Paolo Dalprato

Connecting Claude and NotebookLM

Last updated: March 15, 2026

Claude is a versatile tool for analyzing texts, answering questions, and generating content. However, when the work requires managing extensive documentation — dozens of documents, or just a few but very long ones — the practical limits of the context window become apparent. Google's NotebookLM addresses this problem in a complementary way, allowing you to upload large amounts of material and query it with source-grounded answers.

This manual documents **NotebookLM MCP Structured**, an MCP server I developed to connect Claude Desktop to NotebookLM. The integration allows you to query document collections loaded in NotebookLM directly from a conversation with Claude, combining Claude's expressive capabilities with NotebookLM's document fidelity.

Who this guide is for

The guide is designed for Claude Desktop users who need to work with extensive documentation: technical manuals, regulatory collections, research corpora, corporate archives. No programming skills are required, but you need to be able to run a few commands in the terminal by following step-by-step instructions.

What you need

To use the system you need three components: a **Claude Pro** subscription (or higher) with Claude Desktop installed, a **Google account** (free or paid) for NotebookLM, and **Node.js** version 18 or later installed on your computer. The *Prerequisites* chapter describes in detail how to verify and obtain each component.

What the manual covers

The first chapters explain why the integration is useful and how it works, with particular attention to the problem of source fidelity and the automatic structuring solution. The middle chapters walk you through installation, authentication, and initial operations. The final chapters cover notebook library management, querying strategies, real-world use cases (including a comparative legal analysis that demonstrates the method in action), and troubleshooting.

The project

NotebookLM MCP Structured is a fork of the notebooklm-mcp (https://github.com/PleasePrompto/notebooklm-mcp) MCP server by **Gérôme Dexheimer**, who made the connection between Claude and NotebookLM possible. My fork adds an *automatic prompt structuring* system that transforms user questions into structured requests with explicit source fidelity constraints before sending them to NotebookLM. On the return path, the system instructs Claude to present the response without adding its own knowledge that doesn't come from the documents.

The project is open source, available on GitHub (https://github.com/paolodalprato/notebooklm-mcp-structured).

Under the hood

The fork was developed entirely through *vibe coding* with Claude: design and code writing with Sonnet 4.5, review and refactoring with Opus 4.6. No line of code was written manually — a concrete example of how generative AI tools can be used to develop working software starting from problem definition and dialogue with the model.

Table of contents

Why integrate Claude and NotebookLM

Claude is a powerful tool for answering questions, analyzing texts, and generating content. However, when the work requires managing extensive documentation — dozens of documents, or just a few but very long ones — practical limits become apparent. This integration is designed to overcome them.

Claude's limitations with documents

The first limitation concerns the *context window*. Files loaded into a chat or project documentation take up space in the context window, reducing the number of interactions possible before the system loses track of earlier information.

The second limitation is more subtle and concerns *hallucinations*. When a language model doesn't find precise information in the provided context, it tends to generate plausible but potentially inaccurate responses. In specialized domains, where a single wrong detail can invalidate an entire procedure, this behavior is particularly problematic. If you ask Claude to explain how to configure software based on partial documentation, the model might fill in the gaps with generic or outdated information, producing a response that looks correct but contains errors.

What NotebookLM offers

Google's NotebookLM handles these problems differently. The service allows you to upload many documents, even large ones, and uses Gemini to analyze them. The key characteristic is that NotebookLM responds based *exclusively* on the content of the uploaded documents, without integrating external knowledge — it retrieves information from the documents and uses them as the sole source for building its response.

Documents in NotebookLM are organized into *notebooks*, separate folders containing sources related to a specific topic. Each notebook can accommodate sources in many formats: audio files, images, PDFs, Google documents (Docs, Slides, Sheets), Microsoft Word files, text and Markdown files, web URLs and public YouTube video URLs, as well as text copied and pasted directly. It's best for each notebook to be specialized on a single topic, to get more precise answers.

NotebookLM is described by the official guide as "a research assistant", with capabilities oriented toward document consultation rather than expressive generation. Hence the idea of connecting it to Claude, which excels precisely in the ability to elaborate, synthesize, and produce output in different formats.

Data privacy

Google states that NotebookLM does not use personal data, uploaded sources, queries, or model responses for training.

The vicious circle

Connecting the two tools seems like the ideal solution: Claude's versatility combined with NotebookLM's document fidelity. In practice, a subtle problem emerges that develops in two directions.

- **Outbound.** When the user asks Claude something generic, for example "*analyze the rulings in the documents*", the margin for interpretation is enormous. Claude passes this request to NotebookLM, which responds proportionally to the question's vagueness. A simple question produces a simple answer.
- **Inbound.** NotebookLM's response, already simplified, arrives at Claude. Claude does what it does best: completes, enriches, contextualizes. It integrates the information with background knowledge, especially when the received information is summary. The result is a text that mixes document content and general knowledge, where it becomes difficult to distinguish what comes from the documents and what from Claude's "general culture".

