RAG workflow: AI understands your data

Instead of hoping the LLM knows the answer, you provide the RAG with your factual documents and let it find the relevant information before responding.

Your agent has the tools (lesson 4) and memory (lesson 5) . But when a customer asks, "What is your refund policy?", the virtual assistant has to guess – because it has never read your documentation. RAG changes that. Instead of hoping the LLM knows the answer, you provide RAG with your actual documentation and let it find the relevant information before responding.

What is RAG?

Retrieval-Augmented Generation is a three-step process:

1. **Embedding** – Convert your document into numerical vectors (embeddings) and store them in a vector database.
2. **Retrieval** – When a question arises, find the most relevant document segments by comparing the question's embeddings with the archived embeddings.
3. **Generate** – Provide the retrieved data + questions to the LLM, and the LLM will generate answers based on your actual data.

Result : An AI answers questions from your documents instead of making things up. The LLM's answers are based on your data – and it can cite sources.

```
Câu h?i c?a ng??i dùng ? Câu h?i nhúng ? Tìm ki?m trong kho l?u tr? vector
? 3 nhóm t? khóa hàng ??u ? LLM: "D?a trên các nhóm t? khóa này, ?
ây là câu tr? l?i."
```

Why use RAG instead of putting the entire document into the prompt?

Due to token limitations, GPT-40 has a 128K context window—quite large, but the documentation of a medium-sized company easily exceeds that number. RAG allows you to search millions of documents and only provides relevant sections for LLM.

? **Quick check** : A company has 500 pages of product documentation. Why can't they just paste the whole thing into prompt LLM?

Answer : Token limitations and cost. Even a 128K token model can't handle 500 pages of text. And even if they could, processing all that context for each question would be extremely expensive and slow. RAGs only retrieve

the 3-5 most relevant paragraphs, keeping costs low and response times fast.

RAG process in n8n

The n8n RAG process uses 4 types of nodes:

1. **Document Loaders** - Import your source documents - PDF Loader, Google Drive Loader, Notion Loader, Web Scraper - Convert documents into text that can be split and embedded

2. **Text Splitter** - Splits documents into paragraphs - Recursive Character Text Splitter (default, works well with most text) - Token Text Splitter (splits by the number of tokens - more accurate for LLM input) - Configure paragraph size (usually 500 - 1000 tokens) and overlap (10 - 20%)

3. **Embedding** - Converting text segments to vectors - OpenAI Embeddings (`text-embedding-3-small` cheap and efficient) - Cohere, HuggingFace, or local models via Ollama

4. **Vector repository** - Store and search embeddings

1. **Supabase** (pgvector) - has a free, robust plan, and can query SQL.
2. **Pinecone** - managed service, high performance, free plan
3. **Qdrant** - open source, self-hosted, excellent for privacy.
4. **In-Memory** - for testing purposes only (will reset upon restart)

Building: RAG Knowledge Base Bot

You will build a bot that answers questions from a set of documents. We will use Supabase as the vector repository because it is free, sustainable, and integrates well with n8n.

Part A: Document Import Procedure

This process embeds your document into a vector archive. You run it once (or whenever the document changes).

Step 1: Create a data entry process

1. New procedure ? Add Manual Trigger
2. Add a Document Loader - in this example, use **the Default Data Loader** and paste some test text, or use the PDF Loader if you have a PDF file to upload.
3. Add Recursive Character Text Splitter:
    1. Block size: 800 characters
    2. Block overlap: 100 characters
4. **Add the Supabase Vector Store** node in **Insert** mode :
    1. Connect to your Supabase instance (create a free project at supabase.com)
    2. Table: `documents`(You need to enable the pgvector extension and create this table - Supabase has a one-click setup for this)
5. **Add the OpenAI Embeddings** child node to the vector repository:
    1. Model:`text-embedding-3-small`
    2. This converts each block of text into a 1536-dimensional vector.

Connect them: Trigger ? Loader ? Splitter ? Vector repository (with attached embeddings)

Step 2: Run the data collection process

Click on "Test workflow". Observe the process of segmenting, embedding, and saving your document. Check your Supabase dashboard - you will see rows in the document table, each containing a text segment and its corresponding vector embed.

Part B: Workflow for Answering Questions

This workflow receives user questions and retrieves answers from your stored documents.

Step 1: Create a Q&A workflow

1. New workflow ? Add Chat Trigger
2. **Add a Q&A Chain** node (not an AI Agent - the Q&A Chain is optimized for document retrieval).
3. Attach the **OpenAI Chat Model** child node (gpt-4o-mini is suitable for Q&A).
4. Attach **the Supabase Vector Store child node in Retrieve** mode :
    1. Connect to the same Supabase version and board.
    2. Top K: 4 (retrieving the 4 most relevant text snippets)
    3. Attach your **OpenAI Embeddings** (the same model you used to collect the data - this is crucial).

Step 2: Inspection

Click on "Test workflow". Ask questions about your document:

1. "What is the refund policy?"
2. "How do I reset my password?"
3. "What features are included in the Pro package?"

Q&A Chain will search through the vector archive, retrieve the most relevant information fragments, and generate answers based on your actual documentation.

? **Quick check** : You've imported the document using an `text-embedding-3-small`OpenAI model. For the retrieval process, could you use a different embedded model?

