Unit 2 - AI agents and files

Unit 2
AI agents and files

2.1 Unit Introduction

Welcome to the second unit of the course that teaches you how AI agents deal with information.

If you're wondering how your AI agent can process and work with files, you're in the right place.

In this unit you will learn how AI agents can:

- use knowledge files to access information
- work with files using tools

process files directly

Let's start!

Continue to 2.2: AI agents and files

2.2 AI agents and files

AI agents interact with files for different purposes when working on tasks.

When tasks require it, they use information stored in files, access files through tools to retrieve or manipulate data, and process files directly to extract information or generate new documents.

To accomplish these different objectives, agents use:

- **Knowledge files** for storing information
- **Tools** for retrieving and processing file content

- **Direct file processing within the agent** for parsing and creating files without additional modules

Let's explore each one!

Continue to 2.3: Knowledge files

2.3 Knowledge files

2.3.1 AI agents knowledge

The LLM provides the agent with everything it knows by default.

When the LLM is built, it goes through a **training process where it learns** from massive amounts of data. By processing all this information, the LLM learns to **understand** language, **recognize** patterns, **answer** questions, and **complete** various tasks.

This creates reference data that the agent uses to carry out tasks, answer questions, generate content, and solve problems.

However, this training happens during the LLM's initial development. **After training is complete, the LLM's knowledge becomes fixed**, it's stored into the model's internal structure. While it's technically possible to add new data through specialized processes like fine-tuning, this requires significant computing power and isn't available for all models. This means that, in most cases, your AI agent is **limited to what the LLM learned** during its initial training.

This means your AI agent can only use what its LLM learned during that training period and nothing beyond that cutoff date. The fixed training and training material creates some limitations for your agent.

Click each one to learn more.

📆 Outdated information

The LLM powering your AI agent doesn't know about **anything that happened after its cutoff date**. If you ask your agent about recent events, new products, or current news, **the underlying LLM**

won't have that information.

🤷 Can't perform tasks without the right information

The LLM only knows what it learned during training. If you ask your agent to complete tasks that require **information not in its training data**, like your company's internal policies, proprietary data, or specialized industry knowledge, **the agent can't access that information** on its own.

🫠 Hallucinations

When the LLM doesn't have the information needed to answer a question, it may generate **plausible-sounding but incorrect responses** instead of indicating uncertainty. The model prioritizes creating fluent, natural-sounding text. This can lead to **fabricated details that seem credible** but aren't accurate.

When using your AI agent, you'll often need it to **work with very specific information**, like your company's internal data, recent information, or specialized knowledge in your field. The AI agent **doesn't have access to this information** because it's too specific to have been included in the LLM's training data.

To make the agent useful for your particular needs and capable of performing the specific tasks you require, **you need to provide this information yourself**.

How do you do it in Make?

In Make you can use knowledge files to give your agent access to specific information.

Let's have a look at this.

Continue to 2.3.2: Knowledge files

Knowledge files are documents you give your agent access to so it can search through them and use the information to carry out tasks.

This is how it works in Make:

Step 1

You **upload your files to a knowledge base** shared across your Make Team.

Step 2

Make automatically **generates a description** of the file that helps your agent understand what each file contains.

Step 3

Make breaks the file down into smaller pieces

called **chunks** and stores them in a database that your agent can query.

Step 4

You specify **which files your agent can access**.

Step 5

When you run your agent, it uses these descriptions to **identify relevant files**, **searches them**, and **retrieves relevant chunks** that contain the information it needs to complete the task.

During the last step, when the AI agent retrieves the information from the knowledge files, it uses RAG.

Key Concept

RAG (Retrieval-Augmented Generation) is a technique that allows the AI agent's LLM to retrieve relevant information from external

knowledge documents and use it to carry out its tasks.

Each time your agent queries the knowledge files, RAG works in three steps:

Retrieval

Your agent searches for the chunks most relevant to your request, matching based on meaning rather than just keywords. This is called **semantic search**.

Augmentation

The AI agent adds the retrieved information to the original query as **additional context**, creating an augmented prompt (the original input plus information from the knowledge files).

Generation

The agent's LLM uses **both your query and the retrieved context** to understand your request, determine the appropriate actions, and complete the task.

Using knowledge files has the following benefits:

Click each one to learn more

🎁 Access to specific information

The AI agent has access to the specific information it needs to complete tasks, **beyond what the LLM learned during training**.

🪟 Efficient context usage

Knowledge files **retrieve only the relevant chunks** your agent needs, not entire documents, keeping context usage efficient and without overloading the context window.

🫠 Reduced hallucinations

The agent can complete tasks accurately **without making up information**.

🎮 Flexibility and control

You can easily **add**, **remove**, or **update** the information your AI agent has access to.

And some challenges as well:

🌊 Information overload

If you provide too many knowledge files, the agent **cannot retrieve all relevant information** effectively. The more documents you add, the harder it becomes for the agent to find the right chunks.

🪟 Context window limitations

LLMs can only process a limited amount of text at once. If you retrieve too much information, your agent has to cut some of it off, causing you to **lose important data** from the context.

♻️ Data freshness

The documents you use might need regular updates or they can **become outdated**.

To address these challenges and help your agent find the right information, there are practical steps you can take in Make to optimize your knowledge files.

Best practices to help your agent work more efficiently and accurately:

🍱 Organize by topic and usage pattern

Group related information into separate files based on how often the agent needs them together. This **helps the agent retrieve only relevant context** for each type of query.

For example, keep product specifications separate from troubleshooting guides, and customer policies separate from

✂️ Split files strategically

Break large documents into smaller, focused files rather than uploading one massive document. Make's system selects files based on their names and descriptions, then retrieves relevant chunks. Smaller, topic-specific files **make it easier for the agent to find exactly what it needs** without pulling in irrelevant content.

For example, instead of one company policies file, create

📝 Use clear, descriptive file names and descriptions

Give your files **names that clearly indicate their content**. Check the description Make generates, which explains what the file contains and when to use it. Modify it if needed. Your agent uses both to decide which files to search, so clear labeling **improves retrieval accuracy**.

For example, use Refund_Policy_2025 instead of Document1, with a description like Contains refund eligibility rules and processing

🤹 Adjust number of results

Make allows you to **control the maximum number of results** your agent retrieves. Experiment with this setting: too few results

might miss important information, while too many can overload the context window with unnecessary data.

Start conservative and increase only if the agent consistently

🧹 Keep files current and remove outdated versions

Regularly review and update your knowledge files. Remove old versions that might contradict current information, having both

can confuse the agent or cause it to retrieve the wrong policy. Use the **Knowledge app's update functionality** to refresh files rather than adding duplicate versions. You will learn how to do this in the next part.

For example, when you update your refund policy from 30 days to 60 days, remove the old version and keep only the updated file in your knowledge base. Having both versions confuses the agent and may cause it to provide incorrect information to

🕵️ Monitor what the agent retrieves

Pay attention to which files and chunks your agent retrieves during testing. If irrelevant information consistently appears,

consider restructuring your files or adjusting retrieval settings. The goal is retrieving only the information that directly addresses the query.

For example, if your agent consistently pulls shipping information when customers ask about refunds, the file names or descriptions may be too similar, restructure them to be more

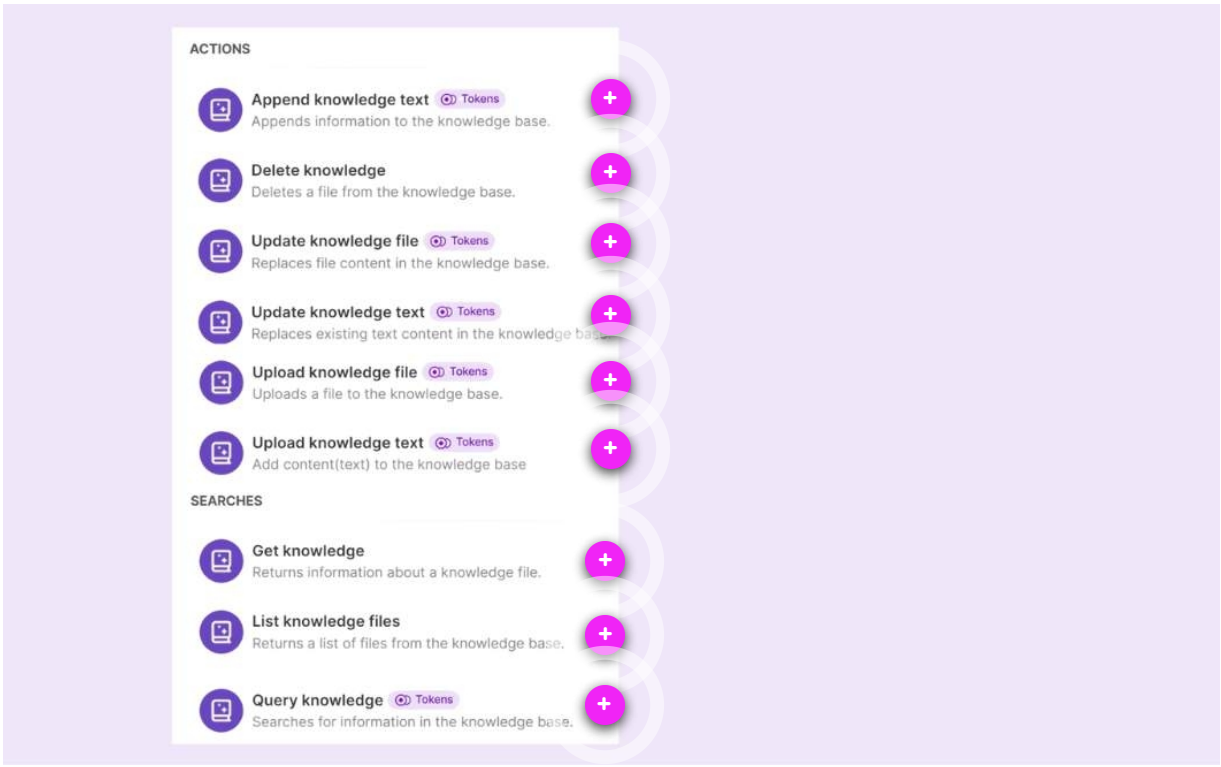
Continue to 2.3.3: Managing the knowledge files

