

Complete Guide - RAG Agent Workflow with n8n

Aziza Neffati

4 août 2025

Table des matières

1	Introduction	2
2	Prerequisites	3
3	Installing Node.js	3
4	Installing n8n	4
5	Starting n8n	4
5.1	General Architecture	4
6	Importing an n8n Workflow	4
7	Part 1 : Document Ingestion	5
7.1	Node 1.1 : Execute Workflow	5
7.2	Node 1.3 : Switch	7
7.3	Node 1.4 : Extract from File1	7
7.4	Node 1.5 : Convert to JSON	8
7.5	Node 1.6 : Qdrant Vector Store	9
7.5.1	Method 1 : Manual Creation via Qdrant Console	9
7.5.2	Method 2 : Direct Creation of the Vector Store via n8n	11
7.6	Node 1.7 : Default Data Loader	12
7.6.1	Configuration	12
7.6.2	Metadata	12
7.7	Token Splitter	13
7.7.1	Configuration Parameters	13
7.7.2	Functionality	13
8	Part 2 : Chat and Responses	15
8.1	Architecture	15
8.2	Node 2.1 : When chat message received	15
8.3	Node 2.2 : AI Agent	15
8.4	2.3. OpenAI Chat Model1	17
8.5	2.5. Qdrant Vector Store1	17
9	Conclusion	19

1 Introduction

This guide details the steps to install and run a RAG (Retrieval-Augmented Generation) workflow locally using **n8n**. n8n is a no-code/low-code automation tool used to orchestrate tasks such as document analysis, API calls, file management, and more.

About Retrieval-Augmented Generation (RAG)

Retrieval-Augmented Generation (RAG) is an architecture that combines the strengths of information retrieval with text generation. Instead of relying solely on a language model's internal knowledge, RAG enhances it by retrieving relevant documents from an external database (like a vector store) and using them to generate more accurate, context-aware responses.

Why Use RAG ?

Traditional language models may hallucinate or provide outdated answers. RAG overcomes this by :

- **Grounding answers in external knowledge** : retrieved documents provide factual support
- **Improving reliability and traceability** : each answer can cite its sources
- **Handling domain-specific data** : RAG can answer questions based on your own private document base

General Architecture of a RAG System

A typical RAG system consists of three main components :

1. **Retriever** : searches a vector database to find relevant documents
2. **Reader/Generator** : uses the retrieved information to generate a response
3. **Vector Store** : stores document embeddings for fast and accurate retrieval (e.g., Qdrant, FAISS)

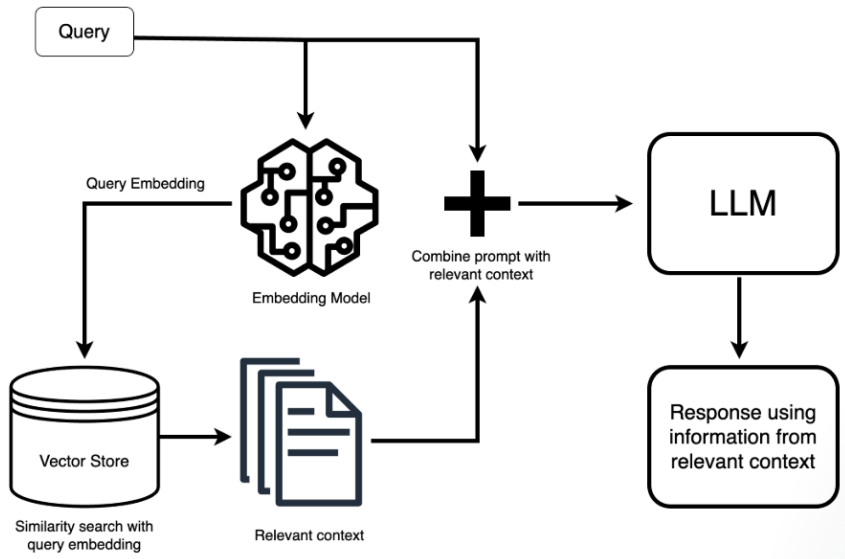


FIGURE 1 – Typical RAG Architecture

Our RAG Workflow with n8n

In this guide, we implement a custom RAG system using :