There's a difference between asking an assistant "*tell me what the contract says*" and asking "*list the contract clauses citing the page number for each one, and if a standard clause is not present, flag it explicitly*". The second question leaves no room for creative interpretation.

In many contexts, the vicious circle behavior is acceptable. But when analyzing legal documents, fact-checking research, or verifying a technical procedure, the distinction between *what the documents say* and *what Claude thinks* becomes essential.

The solution: automatic structuring

The original MCP server by Gérôme Dexheimer works correctly — it connects Claude to NotebookLM and passes questions without modification. If a user structures their question well each time, specifying constraints and output format, they get better results. The problem is that result quality depends entirely on the user's discipline in formulating precise requests at every interaction. NotebookLM MCP Structured breaks the vicious circle by making a good practice systematic. Instead of relying on user discipline, the system automatically intervenes at two moments.

- **Outbound**, it transforms questions into structured prompts before sending them to NotebookLM, adding explicit constraints such as using only information present in the uploaded documents, citing sources for every statement, and explicitly declaring when information is not available.

- **Inbound**, it instructs Claude to present NotebookLM's response faithfully, without enriching it with its own knowledge that doesn't come from the documents.

Both transformations happen transparently. The user continues to ask questions naturally; the system takes care of structuring them and controlling how Claude handles the responses.

When it makes sense to use the integration

The integration is particularly useful with extensive technical documentation, corporate policy collections, product-specific knowledge bases, and research materials with many sources to cite. It doesn't make sense for generic questions, real-time information, or short single documents that can be loaded directly into the conversation.

How it works

The system intervenes at two distinct moments: before sending the question to NotebookLM and after receiving the response. The user doesn't need to do anything differently — both interventions happen behind the scenes.

The three-phase flow

Every interaction goes through three phases.

- **Phase 1: question structuring.** The user asks a question naturally. Claude reads the guidelines contained in the `ask_question` tool description, which specify how to structure the request. Based on these instructions, Claude transforms the question by adding a thematic output format, a citation format, a completeness signal, and a placeholder for missing information.
- **Phase 2: transit.** The MCP server receives the structured question from Claude and passes it to NotebookLM without modification. NotebookLM processes the request by consulting the documents loaded in the selected notebook and returns a source-grounded response.
- **Phase 3: response control.** NotebookLM's response returns to Claude through the MCP server. Before Claude presents it to the user, two control mechanisms kick in. The first is a completeness reminder, automatically appended by the server to every response, which pushes Claude to compare the received response with the user's original question and ask NotebookLM further questions if something is missing or unclear. The second is the faithful presentation instructions, contained in the same guidelines read in phase 1, which tell Claude to present the response without adding external knowledge or "improvements".

What happens outbound: the structuring

In a test conducted on the seven official DaVinci Resolve 20 manuals, loaded into a NotebookLM notebook, an intentionally generic request was made:

> *List the AI-based features of DaVinci Resolve.*

Claude transformed this simple question into a structured prompt similar to this:

```
List the AI-based features of DaVinci Resolve.

Organize the response by thematic topics.
Try to cover all aspects discussed in the documents.
For each topic:
- TOPIC: [identifying title]
- DESCRIPTION: [summary with context, connecting information
  across different documents]
- EVIDENCE: "direct quote" [Source: document]

If a topic appears in multiple documents, show evidence
from each one.
If not present in documents: [NOT FOUND IN DOCUMENTS]
```

The user didn't write any of this. The structuring was automatic — the original question was preserved as-is, but Claude added a structured output format, citation formatting, and instructions for missing information. The result was an organized catalog of AI features with direct quotes from the documentation.

Recognized question types

Claude adapts the prompt structure based on the detected question type. The approach is task-oriented — each question type has a specific pattern that focuses on output structure and cross-references between documents.

- **Comparison**: organizes the response by comparison points with similarities, differences, and cross-references between different documents.
- **List**: organizes by thematic topics with description and citations. If the same item appears in multiple documents, shows all occurrences and any discrepancies.
- **Analysis**: organizes by thematic topics with summaries and connections between different documents.
- **Explanation**: structures the response starting from the base concept with supporting citations, examples from the documents, related concepts, and known limitations.
- **Extraction**: this is the default type for all other questions, organizes by thematic topics with descriptions, citations, and cross-references between sources.

What happens inbound: Claude's added value

At this point, one might ask — if the structuring already produces a well-organized, source-grounded response, why not show NotebookLM's response directly?

Because Claude on the return path is not a mere relay. It's in the return phase that Claude's capabilities become a concrete advantage over using NotebookLM directly, provided these capabilities are channeled in the right direction. What Claude adds is the quality of *presentation*, *synthesis*, and *active research* — not external knowledge.

Active research

Every NotebookLM response, before reaching Claude, is enriched by the server with a hidden reminder. This reminder asks Claude to stop and evaluate whether the response actually covers everything the user asked, whether something is unclear or incomplete, whether something is missing. Claude can autonomously ask another question to NotebookLM within the same session, without the user needing to intervene.

In practice, a single user question can generate two or three successive queries to NotebookLM, each more targeted than the previous one, before Claude presents a complete response. The user sees only the final result.

