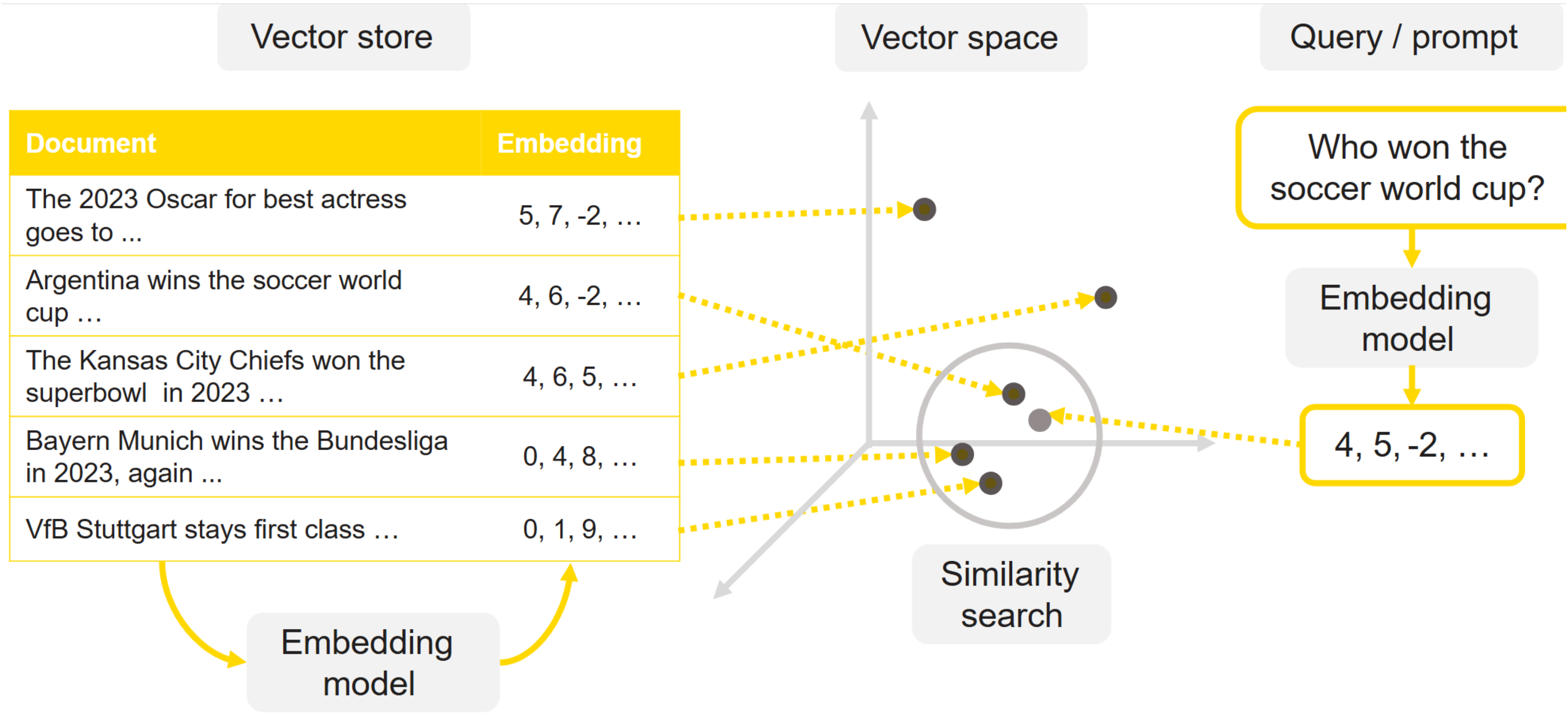
- For an automated search of relevant information, the free-text data needs to be converted into a mathematical form that can be compared. That form is embeddings.
- **Embeddings are** high-dimensional vector representations of text, where the position of each vector reflects the semantic meaning of the text it represents. Similar meanings are represented by similar vectors.
- An embedding model processes each chunk of our knowledge base and converts it into a high-dimensional vector. These vectors are positioned in a semantic vector space such that:
  - **Similar content** is placed **closer together**,
  - **Dissimilar content** is placed **further apart**.
- Once our knowledge base is embedded, the vectors need to be stored in a way that allows for efficient retrieval. That's the job of a vector store.

Vector store

- A database that stores embeddings and supports similarity search using vector distance metrics.
- So how do embeddings and vector stores enable automated retrieval of the relevant information? When a user submits a query:
  - The query is converted into an embedding using the same embedding model.
  - The vector store compares the query embedding to all stored document embeddings.
  - It returns the top k most similar vectors, i.e., the text chunks most relevant to the query.

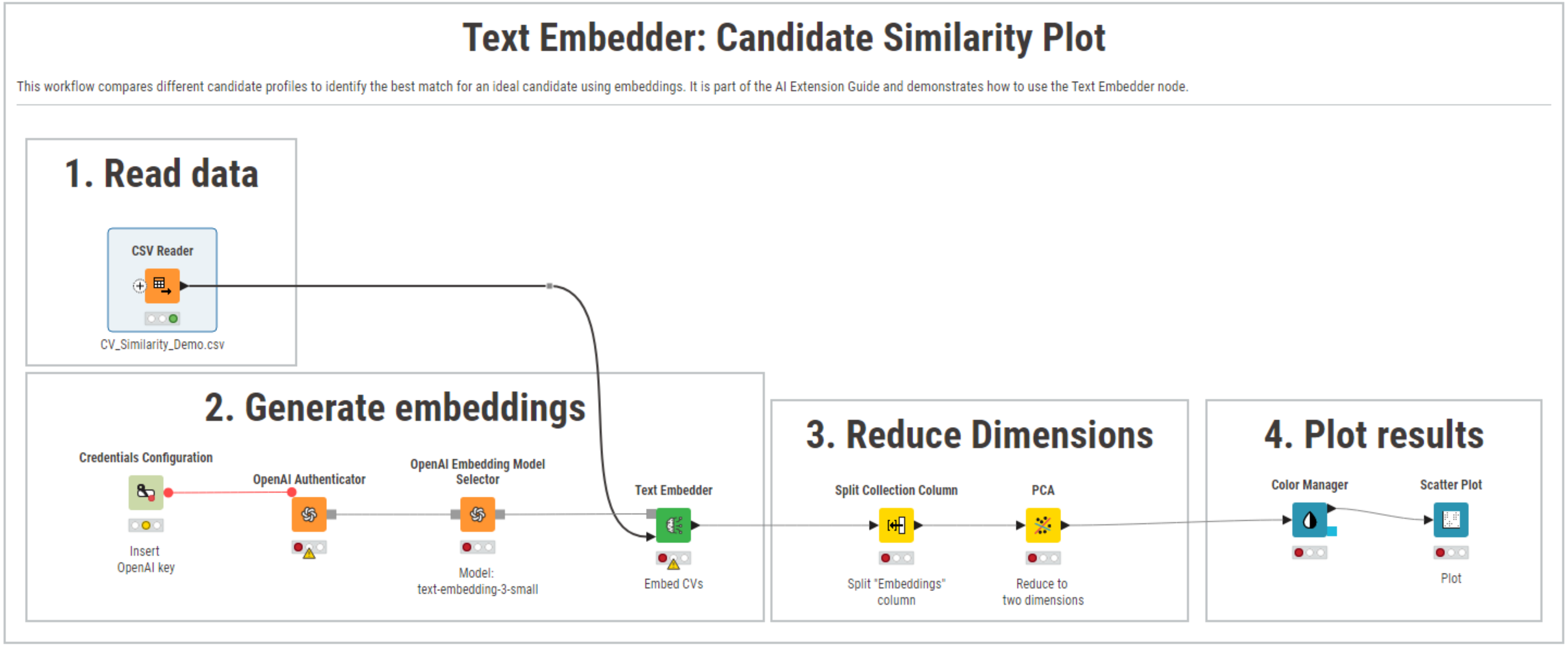
Embeddings and Vector Stores

- Explore the image below to see how embeddings and vector stores work together to enable automated search over unstructured text.