- **n8n** : a powerful no-code/low-code automation platform to orchestrate the workflow
- **Qdrant** : a vector store to store and retrieve document embeddings
- **OpenAI** : to extract embeddings and generate answers

What this RAG does :

- Automatically scans and indexes documents from your local folder
- Splits them into semantic chunks and stores them in Qdrant
- Answers user questions in real-time using OpenAI, based on those indexed documents
- Provides accurate, contextual, and source-cited answers

2 Prerequisites

Before starting, make sure you have the following installed on your machine :

- Node.js (version 18 or higher)
- npm (included with Node.js)
- Terminal or Command Prompt

3 Installing Node.js

1. Download the LTS version from <https://nodejs.org/>
2. Follow the installation instructions
3. Verify the installation with the commands :

```
node -v
npm -v
```

4 Installing n8n

In the terminal :

```
npm install -g n8n
npx n8n
```

5 Starting n8n

Run the command :

```
n8n
```

Then press `o` to open it in the browser. By default, the interface is available at : <http://localhost:5678>

5.1 General Architecture

- Ingestion Part : Read and index documents
- Chat Part : Conversational interface with semantic search
- Vector Database : Qdrant

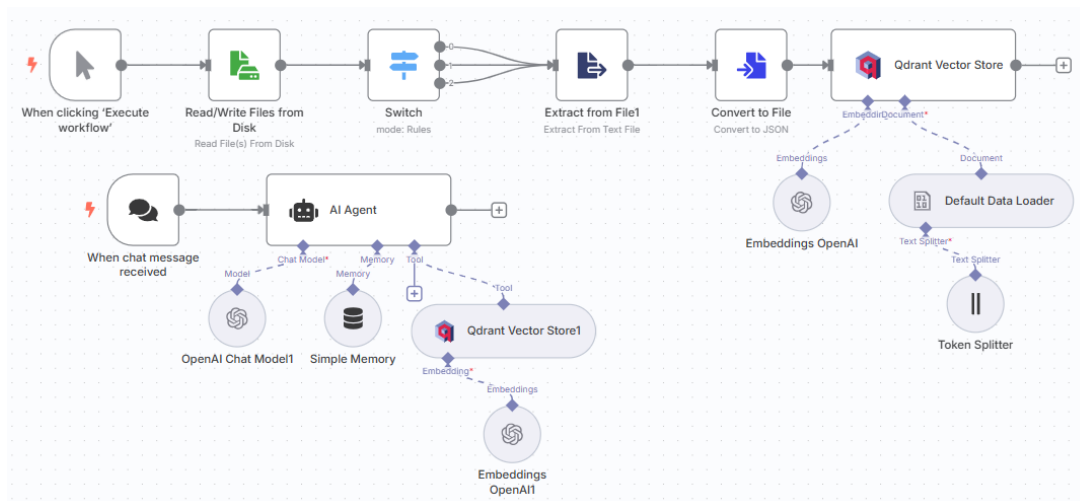


FIGURE 2 – General architecture of the RAG workflow

6 Importing an n8n Workflow

1. Launch n8n

2. Go to <http://localhost:5678>
3. Menu Workflows > + New Workflow
4. Click on > Import from File
5. Select the .json file
6. Click Save then Execute Workflow

7 Part 1 : Document Ingestion

Node Architecture

Execute Workflow → Read Files → Extract from file → Convert to JSON → Qdrant Vector Store → Default Data Loader → Token Splitter / OpenAI Embeddings