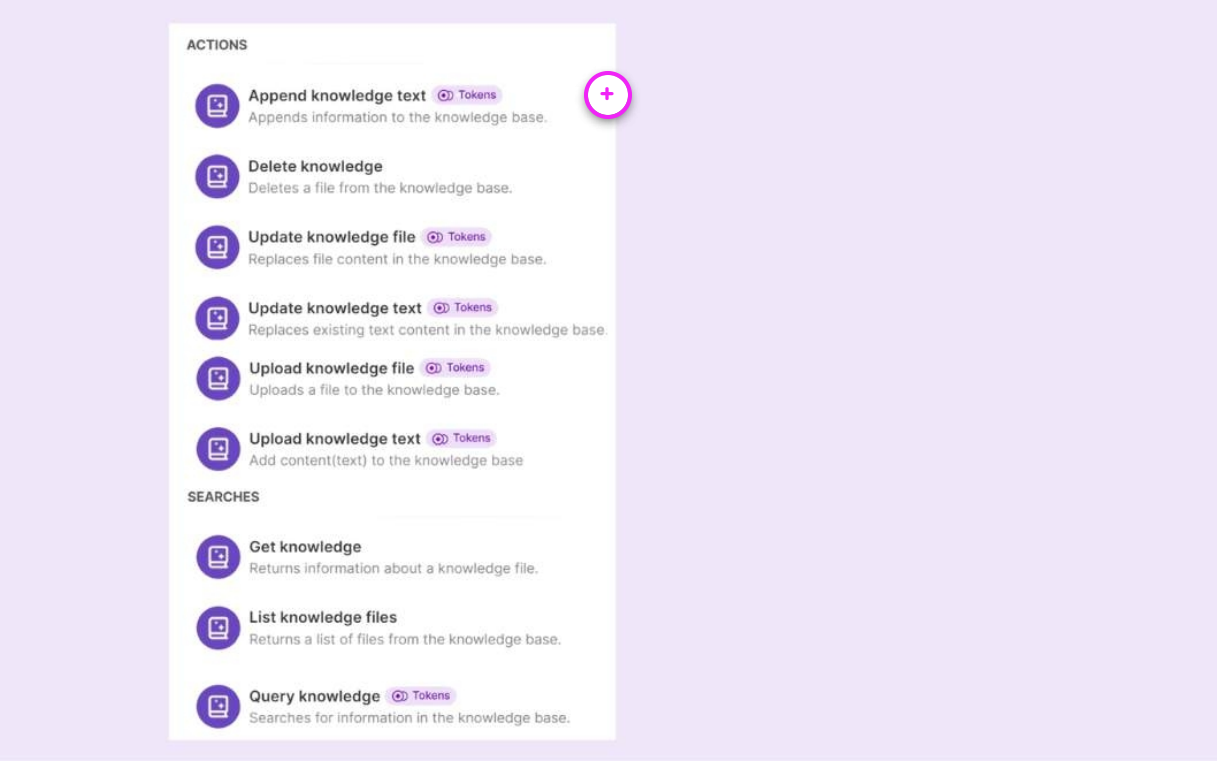
2.3.3 Managing the knowledge files

To use knowledge files with your agent in Make, you need to upload and manage them in your team's knowledge base. You do this by using Knowledge app modules in dedicated scenarios.

The **Knowledge** app has a set of modules that help you manage the files in your team's knowledge base and carry out tasks like uploading, updating, or deleting files.

Click each + to learn what each module does.


ACTIONS
Append knowledge text
Tokens
Appends information to the knowledge base.
Delete knowledge
Deletes a file from the knowledge base.
Update knowledge file
Tokens
Replaces file content in the knowledge base.
Update knowledge text
Tokens
Replaces existing text content in the knowledge base.
Upload knowledge file
Tokens
Uploads a file to the knowledge base.
Upload knowledge text
Tokens
Add content(text) to the knowledge base
SEARCHES
Get knowledge
Returns information about a knowledge file.
List knowledge files
Returns a list of files from the knowledge base.
Query knowledge
Tokens
Searches for information in the knowledge base.

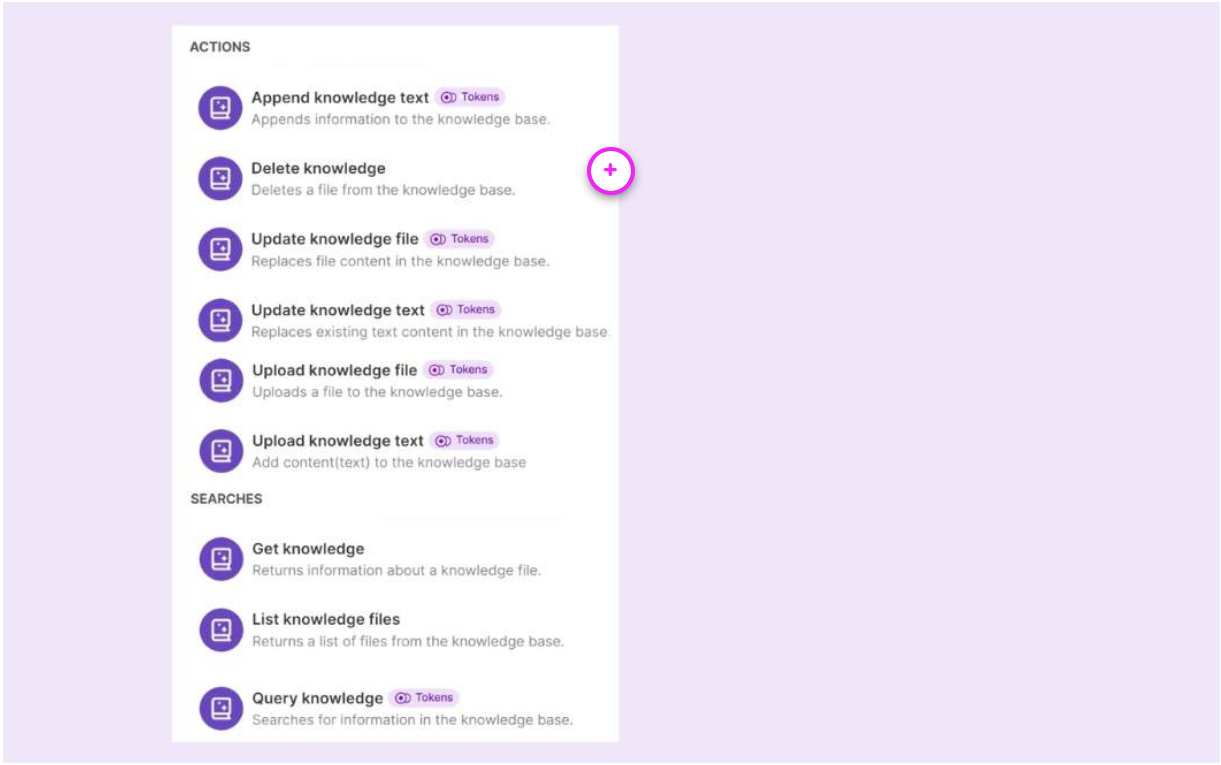


Append knowledge text

Main purpose: adds new content to an existing text file.

When to use it: use this every time you need to add new information to a knowledge file.

Example: you have a knowledge file containing the list of solved tickets so your customer support agent can provide level-1 support by referencing past solutions. You use this module to add new solved tickets to it on a regular basis (e.g., daily or weekly), keeping the knowledge base current with the latest resolutions.