Document	Embedding
The 2023 Oscar for best actress goes to ...	5, 7, -2, …
Argentina wins the soccer world cup …	4, 6, -2, …
The Kansas City Chiefs won the superbowl in 2023 …	4, 6, 5, …
Bayern Munich wins the Bundesliga in 2023, again ...	0, 4, 8, …
VfB Stuttgart stays first class …	0, 1, 9, …

Text Embedding Example



Text Embedder: Candidate Similarity Plot

This workflow compares different candidate profiles to identify the best match for an ideal candidate using embeddings. It is part of the AI Extension Guide and demonstrates how to use the Text Embedder node.

This workflow can be downloaded as following:

1. Download Course Workflows from VClass
2. Goto Generative AI Folder -> AI Extension Guide -> 1. Prompting LLM
3. Open Text Embedder

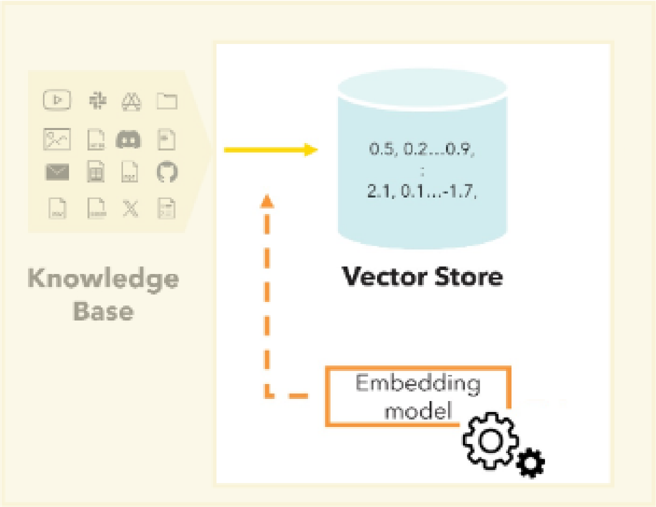
5 steps to a RAG pipeline – Step 1

Build knowledge base

Your knowledge base can include user-curated, domain-specific information in various formats. It may consist of structured (e.g., databases) or unstructured (e.g., PDFs, documents, web pages) content.

5 steps to a RAG pipeline – Step 2

Populate the vector store

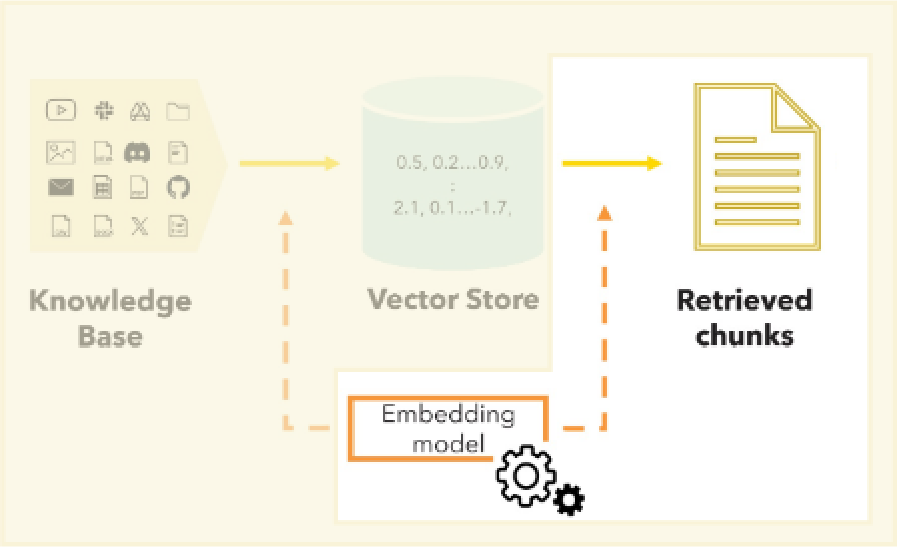


To prepare your knowledge base for retrieval:

1. Split the text into chunks. Whole documents cannot be processed at once. Chunking to smaller, manageable pieces enables efficient indexing and retrieval, improves semantic search, avoids exceeding context length limits, and context dilution in the prompt.
2. Generate embeddings. Use an embedding model to convert each chunk into a vector, then store these vectors in a vector store for fast retrieval.

5 steps to a RAG pipeline – Step 3

Automatically retrieve relevant chunks

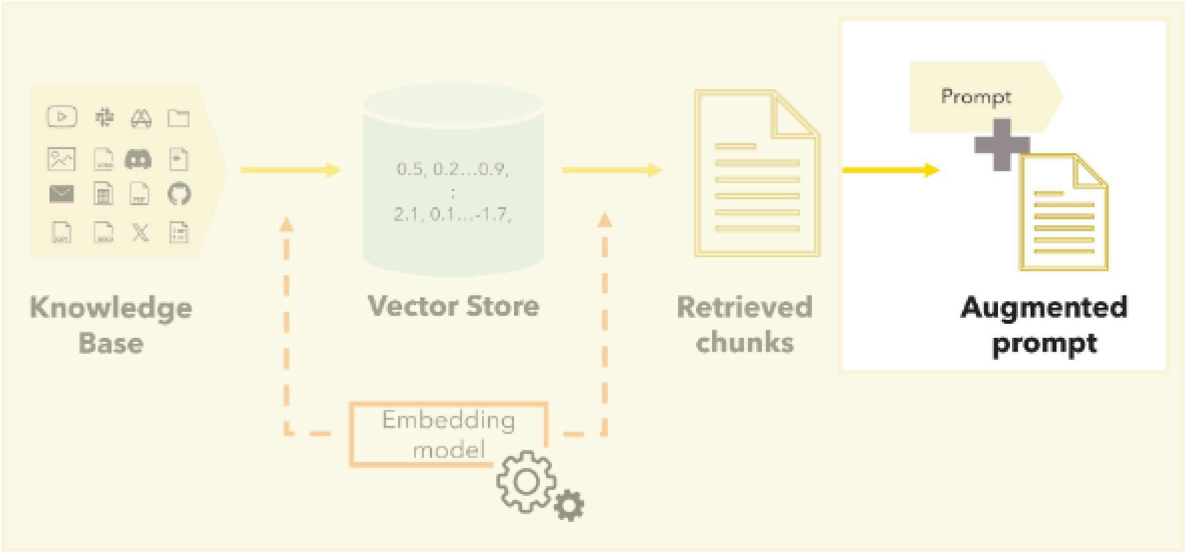


1. Convert the user query or initial prompt into an embedding using the same model.

2. Use the vector store to perform similarity search and retrieve the most relevant chunks - those with the highest semantic similarity to the query.