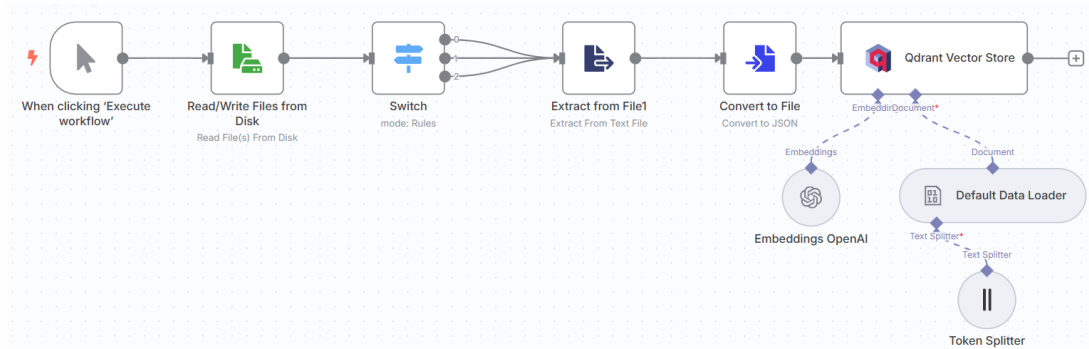


FIGURE 3 – Document Ingestion

Nodes and Functions

7.1 Node 1.1 : Execute Workflow

Role : Manually trigger the process

Configuration :

- Type : Manual trigger
- Usage : Click to start indexing the documents

1.2 Node "Read/Write Files from Disk"

Role : Recursively read files from the local file system

Configuration :

- **Operation :** Read Files
- **File(s) Selector :** D:/your/folder/**/*.{pdf,txt,docx,md}
- **Options :**

- Include file metadata
- Read file content as binary

Glob pattern explained :

- ****** : Recursively browse subfolders
- ***** : All files
- **{pdf,txt,docx,md}** : Specific extensions

FIGURE 4 – Read Files Node

Tip – Accessing all files in subfolders :

To access all files in the subdirectories, add `/**/*.*` to the end of the main path.

Example :

To read all files in `C:/Users/acer laptop/Downloads/`, including those in subfolders :
`C:/Users/acer laptop/Downloads/**/*.*`

Best Practices :

- **Valid file paths** : Ensure the file actually exists at the specified location. Otherwise, the node will throw a read/write error.
- **Use absolute paths** :
 - `C:/Users/Downloads/Documents/data.csv`
 - `../data.csv`
- **Path format** : Always use forward slashes `/` in the path.
 - Wrong : `C:\Users\acer laptop\Downloads\Commande.pdf`
 - Correct : `C:/Users/acer laptop/Downloads/Commande.pdf`
- **File names** : Avoid accents and special characters in file/folder names.
- **Access rights** : Make sure you have read/write permissions on the file.
- **Multiple files** : If you are processing several files, add them all in the File(s) selector (most tools support multi-selection).
- **File type** : Ensure the node supports the file format (e.g., PDF, CSV, JSON...).

7.2 Node 1.3 : Switch

Role : Redirect depending on the file type

Switch Execute step

Parameters Settings Docs

Mode
Rules

Routing Rules

Rule 1:
fx: `{{ $json.fileName.split('.').pop().toLowerCase() }}` A is equal to
md

Rename Output ☐

Rule 2:
fx: `{{ $json.fileName.split('.').pop().toLowerCase() }}` A is equal to
java

Rename Output ☐

Rule 3:
fx: `{{ $json.fileName.split('.').pop().toLowerCase() }}` A is equal to
xml

FIGURE 5 – Switch Node

7.3 Node 1.4 : Extract from File1

Role : Extract textual content based on file type

Convert to File Execute step

Parameters Settings Docs

Operation
Convert to JSON

Mode
Each Item to Separate File

Put Output File in Field
data
The name of the output binary field to put the file in

Options

File Name
fx {{ \$json.fileName }}

Add option

FIGURE 7 – Convert to JSON Node

7.5 Node 1.6 : Qdrant Vector Store

Role : Store embeddings in the vector database

7.5.1 Method 1 : Manual Creation via Qdrant Console

Technical Configuration

- Configuration steps in Qdrant :
 - Create an API key
 - Log in to your Qdrant account
 - Go to the "API Keys" section and generate an API key

Aziza Neffati - Base Account Aziza Neffati

Clusters Create +

CLUSTER	CONFIGURATION	PROVIDER	STATUS	ACTIONS
FREE TIER test Upgrade to a Paid Cluster	1 NODE Disk: 4GiB RAM: 1GiB vCPU: 0.5	aws eu-central-1	HEALTHY	...

