Data Science Project

Advisor: Fco. Javier Nogales Martín

13/01/2026

Final Report

Gen AI User Training - in collaboration with SANDOZ

Universidad Carlos III de Madrid

Data Science and Engineering Degree

César Álvarez-Cascos Hervías 100475191

Lauren Gallego Ropero 100496382

María de la Almudena Martín Gómez 100475249

Víctor Sebastián Marticorena 100496612

Adriana Rupérez Arellano 100496518

INDEX

1. Introduction

1.1 Problem description

Sandoz, a global leader in generic pharmaceuticals and biosimilars, works within a highly regulated and data-heavy environment. To support its decision-making and optimize internal operations, the central makes use of a centralized digital platform known as **SANITY**. This system functions as a core repository and process management tool, combining multiple workflows across different departments.

Despite its central role, SANITY’s practical use has become increasingly complex. The platform includes numerous interconnected fields, specialized capabilities and process hierarchies that can vary greatly across business units and user roles. In addition, the user base of SANITY is large and diverse - consisting of individuals with different backgrounds, responsibilities, and levels of technical expertise. As a result, it has become increasingly hard for users and even process developers to maintain a clear understanding of all the available information and procedures, and to keep an oversight of how knowledge is applied across the organization. This has led to several problems:

- **Information overload**: users have a hard time locating relevant information or understanding correct procedures for specific assignments
- **Low efficiency**: new or infrequent users need significant time to learn and navigate the platform effectively
- **Inconsistent data handling:** the great variation in user expertise often results in errors or process conflicts.
- **High training and support costs**: follow-up and troubleshooting continuously require considerable human resources
-

This creates a strong need for a reliable and intuitive interface that can surface the right information at the right time, grounded in authoritative documentation, reducing the risk of hallucinations and unsupported instructions.

1.2 Proposed solution

To address these challenges, this project proposes the design and implementation of a Generative AI user training assistant powered by Retrieval-Augmented Generation (RAG).

The goal is to create a natural language interface that enables SANITY users to query information, learn about processes, and receive guidance in real time without the need for deep technical knowledge nor human resources.

The aim of the project is not to provide a simple chatbot, but a complete system, with an indexed corporate knowledge base (with a vector database), a document ingestion pipeline to enable structured knowledge growth, mechanisms for chunking and text splitting, a natural language query interface, and a source citations and retrieval system to show traceability.

1.2.1 What is a RAG?

Retrieval-Augmented Generation (RAG) is an advanced AI architecture that combines the knowledge retrieval capabilities of information retrieval systems with the language generation and understanding power of large language models (LLMs).

In a typical RAG pipeline:

1. **User Query**: The user submits a query or question in natural language (e.g. "*How do I update a product record in SANITY?*")
2. **Retrieval Phase:** The system searches for a knowledge base (e.g. SANITY documentation, training materials…) for the most relevant documents or text fragments using embedding-based similarity search.
    a. Text documents are converted into vector embeddings - numerical representations that capture their semantic meaning
    b. These vectors are stored then in a vector database, which enables an efficient retrieval of the most semantic related content.
3. **Augmentation Phase:** The retrieved information then is blended with the original question and passed as input to a Large Language Model (LLM) such as Llama or GPT
4. **Generation Phase**: The LLM then generates a human-like response, taking into account the context that references the original source of the response.

This approach allows the model to generate precise, coherent and up-to-date responses, avoiding hallucinations and providing understandable answers supported by the real company data.

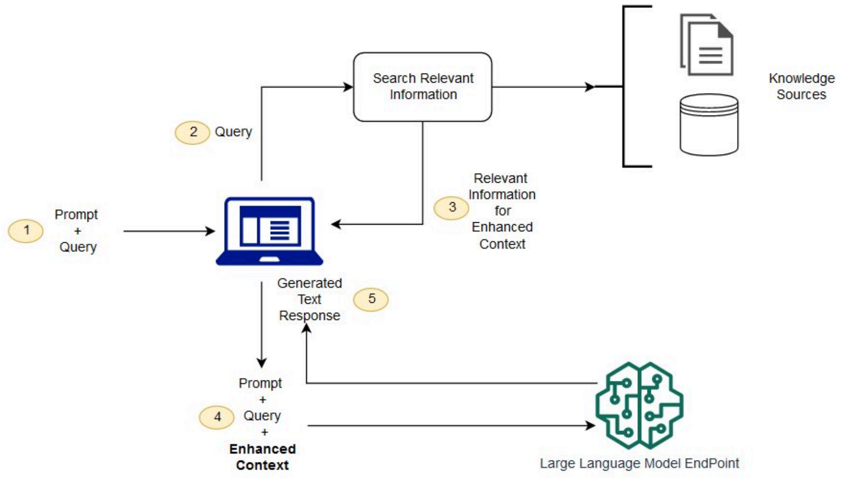

Search Relevant Information
Knowledge Sources
2 Query
1 Prompt + Query
3 Relevant Information for Enhanced Context
Generated Text Response 5
4 Prompt + Query + Enhanced Context
Large Language Model EndPoint

Figure 1. Conceptual flow of the RAG pipeline with LLM

(Image source: https://docs.aws.amazon.com/images/sagemaker/latest/dg/images/jumpstart/jumpstart-fm-rag.jpg)

2. Implementation

This project involves several data science components, including the database structure creation, data ingestion pipeline, retrieval pipeline, topic clustering and semantic structuring, frontend design, and evaluation and testing.

As part of the initial implementation, a **RAG pipeline** was developed to enable natural language queries over SANITY documentation. PDF texts were extracted and converted into semantic embeddings using SentenceTransformer (all-MiniLM-L6-v2) and indexed with FAISS for efficient similarity search. User queries are embedded and compared against the index to retrieve relevant fragments, which are then provided as context to a generative model (Flan-T5-base) to produce concise, context-aware answers. This initial RAG served as a **PoC (**proof of concept), laying the foundation for a subsequent, improved RAG pipeline.

2.1 Data Ingestion Pipeline

This is an essential part of the RAG, and we had to make many different design decisions to get to the best approach. We finally decided to create two different ingestion pipelines, allowing for future developers to decide which is best for a production application.

2.1.1 Cloud ingestion

The core idea of this final pipeline is to upload the knowledge base to a shared database in the cloud. In our case we were using a server hosted in Supabase. This allows for a more scalable and centralized approach, where all users access the same database, therefore, ingestion wouldn’t depend locally on any specific computer of one of the developers of this system, and anyone with access could view the folder that stores the documentation, and add or remove documents as desired. The implementation of Supabase was a major step, as it enables several additional features that will be mentioned later, such as citations with documents links, permission policies…

1. **Connect to Storage bucket**

We open a connection to a storage bucket in the Supabase project, which we access with our private key and url. There, we will find all of the documents that we want to ingest. Developers will upload into this bucket all the information that needs to be ingested as part of the knowledge base, or remove the corresponding files that are not meant to be considered.
After the connection is established, we start processing each document individually following the same process.

The first step is to locally download the file on the computer that is running the ingestion, this way the file can be processed to compute its hash and the corresponding chunks and embeddings. This is done by using a temporary directory *‘temp/’* that is deleted later .

2. **Compute content hash - Detect duplicates**

In order to avoid uploading duplicate content to the database, since it is common for documents already in the database to be in the storage bucket, the first thing is to check the already ingested documents to avoid processing them again by comparing their hash, stored in the *‘documents’* table, with the hash of the document that is to be ingested, this way we check if it already exists in our knowledge base. If it does, we skip its ingestion, as it would be a duplicate.

Each time we ingest a new document, its computed hash (SHA256) is stored with it.

3. Process document with Docling

Docling is a powerful library for document processing developed by IBM. It allows for complex features like image processing and OCR models. In order to get the most information out of every pdf file in our bucket, including images is an essential step.

We first export the file to markdown, where we add all of our text, and include placeholders in lieu of the actual images. Then, using RapidOcr, we extract image annotations and overwrite them in place of the placeholders. This process is what makes this pipeline more complex and computationally demanding, since it requires the use of Graphic Processing Units in order to perform inference with the OCR model.

With this, we obtain documents where, instead of images, we have all the information in readable format.

4. Chunk document

Once we have the document information in this text format, we chunk it. This is essential to avoid presenting all the information of a document to the model at once, instead it is divided into segments, from which embeddings are later computed and indexed in the vector database, allowing a highly efficient search and organization. Embeddings are computed by passing each chunk through the model “text-embedding-3-small”.