Synthesis and organization

Claude can rework the received information with its own communication capabilities — from summarizing long responses to reorganizing content into a clearer structure, to highlighting key points. Most importantly, it can combine results from queries to *different notebooks* into a single coherent analysis, a capability that NotebookLM alone doesn't offer since it always works on one notebook at a time.

The fidelity constraint

These capabilities are channeled by a precise constraint — Claude does not add knowledge that doesn't come from the documents. The structuring guidelines contain explicit instructions for the return phase that constrain Claude to faithful presentation of document content. If information is not present in the documents, the response declares it with the placeholder [NOT FOUND IN DOCUMENTS], instead of filling the gap with general knowledge.

The constraint concerns *content*, not *form*. Claude can and should use its own organization and synthesis capabilities, but the material it works with remains exclusively what comes from the documents. As described in this article (https://www.ai-know.pro/notebooklm-con-gemini-e-claude-interrogare-i-notebook-dallesterno/) on external notebook access, the more elaborate the output, the more the risk of contamination increases, and the fidelity constraint is the mechanism that prevents it.

Multilingual support

The system works with any language supported by Claude. The structuring instructions guide Claude to adapt labels and constraints to the conversation's language, without specific configuration. For example, "TOPIC/DESCRIPTION/EVIDENCE" becomes "ARGOMENTO/DESCRIZIONE/EVIDENZE" in Italian, and "[NOT FOUND IN DOCUMENTS]" becomes "[NON PRESENTE NEI DOCUMENTI]".

The system has been thoroughly tested with Italian.

Language consistency

For optimal results, it's advisable to maintain the same language throughout the conversation. Switching languages during the session may produce unpredictable structuring results.

Verifying what happens behind the scenes

NotebookLM saves chats, which means you can open the notebook directly at notebooklm.google.com to see the structured prompt sent by Claude, along with NotebookLM's original response with all internal reference links.

This transparency mechanism allows you to verify that structuring was applied correctly and to compare the original response with the presentation made by Claude.

Under the hood

The system's architecture is based on a precise division of roles. The structuring instructions — which patterns to use, how to format citations, how to handle missing information — are *defined in the MCP server code*, within the `ask_question` tool description. When Claude Desktop loads the server, it reads these instructions and applies them. The actual structuring therefore happens in Claude, which acts as the client, but following rules written by the server. The MCP server doesn't touch the questions that transit to NotebookLM because it is the transparent intermediary.

This design choice has several advantages. Multilingual support is automatic since Claude natively handles any language. The structuring logic can be updated by modifying a single file in the server code, without touching Claude. And the system adapts to the conversation context without needing fixed templates for each language or question type.

The return control operates on two distinct and complementary levels. The first is a constant in the server code (`FOLLOW_UP_REMINDER`) that is concatenated to every NotebookLM response before returning it to Claude, pushing it to verify completeness and ask additional questions if necessary. The second is the "Response Handling" section in the structuring guidelines, which instructs Claude on faithful presentation. The separation between the two levels is intentional — the completeness check works even if the guidelines are modified, and vice versa.

Compared to the original server version, which handled structuring server-side with templates for each language and different enhancement modes, the fork simplified the architecture by moving the structuring logic into the tool description. This eliminated hundreds of lines of code (multilingual templates, language detection, response wrapping) in favor of a lighter and more maintainable approach.

A technical note — NotebookLM doesn't handle decorative lines in prompts well. Character sequences at the beginning of the request such as `===` or `---` cause timeouts. The structuring instructions specify using only plain text headings, avoiding any decorative typography.

Prerequisites

Before proceeding with installation, you need to verify that you have all the required components. The system consists of three components (Claude Desktop, NotebookLM, and the MCP server) that work together and each require their own prerequisites.

Claude Desktop

A **Claude Pro** subscription (or higher) and the Claude Desktop application installed on your computer are required. Claude Desktop is available for Windows, macOS, and Linux.

The application can be downloaded from Anthropic's official website: claude.ai/download (https://claude.ai/download). Once installed, simply log in with your Claude account credentials.

Why Claude Desktop

The MCP server is designed specifically for Claude Desktop. Other versions of Claude (such as Claude Code or the web interface) may not support all the features described in this manual, particularly the automatic authentication flow.

NotebookLM

A **Google account** is required, either free or paid. NotebookLM is accessible at notebooklm.google.com (https://notebooklm.google.com).

Free accounts have the following limits:

- maximum 100 notebooks;
- up to 50 sources per notebook;
- up to 500,000 words per source;
- maximum 200 MB per uploaded file;
- 50 queries per day to NotebookLM. Paid accounts (NotebookLM Plus) offer higher limits across all categories, including 500 daily queries and up to 300 sources per notebook.

Accepted sources include audio files, images, Google documents (Docs, Slides, Sheets), Microsoft Word files, PDFs, text and Markdown files, web URLs, public YouTube video URLs, as well as text copied and pasted directly.

Node.js

The MCP server is a Node.js application. You need to have Node.js **version 18 or later** installed on your computer.