5 steps to a RAG pipeline – Step 4

Augment prompt

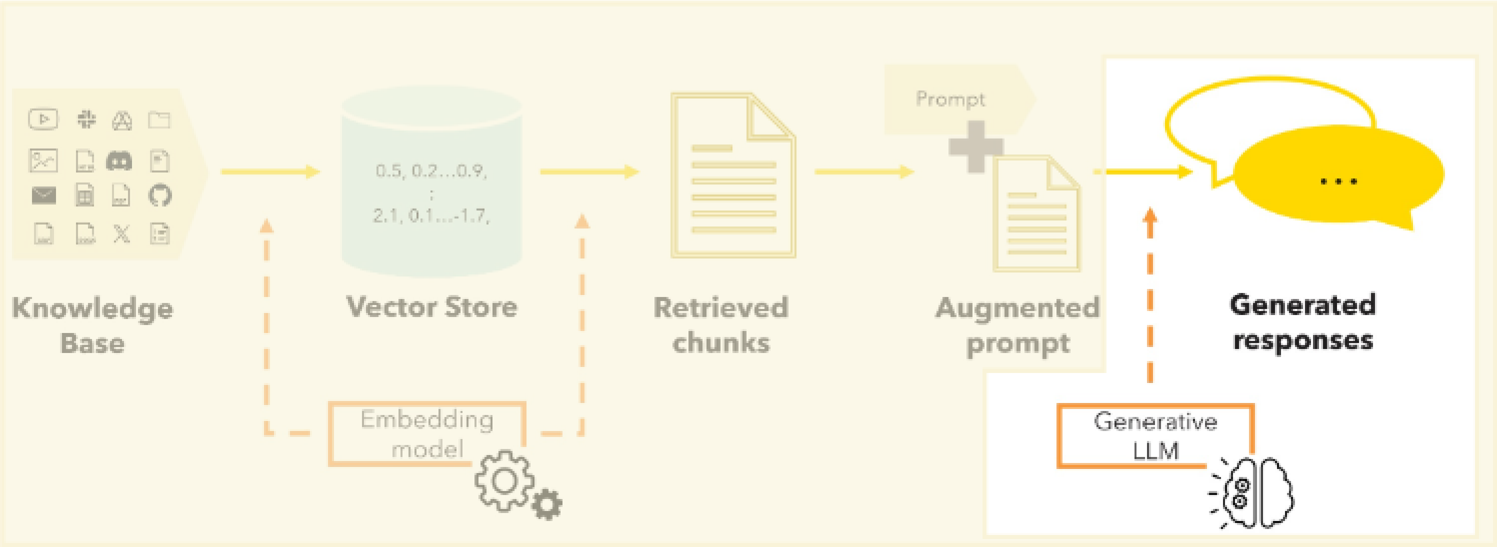


Enhance the original prompt by adding the retrieved chunks. Include a guiding instruction like:

> *“Here is the context relevant to your task: <retrieved chunks>.”*

5 steps to a RAG pipeline – Step 5

Generate responses



Let the LLM generate the final response as usual but now with a prompt contextually enriched for the domain-specific tasks.

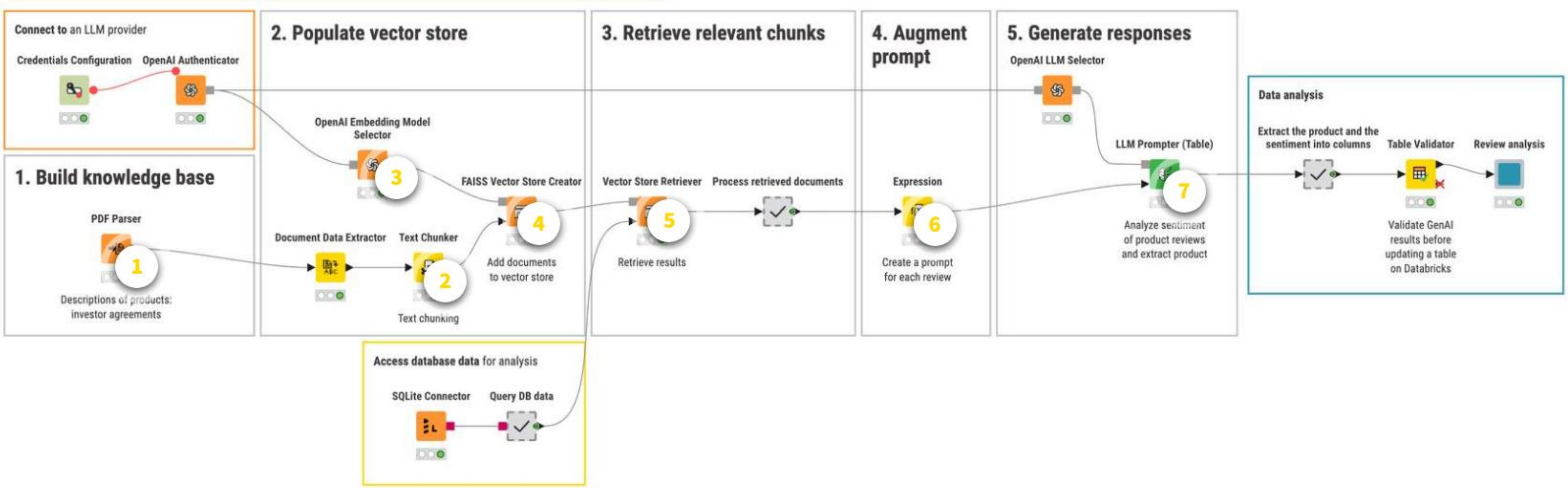
RAG in KNIME

Standard RAG pipeline

- KNIME provides all necessary components to build a complete RAG pipeline, including the ability to:
- Import documents in various formats from your knowledge base,
- Connect to embedding models, whether local or API-based,
- Retrieve relevant information from the vector stores, augment prompt, and generate the response.

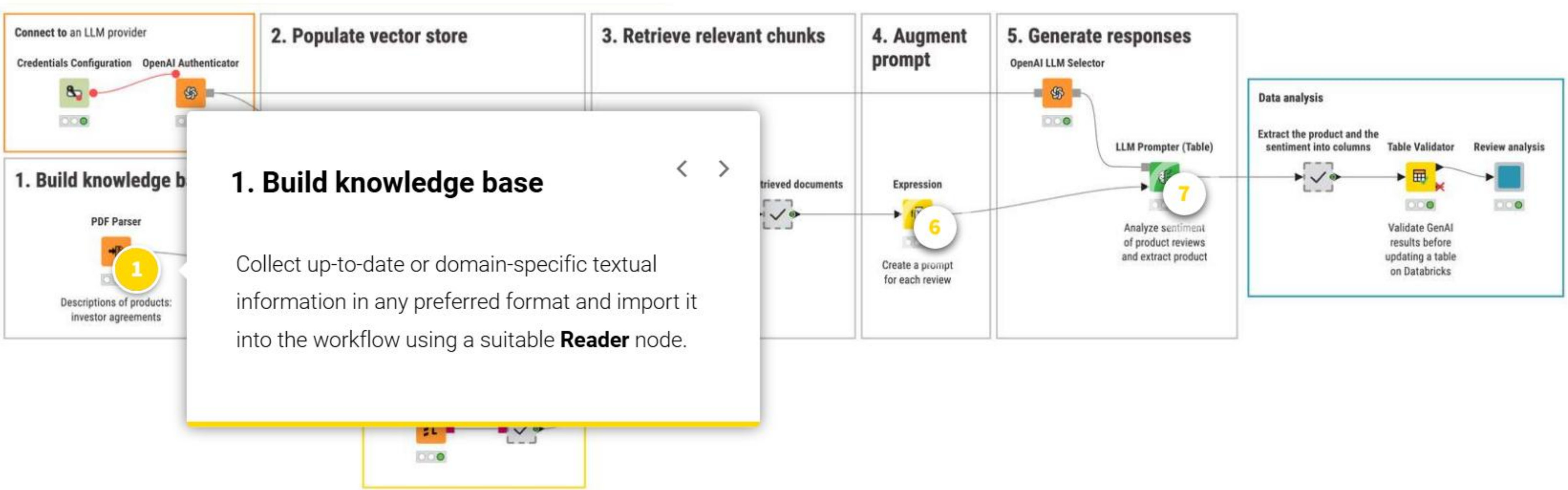
RAG in KNIME Example

Click the buttons below to explore how to implement a RAG workflow in KNIME, using an example that analyzes product reviews with highly specialized, domain-specific language.


