Open for Innovation
KNIME

KNIME AI Extension Guide

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

Table of Contents

Prompting a Model . . . 1
- Install the KNIME AI Extension . . . 1
- Authenticate . . . 2
- Select . . . 3
- Prompt . . . 4
- Provider Reference Table . . . 19

Vector Stores and Retrieval-Augmented Generation (RAG) . . . 22
- Why RAG is useful . . . 22
- How RAG works . . . 22
- Choose the right Vector Store format . . . 23
- Create a Vector Store . . . 23
- Read a Vector Store . . . 23
- Save and reload as Models (optional) . . . 23
- Example: Product FAQ Assistant with RAG . . . 24

Agents . . . 31
- The two layers of KNIME agentic workflows . . . 31
- How a KNIME Agent communicates using Messages . . . 32
- Work with Messages . . . 32
- Add functionality with Tools . . . 33
- Run and test the Agent . . . 41
- Agent Chat View node . . . 42
- Checklist: what you need to build an agent . . . 44
- Example: Build a Restaurant Assistant Agent . . . 45

AI governance . . . 65
- GPT4All (Local Models) . . . 65

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.

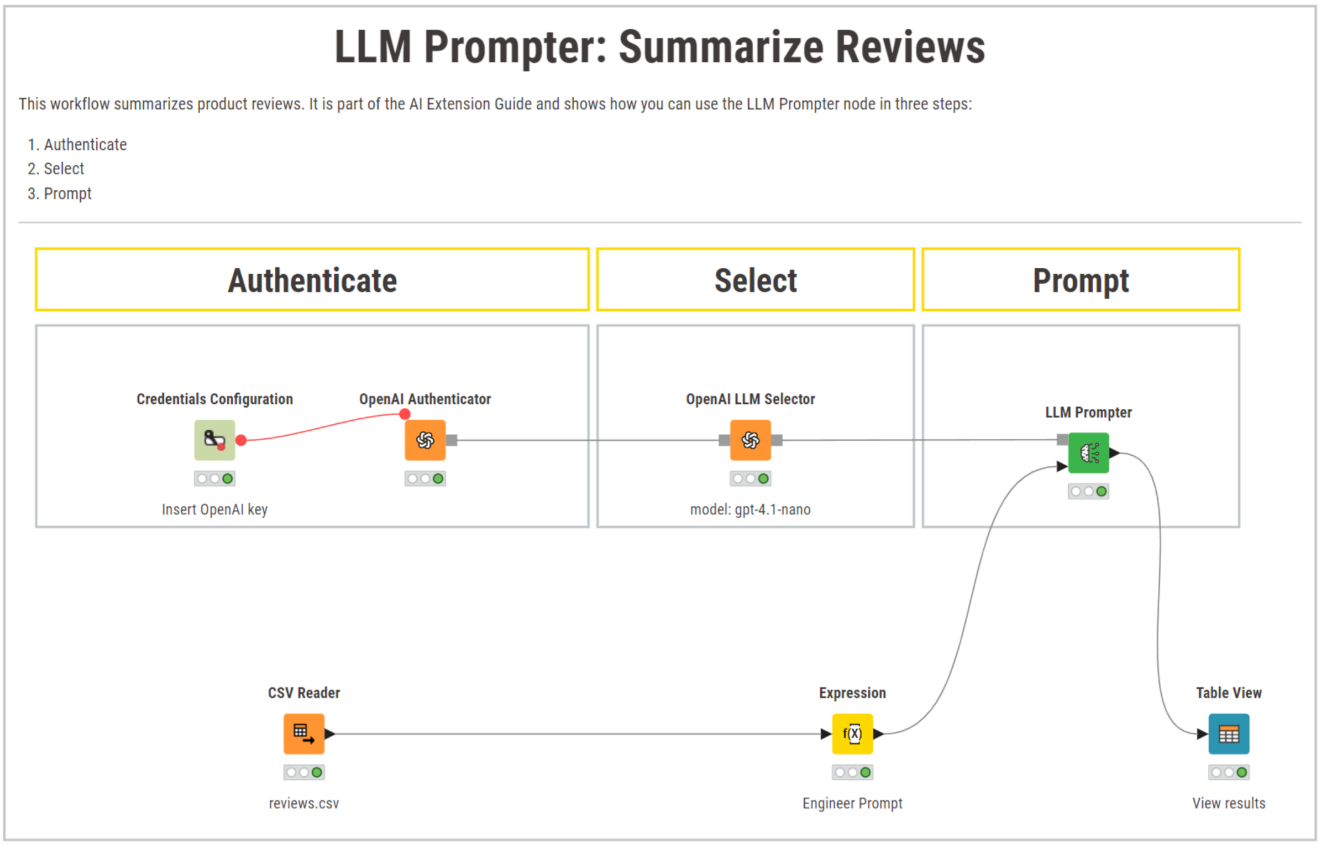


Figure 1. Example workflow using OpenAI's GPT-4.1 in KNIME to summarize product reviews via authentication, model selection, and prompting.

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.

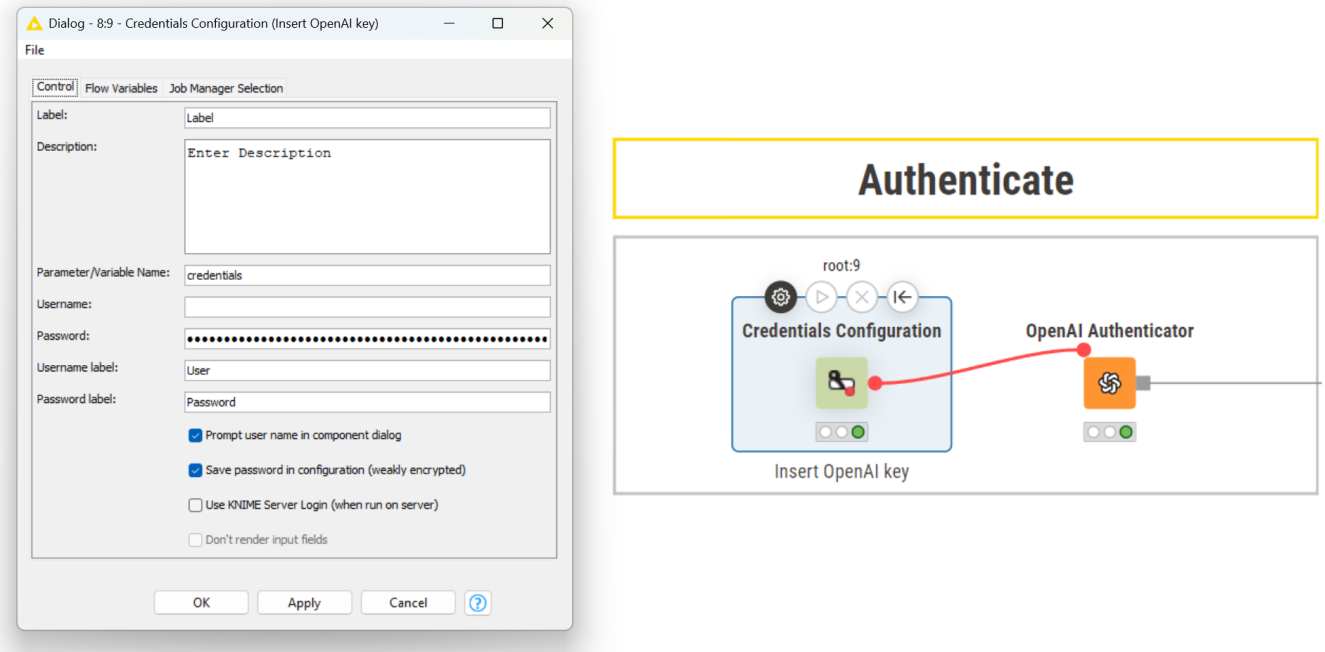


Figure 2. The Credentials Configuration node stores the OpenAI API key in the Password field and creates a credentials flow variable for use in downstream authentication nodes.

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.

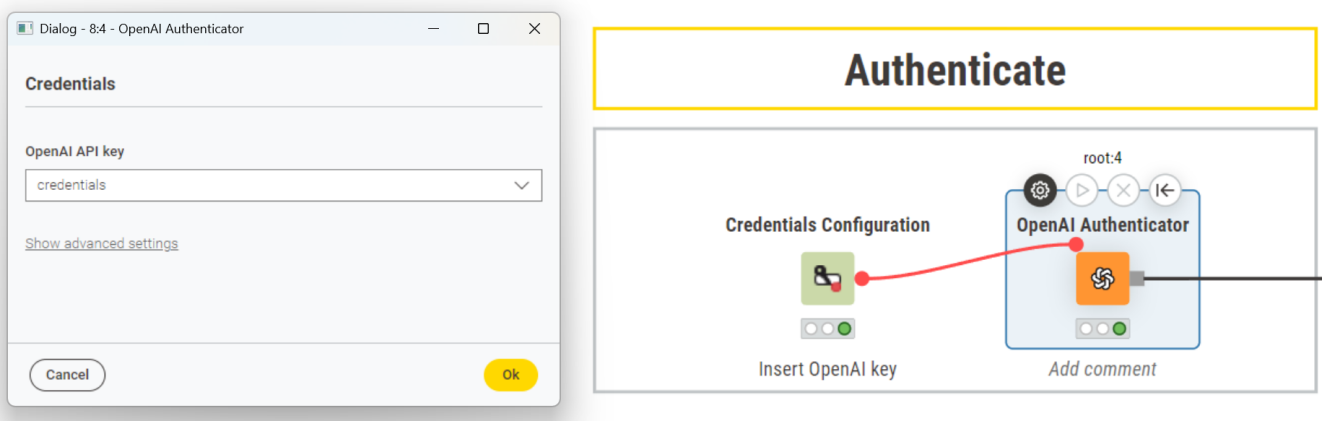


Figure 3. In this example, the OpenAI Authenticator node is configured to retrieve credentials from the flow variable generated by the Credentials Configuration node.

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 Hugging Face, Business 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*.

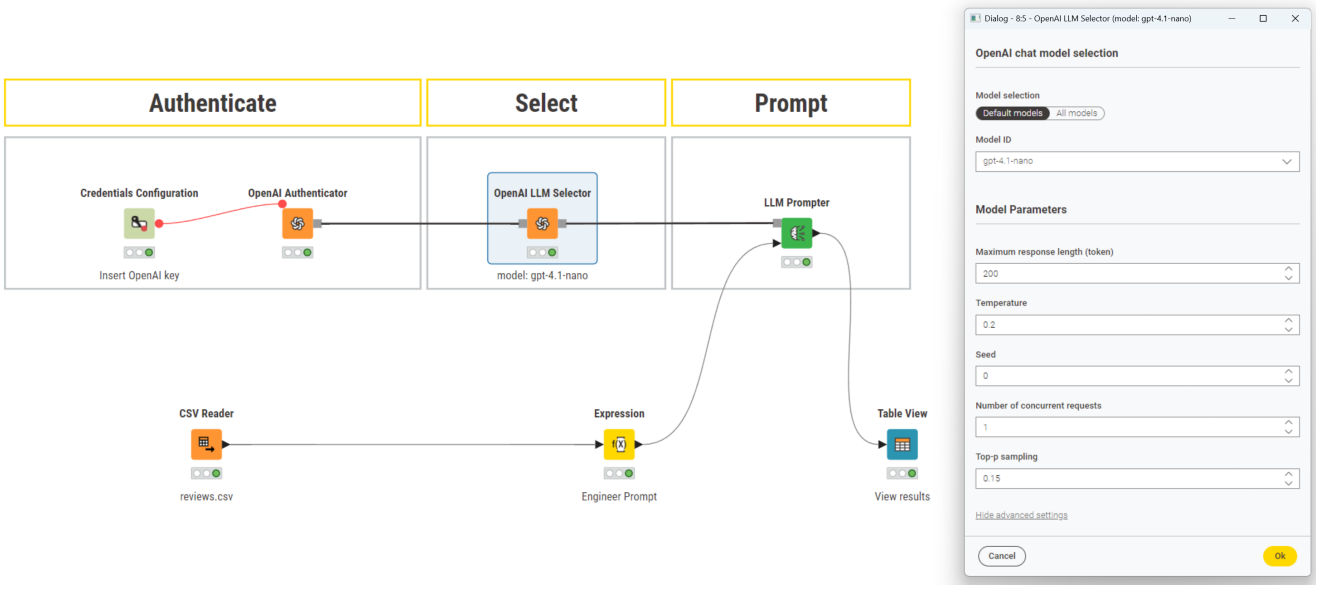


Figure 4. The Model Selector Connector node is configured to use OpenAI's GPT-4.1 model to summarize product reviews.

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.

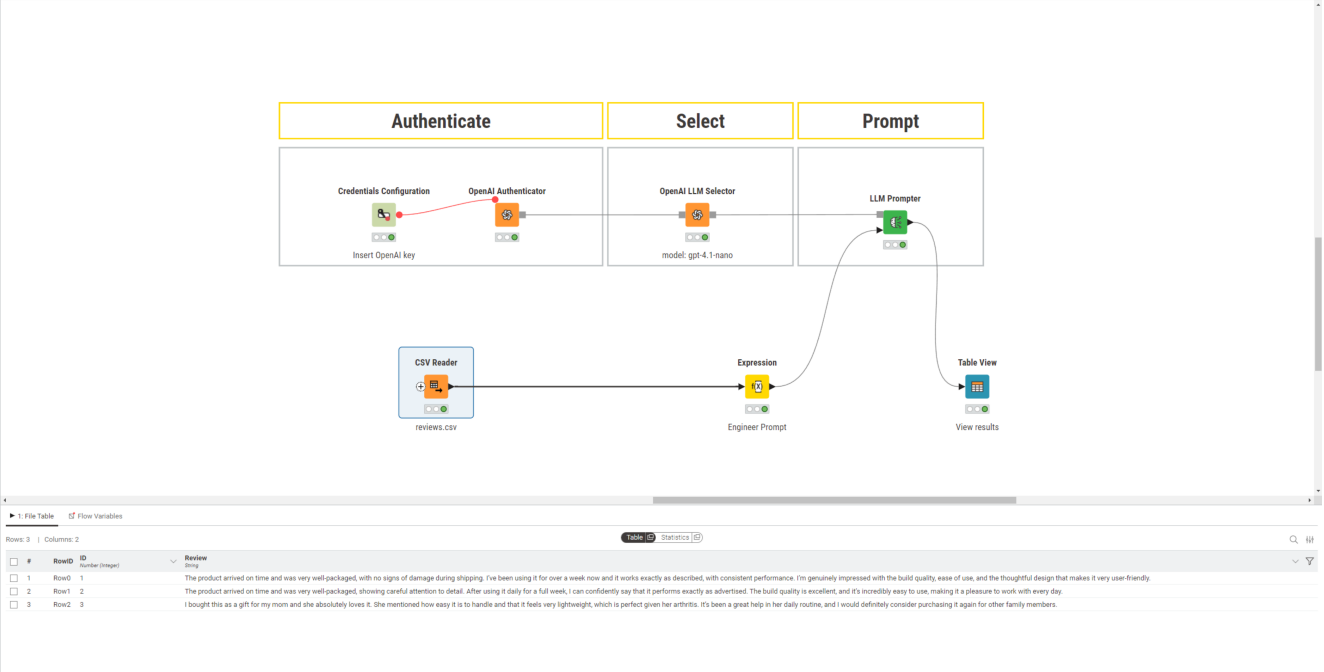


Figure 5. The CSV Reader node loads a table of customer reviews, each stored as a string, to be summarized later.

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.

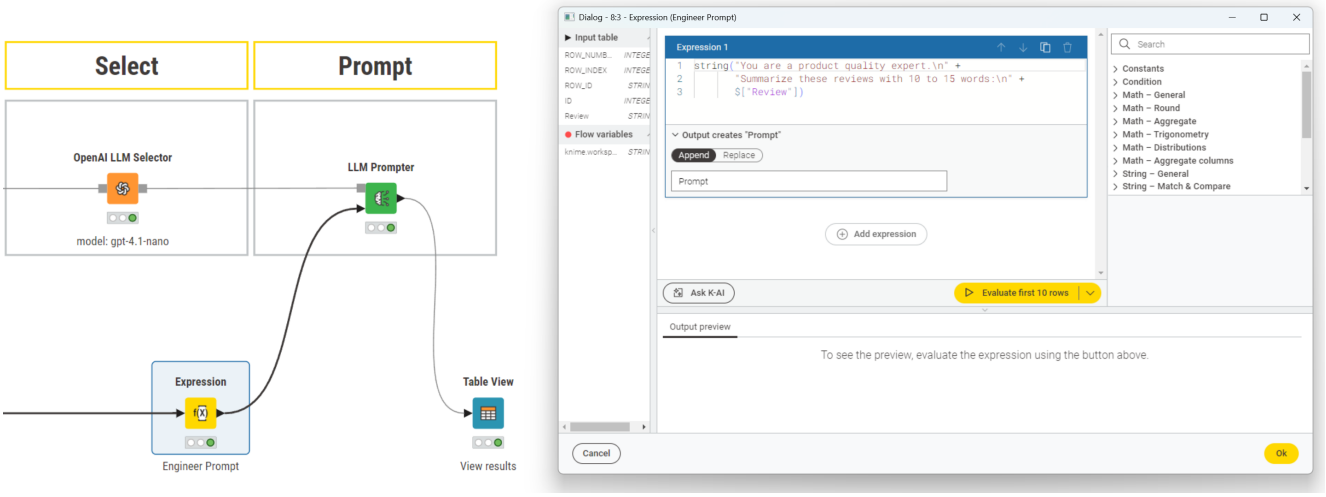


Figure 6. The Expression node creates a new column called prompt that contains the instructions for the LLM to follow.

3. Authenticate to OpenAI

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

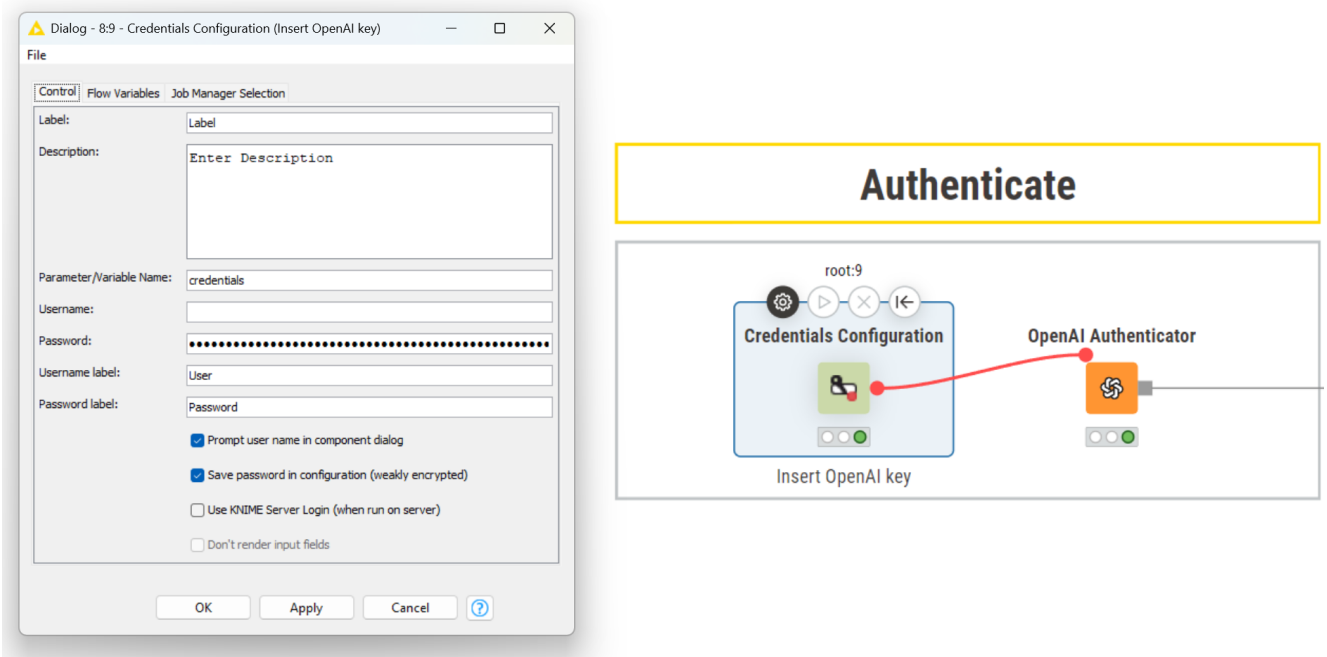


Figure 7. The Credentials Configuration securely stores your API key. The OpenAI Authenticator node reads this key to authorize requests to the OpenAI API.

