Open for Innovation
KNIME

KNIME AI Extension Guide

KNIME AG, Zurich, Switzerland
Version 5.5 (last updated on)

Table of Contents

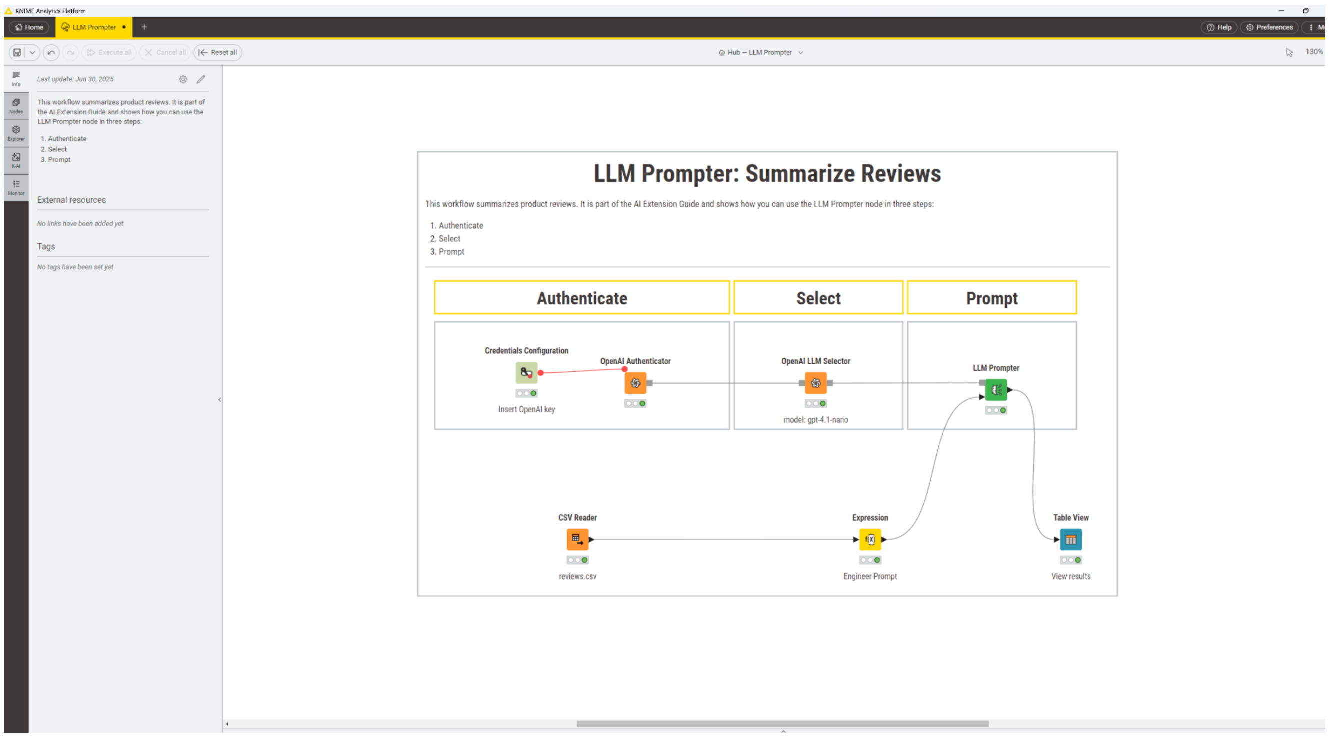
Prompting a Model

This section of the guide explains how to send a prompt to a Large Language Model (LLM) in KNIME Analytics Platform.

The process follows three steps:

1. Authenticate
2. Select
3. Prompt

To clarify each step, this section includes an example workflow. The workflow connects to OpenAI's GPT-4.1 model and summarizes product reviews stored in a *.csv* file.



Last update: Jun 30, 2025

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

External resources

No links have been added yet

Tags

No tags have been set yet

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

Figure 1. Example workflow using OpenAI's GPT-4.1 in KNIME to summarize product reviews via authentication, model selection, and prompting.

Install the KNIME AI Extension

To use AI in KNIME workflows, install the KNIME AI Extension. You can do this in two ways:

- From KNIME Hub: Drag the KNIME AI Extension from the KNIME Hub into your KNIME workspace.
- From within KNIME Analytics Platform:

a. Go to *Menu* from the toolbar.
b. Select *Install extensions*.
c. Search for "AI Extension" and follow the instructions to complete the installation.

Authenticate

Before sending prompts to models, you must authenticate with your chosen model provider. Most providers require an API key or token obtained from your user account.

Authentication typically involves two nodes:

- Credentials Configuration
- [Provider] Authenticator (e.g. OpenAI Authenticator)

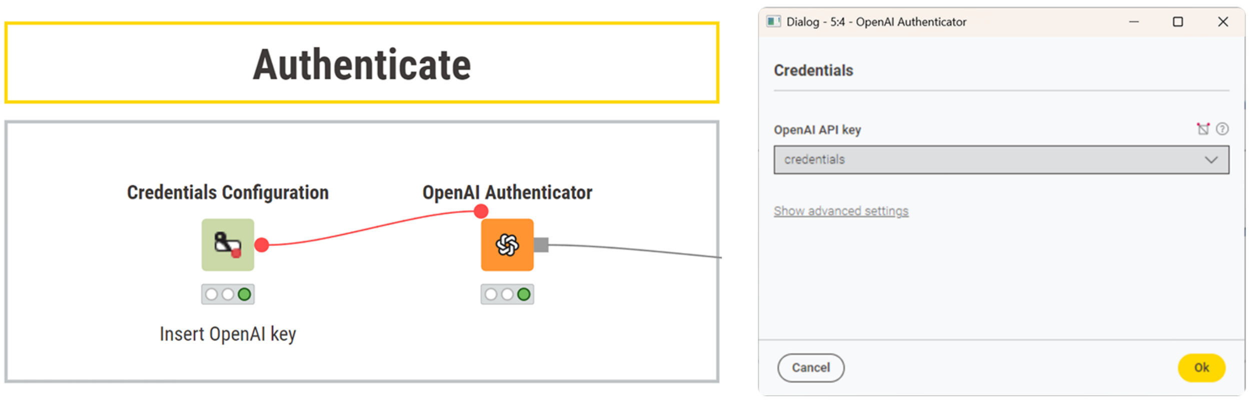
Credentials Configuration node

This node stores credentials as a flow variable for downstream nodes.

In KNIME, a **flow variable** is a named value that passes data, like credentials or configuration settings, between nodes. Using flow variables makes workflow more flexible and avoids hardcoding sensitive information.

To configure the node:

- Enter the API key or token in the *Password* field.
- Leave the *Username* field blank
- If you uncheck *Save password in configuration (weakly encrypted)*, the key is not saved between sessions and must be re-entered when reopening the workflow.


Authenticate
Credentials Configuration
OpenAI Authenticator
Insert OpenAI key
Dialog - 5:4 - OpenAI Authenticator
Cancel
Ok

Credentials

OpenAI API key: credentials

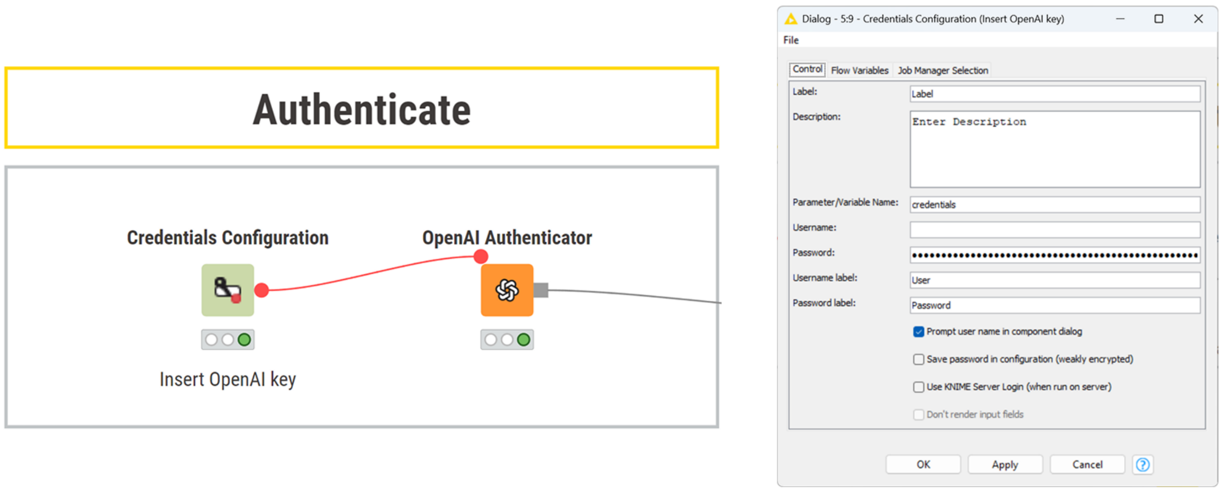
Show advanced settings

Figure 2. The Credentials Configuration node stores the OpenAI API key in the Password field and creates a credentials flow variable for use in downstream authentication nodes.

Authenticator node

Assign the credentials flow variable to the *API key* field in the authenticator node's configuration. Provider-specific parameters (such as *region* or *endpoint*) may also be required.

If the credentials are invalid or incomplete, the authenticator node will fail during execution.



Label:	Label
Description:	Enter Description
Parameter/Variable Name:	credentials
Username:	
Password:	••
Username label:	User
Password label:	Password

[✓] Prompt user name in component dialog
[ ] Save password in configuration (weakly encrypted)
[ ] Use KNIME Server Login (when run on server)
[ ] Don't render input fields

Figure 3. In this example, the OpenAI Authenticator node is configured to retrieve credentials from the flow variable generated by the Credentials Configuration node.

Select

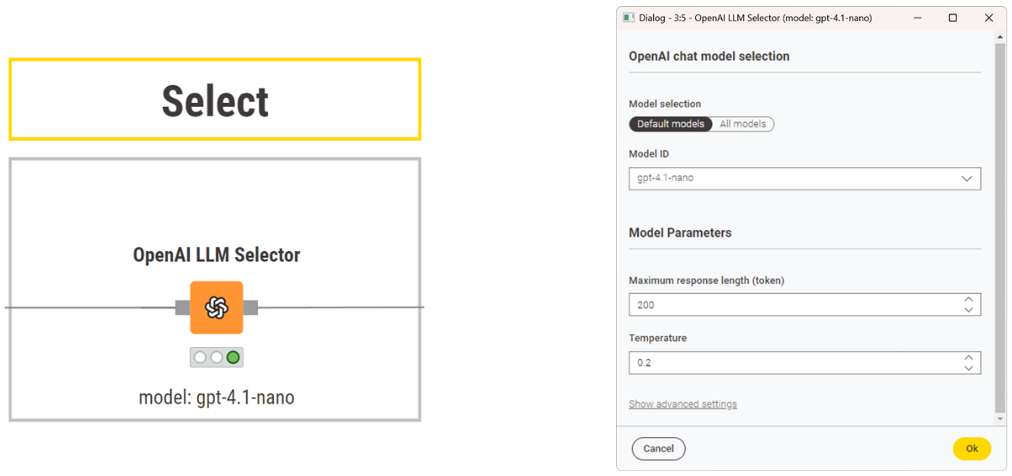
After authenticating, use the appropriate connector node to select the model you want to prompt.

The Selector nodes allow you to:

- Select models from commercial APIs (OpenAI, Gemini, Anthropic, etc.)
- Connect to models hosted on KNIME Hub, or running locally via GPT4All.

Model selection parameters

- *Max New Tokens / Max Response Length:* maximum response size.
- *Temperature*: controls output randomness (0 = deterministic, higher = more creative).
- *Advanced Settings:* additional parameters such as *Top-p Sampling*, *Seed*, or *Parallel Requests*.



OpenAI chat model selection

Model selection: [✓ Default models] All models

Model ID: gpt-4.1-nano

Model Parameters

Maximum response length (token): 200

Temperature: 0.2

Figure 4. The OpenAI LLM Selector node is configured to use OpenAI's GPT-4.1 model to summarize product reviews.

For a full overview of supported providers, available authenticator and selector nodes, required credentials, and example workflows, see the Provider Reference Table.

Prompt

Prompting means sending text instructions to a language model to perform specific tasks. Depending on the node you use, the model can generate text, answer questions, extract information, or return vector representations of the input.

In KNIME, a prompt is simply a string of text. You can create these strings using nodes such as Expression, String Manipulation, or Table Creator.

The KNIME AI extension features three prompting nodes:

- LLM Prompter

  Single-turn text prompting

- [LLM Chat Prompter]

  Chat-style multi-turn prompting

- Text Embedder

  Generates embeddings (vector representations)

LLM Prompter

The LLM Prompter node sends simple text prompts to a language model and returns the model’s response as text. It is used for one-shot prompting that does not require conversation history.

Common use cases:

- Classification
- Summarization
- Rewriting
- Extraction

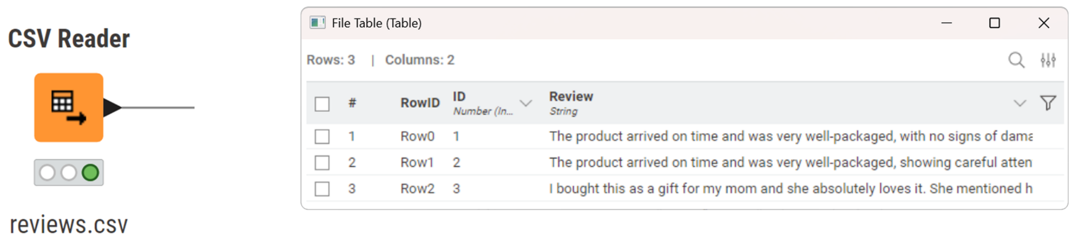
Example: Summarize Product Reviews

You receive product reviews that are too lenghty so you decide to summarize them. The input data looks like this:

ID	Comment
1	The product arrived on time… user-friendly.
2	The product arrived on time… every day.
3	I bought this as a gift… other family members.

1. Read the data

Use the CSV Reader node to load this table into your KNIME workflow. Each review is stored as a string in the comment column.



#	RowID	ID Number (In...	Review String
1	Row0	1	The product arrived on time and was very well-packaged, with no signs of dama
2	Row1	2	The product arrived on time and was very well-packaged, showing careful atten
3	Row2	3	I bought this as a gift for my mom and she absolutely loves it. She mentioned h

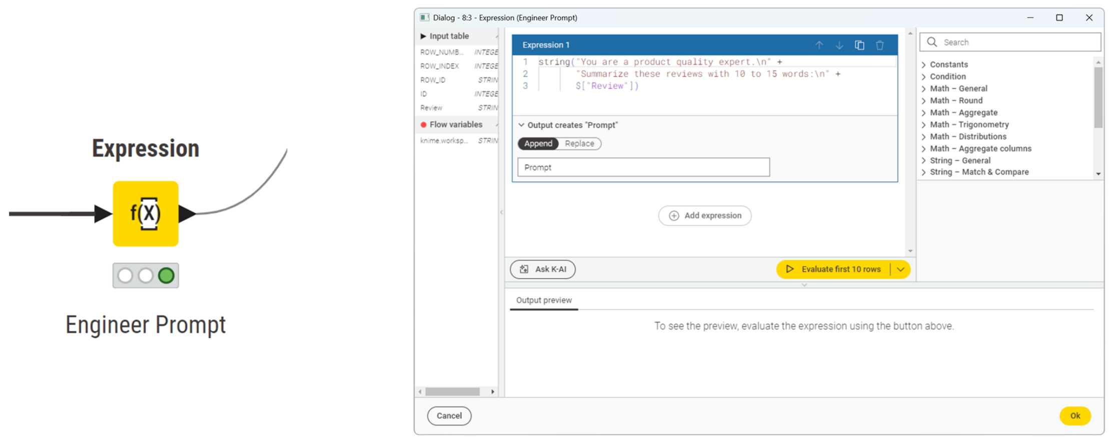
Figure 5. The CSV Reader node loads a table of customer reviews, each stored as a string, to be summarized later.

2. Create the prompts

Use the Expression node to build this prompt:

```
string("You are a product quality expert.\n" +
       "Summarize these reviews with 10 to 15 characters:\n" +
       $["Review"])
```

This creates a new column called *prompt* that contains the instruction for each row.



ROW_NUMB...	INTEGE
ROW_INDEX	INTEGE
ROW_ID	STRIN
ID	INTEGE
Review	STRIN
knime.worksp...	STRIN

```
string("You are a product quality expert.\n" +
       "Summarize these reviews with 10 to 15 words:\n" +
       $["Review"])
```

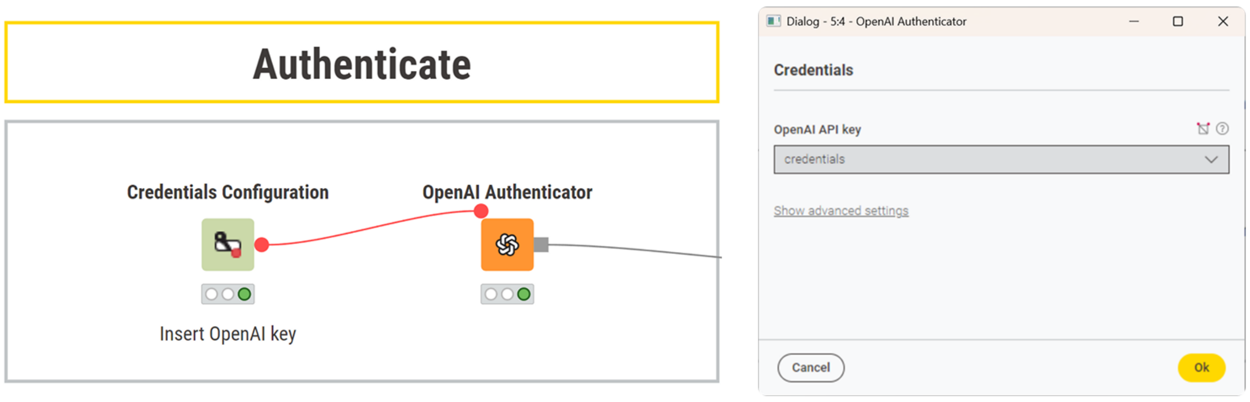
Output creates "Prompt"
Append | Replace
Prompt

Output preview
To see the preview, evaluate the expression using the button above.

Figure 6. The Expression node creates a new column called prompt that contains the instructions for the LLM to follow.

3. Authenticate to OpenAI

Use the Credentials Configuration and the OpenAI Authenticator node to provide your *API key*.

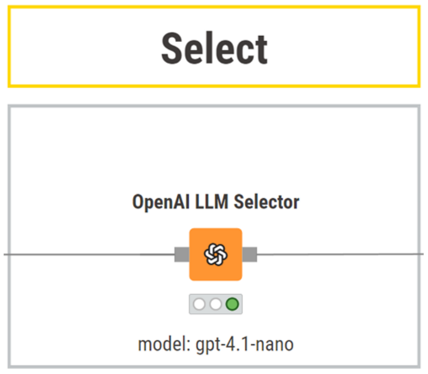


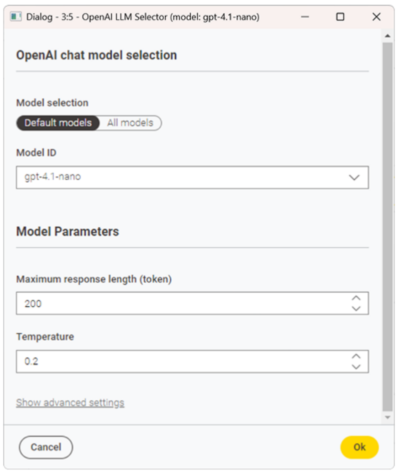
Credentials
OpenAI API key: credentials

Figure 7. The Credentials Configuration securely stores your API key. The OpenAI Authenticator node reads this key to authorize requests to the OpenAI API.

4. Selecting a model

Use the OpenAI LLM Selector node to choose the model you want to use.