To check if Node.js is already installed, open a terminal and type:

```
node --version
```

If the command returns a version number equal to or greater than `v18.0.0`, the requirement is met. If the command is not recognized or the version is lower, you need to install or update Node.js.

To install Node.js, download the LTS (Long Term Support) version from the official website nodejs.org (https://nodejs.org). The guided installer takes care of everything: it downloads the package, installs it, and configures the necessary environment variables. After installation, close and reopen the terminal before verifying the version.

Chrome

Chrome is required only for the authentication phase.

Summary

Component	Requirement	Where to verify
Claude	Pro subscription (minimum) + Claude Desktop	claude.ai/download (https://claude.ai/download)
NotebookLM	Google account	notebooklm.google.com (https://notebooklm.google.com)
Node.js	Version 18+	`node --version` in the terminal

Installation

Installation requires two steps: downloading and building the MCP server, then configuring Claude Desktop to use it.

Downloading the server

Open a terminal and navigate to the folder where you want to install the server. Run the following commands in sequence:

```
git clone https://github.com/paolodalprato/notebooklm-mcp-structured.git
cd notebooklm-mcp-structured
npm install
npm run build
```

The first command downloads the source code from the GitHub repository. The second enters the project folder. The third installs the required dependencies. The fourth compiles the TypeScript code into executable JavaScript.

If git is not installed

As an alternative to `git clone`, you can download the code as a ZIP archive from the project's GitHub page (github.com/paolodalprato/notebooklm-mcp-structured (https://github.com/paolodalprato/notebooklm-mcp-structured)). On the repository page, click the green "Code" button and select "Download ZIP". Extract the archive to a folder of your choice and continue with `npm install` and `npm run build` from the extracted folder.

Configuring Claude Desktop

You need to edit Claude Desktop's configuration file to register the new MCP server.

Configuration file location: | Operating system | Path | |---|---| | Windows | `%APPDATA%\Claude\claude_desktop_config.json` | | macOS | `~/Library/Application Support/Claude/claude_desktop_config.json` | | Linux | `~/.config/Claude/claude_desktop_config.json` |

Open the file with a text editor and add (or modify) the `mcpServers` section. If the file is empty or doesn't exist, create it with this content:

```
{
  "mcpServers": {
    "notebooklm": {
      "command": "node",
      "args": [
        "/full/path/notebooklm-mcp-structured/dist/index.js"
      ]
    }
  }
}
```

Replace `/full/path/` with the actual path to the folder where you downloaded the server.

Example for Windows:

```
{
  "mcpServers": {
    "notebooklm-mcp": {
      "command": "node",
      "args": [
        "D:\\MCP_SERVER\\notebooklm-mcp-structured\\dist\\index.js"
      ]
    }
  }
}
```

⚠ **Path separators on Windows**

On Windows, you must use double backslashes (`\\`) in paths within the JSON file, or alternatively a single forward slash (`/`). A single backslash doesn't work because JSON interprets it as an escape character.

If the file already contains configurations for other MCP servers, add the `"notebooklm-mcp"` entry inside the existing `mcpServers` object, separating it with a comma from the previous entry.

Verifying the installation

After saving the configuration file, **close and restart Claude Desktop**. On startup, Claude automatically detects the new MCP server.

To verify that the installation was successful, type in the chat:

> *Verify that NotebookLM is configured correctly*

Claude queries the MCP server with the `get_health` tool and confirms whether the connection is active. At this point the server is installed but not yet authenticated with Google. This step is described in the next chapter.

Under the hood

The `claude_desktop_config.json` file is Claude Desktop's central registry of MCP servers. When Claude starts, it reads this file and launches each listed server as a separate process. The NotebookLM MCP server communicates with Claude through the MCP protocol (Model Context Protocol), an open standard that Anthropic published to allow anyone to develop extensions for Claude. Each MCP server provides *tools*, instruments that Claude can invoke during a conversation to perform specific actions.

First authentication

The MCP server needs to access NotebookLM with the user's credentials. Authentication happens once and the credentials are saved locally for subsequent sessions.

Login procedure

Type in Claude's chat:

> *Set up authentication for NotebookLM*

Claude uses the `setup_auth` tool and the server automatically opens a dedicated Chrome window. In this window, simply log in with your Google account — the one that contains the notebooks you want to query.

The server uses its own isolated browser profile, separate from Chrome or any other browser you may be using. There is no need to close Chrome or any other application before starting authentication.

The process typically takes 30–60 seconds. Once login is complete, the server automatically saves the authentication state in a local directory. You don't need to repeat the procedure for subsequent sessions, as long as the authentication hasn't expired.

Verifying the connection

After logging in, you can verify that everything is working:

> *Check NotebookLM status*

Claude uses the `get_health` tool and reports information about the authentication status, the connection to NotebookLM, and the browser state. If everything is in order, the system is ready to use.

Automatic renewal

Authentication has a natural expiration (24 hours). When the session expires, the server detects it automatically on the first request and starts the re-authentication procedure without user intervention.