There are many different ways to split the texts, and, after analyzing the structure of our data, we saw that most of the documents were presentations. Because of this, because of its improved behaviour compared to other chunking approaches, information was naturally split in slides, so we decided that the best way to split our data was with the page separators.

Most of the pages were composed of images and a couple of short paragraphs, so distribution of the chunk length in our database is centered and short-tailed. The implemented wrapper specifies the chunk size and defines the chunk overlap, the latter is intended to prevent context loss and improve the model’s understanding.

5. Detect Updated Versions

When no duplicate of the document to be ingested is found in the database, it is considered new and is processed (chunking & embedding). However, before ingesting it into the database it is necessary to check if it is an updated version of an existing document.

In order to detect this we look for similar documents by using the filename (pg_trm for finding similarity) and the chunk embeddings average for each document (this captures an overall information of the document). Both scores are limited with a threshold and then combined, so an updated document will be detected if it exceeds these values. In this case the older version in the database is moved to another folder "archived/" and the new one is ingested to the main folder.

6. Insert new document

Once we have all the information and passed all the checks, we are ready to safely upload our document to the database. With it we include the content, hash and average embeddings. We also generate the document URL, which points to the PDF file in the Supabase bucket, this is useful to give access to the doc from the citations. For each document, we also upload its chunks and chunks' embeddings to the corresponding tables in the database.

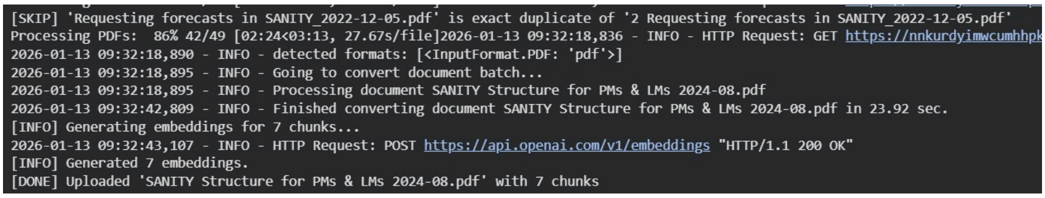
7. Data consistency and deletions

A Cascade Deletion function is implemented to allow the deletion of a document and automatically all its related chunks and chunk embeddings from the database directly, for cases when a document is to be removed from the system's knowledge base.

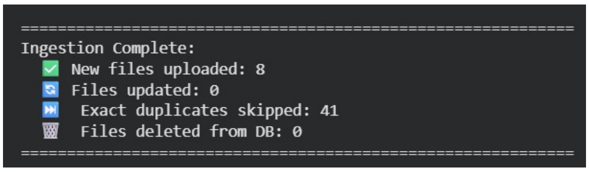
A Sync Deletions is also applied, which synchronizes the bucket with the database, automatically ensuring that each time the ingestion is executed all documents no longer in the bucket are removed from the database (cascaded deleted). In this way removing a document from the bucket (which is quite simple and straight-forward) means that its information is no longer intended to be used by the agent for generating answers.

8. Operational Logging

Finally, in the terminal a large amount of information and logs is provided, allowing the user to see how tube ingestion went; how many files were successfully uploaded, how many were skipped (due to duplicates) and how many were updated. Loading and processing times, number of chunks per document, and other relevant information for debugging and insights is also provided.



```
[SKIP] 'Requesting forecasts in SANITY_2022-12-05.pdf' is exact duplicate of '2 Requesting forecasts in SANITY_2022-12-05.pdf'
Processing PDFs:  86% 42/49 [02:24<03:13, 27.67s/file]2026-01-13 09:32:18,836 - INFO - HTTP Request: GET https://nnkurdyimwcumhhpk
2026-01-13 09:32:18,890 - INFO - detected formats: [<InputFormat.PDF: 'pdf'>]
2026-01-13 09:32:18,895 - INFO - Going to convert document batch...
2026-01-13 09:32:18,895 - INFO - Processing document SANITY Structure for PMs & LMs 2024-08.pdf
2026-01-13 09:32:42,809 - INFO - Finished converting document SANITY Structure for PMs & LMs 2024-08.pdf in 23.92 sec.
[INFO] Generating embeddings for 7 chunks...
2026-01-13 09:32:43,107 - INFO - HTTP Request: POST https://api.openai.com/v1/embeddings "HTTP/1.1 200 OK"
[INFO] Generated 7 embeddings.
[DONE] Uploaded 'SANITY Structure for PMs & LMs 2024-08.pdf' with 7 chunks
```



```
==========================================================
Ingestion Complete:
  ✅ New files uploaded: 8
  🔄 Files updated: 0
  ⏭  Exact duplicates skipped: 41
  🗑  Files deleted from DB: 0
==========================================================
```

9. Automated Ingestion

In order to facilitate and automate the ingestion process, avoiding human intervention and reducing the workload on the team, several alternatives were considered, such as using Github Actions, Supabase Storage functions or triggers. However, the most effective and simple option during this development phase was to use the task scheduler on one of the teammates' computers. This ensures that ingestion runs at regular intervals keeping the database and the bucket stored files synchronized.

2.1.2 Local ingestion

Data in pdf format is ingested using *PyPDFLoader*. This library reads page by page each document. The ingestion pipeline checks first if there is a document already uploaded in the database with the same name to avoid duplicates. In that case, the system notifies the user who is trying to insert new documents that a file with that name already exists in the database, and if it is a different document, it must change before uploading it. Each new document ingested is then divided into smaller chunks of size 1000 characters with an overlap between chunks of 200 characters to avoid losing semantic coherence. Afterwards, these chunks are converted to their corresponding embeddings. Data is then stored and indexed in Supabase in three different tables: documents (file name, summary, body, topic), chunks (document id, chunk index, body and topic id) and chunk embeddings (chunk id and embedding).

This was also useful as a mechanism of quality control in the local environment that allows to process a subset of PDF files and save the extracted text in .txt files for manual inspection purposes. This allows to verify the quality of extraction and estimate processing times before running the full ingestion. Additionally, the local mode is also designed for experimentation and debugging without affecting the cloud workflow or the main database index, allowing faster and less costly iterations.

2.2 Retrieval Pipeline

The retrieval system includes a semantic/vectorial search combined with a search by exact words. Firstly, the user query is transformed into an embedding with the same model (text-embedding-3-small) and dimension. A vectorial search takes place using cosine similarity at the same time as a textual search with pg_trgm. This similarity searches are by comparing each chunk to the user’s query. Both methods, semantic (vector/embedding) and lexic (text) are combined using a weighted fusion, giving slightly more weight to semantic search ($\alpha = 0.6$). When a topic is provided the search is restricted to that topic (each chunk is classified to a topic), and topics can be automatically detected from the user’s query and the pre-defined topics, or by asking the user through the frontend to choose a topic, offering the available options. Focusing on a topic enhances the performance of the agent.

The results are sorted and limited to the top k (using $k = 8$), which are the more relevant chunks for that query and are returned in the citation format JSON. The context that the agent will receive is created with these findings. The chunks found are cited in the output answer of the agent.

A relevance threshold is implemented to avoid retrieving chunks that despite being the most similar are not even relevant (we want the agent to tell us that it doesn’t have enough information in this case).

2.3 Topic Clustering and Semantic Structuring

To enhance the organization and retrieval of information within the document corpus, a topic clustering module was implemented. The primary objective of this step is to automatically identify and group semantically related documents into coherent topics. This provides a structured representation of the knowledge base, which later supports more efficient retrieval, targeted embeddings, and the possible deployment of specialized agents per thematic area.

The process begins with the extraction of text from PDF documents and division into chunks, the smallest unit of semantic analysis. Each chunk is preprocessed with the objective of normalizing language, avoiding lexical noise that does not add value or significance and making easier a vector representation that is consistent along all data. In order to do this *spaCy* (*en_core_web_lg*) is used, a large English language model capable of lemmatization and linguistic normalization. During this stage, stopwords and non-alphabetic tokens are removed to retain only the most meaningful lexical units.

After preprocessing, the cleaned documents are vectorized using the Term Frequency–Inverse Document Frequency (TF-IDF) method. The TF-IDF matrix serves as input for the Non-Negative Matrix Factorization (NMF) algorithm, which decomposes the document-term matrix into latent topic representations.

