Building a RAG Agent with n8n: Create Your Own Work Assistant with Supabase and AI

Table of Contents

Introduction: Let AI Handle Repetitive Document Searches and Answers

Anyone working in a corporate environment has likely experienced the struggle of finding specific information within a vast sea of internal documents. Hundreds of pages of employment rules, constantly updated work manuals, complex customer service guidelines—the information you need is surely there, but finding it is never simple. Wasting time by repeatedly asking a manager or combing through entire documents is the epitome of inefficiency. (Video 0:01)

The technology that has emerged to solve this real-world problem is **RAG (Retrieval-Augmented Generation)**. RAG is an innovative AI architecture that allows an AI to learn and understand specific documents you provide, generating accurate and reliable answers based on that content. It's like asking a subject-matter expert questions about your company's documents and getting immediate answers.

This guide provides a detailed walkthrough of the entire process of building your own RAG Agent in just 15 minutes, using the low-code automation tool **n8n** and the open-source database **Supabase**, making it easy for even those unfamiliar with coding to follow along. By the end of this article, you will have a powerful 'AI Work Assistant' that frees you from getting lost in document piles and delegates repetitive information retrieval tasks to AI, allowing you to focus on core responsibilities. This is the first step toward maximizing not only your personal work efficiency but also the productivity of your entire team.

1. What is RAG and How Does It Work? (Understanding Core Concepts)

Before we dive into building a RAG Agent, it's crucial to understand the core concepts and working principles of the underlying RAG technology. This section explains what RAG is and how it generates answers to questions, tailored for beginners with a visual flow.

Definition of RAG (Retrieval-Augmented Generation)

RAG is an acronym for 'Retrieval-Augmented Generation,' and its name encapsulates its core principle. (Video 0:36) It is a technical approach where a Large Language Model (LLM) **retrieves** relevant information from an external, authoritative knowledge base to **augment**, or supplement, its content before **generating** a response.

Conventional LLMs are pre-trained on vast amounts of data but lack knowledge of information created after their training date or internal data specific to an organization (e.g., company financial reports, legal documents). This can lead to a phenomenon known as 'hallucination,' where the model fabricates inaccurate or non-existent information as if it were fact. RAG complements this limitation of LLMs.

Instead of relying on guesswork, it guides the LLM to generate answers based on a 'foundation of fact'—the specific documents we provide—dramatically improving the accuracy and reliability of its responses.

How RAG Works (A 2-Step Process)

The operation of RAG can be broadly divided into two stages: 'Data Preparation and Indexing' and 'Retrieval and Response Generation.' This can be compared to a library's process of organizing books and creating an index, and then using that index to find the right book to answer a reader's question.

1. Data Preparation and Indexing (Building the Document Library)

This stage lays the foundation of knowledge for the AI to reference and is typically performed once or periodically offline.

- **Document Loading and Splitting (Chunking):** First, we load the documents (PDF, DOCX, TXT, etc.) that the AI will learn from. Since documents can be very long, they need to be broken down into smaller units that the LLM can process at once. This is called 'chunking.' (Video 0:59) The reason for chunking is to stay within the LLM's context length limit (the amount of text it can handle at one time) and to efficiently retrieve only the most relevant parts for a given question. Intelligently splitting by meaningful units like paragraphs or sentences, rather than just by character count, is more beneficial for the quality of the answers.
- **Embedding:** Computers do not understand human language directly. Therefore, the text chunks must be converted into a format that computers can understand and compute: an array of numbers called a 'vector.' This process is called 'embedding.' (Video 1:08) Through embedding, each piece of text gets a unique numerical coordinate, and semantically similar texts are positioned close to each other in this vector space. This vector conversion is essential for later mathematically calculating the relevance between a user's question and the document content.
- **Storing in a Vector Database:** The document chunks, now converted into vectors via embedding, are stored in a special database for easy retrieval. This is called a 'Vector Database.' (Video 1:42) This database is optimized to quickly and efficiently find the most similar vectors to a given vector from among countless others.