FIGURE 8 – API Key

- Collection object
 - Create a collection using the console and entering the code
 - Collection name : [define according to your project]
 - Dimensions : 1536 (Compatible with OpenAI model)
 - Metric : cosine

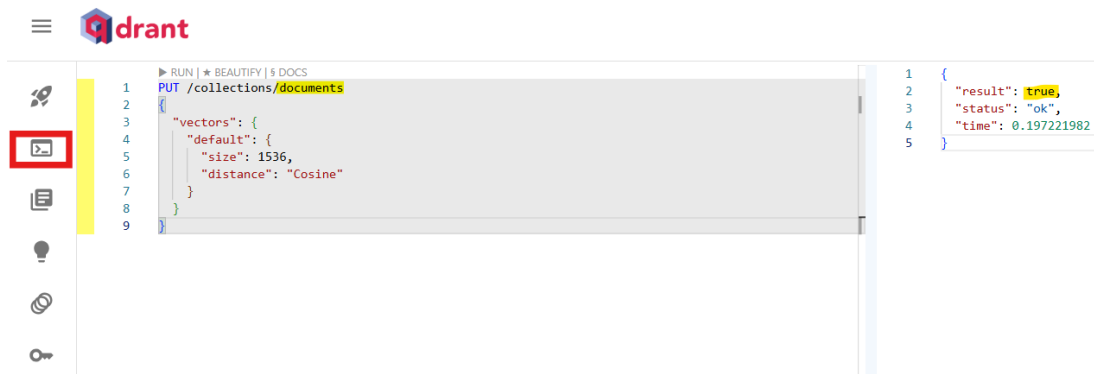


FIGURE 9 – Create Collection

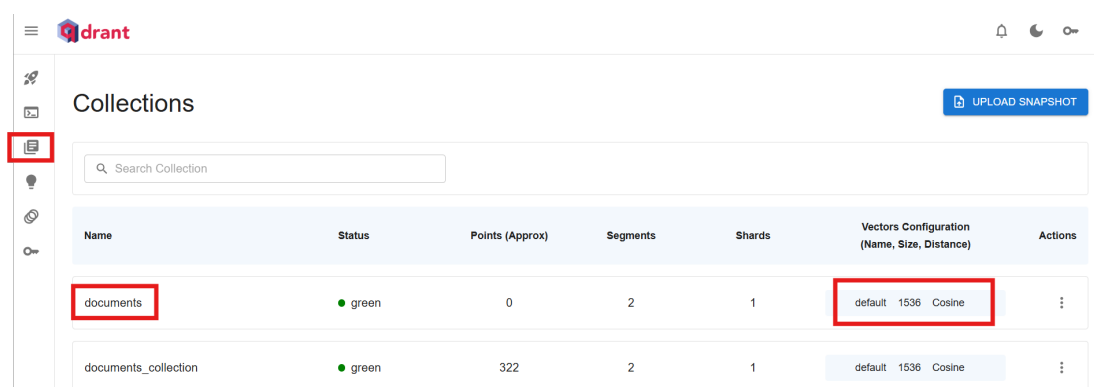


FIGURE 10 – List of Collections

3. Configuration in n8n

- In your n8n workflow :
- Add a **Qdrant Vector Store** node
- Node parameters :
 - API Key : paste the API key generated from Qdrant
 - Operation : Insert Document
 - Index : From List → select the name of your created Collection
 - Batch Size : 200 (or another suitable value based on your data size)
 - Metadata Fields :
 - `file_name` : `{{ $json.fileName }}`
 - `file_path` : `{{ $json.filePath }}`
 - `chunk_id` : `{{ $json.chunkId }}`
 - `created_at` : `{{ $now }}`

Once the workflow is executed, you will see the chunks extracted from your documents in the Qdrant collection.