To determine the optimal number of topics, multiple NMF models were trained with varying component counts (from 2 to 14). Each configuration was evaluated using the **Silhouette Score**, a statistical measure that assesses how well-separated the resulting clusters are. The number of topics yielding the highest score was selected as the optimal configuration.

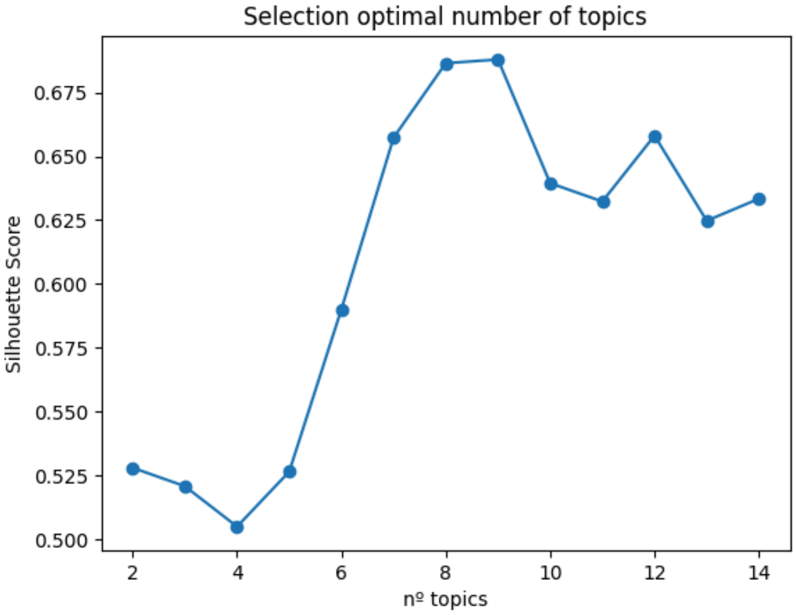


Figure 2. Silhouette Score of different number of topics

To guarantee efficiency, the trained model is serialized and stored locally. In this way, unnecessary retrainings are avoided and coherence is ensured, as the system can automatically charge the existing model if available. If there is not any model available, it is then retrained starting from the already existing chunks.

For interpretability, the nine most representative terms of each topic were extracted. To provide meaningful and human-readable topic names, the list of top keywords for each topic was sent to an LLM (*GPT-4o-mini*)**,** which generated concise and logical topic labels. Moreover, the top 10 chunks representative of that topic, together with the topic name and keywords were sent to the LLM to provide short descriptions. The final clustering produced **nine distinct thematic areas**, exposed below:

- **Business Development Deal Management**
  Keywords*: "deal", "bd", "car", "sanit", "request", "manager", "ed", "partner", "contract", "operations"*
  Description*: This topic encompasses documents related to the management and approval processes of business development deals, including contract requests, compliance checks, and operational responsibilities for BD Managers.*

- **Assumption-Based Forecasting in SANITY**
  Keywords*: "forecast", "assumption", "channel", "update", "base", "create", "request", "size", "sanity", "strength"*
  Description*: This topic encompasses documents related to the creation and updating of assumption-based forecasts within the SANITY system, detailing processes, tools, and methodologies for effective forecasting management.*

- **Packaging and Unit Sizes**
  Keywords*: "pack", "size", "primary", "unit", "pc", "packaging", "selling", "count", "mg", "box"*
  Description*: This topic encompasses documents related to primary and secondary packaging units, sizes, and types for various dosage forms, including inhalation, oral solids, and liquids. It details the strength units and configurations for effective product representation.*

- **SPC Submission Process Overview**
  Keywords*: "spc", "submission", "site", "submissions", “request", "business", "evaluation", "use", "structure", "stage"*
  Description*: This topic encompasses documents related to the SPC submission process, including business processes, site requests, evaluation stages, and transparency measures for managing submissions effectively.*

- **Budget Management Overview**

  Keywords*: "budget", "cost", "snapshot", "planning", "grid", "overview", "dashboard", "initiate", "conga", "benefit"*

  Description*: This topic encompasses documents related to budget planning, cost management, and financial reporting within the SANITY system, including guidelines for initiating budgets, utilizing the Conga grid, and understanding key benefits and snapshots.*

- **Sandoz Product Target Documentation**

  Keywords*: "product", "target", "sandoz", "strength", "dosage", "selling", "api", "button", "create", "click"*

  Description*: This topic encompasses documents related to the creation and management of Sandoz Product Targets, detailing aspects such as API, dosage forms, strengths, and related processes within the SANITY system.*

- **Fast Track Selection Process**

  Keywords*: "selection", "fast", "track", "content", “forecast", "locale", "pre", "issue", "zero", "outcome"*

  Description*: This topic encompasses documents related to the fast track selection process for content and forecasting, including prerequisites, outcomes, and guidelines for managing selections in various locales.*

- **Opportunity Assessment and Evaluation Process**

  Keywords*: "record", "meeting", "assessment", "opportunity", "click", "evaluation", “opp", "document", "agenda", "psg"*

  Description*: This topic includes documents related to the evaluation and allocation of opportunities, detailing processes for creating meeting agendas, documenting assessments, and managing opportunity records.*

- **Project Management and Country Details**

  Keywords*: "project", "country", "sanity", "plm", "launch", "cdl", "use", "detail", "portfolio", "management"*

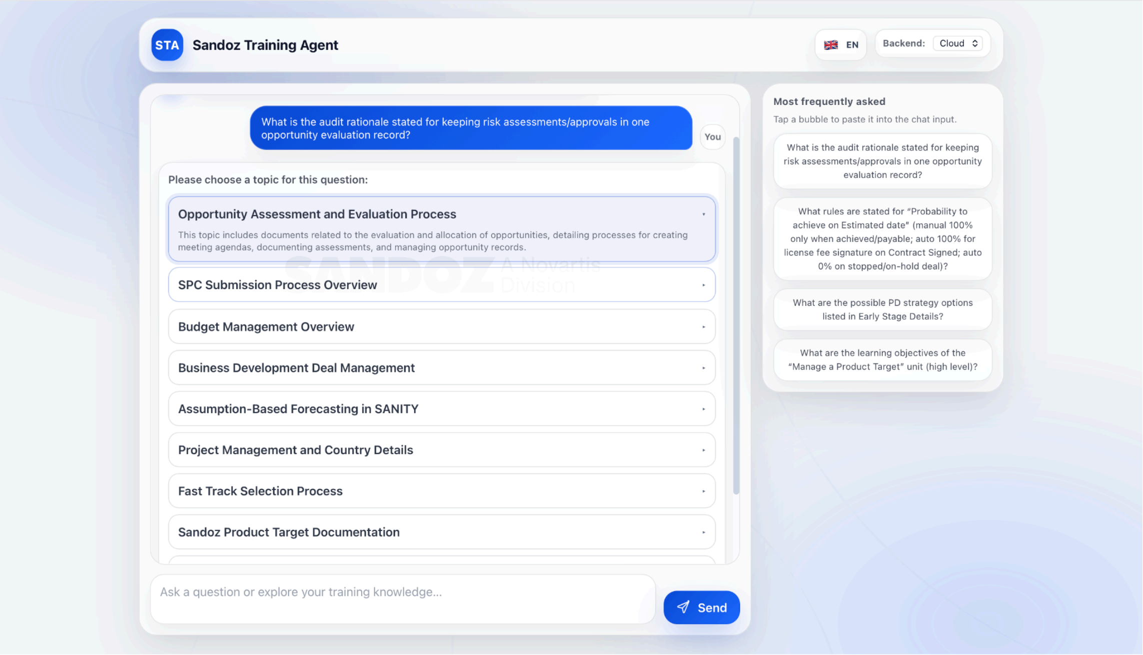
  Description*: This topic encompasses documents related to project management processes, country-specific details, and portfolio management, focusing on project phases, selection criteria, and launch management within the SANITY framework.*

Given the previous process, we started to create a multi-topic aware agent, since any document could handle more than one topic. First, we made some changes to the database: *topics* table (id, name) is created, a topic column for the *chunks* table is added and the *documents* table stores topics in the form of lists.

Although this generation is automatic, this system also contemplates the intervention of a human expert. If the model doesn’t generate the best or the expected descriptions or topic names (as sometimes these are not perfect), humans can improve the performance by giving better descriptions or more clear names to the topics, or even specifying the keywords of a topic. Therefore, a mechanism of exporting and importing topics to and from a modifiable CSV file has been implemented. If any changes are detected during the implementation, they are propagated to the database where topics are stored.