4. Selecting a model

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

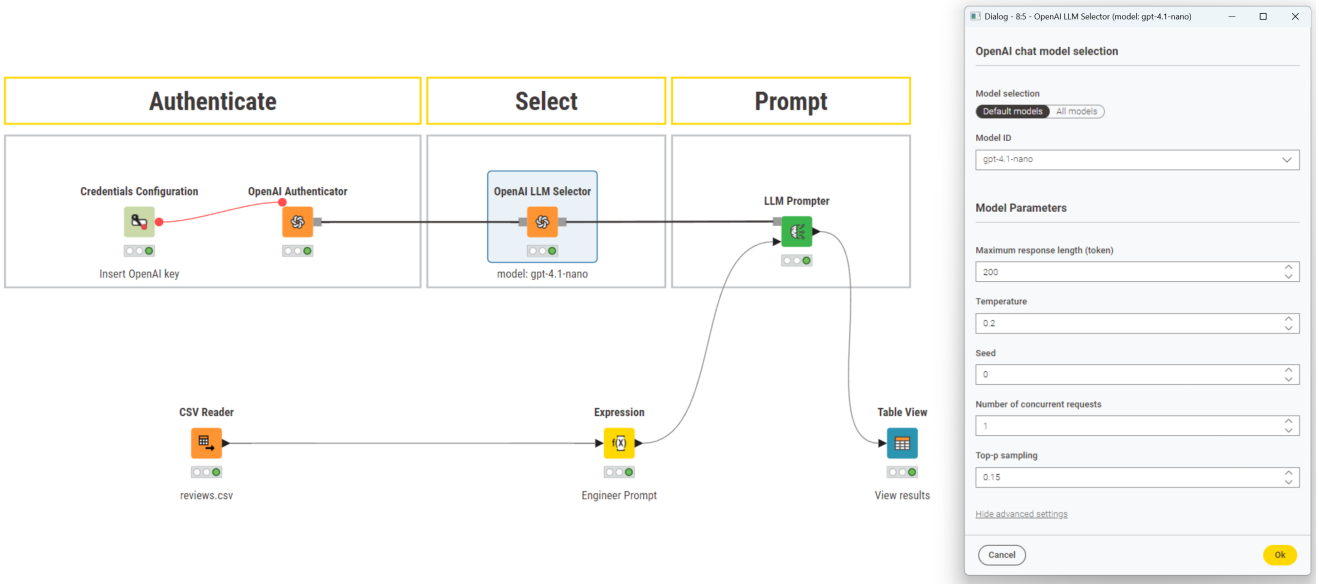


Figure 8. The OpenAI LLM Selector node connects to the API and selects GPT4 as model for prompting.

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.

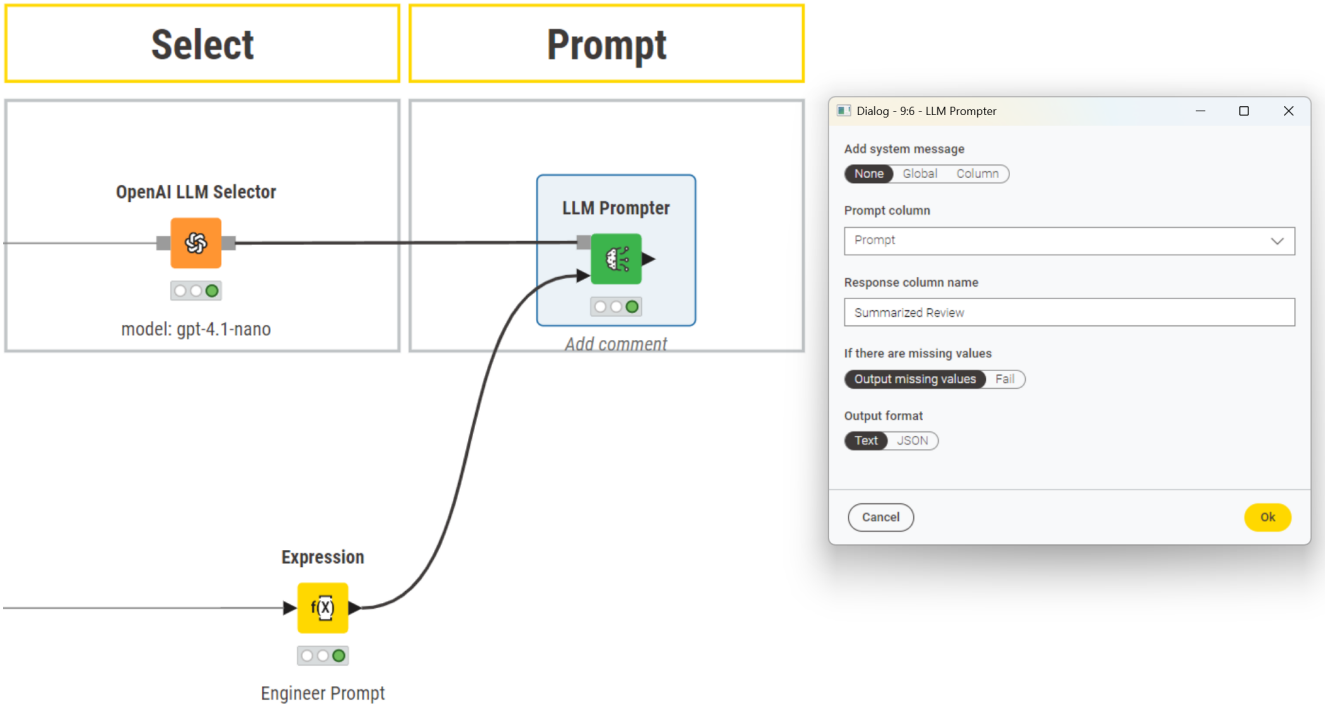


Figure 9. The LLM Prompter node uses the specified prompt column and generates a response from the model, which it stores in a new column called Response.

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.

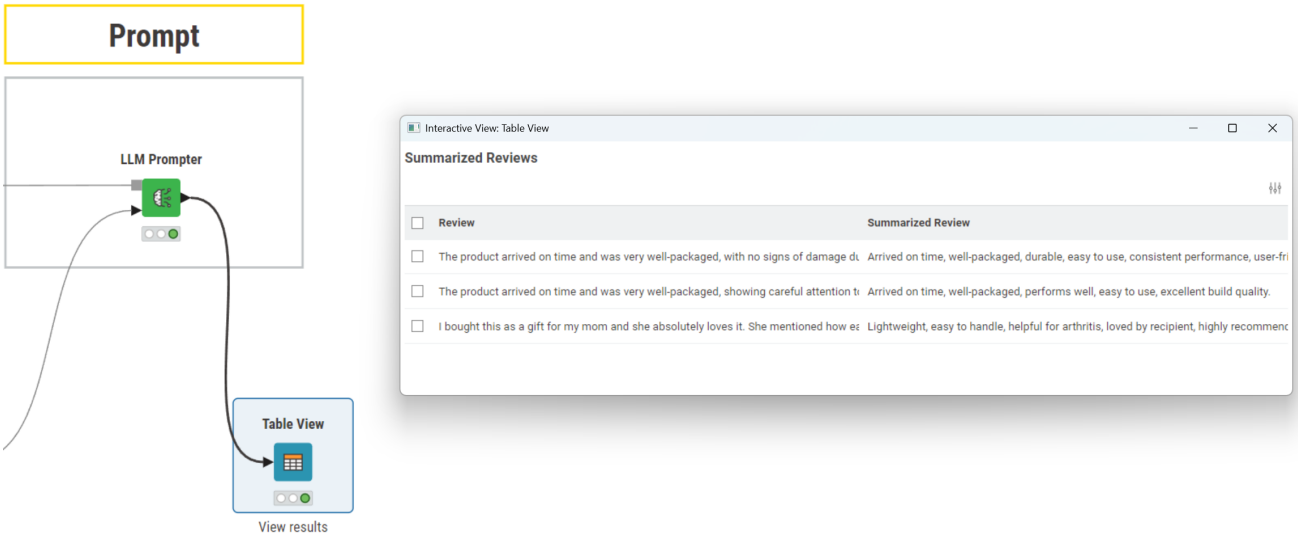


Figure 10. At the end of the workflow, the LLM Prompter node outputs a new column called Response. Each row contains a summary generated by the model based on the corresponding customer review.

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).

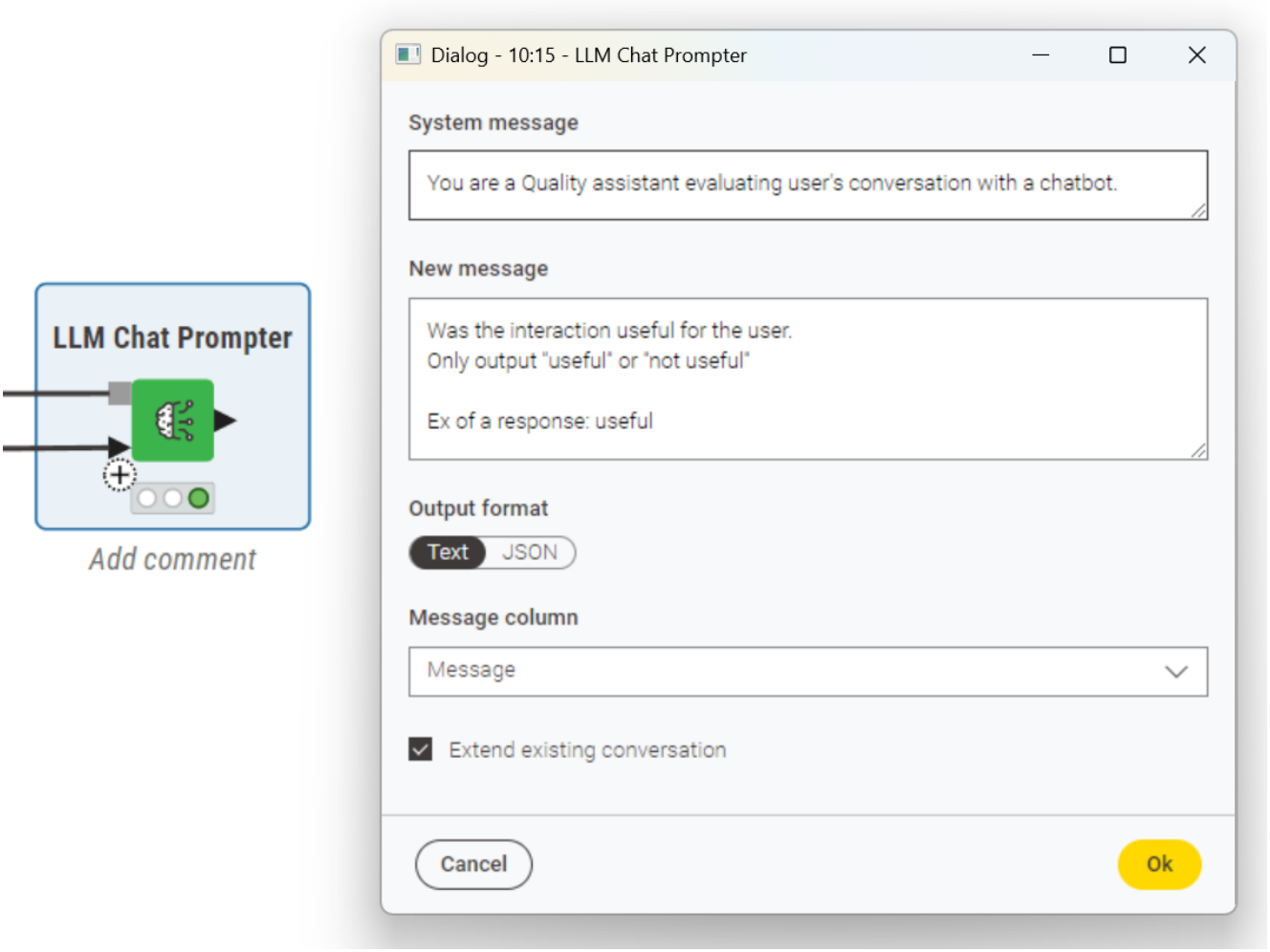


Figure 11. Configuration dialog of the LLM Chat Prompter node.

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.

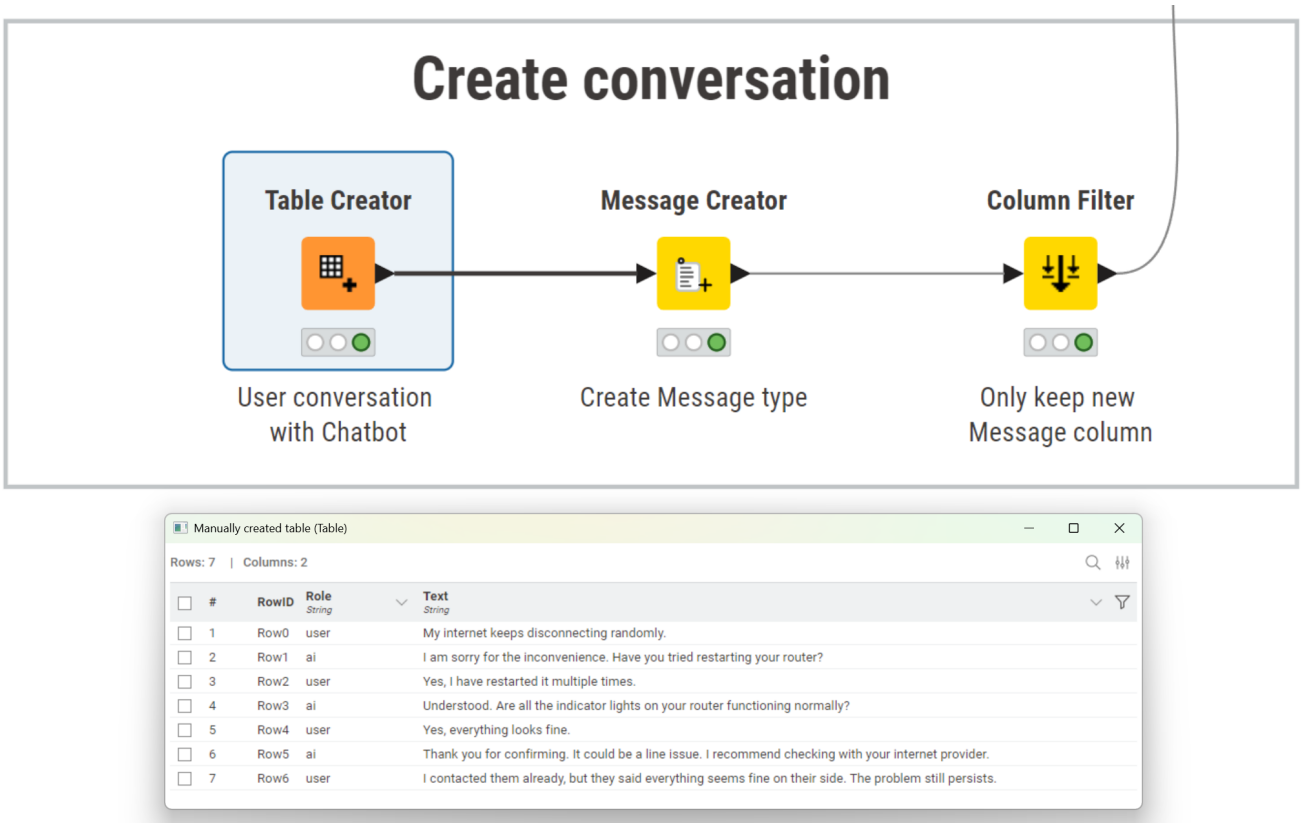
Example: Customer Care Quality Checker

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 message. 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*.

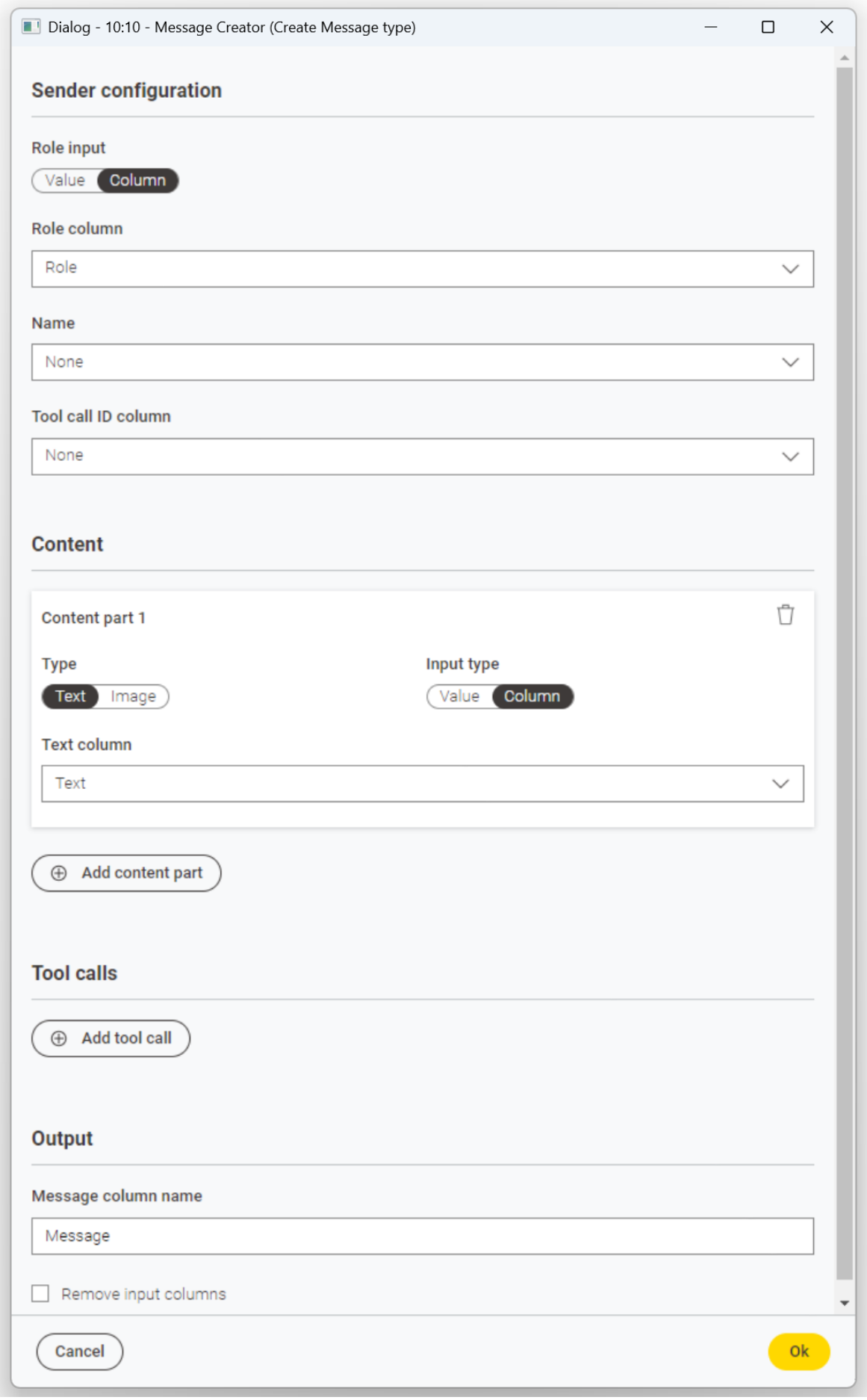


Figure 13. Configuration dialog of the Message Creator node.

After configurating 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.

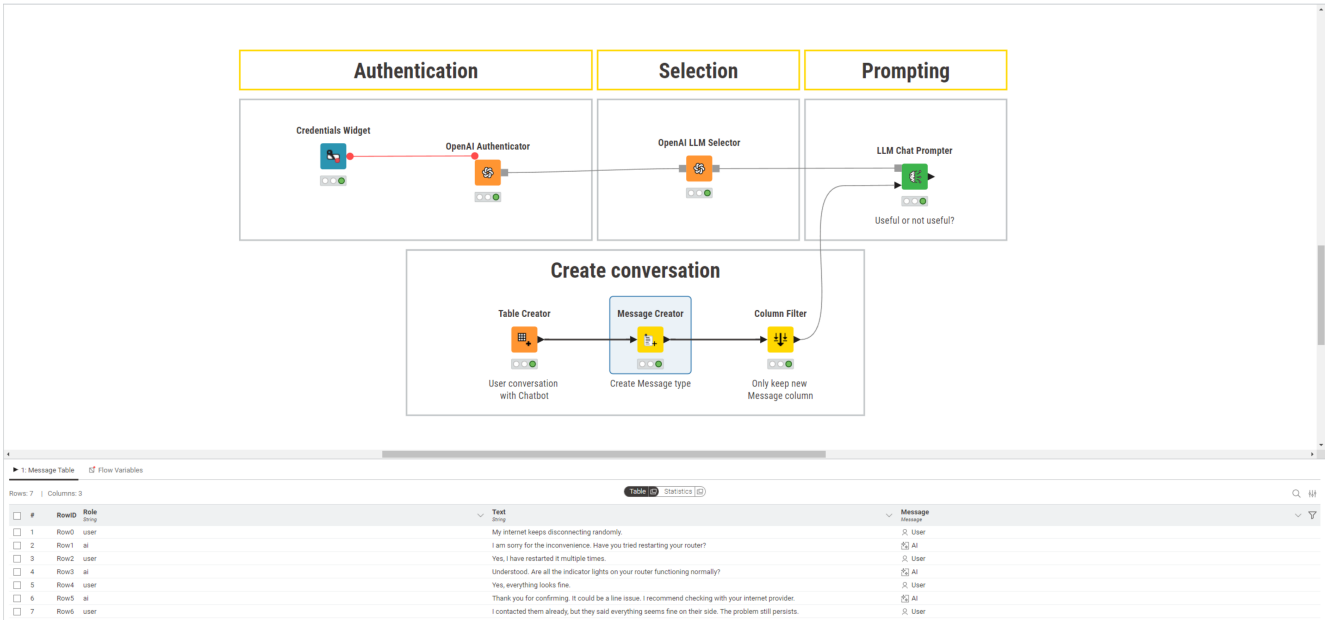


Figure 14. The output table containing KNIME Message column.