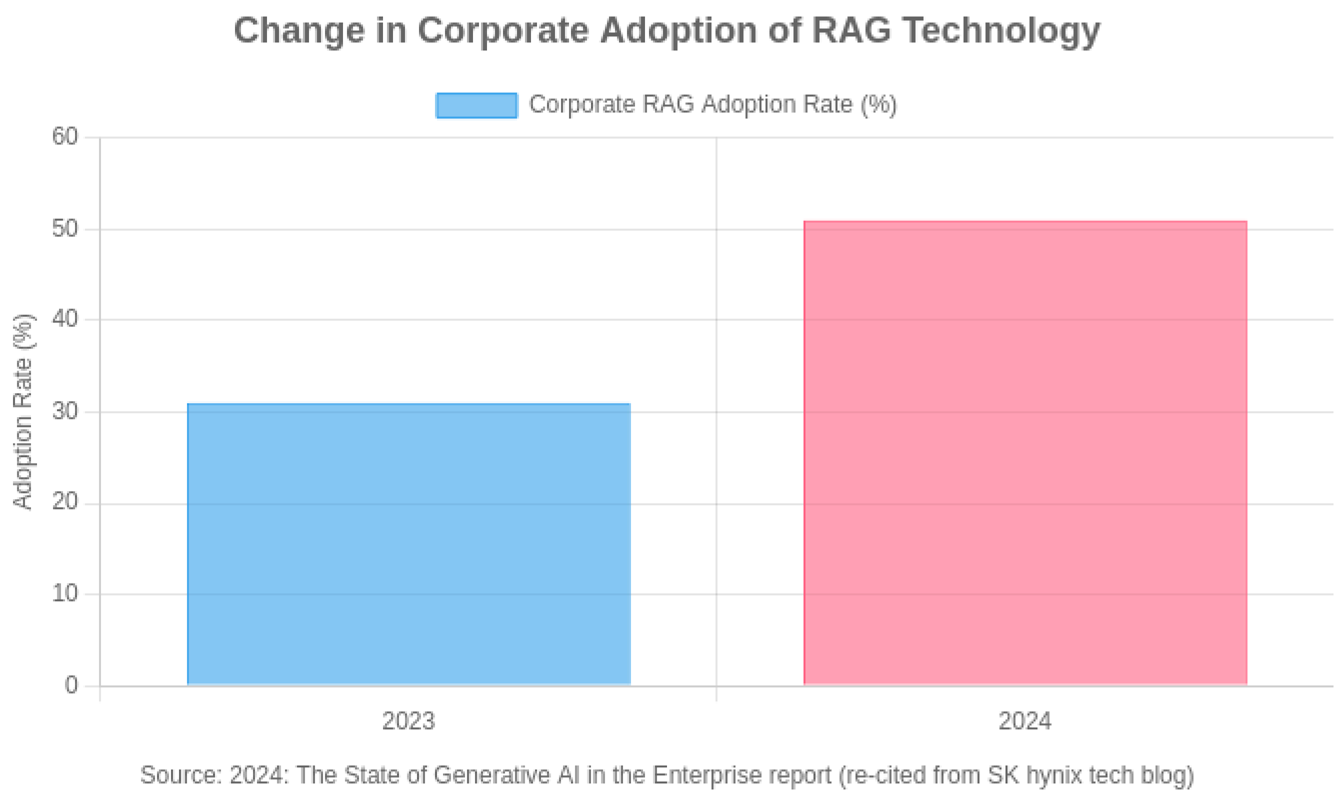
2. Retrieval and Response Generation (Real-time Q&A)

This is the process that occurs in real-time when a user actually asks a question.