Then, given the topics, we trained a model for assigning a single topic for each chunk according to its maximum probability of belonging to a certain topic. Finally, these topics are propagated to the documents each chunk belongs to.

Finally, when the user consults something, the system tries to find the topic to which it is related by preprocessing the query in the same way as the chunks to identify the most probable topic and increase accuracy in the answer. If no topic is identified, the system explicitly asks the user to choose one of the available topics.



What is the audit rationale stated for keeping risk assessments/approvals in one opportunity evaluation record?

Please choose a topic for this question:

Opportunity Assessment and Evaluation Process

This topic includes documents related to the evaluation and allocation of opportunities, detailing processes for creating meeting agendas, documenting assessments, and managing opportunity records.

SPC Submission Process Overview

Budget Management Overview

Business Development Deal Management

Assumption-Based Forecasting in SANITY

Project Management and Country Details

Fast Track Selection Process

Sandoz Product Target Documentation

Ask a question or explore your training knowledge...

Most frequently asked

Tap a bubble to paste it into the chat input.

What is the audit rationale stated for keeping risk assessments/approvals in one opportunity evaluation record?

What rules are stated for “Probability to achieve on Estimated date” (manual 100% only when achieved/payable; auto 100% for license fee signature on Contract Signed; auto 0% on stopped/on-hold deal)?

What are the possible PD strategy options listed in Early Stage Details?

What are the learning objectives of the “Manage a Product Target” unit (high level)?

Figure 3: Topic picker

This clustering module, not only enhances the answer but also the internal organization of knowledge enabling a better understanding of the domain and future scalability to specialized multi agents per topic.

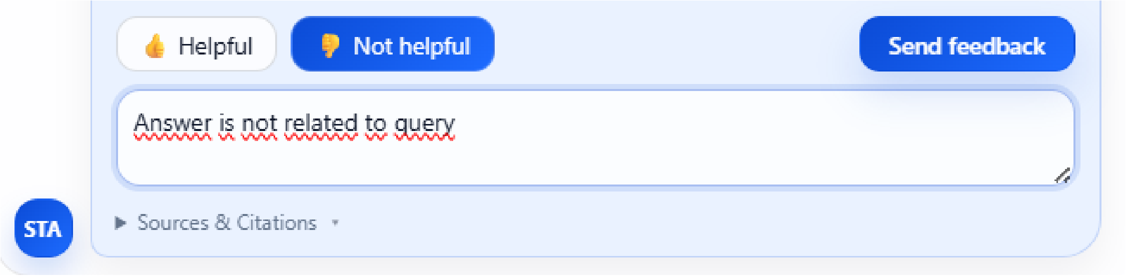
2.4 Feedback System

In a retrieval-augmented generation workflow, the quality of answers depends on both the retrieval process and the generation step. We decided to take this a step further and guide the agent by giving it the ability to learn from real user experience, so, as a result, we implemented a feedback system.

The feedback system is designed to be simple for users yet powerful for backend analytics. Here’s how it works:

1. When the agent answers you can send the following feedback:
    a. If you were satisfied with the answer: “**Thumbs up**”, which is equal to +1
    b. If you were not satisfied: “**Thumbs down**”, which is equal to -1. When you choose this option an optional comment box appears for you to write down what was wrong with the answer or to leave any suggestions.

   Then, you have to press the “*Send feedback*” button for it to be saved.



Answer is not related to query

2. Each time you send feedback, a new instance is created in the *Feedback* table in Supabase, which stores relevant information such as the query, answer and citations, the rating (+1,-1) and optional comment, and some other valuable attributes such as model used, the query embeddings…

How does the feedback improve the agent?

Adaptive prompting (*adaptive_prompting.py*) was introduced to make the RAG system context-aware and performance-driven. Instead of using static prompts for every query, the system dynamically adjusts the prompt based on:

- **User feedback history** (**positive examples**): Each past query with positive feedback is stored along with its embedding. For the current query, the system generates an embedding and compares it against stored embeddings. Only examples above a predefined similarity threshold are considered relevant, ensuring that the retrieved positive examples are contextually close to the user's current question, avoiding irrelevant or misleading patterns. The system then retrieves a limited number of examples defined by 'max_examples' (e.g., 2–3) to keep the prompt concise.

  Once positive examples are retrieved, they are transformed into a structured section within the prompt as: *Original Question + Snippet of the Answer + Similarity Score.* These examples are placed before the user's question in the final prompt, creating a "few-shot" learning scenario. This guides the model by showing concrete patterns of success, improving its ability to generate relevant and high-quality responses.

- **Overall satisfaction metrics:** The system also monitors global satisfaction metrics based on the ratings (+1, -1) of the interactions. If the satisfaction rate falls below a certain threshold and there is enough feedback volume, the system interprets this as a signal that responses may be unclear or incomplete. In such cases, the prompt includes an additional instruction like: *"IMPORTANT: Users have reported dissatisfaction. Be more clear and specific."*. This encourages the model to produce answers that are more detailed and precise.

Although these features are included in the project, we have not implemented them for testing since evaluating the correctness of the answer requires a high degree of understanding of the business, and by providing wrong feedback we could worsen the performance of our agent. Nevertheless, for future development, we have included an *is_verified* column (Boolean) in the *Feedback*, so an administrator with sufficient knowledge of the business can go through the feedbacks and switch to *True* the ones that are considered to be valid, which would be used later for the dynamic learning of the RAG.

2.5 Conversational Memory

Alongside the adaptive selector, which enables the agent to respond more effectively and tells it to be more clear when feedback is generally negative, we also optimize the agent’s behaviour and answers using a conversational memory.

We define a session memory in RAM per user, saving its last ‘max_turns’ turns (query and answer), discarding the older ones to limit the context and not overloading RAM.This allows the agent to retain contextual information of the conversation with a user from previous interactions. This context is added to the prompt among other information as the instructions, few-shot context, user’s query… so the agent will use it to give more coherent and relevant responses.

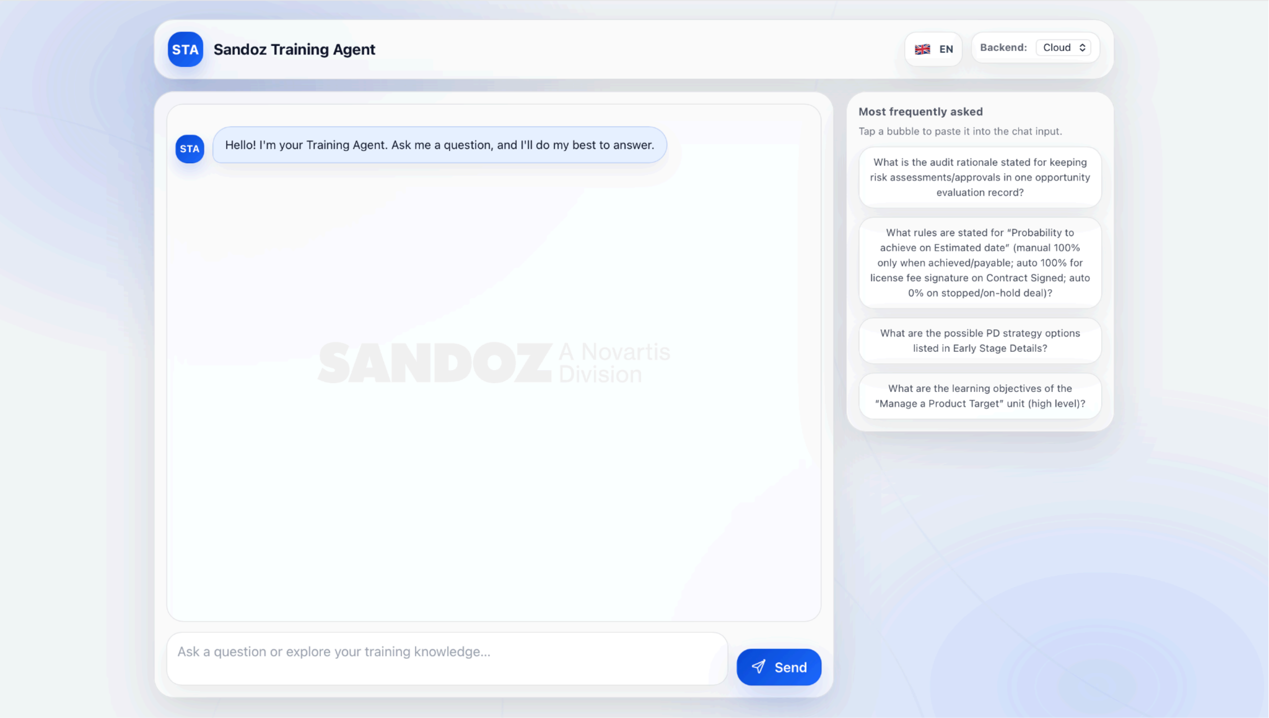
2.6 Frontend Design

Frontend is developed as a static page with HTML, Javascript and TailwindCSS. It is designed as a chat for the user to be able to easily ask questions and maintain a conversation with the agent.