Qdrant Vector Store Execute step

Parameters Settings Docs

Credential to connect with
QdrantApi account

Tip: Get a feel for vector stores in n8n with our [RAG starter template](#)

Operation Mode
Insert Documents

Qdrant Collection
From list documents_collection

Embedding Batch Size
200

Options
No properties
Add Option

FIGURE 11 – Qdrant Vector Store Node

7.5.2 Method 2 : Direct Creation of the Vector Store via n8n

Goal : Create a new Qdrant collection (vector store) directly from n8n, without using Qdrant's web interface.

Simplified Procedure

1. Prerequisite :
 - Have your Qdrant API key (see Method 1 for how to generate it)
2. In the **first Qdrant Vector Store node** of your workflow :
 - In the **Qdrant Collection** field, select **"By ID"**
 - Enter a name for your new collection (vector store)
 - Make sure the following parameters are correctly set :
 - **Operation** : Insert Document
 - **Batch Size** : according to your data (e.g., 200)
 - **Metadata Fields** :
 - `file_name : {{$json.fileName}}`
 - `file_path : {{$json.filePath}}`
 - `chunk_id : {{$json.chunkId}}`
 - `created_at : {{$now}}`
 - Click on **Execute Node**
3. Result :
 - The vector store will be automatically created in Qdrant from this first data submission
 - In subsequent nodes (e.g., an AI Agent node), you will be able to select this vector store from the dropdown menu

Qdrant Vector Store Execute step

Parameters Settings Docs

Credential to connect with
QdrantApi account

Tip: Get a feel for vector stores in n8n with our [RAG starter template](#)

Operation Mode
Insert Documents

Qdrant Collection Fixed Expression
By ID Enter ID...

Embedding Batch Size
200

Options
No properties
Add Option

FIGURE 12 – Creating a Vector Store via n8n (By ID option)

7.6 Node 1.7 : Default Data Loader

Role : Segment documents into overlapping chunks to optimize their processing within the workflow chain

7.6.1 Configuration

- **Data Type :** JSON
- **Mode :** Load Specific Data
- **Data Source :** `{ $('Extract from File1').item.json.data }`
- **Text Splitting :** Custom

7.6.2 Metadata

This node configures the metadata associated with the loaded data :

- **Name :** source
- **Value :** `{ $('Extract from File1').item.json.fileName }`

Default Data Loader

Parameters Settings Docs

This will load data from a previous step in the workflow. **Example**

Type of Data
JSON

Mode
Load Specific Data

Data
fx `{{ $('Extract from File1').item.json.data }}`

Guidelines ## 1. [Utilisation de GIT](gitGuidelines/README.md) ## 2. [Sécurité logicielle](securityGuidelines/R...)

Text Splitting
Custom

Options

Metadata

Name
source

Value
fx `{{ $('Extract from File1').item.json.fileName }}`

README.md

FIGURE 13 – Default Data Loader Node

7.7 Token Splitter

Role : The **Token Splitter** node is an essential preprocessing component that divides long documents into smaller, manageable segments. This segmentation is crucial for optimizing the processing of textual data in natural language processing pipelines.