- **Question Embedding:** The user's input question is also converted into a vector using the **same embedding model** that was used to transform the document data. (Video 1:55) Using the same model is crucial for comparing meaning within the same vector space.
- **Similarity Search:** The system takes the question vector and compares it with the numerous document vectors stored in the vector database. It then finds a few document chunks that are semantically most similar—that is, closest in distance within the vector space. This is called 'similarity search' or 'semantic search.'
- **Prompt Augmentation and Response Generation:** Finally, the retrieved, highly relevant document chunks are combined with the user's original question and passed to the LLM. (Video 2:16) The prompt given to the LLM at this point takes the form of "Answer this [question] by referring to the following [document content]." Now, the LLM generates an accurate, context-aware answer based on clear source material, rather than relying on vague knowledge.

Why Use RAG?

The RAG architecture offers several clear advantages to businesses and developers, making it one of the most prominent technologies in the Generative AI field today. In fact, according to one report, the corporate adoption rate of RAG surged from 31% in 2023 to 51% in 2024, establishing it as a dominant AI design structure.



Key Advantages of RAG

- **Improved Reliability and Accuracy of Answers:** By forcing the LLM to reference an external, verified knowledge source, the phenomenon of 'hallucination'—generating information not based on fact—can be significantly reduced.
- **Utilization of Up-to-Date and Private Data:** Retraining an LLM requires enormous cost and time. With RAG, there's no need to retrain the model; you can simply update the knowledge base with the latest information or sensitive internal corporate data, and the LLM can use it immediately. (Video 2:23)
- **Cost and Efficiency:** Passing entire large documents to an LLM can exceed context length limits or drastically increase API costs. RAG selects and sends only the most relevant parts related to the question, allowing for faster and more cost-effective retrieval of accurate answers. (Video 2:37)
- **Source Traceability:** RAG can cite the sources it used to generate an answer. This increases transparency by allowing users to verify the reliability of the response themselves.

2. Hands-On! Building a RAG Agent with n8n and Supabase (Step-by-Step Guide)

Now that we understand the concept of RAG, it's time to build our own AI work assistant. This section provides a detailed, step-by-step guide to building a RAG Agent using the low-code automation tool n8n and the open-source database Supabase. Each step is explained in detail so that even those with little to no coding experience can easily follow along.

Prerequisites: Setting Up a Supabase Vector Database

The first step in a RAG system is to prepare the 'library' for storing and retrieving document data—the vector database. We will use **Supabase** for this purpose.

About Supabase

Supabase is a service known as an open-source Firebase alternative, providing various backend features necessary for development. A major advantage is its user-friendly interface, which is easy for beginners to handle, and its flexibility to use both a relational database (PostgreSQL) and a vector database (pgvector extension) in an integrated manner. (Video 3:10) It is highly suitable for this exercise as you can build and test a RAG system sufficiently with its free plan.

Creating a Project

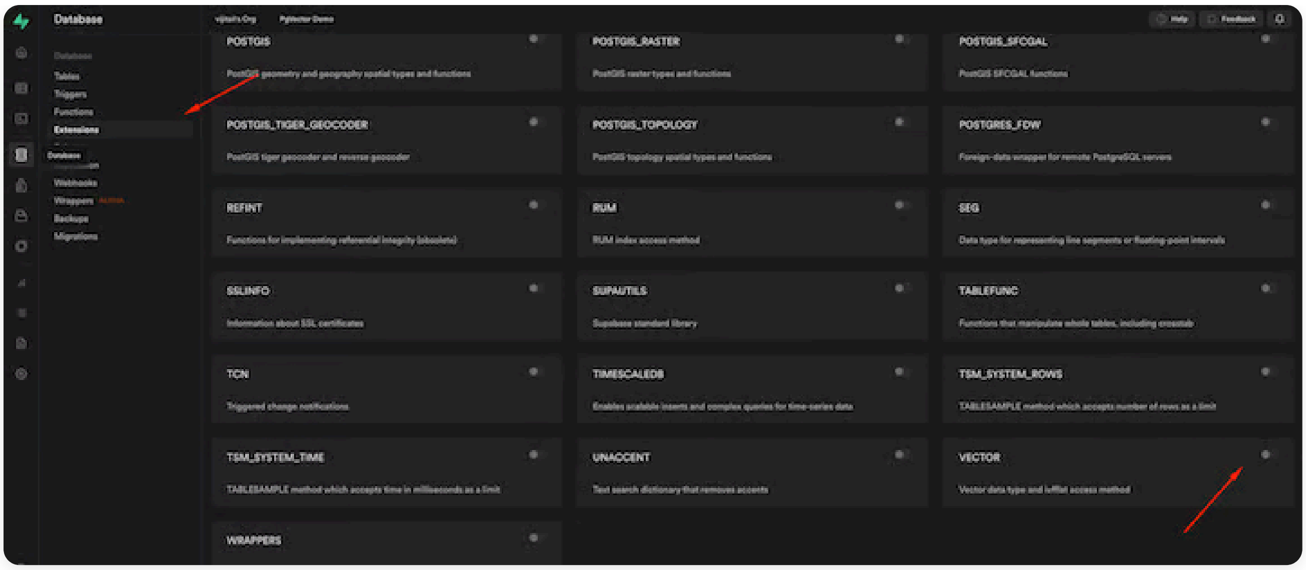
First, you need to sign up for Supabase and create a new project. The process is very straightforward.