OpenAI chat model selection

Model selection: [✓ Default models] All models

Model ID: gpt-4.1-nano

Model Parameters

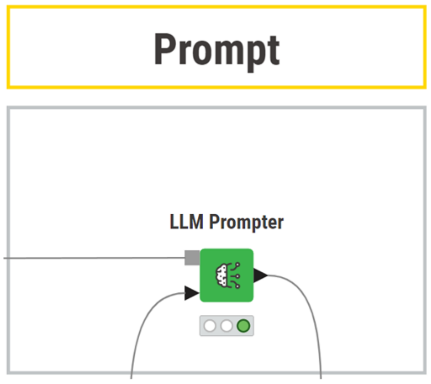
Maximum response length (token): 200

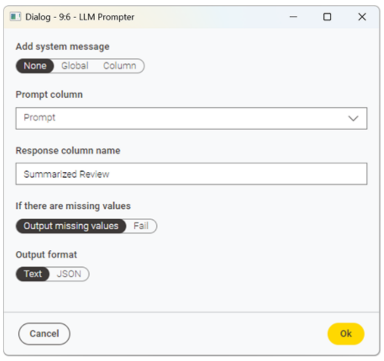
Temperature: 0.2

Figure 8. The OpenAI LLM Selector node connects to the API and selects GPT4 as model for prompting.

5. Prompt

The LLM Prompter node reads the *Prompt* column, sends it to the model, and stores the model's reply in a new column called *Response*.





Add system message: [✓ None] Global Column

Prompt column: Prompt

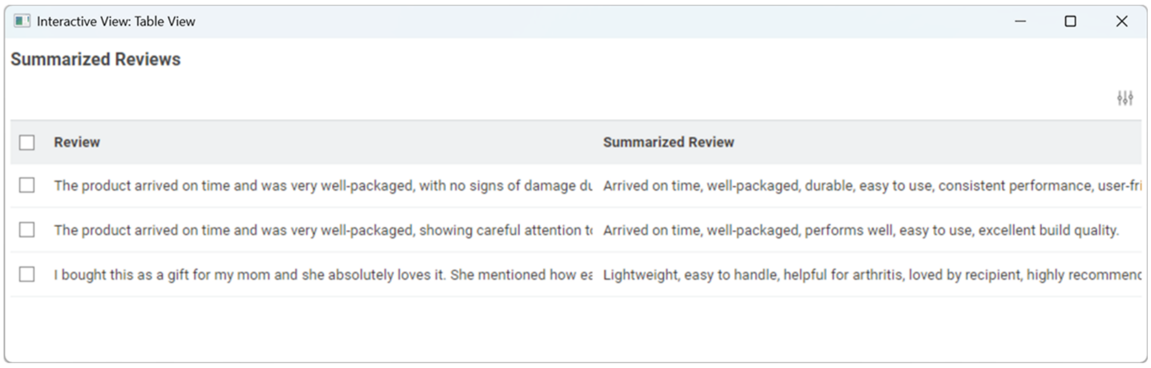
Response column name: Summarized Review

If there are missing values: [✓ Output missing values] Fail

Output format: [✓ Text] JSON

Figure 9. The LLM Prompter node uses the specified prompt column and generates a response from the model, which it stores in a new column called Response.

To receive structured JSON responses, make sure the selected model supports JSON mode. Also, select *JSON* as the *Output format* in the node configuration and explicitly request the JSON format in the prompt itself. In many cases, it helps to include a few examples of the expected structure directly in the prompt to ensure output quality.



Interactive View: Table View

Summarized Reviews

Review	Summarized Review
The product arrived on time and was very well-packaged, with no signs of damage du	Arrived on time, well-packaged, durable, easy to use, consistent performance, user-fri
The product arrived on time and was very well-packaged, showing careful attention t	Arrived on time, well-packaged, performs well, easy to use, excellent build quality.
I bought this as a gift for my mom and she absolutely loves it. She mentioned how ea	Lightweight, easy to handle, helpful for arthritis, loved by recipient, highly recommenc

Figure 10. At the end of the workflow, the LLM Prompter node outputs a new column called Response. Each row contains a summary generated by the model based on the corresponding customer review.

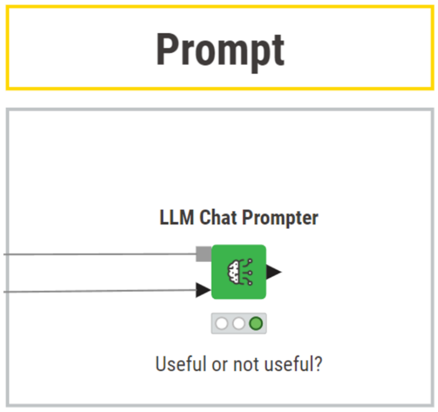
LLM Chat Prompter

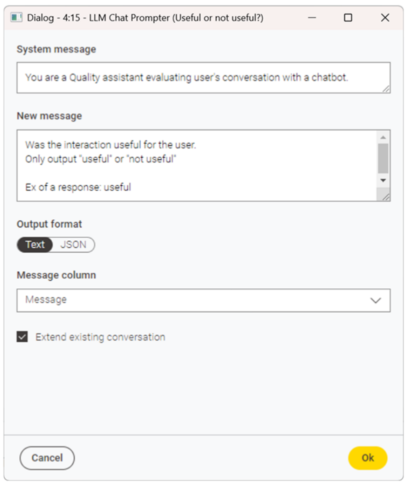
The LLM Chat Prompter node allows chat-style prompts to a language model. Unlike LLM Prompter, it accepts conversation history as input to provide context for generating responses.

❗ The LLM Chat Prompter node does not automatically manage multi-turn conversations. The full conversation history must be provided as input for every execution if previous context is required.

Configuration

- *System Message (optional)*: defines assistant behavior, e.g., *"You are a helpful customer support agent."*
- *New User Message (optional):* appends a new user message.
- *Output Format:* Plain text or JSON.
- *Message Column:* column containing KNIME Message type (role, message).




Dialog - 4:15 - LLM Chat Prompter (Useful or not useful?)

System message

You are a Quality assistant evaluating user's conversation with a chatbot.

New message

Was the interaction useful for the user.
Only output "useful" or "not useful"

Ex of a response: useful

Output format

[✓ Text] JSON

Message column

Message

☑ Extend existing conversation

Cancel Ok

Figure 11. Configuration dialog of the LLM Chat Prompter node.

Inputs

The node accepts two additional input ports:

- *Conversation History (optional):*

  A table containing previous conversation messages in KNIME’s Message format. These messages will be used as context for the current prompt. The node does not automatically manage multi-turn sessions. The full history must be maintained externally and provided again on each node execution if previous context is required.

- *Tool Definitions (optional, advanced):*

  A table containing tool definitions in JSON format. These definitions enable tool calling functionality. Tool definitions are explained in detail in the Agents workflow section of this guide.

Example: Customer Care Quality Checker

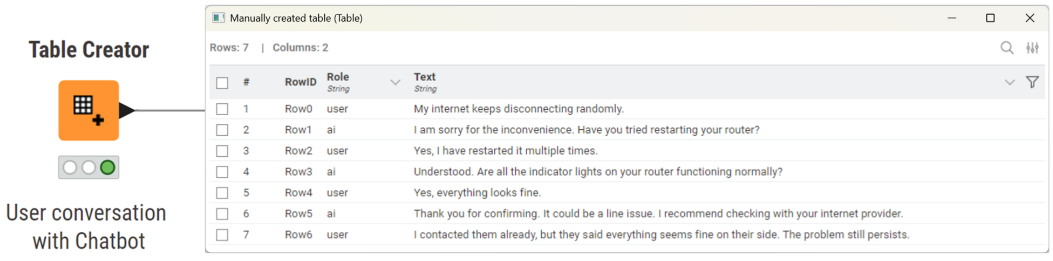
You want to evaluate a conversation between a user and an AI. The input data looks like this:

Role	Message
user	My internet keeps disconnecting randomly.
ai	I’m sorry for the inconvenience. Have you tried restarting your router?

user	Yes, I have restarted it multiple times.
ai	Understood. Are all the indicator lights on your router functioning normally?
user	Yes, everything looks fine.
ai	Thank you for confirming. It could be a line issue. I recommend checking with your internet provider.
user	I contacted them already, but they said everything seems fine on their side. The problem still persists.

1. Create conversation history

The Table Creator node builds a conversation table with two columns: *Role* and *Text*. This simulates previous turns between the user and the AI assistant.



Manually created table (Table)

Rows: 7 | Columns: 2

#	RowID	Role *String*	Text *String*
1	Row0	user	My internet keeps disconnecting randomly.
2	Row1	ai	I am sorry for the inconvenience. Have you tried restarting your router?
3	Row2	user	Yes, I have restarted it multiple times.
4	Row3	ai	Understood. Are all the indicator lights on your router functioning normally?
5	Row4	user	Yes, everything looks fine.
6	Row5	ai	Thank you for confirming. It could be a line issue. I recommend checking with your internet provider.
7	Row6	user	I contacted them already, but they said everything seems fine on their side. The problem still persists.

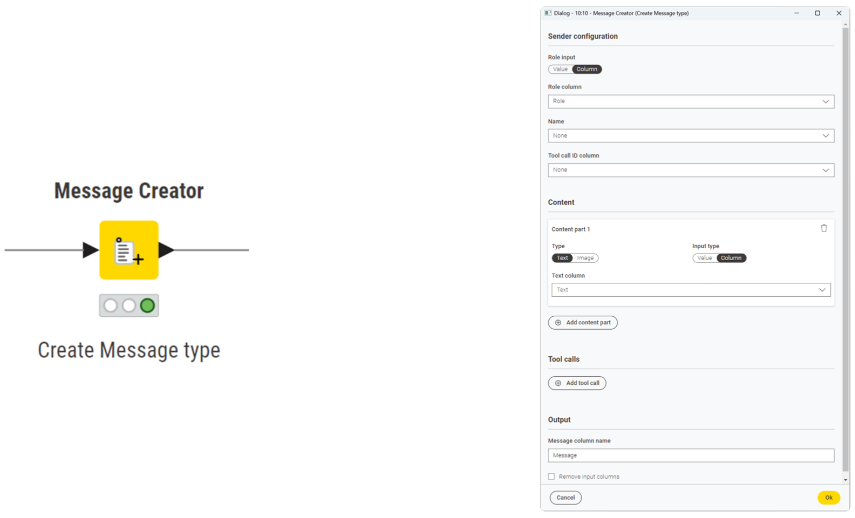
Figure 12. The Table Creator node builds a table with two columns: role and message.

2. Convert to Message type

The Message Creator node transforms the conversation table into KNIME's Message data type.

The input table already contains a column named Role. In the configuration dialog, select this column under Role Column.

The message text is already in a column containing the conversation content. Select this column under Text.



Dialog - 10:10 - Message Creator (Create Message type)

Sender configuration

Role input: Value [✓ Column]

Role column: Role

Name: None

Tool call ID column: None

Content

Content part 1

Type: [✓ Text] Image

Input type: Value [✓ Column]

Text column: Text

Tool calls

Output

Message column name: Message

☐ Remove input columns

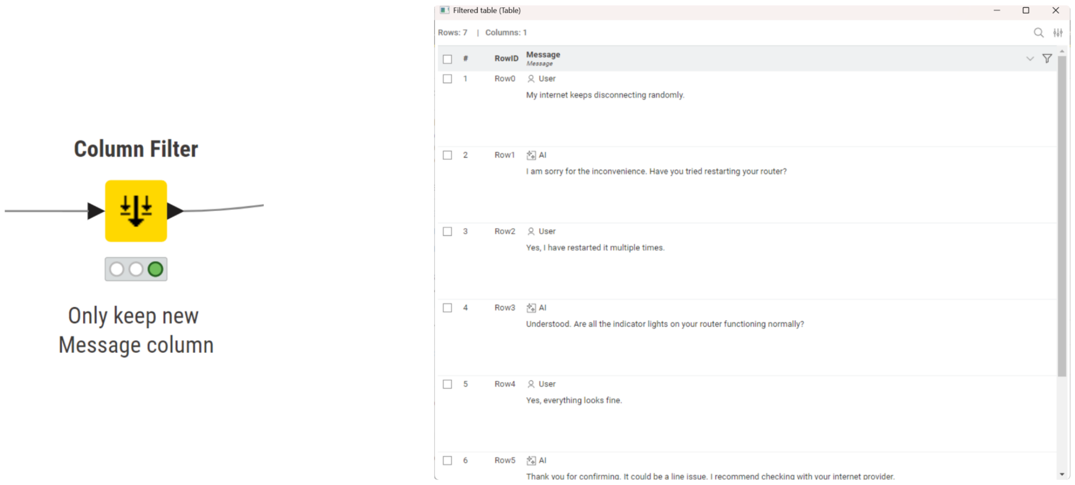
Figure 13. Configuration dialog of the Message Creator node.

After configuring the node, this creates a new column named Message, which contains the KNIME Message data type.

This column combines both role and message content in a format required by downstream nodes such as the LLM Chat Prompter.

3. Filter Message column

The Column Filter node keeps only the Message column for input into the LLM Chat Prompter.



Filtered table (Table)

Rows: 7 | Columns: 1

#	RowID	Message (Message)
1	Row0	User My internet keeps disconnecting randomly.
2	Row1	AI I am sorry for the inconvenience. Have you tried restarting your router?
3	Row2	User Yes, I have restarted it multiple times.
4	Row3	AI Understood. Are all the indicator lights on your router functioning normally?
5	Row4	User Yes, everything looks fine.
6	Row5	AI Thank you for confirming. It could be a line issue. I recommend checking with your internet provider.

Figure 14. Message column passed into the LLM Chat Prompter node.

4. Authenticate to OpenAI

The Credentials Widget and OpenAI Authenticator nodes manage authentication.

5. Select a model

The OpenAI LLM Selector node selects *gpt-4o-nano* as LLM.

6. Prompt

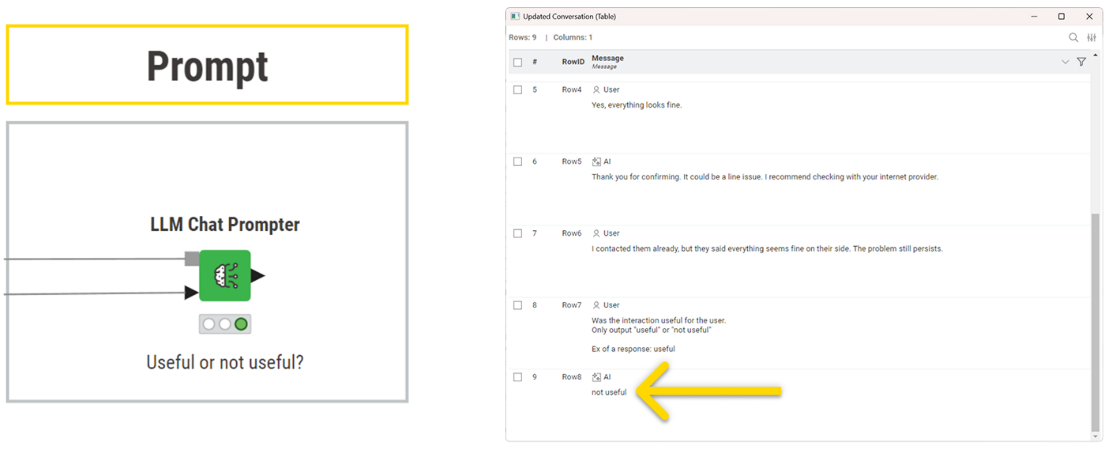
The LLM Chat Prompter node appends the new user instruction to the provided conversation history and requests a response from the model.

- *System Message*:

```
You are a customer care quality checker.
```

- *New User Message*:

```
Was the interaction useful for the user? Only output 'useful' or 'not useful'.
```



#	RowID	Message
5	Row4	User Yes, everything looks fine.
6	Row5	AI Thank you for confirming. It could be a line issue. I recommend checking with your internet provider.
7	Row6	User I contacted them already, but they said everything seems fine on their side. The problem still persists.
8	Row7	User Was the interaction useful for the user. Only output "useful" or "not useful" Ex of a response: useful
9	Row8	AI not useful

Figure 15. Output of the LLM Chat Prompter node with updated conversation.

Text Embedder

The Text Embedder node converts text into embeddings, which are numerical vectors that capture the meaning of the text.

What is a vector

A vector is a list of numbers representing a text in mathematical space. Texts with similar meaning are placed close together in this space, even if they use different words. For example, *dog* and *puppy* would be represented by nearby vectors. Embeddings allow models to compare meaning, group similar texts, and search based on semantics rather than simple keyword matching.

Node Output

The node processes each row of the input table and creates a new vector column containing the embedding. Each embedding typically contains hundreds or thousands of numeric values. Unlike the LLM Prompter node (which generate natural language text), the Text Embedder produces purely numeric embeddings, fully compatible with distance calculations, clustering, and dimensionality reduction operations in KNIME.

Common use cases:

- Semantic search: finding texts with similar meaning
- Clustering: grouping texts into categories
- Similarity scoring: measuring how close two texts are in meaning

Example: Job Candidate Similarity Plot

You want to find the best matching candidate for a machine learning specialist position that requires a deep knowledge of NLP, reinforcement learning, and experience with large language models.

The input data looks like this:

Candidate	CV Text
Candidate 1	Senior data scientist with 5 years in NLP and deep learning. Experienced in Python, TensorFlow, and cloud computing.
Candidate 2	Business analyst with expertise in market research, Excel, and customer insights. Limited programming experience.
Candidate 3	Data engineer skilled in ETL pipelines, big data processing, Spark, and distributed systems.
Candidate 4	Machine learning researcher specialized in natural language understanding, reinforcement learning, and generative models.
Candidate 5	Software developer with strong background in Java, web development, and full-stack applications.

Candidate	CV Text
Ideal Profile	Looking for a machine learning specialist with deep knowledge of NLP, reinforcement learning, and experience with large language models.

1. Read the data

The CSV Reader node loads the table of CVs and the ideal profile into KNIME.

2. Authenticate

The Credentials Configuration node stores the OpenAI API key. The OpenAI Authenticator node uses these credentials to authenticate with the OpenAI service.

3. Select

The OpenAI Embedding Model Selector node selects the embedding model *text-embedding-3-small* that will be used to generate embeddings.

4. Prompt (Generate embeddings)

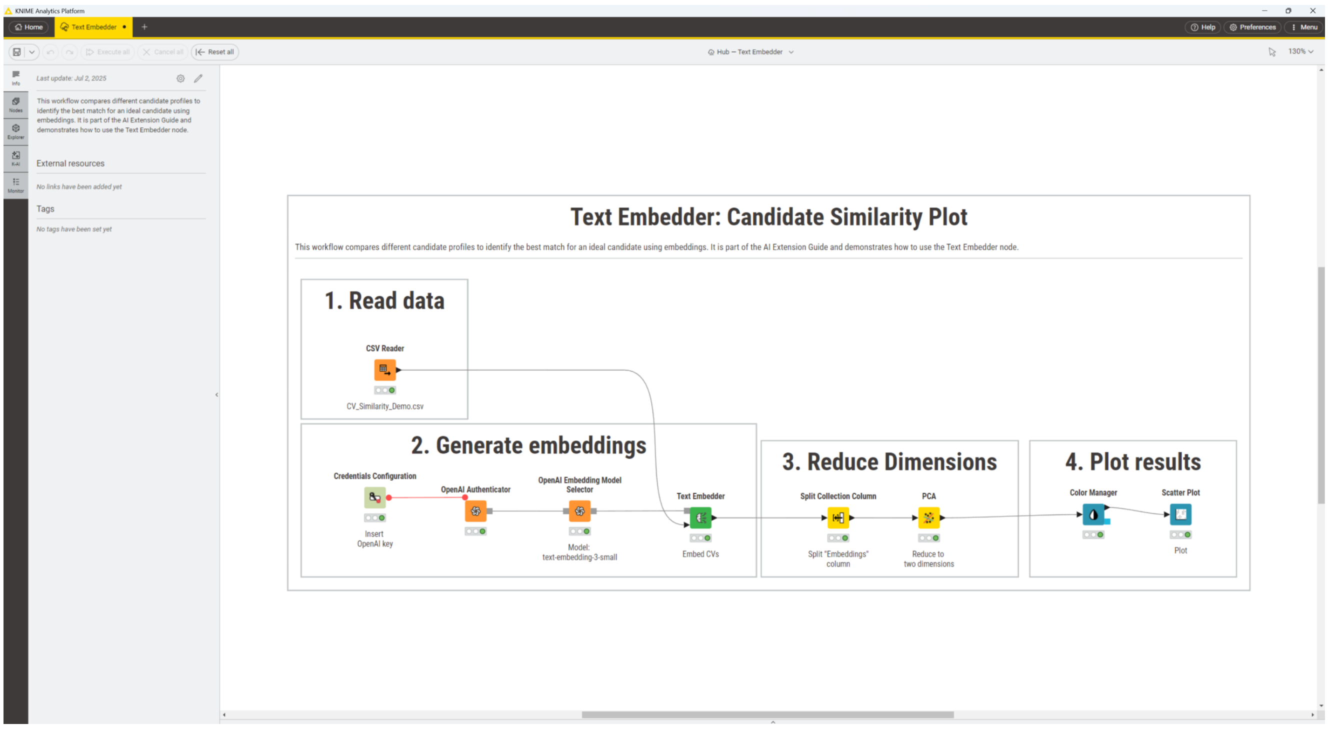
The Text Embedder node converts each CV text into an embedding vector using the selected embedding model.