3. Filter Message column

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

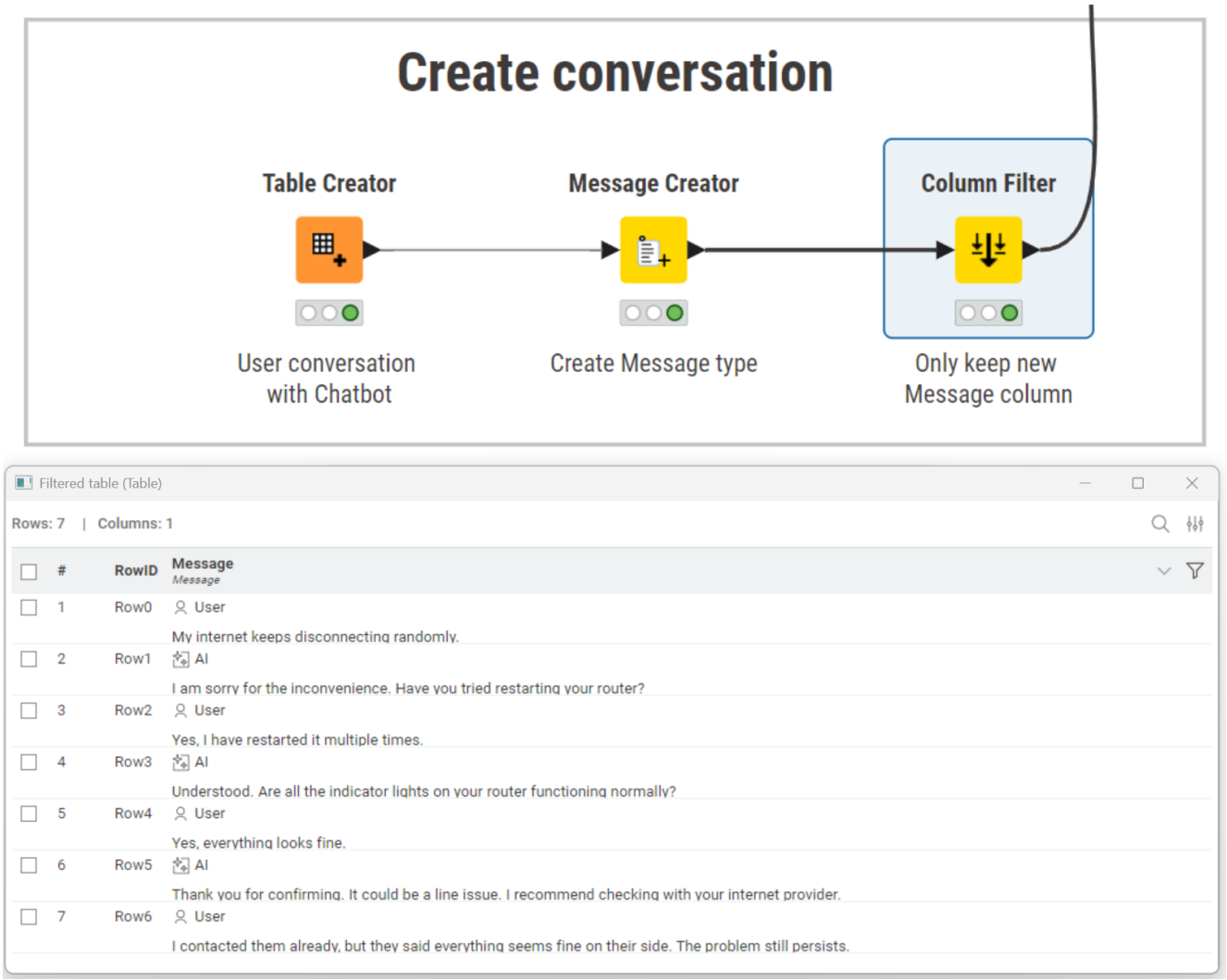


Figure 15. Message column passed into the LLM Chat Prompter node.