1. Go to the Supabase official website and log in.
2. On the dashboard, click the 'New Project' button.
3. Set a project name (e.g., `rag-agent`) and a database password. This password will be needed later to connect with n8n, so be sure to save it in a secure place. (Video 3:45)
4. It's a good idea to choose a region that is physically close, such as 'Asia Pacific (Seoul)'.
5. Click 'Create new project,' and the project will be created within a few minutes.

Creating the Vector Store Table

Just because the project is created doesn't mean you can store documents right away. You need to create the 'bookshelf'—the table—to store the document vectors. This can be done simply by copying and running an SQL script.

1. Copy the SQL query provided in the n8n official documentation template. This is a standard script used to set up a vector store with Supabase and LangChain. (Video 7:31)
2. In your newly created Supabase project dashboard, navigate to the 'SQL Editor' in the left menu.
3. Click 'New query' and paste the copied SQL code. (Video 7:49)
4. Press the 'RUN' button to execute the query. If you see the message 'Success. No rows returned,' the table has been created successfully.



POSTGIS
PostGIS geometry and geography spatial types and functions

POSTGIS_RASTER
PostGIS raster types and functions

POSTGIS_SFCGAL
PostGIS SFCGAL functions

POSTGIS_TIGER_GEOCODER
PostGIS tiger geocoder and reverse geocoder

POSTGIS_TOPOLOGY
PostGIS topology spatial types and functions

POSTGRES_FDW
Foreign-data wrapper for remote PostgreSQL servers

REFINT
Functions for implementing referential integrity (obsolete)

RUM
RUM index access method

SEG
Data type for representing line segments or floating-point intervals

SSLINFO
Information about SSL certificates

SUPAUTILS
Supabase standard library

TABLEFUNC
Functions that manipulate whole tables, including crosstab

TCN
Triggered change notifications

TIMESCALEDB
Enables scalable inserts and complex queries for time-series data

TSM_SYSTEM_ROWS
TABLESAMPLE method which accepts number of rows as a limit

TSM_SYSTEM_TIME
TABLESAMPLE method which accepts time in milliseconds as a limit

UNACCENT
Text search dictionary that removes accents

VECTOR
Vector data type and ivfflat access method

WRAPPERS

Screen showing the activation of database extensions like pgvector in the Supabase dashboard

This SQL code performs three important tasks:

- `create extension if not exists vector;` : Activates the **pgvector** extension in Supabase's PostgreSQL database, which enables vector operations. (Video 7:56)
- `create table documents (...);` : Creates a table named 'documents'. This table defines columns to store the original content of the document (`content`), metadata (`metadata`), and most importantly, the vector data (`embedding`).
- `vector(1536)` : Specifies the dimension of the column that will store the embedding vector as 1536. This number must match the dimension of the vectors generated by the OpenAI `text-embedding-3-small` model we will be using. Different models have different dimension counts, so it's important to set this value according to the embedding model you plan to use. (Video 8:14)
- `create function match_documents (...);` : Defines a function to perform similarity searches. This function is responsible for finding the documents in the database that are most similar to the user's query vector.