5. Reduce dimensions

Embeddings are high-dimensional vectors that cannot be directly plotted. The Split Collection Column node separates the vector into individual numeric columns. Then, the PCA node reduces the many dimensions down to two, making the embeddings suitable for 2D visualization.

6. Plot Results

The Scatter Plot node displays how closely each candidate matches the ideal profile based on semantic similarity.



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.

No links have been added yet

No tags have been set yet

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.

Figure 16. The KNIME workflow that compares candidate profiles using embeddings and plots the results

The plot shows that *Candidate 4* is closest to the ideal profile, confirming semantic similarity between their experience and the target profile.

Ideal Profile	Looking for a machine learning specialist with deep knowledge of NLP, reinforcement learning, and experience with large language models.
Candidate 4	Machine learning researcher specialized in natural language understanding, reinforcement learning, and generative models.

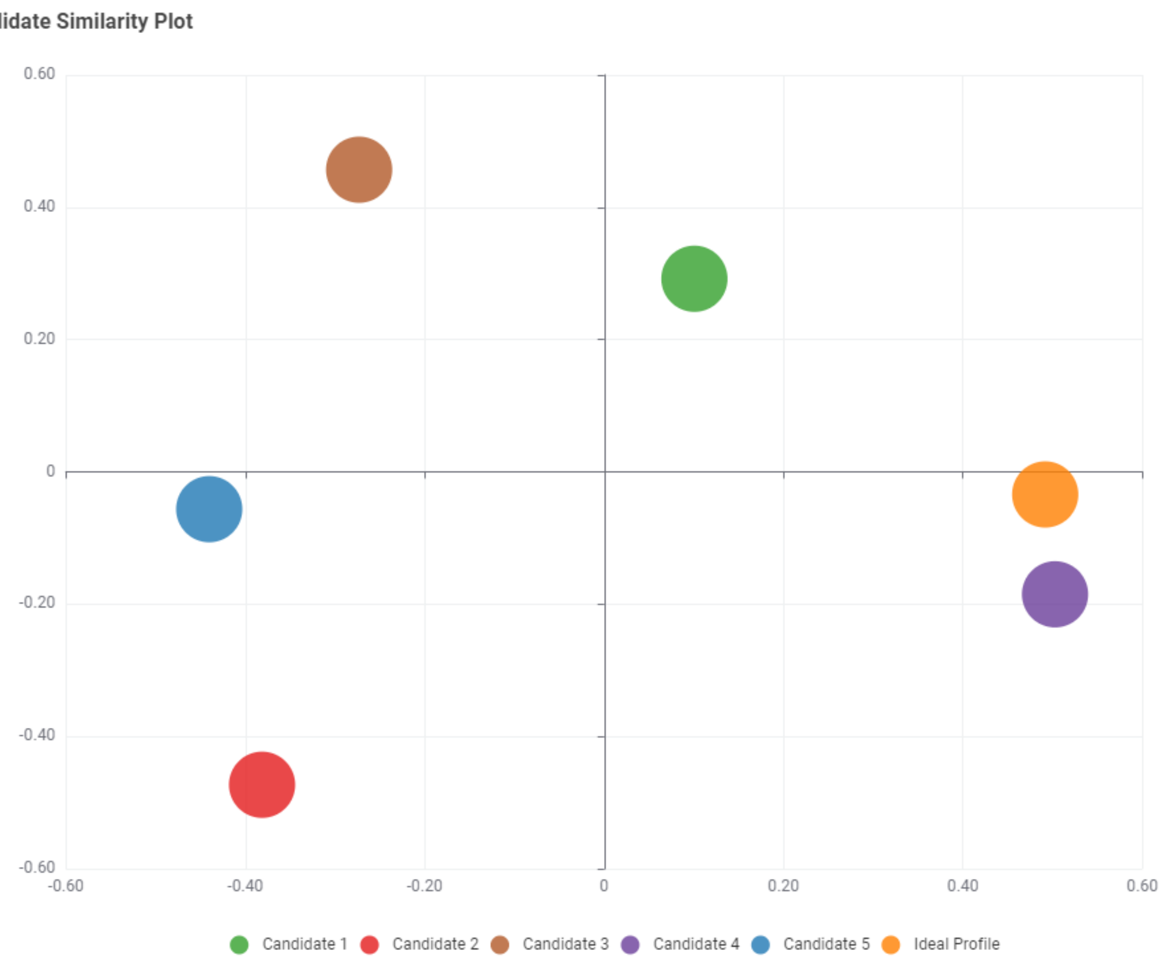


Figure 17. The resulting 2D plot of candidate similarity

Provider Reference Table

Authenticator nodes (by provider)	**Selector**	**Required Credentials**	**Link to Provider**	**Example workflow**
OpenAI Authenticator	• OpenAI LLM Selector • OpenAI Embeddings Model Selector	• OpenAI API key • OpenAI base URL (optional)	OpenAI	workflow

Authenticator nodes (by provider)	Selector	Required Credentials	Link to Provider	Example workflow
Google AI Studio Authenticator	• Gemini LLM Selector • Gemini Embedding Model Selector	• Google AI Studio API key	Google	workflow
Anthropic Authenticator	• Anthropic LLM Selector	• Anthropic AI key	Anthropic	workflow
IBM watsonx.ai™ Authenticator	• IBM watsonx.ai LLM Selector • IBM watsonx.ai Embedding Model Connector	• IBM wasonx.ai API key • Project or space connection	IBM	workflow
DeepSeek Authenticator	• DeepSeek LLM Selector	• DeepSeek API key • Base URL (optional)	DeepSeek	workflow
Google AI Studio Authenticator	• Vertex AI connector	• Google Cloud’s Project ID • Google Cloud’s Location	Vertex ai	workflow
Databricks Workspace Connector	• Databricks LLM Selector • Databricks Embeddings Model Selector	• Databricks workspace URL • Personal access token	Databricks	workflow

Retrieval-Augmented Generation (RAG) in KNIME

Large language models (LLMs) are powerful but limited. They do not have access to private documents or real-time information, and cannot easily be updated with new knowledge. Retrieval-Augmented Generation (RAG) solves this by injecting external information into the prompt at query time.

To build a RAG pipeline in KNIME:

1. Connect to model provider
2. Create a vector store
3. Retrieve context
4. Generate response

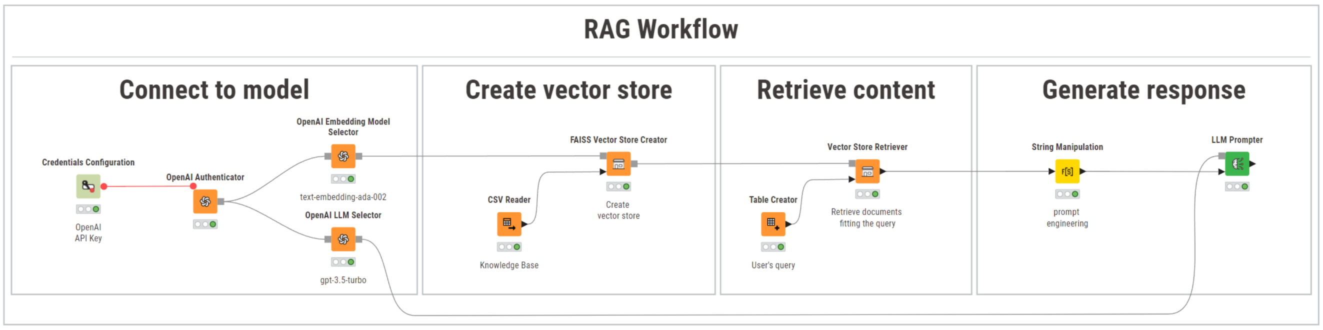


Figure 18. Four phases of a RAG pipeline in KNIME: connect to a model, create a vector store, retrieve relevant content, and generate a grounded response.

Connect to Model Provider

Before retrieving documents or generating responses, connect to a language model provider and select an embedding model.

Use the following nodes:

- Credentials Configuration

  Securely stores your API key or token.

- Authenticator node (e.g., OpenAI Authenticator)

  Authenticates access to your embedding and LLM provider.

- Embedding Model Selector (e.g., OpenAI Embedding Model Selector)

  Selects the embedding model used to vectorize your documents

- LLM Selector (e.g., OpenAI LLM Selector)

  Used later to generate the final response from the retrieved context.

For a full list of supported models, see the Provider Reference Table.

Create Vector Store

A vector store is an index of embeddings that enables semantic search by mapping documents to numerical vectors. KNIME supports two backends:

FAISS	Flat file–based index optimized for fast, local retrieval.
Chroma	JSON + SQLite format with support for metadata and document collections.

To build the vector store, use one of the following nodes:

- FAISS Vector Store Creator

  Generates a FAISS index using the selected embedding model. You can also use a column with precomputed embeddings instead of generating them in-node.

- Chroma Vector Store Creator

  Creates a Chroma store with optional metadata for advanced filtering or grouping. Supports both inline and precomputed embeddings.

To reuse a vector store across workflows:

- Save it with the Model Writer.
- Reload it using the Model Reader.

Chroma has updated its data layout. Older vector stores may require temporary copies at load time. Use the Vector Store Data Extractor node to migrate existing data into a new store.

Retrieve Context

Once your documents are indexed in a vector store, you can retrieve the most relevant entries for a user query.

In the example workflow above, the vector store is created in-memory using the FAISS Vector Store Creator node and passed directly to the retriever.

If instead you want to reuse a saved vector store (e.g., from disk or another workflow), use one of these nodes to load it:

- FAISS Vector Store Reader
- Chroma Vector Store Reader

Then, follow these steps:

1. **Input the user query**

   Use the Table Creator node to simulate the user's question. This creates a string column representing the input query. Any node that outputs a string is valid here.

2. **Retrieve relevant context**

   Connect the query to the Vector Store Retriever node. It embeds the query using the same model as the vector store and returns the most semantically similar results.

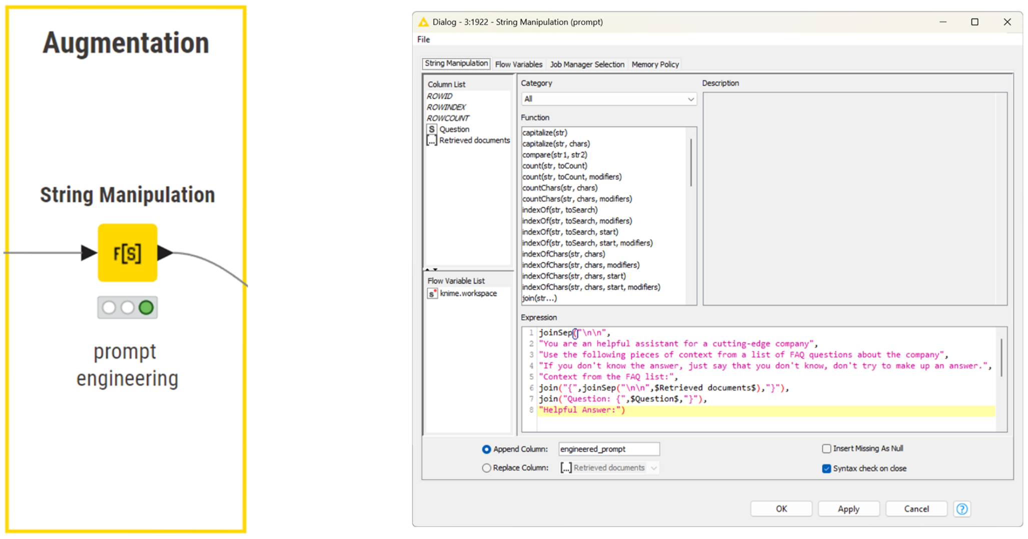
The output provides the contextual documents that will ground the LLM's final response.

Generate Response

After retrieving the context, prepare the final prompt by merging the results into a single string.

- Use String Manipulation to format the context:

  ```
  JOINSEP("\n", column("Answer"))
  ```



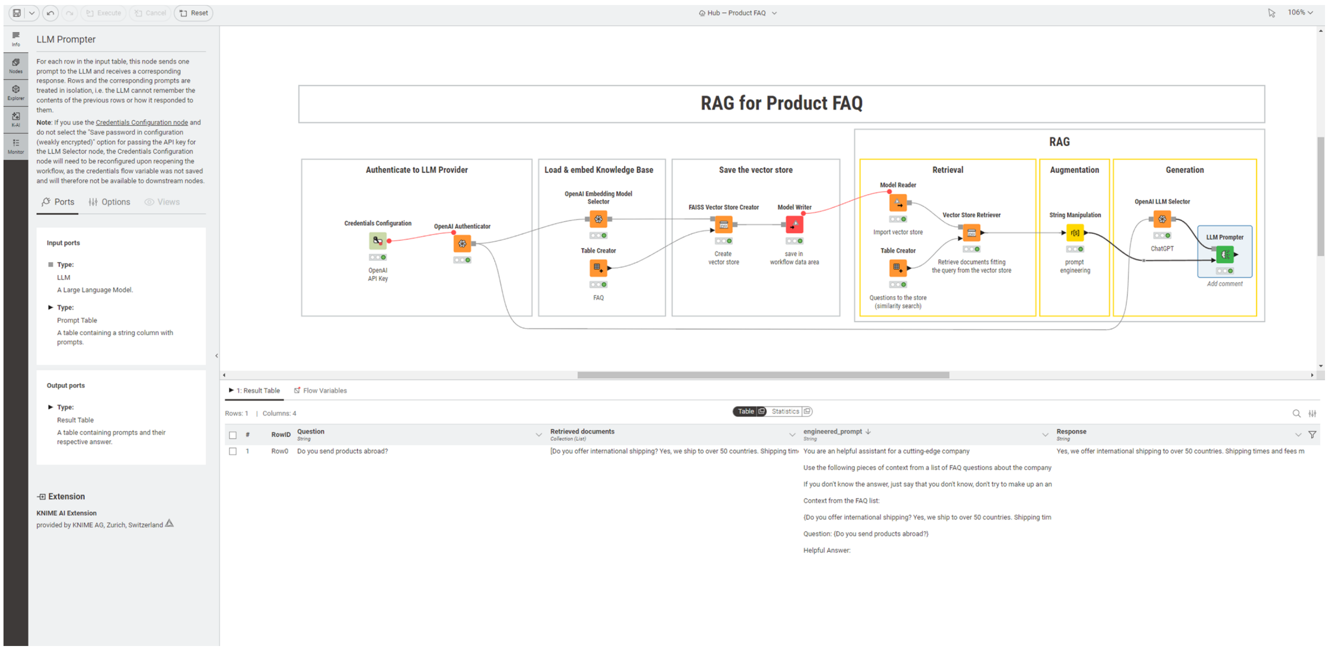
```
joinSep("\n\n",
"You are an helpful assistant for a cutting-edge company",
"Use the following pieces of context from a list of FAQ questions about the company",
"If you don't know the answer, just say that you don't know, don't try to make up an answer.",
"Context from the FAQ list:",
join("{",joinSep("\n\n",$Retrieved documents$),"}"),
join("Question: {",$Question$,"}"),
"Helpful Answer:")
```

Append Column: engineered_prompt
Replace Column: Retrieved documents
Insert Missing As Null
Syntax check on close [✓]

Figure 19. The prompt created by combining the query and retrieved context.

- Send the full prompt (question + retrieved context) to the LLM using: LLM Prompter

The final answer is stored in a column named *Response*.



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.

Note: If you use the Credentials Configuration node and do not select the "Save password in configuration (weakly encrypted)" option for passing the API key for the LLM Selector node, the Credentials Configuration node will need to be reconfigured upon reopening the workflow, as the credentials flow variable was not saved and will therefore not be available to downstream nodes.

Ports Options Views

Input ports

Type:
LLM
A Large Language Model.

Type:
Prompt Table
A table containing a string column with prompts.

Output ports

Type:
Result Table
A table containing prompts and their respective answer.

Extension
KNIME AI Extension
provided by KNIME AG, Zurich, Switzerland

1: Result Table   Flow Variables

Rows: 1 | Columns: 4

#	RowID	Question (String)	Retrieved documents (Collection (List))	engineered_prompt (String)	Response (String)
1	Row0	Do you send products abroad?	[Do you offer international shipping? Yes, we ship to over 50 countries. Shipping tim	You are an helpful assistant for a cutting-edge company Use the following pieces of context from a list of FAQ questions about the company If you don't know the answer, just say that you don't know, don't try to make up an an Context from the FAQ list: {Do you offer international shipping? Yes, we ship to over 50 countries. Shipping tim Question: {Do you send products abroad?} Helpful Answer:	Yes, we offer international shipping to over 50 countries. Shipping times and fees m

Figure 20. Output of the RAG process: the grounded response generated by the LLM.

Example: Product FAQ Assistant with RAG

You want to build an assistant that can answer questions about your company's products and services. To do this, you decide to use a Retrieval-Augmented Generation (RAG) architecture, with a file containing Frequently Asked Questions (FAQs) as your knowledge base.