In most cases, you don't need to worry about expiration. The system handles renewal transparently within the conversation. In a document analysis test, the server detected the authentication expiration and renewed it autonomously before proceeding, without requiring any intervention.

Switching accounts

If you need to use a different Google account (for example, to access notebooks on another account, or when you reach the daily query limit), you can force a new authentication:

> *Switch the NotebookLM account*

Claude uses the `re_auth` tool, which deletes the saved credentials and opens a new login window.

Under the hood

The server saves authentication cookies in a dedicated local directory: - **Windows**: `%LOCALAPPDATA%\notebooklm-mcp\` - **macOS**: `~/Library/Application Support/notebooklm-mcp/` - **Linux**: `~/.local/share/notebooklm-mcp/`.

The directory contains the persistent browser profile (with the login session) and the state file with cookies. The authentication state is considered valid for 24 hours; beyond this threshold, the server verifies the cookies and starts re-authentication if necessary.

```
The server uses Patchright, a fork of Playwright with stealth features, to simulate
realistic browsing behavior during authentication. The browser profile is completely
isolated from Chrome and other browsers installed on the system: there are no
interferences or conflicts.
```

Getting started

With the server installed and authentication completed, it's time to register your first notebook and try a first query.

Registering a notebook in the library

NotebookLM notebooks are not automatically accessible to Claude. You need to register them in the **library**, a local store that the MCP server consults to know which notebooks are available and when to query them.

To register a notebook you need the notebook's URL on NotebookLM and a description of its content.

How to get the notebook URL:

1. Go to notebooklm.google.com (https://notebooklm.google.com)
2. Open the notebook you want to register
3. Click **Share** in the top right
4. Select **Anyone with the link**
5. Copy the resulting URL

The URL has a structure similar to `https://notebooklm.google.com/notebook/abc123def456`.

Registering the notebook in Claude:

Start a conversation with Claude and describe the notebook naturally:

> *I have a NotebookLM notebook dedicated to n8n documentation, with tutorials, official guides, and personal notes. I'd like to add it to the library so I can query it. The URL is https://notebooklm.google.com/notebook/abc123def456*

Claude asks a few questions to better understand the content and proposes a structured card with name, description, topics, and use cases. After confirmation, the notebook is saved in the local library.

The importance of metadata

The description and topics provided during registration are not mere labels. Claude uses them to decide which notebook to query when it receives a question. An accurate description improves the precision of automatic selection — if you indicate that the notebook contains "*tutorials on automation with n8n and API integrations*", Claude will know to associate it with questions on those specific topics.

The first question

With the notebook registered, you can make your first query. It's not mandatory to specify which notebook to consult since Claude chooses based on metadata, but it's advisable when the registered notebook library contains similar ones.

> *How do you install n8n locally on Windows 11?*

Claude identifies the relevant notebook, creates a research session with NotebookLM, structures the question with source fidelity constraints, and returns the source-grounded response. The entire process happens transparently — the user doesn't need to worry about the structuring, which is applied behind the scenes as described in the **How it works** chapter.

Reading the structured response

The response follows the format imposed by the automatic structuring. Each piece of information provided includes a reference to the document source. If a requested piece of information is not present in the documents, the label *[NOT FOUND IN DOCUMENTS]* appears.

Trust the [NOT FOUND]

The absence declaration is a positive result, not an error. It means the system is effectively limiting responses to what is documented, without inventing or integrating with external knowledge. This is the expected and desirable behavior, especially when working with technical or legal documentation where precision is essential.

Verifying on NotebookLM

NotebookLM saves chats, which provides a **direct verification** mechanism for what happened during the query:

1. Open the notebook at notebooklm.google.com (https://notebooklm.google.com)
2. Check the chat history
3. Read the **structured prompt** that Claude sent
4. Read the **original response** from NotebookLM with internal reference links to the documents This transparency mechanism allows you to verify that structuring was applied and to compare the original response with the presentation provided by Claude. In contexts where source fidelity is critical, such as legal analysis or technical procedure verification, this audit capability is particularly valuable.

Continuing the conversation

After the first question, you can dig deeper without starting over. Claude maintains the session with NotebookLM, which retains the context of previous questions.

> *What if I wanted to configure an external PostgreSQL database instead of the built-in one?*

NotebookLM uses the context of the previous conversation (the Windows 11 installation) to provide a more precise and contextualized response. This progressive approach — starting with a broad question and deepening step by step — is one of the most effective ways to work with the system. Each answer can suggest the next question, building an increasingly detailed understanding of the topic.

Under the hood

When Claude queries NotebookLM for the first time on a notebook, the server creates a research session identified by a unique ID. The session corresponds to a persistent chat in NotebookLM. As long as the session remains active, subsequent questions are sent to the same chat, allowing NotebookLM to use the accumulated context. Sessions have a configurable timeout (15 minutes of inactivity by default) and the maximum number of concurrent sessions is limited to 10.

Managing notebooks

Over time the library grows — each new project or topic can have a dedicated notebook in NotebookLM. Effectively managing this library is important to keep the system useful and organized.