Now, if you go to the 'Table Editor' in the left menu, you can confirm that the 'documents' table has been successfully created. (Video 9:01) With this, our knowledge repository for the AI is ready.

Step 1: Workflow for Automated Document Upload and Embedding

Now that the database is ready, let's build an n8n workflow to automatically save documents to it. The goal is to have n8n detect when a document is uploaded to a specific folder in Google Drive, automatically read its content, and then chunk, embed, and store it in Supabase.

n8n Node Configuration

Create a new workflow in n8n and connect the following nodes in order.

1. **Google Drive Trigger (Detect File Creation):**
   - Add a new node, search for 'Google Drive,' and select 'Google Drive Trigger.'
   - In the 'Event' option, select 'File Created.' This will trigger the workflow whenever a new file is created in the folder. (Video 4:23)
   - Connect your Google account and specify the folder to monitor (e.g., `n8n-projects`). You can set a 'Poll Time' to check the folder periodically.

2. **Google Drive (Download File):**
   - Click the '+' on the preceding trigger node and add a 'Google Drive' node.
   - Set 'Resource' to 'File' and 'Operation' to 'Download.'
   - Drag and drop the file's ID value from the previous node (Trigger) into the 'File ID' field. This ensures that the exact file detected by the trigger is downloaded. (Video 6:09)

3. **Supabase Vector Store (Add/Insert Documents and Embed):**
   - This is the most crucial node. Click the '+' and add a 'Supabase Vector Store' node.
   - **Account Connection:** Under 'Credential,' select 'Create New.' Copy and paste the 'Project URL' (Host) and the 'service_role' secret key from the 'API Keys' section of your Supabase project's 'Settings' > 'API' menu. (Video 6:50)
   - **Operation Setup:** Select 'Add/Insert Documents' for 'Operation' and 'documents' for 'Table Name.'
   - **Text Splitting:** Enable the 'Text Splitting' option. It is highly recommended to select 'RecursiveCharacterTextSplitter' for the 'Splitter.' (Video 10:30) This splitter intelligently divides the text not just by character count but by preserving meaning, following the order of paragraph flow (newlines), sentences (periods), and words (spaces).
   - **Add Metadata:** You can improve search performance through the 'Metadata' option. For example, by entering `document_title` for 'Name' and mapping the file name from the previous node to 'Value,' you can implement advanced features later, such as restricting searches to a specific document title. (Video 11:02)
   - **Embedding Model Setup:** In the 'Embeddings' section, select 'Use OpenAI' and connect your OpenAI account. For 'Model Name,' choose `text-embedding-3-small`. This is the model that

generates 1536-dimensional vectors. (Video 11:48)

Execution and Verification

All settings are now complete. Upload a sample document (e.g., 'GushiCompany_EmploymentRules.docx') to the specified folder in Google Drive, then run the n8n workflow. After the execution is complete, refresh the 'documents' table in Supabase to check if the data has been entered correctly. (Video 12:26) You will see that the document content is stored as multiple rows in the table, with each row filled with `content` (the split text), `metadata` (document title, etc.), and `embedding` (a vector value of 1536 numbers). This automates the building of our document library.

Step 2: Building the AI Agent and Testing Q&A

Now it's time to create an AI Agent chatbot that uses the established knowledge base to answer user questions in real-time. This workflow will serve as the user's conversational interface.

n8n Node Configuration

You can create a new workflow or continue with the existing one.