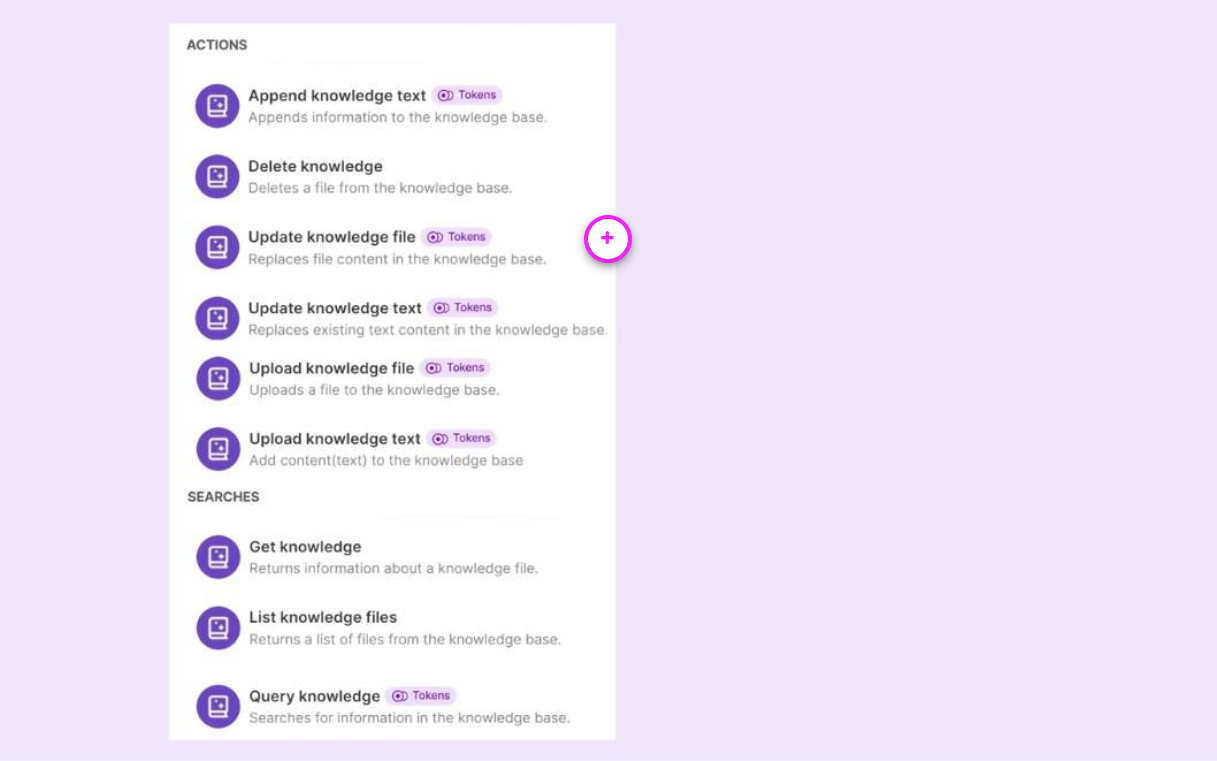


Delete knowledge

Main purpose: removes one or more knowledge files from your team's knowledge base. You can delete multiple files at once by providing a comma-separated list of IDs.

When to use it: use this to remove useless or obsolete knowledge files that your agents no longer need.

Example: you have outdated policy documents from 2023 that have been replaced with current versions. You won't need the old file anymore, so you can delete it.

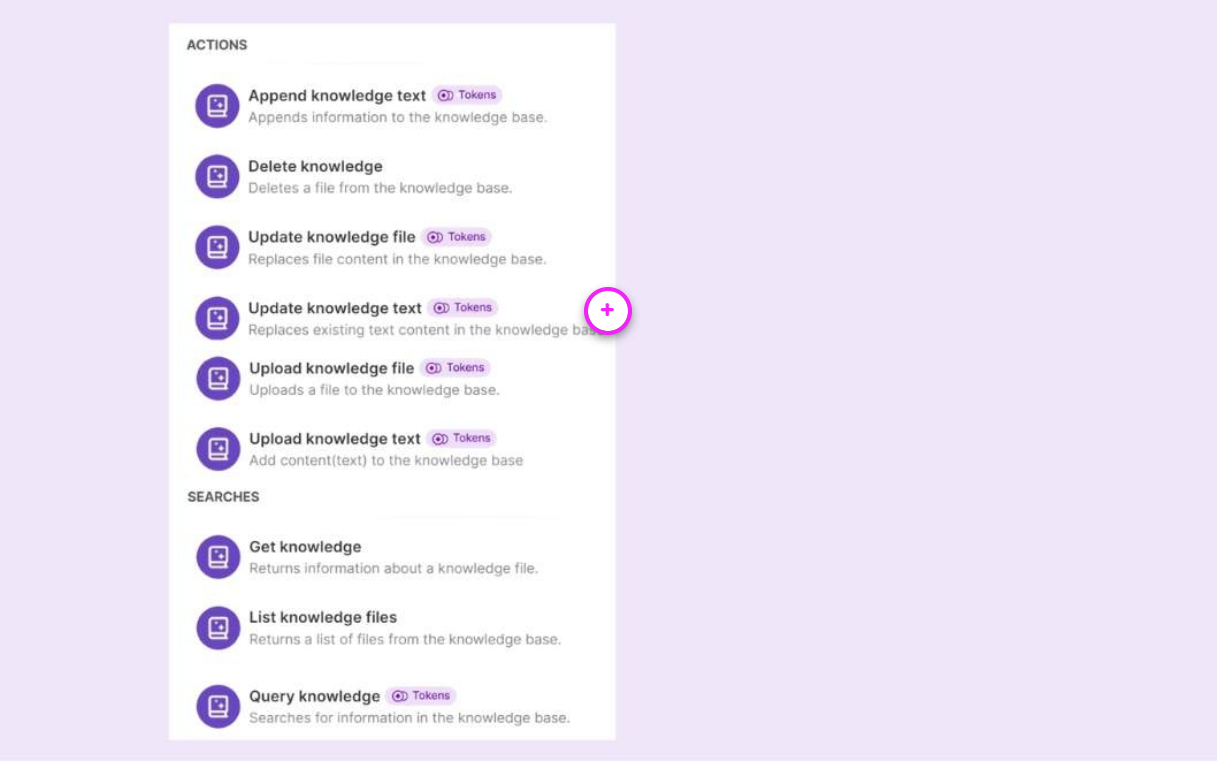


Update knowledge files

Main purpose: replaces all the content of a binary file.

When to use it: use this every time you need to update a binary knowledge file.

Example: you have a knowledge file in PDF format containing your product catalog with specifications and images. You use this module to update it when you release new product versions or update the catalog, keeping the agent's reference information current.

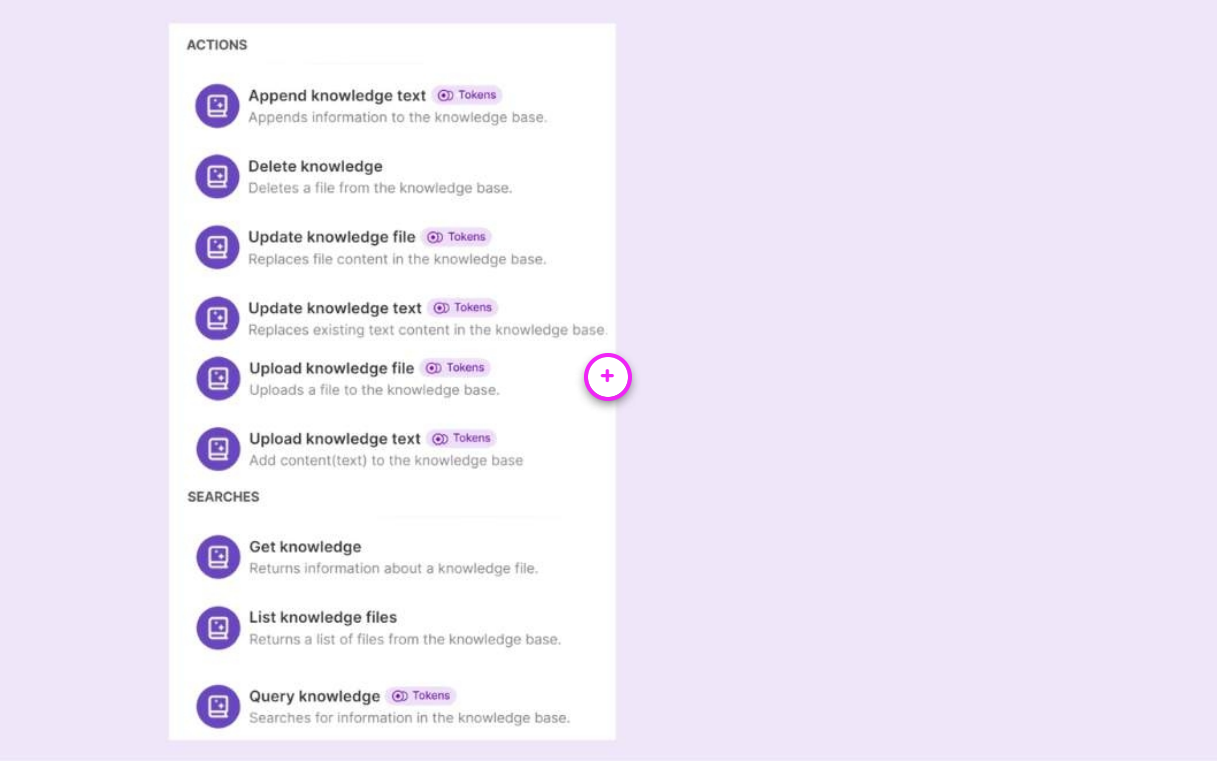


Update knowledge text

Main purpose: replaces all the content of a text file.

When to use it: use this every time you need to update a text-based knowledge file.

Example: you have a knowledge file containing your company's FAQ responses in text format. You use this module to update it when you add new questions, revise answers, or remove outdated information, keeping the agent's reference information current.