2.6.1 Layout and interface

The main structure of the user interface is organized into two columns: the main chat and a FAQ dashboard. The main chat is a common chat with messages in bubble format distinguishing between user and assistant. Besides, for the assistant messages, the answer is divided into the main answer with the information and a dropdown with the sources and citations. From that dropdown the user is able to open the documents in the exact pages where the information is taken without interfering with the main message.

The interface also includes a language selector enabling to exchange the interface between spanish and english. This switcher does not affect the content of the answers, it only switches static placeholders such as the button “Send” / “Enviar”



STA Hello! I'm your Training Agent. Ask me a question, and I'll do my best to answer.

Most frequently asked

Tap a bubble to paste it into the chat input.

What is the audit rationale stated for keeping risk assessments/approvals in one opportunity evaluation record?

What rules are stated for "Probability to achieve on Estimated date" (manual 100% only when achieved/payable; auto 100% for license fee signature on Contract Signed; auto 0% on stopped/on-hold deal)?

What are the possible PD strategy options listed in Early Stage Details?

What are the learning objectives of the "Manage a Product Target" unit (high level)?

Ask a question or explore your training knowledge...

Figure 4: User interface

2.6.2 Connection with API flo

The flow is simple but efficient. The frontend incorporates a selector to change the execution, cloud or local, without changing the interface or reloading it. The user can write a query or even select one from the FAQ dashboard. This message is sent to a single endpoint */ask* together with the selection of the backend mode. The request is processed and routed to its corresponding agent (cloud or local), and the response rendered in UI.

For the cloud backend, if the system is unable to relate a topic to the user's query, it activates and shows a topic picker showing the topics names and descriptions. Finally, the user is asked to rate the content of the answer by clicking any of the two simple buttons and optionally adding a comment. The information is sent to a corresponding endpoint */feedback*.

2.7 Prompt Assembly

The response generation logic is designed to transform each user request into a structured and controllable LLM call by combining:

- A set of system-level instructions (Formatting, language, citation rules and how to behave)
- Most relevant retrieved passages (chunks) as numbered fragments to enable grounded citations

- Few-shot guidance from positive feedback examples from similar queries (as an adaptive orientation)
- Conversational memory (recent turns, enables the agent to know the context)
- The user’s query
- Optional customizable attributes produced by auxiliary modules (user role, time budget, difficulty level, selected topic, feedback type…)

To enforce Database grounded responses the assemble instructions explicitly constraint the model to use only the provided passages and to return an “insufficient information” response when the retrieved context is not relevant enough. This gives reliability and reduces the risk of hallucinations, as the model is prevented from generating answers based on external knowledge or assumptions not present in the actual documentation. It is key to assure factual consistency and trust in the system’s output.

This format is defined at API level to standardize how the backend and frontend communicate. Finally the generated answer is post-processed and returned together with a structured citation metadata with direct document links.

3. Local Agent

The objective of the local agent is to build a Trainer Agent able to answer the users’ queries without sending prompts or sensible information to external services such as OpenAI. This agent runs locally in the user’s computer through LM Studio and generates local embeddings through Sentence-Transformers. The same user interface is used but an alternative backend is executed. It is important to note that this agent is not 100% local for simplicity and in order to work collaboratively. Supabase is used as the centralized database. In Supabase we store documents, chunks, and chunk embeddings to semantically search for context. Nevertheless, by executing the LLM locally, the content of the queries, the context retrieved (internal and confidential information) and the prompt and reasoning of the model are not shared with third parties. The local agent has the following structure and components.

3.1 Model and server:

In order to build the local agent, LM Studio has been used. It exposes an OpenAI compatible endpoint which means it has the same structure as the cloud one. The request is sent to *http://localhost:1234/v1/chat/completions* with the model used, the whole message (system prompt and user content), temperature and a maximum number of tokens.

Qwen2.5-1.5B-Instruct-MLX-4bit, a small model (1.63GB and 1.5B parameters) has been used in order to reduce latency while obtaining good results. It is optimized for instructions and follows the prompt well. However, it can be easily changed as it is placed as an environment variable and the model is charged in LM Studio.

3.2 API orchestration

The same endpoint is used for both cloud and local. Hence, a similar structure is followed. This endpoint receives information from the frontend and orchestrates the routing. For the local branch, it retrieves the useful chunks to use as context to answer the query (*search_kb_local(query, top_k=6)*). Firstly, it checks that the query is not empty and calls the function that generates the embeddings (*_embed_query_local(query)*). With those local embeddings, it executes a query to retrieve the best snippets ordered by vector distance (*vector_search_local(qvec, k=8)*). If there are no snippets found, the endpoint returns "I have not found enough information" without calling the LLM. In this way any hallucination is avoided. Then another function is called (*answer_with_local_rag(query, passages)*) which has two main responsibilities: to create the context and to define the prompt. The context generated is received by the model as the only possible source of information. The prompt is created with a strict system prompt and the user content (the query). Finally the LLM local model is called (l*ocal_chat(system_prompt, user_content)*) and an answer is generated and sent to the frontend.

3.3 Local Embeddings

The local embeddings are computed using the model *all-MiniLM-L6-v2* of *sentence-transformers.* They have 384 dimensions which is reasonable for running locally but still stable. Embeddings are generated in two key parts: during the ingestion and for the users' queries. For the ingestion an embedding is calculated for each chunk and an average for the whole document. For the query it is converted to an embedding and used as the search vector.

3.4 Vectorial Search

The vector search is done over Supabase for the tables storing local information (*local_*)*. The retrieval uses vector distance with *pgvector.* It is ordered by distance and returns a top-k. Besides, a score of similarity (1- distance) is computed and shown in the frontend.

3.5 Frontend

The user interface remains the same. There is a selector in the top right corner that enables the user to choose whether he wants to use a fully cloud agent or a local agent. The frontend sends that information to an API endpoint that decides which mode to use. In each request each information is serialized making it simple and efficient to switch modes at any time without breaking the flow. As for the cloud agent, it shows the citations and sources and a link to access each of them.

4. Results

4.1 Evaluation & testing

For the testing of the RAG system, our goal is to measure **whether the assistant cites the correct evidence** for each question, since evaluating other metrics such as the answers correctness and completeness would require a much deeper comprehension of all the PDFs and company’s workflows and data, which we do not have. For each query, the system returns a ranked list of citations (sources), and the evaluation checks (1) if the correct PDF is cited and (2) for “local” questions, whether the cited page matches the expected evidence location.

To reflect two different evidence needs, questions are classified into two categories:

- **GLOBAL_SUMMARY**: the question requires a summary across multiple parts of a document; only PDF-level evidence is evaluated.
- **LOCAL**: the answer is expected to come from a specific page or page range; both PDF and page-level evidence are evaluated.

Why this metric?

As a result, **MRR-based** scoring was chosen because it directly captures the user-relevant behavior in evidence retrieval: how quickly the correct source appears in the ranked citations.

MRR assigns full credit when the first correct source appears at rank 1, partial credit when it appears later (e.g., 1/2 at rank 2), and zero if it does not appear in the retrieved list.

Alternative retrieval metrics were considered (e.g., Recall@K / Hit@K and Precision@K), but MRR was preferred as the primary component because it rewards both presence of the correct source and its rank position, which is important when only a small number of citations are shown to users. Recall@K is valuable to verify that relevant items appear within the top-K results, but it does not distinguish well between rank 1 and rank K, whereas MRR explicitly captures that difference.

How does the metric work?