Answer : No. You must use the same embedded model for both import and retrieval. Different models produce different vector representations – searching with mismatched embedded models will return irrelevant results. This is a common error when switching models mid-project.

Segment: Life-or-death decisions

Your segmentation strategy determines the quality of your RAG more than your model choice. Poor quality segments = poor quality answers.

Segment size	Advantage	Disadvantages	Suitable for

Small (200 - 400 tokens)	Accurate retrieval	The important context can be broken down into smaller parts.	Knowledge base in question-and-answer format
Average (500 - 1000 tokens)	Accuracy/Context Balanced	Standard trade-offs	Most documents
Large (1000 - 2000 tokens)	The entire context is preserved.	May include irrelevant text	Long article, report

Overlapping is also crucial. Without overlapping, paragraph boundaries can split sentences—and important information may not be present in both paragraphs. 10-20% overlapping ensures that sentences at the paragraph boundaries appear in both adjacent paragraphs.

General principle : Start with 800-token paragraphs and 100-token overlap. Test with real-world questions. If the answer lacks relevant context, increase the paragraph size. If the answer contains too much irrelevant text, decrease the paragraph size.

RAG, fine-tuning, long context

Three approaches to imparting knowledge to an LLM:

approach	When should it be used?	The trade-off
RAG	Dynamic knowledge, constantly changing (documents, policies, product information)	Dependence on retrieval quality — poor quality input data will lead to poor quality output.
Fine-tuning	Teaching a model of a specific style or pattern.	Expensive, slow updates, requires training data.
Long context	Compact, fixed knowledge base (High cost per query, cannot scale beyond contextual limits).	

For most business processes, RAG is the right choice. Your documentation changes, your policies update, your products evolve—and you don't want to have to retrain the model every time.

Advanced: RAG with AI Agent

Q&A Chain is great for simple document retrieval. But what if you want an agent that can both search your documents and utilize other tools (web search, code execution)?

Connect the vector store as **a Vector Store Tool** to the AI ??Agent node instead of using a Q&A Chain. The agent can then decide when to check your documents and when to search the web – combining RAG with the tool usage patterns from Lesson 4.

Update system prompt:

```
B?n có quy?n truy c?p vào: - Kho ki?n ??th?c n?i b? (vector store): S? d?
ng cho các chính sách n?i b?, tài li?u s?n ph?m, quy trình - Tìm ki?
m trên web: S? d?ng cho thông tin bên ngoài, tiêu chu?n ngành, d? li?u ??
```

i th? c?nh tranh Luôn ki?m tra kho ki?n ??th?c TR??C tiên ??i v?i các câu h
?i n?i b?. Ch? s? d?ng tìm ki?m trên web n?u kho ki?n ??th?
c không có câu tr? l?i.

Key points to remember

1. RAG allows your AI to answer questions directly from your documents—not from LLM training data.
2. The process includes: Data input (load ? split ? embed ? store) followed by querying (query ? embed ? search ? generate).
3. Splitting is the most impactful decision – start with 800 tokens and 100 duplicate tokens, then refine.
4. Use the same embedded model for both input and output – mismatched models will produce poor results.
5. Supabase (pgvector) is a good free option for vector storage; Pinecone and Qdrant are alternatives.
6. Combine RAG with AI Agent to build your document search assistants and utilize external tools.

1. Question 1:

   You need to build an FAQ bot that answers questions from 200 PDF documents. Which n8n vector archive would you choose?

   1. A. In-memory vector storage - this is the simplest way.
   2. B. Supabase vector archive - robust, supports pgvector, handles large document collections.
   3. C. You don't need a vector archive - just transfer all 200 PDF files directly to LLM.

   EXPLAIN:

   200 PDF files is too much data to store in memory (in-memory storage will be reset on reboot and cannot handle large collections). Directly transferring all PDF files to the LLM would exceed the context window of any model. Supabase with pgvector provides you with persistent storage, similarity searching, and scalability to millions of embeddings. Pinecone and Qdrant are also good options.

2. Question 2:

   Your RAG bot is returning incorrect answers even though the correct information is in your documentation. What is the most likely cause?

   1. A. The LLM model is not strong enough.
   2. B. Inappropriate block size - either too large (irrelevant context dilutes the answer) or too small (important context is fragmented into many blocks)
   3. C. You need to add more documents to the vector archive.

   EXPLAIN:

   Block splitting is the most common error point in RAGs. Blocks that are too large include irrelevant text, confusing the LLM. Blocks that are too small fragment crucial context—'the answer to question A' is in one block and 'supporting data' is in another. Start with blocks of 500-1000 tokens with 10-20% overlap between blocks, then adjust based on your results.

3. Question 3:

   What problem does RAG solve that tweaking doesn't?

   1. A. RAG is faster than fine-tuning.
   2. B.RAG allows you to add or update knowledge without retraining the model—simply update the documentation in your vector repository.
   3. C. RAG produces higher quality feedback than fine-tuning in all cases.

   EXPLAIN:

   Fine-tuning integrates knowledge into the model's weights—updating it requires retraining, which is costly and time-consuming. RAG stores knowledge in an external repository that you can update at any time. Change your product documentation? Just re-embed it. No retraining required. The trade-off: RAG relies on retraceability, while fine-tuning internalizes patterns. For most business use cases, RAG wins in terms of flexibility.

Submit your work

Training results

You have completed **0** questions.

-- / --

Review the lesson