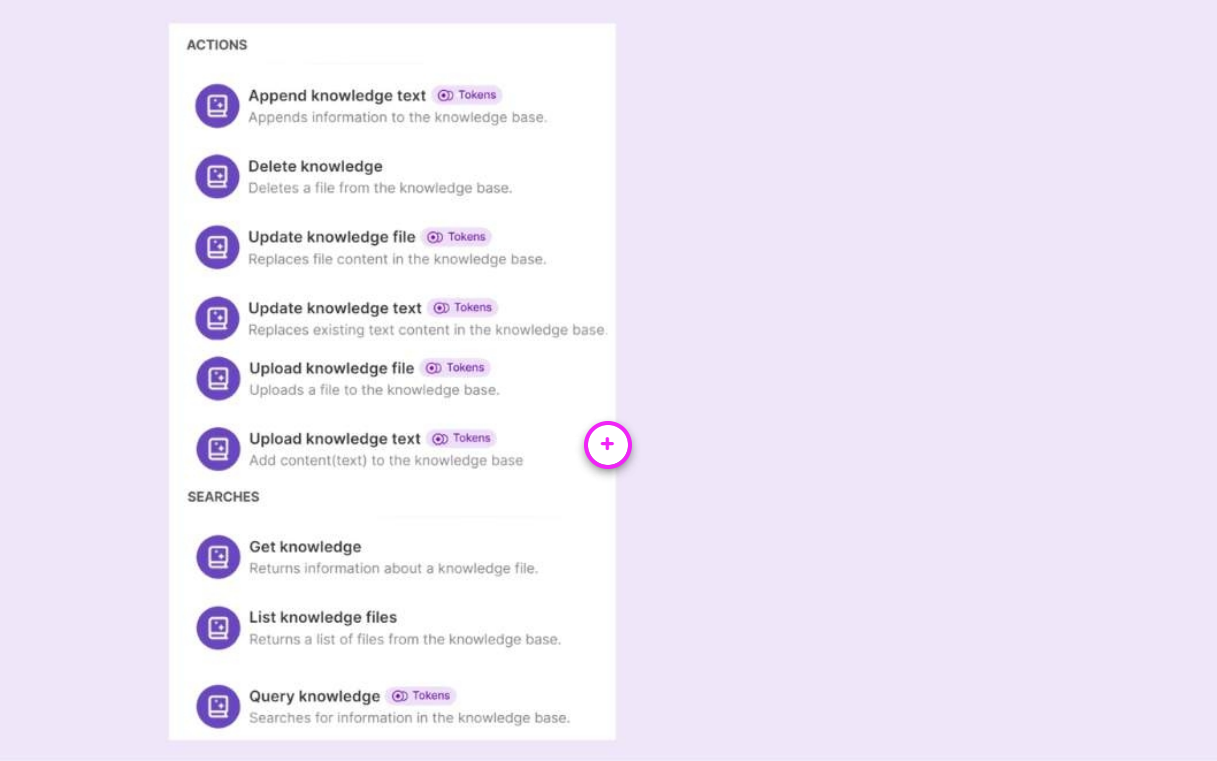


Upload knowledge files

Main purpose: uploads a new binary file (PDF) to the knowledge base. Make automatically parses the file, recognizes text (ignoring images), creates a unique ID, indexes it for future searches, and generates a description automatically.

When to use it: use this when you need to upload a PDF file to your knowledge base and want to automate the upload process.

Example: you receive product specification sheets as PDFs via email. You use Upload knowledge file in your scenario to automatically upload these PDFs to your knowledge base so your customer support agent can reference them when answering product questions.

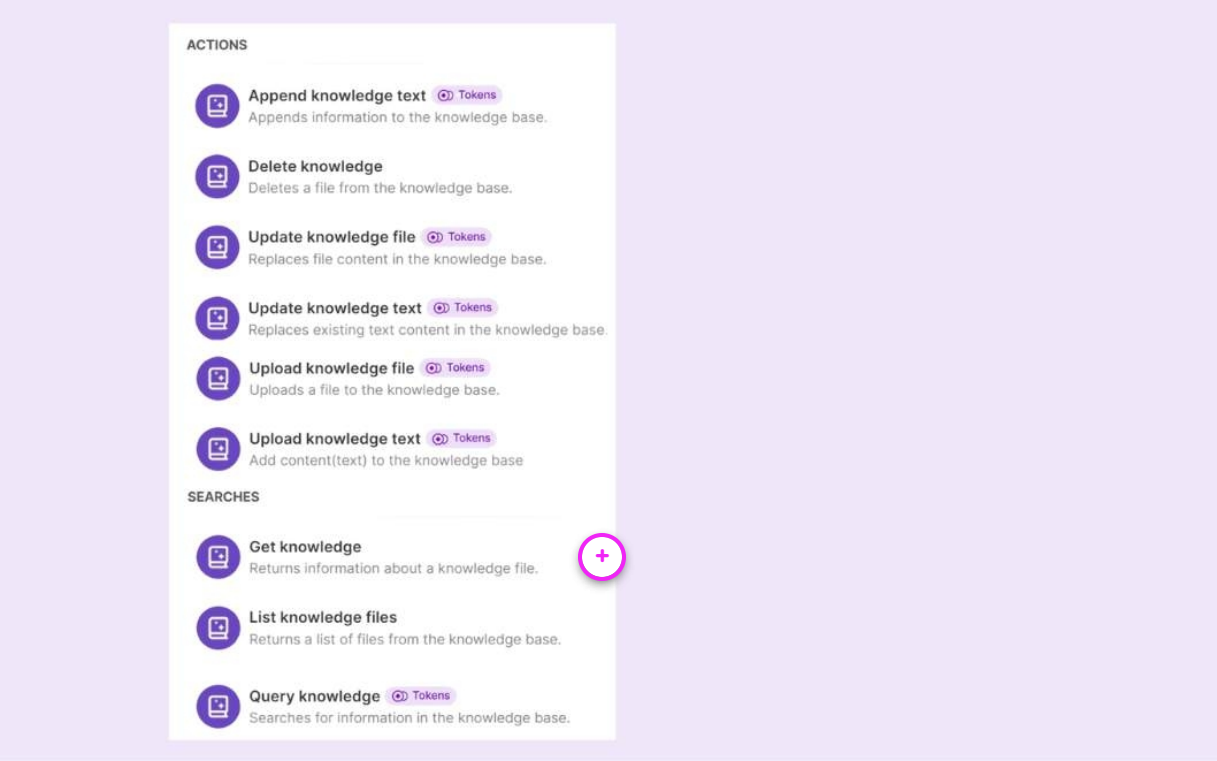


Upload knowledge text

Main purpose: uploads a new text-based file (JSON, CSV, TXT) to the knowledge base. Make creates a unique ID for the file, indexes it for future searches, and generates a description automatically.

When to use it: use this when you need to upload a text-based file to your knowledge base and want to automate the upload process, especially when pulling content from external systems.

Example: you're creating a new knowledge file for your support tickets. You use Upload knowledge text in your scenario to generate the file with all the content you pulled from your ticketing system.

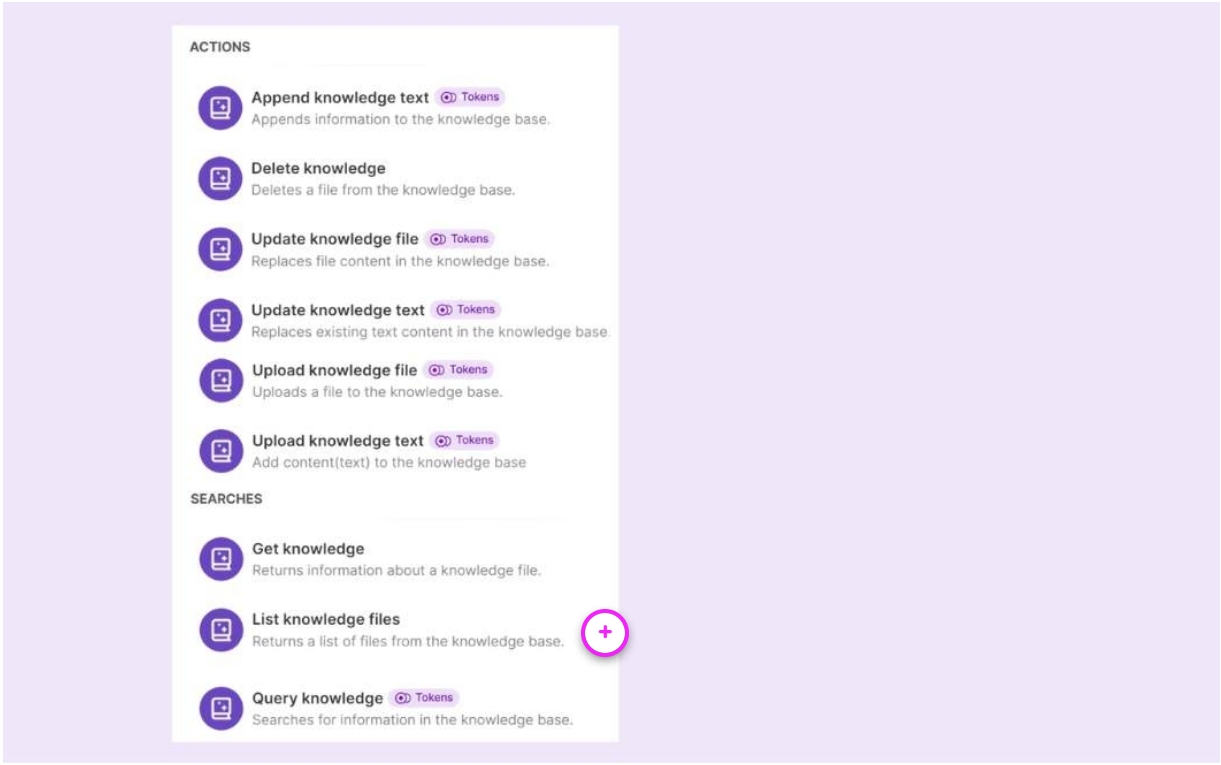


Get knowledge

Main purpose: retrieves details about a specific knowledge file, including its name, ID, and description.