Connect to an LLM provider
Credentials Configuration
OpenAI Authenticator
1. Build knowledge base
PDF Parser
1
Descriptions of products: investor agreements
2. Populate vector store
OpenAI Embedding Model Selector
3
FAISS Vector Store Creator
4
Add documents to vector store
Document Data Extractor
Text Chunker
2
Text chunking
3. Retrieve relevant chunks
Vector Store Retriever
Process retrieved documents
5
Retrieve results
4. Augment prompt
Expression
6
Create a prompt for each review
5. Generate responses
OpenAI LLM Selector
LLM Prompter (Table)
7
Analyze sentiment of product reviews and extract product
Data analysis
Extract the product and the sentiment into columns
Table Validator
Review analysis
Validate GenAI results before updating a table on Databricks
Access database data for analysis
SQLite Connector
Query DB data

Collect Relevant Data

Collect up-to-date or domain-specific textual information in any preferred format and import it into the workflow using a suitable **Reader** node.



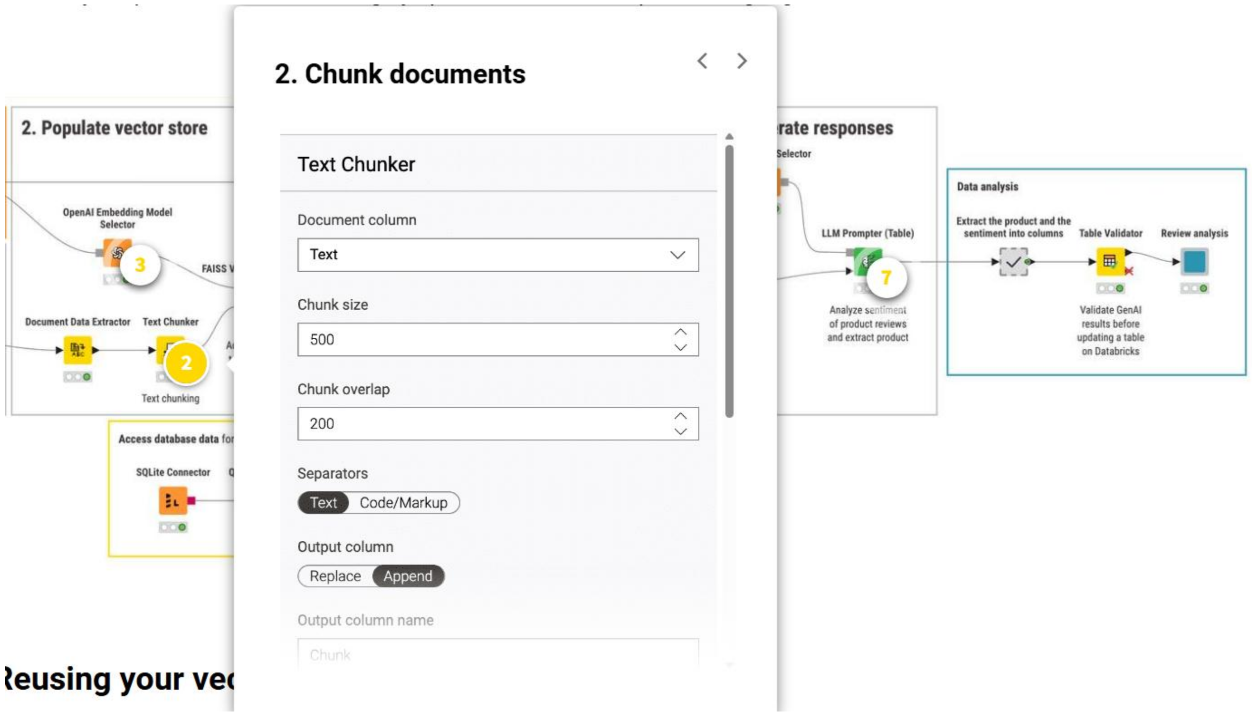
Connect to an LLM provider
Credentials Configuration
OpenAI Authenticator
1. Build knowledge b
PDF Parser
1
Descriptions of products: investor agreements
2. Populate vector store
3. Retrieve relevant chunks
trieved documents
4. Augment prompt
Expression
6
Create a prompt for each review
5. Generate responses
OpenAI LLM Selector
LLM Prompter (Table)
7
Analyze sentiment of product reviews and extract product
Data analysis
Extract the product and the sentiment into columns
Table Validator
Review analysis
Validate GenAI results before updating a table on Databricks

1. Build knowledge base

Collect up-to-date or domain-specific textual information in any preferred format and import it into the workflow using a suitable **Reader** node.

Chunk Documents

- If the knowledge base is a large document, split it into meaningful chunks for retrieval (not too short, not too long), taking into account the LLM’s context window.
- We can use nodes such as Text Chunker or Sentence Extractor for this step.
- We can use the Text Chunker node to create chunks automatically while keeping semantic relations and considering formatting language syntax.



2. Chunk documents

Text Chunker

Document column: Text

Chunk size: 500

Chunk overlap: 200

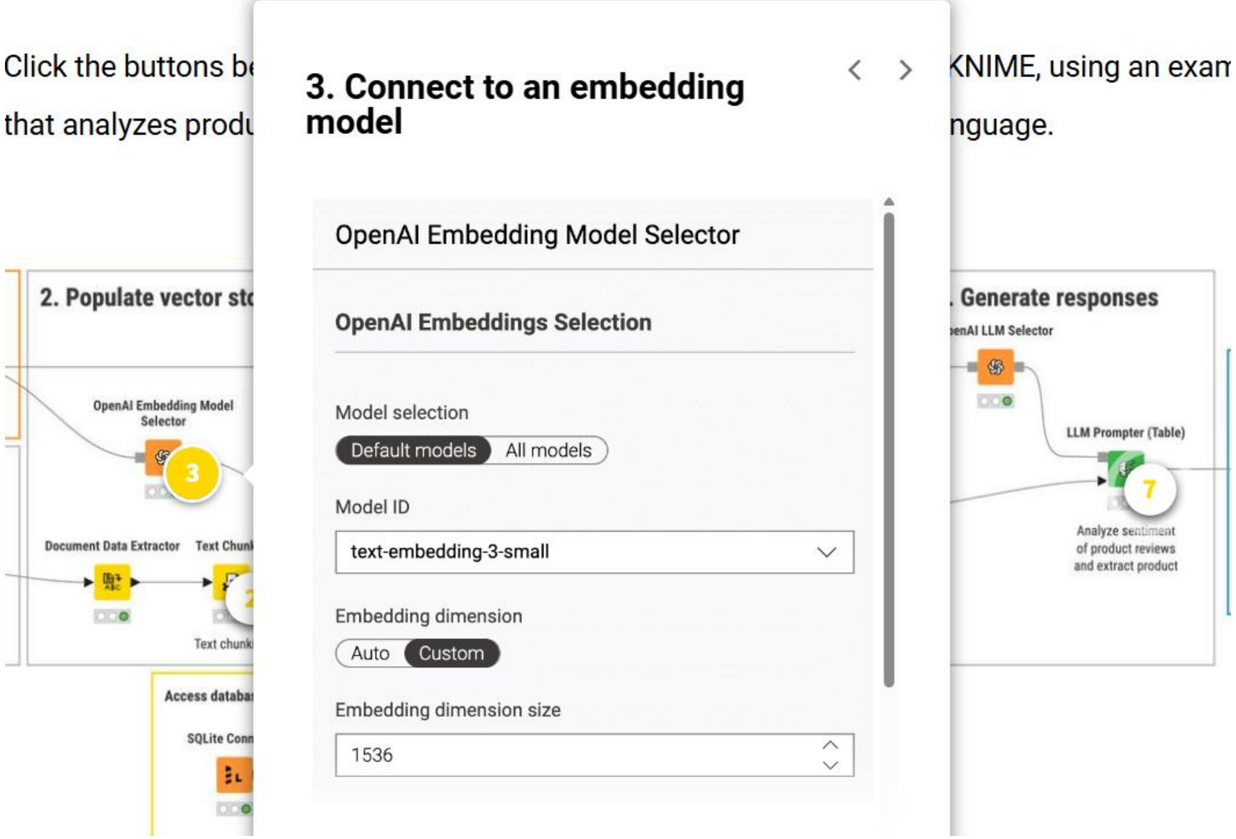
Separators: [✓ Text] Code/Markup

Output column: Replace [✓ Append]

Output column name: Chunk

Connect to an Embedding Model

- Connect to our LLM provider and select an embedding model of your choice using a suitable **Embedding Model Selector** node.
- Alternatively, connect to a local embedding model using the **GPT4All Embedding Model Selector**.