7.7.1 Configuration Parameters

The Token Splitter has two main configurable parameters :

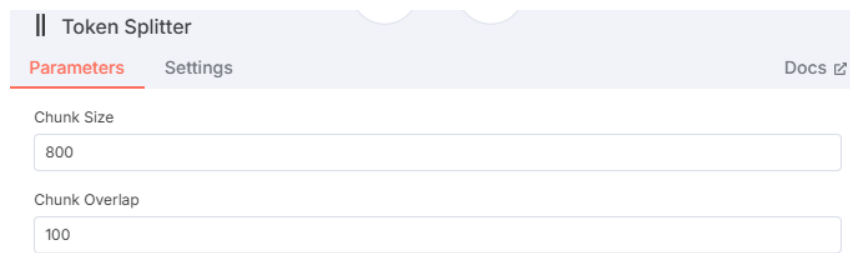
- **Chunk Size :** Set to 800 tokens, this value determines the maximum size of each generated segment. This configuration helps maintain a balance between contextual coherence and processing limitations.
- **Chunk Overlap :** Set to 100 tokens, this parameter defines the number of tokens that overlap between consecutive segments. This overlap ensures context preservation at segment boundaries and avoids losing important information.

7.7.2 Functionality

The segmentation process follows these steps :

1. The input text is tokenized
2. Segments of up to 800 tokens are created
3. Each new segment starts 700 tokens after the beginning of the previous one (800 - 100 overlap)

4. This approach ensures contextual continuity between segments



The image shows a user interface for a 'Token Splitter' node. At the top, there is a header bar with the title 'Token Splitter' and two tabs: 'Parameters' (which is selected and highlighted with a red underline) and 'Settings'. To the right of the tabs is a 'Docs' link with an external icon. Below the tabs, there are two input fields. The first is labeled 'Chunk Size' and contains the value '800'. The second is labeled 'Chunk Overlap' and contains the value '100'.

Token Splitter	
Parameters	Settings
Chunk Size	800
Chunk Overlap	100

FIGURE 14 – Token Splitter Node

8 Part 2 : Chat and Responses

8.1 Architecture

Chat Message → AI Agent → OpenAI Chat Model → Simple Memory → Qdrant Vector Store → OpenAI Embeddings

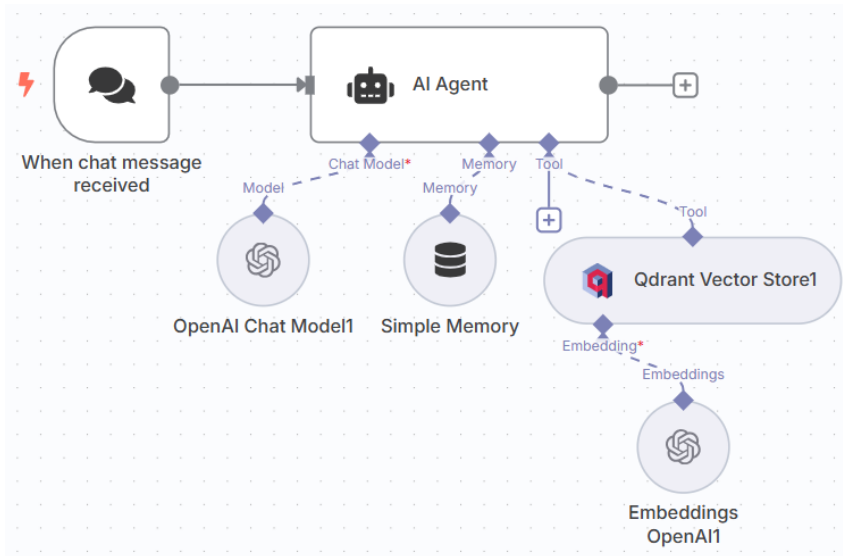


FIGURE 15 – Conversational Interface

Nodes and Functions :

8.2 Node 2.1 : When chat message received

Role : Starting point of the workflow

Function : Triggers the flow each time a user sends a message in the chat.

8.3 Node 2.2 : AI Agent

Role : Central intelligent agent

Function : Handles the processing of incoming messages.