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'.
```

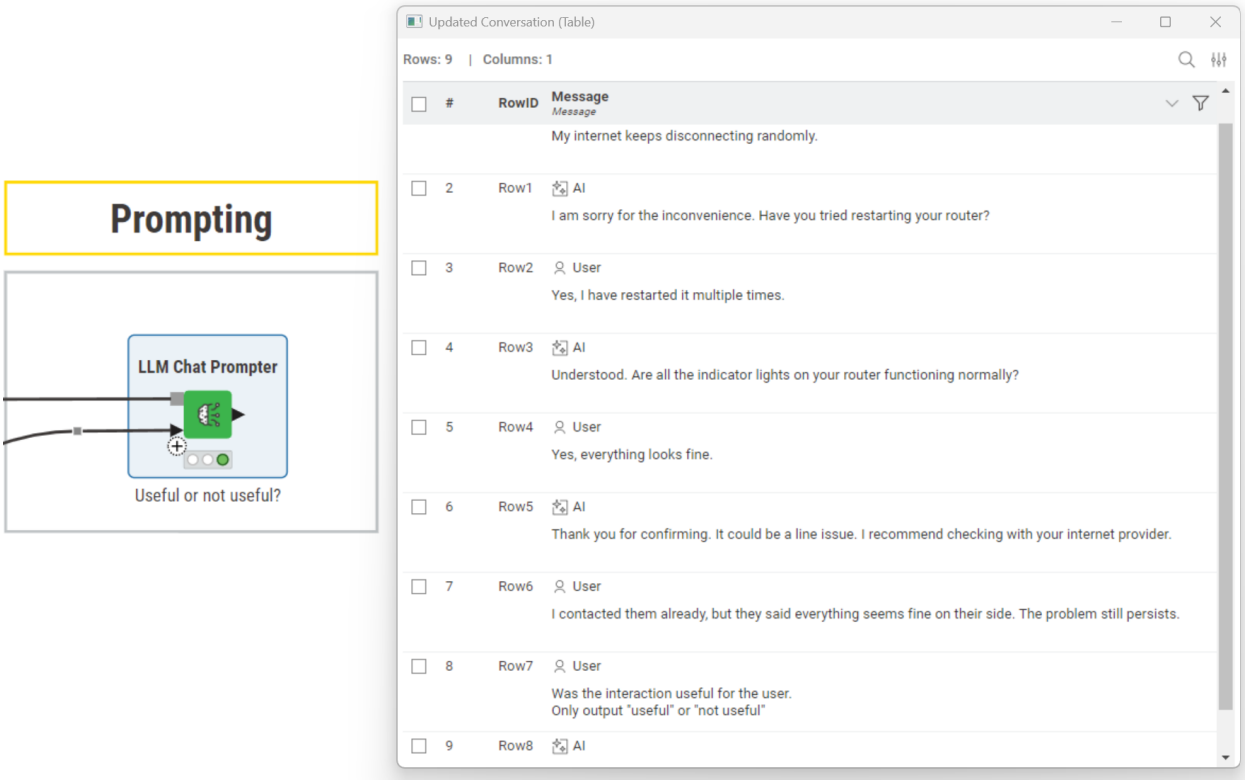


Figure 16. Output of the LLM Chat Prompter node with updated conversation.

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 nodes (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.
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.

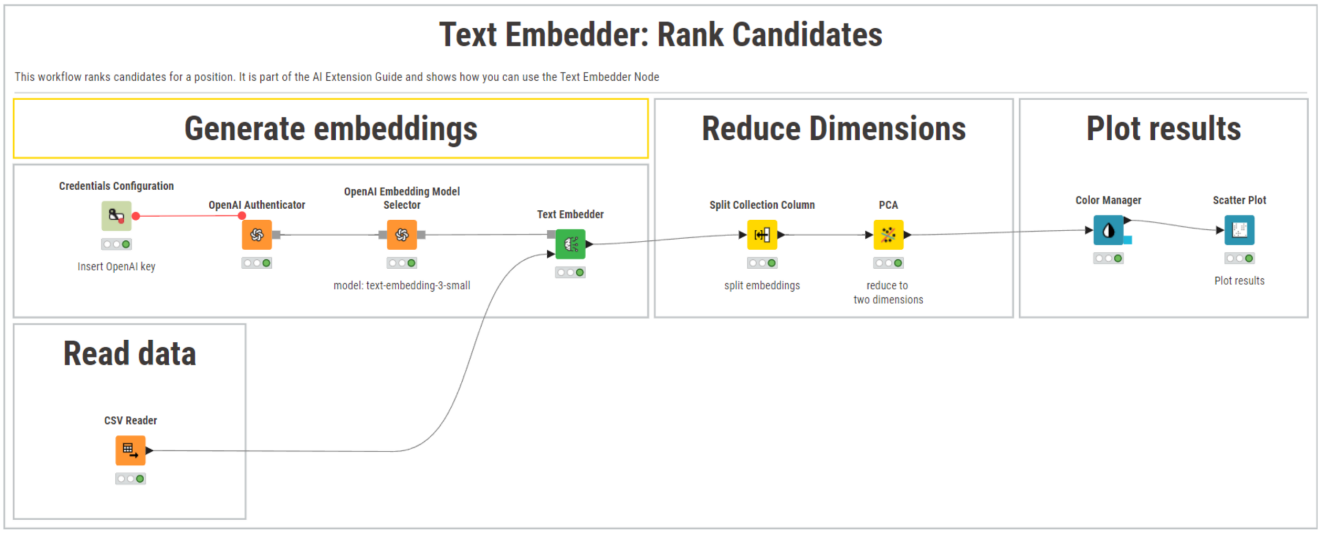


Figure 17. The KNIME workflow that compares candidate profiles using embeddings and plots the results

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.

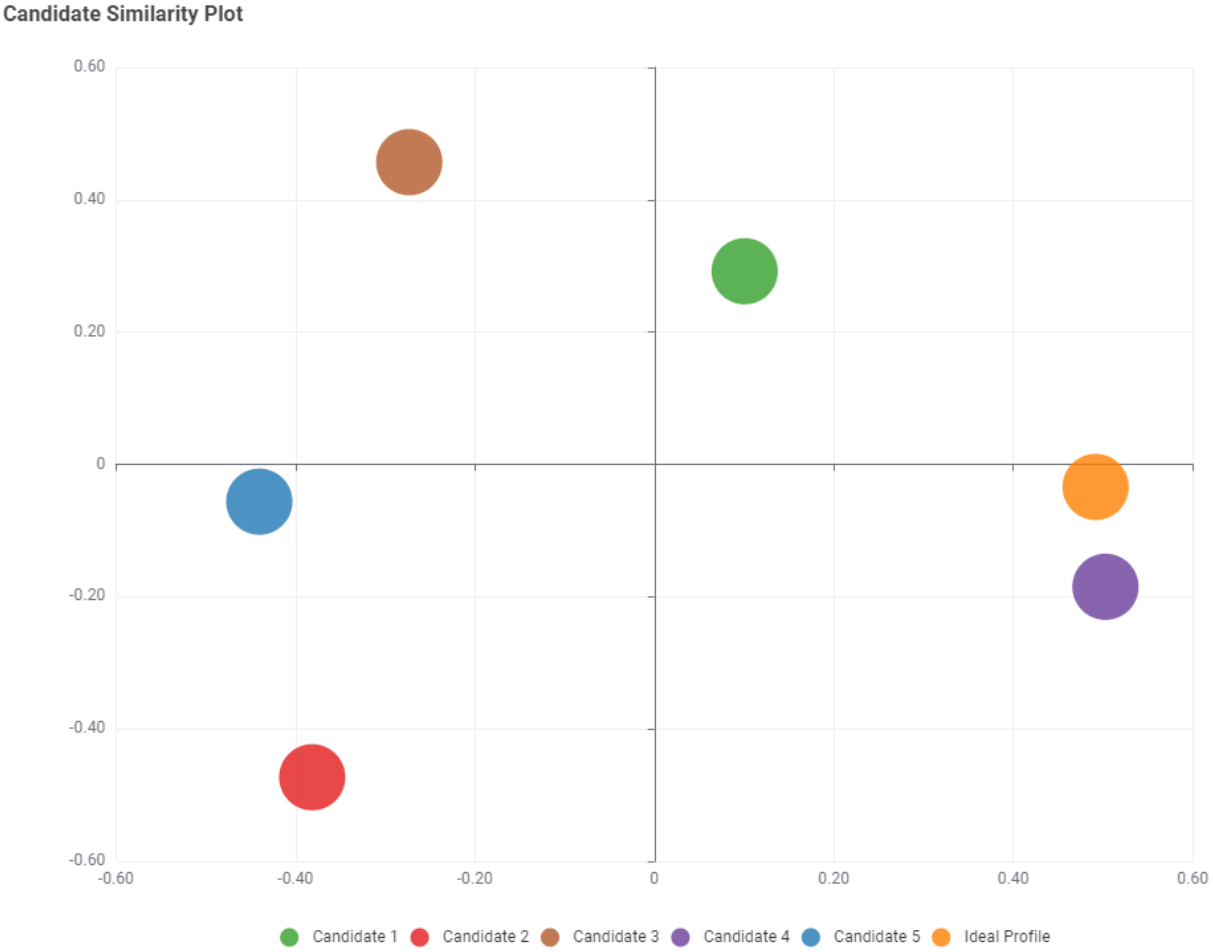


Figure 18. The resulting 2D plot of candidate similarity

Provider Reference Table

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

Authenticator nodes (by provider)	Selector	Required Credentials	Link to Provider	Example workflow
Google Authenticator	• Gemini LLM Selector • Gemini Embedding Model Connector	• Google AI Studio API key	Google	workflow
Anthropic Authenticator	Anthropic Chat Model Selector node	• Anthropic AI key	Anthropic	workflow
IBM watsonx.ai™ Authenticator	• IBM watsonx.ai Chat Model 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 Authenticator	Vertex AI connector	• Google Cloud's Project ID • Google Cloud's Location	Vertex ai	workflow

Authenticator nodes (by provider)	Selector	Required Credentials	Link to Provider	Example workflow
Hugging Face Connector	• HF Hub Chat Model Selector • HF Hub Embeddings Connector • HF TGI Chat Model Selector • HF TEI Embeddings Connector	• Hugging Face API key	Hugging Face	workflow
KNIME Hub Authenticator	• KNIME Hub LLM Selector • KNIME Hub Embeddings Selector	KNIME Business Hub	KNIME Hub	workflow
Databricks Workspace Connector	• Databricks LLM Selector • Databricks Embeddings Model Selector	• Databricks workspace URL • Personal access token	Databricks	workflow

Vector Stores and Retrieval-Augmented Generation (RAG)

This section builds on the nodes described in the Prompting a Model part of the guide. For a list of supported providers and nodes, see the Provider Reference Table.

Why RAG is useful

Language models have limitations: they can't access private data, may return outdated responses, and are not easily retrained. Retrieval-Augmented Generation (RAG) addresses these gaps by injecting external knowledge in the prompt. With KNIME, you can build custom RAG pipelines using:

- Local Vector Stores (FAISS or Chroma)
- Embedding services (OpenAI, Hugging Face)
- KNIME nodes for retrieval and generation

How RAG works

RAG combines retrieval and generation in three steps:

1. Embed documents into vectors

Text documents are transformed into embeddings (high-dimensional vectors) and stored in a vector database.

2. Retrieve context at query time

When a user submits a question, the system retrieves relevant documents by comparing vector similarity.

3. Generate grounded responses

The language model receives both the user query and retrieved content. It uses this context to generate informed, grounded answers. Vector Stores A Vector Store is a searchable database of embeddings. It retrieves semantically similar documents using vector math.

Choose the right Vector Store format

KNIME offers two Vector Store backends, each suited for different retrieval scenarios. Select the one that aligns with your workflow needs:

- FAISS
  - Stores vectors in a flat file format.
  - Best suited for fast local retrieval and simple storage scenarios.
  - Ideal when metadata or advanced filtering is not required.
- Chroma
  - 0Stores vectors in a JSON + SQLite format.
  - Supports document collections and metadata.
  - Recommended when you need to group documents or filter search results based on metadata.

Create a Vector Store