3. Connect to an embedding model

OpenAI Embedding Model Selector

OpenAI Embeddings Selection

Model selection

[✓ Default models] All models

Model ID

text-embedding-3-small

Embedding dimension

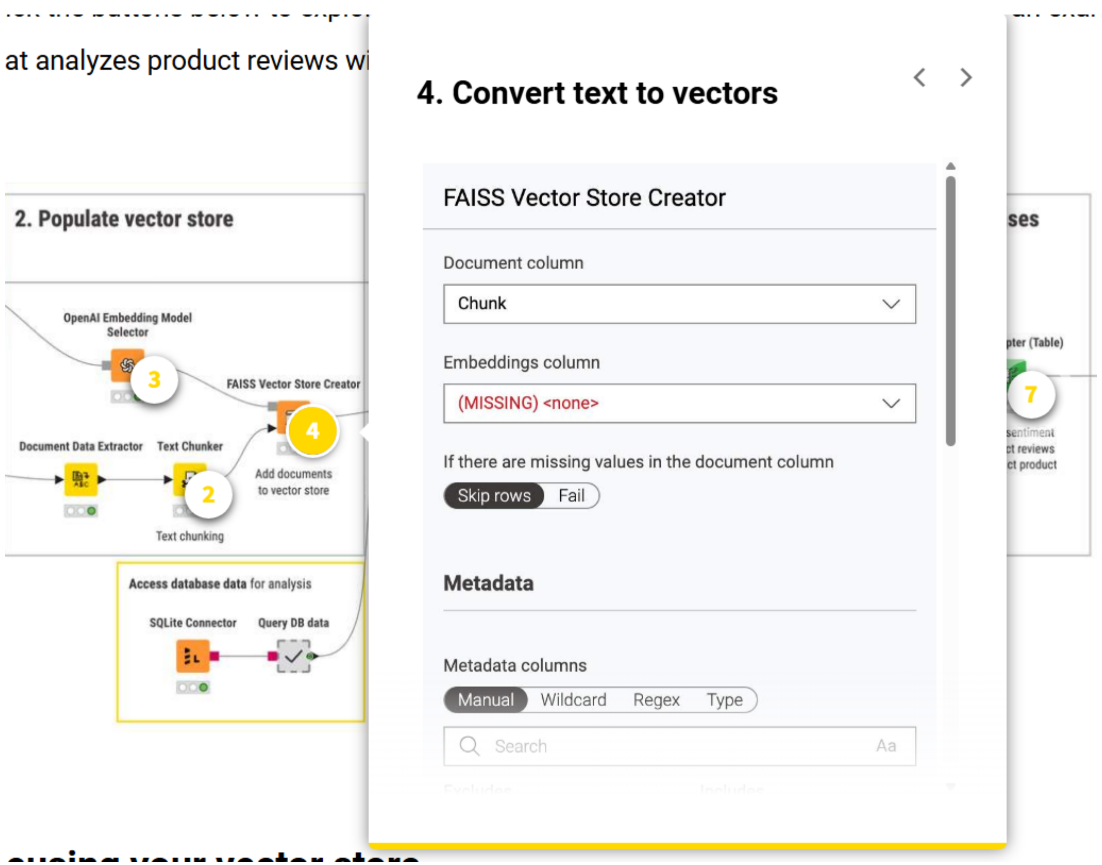
Auto [✓ Custom]

Embedding dimension size

1536

Create Vector Store

- Depending on the vector store (e.g., FAISS, Chroma), use the appropriate **Vector Store Creator** node to convert each text chunk into a high-dimensional vector with the selected embedding model.
- The original text data will also be stored, and optionally, we can include relevant metadata to help refine retrieval during later queries.



at analyzes product reviews wi

4. Convert text to vectors

FAISS Vector Store Creator

Document column: Chunk

Embeddings column: (MISSING) <none>

If there are missing values in the document column: Skip rows | Fail

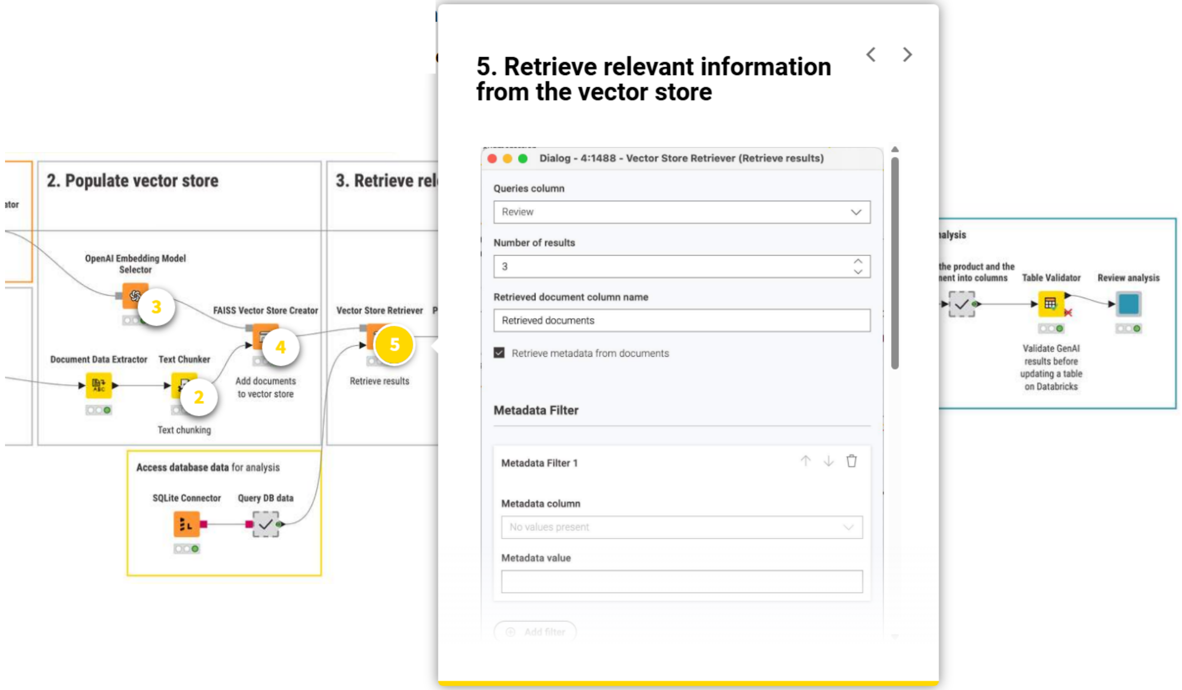
Metadata

Metadata columns: Manual | Wildcard | Regex | Type

Search

Retrieve relevant content from the vector store

- For a query that will be part of the prompt (or the initial, non-augmented prompt), retrieve the most relevant chunks from the knowledge base automatically using vector-based similarity search.
- Use the **Vector Store Retriever** node, regardless of the vector store used. We can:
  - We can specify how many documents to extract—they will be returned as a list.
  - Apply filters to narrow results based on metadata