1. **Chat Trigger (Start Chat):**
    - Add a 'Chat Trigger' node to start the workflow when a user sends a message. This node provides a test chat interface within n8n.
2. **AI Agent (Core Logic):**
    - Add an 'AI Agent' node. This node acts as the brain of our RAG system.
    - **Model Setup:** In the 'Model' section, select the LLM to be used for generating answers. For example, choose 'OpenAI Chat Model' and specify `gpt-4-turbo` or `gpt-4o` as the model. (Video 13:36)
    - **System Prompt Setup:** Write a 'System Prompt' that defines the agent's identity and rules of conduct. This is very important. For example, you can give clear instructions like: "You are an expert on the company's internal documents. You must answer based only on the provided document content. If the information is not in the document, honestly reply with 'The relevant information is not available in this document.' All responses must be in English." (Video 13:06)
    - **Tools Setup:** Register the 'tools' that the agent can use.
        - Click 'Add Tool' and select 'Supabase Vector Store.'

        - Set 'Operation' to 'Retrieve from Documents.'
        - In the 'Description,' clearly explain when the agent should use this tool. (e.g., "Use this to answer questions related to internal documents such as company employment rules, work manuals, and CS guides.") (Video 14:01)
        - Select 'documents' for 'Table Name.'
        - For the 'Embeddings' model, you **must select the same model (`text-embedding-3-small`)** used in Step 1 to store the documents. This is essential for comparing the question and documents on the same basis. (Video 14:38)
    - **Memory Setup:**
        - Set up 'Memory' to allow the agent to remember previous conversation content for more natural dialogue.
        - Select 'Postgres Chat Memory.' This memory can store conversation history in a database for permanent retention. (Video 15:11)
        - Under 'Credential,' create a new account using the database connection information from Supabase. This information can be found in your Supabase project's 'Settings' > 'Database' > 'Connection string.' (Video 15:28)
        - Specify a 'Table Name' (e.g., `chat_history`), and n8n will automatically create the table and save the conversation history.

Testing and Verifying Results

All settings are now complete. Click the 'Open Chat' button in the top right of n8n to open the chat interface and ask a real question.

> *"What are the provisions regarding concurrent employment at Gushi Company?" (Video 16:37)*

When you enter the question, you can see the AI Agent using the Supabase tool to search for document fragments related to 'concurrent employment' from the knowledge base and generating an accurate answer based on them, such as "Employees may not engage in other occupations without prior approval from the company..." (Video 16:49)

If the agent doesn't behave as expected, you can check the execution history in n8n's 'Executions' tab. By examining the logs of the AI Agent node, you can see in detail what search terms the agent used to query the documents and which document chunks it referenced to generate the answer, which is very useful for debugging. (Video 17:26) Through this process, we have successfully built and verified the operation of our own RAG Agent.

3. RAG Agent Use Cases and Expansion Ideas in a Corporate Setting

Beyond simply implementing the technology, let's explore how this RAG Agent can be used in a real business environment to create value through specific examples. The basic model we've built has powerful potential to be applied to various corporate scenarios.

Core Use Cases (Based on the Video Demo)

The two cases demonstrated in the video clearly show how effectively a RAG Agent can automate repetitive tasks in a company.

- **Automating Internal HR Inquiries:** Every company has numerous internal regulations, such as employment rules, welfare policies, vacation policies, and expense processing guidelines. When employees have questions, they must either contact the HR team or search through company documents. By training a RAG Agent on these regulation documents, employees can get instant and accurate answers to questions like "Can I use annual leave if I've been with the company for less than six months?" (Video 17:51) at any time, 24/7. This reduces the burden of repetitive inquiries on the HR team and increases employee satisfaction.
- **Improving Customer Support (CS) Efficiency:** The customer support department is a prime example of a field where it takes a lot of time for new employees to get up to speed. It's difficult to memorize complex product information, various situational response manuals, and refund policies. By training a RAG Agent on customer service manuals, even new employees can provide consistent and accurate answers to customers, just like veteran staff. (Video 18:48) For example, to a question like "How should I respond when a customer requests a refund due to a shipping delay?" (Video 19:45), the agent can immediately provide a step-by-step response procedure, which can significantly contribute to reducing errors and maintaining customer satisfaction.