When to use it: use this when you need to check the details of a specific file, either by providing its ID or manually selecting it when configuring the module.

Example: you want to analyze or verify the descriptions of your knowledge files. You use this module to check if the description accurately reflects what's in the file.

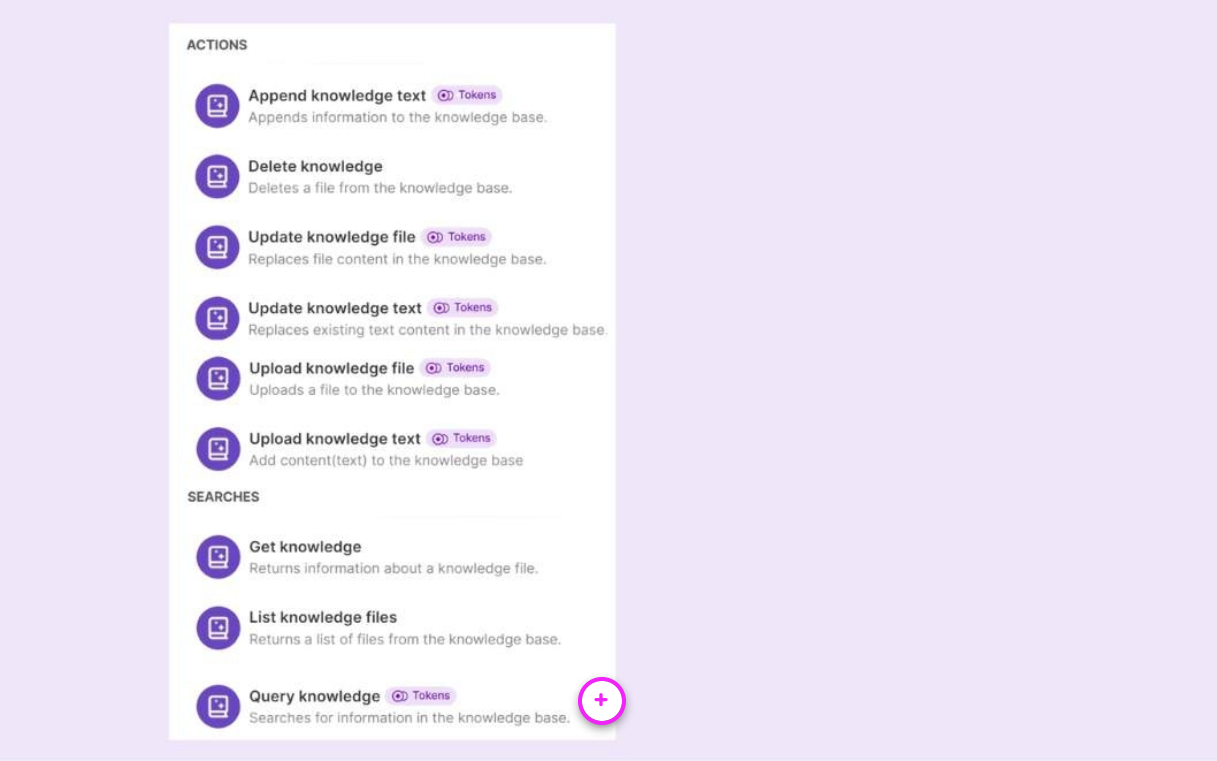


List knowledge files

Main purpose: retrieves all files currently in your team's knowledge base, providing the name, ID, and description of each file.

When to use it: use this when you want to maintain your file list, ensure everything is up to date, check if specific files exist, or identify which files need updating or deleting.

Example: you want to build a scenario that regularly checks the files in your knowledge base to confirm a specific file exists or to identify which files need updating or deleting. The module returns the list of files and their IDs, which you can then use in other modules like Delete knowledge or Update or append text knowledge.



Query knowledge

Main purpose: performs a semantic search in a specific knowledge file and returns all relevant chunks, the same way an AI agent would search it.

When to use it: use this when you need to search your knowledge files outside of an AI agent context, for example, to check if specific topics exist in your knowledge base or to retrieve information without involving an agent.

Example: you need to regularly check if your list of solved tickets contains specific topics. You query the file and review what it returns to verify the information is there.

Note that most of the time, you'll use these modules in separate scenarios dedicated to maintaining your knowledge files, rather than in the same scenario where your agent runs.

This is because these modules manage the knowledge base, while your agent accesses it, these are two separate functions

that work better in dedicated scenarios.

Pro Tip

You can build a dedicated agent to manage your knowledge base. This agent uses **Knowledge** app modules as tools to automatically maintain your files based on your specific needs. This keeps your knowledge base current without manual work.

Continue to 2.3.4: Give knowledge files to the agent

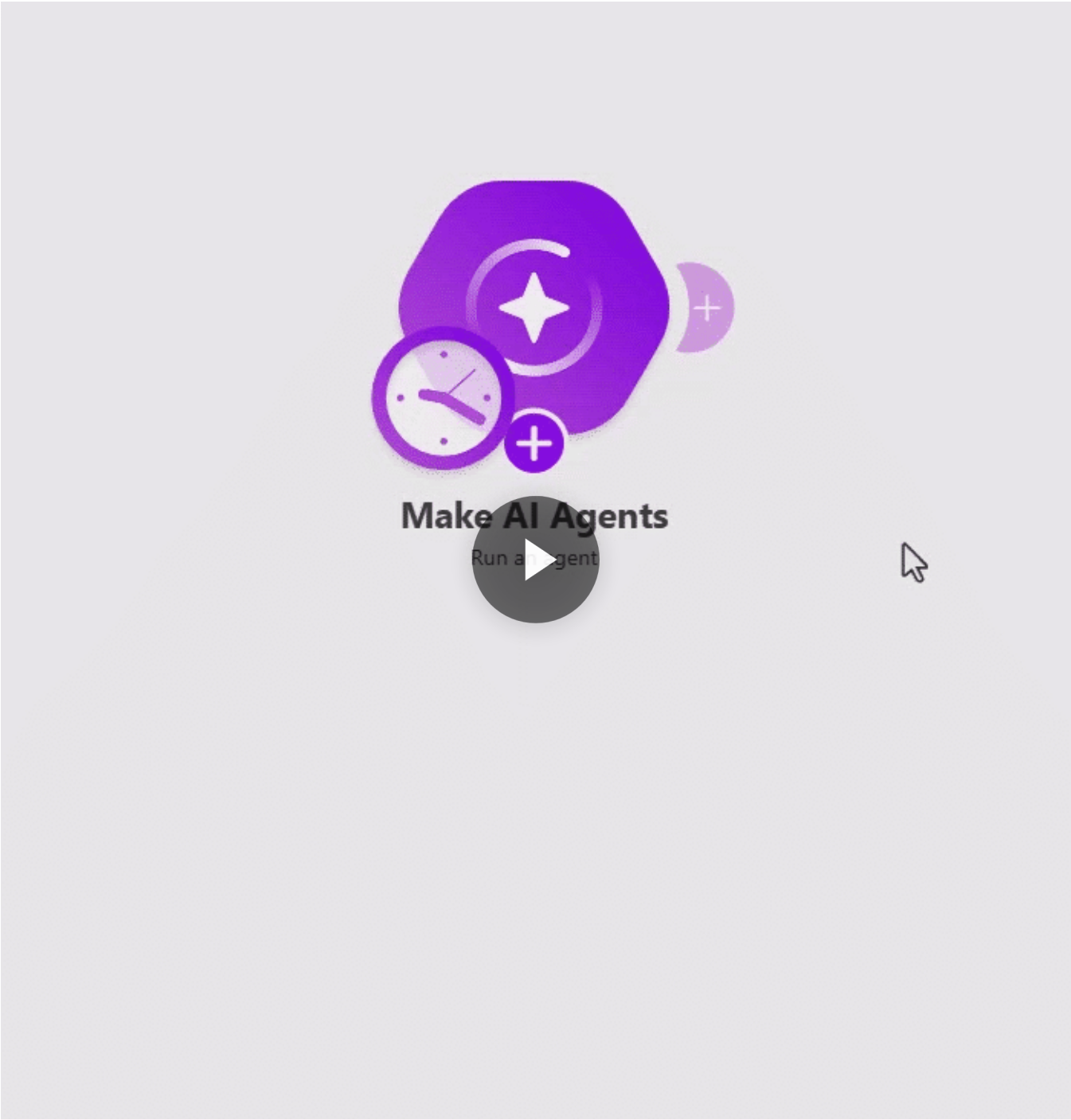
2.3.4 Give knowledge files to the agent

When setting up your AI agent, you can specify which knowledge files it has access to. You can choose from existing files in your knowledge base or upload new ones directly.

