

Deliverable 2.3

Report on the Implementation of the Text Mining Module in GEMINI

Version March 2026



Authors**Company:** TNO**From:** Jonah Poort
Demetris Palochis
Pejman Shoeibi Omrani**Email:** jonah.poort@tno.nl
demetris.palochis@tno.nl**Project coordinator:** Laura Precupanu
laura.precupanu@tno.nl**Subject:** (Report on) text mining module for analyzing unstructured data in geothermal plants**Reviewers****Company:** WEP**From:** Floris Degener**Email:** floris.degener@we-p.nl**Date:** 11-03-2026

This project is executed with funding of the National Growth Fund of the Ministry of Economic Affairs, implemented by the Netherlands Enterprise Agency. The specific funding for project *Data gedreven optimalisatie van aardwarmte systemen* (NWN10005) concerns the funding rules 2022.

Disclaimer: deliverables D2.1, D2.2, and D2.3 of the Nieuwe Warmte Nu! GEMINI project contain or discuss developed models that have been delivered to the consortium members as a formal project deliverable via a shared access link to the project Github: <https://github.com/GEMINI-Digital-Twin>. All partners currently have access to the repository. At this stage, the models remain restricted to consortium partners; however, they are planned to be released as open-source in a subsequent phase, after which the public open-source version will become the primary reference for all further development beyond the current closed repository.

Executive summary

Geothermal plants generate large volumes of unstructured text data over their lifetime, such as reports, logs, and technical documentation, which contain valuable contextual knowledge but are difficult to search efficiently. To address this, the Nieuwe Warmte Nu! (NWN) GEMINI project has developed a Retrieval-Augmented Generation (RAG) workflow integrated into the GEMINI digital twin.

The RAG system enables operators and analysts to query document collections through a chat-based interface, rapidly retrieving relevant information without manual searching. The workflow converts documents to text, splits them into chunks, embeds them as vectors, and stores them in a ChromaDB vector database. User queries retrieve the most relevant fragments, which an LLM uses to generate answers strictly grounded in the source documents, preventing hallucinations.

Built entirely on open-source tools such as Ollama, ChromaDB, and Docker, the system supports flexible local and cloud deployment, including Azure. An optional connection to proprietary Azure OpenAI models provides high performance when needed, at the cost of an additional usage fee. Overall, the solution offers a transparent and reliable way to extract operational knowledge from unstructured data, improving decision-making and knowledge retention while laying a solid foundation for future RAG extensions.

Introduction

The integration of Digital Twin technology into a low-enthalpy geothermal plant represents a significant advancement in real-time monitoring, predictive maintenance, and overall operational efficiency. The focus of the Nieuwe Warmte Nu! (NWN) project is to develop and implement a Digital Twin for deep geothermal plants (Doublets). This will enable monitoring and simulation of plant operations, allowing for performance optimization, optimized maintenance planning, and risk mitigation.

Throughout the lifetime of a geothermal plant, large volumes of unstructured text data are generated, particularly during operation and maintenance phase. This includes product documents, test and characterization reports, daily operation logs, maintenance reports, failure descriptions, inspection notes, incident reports, work orders, and communication between operators, service companies, and asset managers. Unlike sensor data, these text-based records capture contextual and qualitative information, such as observed anomalies, operator reasoning, corrective actions, and lessons learned from past interventions. Analyzing this text data is important because it often contains early indicators of degradation, recurring failure patterns, and operational risks that are not explicitly reflected in numerical measurements. Systematic processing of text data enables the extraction of knowledge embedded in historical documentation, supports consistent interpretation across teams and time, and helps bridge the gap between human expertise and data-driven decision-making. As geothermal assets operate over long lifetimes with evolving conditions, leveraging textual information is essential for improving maintenance planning, knowledge retention, and operational reliability.

One of the main new functionalities developed in the project are workflows that allow for the processing and analysis of text-based data (e.g. well intervention end-of-job reports, fluid sample analyses, interference tests, etc.) through the use of Large Language Models (LLM). Such a functionality allows users of the GEMINI digital twin to find information and data found within documents uploaded to the system in a more efficient and interactive manner by way of a chat interface.

The report on Milestone 2.1 of the NWN! GEMINI project (“Large Language Models in the GEMINI Digital Twin”) provides an overview of the different options there were for implementing Large Language Models into the GEMINI digital twin (open-source vs. proprietary LLM’s, local vs. cloud infrastructure, etc.), and for which purposes they could be developed (information retrieval, sentiment analysis, etc.). In discussion with consortium partners, it was decided to focus on developing a retrieval-augmented generation (RAG) application. Additionally, to align with the open-source release of the GEMINI digital twin, it was decided to focus on the implementation of open-source LLM’s and running the workflow using open-source tools and software.

This document gives an overview of the technical implementation of this RAG solution that has been developed and integrated into the GEMINI digital twin to provide these functionalities. Since many of the conceptual details are already covered in the report on Milestone 2.1, the current report will focus primarily on the software and operational aspects of the implemented solution.

The final application is integrated into the GEMINI digital twin as an interactive chatbot. Once public, the code of this chatbot will be available on the GEMINI project website: : <https://github.com/GEMINI-Digital-Twin>. This document provides a description of the RAG workflow (next section), the details of the technical implementation of this workflow into the GEMINI digital twin (subsequent sections), and ends with the main conclusions drawn from the implementation of the RAG workflow.

Building blocks of a RAG system

RAG allows a user to efficiently find information in databases of text documents by directly interacting with a chat interface. It allows the user to ask a question about the information in the documents in the database, for which it then formulates a reply, based solely on snippets of text found in said reports.