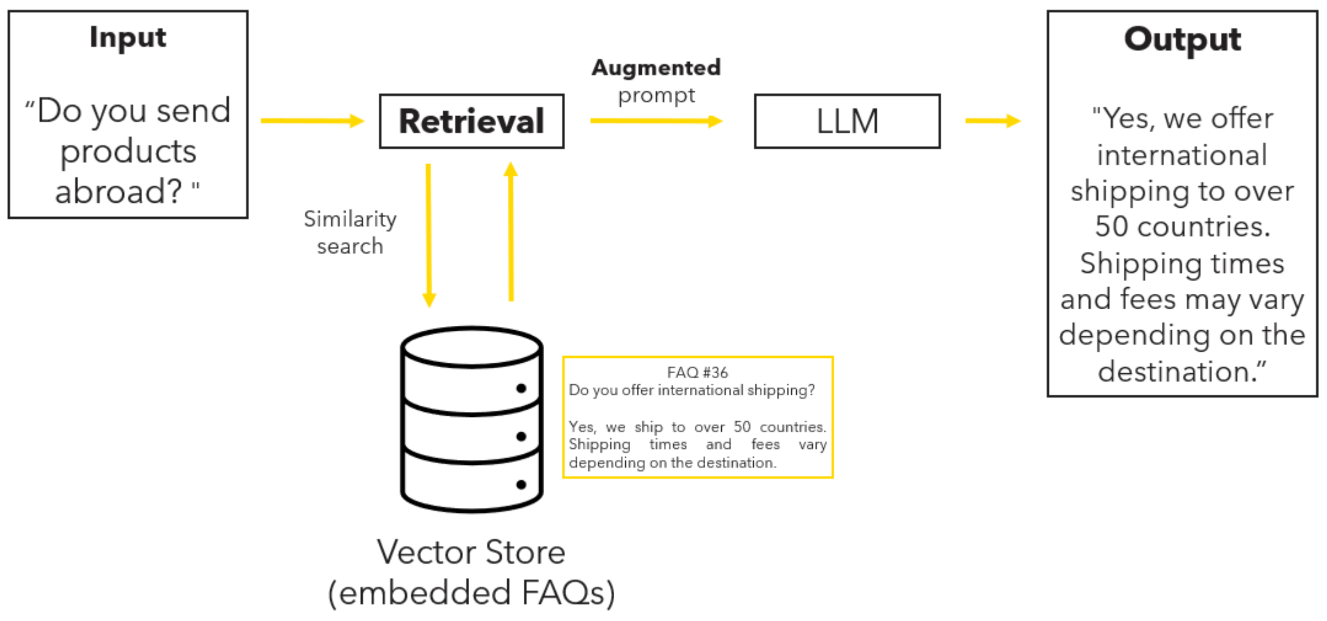

Input
“Do you send products abroad? "
Retrieval
Augmented prompt
LLM
Output
"Yes, we offer international shipping to over 50 countries. Shipping times and fees may vary depending on the destination.”
Similarity search
FAQ #36
Do you offer international shipping?
Yes, we ship to over 50 countries. Shipping times and fees vary depending on the destination.
Vector Store (embedded FAQs)

Figure 21. An overview of a RAG pipeline

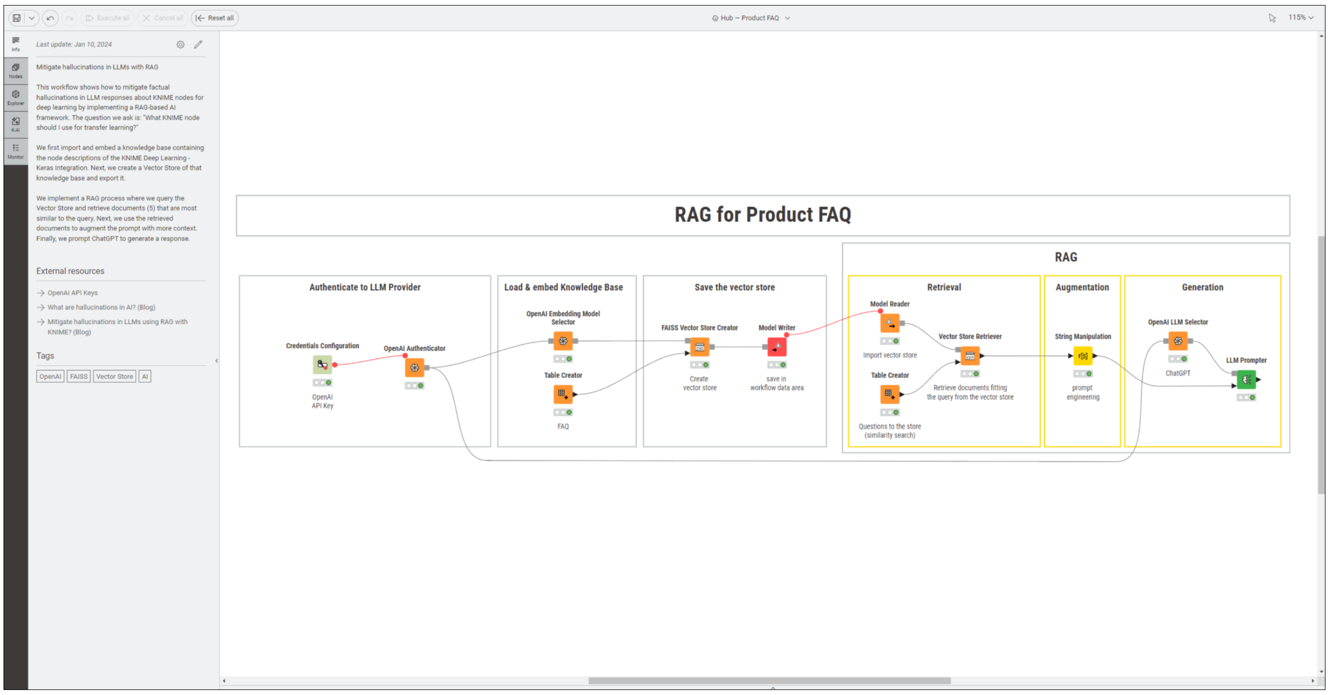
Sample FAQ data (CSV)

This is what the file containing FAQs looks like:

ID	FAQ
1	What is the return policy? You have the right to return a product within 30 days
2	How can I reset my password? You can reset your password by clicking 'Forgot Password' on the login page and following the instructions.
3	Do you offer international shipping? Yes, we ship to over 50 countries. Shipping times and fees vary depending on the destination.

Workflow: FAQ Product Assistant

This workflow implements Retrieval-Augmented Generation (RAG) on a product FAQ file. It includes three main steps: authentication, vector store creation, and retrieval-augmented generation.


Last update: Jan 10, 2024

Mitigate hallucinations in LLMs with RAG

This workflow shows how to mitigate factual hallucinations in LLM responses about KNIME nodes for deep learning by implementing a RAG-based AI framework. The question we ask is: "What KNIME node should I use for transfer learning?"

We first import and embed a knowledge base containing the node descriptions of the KNIME Deep Learning - Keras Integration. Next, we create a Vector Store of that knowledge base and export it.

We implement a RAG process where we query the Vector Store and retrieve documents (5) that are most similar to the query. Next, we use the retrieved documents to augment the prompt with more context. Finally, we prompt ChatGPT to generate a response.

External resources

- OpenAI API Keys
- What are hallucinations in AI? (Blog)
- Mitigate hallucinations in LLMs using RAG with KNIME? (Blog)

Tags

OpenAI FAISS Vector Store AI

Figure 22. The Workflow that uses a RAG architecture to answer FAQs

1. Authenticate

- **Provide API credentials**

  Use the Credentials Configuration node to store the OpenAI API key.

- **Authenticate**

  The OpenAI Authenticator node authenticates the connection to OpenAI.

2. Create Vector Store

- **Select embedding model**

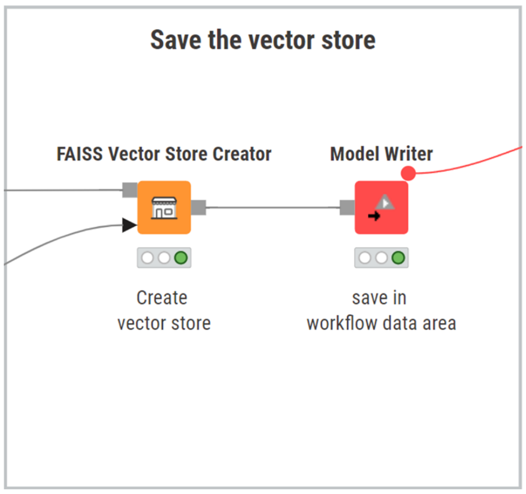
  The OpenAI Embedding Model Selector node selects the embedding model (text-embedding-3-small).

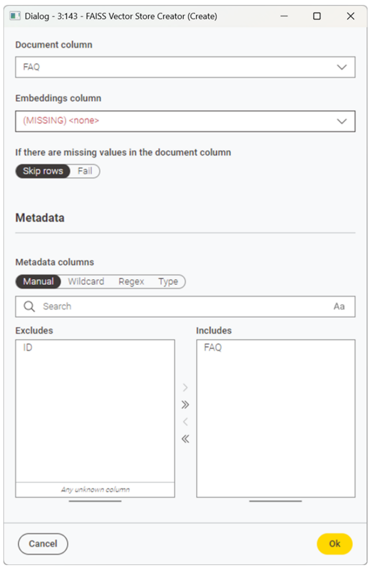
- **Read data**

  CSV Reader node loads the FAQ file.

- **Create Vector Store**

  The FAISS Vector Store Creator creates embeddings and store them in a Vector Store




Dialog - 3:143 - FAISS Vector Store Creator (Create)

Document column: FAQ

Embeddings column: (MISSING) <none>

If there are missing values in the document column: [✓ Skip rows] Fail

Metadata

Metadata columns: [✓ Manual] Wildcard Regex Type

Excludes: ID
Any unknown column

Includes: FAQ

Figure 23. Configuration dialog of the FAISS Vector Store Creator. In this example, the node generates embeddings internally because no precomputed embeddings are provided. The AI Extension also includes a dedicated node for embedding generation: the Text Embedder (see earlier section).

3. Retrieval Augmented Generation (RAG)

Retrieval

- **Load Vector Store**

  The Model Reader node loads the saved Vector Store from disk. The FAISS Vector Store Reader node brings the store into memory.

- **Provide user query**

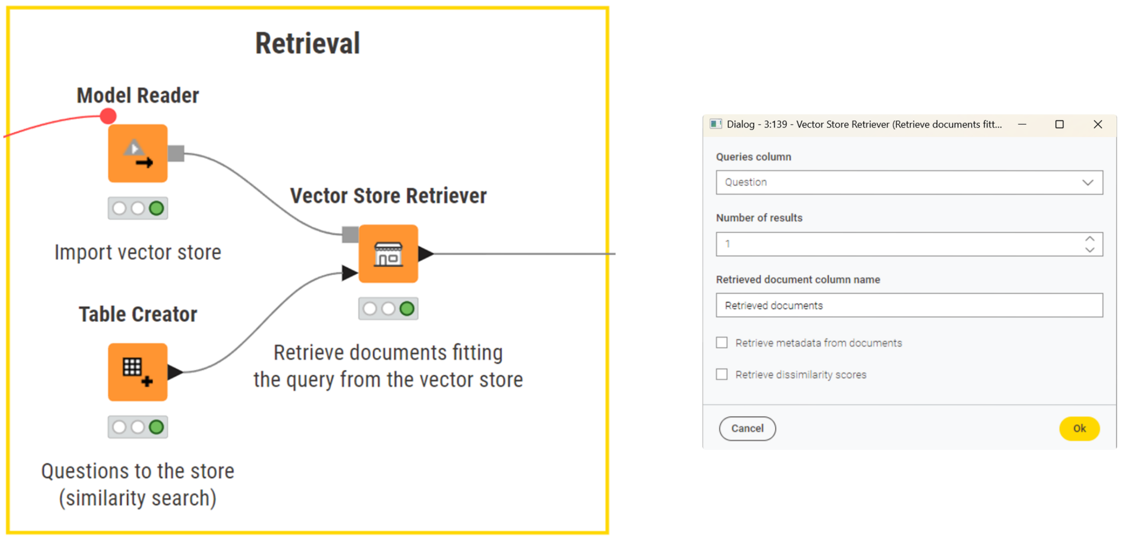
  The Table Creator node simulates a user query.

- **Generate query embedding**

  The OpenAI Embeddings Connector node generates an embedding for the user query using the same embedding model.

- **Retrieve similar entries**

  The Vector Store Retriever node compares the query embedding against the stored embeddings to retrieve the most similar FAQ entries. In this example, the number of retrieved results is set to 1 due to the small dataset. In real use cases, retrieving multiple results can improve grounding by providing the model with more context.


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)
Dialog - 3:139 - Vector Store Retriever (Retrieve documents fitt...
Cancel
Ok

Queries column: Question

Number of results: 1

Retrieved document column name: Retrieved documents

☐ Retrieve metadata from documents

☐ Retrieve dissimilarity scores

Figure 24. Configuration dialog of the Vector Store Retriever node.

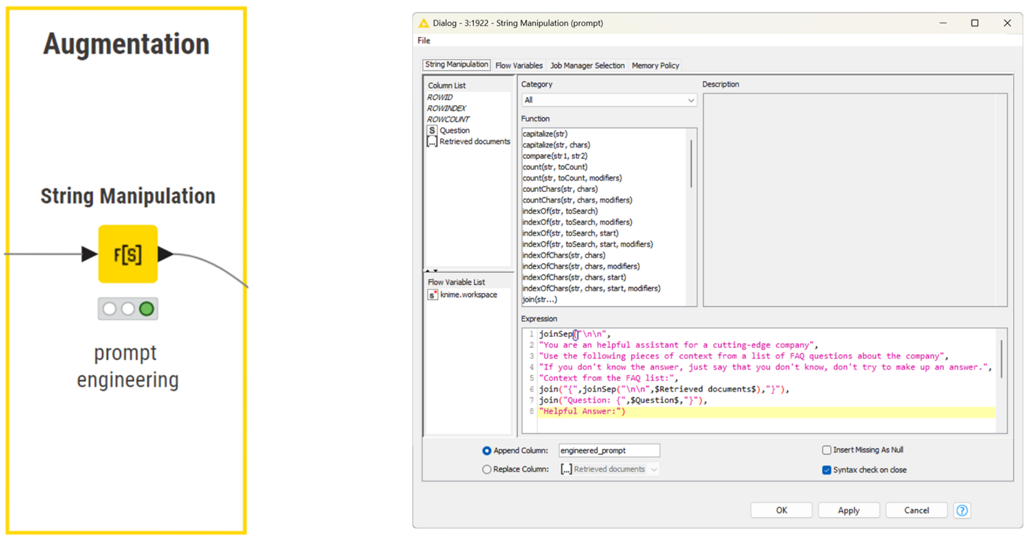
Augmentation

- **Prepare context**

  The String Manipulation node merges multiple retrieved FAQ answers into a single string using:

  ```
  JOINSEP("\n", column("Answer"))
  ```

This creates a combined context block for the language model.



Expression

```
joinSep("\n\n",
"You are an helpful assistant for a cutting-edge company",
"Use the following pieces of context from a list of FAQ questions about the company",
"If you don't know the answer, just say that you don't know, don't try to make up an answer.",
"Context from the FAQ list:",
join("{",joinSep("\n\n",$Retrieved documents$),"}"),
join("Question: {",$Question$,"}"),
"Helpful Answer:")
```

● Append Column: engineered_prompt

○ Replace Column: Retrieved documents

☐ Insert Missing As Null

☑ Syntax check on close

Figure 25. Prompt constructed using the String Manipulation node.

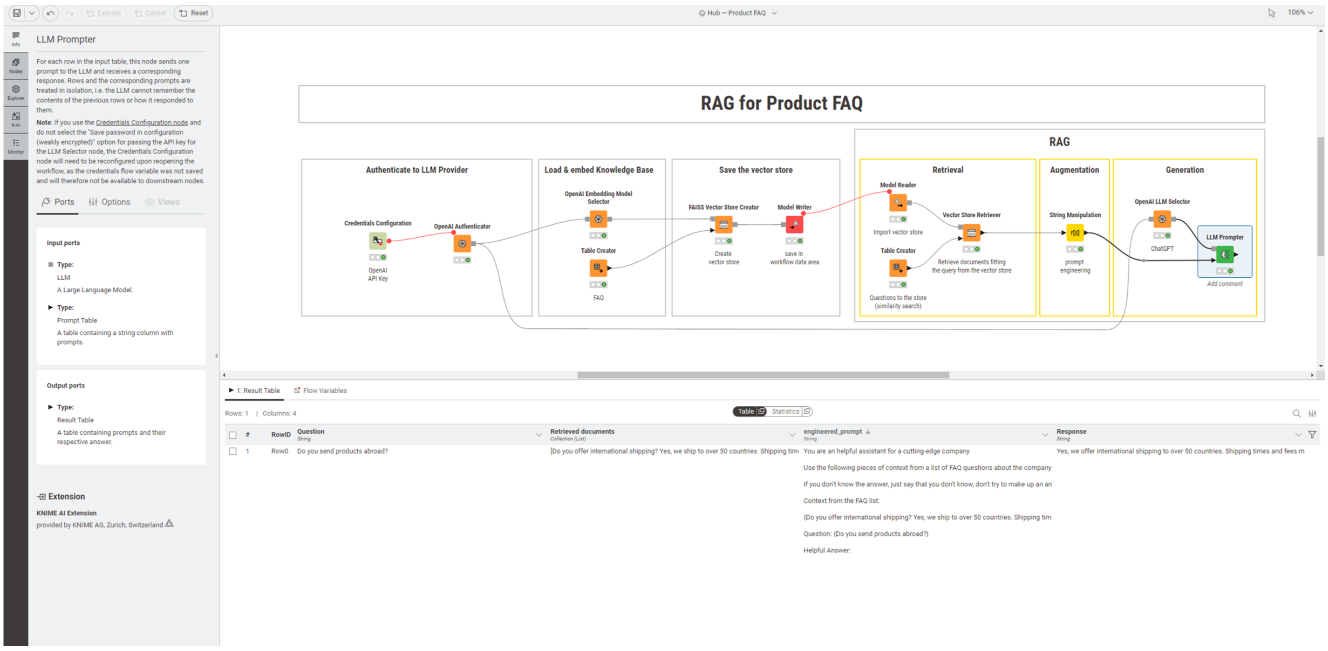
Generation

- **Send Prompt to model**

  The LLM Prompter node sends both the user question and the retrieved FAQ context to the language model.

- **Output Response**

  The model's reply is written into a new column called *Response*.



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.

Note: If you use the Credentials Configuration node and do not select the "Save password in configuration (weakly encrypted)" option for passing the API key for the LLM Selector node, the Credentials Configuration node will need to be reconfigured upon reopening the workflow, as the credentials flow variable was not saved and will therefore not be available to downstream nodes.

Ports | Options | Views

Input ports

Type:
LLM
A Large Language Model.

Type:
Prompt Table
A table containing a string column with prompts.

Output ports

Type:
Result Table
A table containing prompts and their respective answer.

Extension

KNIME AI Extension
provided by KNIME AG, Zurich, Switzerland

Rows: 1 | Columns: 4

#	RowID	Question (String)	Retrieved documents (Collection (List))	engineered_prompt (String)	Response (String)
1	Row0	Do you send products abroad?	[Do you offer international shipping? Yes, we ship to over 50 countries. Shipping tim	You are an helpful assistant for a cutting-edge company Use the following pieces of context from a list of FAQ questions about the company If you don't know the answer, just say that you don't know, don't try to make up an an Context from the FAQ list: {Do you offer international shipping? Yes, we ship to over 50 countries. Shipping tim Question: {Do you send products abroad?} Helpful Answer:	Yes, we offer international shipping to over 50 countries. Shipping times and fees m

Figure 26. LLM output preview in the response column

Agentic AI in KNIME

An **agent** in KNIME is a workflow that uses a large language model (LLM) to solve tasks. It can read a user's request, think step by step, and run other workflows, called **tools**, to complete the task.

To build an agentic workflow in KNIME Analytics Platform, follow these steps:

1. Connect to LLM provider
2. Access tools
3. Prompt agent
4. Provide data input (optional)

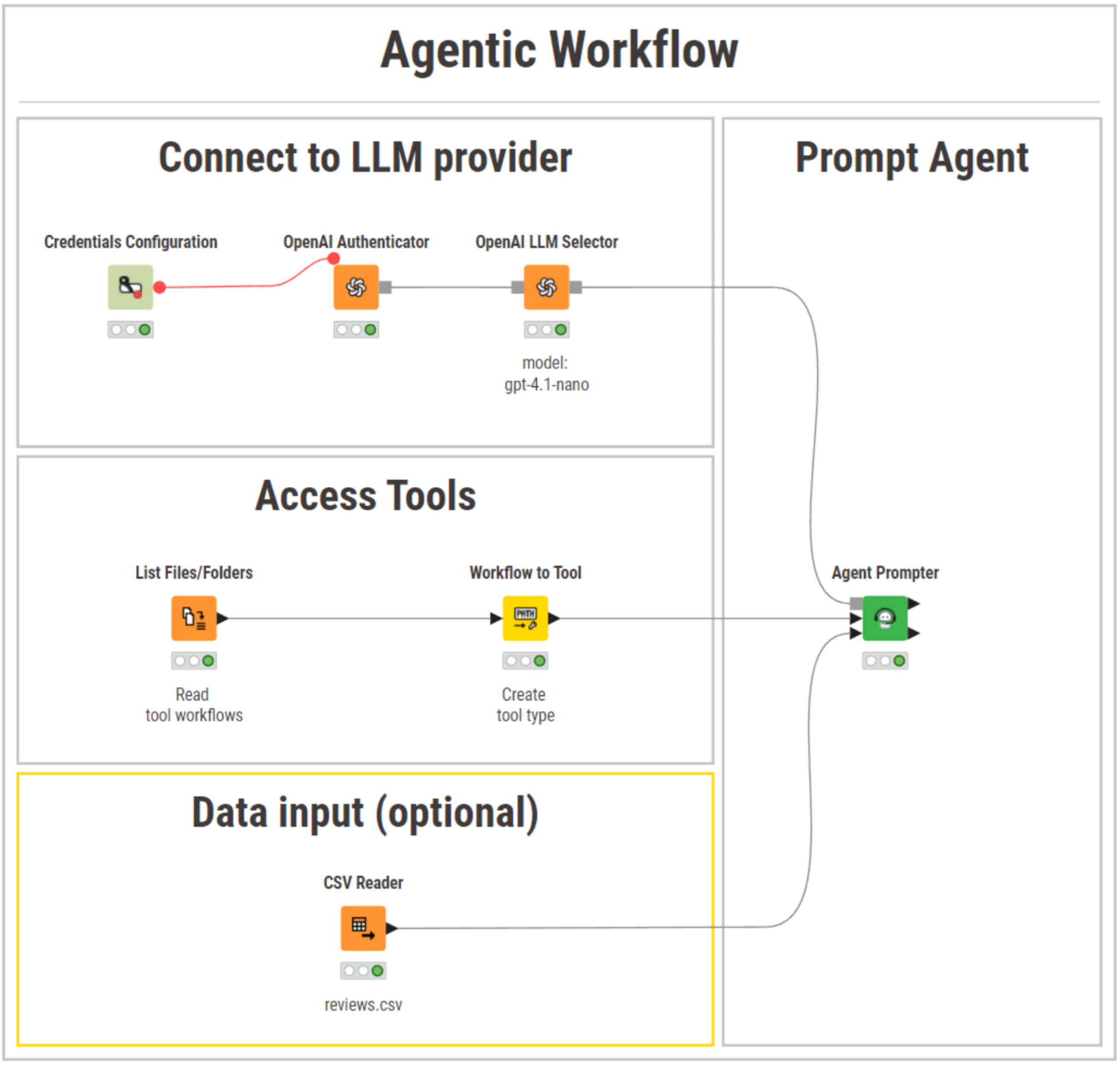


Figure 27. The architecture of an agentic workflow in KNIME, showing the main steps: connect to an LLM provider, access the available tools, and prompt the agent to start reasoning. Optionally, data can be provided as input.