If you upload new files, you can do this right from the agent setup menu without needing to create a scenario with the Upload Knowledge file module.

To do this, hover over the **+** icon of the **Run an agent** module and select **Knowledge**. From here, you can give your agent access to existing files or upload new ones.

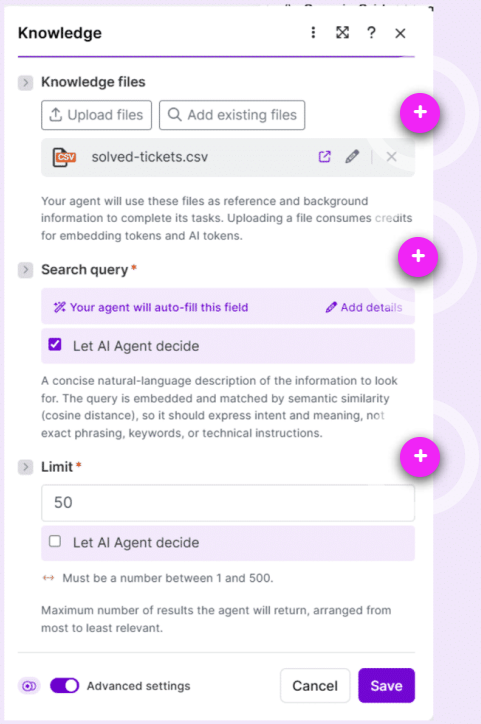
The files you upload become part of your team's knowledge base, and any agent within that team can have access to them.

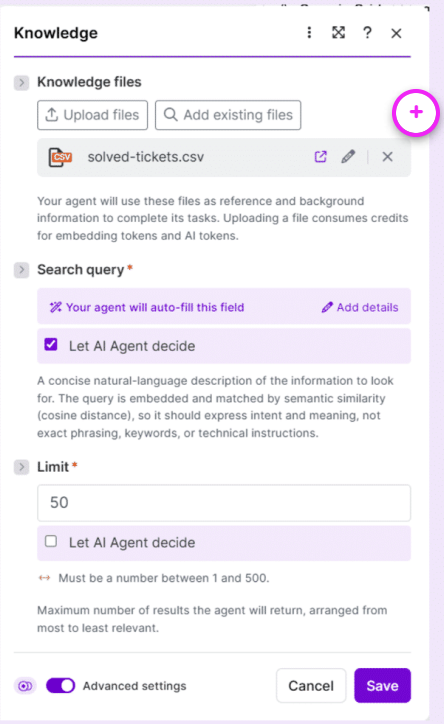


Here you can also configure how your agent works with these knowledge files. Toggle the **Advanced settings** to see all the

available options.

Click each + to learn more.

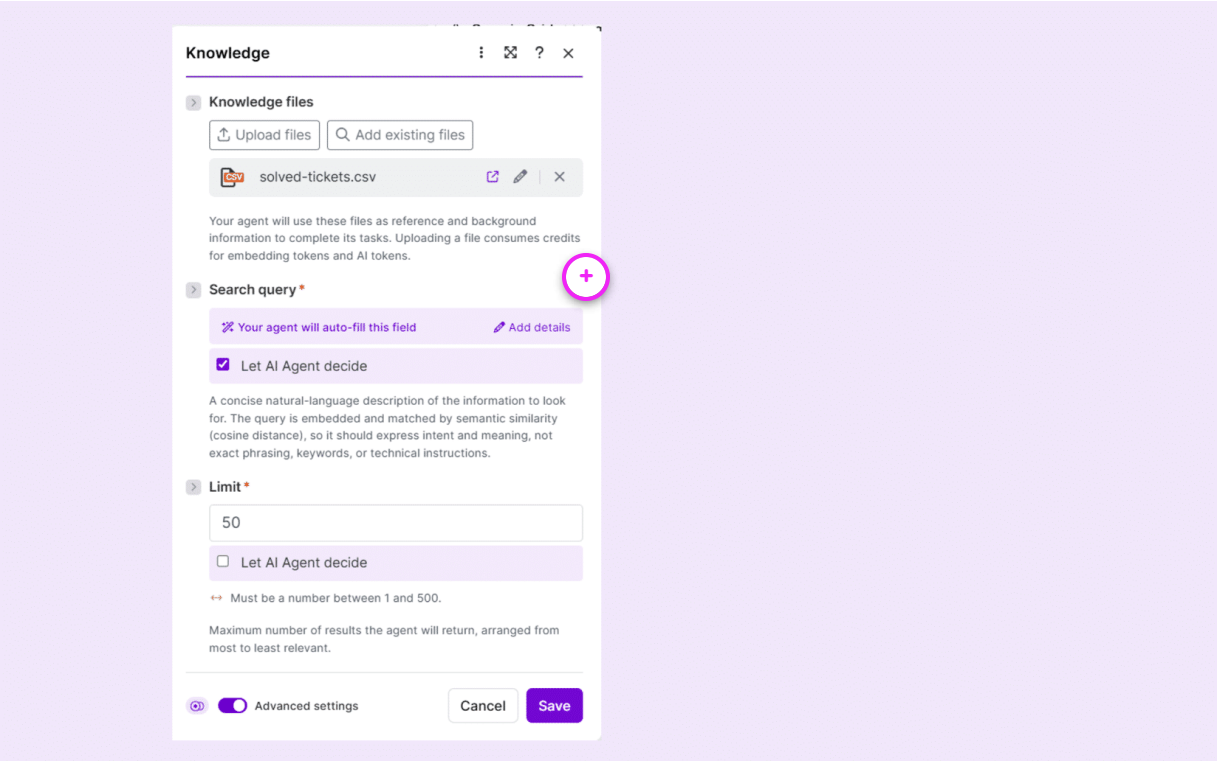




Knowledge files

This is the name of the knowledge file that your agent can access.

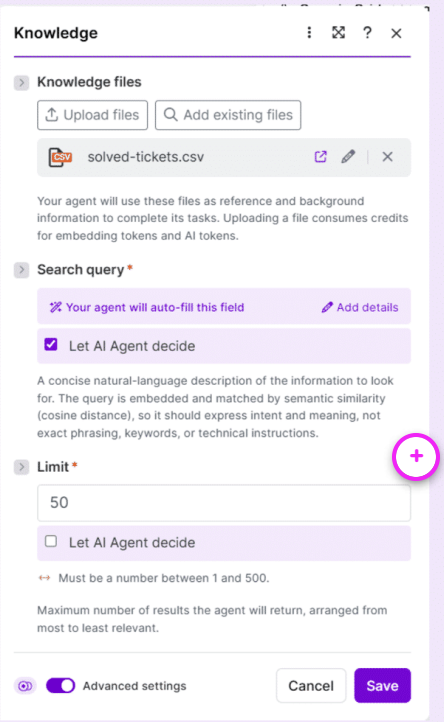
If needed, you can edit the description that Make generated automatically. The agent uses this description to understand what's in the file and decide whether to use it.



Search query

You can specify the query that the agent uses to retrieve information from the knowledge files.

By default, it's set to **Let the AI Agent decide**, allowing the agent to use the appropriate query based on the specific task.



Limit

You can specify the number of results the agent gets when searching the knowledge files.

Note that this information becomes part of the context window of the LLM your agent uses. Keep the number low to avoid overloading it.

Your agent can have access to multiple knowledge files at once. When carrying out a task, the agent uses the file names and descriptions to decide which files to search.

This means the agent automatically selects the most relevant files for each query rather than searching through everything, making retrieval more efficient and accurate.

Continue to 2.3.5: See it in action

2.3.5 See it in action

Let's look at a use case where the agent uses knowledge files to get specific information it needs for its task.

You have a customer support agent that helps resolve IT issues by first checking if a similar ticket has been solved before. If the agent finds a matching ticket in the knowledge base, it guides the customer through the resolution steps; if no match is found, it escalates the issue by creating a new ticket.