It uses :

- A language model to understand and respond
- A memory to retain the context of the conversation
- An external tool (Qdrant) to enrich responses with vector data

It combines :

- Chat Model → to understand the query
- Memory → to retain chat context
- Tool → to interact with vector data (Qdrant)

AI Agent Execute step Docs

Parameters Settings

Tip: Get a feel for agents with our quick [tutorial](#) or see an [example](#) of how this node works

Source for Prompt (User Message)
Connected Chat Trigger Node

Prompt (User Message)

```
fx {{ $json.chatInput }}
```

 bonjour parle moi du process du commit

Require Specific Output Format
☐

Options

System Message
 You are an AI assistant that answers questions using a knowledge base stored in Qdrant. You must ALWAYS cite your sources and provide cross-references between related documents.
 Enhanced Response Requirements:
 Multi-document Analysis: For complex questions, search across ALL relevant documents
 Cross-referencing: When citing information, link related concepts from different documents

Add Option

FIGURE 16 – AI Agent Node

System message :

You are an AI assistant that answers questions using a knowledge base stored in Qdrant. You must ALWAYS cite your sources and provide cross-references between related documents.

Enhanced Response Requirements :

- Multi-document Analysis : For complex questions, search across ALL relevant documents
- Cross-referencing : When citing information, link related concepts from different documents
- Process Flow : For process questions, show the complete flow across documents
- Documentation Integration : Connect different processes with their associated standards and guidelines

Mandatory Response Format :

Complete Answer [Provide comprehensive answer with cross-document insights]

Cross-Document Citations Process Flow :

Step 1 : [Document A] - [specific content]

Step 2 : [Document B] - [specific content]

Step 3 : [Document C] - [specific content]

Standards Referenced [List applicable standards from your organization's standard documents]

Source Metadata Primary Sources :

File : [file_name] | Chunk : [chunk_id] | Score : [score]

Supporting Sources :

File : [file_name] | Chunk : [chunk_id] | Score : [score]

Special Instructions for Process Questions :

- Always check process documentation, review procedures, and standard guidelines
- Show the relationship between different workflow steps and quality requirements
- Provide specific examples from the standards when relevant
- Cross-reference related procedures and dependencies

8.4 2.3. OpenAI Chat Model1

Role : Language model

Function : Provides the ability to understand and generate text.

Specificity : Uses a GPT-4o mini model via OpenAI.

Linked to :

- AI Agent via Chat Model

2.4. Simple Memory

Role : Conversational memory

Function : Stores the history of interactions to provide contextual responses.

Example : If the user refers to something mentioned earlier, the agent can retrieve it.

Linked to :

- AI Agent via Memory

8.5 2.5. Qdrant Vector Store1

Role : Vector database

Function : Allows fast storage and retrieval of embeddings (vectors)

Used for RAG (Retrieval-Augmented Generation) :

The agent retrieves the most relevant documents and uses them to formulate a better response.

Linked to :

- AI Agent via Tool
- OpenAI Embeddings (source of vectors)

Qdrant Vector Store1 Execute step

Parameters Settings Docs

Credential to connect with
QdrantApi account

Tip: Get a feel for vector stores in n8n with our [RAG starter template](#)

Operation Mode
Retrieve Documents (As Tool for AI Agent)

Description
Recherche dans la base de documents pour répondre aux questions

Qdrant Collection
From list documents_collection

Limit
4

Include Metadata
☒

Rerank Results
☐

Options
No properties
Add Option

FIGURE 17 – Qdrant Vector Store Node

Parameter	Value / Explanation
Credential to connect with	QdrantApi account (configured to connect to Qdrant Cloud or local instance)
Operation Mode	Retrieve Documents (As Tool for AI Agent) → this mode allows the agent to query the Qdrant database
Description	Search the document base to answer questions
Qdrant Collection	documents_collection (name of the embeddings collection in Qdrant)
Limit	4 → limits to 4 relevant documents returned per query
Include Metadata	Enabled → metadata (file name, chunk ID, etc.) are included in the results
Rerank Results	Disabled → results are not re-ranked by secondary relevance
Options	No custom parameters here

Table 1 : Configuration of the Qdrant Vector Store Node

9 Conclusion

This guide allows you to set up a local RAG agent using n8n and Qdrant, to intelligently index, search, and query your documents. This architecture is ideal for any AI-augmented document knowledge base project.