Connect to LLM Provider

To give your agent reasoning capabilities, connect it to an LLM.

KNIME provides dedicated nodes for this:

- Authenticator nodes (e.g., OpenAI Authenticator)

  Stores your API key or token to access the chosen language model provider.

- LLM Selector nodes (e.g., OpenAI LLM Selector):

  Lets you select the language model (e.g., *GPT-4.1-nano*) that powers the agent's reasoning process.

For supported models and configuration tips, see the LLM reference table.

Access Tools

Tools are callable workflows that perform individual subtasks, like answering a question, checking a condition, or transforming a dataset.

To make tools available to the agent, use:

- List Files/Folders node

  Scans the folder where your tool workflows are saved (*.knwf* files).

- Workflow to Tool node

  Converts each workflow into a tool the agent can reason about.

This step ensures the agent knows:

- What each tool does (via its description).
- What inputs it needs (parameters or datasets).
- What output it returns (messages or tables).

For guidance on how to structure a tool workflow, see the section Turn a Workflow Into a Tool.

Prompt Agent

After connecting a language model and registering tools, you can prompt the agent to begin reasoning. KNIME provides two nodes for this:

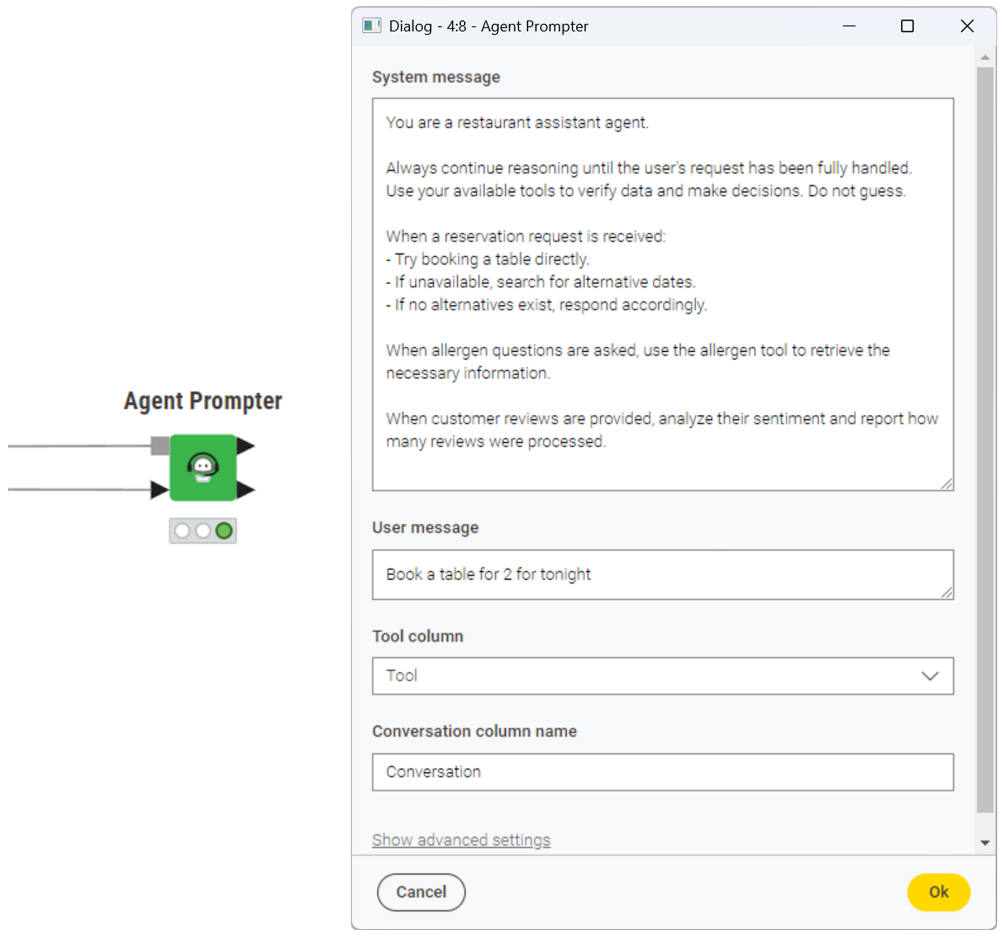
- Agent Prompter
- Agent Chat View

Each supports different use cases, as described below.

Agent Prompter node

Use the Agent Prompter node to start the agent's reasoning loop from within a workflow.

This node is best for automated execution or debugging single prompts in workflows, without user interaction.



System message

You are a restaurant assistant agent.

Always continue reasoning until the user's request has been fully handled.
Use your available tools to verify data and make decisions. Do not guess.

When a reservation request is received:
- Try booking a table directly.
- If unavailable, search for alternative dates.
- If no alternatives exist, respond accordingly.

When allergen questions are asked, use the allergen tool to retrieve the necessary information.

When customer reviews are provided, analyze their sentiment and report how many reviews were processed.

User message

Book a table for 2 for tonight

Tool column

Tool

Conversation column name

Conversation

Figure 28. The Agent Prompter node configuration dialogue.

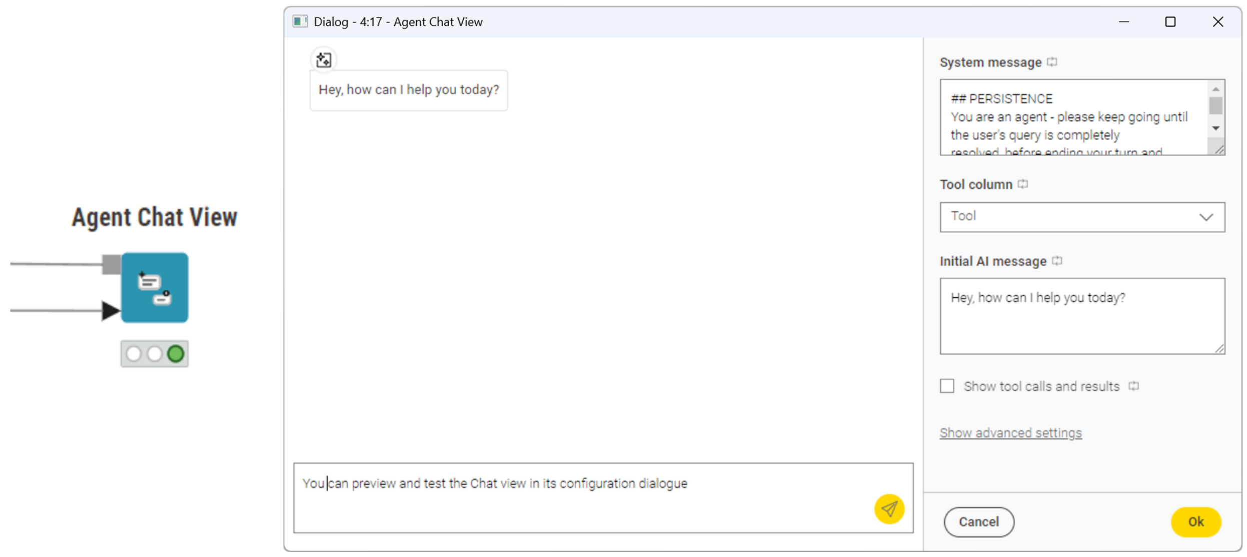
The node takes the following inputs:

- *System Message*: defines the agent's role, behavior, or rules. *Example*: `"You are a support agent answering product questions."`
- *User Message*: the actual task or question to solve. *Example*: `"What is the warranty for product X?"`
- *Tool List*: the set of available tools, generated using the Workflow to Tool node.

Agent Chat View node

To make your agent interactive, use the Agent Chat View node. This opens a live chat interface where users can talk to the agent, ask questions, and receive responses in real time.

Use this node when you want to build an assistant-style interfacefor end users.



Hey, how can I help you today?

You can preview and test the Chat view in its configuration dialogue

System message

PERSISTENCE
You are an agent - please keep going until the user's query is completely resolved, before ending your turn and

Tool column

Tool

Initial AI message

Hey, how can I help you today?

☐ Show tool calls and results

Show advanced settings

Figure 29. The configuration dialogue of the Agent Chat View node

This node takes:

- *System Message*: defines the agent's role, purpose, and behavior for the conversation.
- *Tool Column*: a column with the list of available tools (from the Workflow to Tool node)
- *Initial AI message (optional)*: a greeting or opening message to show before the user sends anything *Example*: *"Hey, how can I help you today?"*

During execution, users can type questions or requests. The agent reasons through the request, uses tools if needed, and responds in real time.

You can also choose what users see:

- If you tick *Show tool calls and results*, the interface displays the full conversation. This includes the internal reasoning, tool usage and responses.
- If you leave it unticked, the user sees only the agent's final replies, making the interaction feel more like a typical assistant chat.

You can embed the Agent Chat View inside a component, then deploy it to KNIME Hub to make your agent available as an interactive service for end users.

Input Data

Some tools operate on data tables. For example, filtering records, analyzing reviews, or generating summaries. In these cases, the agent must pass data to the tool and optionally retrieve the result after processing.

By default, the Agent Prompter and Agent Chat View nodes do not include data ports.

To enable data exchange:

1. Click the node in the workflow editor.
2. Select the plus icon that appears.
3. Choose *Add Input Port* or *Add Output Port*, and select a data table type.

The Agent Prompter node allows both input and output ports. The Agent Chat View supports input only.

Note that the agent itself never inspects the table directly. Instead, it calls a tool that processes the data and returns a result. The agent then uses that result to continue reasoning.

For details on how data flows through tools, see the dedicated section on the data layer.

How to create Tools

For an agent to use a workflow as a tool, the workflow must be structured in a way that the agent can understand and execute. This section shows how to prepare a KNIME workflow so the agent can reason about it, pass in inputs, and retrieve results.

Each Tool interacts with the agent through two layers:

1. **Communication layer**

   Handles the exchange of messages between the agent and the tool. It enables the agent to decide which tool to use, why, and how, based on user intent and the task at hand.

2. **Data layer**

   Manages the actual flow of data. Although the agent cannot directly view data tables, it can activate tools that operate on data in the background and return results.

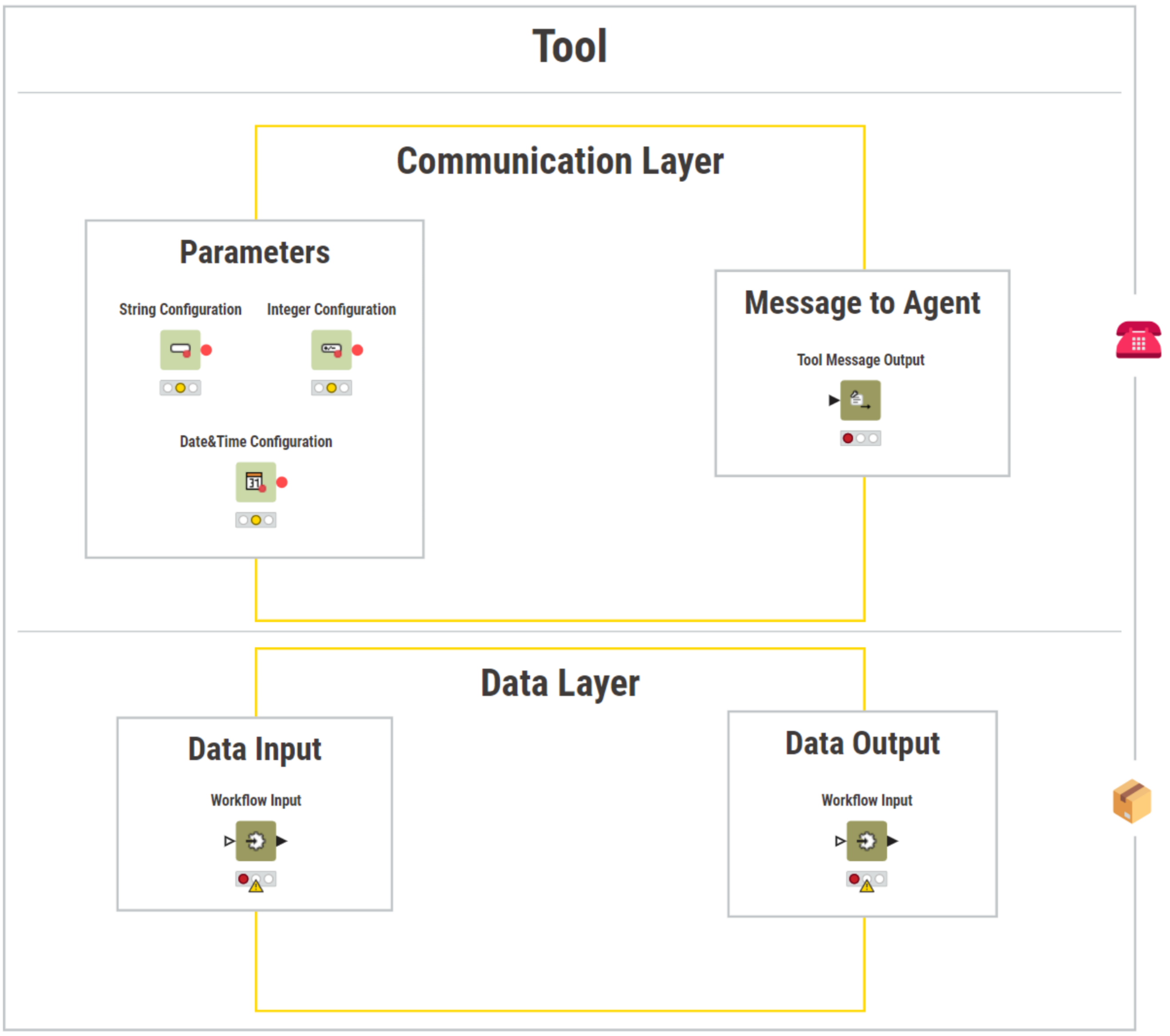


Figure 30. Structure of a tool workflow in KNIME Analytics Platform

Communication layer: Guide the Agent’s reasoning

Describe Tool behavior for the Agent

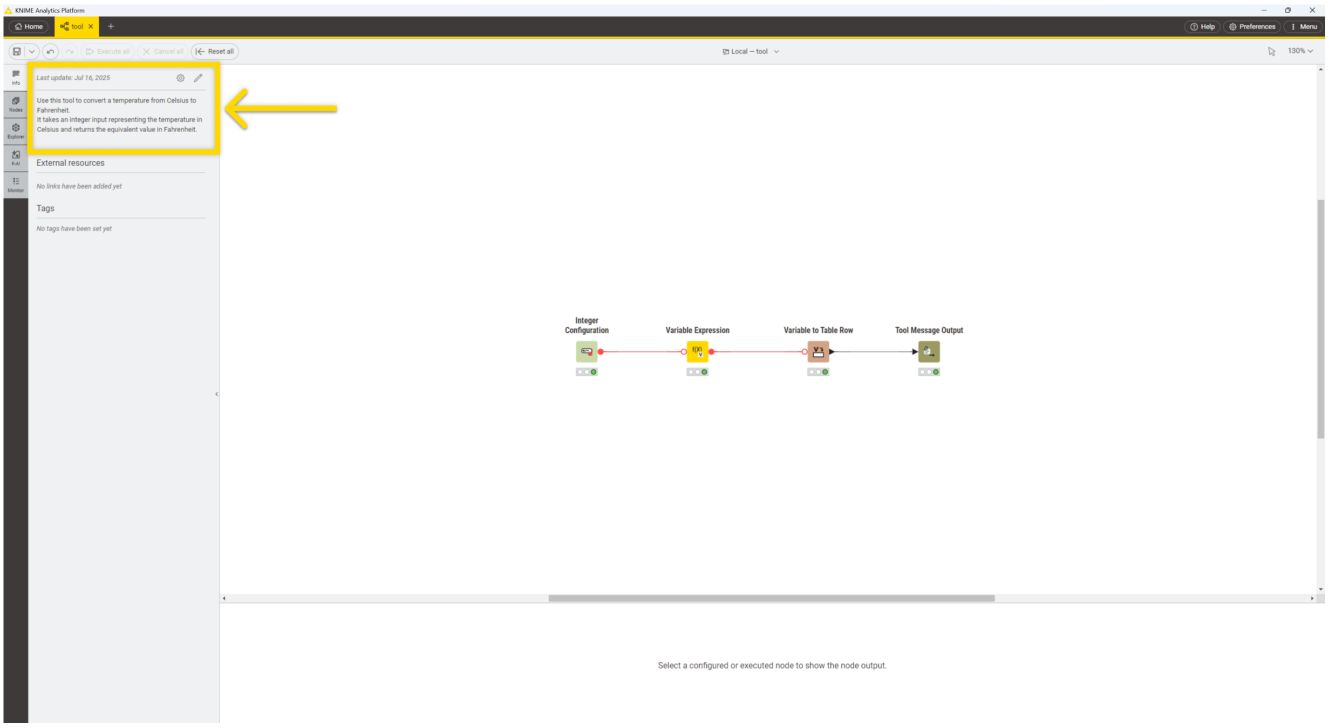
Each tool must include a clear description in the workflow’s Info section. The agent reads this description to determine when and how the tool should be used.

The description should include:

- What the tool does

- What inputs it expects
- What it returns
- The kinds of requests it can handle

A precise description improves the agent’s ability to reason about which tool to use in response to a user’s request.


Last update: Jul 16, 2025

Use this tool to convert a temperature from Celsius to Fahrenheit.
It takes an integer input representing the temperature in Celsius and returns the equivalent value in Fahrenheit.

External resources

No links have been added yet

Tags

No tags have been set yet

Select a configured or executed node to show the node output.

Figure 31. The tool description field in KNIME Analytics Platform. This is used by the agent to understand when to call the tool.

Return Results to the Agent

To send back reasoning-relevant feedback to the agent, use the Tool Message Output node.

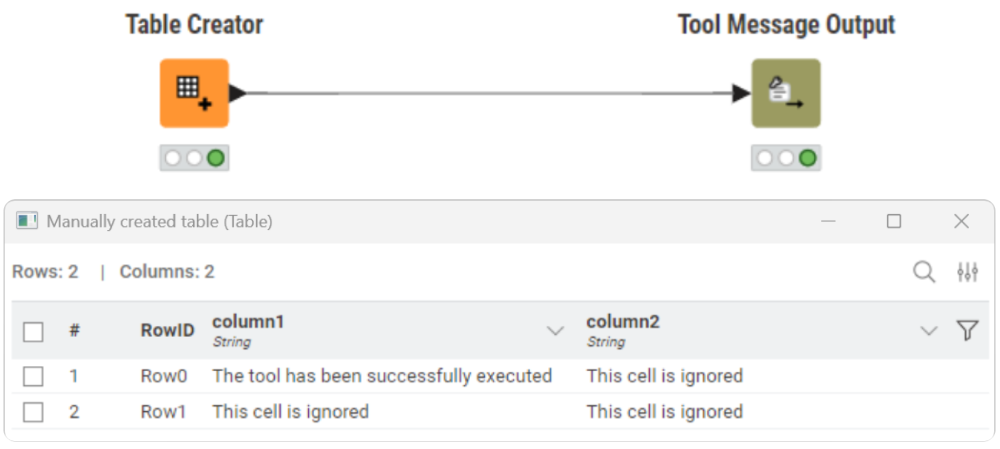
- Add this node when the tool needs to return a message the agent can read and reason with.
- Leave it out if the tool returns only structured data.