5. Retrieve relevant information from the vector store

Dialog - 4:1488 - Vector Store Retriever (Retrieve results)

Queries column: Review

Number of results: 3

Retrieved document column name: Retrieved documents

☑ Retrieve metadata from documents

Metadata Filter

Metadata Filter 1

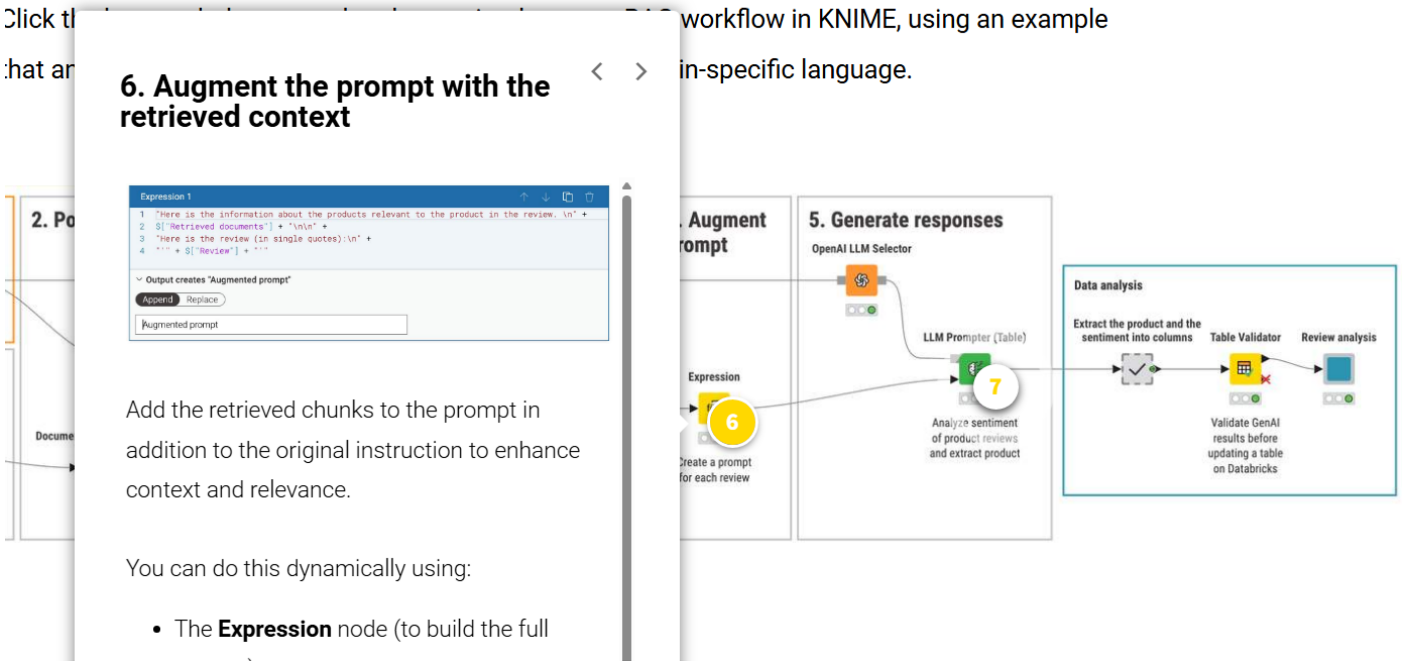
Metadata column: No values present

Metadata value:

ator
2. Populate vector store
OpenAI Embedding Model Selector
3
FAISS Vector Store Creator
4
Add documents to vector store
Document Data Extractor
Text Chunker
2
Text chunking
3. Retrieve rel
Vector Store Retriever
5
Retrieve results
Access database data for analysis
SQLite Connector
Query DB data
alysis
the product and the ent into columns
Table Validator
Review analysis
Validate GenAI results before updating a table on Databricks
Add filter

Augment the prompt

- Add the retrieved chunks to the prompt in addition to the original instruction to enhance context and relevance.
- We can do this dynamically using:
  - The **Expression** node (to build the full prompt),
  - The **Message Creator** node (to combine original prompt and retrieved chunks).



Click t… workflow in KNIME, using an example

hat ar… in-specific language.

6. Augment the prompt with the retrieved context

Expression 1

```
"Here is the information about the products relevant to the product in the review. \n" +
$["Retrieved documents"] + "\n\n" +
"Here is the review (in single quotes):\n" +
"'" + $["Review"] + "'"
```

Output creates "Augmented prompt"

Append / Replace: Append

Augmented prompt

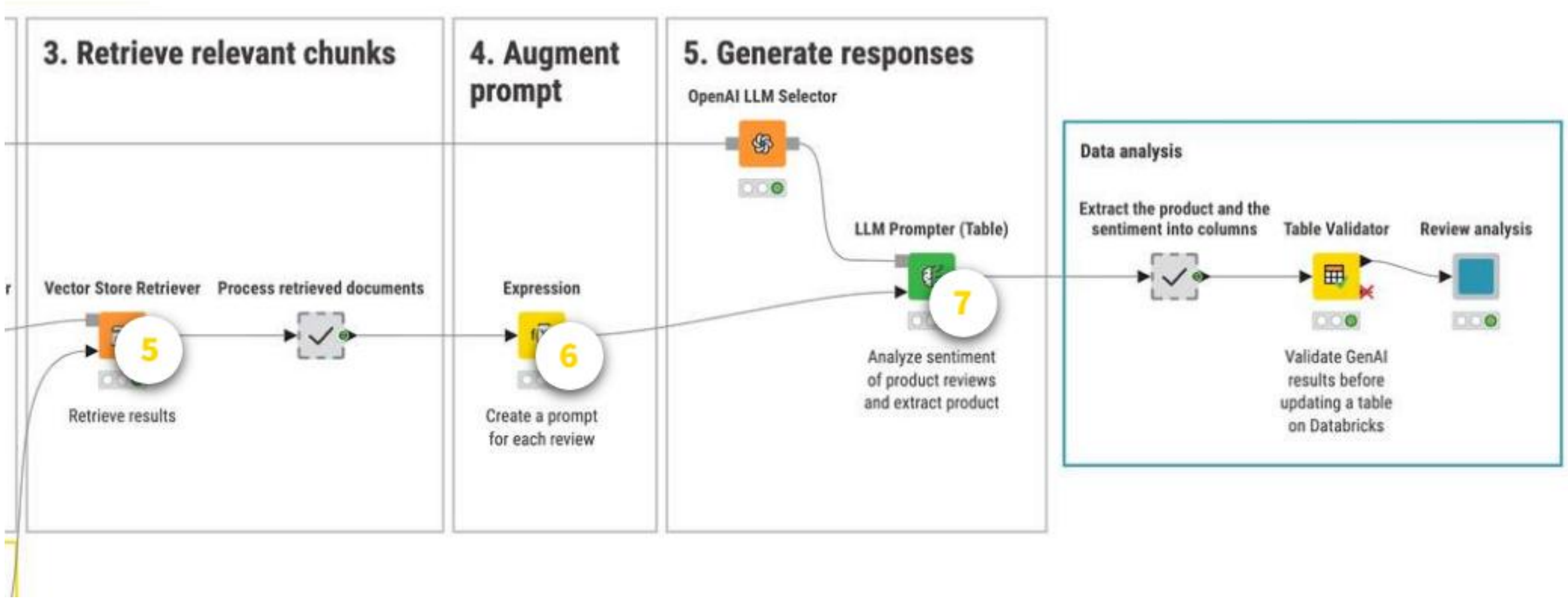
Add the retrieved chunks to the prompt in addition to the original instruction to enhance context and relevance.