Listing notebooks

To get an overview of all registered notebooks:

> *Show me the notebooks in the library*

Claude uses the `list_notebooks` tool and presents the complete list with the name, description, topics, and use cases for each one. This overview helps you remember which resources are available and decide which one to query.

Adding a notebook

The procedure is the one described in the previous chapter — provide the URL and a description of the content. Claude builds the metadata through a dialogue and asks for confirmation before saving.

> *I have a new NotebookLM notebook with the DaVinci Resolve 20 manuals. The URL is https://notebooklm.google.com/notebook/xyz789. Add it to the library.*

Metadata matters — Claude uses it to decide which notebook to query when it receives a question. An accurate description of topics and use cases improves the precision of automatic selection.

Updating metadata

Notebooks evolve — documents are added, covered topics expand. You can update metadata without removing and re-adding the notebook:

> *The DaVinci Resolve notebook also covers Fusion integration and AI features. Update the description.*

Claude proposes the changes and waits for explicit confirmation before applying them. Updating metadata when you modify the sources in NotebookLM is a good practice — Claude does not automatically detect changes made inside notebooks.

Selecting a specific notebook

In some cases it's useful to explicitly indicate which notebook to query, especially when the library contains notebooks on similar or partially overlapping topics:

> *Use the legal documentation notebook*

Claude uses the `select_notebook` tool to set that notebook as the default for subsequent questions in the conversation. This prevents the automatic selection from choosing a different notebook than the one you want.

Searching among notebooks

With many registered notebooks, it's useful to be able to filter by topic:

> *Which notebooks do I have on artificial intelligence?*

Claude uses the `search_notebooks` tool to search the library based on keywords, returning only the relevant notebooks.

Removing a notebook

When a notebook is no longer needed, for example because it's been replaced by an updated version, you can remove it from the library:

> *Remove the "Old n8n documentation" notebook from the library*

Claude always asks for confirmation before proceeding.

Removal from the library, not from NotebookLM

Removing a notebook from the library only means that Claude will no longer query it automatically. The notebook remains intact in NotebookLM and can be re-added at any time.

Best practices for the library

Some guidelines to keep the library effective over time.

- **Specialized notebooks**: it's best for each NotebookLM notebook to be dedicated to a single topic. A notebook that mixes technical documentation and marketing material produces less precise answers because NotebookLM struggles to isolate relevant information in an overly heterogeneous context.

- **Up-to-date metadata**: when you add or remove sources from a notebook, also update the metadata in the library. Claude doesn't automatically see changes made in NotebookLM because it relies exclusively on the registered metadata.
- **Descriptive names**: a name like "n8n – Tutorials and workflows" is more useful than "Notebook 3". Claude uses names and descriptions to choose which notebook to query, so metadata clarity directly translates into response precision.
- **Periodic cleanup**: removing obsolete notebooks keeps the library manageable and reduces the risk that Claude queries outdated sources. A lean library with accurate metadata works better than a vast library with approximate descriptions.

Working with documents

Registering notebooks and asking questions is the starting point. To get the most out of the system, it's useful to know some strategies that leverage session management, progressive research, and the ability to query multiple notebooks.

Conversational sessions

When Claude queries NotebookLM for the first time on a topic, it creates a **research session**. This session maintains context between subsequent questions, enabling progressive conversations within the same notebook.

Sessions remain active for the entire conversation with Claude. If you switch to a completely different topic, Claude may decide to create a new session by closing the existing one. This automatic behavior maintains response precision by avoiding confusion between different contexts.

To view active sessions:

> *Show me the active sessions with NotebookLM*

To start over on a notebook while maintaining the same session identifier:

> *Reset the current session with NotebookLM*

Managing the daily quota

Each question to NotebookLM consumes a query from the daily quota (50 for free accounts, 500 for NotebookLM Plus). Using the same session for related questions is more efficient than starting over each time, because the accumulated context makes responses more precise. Explicitly closing sessions that are no longer needed is a good practice to keep the system tidy.

Progressive research

Instead of asking a single generic question, the most effective approach is to proceed through successive refinements, letting each answer inform the next question.

An example with technical documentation: 1. *Give me an overview of the n8n installation options* 2. *What are the practical differences between Docker and npm?* 3. *For professional use with a team of 5 people, which approach is most suitable?*

Each question builds on the previous answers, and NotebookLM constructs increasingly precise responses because it maintains the conversational context. This approach is particularly useful when exploring complex documentation or trying to solve intricate technical problems.

Querying multiple notebooks

One of the most significant features of the integration is the ability to work with **multiple notebooks in the same conversation**. NotebookLM alone doesn't allow querying multiple notebooks simultaneously: with the integration, Claude acts as an orchestrator and manages the switching between different notebooks.

A concrete example — analyzing documents contained in two separate notebooks.

1. *Analyze the cases in the "AI Case Law" notebook*
2. *Now analyze the rulings in the "AI Case Law – 2025 Update" notebook*
3. *Compare the recurring patterns in the documents loaded in both notebooks*

Claude keeps the results of the first analysis in memory and can compare them with those from the second, producing a comparative analysis based on sources from different notebooks. This approach was concretely tested in a comparative case law analysis on AI, described in the **Use cases** chapter.