The node reads the first cell (row 1, column 1) of its input table and sends it as the result message to the agent.

Use it to return:

- Summaries of processed data.
- Key findings or confirmations.

- Intermediate reasoning steps.



#	RowID	column1 (String)	column2 (String)
1	Row0	The tool has been successfully executed	This cell is ignored
2	Row1	This cell is ignored	This cell is ignored

Figure 32. A table passed to the Tool Message Output node. Only the first cell (first row, first column) is used as the output message.

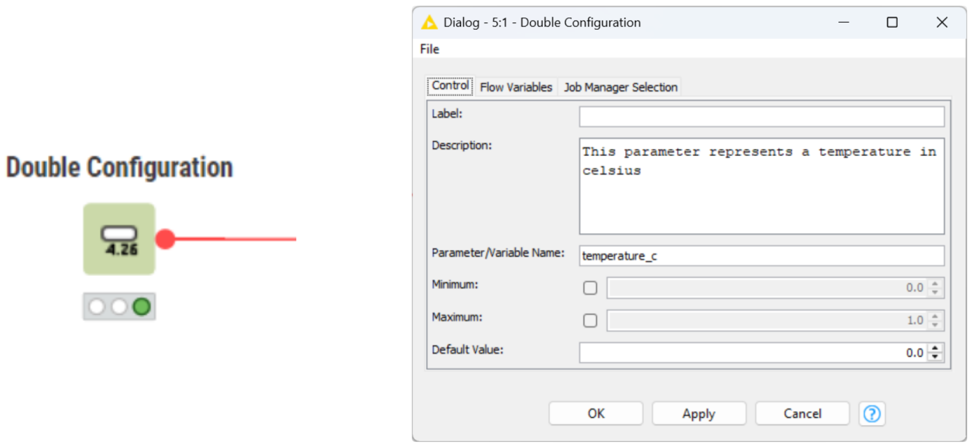
Let the Agent Set Parameters

Use KNIME Configuration nodes (such as String Configuration, Integer Configuration) to define adjustable parameters.

Each parameter should have:

- A clear name (used as a variable)
- A short description explaining what it controls

The agent uses this metadata to assign values to parameters dynamically.



Dialog - 5:1 - Double Configuration

File

Control | Flow Variables | Job Manager Selection

Label:	
Description:	This parameter represents a temperature in celsius
Parameter/Variable Name:	temperature_c
Minimum:	☐ 0.0
Maximum:	☐ 1.0
Default Value:	0.0

OK | Apply | Cancel

Figure 33. The configuration window for a parameter, with fields for Name and Description to guide the agent.

Data layer: automate data flow

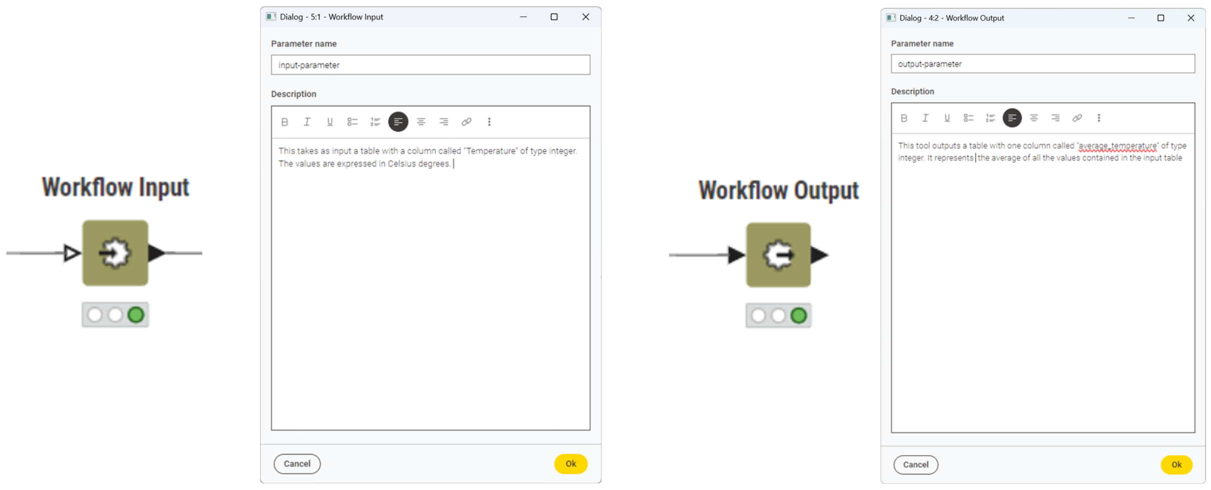
Define what data the Tool uses and returns

Use the Workflow Input node to specify the structure of data input in a tool.

In the node configuration dialog, add a clear description of the input table that the tool expects. Be sure to include column names and data types.

Use the Workflow Output node to describe the data produced by the tool.

Providing a description of the output table helps the agent understand how to use the result.



Dialog - 5:1 - Workflow Input

Parameter name: input-parameter

Description: This takes as input a table with a column called "Temperature" of type integer. The values are expressed in Celsius degrees.

Dialog - 4:2 - Workflow Output

Parameter name: output-parameter

Description: This tool outputs a table with one column called "average_temperature" of type integer. It represents the average of all the values contained in the input table

Figure 34. The Workflow Input and Output nodes contain descriptions of the data processed by the tool

The agent **does not** access raw data. Instead, it can call tools that handle data processing and return summaries or structured results.

Make Tools discoverable for the Agent

Once your Tool workflows are ready, you need to register them so the agent can find and use them.

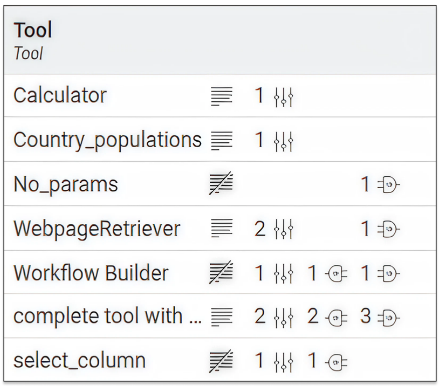
To do this:

1. Place all Tool workflows in a single folder.
2. In the same directory, create a new workflow to act as the tool registry.
3. Use the List Files/Folders node to read all workflows in the folder.
4. Pass the list to the Workflow to Tool node to convert each one into a callable Tool.

The output of the Workflow to Tool node includes metadata for each Tool, helping you verify they are correctly set up:

- ▨ shows whether a description is present.
- 1 shows how many parameters are defined.
- 2 3 shows how many data inputs and outputs are available.

This makes it easy to check at a glance whether tools are complete and ready to use.



Tool *Tool*				
Calculator	☰	1		
Country_populations	☰	1		
No_params	▨			1
WebpageRetriever	☰	2		1
Workflow Builder	▨	1	1	1
complete tool with ...	☰	2	2	3
select_column	▨	1	1	

Figure 35. Output of the Workflow to Tool node showing metadata for each tool.

Checklist: what you need to build an agent

Step	Task	Action
1. Design Tools	Describe	Add a clear tool workflow description (task, inputs, outputs, parameters)
	Parameters (optional)	Use Configuration nodes with clear names and descriptions.
	Communication layer (optional)	Add Tool Message Output node to return text to agent (reads first row, first column).
	Data Layer (optional)	Add Workflow Input/Output nodes if data needs to flow through the tool.

2. Build Tool list	Prepare Workflow	Create a separate agentic workflow to collect tools.
	List Tools	Use the List Files/Folders node to scan the folder with Tool workflows.
	Convert	Use Workflow to Tool node to generate Tool List.
	Verify	Check metadata with icons: description present, parameters defined, data ports assigned.
	Refresh	After modifying any Tool, save workflow and re-run Workflow to Tool node.
3. Configure Agent	Agent Prompter	Provide System Message, User Message, and Tool List.
	Data Ports	Manually add input/output ports if Tools require external data.
4. Optional Deployment	Interactive View	Use Agent Chat View for live conversations.
	Deployment	Wrap as KNIME Component and deploy via KNIME Business Hub.

Example: Build a Restaurant Assistant Agent

This example walks you through the process of building a restaurant assistant agent using the KNIME AI Extensions. The assistant is designed to support restaurant staff by handling common tasks through simple, conversational language.

Each step adds a new concept, starting with a simple tool and gradually introducing parameters, conditional logic, and data handling.

By the end, the agent will be able to:

- Answer questions about allergens with Tool 1
- Handle reservation requests with Tool 2
- Suggest alternative booking options with Tool 3
- Analyze customer reviews with Tool 4

Tool 1: Answer Allergen Questions (Communication layer)

This first tool introduces the simplest type of agent interaction: a tool that uses only the communication layer. It requires no parameters and no data input.

When a user asks a question about allergens, the agent can call this tool to retrieve a predefined string containing allergen information. The agent then uses this information to formulate its response.

1. Design the Tool

The tool workflow contains only two nodes:

- Table Creator node

  Use this node to create a one-row, one-column table containing allergen details for each menu item.

  Enter the following string as the table content:

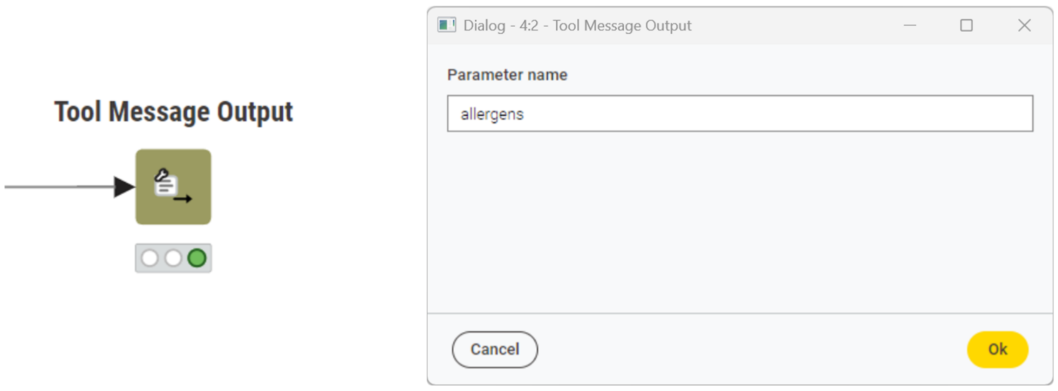
  ```
  Grilled Chicken: Gluten: No, Dairy: No, Nuts: No, Shellfish: No, Fish: No, Sesame: No
  Salmon Teriyaki: Gluten: No, Dairy: No, Nuts: No, Shellfish: No, Fish: Yes, Sesame: Yes
  Shrimp Tacos: Gluten: No, Dairy: Yes, Nuts: No, Shellfish: Yes, Fish: No, Sesame: No
  Vegan Burger: Gluten: Yes, Dairy: No, Nuts: No, Shellfish: No, Fish: No, Sesame: No
  Chocolate Cake: Gluten: Yes, Dairy: Yes, Nuts: Yes, Shellfish: No, Fish: No, Sesame: No
  Pad Thai: Gluten: No, Dairy: No, Nuts: Yes, Shellfish: No, Fish: No, Sesame: Yes
  ```

  The agent will use this information to answer questions like: *"Does the grilled chicken contain sesame?"* or *"Which dishes are gluten free?"*

- Tool Message Output node

  This node converts the table content into a message that the agent can read.

In the configuration dialogue, rename the parameter name to *allergens*. This makes it clearer for the agent to understand what kind of information is being returned, especially when multiple tools are available.



Parameter name: allergens

Figure 36. The Tool Message Output configuration. The parameter is renamed to allergens to better describe the content.

2. Describe the Tool

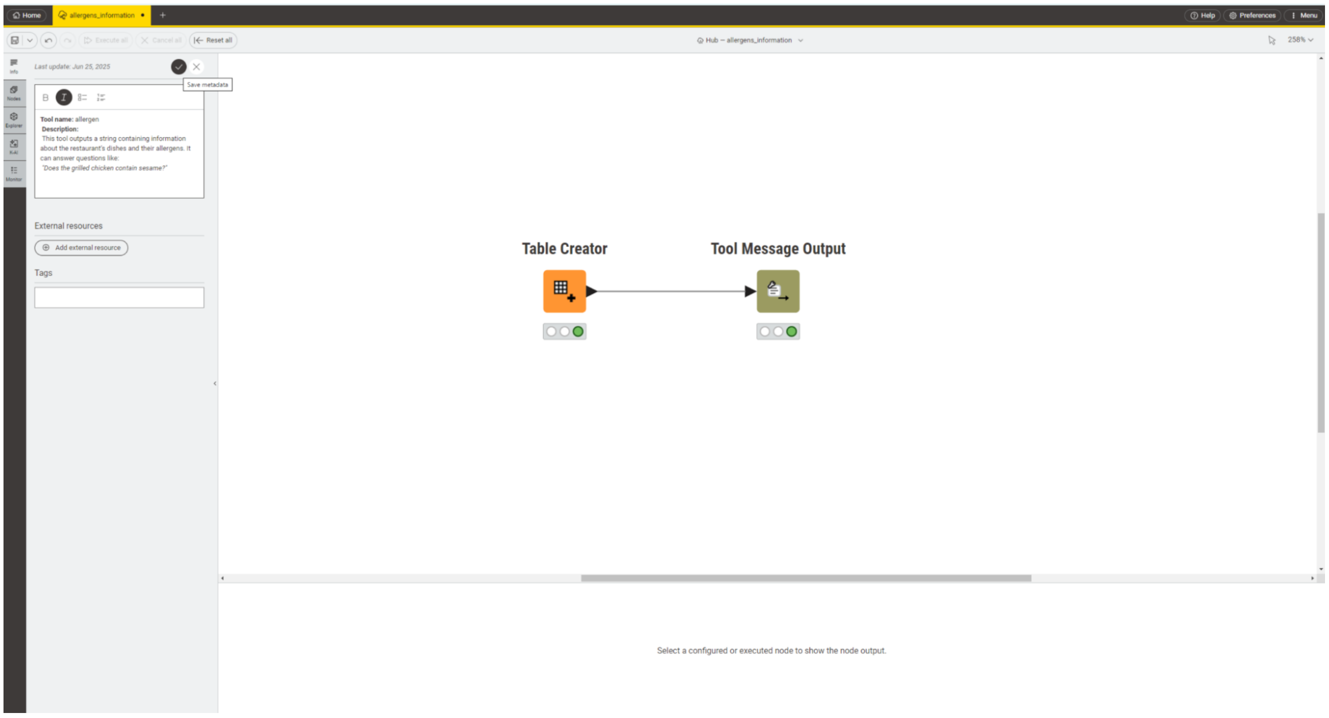
Once the workflow logic is complete, the last step is to describe the tool so the agent knows when to use it. This description is added in the *Workflow Description* field, found under the *Workflow Info* panel in KNIME Analytics Platform.

The agent will rely on this description to decide whether the tool matches a user's question. The more precise and informative the text, the more likely the agent will use the tool effectively.

Use the following:

```
Tool name: allergens_information
Description: This tool returns a string containing information about the restaurant's
dishes and their allergens. It can answer questions such as:
"Does the grilled chicken contain sesame?"
```

Once added, this description becomes part of the tool's metadata and will be picked up during registration via the Workflow to Tool node.



Tool name: allergen
Description:
This tool outputs a string containing information about the restaurant's dishes and their allergens. It can answer questions like:
"Does the grilled chicken contain sesame?"

Figure 37. tool workflow now has a description

Tool 2: Handle Booking Requests (Parameters)

This tool introduces parameterized tool calls. The agent reads the user request (e.g. the number of people and the desired date), extracts this information, and sends it as parameters to the tool workflow.

1. Add Parameters to the Tool

This tool uses two parameters:

- *number_people*: the number of seats the user wants to book
- *booking_date*: the requested date for the reservation

By giving each parameter a clear name and a short description, you help the agent understand what values to extract from the user's request.

Example:

User input: *I need a table for two people for 6/25/2025.*

Extracted parameters:

- *number_people*: 2

- *booking_date*: 2025-06-25

2. Check the Dataset

The tool works with a table of current availability, stored in a file called *restautant_reservations.csv*.

The table includes:

Table ID	Seats	Date	Time	Available
T1	2	2025-06-24	19:00	Yes
T2	4	2025-06-24	19:00	Yes
T3	6	2025-06-24	19:00	No
T4	4	2025-06-24	20:00	Yes
T5	2	2025-06-24	21:00	Yes

3. Design the Tool

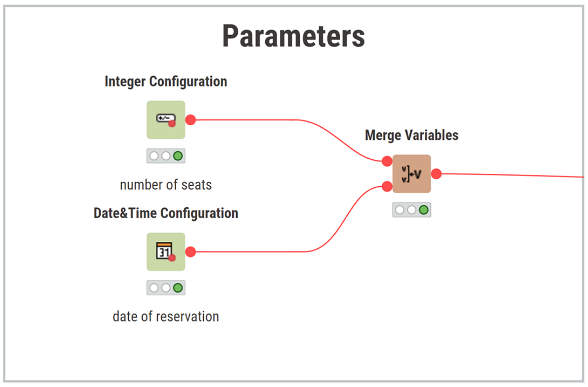
Configure parameters

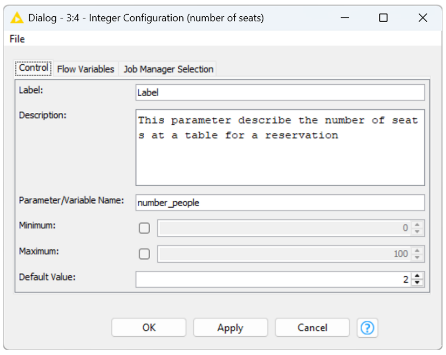
Use:

- Integer Configuration to collect *number_people*
- Date&Time Configuration to collect *booking_date*

Make sure both parameter nodes include **descriptive labels** and **short explanations**. This helps the agent understand what information to pass.

Merge the two parameters with a Merge Variable node to use them in the filtering logic.