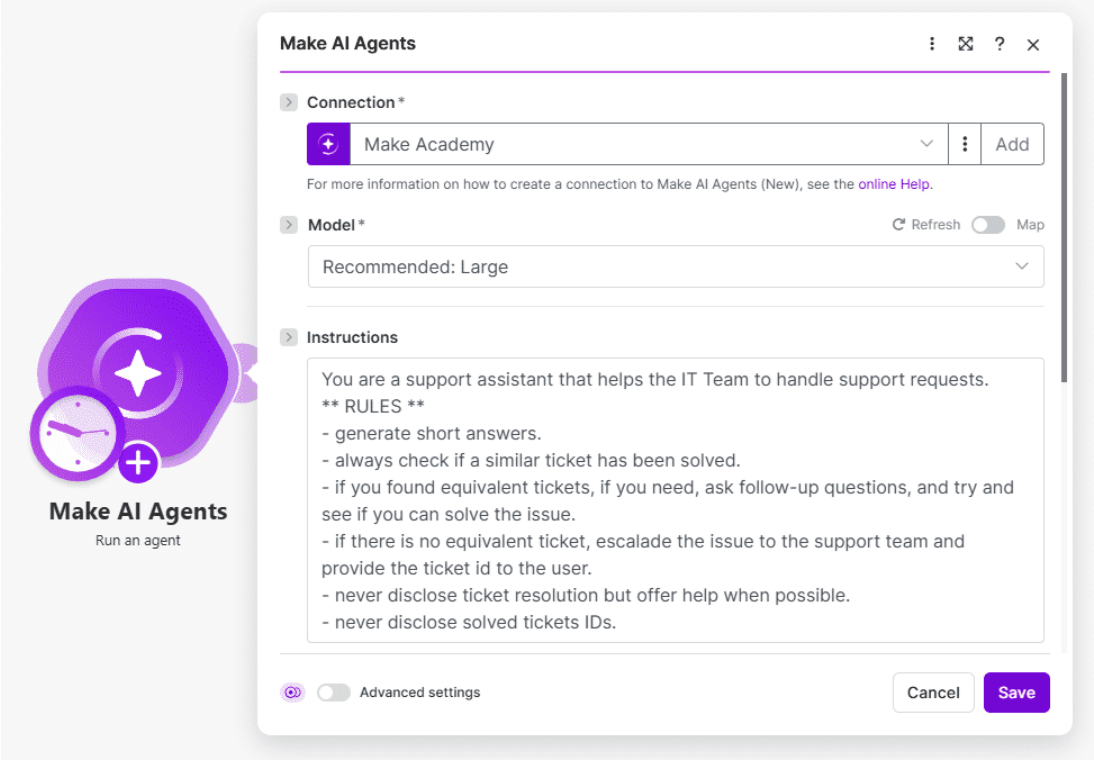
To perform its task, the agent needs access to a knowledge file containing all solved tickets. This allows the agent to search through past resolutions and use that information to provide solutions to customers with similar issues.

Let's see how to add the knowledge file and see the agent in action.

Work through each stage before you continue.

Step 1

Build the agent



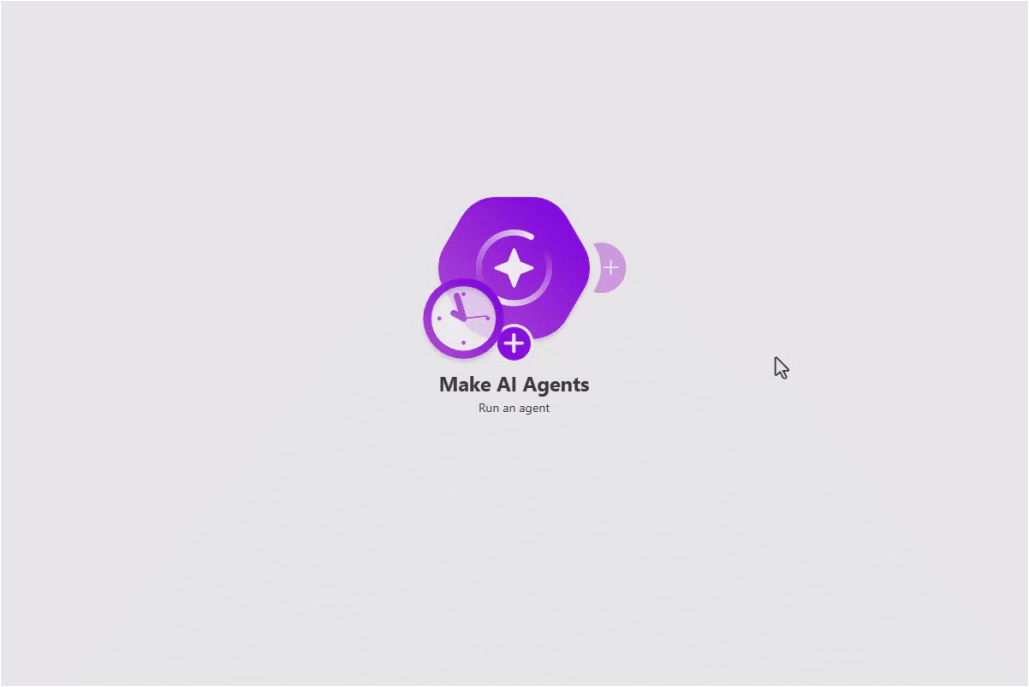
In a Make scenario, add a **Run an agent** module to start building your agent.

In the **Instructions**, specify how you want your agent to behave when managing tickets: define its role, the steps it should follow when helping customers, when to use the knowledge base, and when to escalate issues.

Add any constraints about what information the agent should or shouldn't share with customers. Then provide the agent with the **Escalate issue tool** so it can create new tickets when needed.

Step 2

Add the knowledge file



After you build your customer support agent, you need to give it access to the knowledge file containing the solved tickets.

Hover over the **+** icon of the **Run an agent** module. Select **Knowledge** and upload the file. Leave all other settings as default and click **Save**.

Step 3

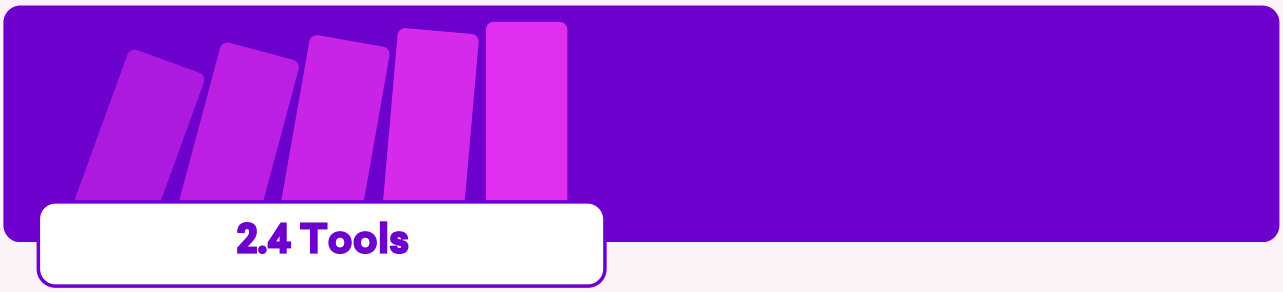
Run the agent

Use the **chat** to report an issue to your agent. Notice that it queries the knowledge base and finds a related ticket. After you provide more details, the agent offers a solution based on what the information it found in the knowledge file.

To test this go to the Make website and go to the scenario with your AI agent.

Click on the lightbulb in the bottom right to open the **Resources**. Select the **Give knowledge to the agent** one and follow the instructions.

Continue to 2.4: Tools



2.4 Tools

If you're familiar with Make, you know you can use Make apps to retrieve information from documents, like reading Google Sheets or querying databases.

You might wonder if you could build a tool with these apps to let your agent access information instead of using knowledge files.

Knowledge files and tools have different purposes.

Using tools works well in these cases:

- Your document is in a **file format not supported** by knowledge files, like Excel spreadsheets, especially when you can clearly specify what information to retrieve.
- You want your agent to **work on a document** rather than use it as a knowledge source, for example, summarizing a file or analyzing its contents.
- Your agent only needs to **access a document once** rather than referencing it repeatedly.

However, when you use tools to retrieve document content, you typically import all the information at once. In contrast, knowledge files use **RAG with semantic search** to retrieve only relevant chunks, avoiding context window overload. This **makes knowledge files more efficient** for reference documents.

When should you use knowledge files vs tools to work with files?

⭐ Use knowledge files as your default

Knowledge files are built specifically for this purpose and retrieve only relevant information without filling the context window. Use them whenever possible.

🎡 Use knowledge files for repeated access

If your agent needs access to the same information multiple times across conversations, use knowledge files to keep the document as part of the knowledge base.

💥 Use tools for one-time tasks

If your agent needs to access a file once or perform work on it (like summarizing or analyzing), use tools, they're designed for this and keep the file separate from your agent's knowledge base.

🤦 Use tools for unsupported file types

Use tools for file formats not supported by knowledge files. Supported types for the knowledge files: JSON, TXT, CSV, PDF (text only, no images).

Continue to 2.5: Direct file processing

2.5 Direct file processing

The agent has the ability to process files directly. It does it using the LLM's native capabilities, without requiring extra modules.

It has two capabilities:

- Take a file as input and **parse** it to extract information
- Generate a file as **output** containing specific information

Let's have a look at each of them.

Continue to 2.5.1: File parsing

2.5.1 File parsing

You can pass a binary file to the agent and it extracts information from it.

These are the supported file types:

For the agent to parse a file, you need to provide the file name and the actual content you want to process. Then you **specify in the agent's input** what kind of parsing you want your agent to perform.

The advantage is that you can **ask the agent to do the parsing directly**. You don't have to create a tool to do so.

Pro Tip

When the agent directly parses the file, it doesn't keep it in the conversation history. It **only saves the file metadata** (name, file type,

file size). This prevents clogging up space in the context.

You can provide the file to the agent in two ways:

1. Build a tool
2. Pass the file directly to the agent

Let's look at each of them.

1: Build a tool

1

Build a tool

You can create a scenario tool that **retrieves the file from its source** and returns it to the agent.

> **Note that the agent decides when to call this tool: when the task requires the file, the agent will call the tool and receive it for parsing.**

You have an agent that processes customer inquiries.

You build a **Get customer contract** tool that retrieves contract PDFs from your document system. When a customer asks about their specific agreement terms, the agent:

- **identifies the customer ID** from the conversation,
- **calls this tool** to fetch their particular contract file,
- then **parses it** to answer their question.