For **global questions**, compute the **reciprocal rank** (RR_{doc}) of the first cited PDF that matches the expected PDF:

- If the correct PDF appears at rank r: $RR_{doc} = 1/r$.
- If the correct PDF does not appear in the returned citations: $RR_{doc} = 0$.

The GLOBAL score is then the mean of RR_{doc} over all global questions, i.e., $MRR_{doc.}$

For each **local question**, the score is designed to be simple and interpretable:

1. **Document component**: compute RR_{doc} exactly as above (based on where the correct PDF first appears).
2. **Page component**: a binary bonus *page_hit* is applied:
   - ○ *page_hit* = 1 if at least one citation to the correct PDF includes a page number that falls within the expected page range.
   - ○ *page_hit* = 0 otherwise.

The local score is then:

$$\textbf{(1)}\ Score_{LOCAL}(q) = RR_{doc}(q) \cdot ((1-\beta) + \beta \cdot page\_hit(q))$$

where $\boldsymbol{\beta} \in [0,1]$ controls how much extra credit is given for citing the correct page(s). This structure guarantees that citing the correct PDF is the core requirement, and citing the correct page is an explicit “plus” that can only help if the PDF is correct.

The **overall score** blends both categories:

$$\textbf{(2)}\ ScoreTotal = \alpha \cdot mean(Score_{LOCAL}) + (1-\alpha) \cdot mean(Score_{GLOBAL})$$

where **α** controls the weight of local vs global questions, which would be the proportion of local questions with respect to the total.

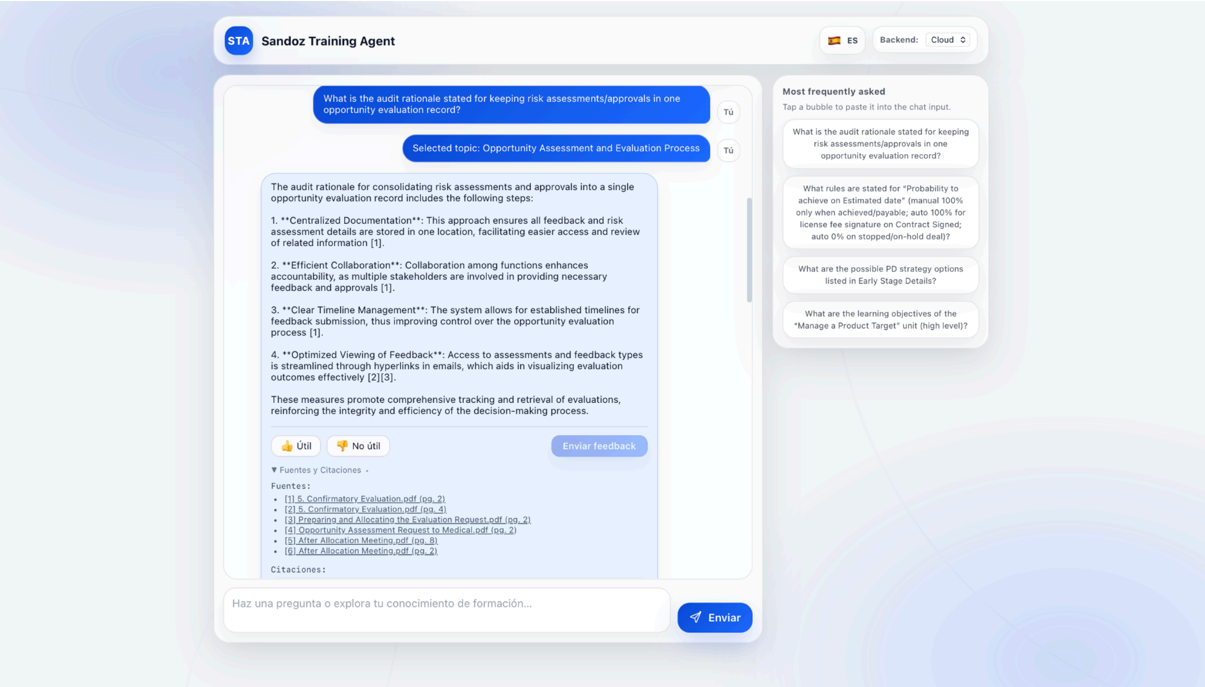
How was the testing carried out?

From Sandoz, we were given 11 prompt examples in the following form:

Difficulty	Question	Document	Pages of expected answer
Easy/Medium/Hard	*What does the document say about X?*	*"Doc1.pdf"*	*Pages 2-4* or *Summary of whole document*

Based on these questions, we expanded them so a solid metric can be built, also adding the new GLOBAL/LOCAL column according to the prompt needs, so it can be evaluated accordingly. Since citing the right PDFs/pages does not essentially verify answer correctness, or the other way around, we did the questioning process manually, computing the RR and page_hit of each individual question and simultaneously ensuring that the answers were "*correct"* (the model does not hallucinate)*,* in order to give our metric a real and valid meaning. Finally, from these individual scores, we computed the Total Score.

4.2 Results



What is the audit rationale stated for keeping risk assessments/approvals in one opportunity evaluation record?

Selected topic: Opportunity Assessment and Evaluation Process

The audit rationale for consolidating risk assessments and approvals into a single opportunity evaluation record includes the following steps:

1. **Centralized Documentation**: This approach ensures all feedback and risk assessment details are stored in one location, facilitating easier access and review of related information [1].

2. **Efficient Collaboration**: Collaboration among functions enhances accountability, as multiple stakeholders are involved in providing necessary feedback and approvals [1].

3. **Clear Timeline Management**: The system allows for established timelines for feedback submission, thus improving control over the opportunity evaluation process [1].

4. **Optimized Viewing of Feedback**: Access to assessments and feedback types is streamlined through hyperlinks in emails, which aids in visualizing evaluation outcomes effectively [2][3].

These measures promote comprehensive tracking and retrieval of evaluations, reinforcing the integrity and efficiency of the decision-making process.

Fuentes:

- [1] 5. Confirmatory Evaluation.pdf (pg. 2)
- [2] 5. Confirmatory Evaluation.pdf (pg. 4)
- [3] Preparing and Allocating the Evaluation Request.pdf (pg. 2)
- [4] Opportunity Assessment Request to Medical.pdf (pg. 2)
- [5] After Allocation Meeting.pdf (pg. 8)
- [6] After Allocation Meeting.pdf (pg. 2)

Citaciones:

Most frequently asked

Tap a bubble to paste it into the chat input.

What is the audit rationale stated for keeping risk assessments/approvals in one opportunity evaluation record?

What rules are stated for "Probability to achieve on Estimated date" (manual 100% only when achieved/payable; auto 100% for license fee signature on Contract Signed; auto 0% on stopped/on-hold deal)?

What are the possible PD strategy options listed in Early Stage Details?

What are the learning objectives of the "Manage a Product Target" unit (high level)?

Figure 5: Agent answer to a user query

Local Agent - Results Analysis

For the local agent we developed a small evaluation set of **16 questions** (all LOCAL, using formula (1)*) in the format shown above, and the results were the following:

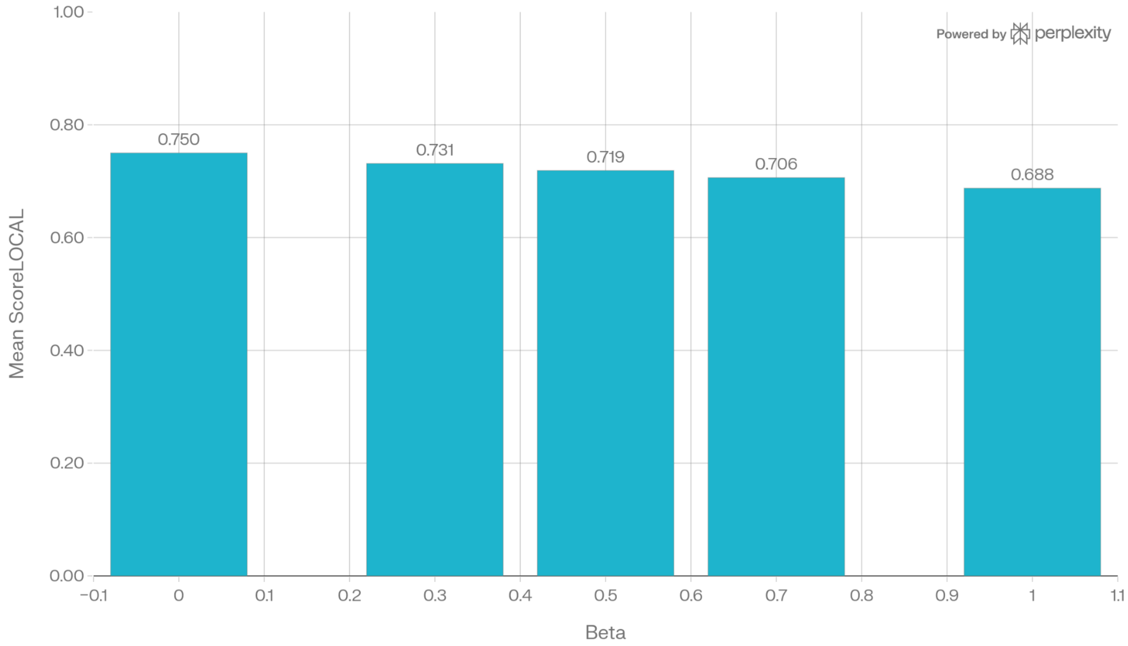


Figure 6: Local Agent - Mean Local Score declining as Beta Increases

The score behavior across β is stable and degrades only moderately as β increases, because most correct-document cases are also correct at page level, reaching its highest when β = 0 (document-only credit) with a 75 % of correct answers and decreases gradually as the metric demands stricter page hits.

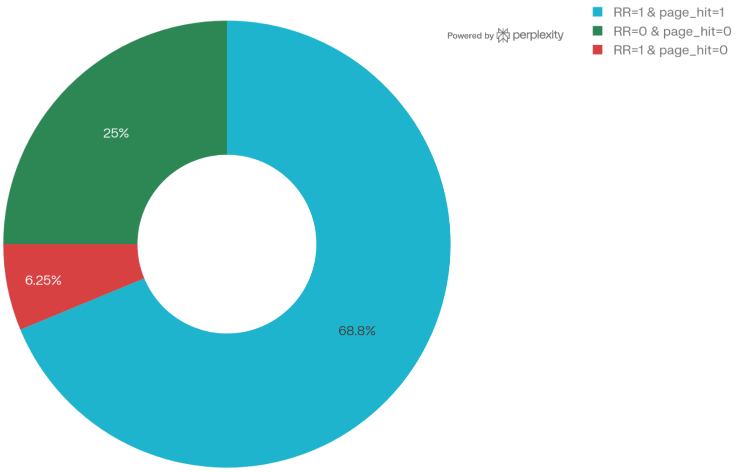


Figure 7: Local Agent - Outcome Breakdown (15 questions)

The table indicates that most *RR=1* examples also have *PAGE_HIT=1*, meaning the model is not only locating the correct PDF but also anchoring evidence at the correct page level with high consistency. This is exactly the behavior expected from a trustworthy document assistant, as users can verify the claim quickly and the system’s answers remain auditable.

Cloud Agent - Results Analysis

For the Cloud Agent, we developed an evaluation set of 200 questions (184 local & 16 global) and computed the score (2)*. The parameters were a fair β = 0.5 and α = 0.92

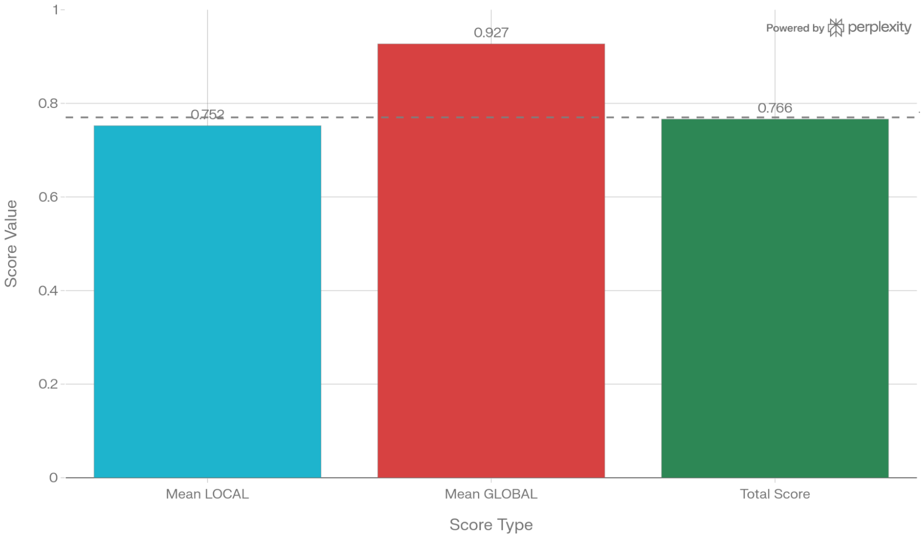


Figure 8: Cloud Agent - RAG Evaluation Scores (200 Questions)

The final blended score is 0.766 (76.6%) with $\beta = 0.5$. GLOBAL performs better than LOCAL because GLOBAL only requires “correct PDF somewhere in the ranked citations”, while LOCAL additionally rewards correct page-level evidence via PAGE_HIT, and that extra constraint inevitably reduces the mean score when $\beta>0$. The slightly better performance of the cloud agent with respect to the local might suggest that the use of topic-based approach (and other features exclusive of the cloud agent) improves the precision of the context retrieval.

The score is **conservative** by design—so the current score is meaningful. Because the score multiplies by RRdoc, any wrong-document retrieval collapses the score for that item to 0, which intentionally prioritizes factual traceability over fluent generation. Therefore, achieving a high total Score (around 80 %) in this framework is a stronger indicator of reliability than a metric that allows partial credit for unsupported answers.

5. Discussion

The choice of a RAG architecture over fine-tuning reflects the need to manage frequently changing SANITY documents efficiently. Fine-tuning would require retraining with every update, increasing costs and risking the use of outdated information. In contrast, RAG allows new or modified documents to be incorporated immediately through the ingestion pipeline. The hybrid retrieval strategy, combining semantic and lexical search, addresses practical challenges observed during testing, such as internal acronyms, exact field names, or process codes that embeddings alone sometimes fail to capture. Weighting semantic search more heavily improved contextual relevance, while lexical search ensured precise terminology was recognized. Additionally, chunking strategies were adapted to document structure: page-based chunking preserved coherence in slide-style documents, whereas fixed-size chunks offered more uniform embeddings and simplified processing for the local agent.

Topic modeling also proved valuable by imposing a higher-level structure on the knowledge base. Organizing documents and chunks into semantically related topics creates an interpretable layer beyond raw vector similarity, enabling more focused retrieval and supporting the future development of topic-specific agents. Labels generated by a large language model were subsequently reviewed, corrected, and validated by a company professional, ensuring that the resulting topics aligned with internal terminology and business processes. This human-in-the-loop step underscores that unsupervised methods alone may not suffice in enterprise contexts.

The system’s hybrid architecture further balances privacy, performance, and usability. Local models protect sensitive queries and internal context, whereas cloud models provide stronger language generation for complex queries. Smaller local models reduce latency and cost but may produce less nuanced answers, while cloud models improve fluency at the expense of governance and data exposure. A centralized vector database (Supabase) enables consistent knowledge sharing and collaboration while preserving privacy at inference time, demonstrating a practical compromise between accessibility and security.

Several lessons emerge from these design choices. First, a RAG system is only as effective as its retrieval layer; document ingestion, chunking, and indexing are critical to performance. Second, explainability through citations and page references is essential for enterprise adoption, especially in regulated environments. Third, local language models are viable for internal tools when privacy and compliance are prioritized over generative power. Finally, topic structuring is not just a retrieval improvement—it is a broader knowledge management strategy that can support scalability, improve interpretability, and guide long-term evolution of enterprise AI systems.

6. Conclusion

Due to SANITY's growing complexity we had to seek a solution that addressed this issue by designing a user-friendly and accessible agent, built with a Retrieval-Augmented Generation assistant with an intuitive interface that supports natural language, grounded in internal documentation and backed by traceable citations. This high-performance, reliable and scalable solution delivers an end-to-end system that includes cloud and local ingestion workflows with source linking and personalization modules, such as conversational memory and a feedback-ready schema.

Beyond its robust architecture the system’s value was validated through testing, achieving a reliable retrieval score that emphasizes factual traceability over mere text generation. By prioritizing verifiable, page-level citations, this agent serves as a trustworthy tool within

Sandoz's regulated environment. In addition, the implementation of topic clustering and the hybrid cloud-local deployment model provides a well-balanced approach for scalability and data privacy.