You can do this dynamically using:

- The **Expression** node (to build the full

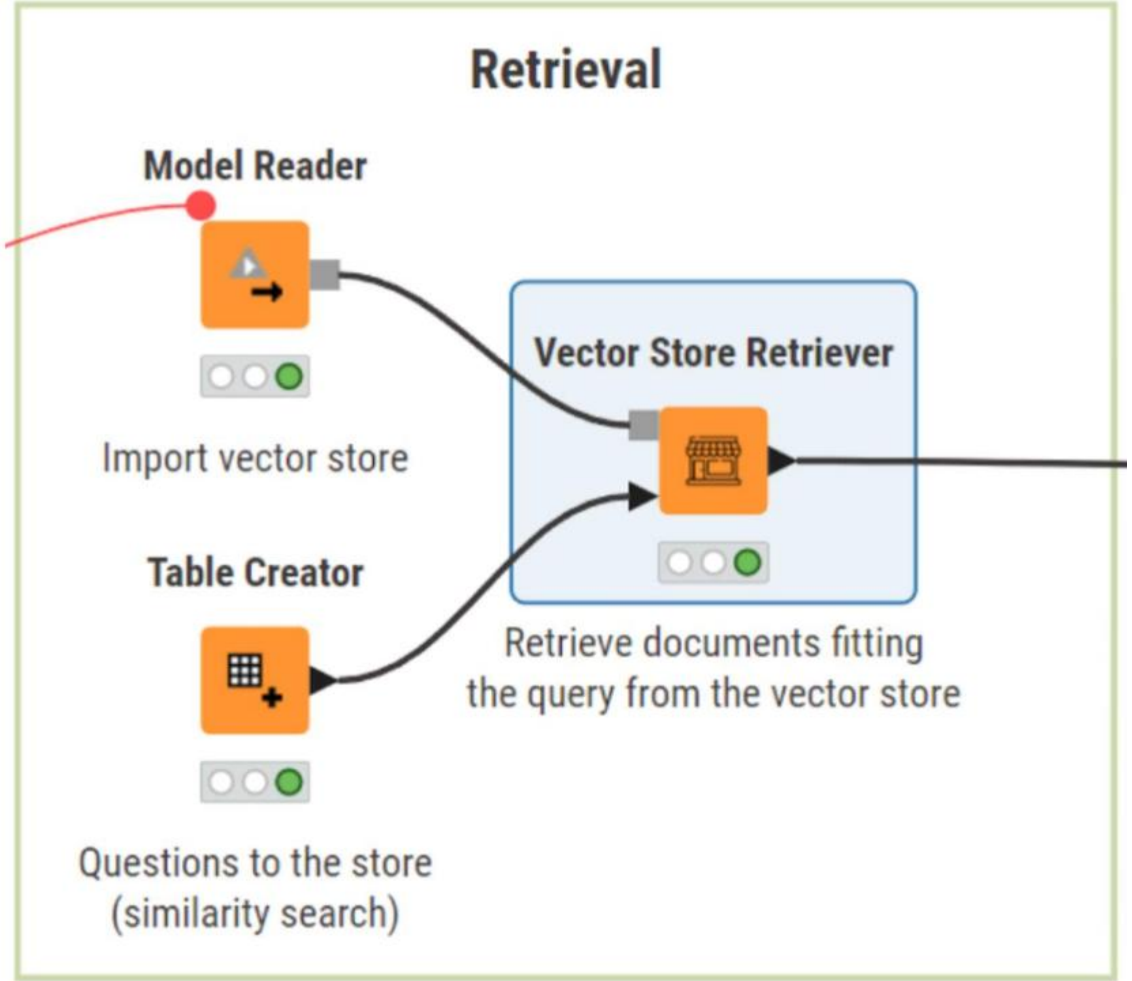
Submit the augmented prompt(s) to an LLM

- **7. Submit the augmented prompt(s) to an LLM**
- Send the augmented prompt as usual to the LLM and receive more tailored responses, enriched by the additional knowledge.


3. Retrieve relevant chunks
Vector Store Retriever
5
Retrieve results
Process retrieved documents
4. Augment prompt
Expression
6
Create a prompt for each review
5. Generate responses
OpenAI LLM Selector
LLM Prompter (Table)
7
Analyze sentiment of product reviews and extract product
Data analysis
Extract the product and the sentiment into columns
Table Validator
Validate GenAI results before updating a table on Databricks
Review analysis

Reusing your vector store

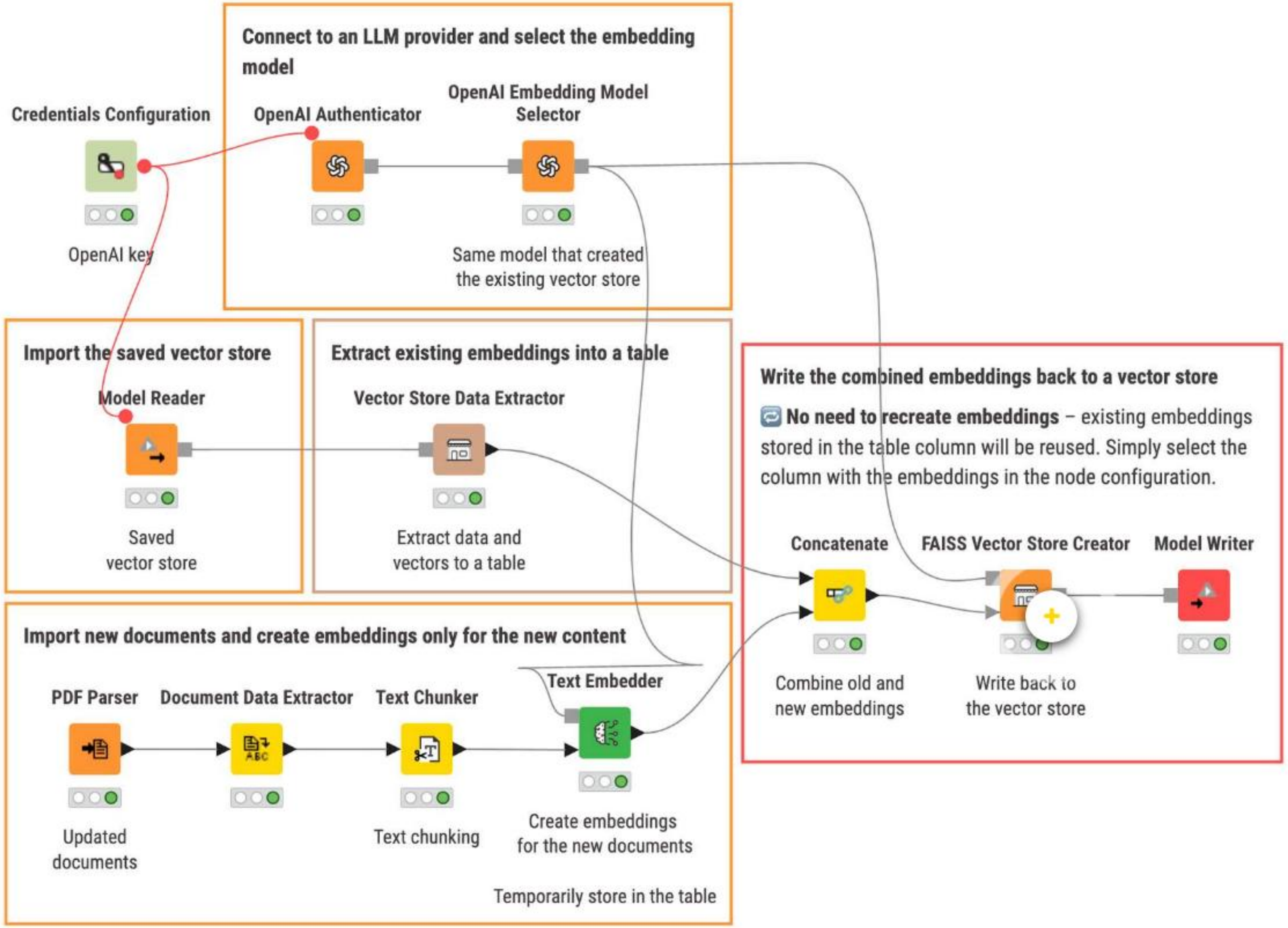
- Generating embeddings every time we run a RAG workflow can be costly and time-consuming. So how can we reuse them efficiently?
- Once your vector store is created, we can save it using the Model Writer node to a location of your choice. This allows us to:
  - Create the vector store once,
  - Reuse it across multiple workflows that perform RAG in real time.
- To import a saved vector store into a KNIME workflow:
  - Use the generic Model Reader node,
  - Or use the Vector Store Reader node specific to your vector store provider.
- After importing the vector store, continue with the standard RAG steps: retrieval → prompt augmentation → generation.