Dialog - 3:4 - Integer Configuration (number of seats)

File

Control | Flow Variables | Job Manager Selection

Label:	Label
Description:	This parameter describe the number of seats at a table for a reservation
Parameter/Variable Name:	number_people
Minimum:	☐ 0
Maximum:	☐ 100
Default Value:	2

Figure 38. The configuration dialogue of the Integer configuration node

Filter results

Use an Empty Table Switch node to handle two paths:

- If matching tables are found:
  - Top K Row Filter node selects one available table.
  - *Update .csv Metanode* updates the restaurant_reservations.csv file by overwriting it, changing the table’s availability from "Yes" to "No" to register the booking.
  - Expression creates a confirmation message:

    ```
    string("The booking for table " + $["Table ID"] +
    " with " + $["Seats"] + " people, on " + $["Date"] +
    " was confirmed!")
    ```

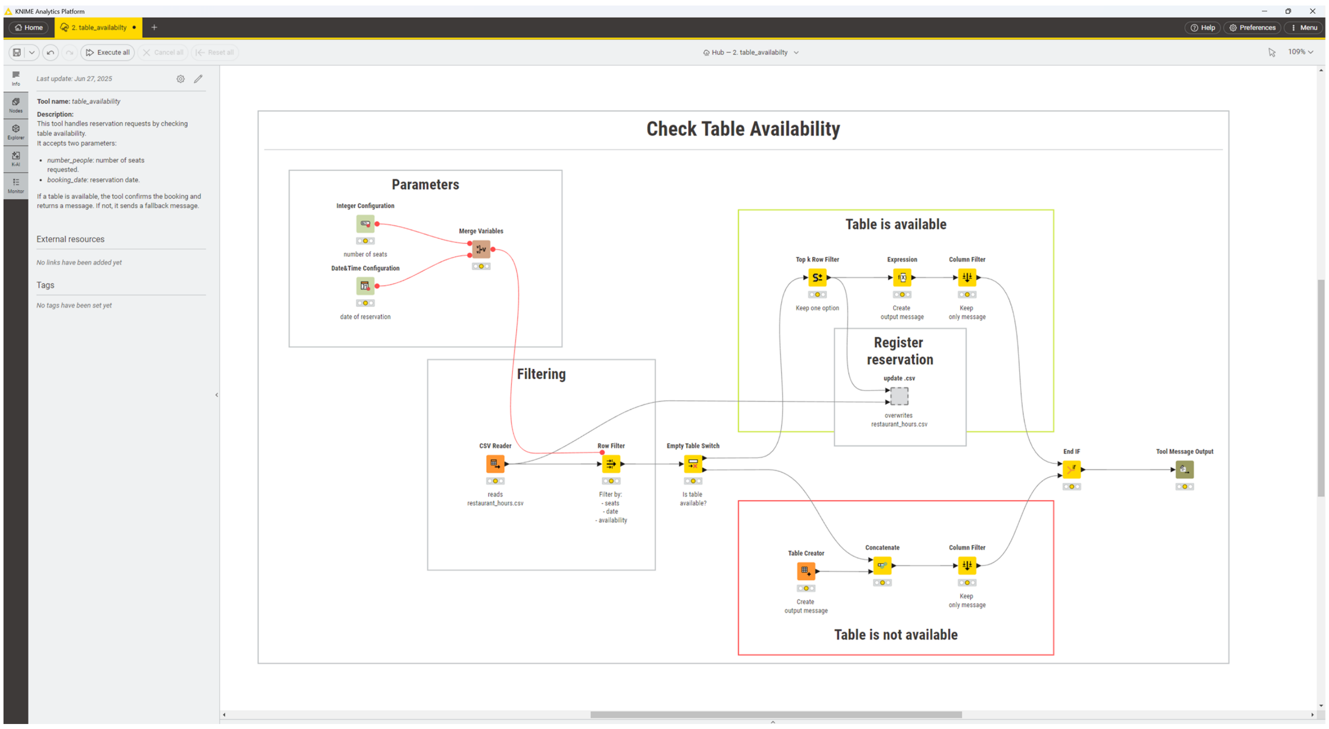
  - Column Filter keeps only the message column
- If no table is available:
  - Table Creator creates the fallback message: *"No tables are available for the desired date."*
  - Column Filter keeps only the message column.

Join the two branches with an END IF node to return a single result.

Configure the Communication Layer

Use the Tool Message Output node to send the final message to the agent. The parameter name is set to *table_availability* to clearly describe the content.

The tool now returns either a booking confirmation or an unavailability message, based on the input.



Last update: Jun 27, 2025

Tool name: *table_availability*

Description:
This tool handles reservation requests by checking table availability.
It accepts two parameters:

- *number_people*: number of seats requested.
- *booking_date*: reservation date.

If a table is available, the tool confirms the booking and returns a message. If not, it sends a fallback message.

External resources

No links have been added yet

Tags

No tags have been set yet

Figure 39. The Table Availability tool workflow with two paths: available or unavailable, ending in a single message output.

4. Describe the Tool

Once the tool workflow is complete, add a description in the *Workflow Info panel*. This helps the agent understand when to use it.

Use the following:

```
Tool name: table_availability
Description: This tool handles reservation requests by checking table availability.

It accepts two parameters:

- number_people: number of seats requested.
- booking_date: reservation date.

If a table is available, the tool confirms the booking and returns a message. If not, it
sends a fallback message.
```

Tool 3: Suggest Alternative Booking Dates (Concatenate tools)

This tool builds on the previous one by offering a fallback. If no tables are available on the requested date, the agent can use this tool to check availability for the next day and suggest an alternative.

1. Add parameters

The tool uses the same parameters as Tool 2:

- *number_people*
- *booking_date*

Use configuration nodes to collect these values. Then, apply a Date&Time Shift to move the *booking_date* one day forward.

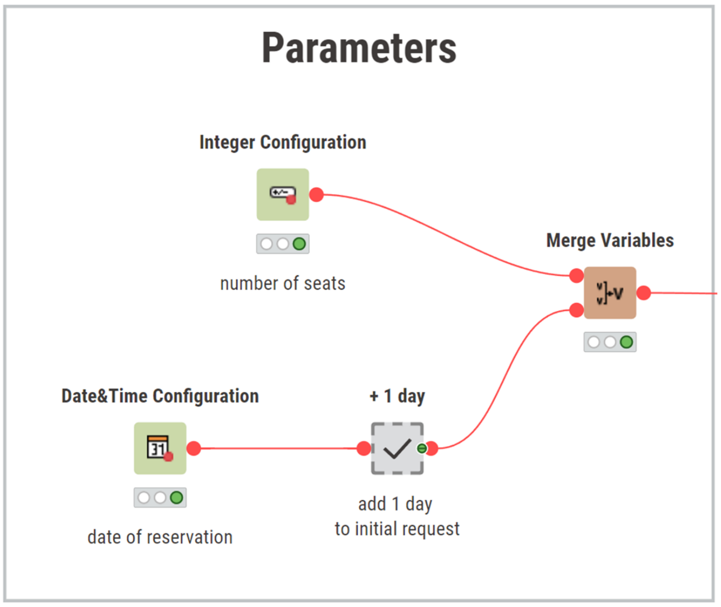
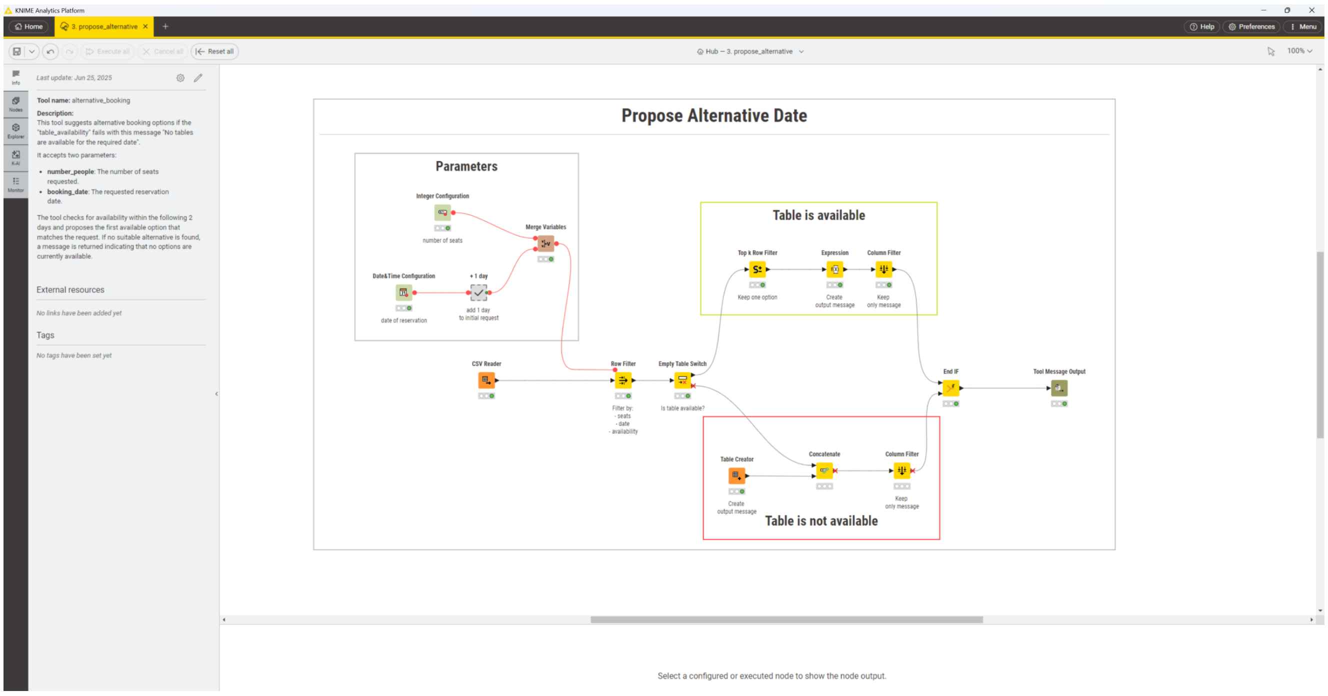


Figure 40. The parameter configuration of Propose Alternative tool. A Date&Time Shift adds one day to the requested reservation date to search for availability on the following day.

2. Workflow Design

The workflow is similar to Tool 2, with a key difference: it checks table availability for the day after the original request. If an alternative table is found, the tool returns a suggestion like: *"There is an alternative for the day after where table T2 with 4 people, on 2025-06-25 is free."* If nothing is available, the tool returns a fallback message. This tool does not confirm bookings. It only proposes alternatives.


Last update: Jun 25, 2025

Tool name: alternative_booking

Description:
This tool suggests alternative booking options if the "table_availability" fails with this message "No tables are available for the required date".

It accepts two parameters:

- **number_people**: The number of seats requested.
- **booking_date**: The requested reservation date.

The tool checks for availability within the following 2 days and proposes the first available option that matches the request. If no suitable alternative is found, a message is returned indicating that no options are currently available.

External resources

No links have been added yet

Tags

No tags have been set yet

Select a configured or executed node to show the node output.

Figure 41. The Propose Alternative tool workflow that proposes an alternative reservation date to the user

3. Communication layer

Use the Tool Message Output node to return the message. Set the parameter name to *alternative_booking*.

4. Describe the Tool

Go to the *Workflow Info* panel and add:

```
Tool name: alternative_booking

Description: This tool suggests alternative booking options if the requested date is
fully booked. It checks for availability on the following day and returns a suggestion
if an open table is found.
```

Tool 4: Analyze Customer Review Sentiment (Data Layer)

This tool introduces the **data layer**. It processes a table of customer reviews, analyzes the sentiment of each review using an LLM, and returns:

- A short message with the number of positive and negative reviews.
- A data table where each review is labeled as positive or negative.

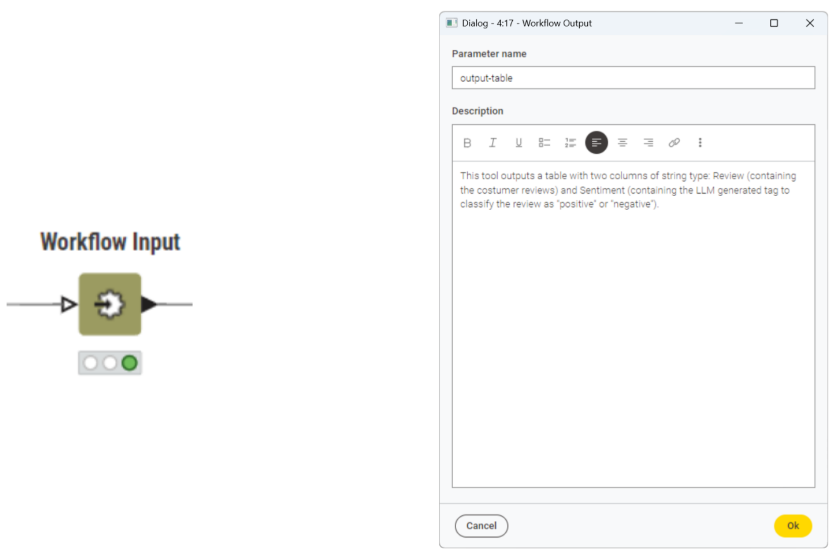
1. Define the Data Input

This tool receives a table with one column named Review, containing user-written feedback like:

Review
The food was amazing, great service!
Terrible experience. Long wait and cold food.

To allow the agent to pass this table into the tool, you need a Workflow Input node. This node defines what the tool expects in the data layer. The agent itself cannot see or inspect the data: it only triggers the tool and reads the resulting message.

It's useful to connect a small mock dataset (e.g., using a Table Creator) during development so you can test the tool's logic. This mock data is only used when the tool is run on its own. When called by the agent, the input is replaced with the actual data table provided in the agentic workflow.



output-table

This tool outputs a table with two columns of string type: Review (containing the costumer reviews) and Sentiment (containing the LLM generated tag to classify the review as "positive" or "negative").

Figure 42. The Workflow Input configuration dialogue

2. Workflow Design

The tool workflow has a data layer and a communication layer.

Data Layer

The data layer handles the ingestion and transformation of external data:

- Workflow Input

  Receives the input table from the agent. The input must contain a single column named Review.

- Expression

  Builds a prompt for each review using:

  ```
  “string("Is this review positive or negative? only return one label in lowercase:
  " + $["Review"])"
  ```

- LLM Prompter

  Sends the prompt to a selected model (e.g. GPT-4.1-nano) to classify the sentiment and outputs the model’s predictions as a new column called Sentiment.

- Workflow Output

  The model’s predictions are appended to the input table as a new Sentiment column. This enriched dataset is sent back to the agent if needed.

Communication Layer

the communication layer builds a natural language message the agent can reason with:

- Value Counter

  Counts how many reviews fall into each sentiment category (e.g., positive, negative).

- Table Transposer

  Converts counts to a row format so a single message can be built from it.

- Expression

  creates a message string such as:

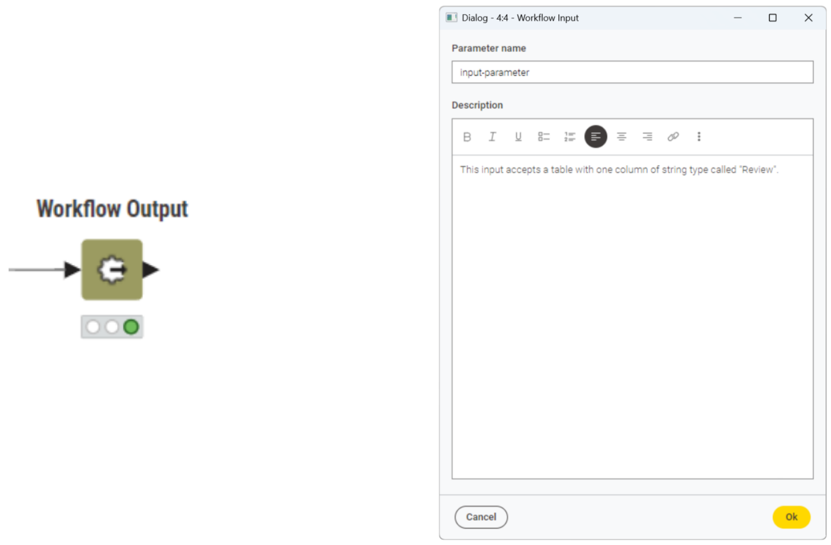
```
“There are 15 positive review(s) and 5 negative review(s).”
```

- Column Filter

  Keeps only the message column (first row, first cell required by Tool Message Output).

- Tool Message Output

  Sends the final summary message to the agent. The parameter is named review_summary.



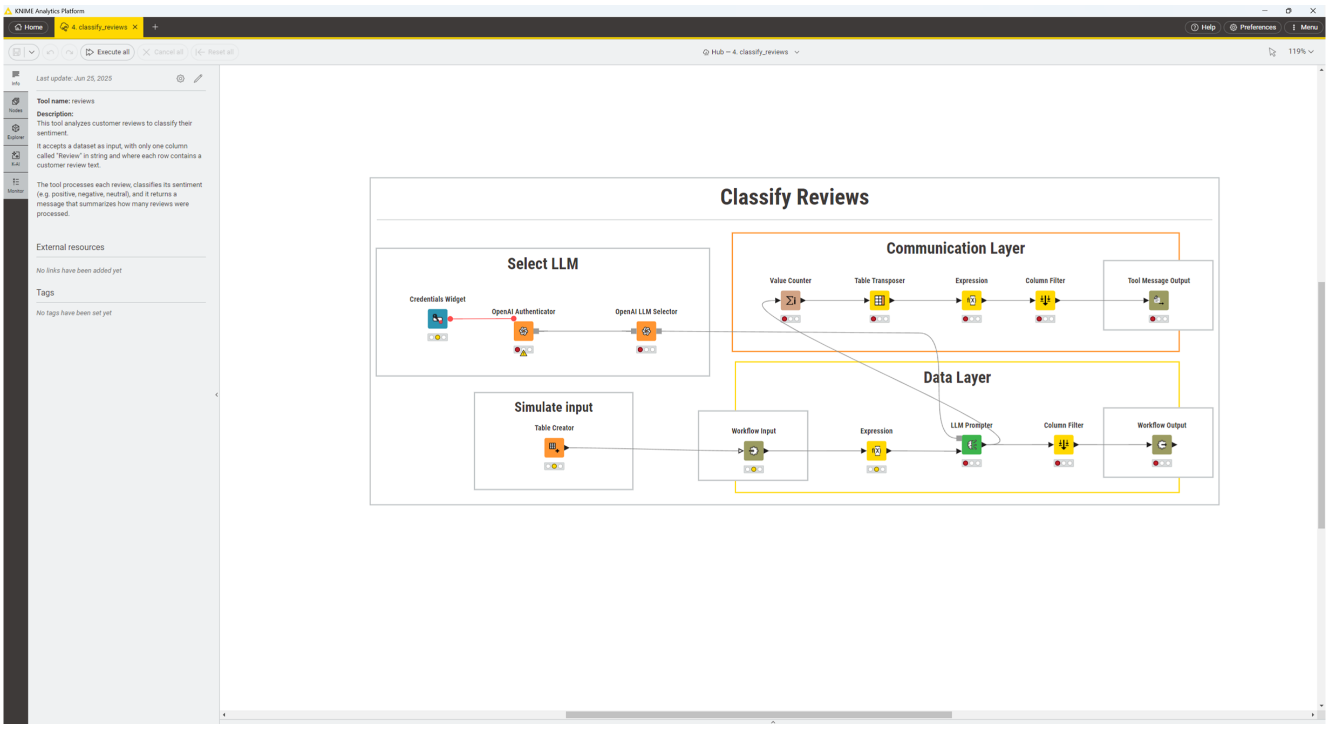
Parameter name

input-parameter

Description

This input accepts a table with one column of string type called "Review".

Figure 43. Workflow Output node configuration


Last update: Jun 25, 2025

Tool name: reviews

Description:
This tool analyzes customer reviews to classify their sentiment.

It accepts a dataset as input, with only one column called "Review" in string and where each row contains a customer review text.

The tool processes each review, classifies its sentiment (e.g. positive, negative, neutral), and it returns a message that summarizes how many reviews were processed.

External resources

No links have been added yet

Tags

No tags have been set yet

Figure 44. The Classify Reviews tool workflow that classifies reviews using an LLM.

The agent only reads the message from Tool Message Output. If needed, the enriched table is available as a data output for other tools.

3. Describe the Tool

Open the *Workflow Info* panel and add:

```
Tool name: classify_reviews

Description: This tool analyzes customer reviews to classify their sentiment. It accepts
a dataset with one column called Review, containing text.Each review is labeled as
either positive or negative.

The tool returns:
- A summary message indicating how many reviews were classified as positive or negative.
- A transformed table including a new Sentiment column.
```

Final Steps: Connect and Run Your Agent

With all four tools complete, your agent is ready to reason, trigger tools, and return helpful responses based on user requests.