Use these nodes to generate a Vector Store from embeddings:

- FAISS Vector Store Creator Builds a FAISS-based index for high-performance local retrieval.
- Chroma Vector Store Creator Stores embeddings along with metadata using the Chroma backend.

Read a Vector Store

To reuse an existing store in your workflow:

- **FAISS Vector Store Reader** loads a saved FAISS index from disk.
- Chroma Vector Store Reader loads a saved Chroma store, including metadata if present.

Save and reload as Models (optional)

To avoid rebuilding your Vector Store every time a workflow runs, you can save it and reload it later. This is especially helpful for large or frequently reused indexes.

KNIME provides two nodes for this purpose, although they are not part of the AI Extension:

- Model Writer saves Vector Store as a .model file
- Model Reader loads Vector Store from .model file into a new workflow.

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.

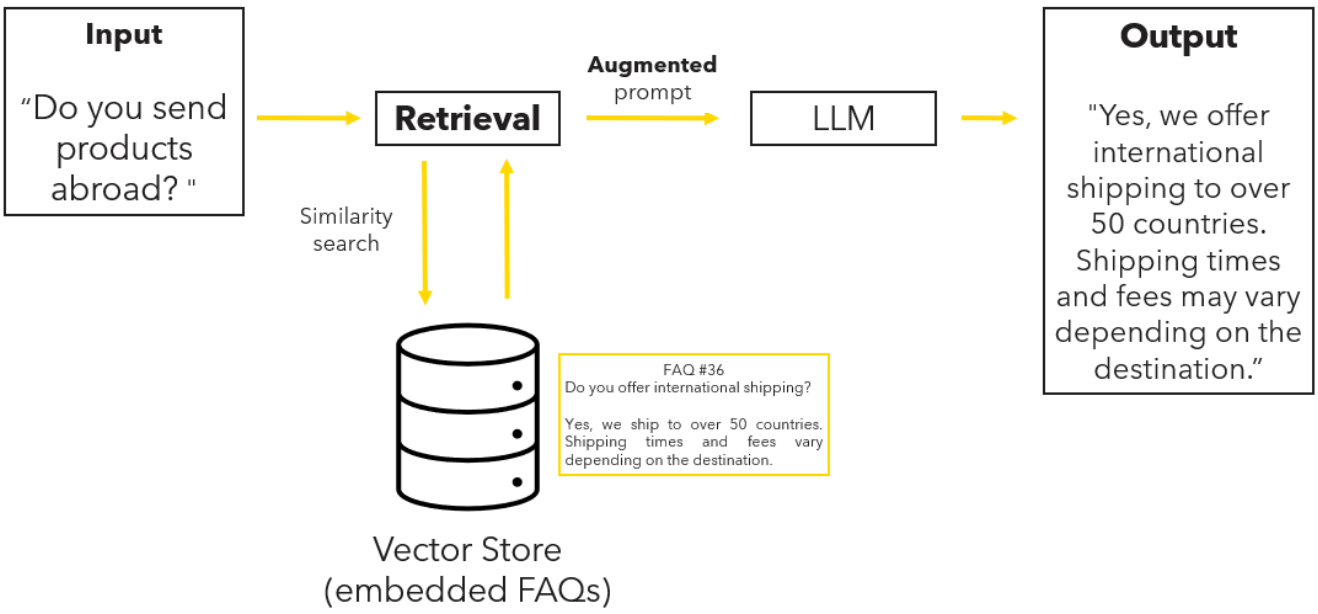


Figure 19. An overview of a RAG pipeline

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.

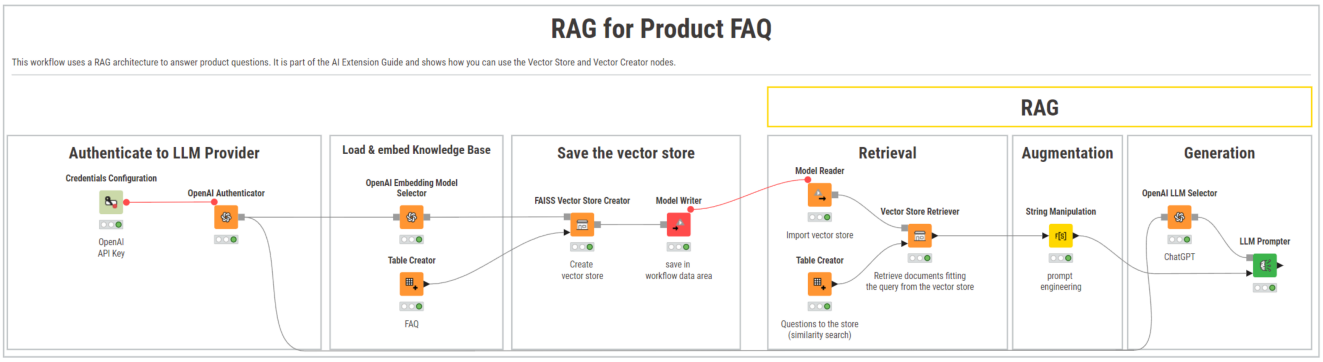


Figure 20. The Workflow that uses a RAG architecture to answer FAQs

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

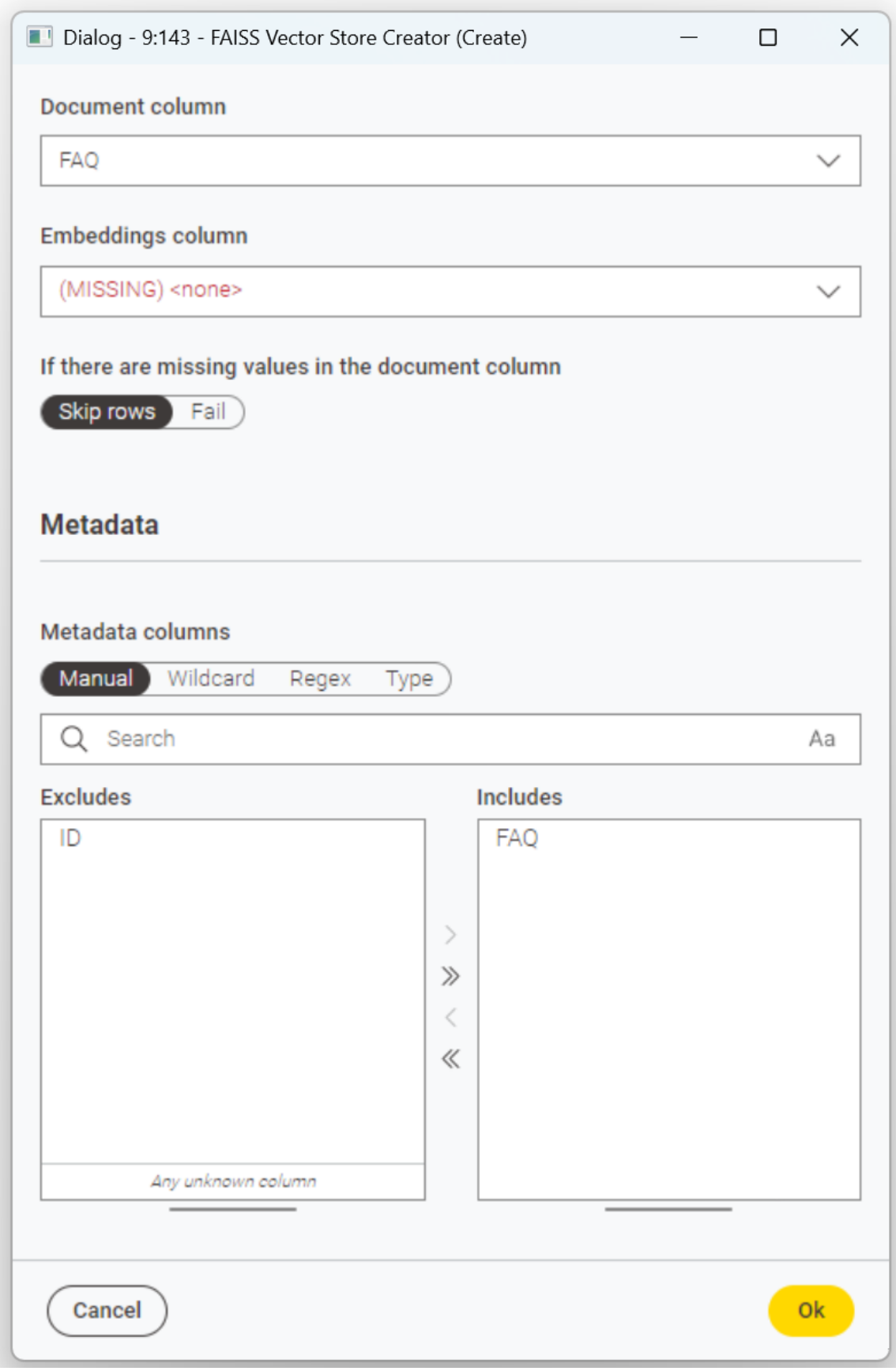


Figure 21. Configuration dialog of the Vector Store Creator. In this example, the node generates embeddings internally because no precomputed embeddings are provided. The AI Extension also includes a dedicated node for embedding generation: the Text Embedder (see earlier section).

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.

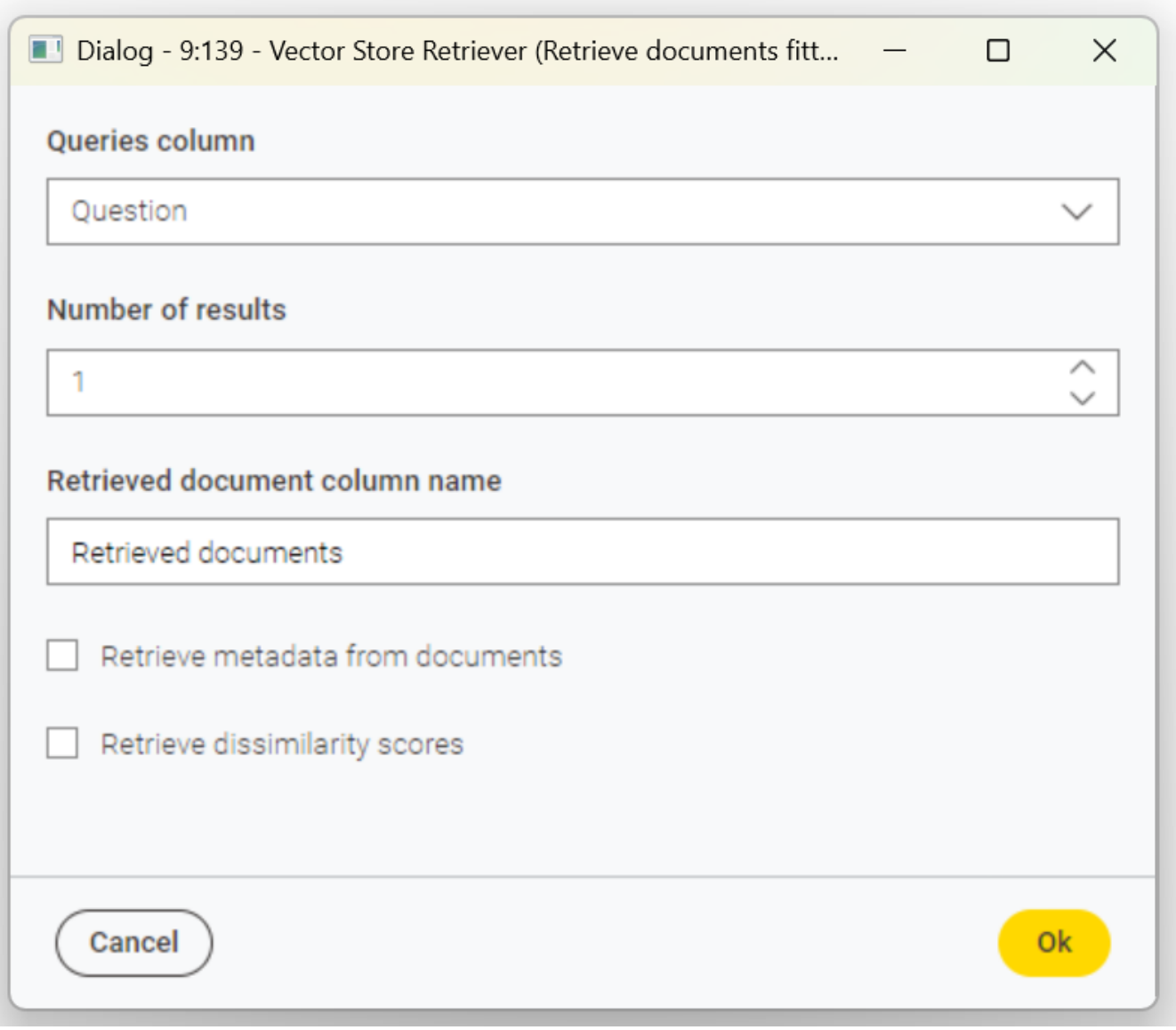


Figure 22. Configuration dialog of the Vector Store Retriever node.

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.

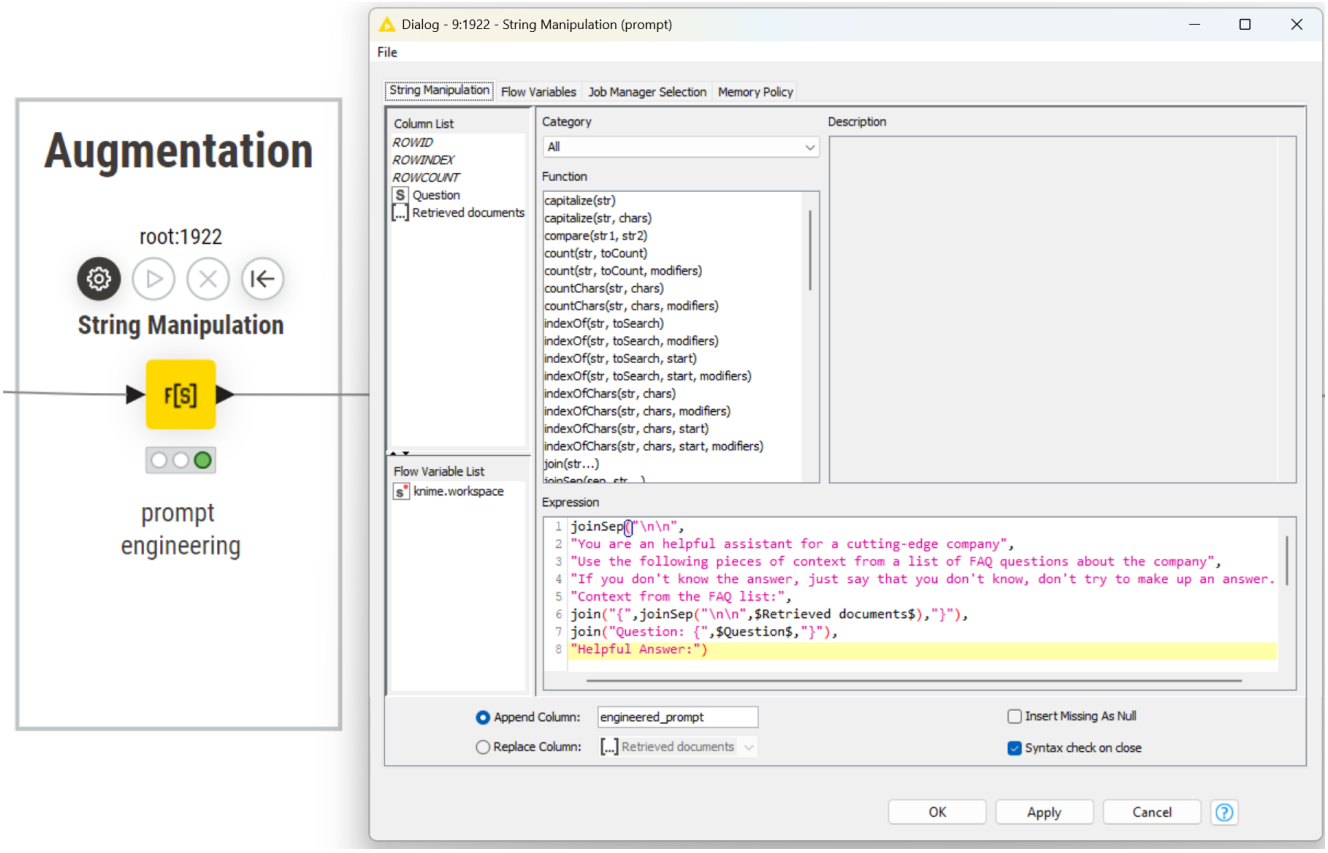


Figure 23. Prompt constructed using the String Manipulation node.

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*.

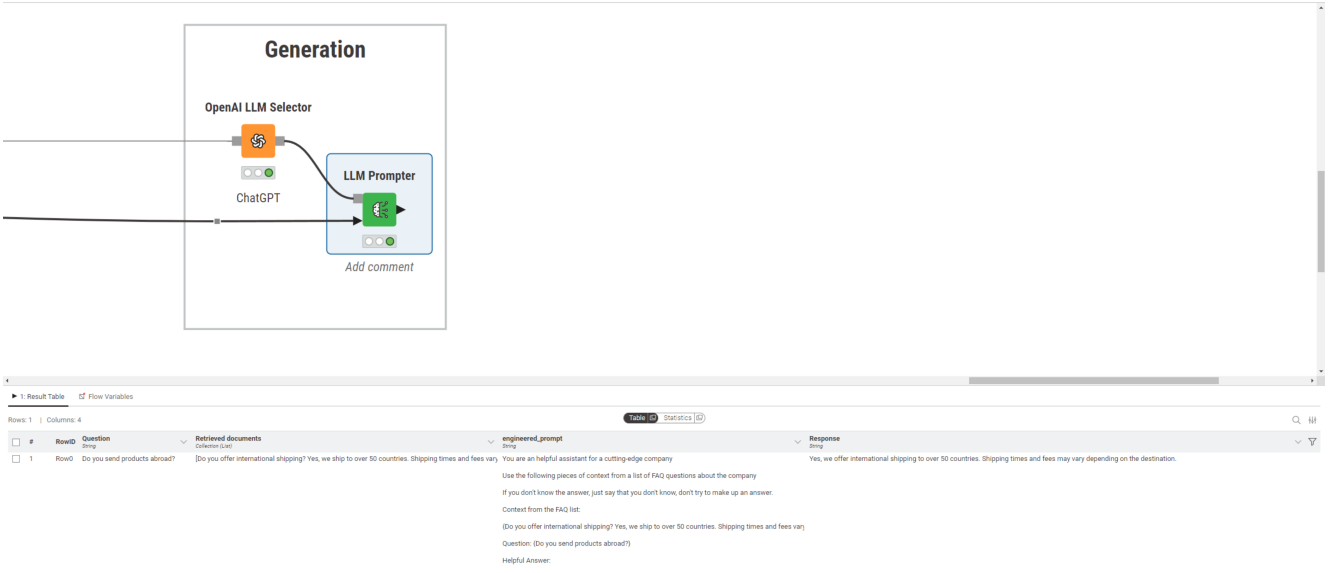


Figure 24. LLM output preview in the response column

Agents

An **agent** is a language model that solves tasks step by step by choosing from a set of available tools. It reads a user's request and selects the most relevant sequence of **tools** to generate a useful response.

When an agent receives a task from the user, it follows a step-by-step thinking process:

1. Understand a task.
2. Call a tool.
3. Evaluate the result.
4. Decide if more steps are needed.
5. Continue until the task is completed.

In KNIME, agents and tools are workflows. The agent orchestrates these workflows to complete tasks, while the tools provide specific capabilities like data processing, analysis, or external API calls.

In this guide, you will learn how to build agentic workflows in KNIME Analytics Platform. You will discover how to create tools, define their behavior, and set up the agent's reasoning process.

For more information about agents, see the Agentic AI and KNIME blog post.

The two layers of KNIME agentic workflows

A KNIME agentic workflow consists of two layers:

1. **Communication layer**

   The **communication layer** manages the agent's reasoning. It interprets the user's request, chooses which tool (or tools) to call, and evaluates the results. The agent moves step by step, making decisions and building a response based on the information it receives. Each step is represented as a message, allowing the full reasoning process to be tracked.

2. **Data layer**

   The **data layer** handles actual data processing. While the agent cannot directly view data tables, it can activate tools that read, filter, or analyze data. These tools run in the background and return only the final result. The agent can then use this result to