Overall, this project transforms SANITY from a passive and complex repository into an active, smart partner, establishing a solid foundation for the future deployment of specialized agent workflows.

7. Future Work

While the current system demonstrates strong performance and reliability, it remains for further enhancement and improvement, as well as deployment in a real, professional environment. It is still a development prototype, with a lot of possibilities to evolve and enhance. Some of the future improvements and implementations expected are:

- **Advanced Customization and Adaptative Behavior:**

Full deployment of the adaptive prompting module to allow the agent to adjust its tone, behavior and instructions based on real-time user satisfaction. Development of a profile-driven response mechanism that transforms the system into an individual agent for each user that adapts and learns from the specific needs and requirements of its owner. Each agent becomes a personalized companion, providing tailored support to every employee. The agent's behavior is uniquely aligned with its user and their role (it wouldn't provide the same technical depth to a junior as it would to a manager), identifying challenges within the project scope and delivering concise, high-level summaries and explanatory guides for each case.

- **Enterprise-Grade Automated Ingestion:**

Migration of the current automatic ingestion workflow, currently scheduled locally in a computer, to a dedicated Sandoz cloud server, in order to ensure a 24/7 autonomous ingestion pipeline that keeps the bucket and database continuously updated and synchronized, removing hardware and human dependencies and guaranteeing up-to-date of the latest corporate documentation.

- **Infrastructure Control and Database Migration:**

Migrating the backend infrastructure from the current Supabase cloud project to Sandoz's private servers. This shift aims absolute control and ownership of the system and database over the data lifecycle, reducing external dependencies and risk of hacking intrusion or data breaches, and ensuring all vector indices and document embeddings reside exclusively within the company's perimeter.

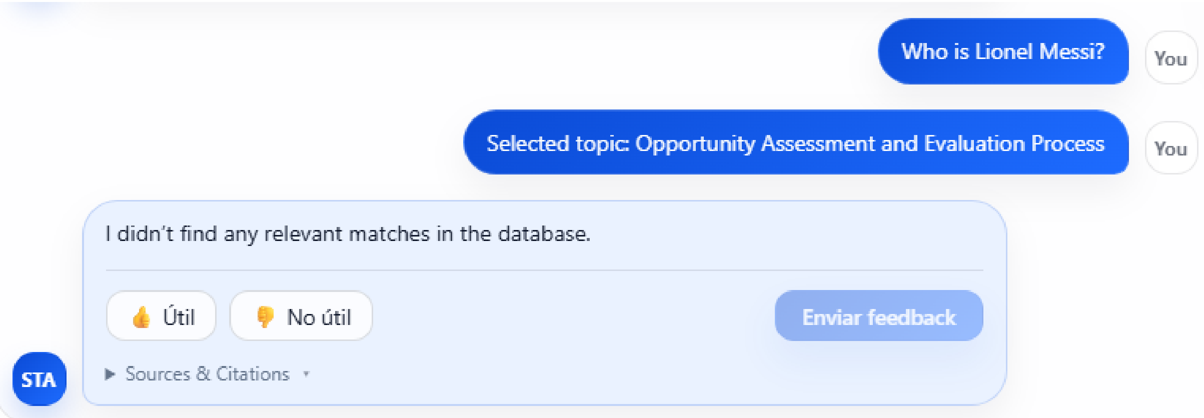
- **Granular Access Control and Governance:**

Implementation of a Role-Based Access Control system, with predefined permissions and security policies, as well as a hierarchy of authority. This includes defining strict policies and row-level security to ensure that users (for example managers vs external partners) only access documents and data passages relevant to their level and sector, reinforcing data privacy and regulatory compliance.

8. APPENDIX - Prompt Examples

Case 1: Prompt Irrelevant to Documents

When the RAG agent is given a prompt completely unrelated to the documents, it answers the following way. The agent has been proven to work well against hallucinations (the use of made-up or outside information).



Who is Lionel Messi?

You

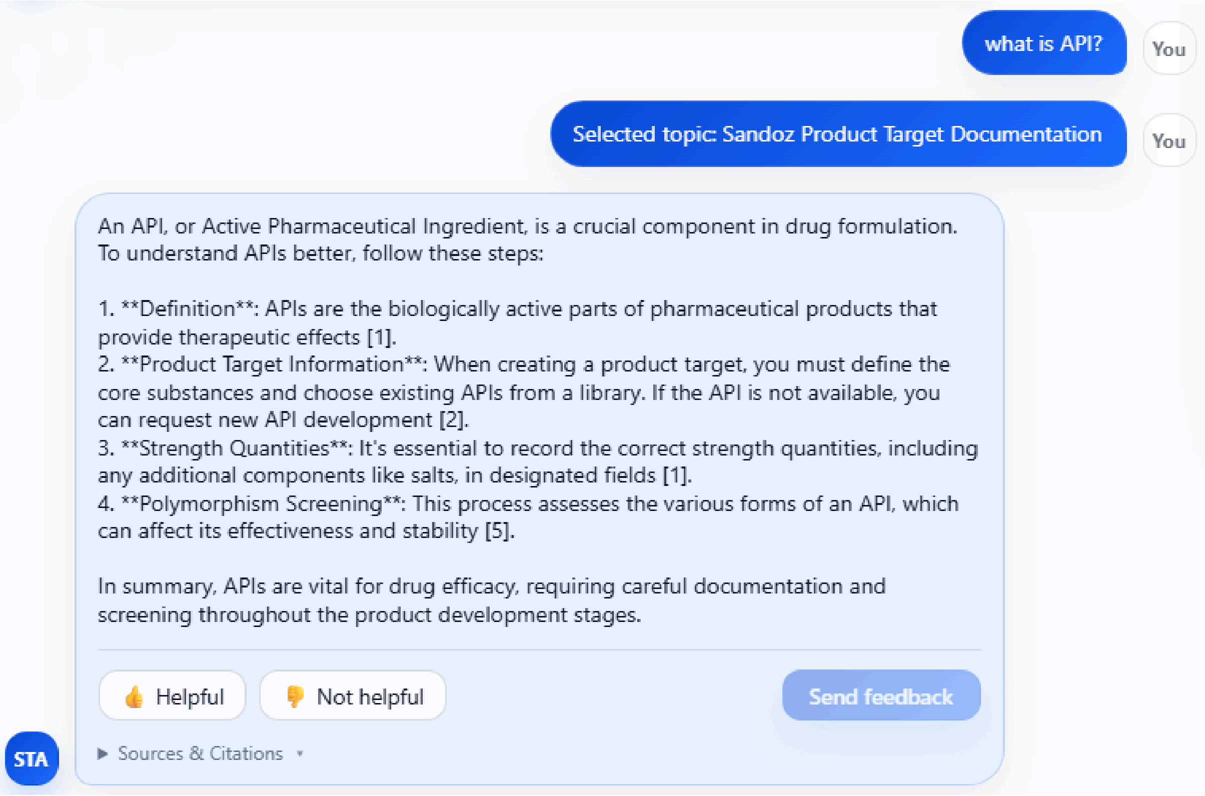
Selected topic: Opportunity Assessment and Evaluation Process

You

I didn't find any relevant matches in the database.

Case 2: API

In this case, we will ask the agent about the API. In SANDOZ context, API stands for Active Pharmaceutical Ingredient, a component in drug formulation. However API also has other meanings, being one of the most common in informatics: Application Programming Interface. Let's see how the agent answers:



what is API?

Selected topic: Sandoz Product Target Documentation

An API, or Active Pharmaceutical Ingredient, is a crucial component in drug formulation. To understand APIs better, follow these steps:

1. **Definition**: APIs are the biologically active parts of pharmaceutical products that provide therapeutic effects [1].
2. **Product Target Information**: When creating a product target, you must define the core substances and choose existing APIs from a library. If the API is not available, you can request new API development [2].
3. **Strength Quantities**: It's essential to record the correct strength quantities, including any additional components like salts, in designated fields [1].
4. **Polymorphism Screening**: This process assesses the various forms of an API, which can affect its effectiveness and stability [5].

In summary, APIs are vital for drug efficacy, requiring careful documentation and screening throughout the product development stages.

As you can see, the agent only takes into account the SANDOZ context, which is the one it has been given.