Additional Business Use Case Ideas

Besides the cases above, a RAG Agent can be utilized in various departments within a company.

- **Sales and Marketing:** Extensive product technical data, proposal templates, competitor analysis reports, and past success stories can be fed into a RAG Agent. During a meeting with a client, a salesperson can get real-time information by asking questions like "What are the main differences between Product A and competitor's Product B?" or "Summarize the success stories in the C industry," allowing them to respond nimbly to customer inquiries.

- **Legal and Compliance:** Searching through tens of thousands of pages of legal documents, past contracts, privacy regulations, and internal control guidelines is a major part of a legal team's job. By using a RAG Agent, they can quickly find relevant clauses for complex queries like "What are the provisions related to data processing consent in the latest GDPR regulations?", drastically reducing the time spent on legal risk review.
- **R&D and Technology Planning:** In the research and development department, numerous technical papers, patents, and research reports must be reviewed to develop new technologies. By connecting a RAG Agent to a database of relevant academic materials, researchers can increase the efficiency of their data investigation and analysis by asking questions like "What are the latest research trends in improving battery efficiency using nanotechnology?", allowing them to spend more time on generating innovative ideas.
- **Manufacturing and Production:** By training the agent on complex equipment maintenance manuals, safety protocols, and process management documents, field workers can quickly find solutions when problems arise. An immediate answer to a question like "What is the procedure for error code 503 on the X-200 equipment?" can minimize production line downtime.

System Expansion Plans

The RAG Agent we've built is just the beginning. By adding a few features, it can be evolved into a much more powerful and user-friendly system.

- **Integration with Collaboration Tools:** While we used n8n's test chat window, it can be integrated with internal messengers that employees use daily, such as Slack, Microsoft Teams, or Telegram. (Video 20:34) This way, employees can naturally converse with the AI chatbot and get the information they need in a familiar environment without having to log into a separate system.
- **Handling Multiple Documents and Data Sources:** Currently, all document chunks are stored in a single table. As the system grows, tables can be separated by data source, or metadata can be used to dynamically control the search scope. For example, if a user specifies "Search in the HR regulations," the system can be made to search only within documents tagged with 'hr-documents' metadata, thereby increasing search accuracy and speed.
- **Evolution to Agentic RAG:** The current RAG performs a single search and response generation for a given question. It can be advanced to an 'agentic' approach, where a complex question is broken down into several sub-questions, and multiple tools are used sequentially to gather and synthesize

information to produce a final answer. This is called 'Agentic RAG' and enables solving more complex problems.

4. Limitations and Future Challenges

Although we were able to easily create a powerful RAG Agent using n8n and Supabase, this basic model is not a silver bullet that solves all problems. To operate it reliably in a production environment, it is necessary to clearly recognize the technical limitations of the basic RAG implemented and to consider future tasks to overcome them.

Clear Limitations of Basic RAG

As mentioned at the end of the video, the RAG Agent we created has several clear limitations. (Video 20:58)

- **Limitations of Keyword-Based Search:** RAG excels at searching for similarity based on the 'meaning' of sentences. However, it can be relatively weak for searches where exact keyword matching is important, such as for specific product codes like 'A-model-SN12345', model names, or proper nouns. It might miss important information by judging it as semantically irrelevant. To compensate for this, a 'Hybrid Search' strategy that combines traditional keyword search (Lexical Search) and semantic search is needed.
- **Inability to Summarize Entire Documents:** The working principle of RAG is to retrieve only the 'parts' of chunks most relevant to a question to generate an answer. Therefore, a request like "Summarize this entire 100-page report" cannot be properly executed. (Video 21:29) A full summary requires providing the entire document as context, which is different from the basic design purpose of RAG.
- **Inability to Perform Calculations and Analyze Structured Data:** RAG is primarily specialized in handling unstructured text data. It cannot perform tasks like fetching numerical data from a spreadsheet or database table to calculate sums, averages, or perform complex analysis. (Video 21:36) Such tasks are better suited for Text-to-SQL technology that queries a SQL database directly or an AI Agent equipped with code execution capabilities.