complete the task or continue reasoning.

How a KNIME Agent communicates using Messages

Messages are the data structure used to represent all communication between the agent, user, and tools.

Each Message includes:

- **Type**:
  - *User*: a user question or instruction
  - *AI*: a response generated by the agent
  - *Tool*: a response returned by a tool
- **Content**

  Text or images that make up the message body.

- **Tool calls (AI Messages only)**

  If the agent decides to call a tool, this section records which tool was called, the call ID, and the parameters provided.

- **Tool call ID (Tool Messages)**

  Links the tool response back to the specific tool call that triggered it.

Work with Messages

To work with messages in your workflows, use the following two nodes:

- **Message Creator node**

  Creates new Message objects by specifying the message type (User, AI, or Tool), content (text or images), and any relevant properties.

- **Message Part Extractor node**

  Extracts specific components from an existing Message, such as its content, tool call details, tool call ID, or other metadata.

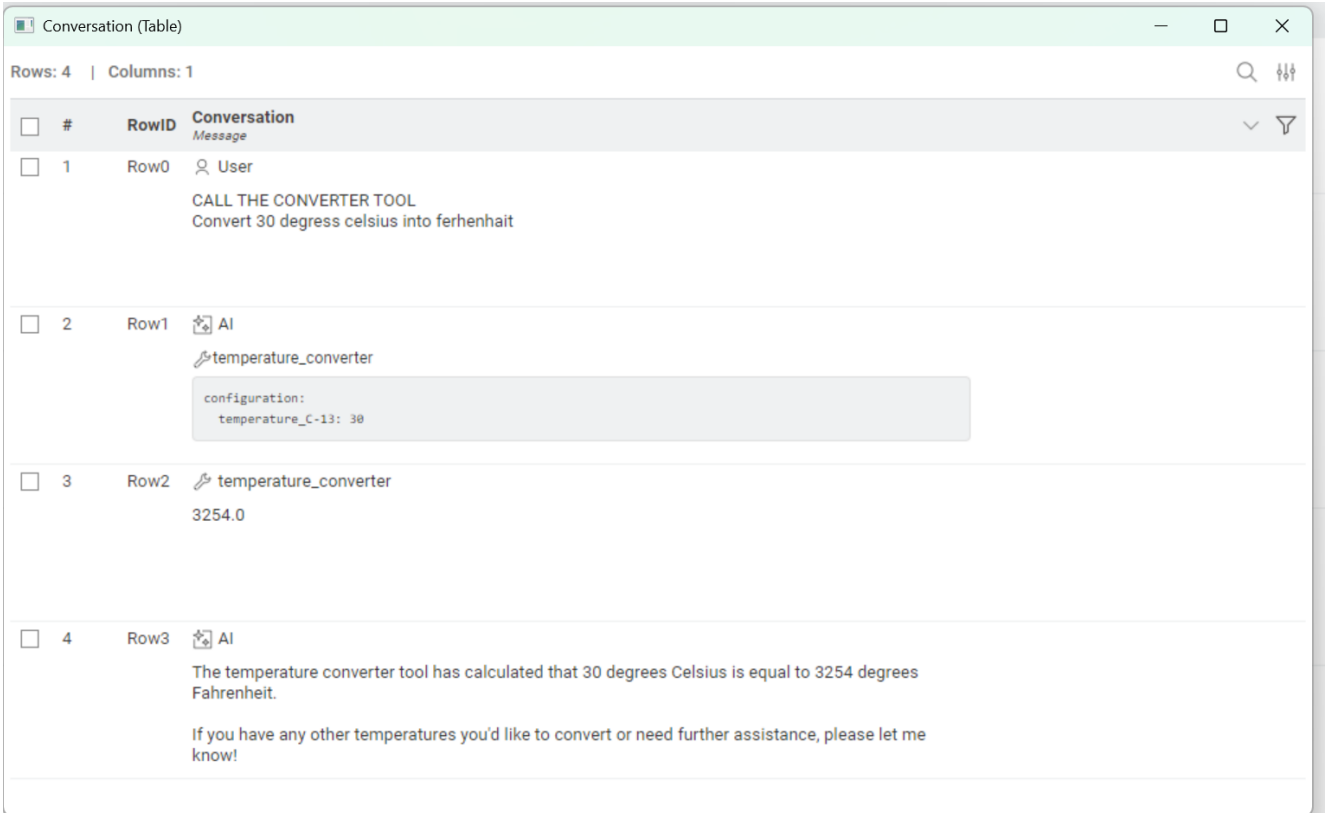


Figure 25. Example of a conversation between user, AI and a tool called "temperature converter".

Add functionality with Tools

In KNIME, a **Tool** is a workflow that an agent can call to help complete a task. As the agent reasons through a problem step by step, it decides when and how to use available tools to get the job done. Each tool adds a specific capability the agent can rely on during its decision-making process.

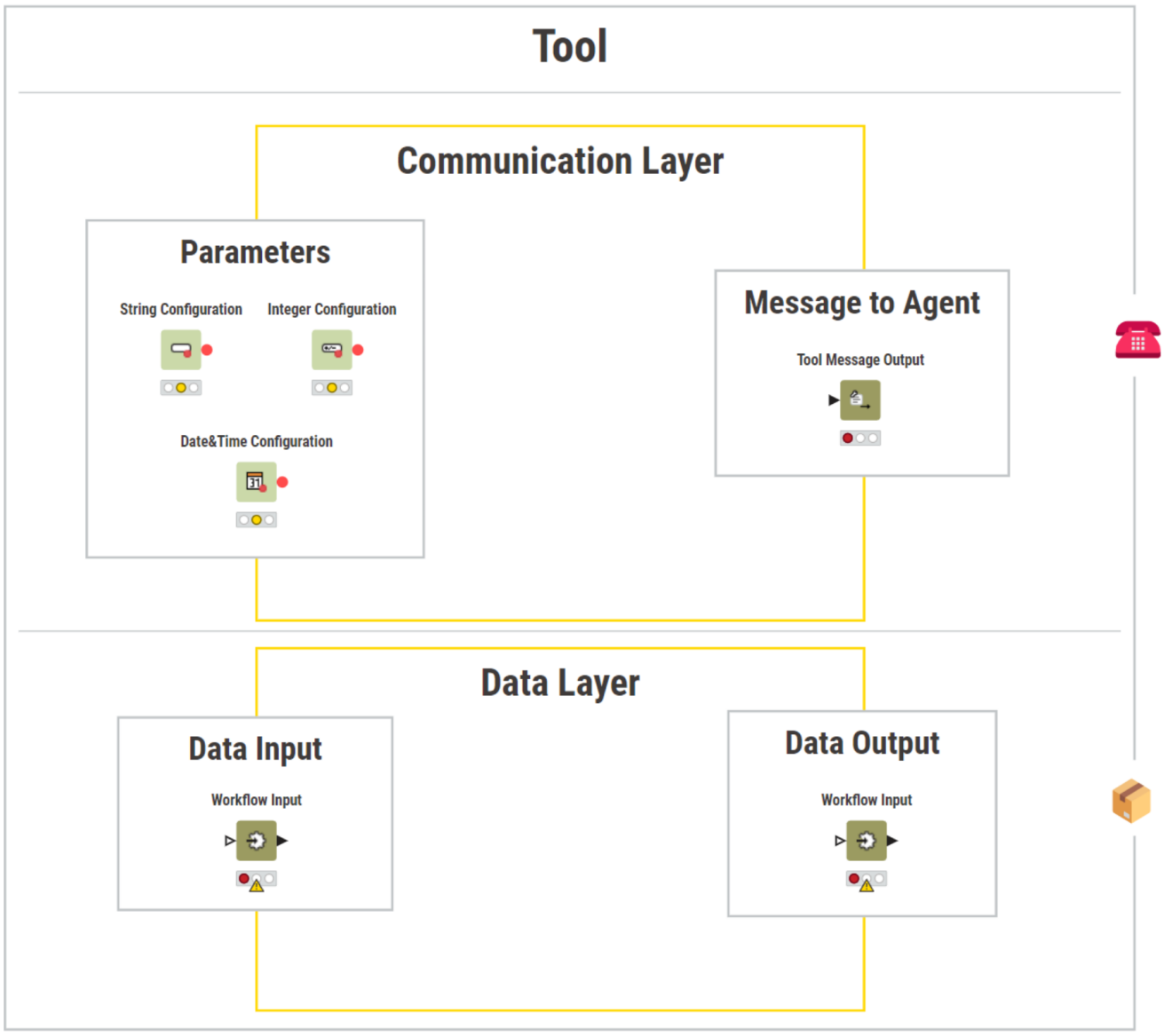


Figure 26. Structure of a tool workflow in KNIME Analytics Platform

Communication layer: guide the Agent's reasoning

Describe Tool behavior for the Agent

Define a clear description for each Tool in the workflow's info field. The Agent reads this description and decides when to use the Tool. The description should explain:

- The task performed by the Tool.
- The expected input data.
- The output produced.
- The types of questions the Tool is designed to answer.

A well-written description allows the agent to reason effectively about the available options.

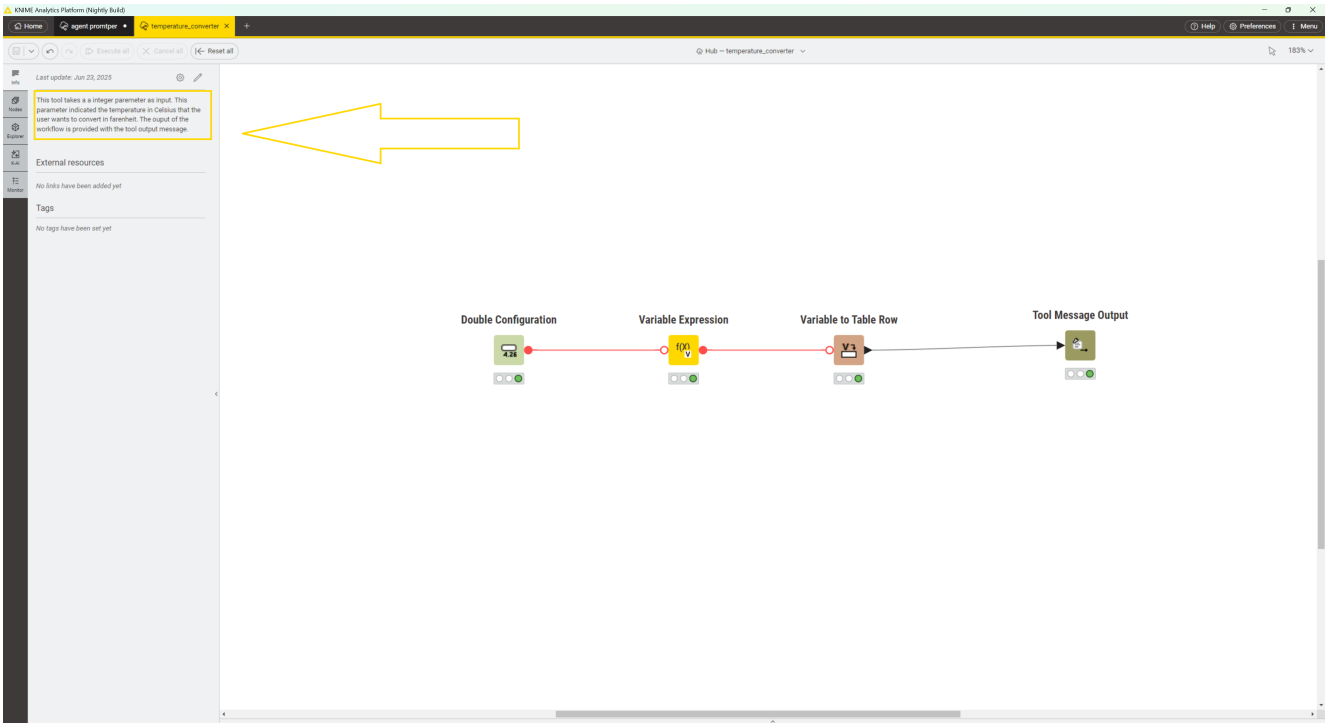


Figure 27. The tool description field in KNIME Analytics Platform. This is used by the agent to understand when to call the tool.

Return Tool results to the Agent

The Tool Message Output node provides optional feedback to the agent after a tool execution.

- Include this node if textual output is needed for the agent to reason with after the tool call.
- Omit it if no cognitive output is necessary.

The node reads the first value from the first row and first column of its input table. This string becomes the content of the Tool Message returned to the agent.

Use this node to return:

- Summaries of processed data.
- Short textual insights.
- Confirmations or intermediate results.

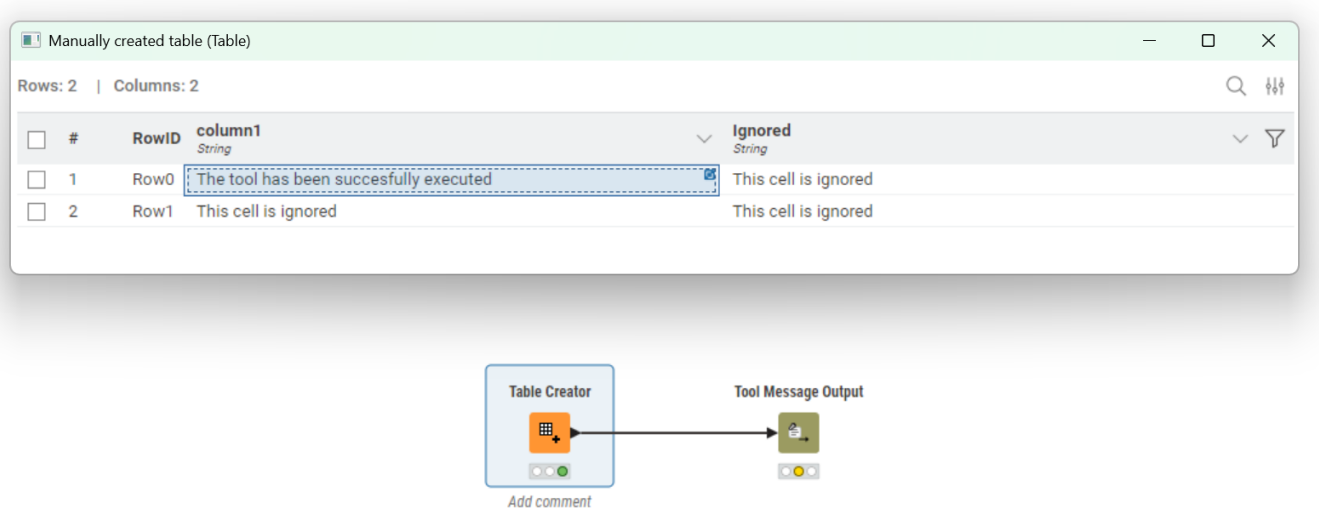


Figure 28. A table passed to the Tool Message Output node. Only the first cell (first row, first column) is used as the output message.

Let the Agent set parameters

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

For each parameter:

- Provide a clear parameter name (used as the variable name).
- Write a concise description explaining its purpose.

The agent reads these definitions to determine which parameter values it can set during the tool execution.

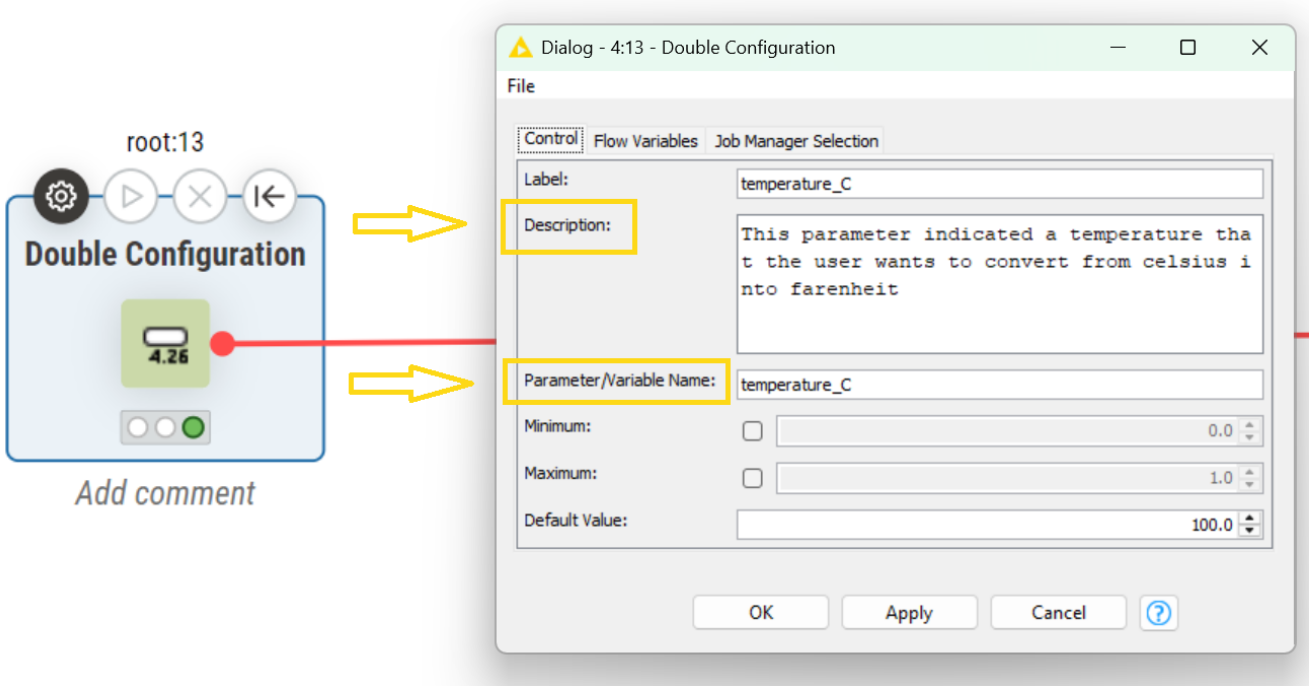


Figure 29. The configuration window of a Double Configuration node where both Name and Description fields are entered to guide the agent.

Data layer: automate data flow

Define what data the Tool uses and returns