Human review in the process

When working on complex analyses, human review between phases is essential.

The recommended method for multi-phase structured work is:

1. Clearly define objectives, methodology, and expected outputs before starting
2. Verify the results of each phase before moving to the next
3. Correct any inaccuracies immediately, to prevent them from amplifying in subsequent phases
4. Use direct verification on NotebookLM (by checking the chat history) to confirm source fidelity

This *human-in-the-loop* approach, where artificial intelligence works and the user validates, is the most reliable way to obtain accurate results from complex document analyses.

Optimizing requests

Result quality depends on question specificity. NotebookLM works best with precise and contextualized questions.

Less effective:

> *How does Docker work?*

More effective:

> *In the context of installing n8n on Windows 11, what are the Docker commands needed to create and start the container?*

Better precision in request formulation allows NotebookLM to retrieve exactly the relevant documentation sections, while vague questions return generic information. Automatic structuring improves any question, but starting from a precise request produces better results.

Information verification

The system is particularly useful as a **fact-checking** tool when writing documentation, technical articles, or training materials:

> *I'm writing that n8n supports up to 200 nodes in a single workflow. What do the documents say about this?*

Claude queries the notebook and returns the response with citations, confirming or refuting the claim based on the documentation. If the information is not present, it explicitly declares so.

Similarly, you can compare information found in external sources with the documentation:

> *I read that to install n8n on Windows it's better to use WSL2 instead of Docker Desktop. What does the official documentation say?*

The response distinguishes between what is documented and what is not, allowing you to separate opinions and informal advice from official guidance.

Under the hood

The server manages up to 10 concurrent sessions, each with an inactivity timeout of 15 minutes. When a session expires, it is automatically closed, freeing resources. Switching between different notebooks in the same conversation is managed by the server, which creates separate sessions for each notebook: Claude maintains the overall context in its own context window, while each NotebookLM session is independent. This allows comparing information from different notebooks without contexts mixing on the NotebookLM side.

Use cases

The system is versatile and adapts to different professional contexts. This chapter presents concrete scenarios that illustrate how to leverage the integration, including two documented real-world cases.

Legal document analysis

Legal analysis requires absolute source fidelity — every statement must be anchored to a specific document, citations must be precise, and documentation gaps must be explicitly declared.

With the system, you can load rulings, contracts, or regulations into specialized notebooks and query them with targeted requests:

> *Extract all liability clauses from the contract, citing the page number for each one. If a standard clause is not present, flag it explicitly.*

The automatic structuring adds the necessary constraints: exclusive use of sources, mandatory citations, declaration of missing information. The result is an analysis that clearly distinguishes between what the documents contain and what they don't.

Research and scientific literature

For those working with scientific publications, the system offers a structured way to conduct literature reviews:

> *What methodology did the authors use in the different studies? Cite the specific section of each paper.*
>
> *What limitations are explicitly mentioned by the authors?*
>
> *Are there contradictions between the results of the different studies?*

Each response is grounded in the documents with citations that allow tracing back to the original source. This is particularly useful when preparing a systematic review or verifying consistency across different sources.

Technical documentation and training

When working with extensive manuals, such as a software's complete documentation, the system allows extracting specific information without reading hundreds of pages.

An example — an intentionally generic request to query the notebook with the DaVinci Resolve 20 manuals, 7 manuals totaling thousands of pages:

> *List the AI-based features of DaVinci Resolve.*

Despite the simplicity of the question, the automatic structuring transformed it into a prompt with operational constraints, structured output format, and instructions for handling missing information. The result was an organized catalog of AI features with name, description, usage context, and direct quotes from the documentation, without needing to intervene to correct inaccuracies.

The case is described in detail in the **How it works** chapter, where the automatically generated structured prompt is also shown.

A real case: comparative analysis of AI case law

This is a documented real case. The objective was to analyze the evolution of case law on generative artificial intelligence, comparing cases collected in a book with two subsequent rulings not included in the volume.

The context

The material was organized in two separate notebooks: one containing a book with a collection of case law, the other with two recent rulings. The separation into different notebooks is one of the points where the integration shows concrete advantages over direct NotebookLM use, which doesn't allow querying multiple notebooks in the same session.

The working method

The work was organized in four phases with **human review between each one**. The clear initial definition of objectives, methodology, and expected outputs determined the quality of the entire process.

- **Phase 1: analysis of the first corpus.** Claude queried the first notebook, extracting legal acts and identifying recurring patterns in legal reasoning. In this phase, the value of human review emerged: the analysis contained some inaccuracies that were corrected before proceeding. Without the intermediate verification, those errors would have propagated to subsequent phases.
- **Phase 2: analysis of the second corpus.** With the method validated in the previous phase, Claude applied the same approach to the second notebook containing the recent rulings. The automatic management of switching between notebooks simplified the process.
- **Phase 3: comparative analysis.** Claude integrated the results of the two previous analyses, identifying common patterns and divergences in legal reasoning across the different corpora. In this phase, the ability to work with multiple notebooks in the same conversation proved decisive.