1. Register All Tools

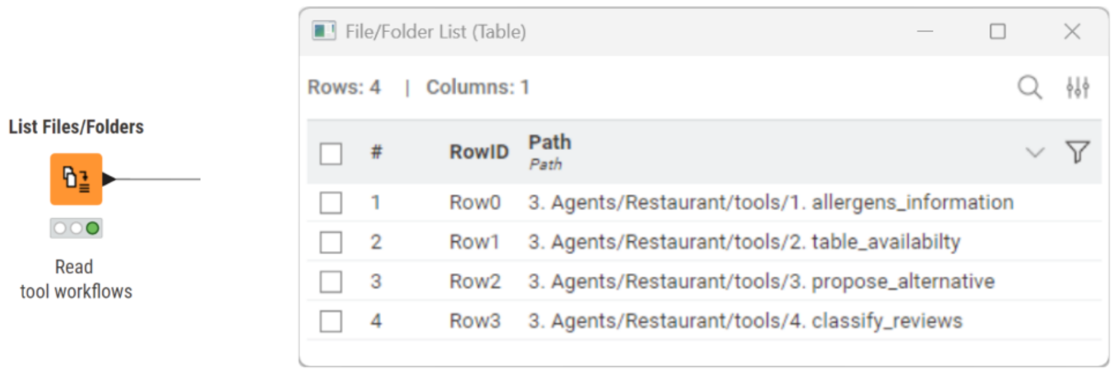
Place these tool workflows in a single folder named *tools*:

- tools/allergens_information
- tools/table_availability
- tools/alternative_booking
- tools/classify_reviews

2. Create Tool List Workflow

In a new workflow, create the tool list:

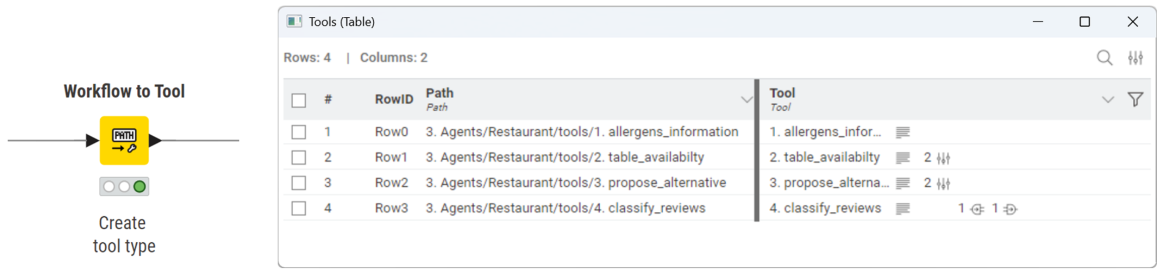
- List Files/Folders
  - Configure it to point to the tools folder
  - This retrieves all .*knwf* tool workflows



#	RowID	Path
1	Row0	3. Agents/Restaurant/tools/1. allergens_information
2	Row1	3. Agents/Restaurant/tools/2. table_availabilty
3	Row2	3. Agents/Restaurant/tools/3. propose_alternative
4	Row3	3. Agents/Restaurant/tools/4. classify_reviews

Figure 45. The List Files/Folders node reads all tool workflows from the selected "tools" directory.

- Workflow to Tool
  - This converts each workflow into a Tool object with associated metadata
  - Icons indicate whether the tool includes parameters, data ports, or is missing a description



Tools (Table)

Rows: 4 | Columns: 2

#	RowID	Path *Path*	Tool *Tool*
1	Row0	3. Agents/Restaurant/tools/1. allergens_information	1. allergens_infor... ≡
2	Row1	3. Agents/Restaurant/tools/2. table_availabilty	2. table_availabilty ≡ 2
3	Row2	3. Agents/Restaurant/tools/3. propose_alternative	3. propose_alterna... ≡ 2
4	Row3	3. Agents/Restaurant/tools/4. classify_reviews	4. classify_reviews ≡ 1 1

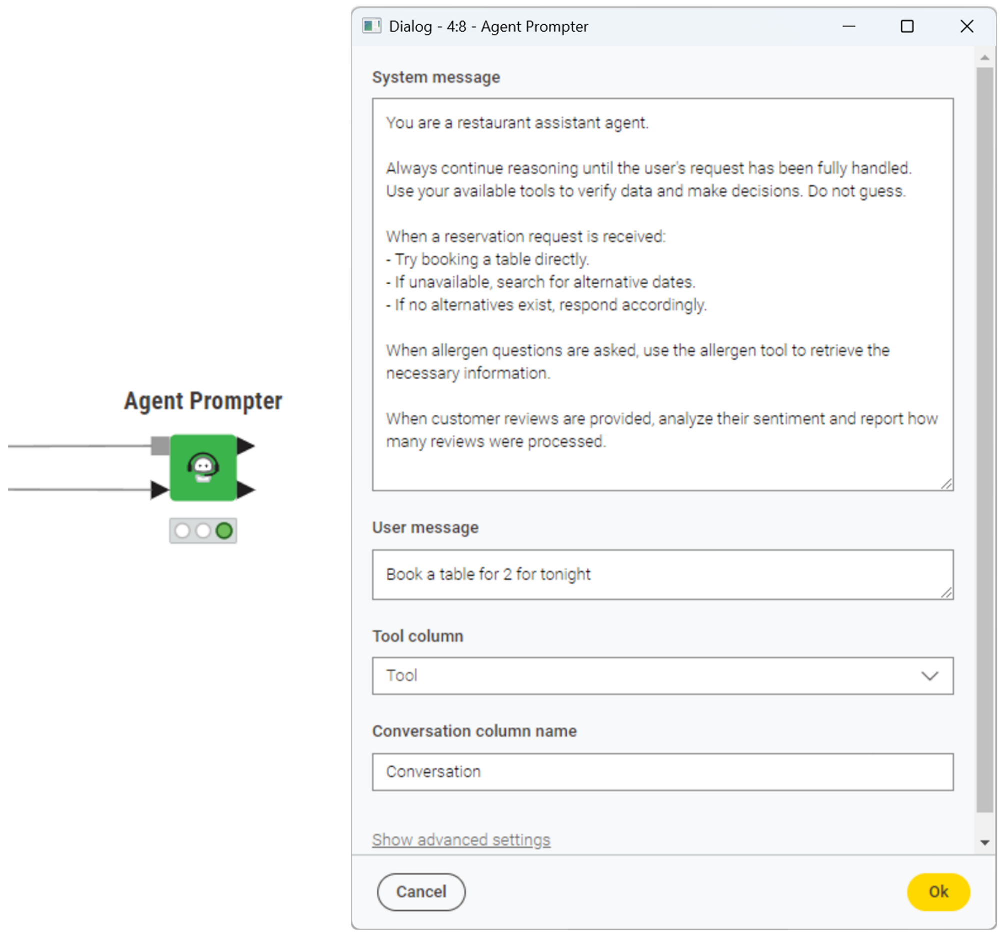
Figure 46. The output of the Workflow to Tool node shows icons that help verify tool descriptions, parameters, and data inputs/outputs.

3. Set up the Agentic Workflow

- Add the Agent Prompter node and set this as *System Message*:

```
You are a restaurant assistant agent.
Always continue reasoning until the user's request is fully handled.
Use your available tools to verify data and make decisions. Do not guess.
When a reservation request is received:
- Try booking a table directly.
- If unavailable, search for alternative dates.
- If no alternatives exist, respond accordingly.
When allergen questions are asked, use the allergen tool to retrieve the necessary
information.
When customer reviews are provided, analyze their sentiment and report how many
reviews were processed.”
```

- Optionally prefill the *User message* field (e.g., *Can you book a table for two people for June 26th?*)
- In the *Tool column* add the output of the Workflow to Tool node



System message

You are a restaurant assistant agent.

Always continue reasoning until the user's request has been fully handled.
Use your available tools to verify data and make decisions. Do not guess.

When a reservation request is received:
- Try booking a table directly.
- If unavailable, search for alternative dates.
- If no alternatives exist, respond accordingly.

When allergen questions are asked, use the allergen tool to retrieve the necessary information.

When customer reviews are provided, analyze their sentiment and report how many reviews were processed.

User message

Book a table for 2 for tonight

Tool column

Tool

Conversation column name

Conversation

Figure 47. The Agent Prompter configuration dialog

- Enable Data Ports (for Tool 4)

  To allow the agent to work with external data (such as customer reviews for sentiment analysis), you need to add data input and output ports to the Agent Prompter node:

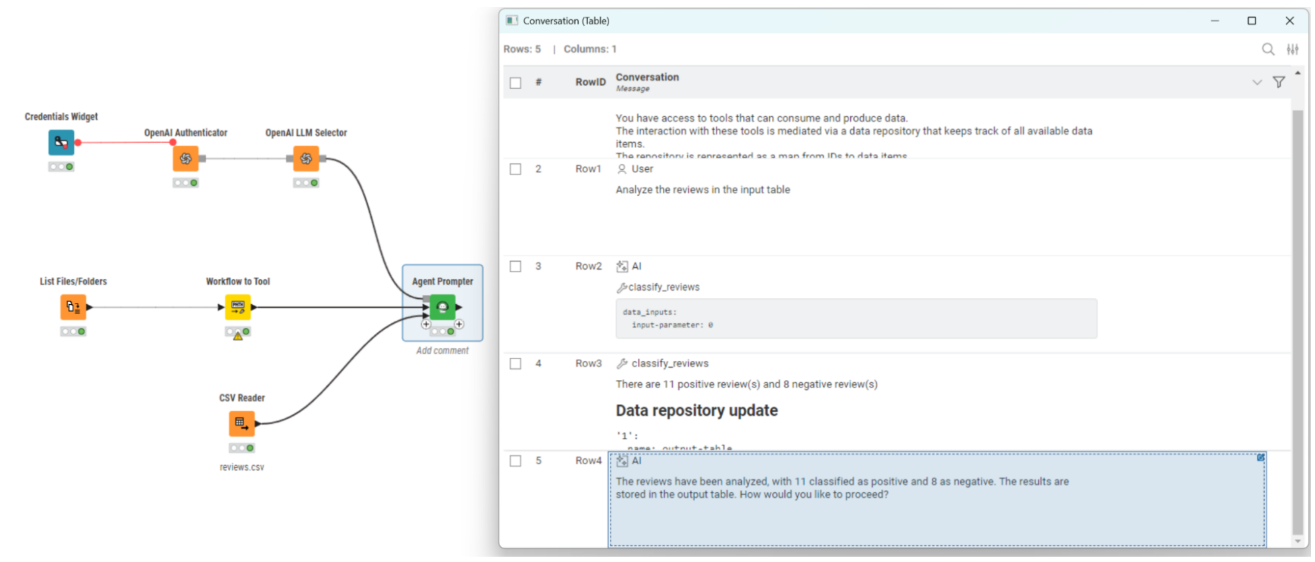
  1. Import your dataset using a CSV Reader node.

     This should contain a column named *Review*, with one review per row.

  2. Right-click the Agent Prompter node.

     Select *Add Input Port* and *Add Output Port* from the context menu. This enables the agent to receive and process data through the tools.

  3. Connect the CSV Reader to the input port of the Agent Prompter.



Conversation (Table)

Rows: 5 | Columns: 1

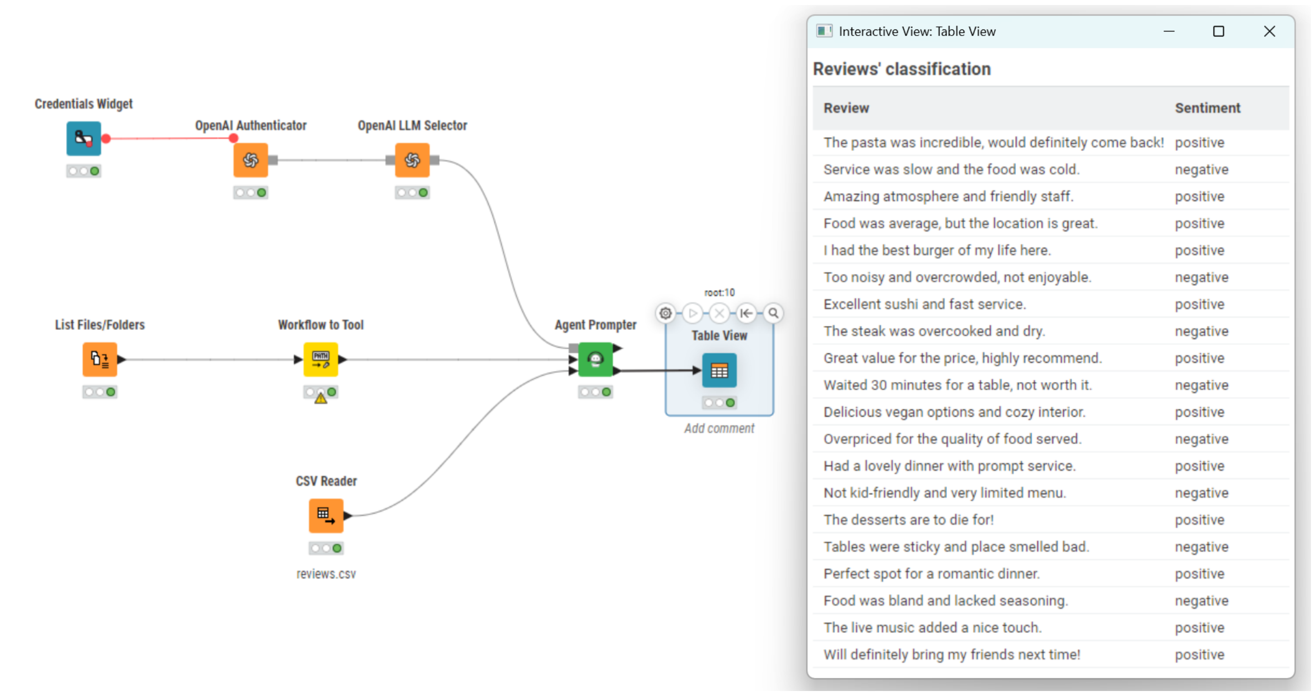
#	RowID	Conversation *Message*
		You have access to tools that can consume and produce data. The interaction with these tools is mediated via a data repository that keeps track of all available data items. The repository is represented as a map from IDs to data items.
2	Row1	User Analyze the reviews in the input table

3	Row2	AI
		classify_reviews
		`data_inputs:`
		`  input-parameter: 0`

4	Row3	classify_reviews
		There are 11 positive review(s) and 8 negative review(s)
		Data repository update
		`'1':`
		`  name: output-table`

5	Row4	AI The reviews have been analyzed, with 11 classified as positive and 8 as negative. The results are stored in the output table. How would you like to proceed?

Figure 48. Agent Prompter with an added output port: the agent can now return data along with its response.



Interactive View: Table View

Reviews' classification

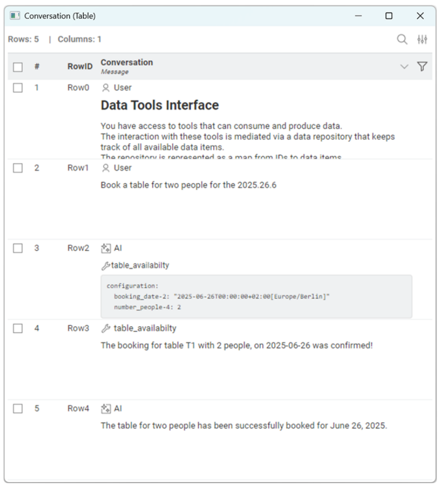
Review	Sentiment
The pasta was incredible, would definitely come back!	positive
Service was slow and the food was cold.	negative
Amazing atmosphere and friendly staff.	positive
Food was average, but the location is great.	positive
I had the best burger of my life here.	positive
Too noisy and overcrowded, not enjoyable.	negative
Excellent sushi and fast service.	positive
The steak was overcooked and dry.	negative
Great value for the price, highly recommend.	positive
Waited 30 minutes for a table, not worth it.	negative
Delicious vegan options and cozy interior.	positive
Overpriced for the quality of food served.	negative
Had a lovely dinner with prompt service.	positive
Not kid-friendly and very limited menu.	negative
The desserts are to die for!	positive
Tables were sticky and place smelled bad.	negative
Perfect spot for a romantic dinner.	positive
Food was bland and lacked seasoning.	negative
The live music added a nice touch.	positive
Will definitely bring my friends next time!	positive

Figure 49. The Agent Prompter has one input and one output port: only the communication layer is visible to the user, no data output is returned.

4. Run and Inspect

Run the workflow. This is an interactive process and may not work perfectly on the first try. To troubleshoot, use the *Debug mode* in the Agent Prompter view. This mode helps you see the agent's reasoning process step by step.

After completing this process, the Agent Prompter outputs a conversation between User, AI and Tools.



Conversation (Table)

Rows: 5 | Columns: 1

#	RowID	Conversation *Message*
1	Row0	User **Data Tools Interface** You have access to tools that can consume and produce data. The interaction with these tools is mediated via a data repository that keeps track of all available data items. The repository is represented as a map from IDs to data items.
2	Row1	User Book a table for two people for the 2025.26.6

3	Row2	AI
		table_availabilty
		`configuration:`
		`  booking_date-2: "2025-06-26T00:00:00+02:00[Europe/Berlin]"`
		`  number_people-4: 2`

4	Row3	table_availabilty The booking for table T1 with 2 people, on 2025-06-26 was confirmed!
5	Row4	AI The table for two people has been successfully booked for June 26, 2025.

Figure 50. The Output view of the Agent Prompter node message

If a tool fails, for example because a language model is missing credentials, the debug trace will clearly show where the failure happened. This makes it easier to identify and fix the problem.

5. Add Chat Interface

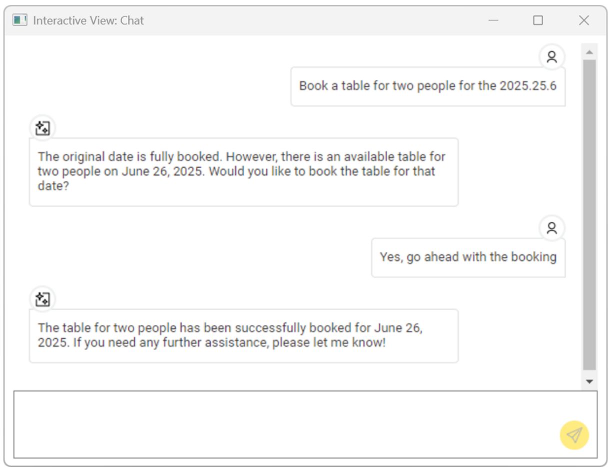
To make the assistant interactive for end users, use the Agent Chat View node.

To do this:

1. Add the Agent Chat View node to your workflow.
2. Connect the tool list output from the Workflow to Tool node.
3. If your agent uses external data, connect the appropriate input tables as well.

Once configured, you can wrap the workflow into a component and deploy it via KNIME Business Hub.

This makes your assistant accessible as a service, ready to receive user queries and return tool-based responses.



Book a table for two people for the 2025.25.6

The original date is fully booked. However, there is an available table for two people on June 26, 2025. Would you like to book the table for that date?

Yes, go ahead with the booking

The table for two people has been successfully booked for June 26, 2025. If you need any further assistance, please let me know!

Figure 51. The Agent Chat View provides a live conversation interface.

AI governance

GPT4All (Local Models)

KNIME supports local execution of open-source models through **GPT4All**, allowing you to run large language models (LLMs) and embedding models directly on your machine. This enables full offline operation, removes dependency on external APIs, protects privacy-sensitive data, and eliminates usage-based costs associated with paid providers.

System Prerequisite: GPT4All nodes may not be compatible with Red Hat 8. Please verify your operating system before using these nodes.

Key characteristics

- **No external APIs**

  GPT4All runs fully on your local hardware. No internet connection or external services (such as OpenAI or Hugging Face) are required.

- **No authentication needed**

  Since models are executed locally, no Authenticator nodes or API keys are necessary.

- **Open-source models**

  You can choose from a variety of community-maintained models.

Model setup

Before using GPT4All models in KNIME, you need to obtain the model files:

1. Download models from Hugging Face Hub that are available in .gguf format.

   (e.g. *NousResearch/Nous-Hermes-llama2-GGUF*).

2. Save the model file locally on your machine.
3. In the Connector node configuration, specify the full file path to the downloaded .gguf model file.

Hardware configuration

You can choose which processing unit should be used to run the model:

- *cpu* uses the system's central processor (default setting).
- *gpu* uses the best available GPU, regardless of vendor.
- *amd*, *nvidia*, or *intel* choose a specific vendor.
- *Specific GPU name* runs on a particular GPU if multiple are available and properly configured.

Selecting a GPU can significantly improve inference speed for larger models.

GPT4All Connector nodes

The KNIME AI Extension includes dedicated connector nodes for GPT4All models:

- Local GPT4All LLM Connector
- GPT4All Embeddings Connector

Example workflow

For enterprise users interested in centralized model governance and access control, KNIME Business Hub also supports GenAI Gateway. This feature allows admins to configure and manage chat and embedding models centrally across an organization. For more details, refer to the Business Hub Admin Guide.

KNIME AG
Talacker 50
8001 Zurich, Switzerland
www.knime.com
info@knime.com