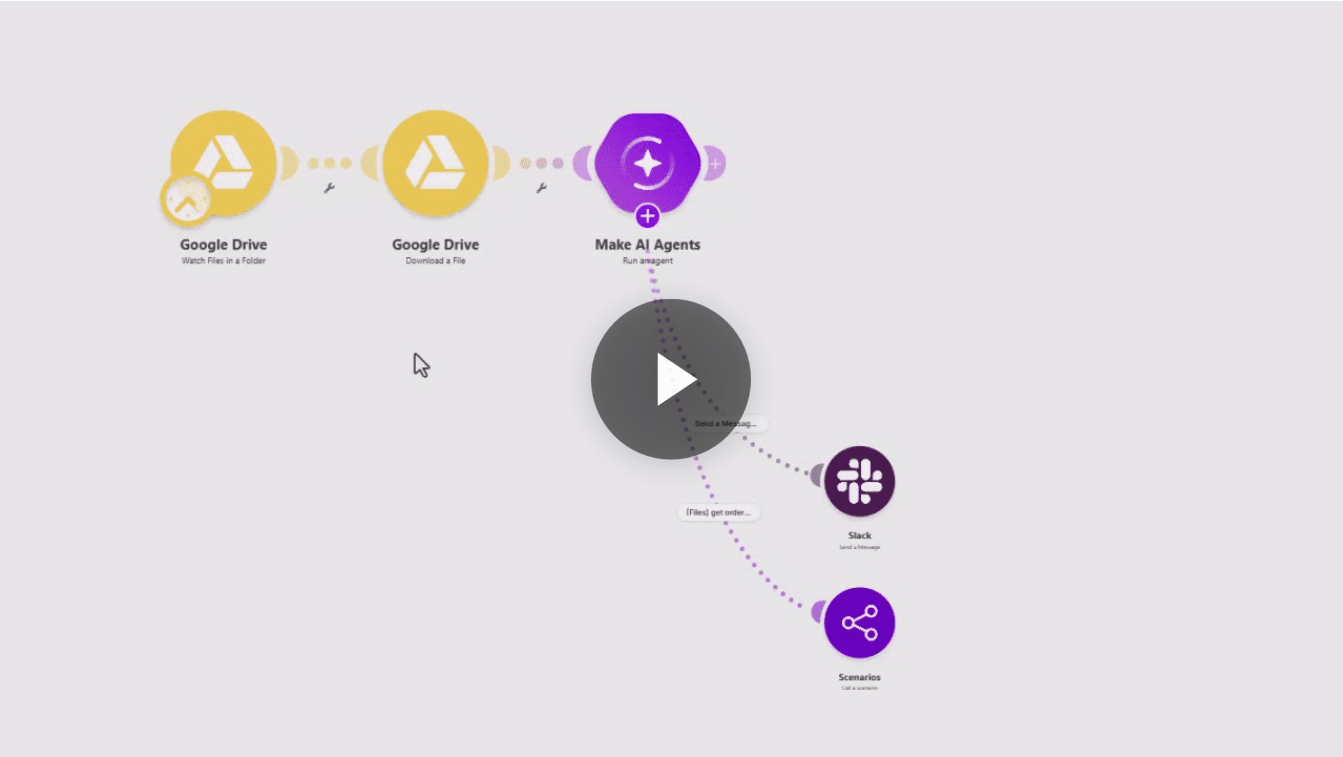
The agent only retrieves contracts when the inquiry requires it and automatically selects the correct file based on the customer context.

2

Pass the file directly to the agent

You can download the file in your scenario before the agent runs and pass it directly to the agent through the agent setup window. The file is available immediately when the agent starts.

To do so, add modules to your scenario that download or retrieve the files before the **Run an Agent** module. Then map the output to the **Input files** field in the agent setup window so your agent has access to the file and can process it:



You have an agent that validates sales orders.

The agent **receives a sales order PDF** from **Google Drive** and processes it to extract order details. Then it calls a **Get order information** tool to retrieve corresponding data and **compares both sources** to identify discrepancies.

If it encounters any discrepancies, it adds the info to a Google Sheets and notifies the sales team.

When should you use each method to give your agent the file for processing?

It depends on your use case:

🛠️ Tools

This method allows you to retrieves the files dynamically. The agent decides which file to get, when to get it, and can retrieve multiple files as needed.

🚀 Pass files directly

This method allows you to provide the file to the agent before it runs.

Use this when:

- **The agent selects the file based on context:** for example a customer support agent retrieves the contract for a specific customer based on their ID
- **The agent only needs files for certain queries:** for example an HR support agent it needs to retrieve the policy document when asked about benefits, and not for general questions
- **Multiple related files are needed to complete one task:** for example a customer support agent retrieves the main contract, plus all the addendums

Use this when:

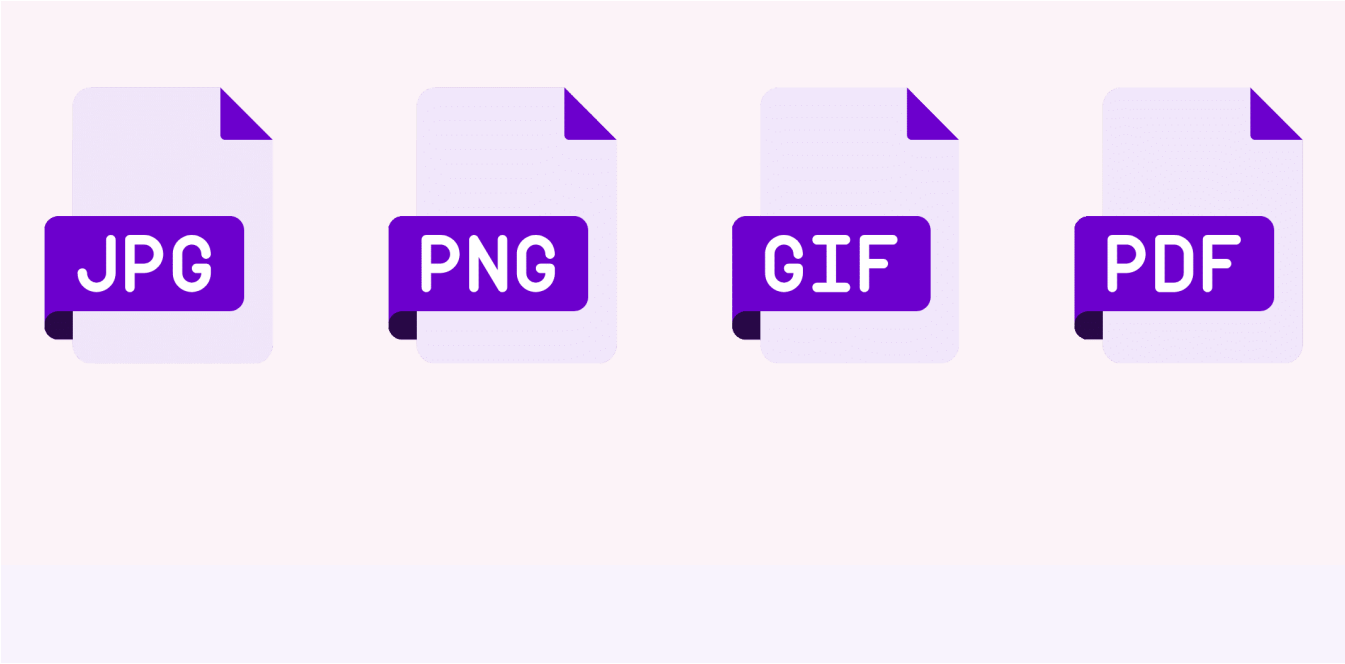
- **You want the file to trigger the scenario:** for example, an invoice PDF arrives via email and automatically starts the scenario with the sales support agent, which validates the invoice details and processes the payment
- **The file selection is simple and straightforward, you have no dynamic decisions about which file to get:** for example, a quality control agent processes the daily production report downloaded every morning at 9 AM

Continue to 2.5.2: Generate file

2.5.2 Generate file

The agent can generate files directly using the information it has processed.

It can generate text files in the following formats:



You have a customer support agent that retrieves customer data from a CRM.

You want to **create a PDF report** for all customers in a specific country to prepare for your next meeting. You **use a tool to retrieve the data** from the CRM for all customers from a specific country. Then you **ask the agent directly to generate the PDF report**. You don't need to add any extra module.

The advantage is that the **agent can generate the file directly**. You don't have to create a tool to do so.

Continue to the wrap up for this unit

2.6 Wrap up

1

Knowledge files extend your agent's knowledge beyond what the LLM learned during training. You upload them to your team's knowledge base, and when a task requires it, the agent uses RAG to retrieve only the relevant chunks. This means only the needed information enters your agent's context, not the entire file, keeping context usage efficient.

2

Tools give your agent access to files it needs to work on. The agent decides which files to retrieve and when, then processes them to complete its tasks.

3

Direct file processing allows your agent to **parse and generate files using the LLM's native capabilities** without adding a specific tool for file handling. You can provide files to your agent by building a tool that retrieves them or by passing them directly to the agent.

You've completed the second unit! Great!

You now understand the different ways your agent can work with files and when to use each one.

In the next unit, you'll learn about security considerations when building and using agents.

make | academy

Mark this task complete to continue to the next unit.