- **Phase 4: output production.** The work was synthesized into a structured markdown document and a PowerPoint presentation.

What the case taught

The most significant element is the importance of human review at every step. The system produces source-grounded results, but interpretation and validation require domain expertise. The *human-in-the-loop* approach, where the system analyzes and the user validates, prevented errors from early phases from amplifying in later ones.

Another lesson concerns authentication — during the work, the Google authentication session expired. The server detected the expiration and renewed credentials automatically, without interrupting the workflow.

The approach is transferable to any context requiring structured document analysis: scientific literature reviews, market analyses, technical documentation audits, comparison of regulation versions.

When not to use the integration

The system is not the best choice in every case:

- **Generic questions**: if the answer is general knowledge and doesn't require fidelity to specific sources, Claude alone is more than sufficient
- **Real-time information**: for news or up-to-date data, web search is more appropriate
- **Short documents**: a single document of just a few pages can be loaded directly into the Claude conversation, without needing to go through NotebookLM
- **Exploratory conversations**: when you don't need grounding to specific sources but want to freely explore a topic

Troubleshooting

Even with a correct configuration, problems can occur. This chapter collects the most common situations and their solutions.

Authentication problems

Authentication doesn't start automatically

If a request to NotebookLM fails without opening the login window, check the server status:

> *Check NotebookLM status*

Claude uses the `get_health` tool to diagnose the problem. Based on the result, it may be sufficient to manually start authentication with `setup_auth`.

Persistent authentication problems

In case of repeated errors, the most effective solution is a complete cleanup followed by a new authentication:

1. Ask Claude: *Run NotebookLM cleanup preserving the library*
2. Verify the preview of what will be deleted
3. Confirm the operation
4. Re-authenticate: *Set up authentication for NotebookLM*

The cleanup removes browser data, cache, and corrupted profiles, while preserving the registered notebook library.

Response problems

Responses contain external knowledge

If responses seem to integrate information not present in the documents: 1. Verify that the system is actually using the `ask_question` tool (automatic structuring is only active with this tool) 2. Try rephrasing the question more specifically 3. Verify on NotebookLM that the structured prompt was actually sent, by checking the chat history in the notebook 4. Verify that the `ask_question` tool actually returned a response. In Claude Desktop's chat, expand the tool call detail and check that the Response field contains NotebookLM's text. If the response is absent or contains an error, Claude may have generated the response autonomously with its own knowledge.

Timeout or no response

NotebookLM can time out if the prompt contains problematic formatting. The system automatically handles this by avoiding decorative lines in prompts, but if the server source code has been customized, verify that no sequences of `===` or `---` are present at the beginning of the structured request sent.

Missing citations in the response

Very generic questions can produce less structured responses. Rephrasing with greater specificity improves citation quality. For example, instead of *"Tell me about the project"*, ask *"What are the project milestones and their deadlines, with reference to the documents?"*.

Unexpected [NOT FOUND IN DOCUMENTS]

This declaration is correct behavior — it means the sought information is not in the uploaded documents. Verify that the selected notebook is the right one and that the relevant documents are actually present in NotebookLM.

Connection problems

Error "browserContext.newPage: Target page/context/browser has been closed"

The server recovers automatically from this error by recreating the browser context. Simply retry the request.

"ProcessSingleton" error or profile conflicts

The server is configured to automatically handle this situation using isolated browser profiles. If the problem persists, verify that there are no residual server processes and retry.

Account limits

Daily limit reached

Free accounts have a limit of 50 queries per day to NotebookLM. The counter resets at midnight (UTC time). Available options:

- wait for the counter to reset
- use the `re_auth` tool to switch to a different Google account (if you have multiple accounts)
- consider a NotebookLM Plus subscription for higher limits (500 queries per day)

Monitor consumption

There is no built-in counter in the system, but you can keep track of queries by counting the queries made during the day. If the work involves intensive use, plan the most important requests for the early hours of the day.

Complete cleanup

If problems persist despite troubleshooting attempts, you can perform a deep cleanup that removes all server data while keeping the notebook library:

> *Run complete NotebookLM cleanup preserving the library*

The cleanup scans eight categories of files: current installation data, NPM cache, server logs, temporary files, and other residual data. Before deleting anything, the system shows a detailed preview of what will be removed and waits for explicit confirmation. After cleanup, re-authentication is required.

Under the hood

Server data is saved in an operating system-specific directory: `%LOCALAPPDATA%\notebooklm-mcp\` on Windows, `~/Library/Application Support/notebooklm-mcp/` on macOS, `~/.local/share/notebooklm-mcp/` on Linux. This directory contains the persistent browser profile (with the login session), the authentication state with cookies, and the `library.json` file with the registered notebook metadata. The cleanup removes everything except `library.json` when the library preservation option is chosen. If necessary, you can manually delete the entire directory for a complete reset, keeping in mind that in that case you will also lose the notebook metadata and will need to register them again.