Use the Workflow Input nodes 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 type.

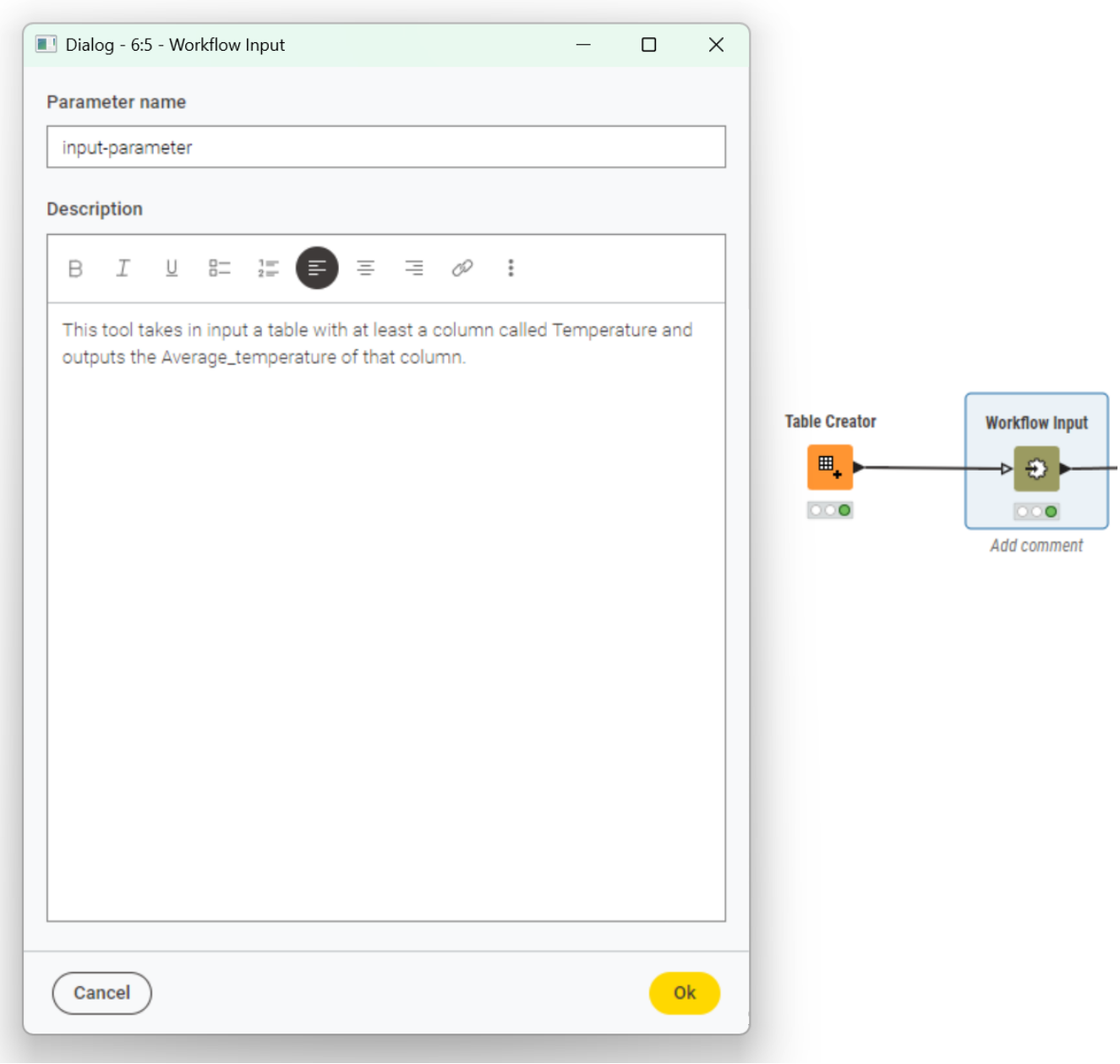


Figure 30. The Workflow Input node contains a description of the data that the tool can process

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

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

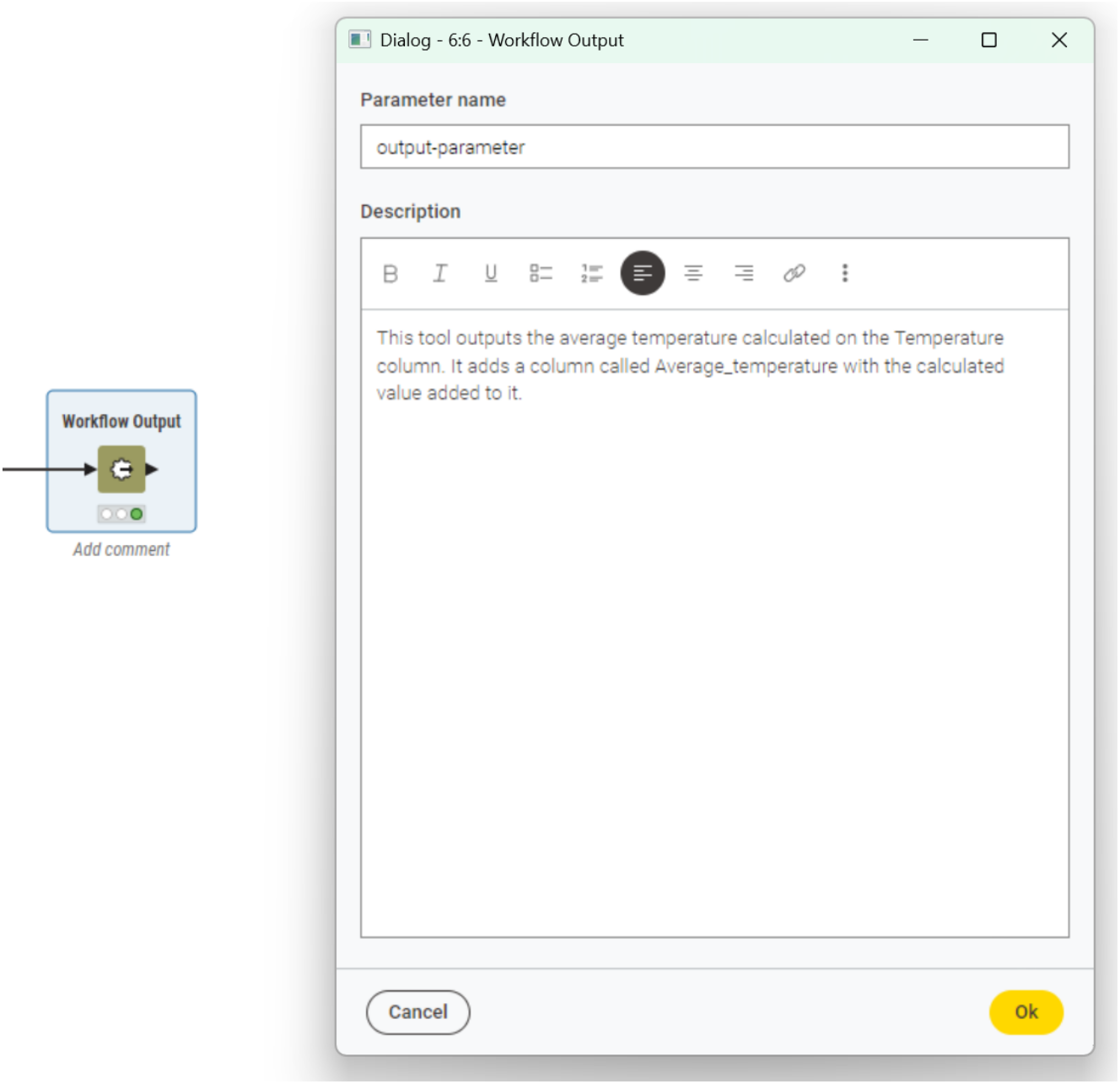


Figure 31. The Workflow Output node contains a description of the output table, after processin in tool

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

After defining individual Tool workflows, create a separate workflow to collect and prepare the list of available Tools for the agent:

1. Store all Tool workflows inside a folder.

2. In the same directory of the tool's folder, create a new workflow
3. Use the List Files/Folders node to retrieve the list of workflows from the folder.
4. Pass this list into the Workflow to Tool node to convert each workflow into a Tool object.

The output of the Workflow to Tool node includes metadata for each Tool:

- The icon shows wheter the Tool has a description.
- The icon 1 tells you how many parameters the Tool has.
- The icons 2 3 tell you how many data inputs and outputs the Tool has.

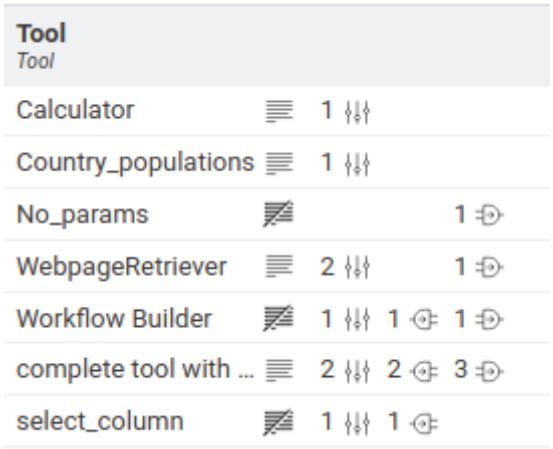


Figure 32. Output of the Workflow to Tool node showing metadata for each tool, including description status, number of parameters, and number of data inputs and outputs.

This feature makes inspecting and validating the agentic architecture significantly easier. The metadata allows quick verification of whether any tool is missing critical elements (such as a description or parameter definitions) before execution. It ensures that Tools are fully prepared and well-described for the agent to reason effectively.

Keep in mind that:

- Every time a Tool workflow is modified (e.g., description, parameters, or data layer adjustments), save the updated Tool workflow.
- Then re-execute the workflow containing the Workflow to Tool node to refresh the Tool list.
- The Workflow to Tool node does not automatically detect changes; re-running it is required to update the Tool definitions accordingly.

Run and test the Agent

Agent Prompter node

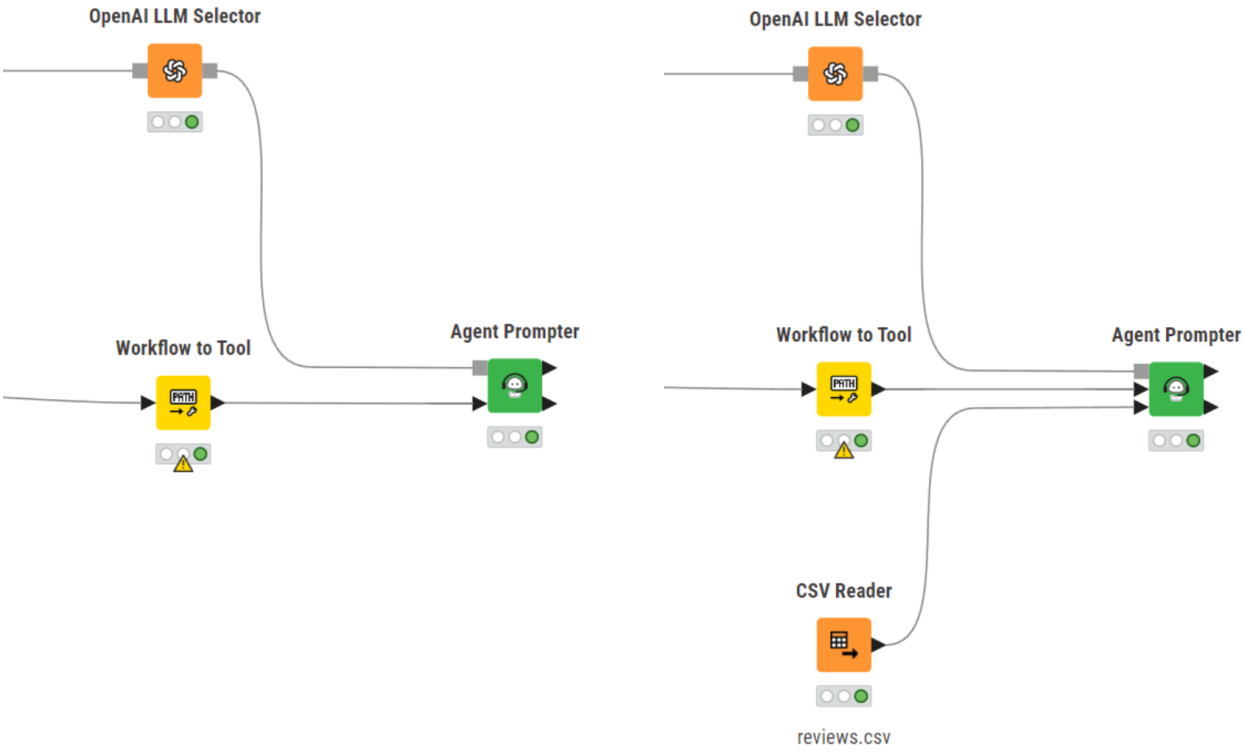
The Agent Prompter node executes the agentic reasoning loop. It takes:

- *System Message*: defines the agent's role, goals, and behavior (e.g. "You are a support agent answering product questions").
- *User Message*: the task or question to solve (e.g. *"What is the warranty for product X?"*).
- *Tool List*: the set of Tools available for the agent, generated using Workflow to Tool node.
- *Data Inputs and Outputs (optional)*: Tools can handle data tables using Workflow Input and Workflow Output nodes. By default, the Agent Prompter node has no data ports.

To enable data ports:

1. Right-click on the Agent Prompter node.
2. Select *Add Input Port* or *Add Output Port* from the context menu.
3. Choose the type of port to add (e.g., data table input or output).

Table 1. On the left: The Agent Prompter node is configured without any data ports. Only the communication layer is used, the agent reasons and calls tools without exchanging external data. On the right: The Agent has data input and output ports. External data (from the Table Reader) is provided as input and can be passed into the tools via the data layer.



Agent Chat View node

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

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 (e.g. *"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.

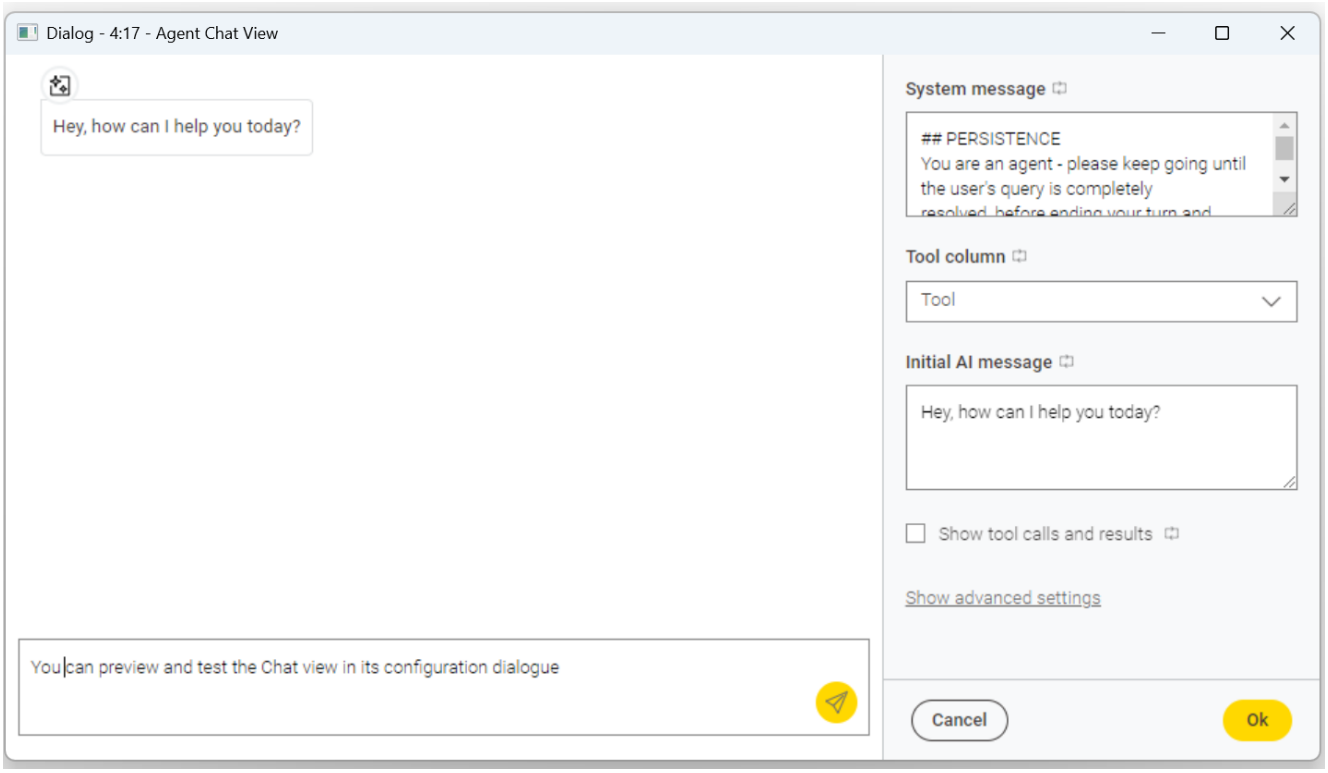


Figure 33. The configuration dialogue of the Agent Chat View node

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