Additional Considerations

To successfully introduce and operate a RAG system in a real corporate environment, the following points must be considered in addition to technical implementation.

Seven Failure Points of Enterprise RAG Systems

A recent study pointed out seven common failure points when designing RAG systems in an enterprise environment. Some of the key failure factors are as follows:

1. **Missing Content:** This occurs when the information for the user's question does not exist in the knowledge base (documents) in the first place. In this case, RAG may not be able to generate an answer or may cause hallucinations.
2. **Missed Top Ranked:** This is when relevant information exists in the document, but the search algorithm fails to bring that document to the top rank. This can be caused by an inappropriate choice of embedding model or an inefficient chunking strategy.
3. **Not in Context:** This happens when the document is retrieved correctly, but the answer to the question is not included in the chunk passed to the LLM. For example, this can occur if the answer to the question is split across the boundaries of chunks.
4. **Incorrect Specificity:** This is when the retrieved information is too general or, conversely, too detailed, leading to an answer that does not match the user's question intent.

To reduce these failures, data quality management, advanced chunking strategies, and continuous system evaluation are essential.

- **The Importance of Data Quality (Garbage In, Garbage Out):** The performance of a RAG system depends entirely on the quality of the source documents it learns from. If you provide documents with inaccurate, outdated, or poorly structured content, the AI Agent will inevitably produce low-quality answers. Therefore, establishing a data cleaning and management process before building a RAG system is crucial.
- **Advancement of Chunking Strategy:** While `RecursiveCharacterTextSplitter` may be sufficient for simple text documents, a different approach is needed for documents containing complex tables, charts, and images. Tables should be split while maintaining the meaning of rows or columns, and advanced chunking strategies are required, such as converting images to text using a separate image captioning model.
- **Continuous Evaluation & Tuning (Evaluation & Monitoring):** A RAG system is not something you build once and are done with. Key performance indicators (KPIs) such as Retrieval Accuracy, Faithfulness of the answer, and user satisfaction must be continuously monitored. It is essential to

have a process of constantly improving the system by collecting user feedback, analyzing failure cases to improve prompts, modifying chunking strategies, or fine-tuning the embedding model.

- **Security and Access Control:** In a corporate environment, not all employees can access all documents. A RAG system must also have the functionality to control the scope of accessible documents based on the user's role and permissions. By using features like Supabase's RLS (Row-Level Security), you can filter search results for specific documents based on the user's authentication information, preventing the leakage of sensitive information.

Conclusion: Build Your First AI Work Assistant Today

So far, we have journeyed together from the concept of RAG technology to building an AI Agent ourselves using powerful and accessible tools like n8n and Supabase. Although the term RAG might have seemed complex and difficult at first, this guide has shown that anyone can create a powerful AI work assistant tailored to their needs by using a low-code platform.

The RAG Agent we created is more than just a technical demonstration; it holds the potential to fundamentally change the way an organization works. Answering endless questions about company regulations, explaining the same things over and over to new employees, wandering through vast materials to find specific information—by automating these simple, repetitive information retrieval tasks, we can focus on core tasks that create more creative and strategic value.

> *"RAG is one of the most effective ways to leverage an organization's hidden knowledge assets. It goes beyond simply increasing efficiency; it can be a catalyst for accelerating data-driven decision-making and strengthening collective intelligence."*

This guide is the starting point of your journey. Based on what you've learned today, try training it on documents relevant to your work environment. Whether it's customer service manuals, product development documents, or marketing materials, anything will do. How about creating your own RAG Agent and sharing the experience and new application ideas you gain in the process with your colleagues? Your small attempt will be a powerful first step in taking your team's and organization's productivity to the next level.

References

[1] 여러분, 회사에
https://static-us-img.skywork.ai/prod/analysis/2025-07-27/5363768134482727537/1949337399479050240_3f3849b46a6e4140ce3791baa59eb3b7.txt