Retrieval
Model Reader
Import vector store
Vector Store Retriever
Retrieve documents fitting the query from the vector store
Table Creator
Questions to the store (similarity search)

- If we use a Vector Store Reader node, we must reconnect to the embedding model using the appropriate Embeddings Model Connector node.
- This is because the embedding model is not saved with the vector store.
- If we're using the Model Reader, we need to ensure it has access to our embedding model provider credentials in order to function properly with the Vector Store Retriever node.

Keeping our knowledge base up to date

- Our knowledge base should contain not only domain-specific but also the most current information.
- To keep our RAG system relevant over time, we'll need to update both the knowledge base and the vector store as new information becomes available.
- Earlier, we learned how to save and reuse vector stores to avoid re-generating embeddings every time a RAG pipeline executes. But how do we handle updates? Do you need to re-embed the entire knowledge base from scratch each time?
- Using the Vector Store Data Extractor node, we can extract the contents of a vector store (including embeddings) into a KNIME table.
- With the Text Embedder node, we can generate and save embeddings from new text to a table.
- Once we have two tables, we can simply concatenate them and save the combined result back into a vector store. Explore the example workflow.



Connect to an LLM provider and select the embedding model

Import the saved vector store

Extract existing embeddings into a table

Import new documents and create embeddings only for the new content

Write the combined embeddings back to a vector store

🔁 **No need to recreate embeddings** – existing embeddings stored in the table column will be reused. Simply select the column with the embeddings in the node configuration.

Credentials Configuration
OpenAI key
OpenAI Authenticator
OpenAI Embedding Model Selector
Same model that created the existing vector store
Model Reader
Saved vector store
Vector Store Data Extractor
Extract data and vectors to a table
PDF Parser
Updated documents
Document Data Extractor
Text Chunker
Text chunking
Text Embedder
Create embeddings for the new documents
Temporarily store in the table
Concatenate
Combine old and new embeddings
FAISS Vector Store Creator
Write back to the vector store
Model Writer

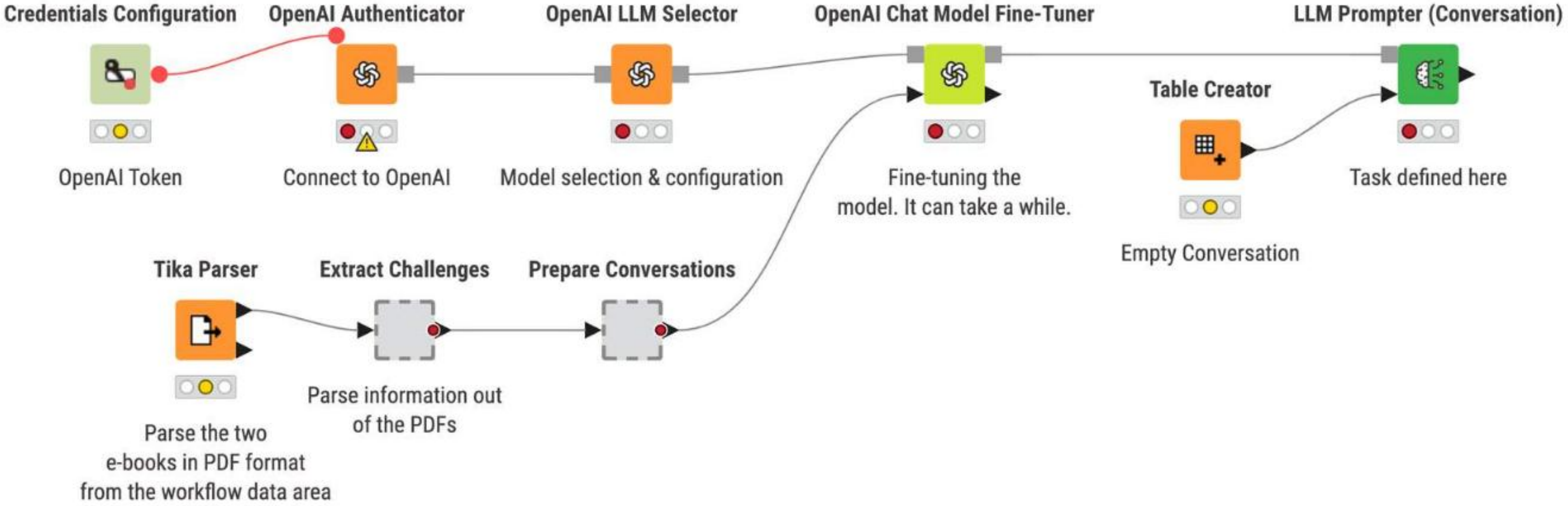
Fine-tuning

- Another strategy to reduce hallucinations in LLMs is fine-tuning.
- Retraining an entire large language model from scratch is impractical because of their size.
- Why consider fine-tuning when RAG already helps mitigate hallucinations?
- RAG has its limitations and can fail, for example, if the model lacks the foundational knowledge to simply understand the user’s question (with or without retrieved context).
- In such cases, fine-tuning can help by training the model on custom data to adapt its behavior or extend its capabilities.
- **Fine tuning (LLM)** is the process of training an existing language model on a smaller, task-specific dataset to improve performance on a particular domain or application. This adjusts the model's internal parameters based on the new data without retraining it from scratch.

- When working with closed-source models (like OpenAI’s), fine-tuning typically involves:
- Uploading your training data via an API call,
- Letting the provider perform the fine-tuning on their infrastructure,
- Accessing the newly fine-tuned model through a dedicated model ID via future API calls

Fine-tuning incurs additional costs—both for the fine-tuning process itself and for using the resulting model. Besides, as base models evolve over time, you may need to re-fine-tune periodically to keep up with updates.

KNIME makes it possible to fine-tune OpenAI chat models through the dedicated node. You can also delete fine-tuned models from your OpenAI account when no longer needed. See the example in the image below.


Credentials Configuration
OpenAI Token
OpenAI Authenticator
Connect to OpenAI
OpenAI LLM Selector
Model selection & configuration
OpenAI Chat Model Fine-Tuner
Fine-tuning the model. It can take a while.
Table Creator
Empty Conversation
LLM Prompter (Conversation)
Task defined here
Tika Parser
Parse the two e-books in PDF format from the workflow data area
Extract Challenges
Parse information out of the PDFs
Prepare Conversations