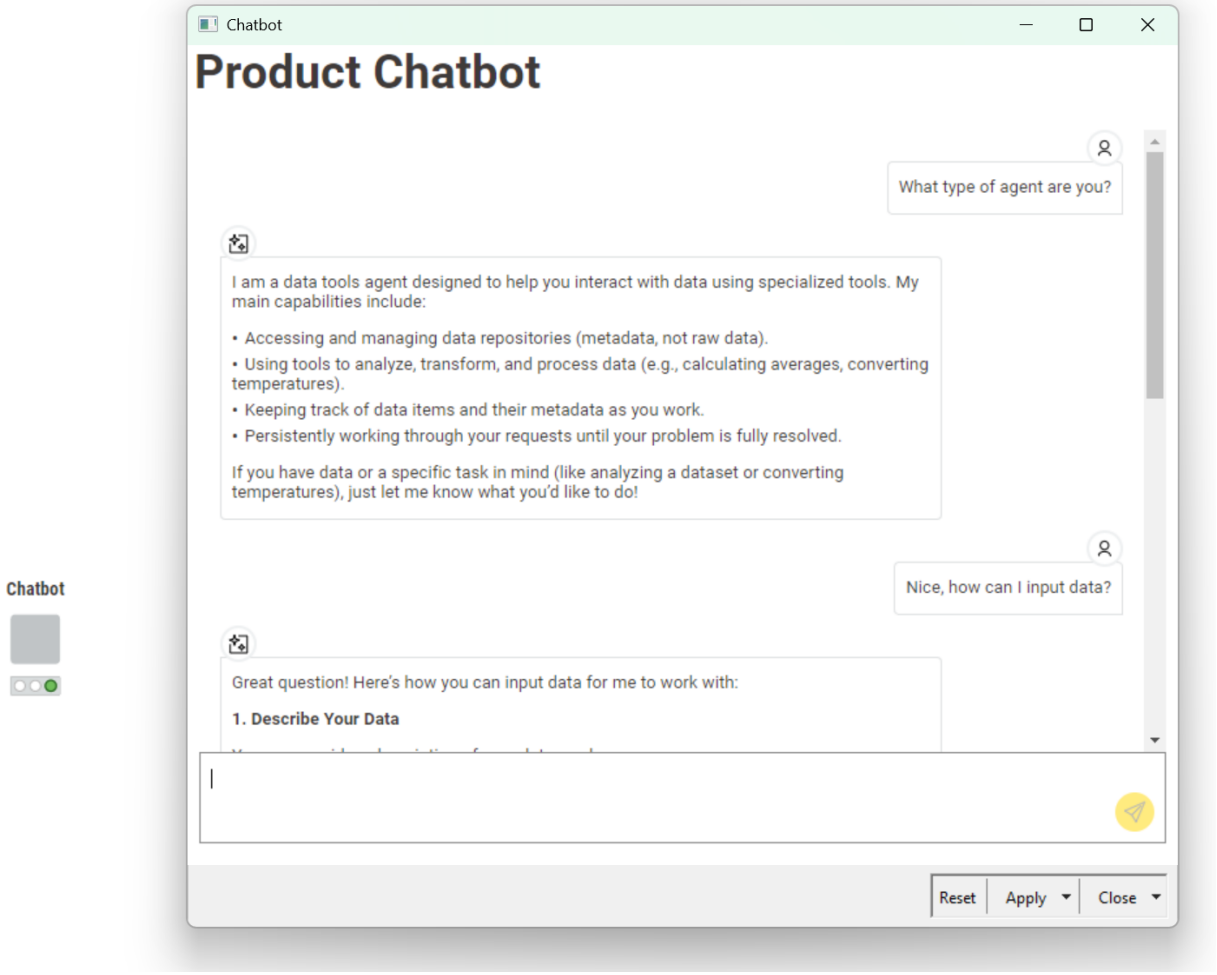


Figure 34. The Agent Chat View node is embedded inside a component named chatbot, enabling real-time conversation with the agent.

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 List Files/Folders 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.

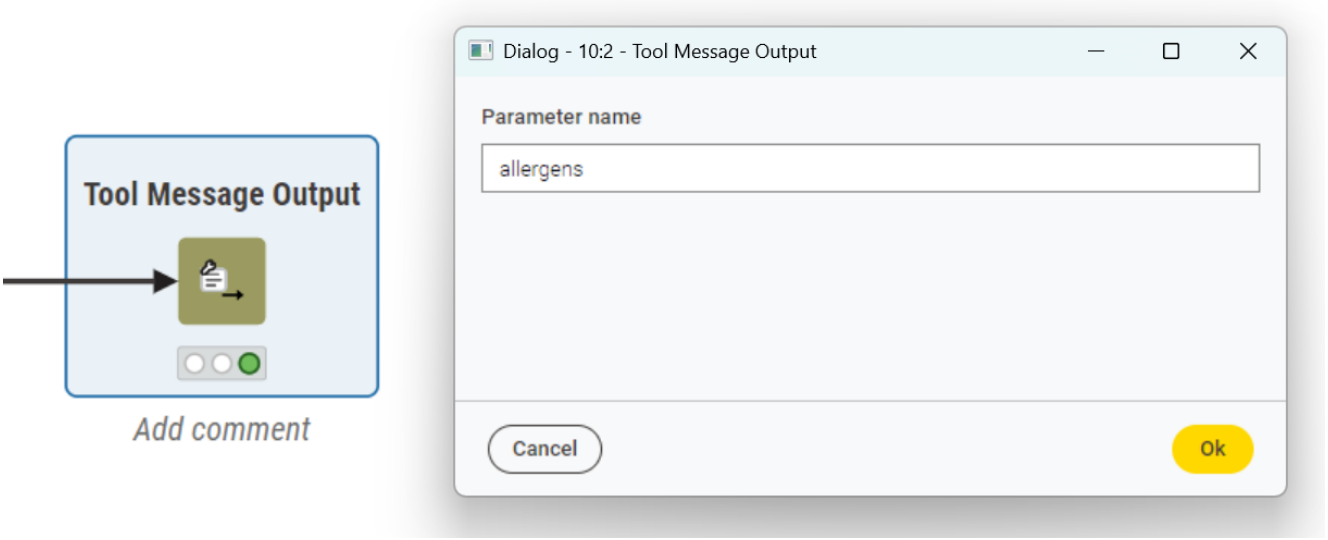


Figure 35. The Tool Message Output configuration. The parameter is renamed to allergens to better describe the content.

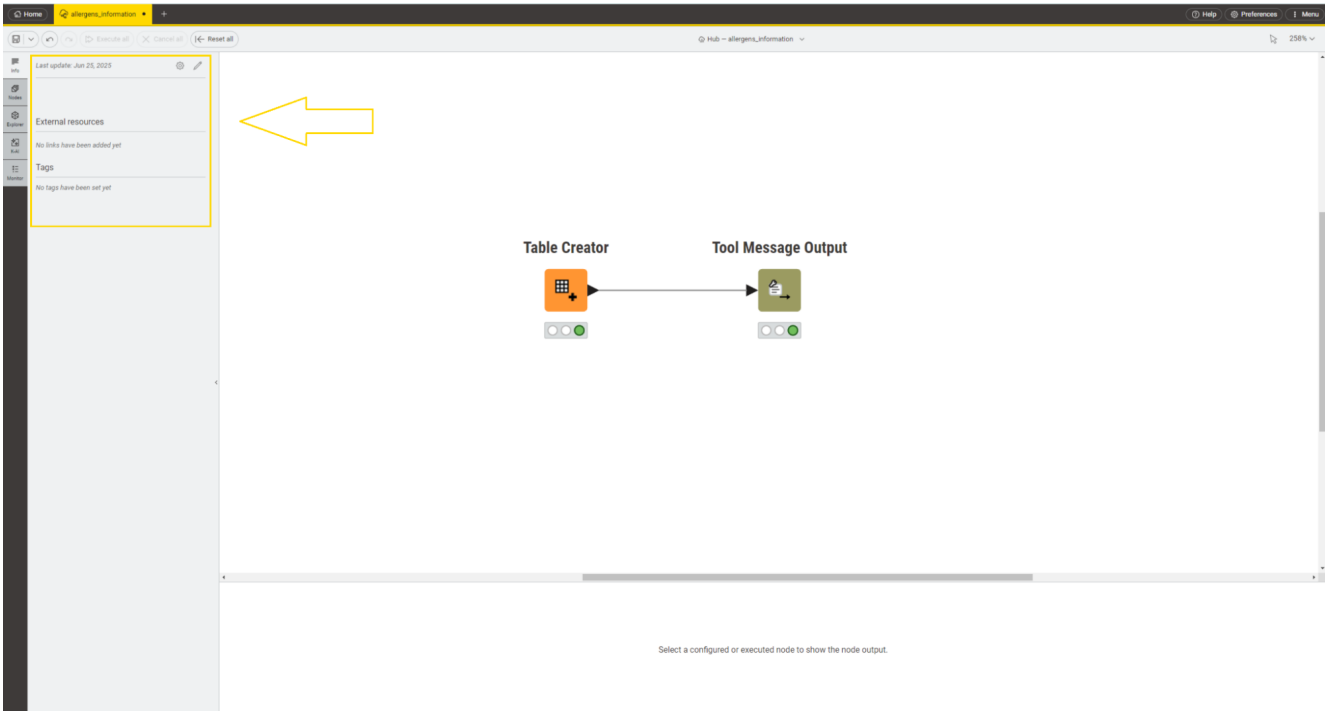


Figure 36. The tool is complete in structure but still needs a description so the agent can understand when to use it.

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* tab 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.

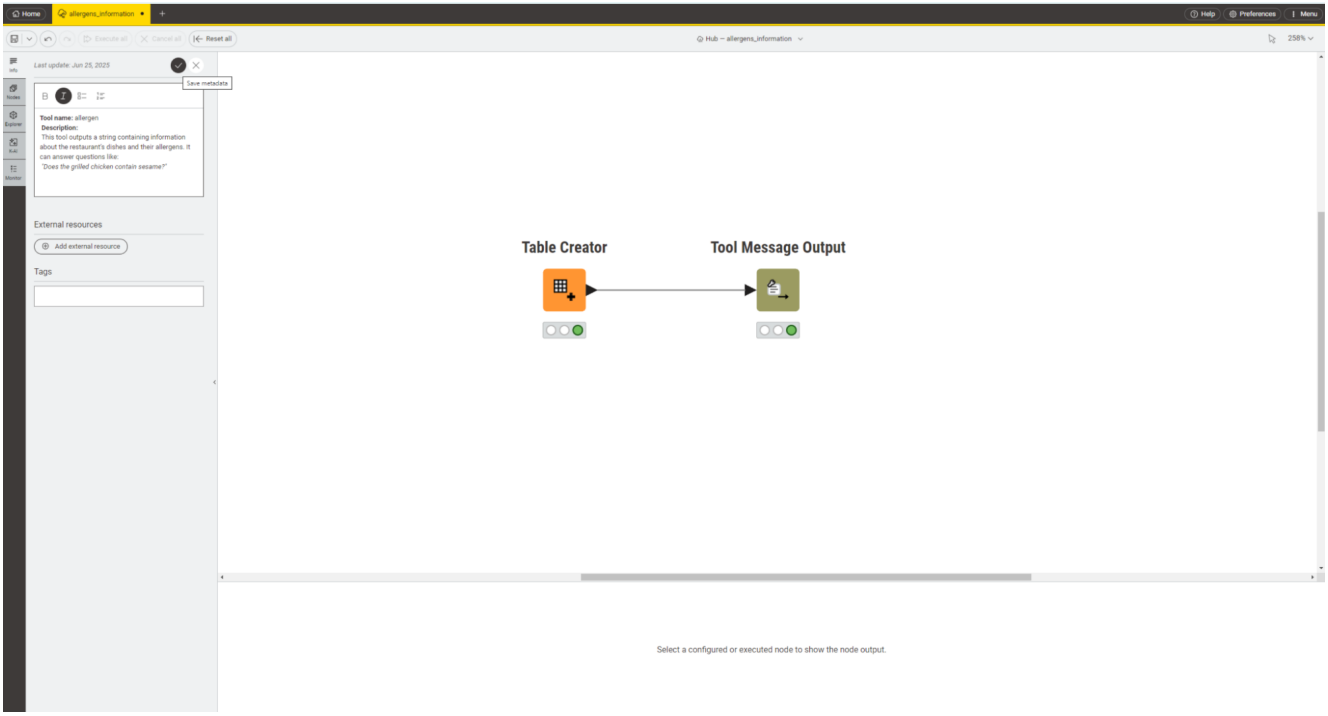


Figure 37. The workflow now has a description

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.

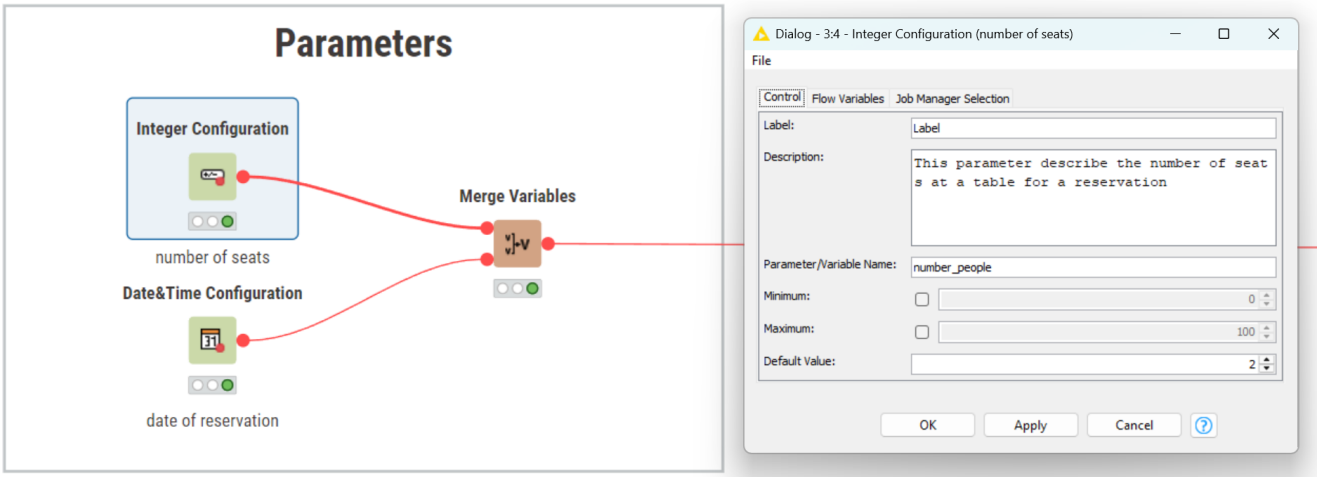


Figure 38. The configuration dialogue of the Integer configuration node

Filter results

Use an Empty Table Switch node to handle two paths:

- If matching tables are found:
  - Top K Row Filter 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.

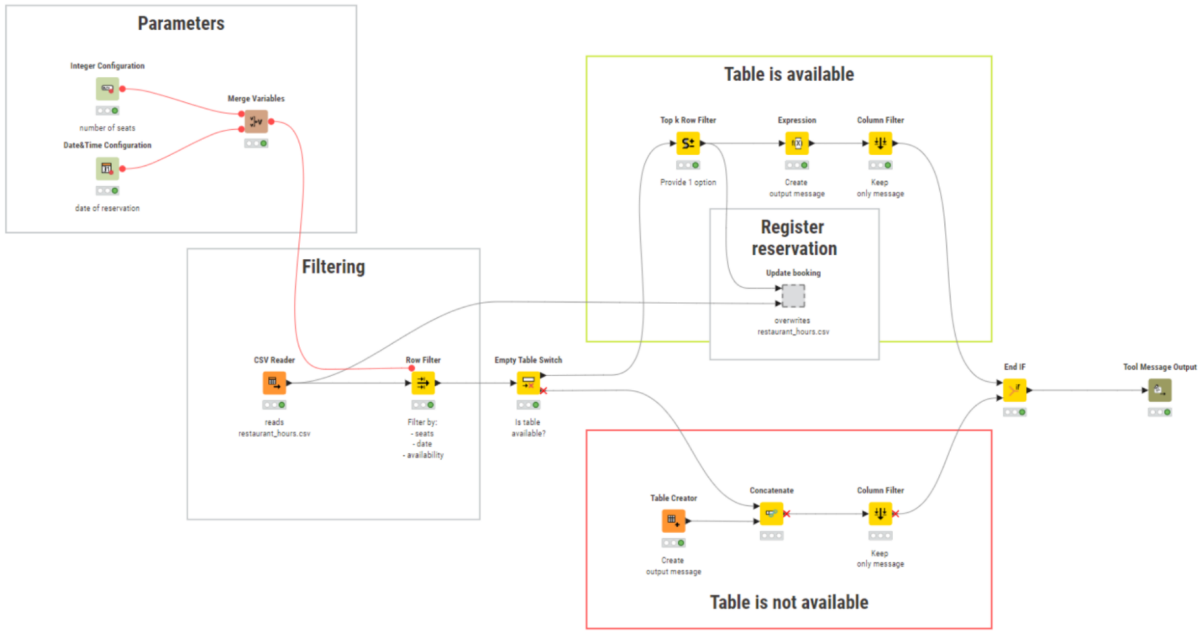


Figure 39. The Table Availability tool workflow with two paths: available or unavailable, ending in a single message output.

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.

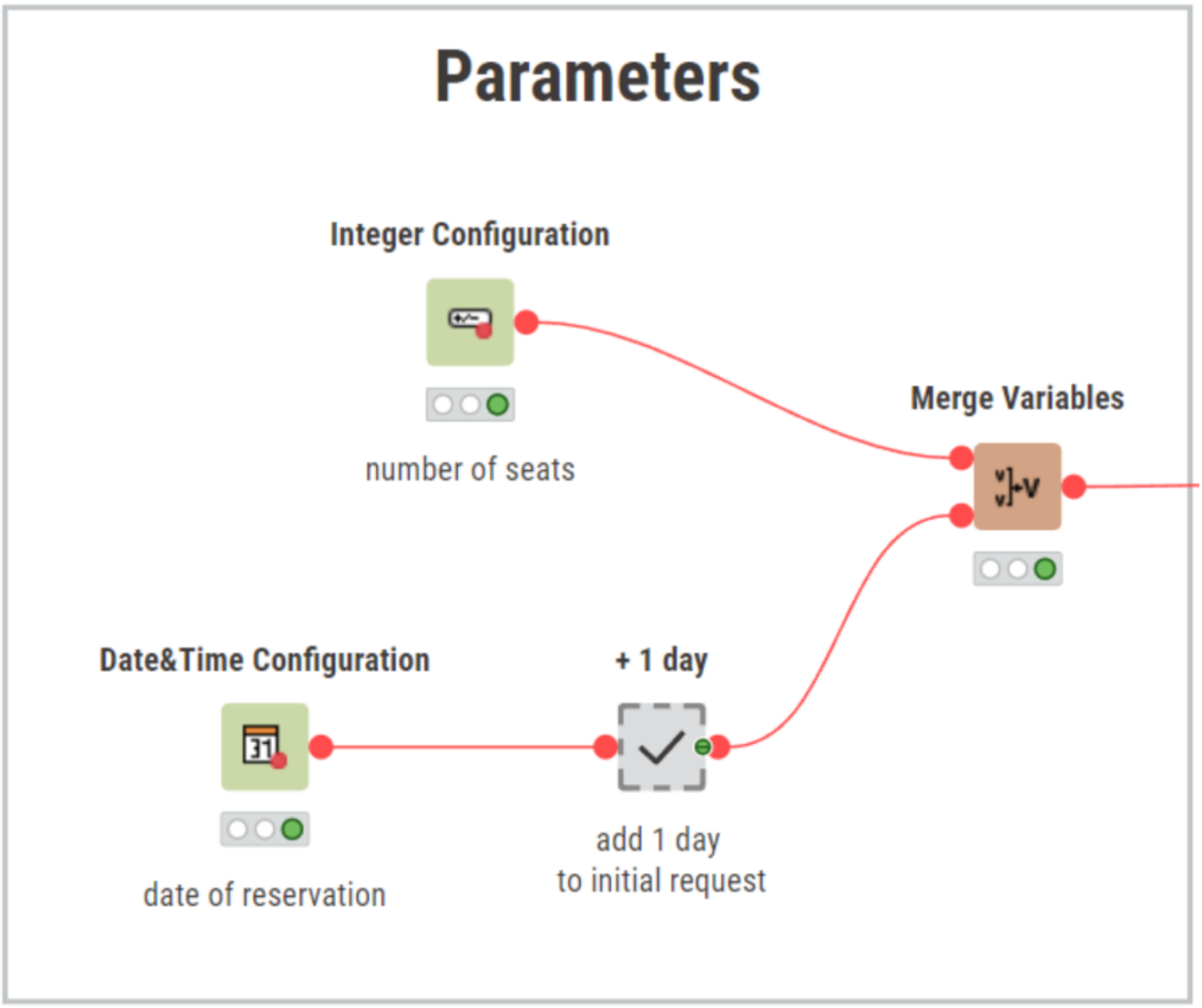


Figure 40. The parameter configuration of Tool 3. A Date&Time Shift adds one day to the requested reservation date to search for availability on the following day.

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.

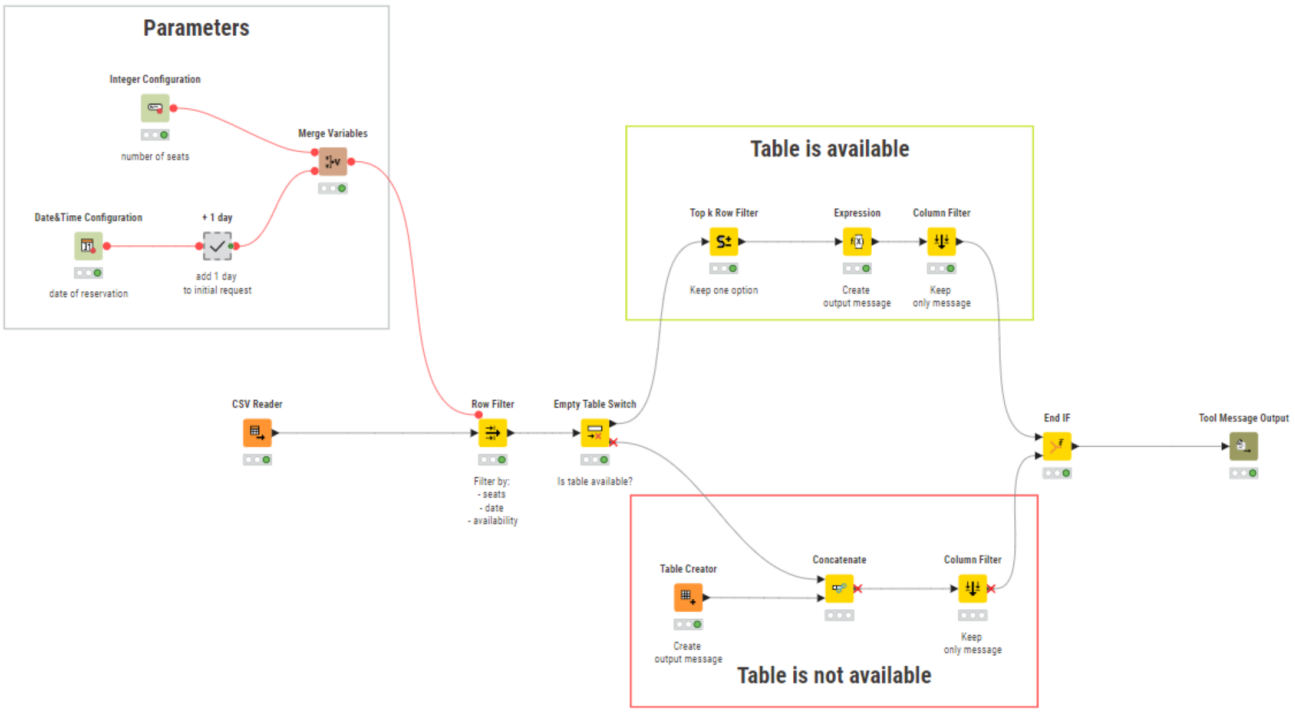


Figure 41. The Propose Alternative tool workflow that proposes an alternative reservation date to the user

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.

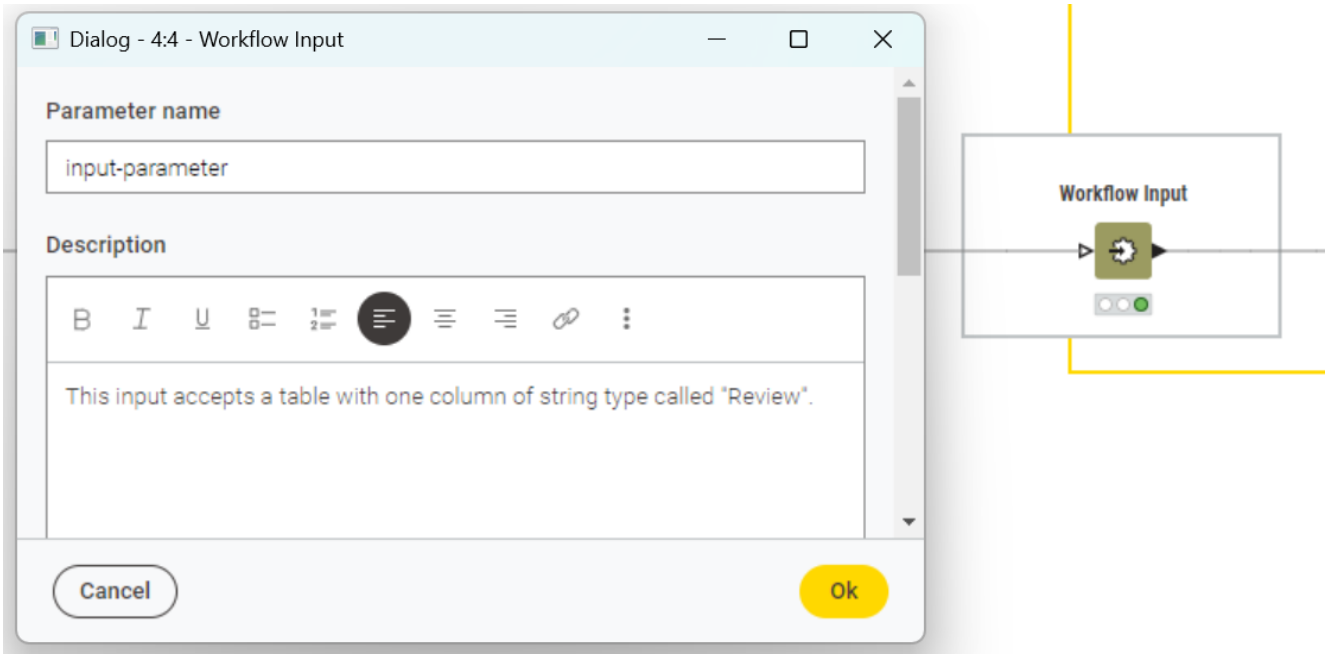


Figure 42. The Workflow input configuration dialogue

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.

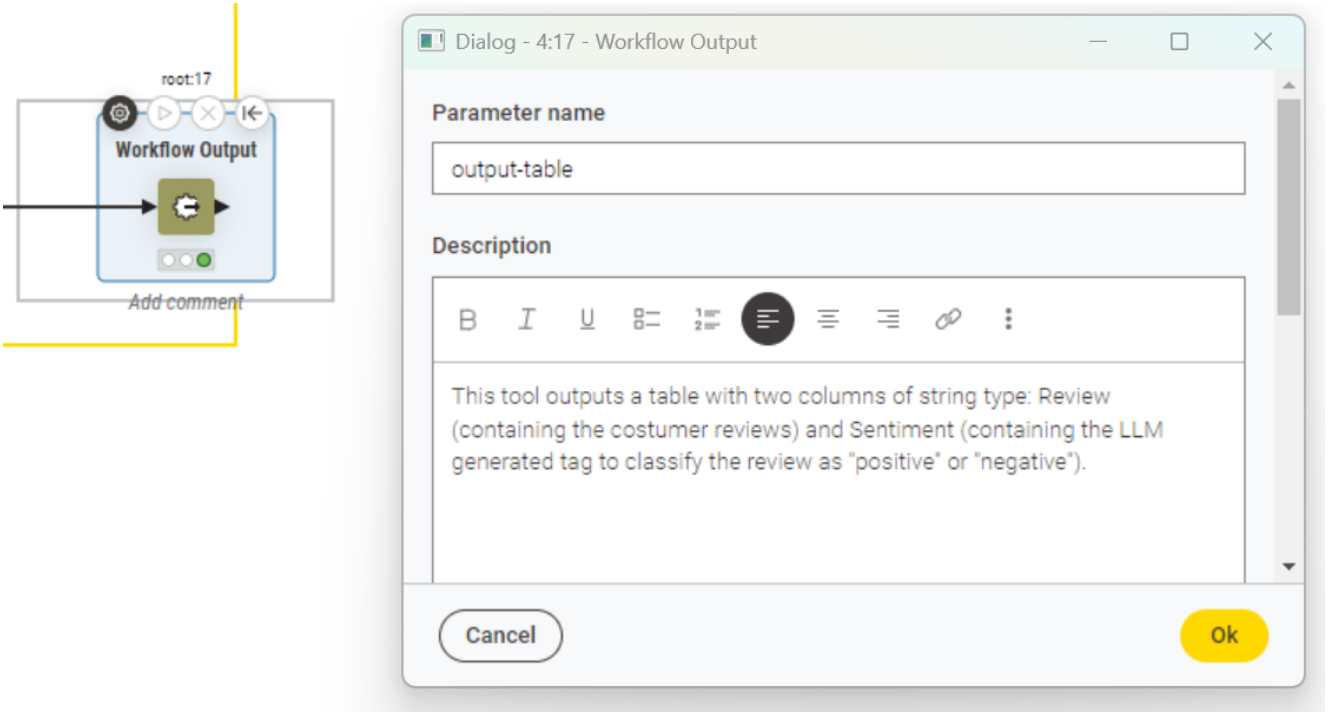


Figure 43. Workflow output node configuration

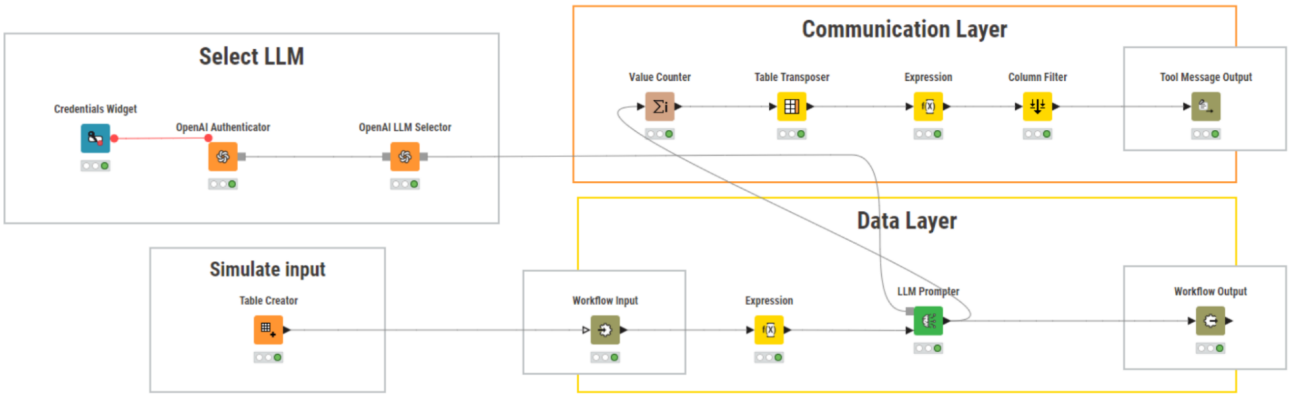


Figure 44. The Classify Reviews tool workflow that classifies reviews using an LLM.

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

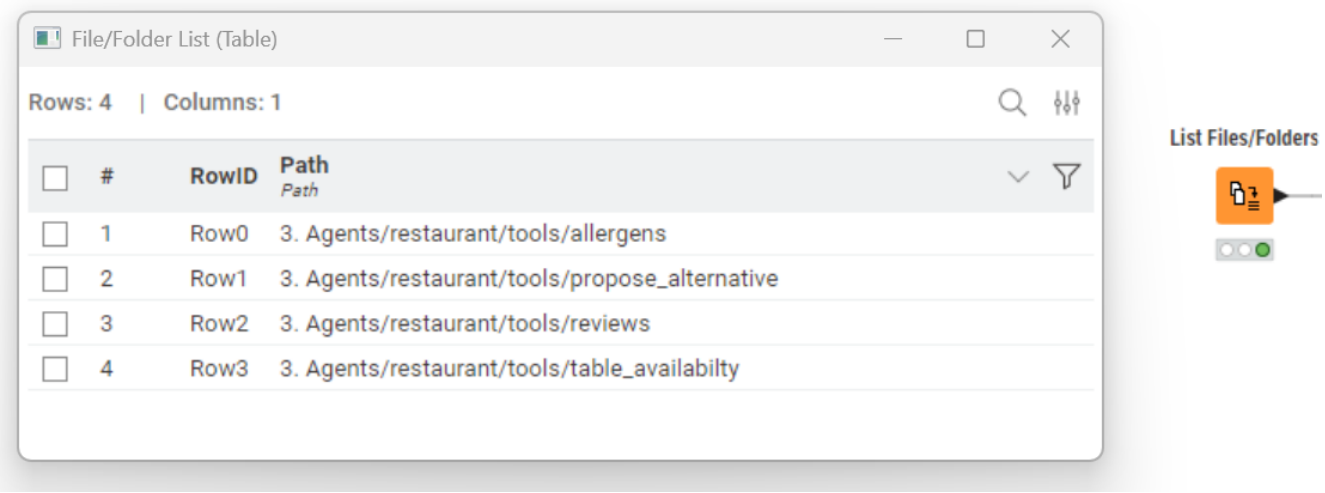


Figure 45. The List Files/Folders node reads all tool workflows from the selected "tools" directory.

- 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

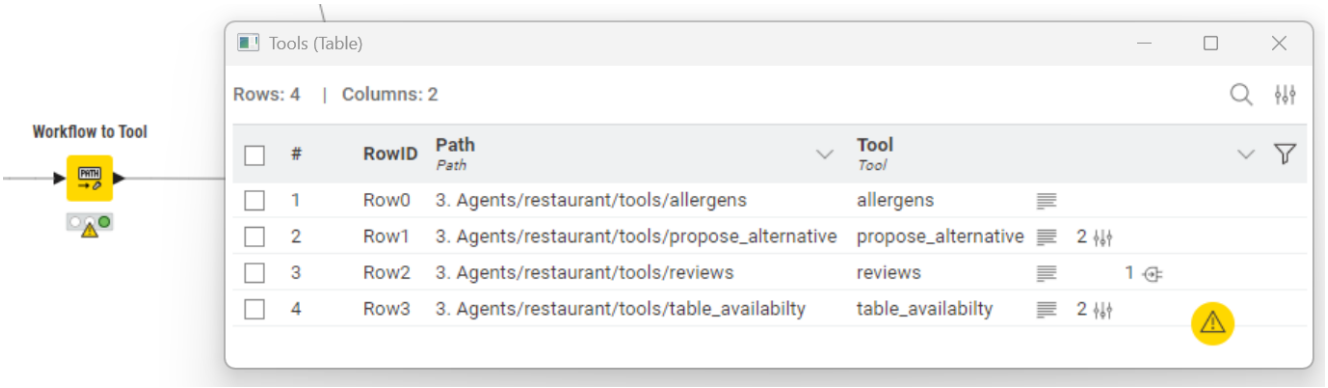


Figure 46. The output of the Workflow to Tool node shows icons that help verify tool descriptions, parameters, and data inputs/outputs.

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

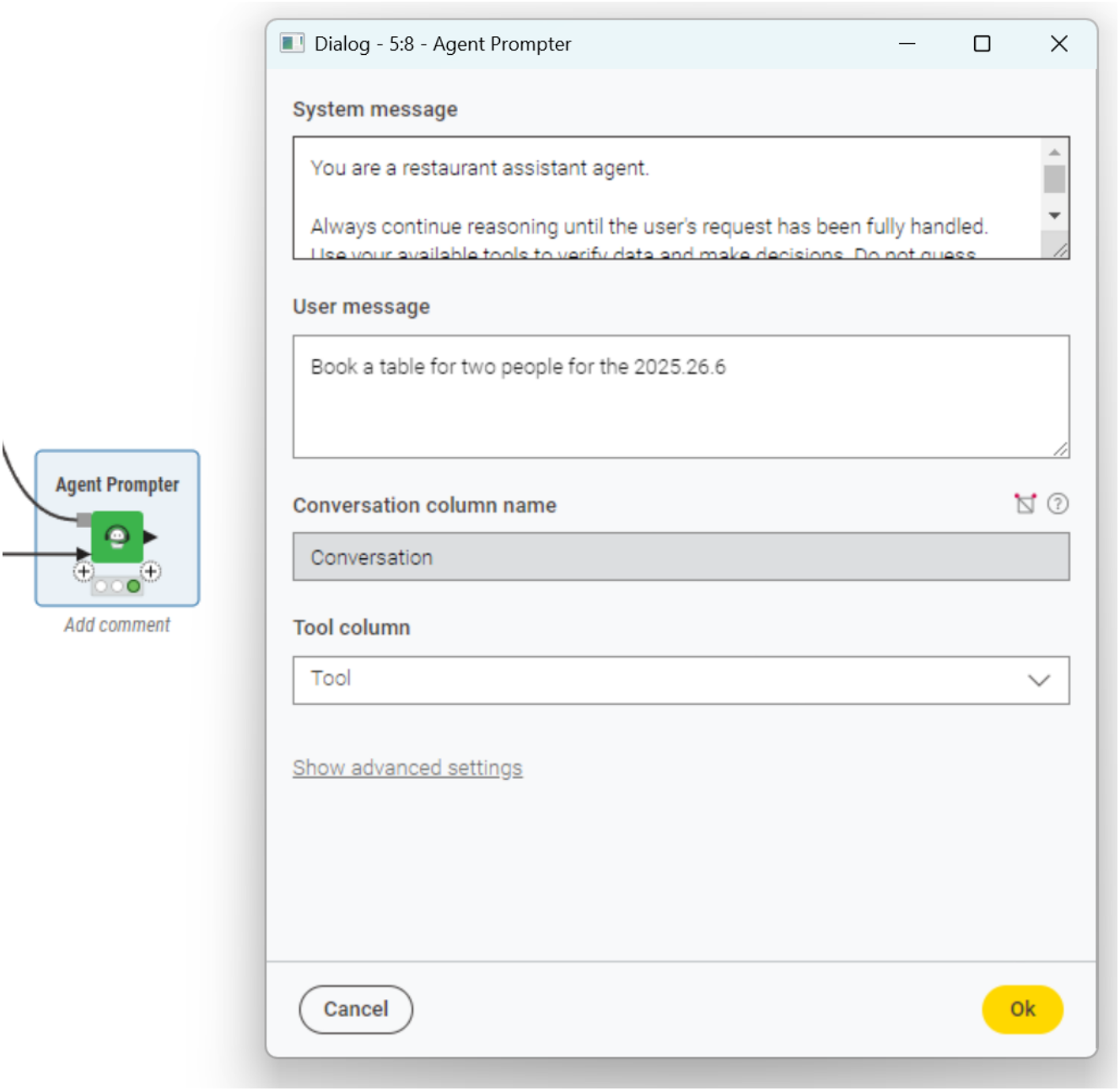


Figure 47. The Agent Prompter configuration dialog

- 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.

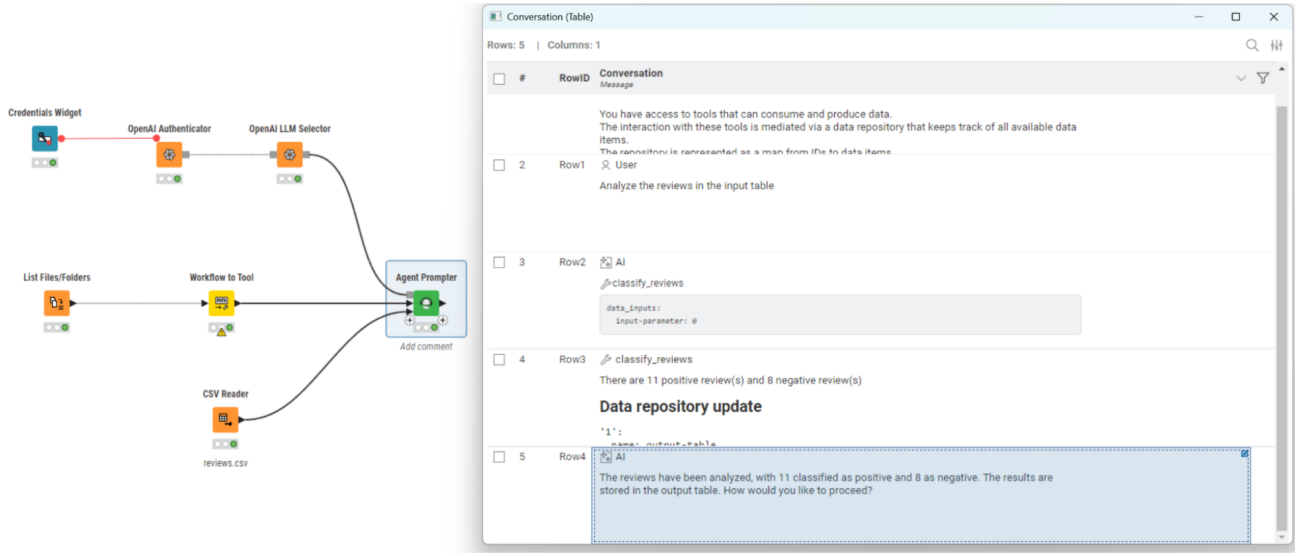


Figure 48. Agent Prompter with an added output port: the agent can now return data along with its response.

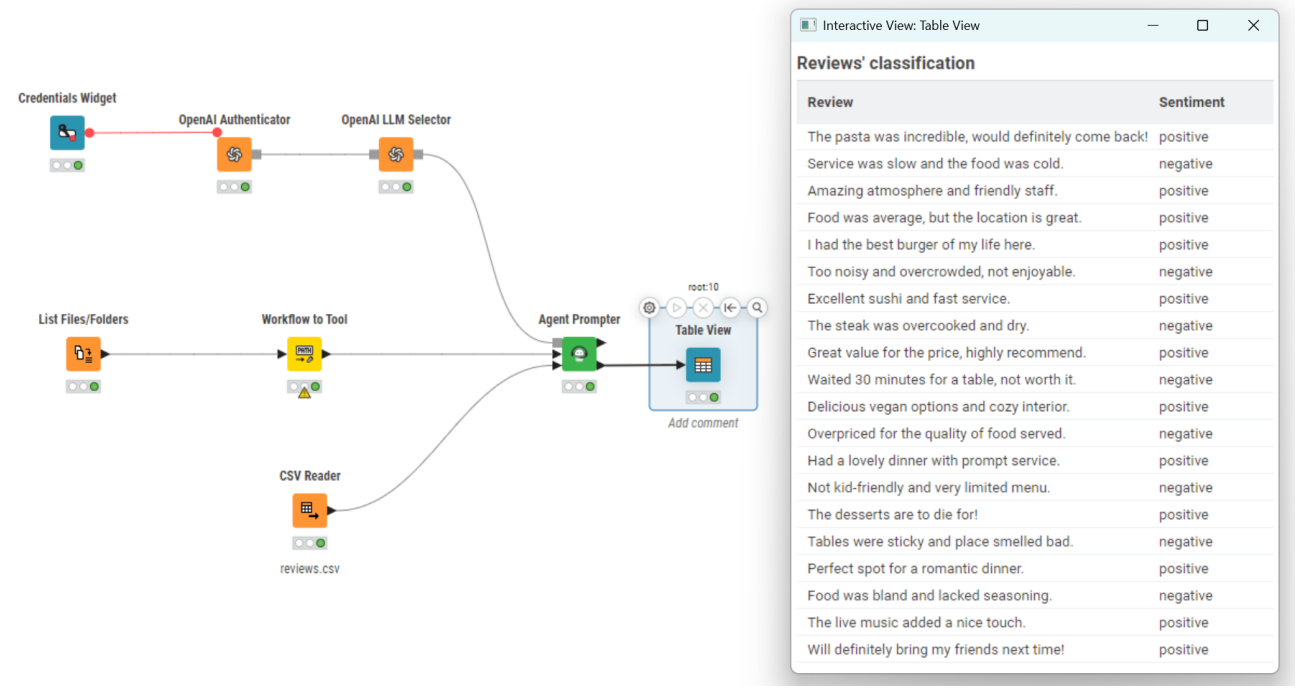


Figure 49. The Agent Prompter has one input and one output port: only the communication layer is visible to the user, no data output is returned.

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.

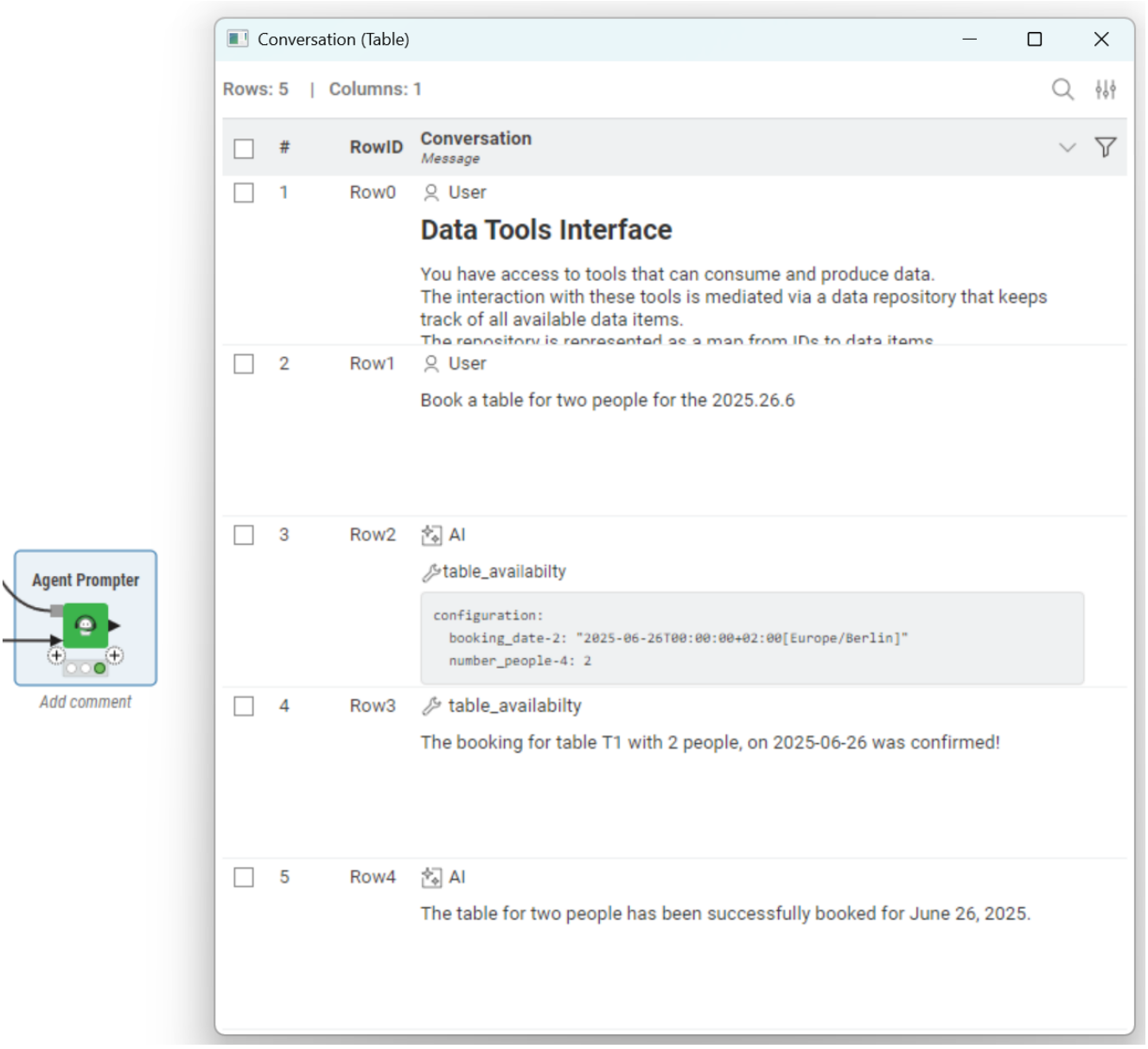


Figure 50. The Output view of the Agent Prompter node message

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.

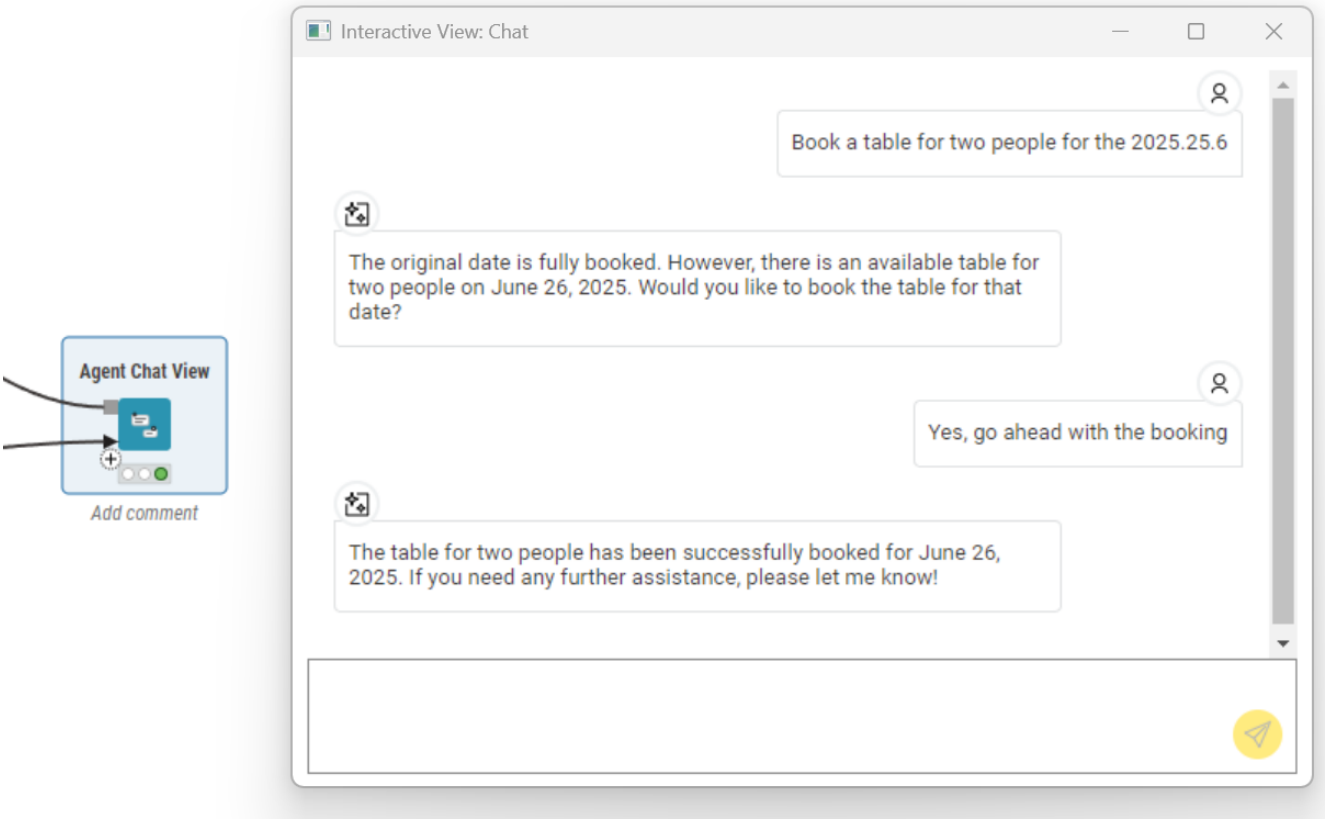


Figure 51. The Agent Chat View provides a live conversation interface.

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.

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

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