RAG solves two main issues that general-purpose LLM's suffer from: lack of specialized knowledge, and hallucinations (providing unverified answers). By providing the LLM access to a database containing the specialized knowledge and restricting it to only answering with information found in it, both of the previously mentioned problems can be circumvented.

Figure 1 gives a high-level schematic overview of the different functionalities needed in a RAG workflow. These are:

1. **Document conversion;** Documents can be found in a variety of file formats (.pdf, .doc, etc.), which can include regular text, tables, as well as images. In order to be able to store them as well as use an LLM to analyze them, all documents need to be converted into plain text. Images can only be processed when using a multi-modal model (one that can process not only text but also images). Image processing is currently not implemented within the GEMINI framework RAG solution.
2. **Chunking;** The bigger a text, the longer it will take for an LLM to process it. Therefore, to make managing a large collection of files more efficient, the texts from step 1 are divided into smaller chunks of text.
3. **Embedding;** An LLM does not work with text data directly, but rather with text data that has been converted into a numerical representation of the original text data, called its embedding. Thus, before they can be used in the RAG workflow, each chunk of text is embedded using an LLM. When using a multi-modal LLM model, all images in the document will also need to be embedded in this step.
4. **Vector store;** Once embedded, each text chunk, together with its metadata, such as the source document, upload time, and document author, is stored in a specialized database designed for numerical representations, commonly referred to as a vector store.
5. **User-LLM interaction;** When a user asks a question to the chatbot, the question is embedded using an LLM, and the vector store is searched through to find chunks in the database that have the most relevancy to the posed question. Then, the most relevant chunks of the text (called the context), along with the original question of the user are given to an LLM, which is asked to formulate an answer to the question by only making use of the information found in the provided context. Once the LLM formulates the answer, it is given back to the user. If the context provided to the LLM does not contain enough information to answer the question, the LLM is instructed to reply that it does not know.

In the next subsections, each of these steps is covered in more detail, and details are provided on how they were implemented in the GEMINI digital twin.

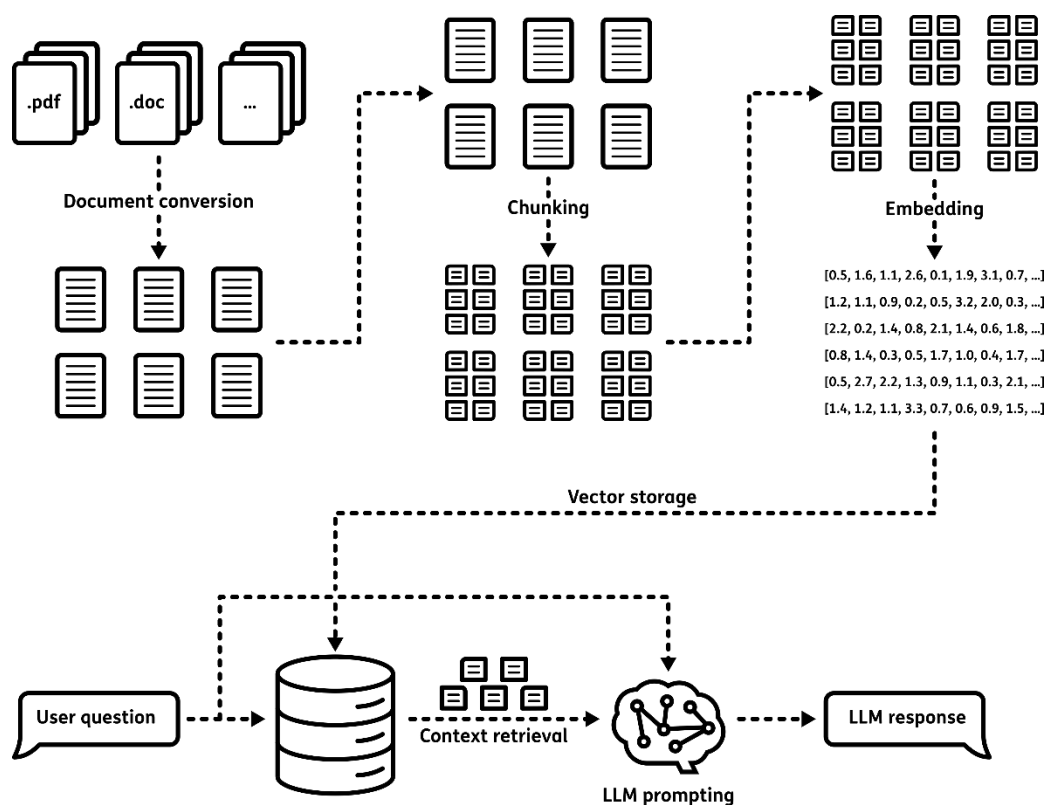


Figure 1 Schematic overview of the different parts of a retrieval-augmented generation workflow.

Document conversion

Documents can be found in a variety of file formats, such as PDF or plain text files. In order to store them and make them usable for analysis by a large language model (LLM), all documents must first be converted into plain text. This conversion step ensures that information from different sources can be processed in a uniform way.

In the developed RAG system, text files are read directly, while PDF documents require a dedicated loading step. PDF files are processed page by page using the document loader *PyPDFLoader* from the package *langchain_community.document_loaders* (Community, sd), which extracts the textual content of each page separately. Processing documents at the page level is important because it preserves the structure of the original document and allows later references to point to specific pages. Once extracted, the raw text is cleaned and normalized using a built-in module in *Python* for working with regular expressions. This cleaning step removes excessive whitespace, collapses repeated spaces, and reduces unnecessary line breaks that often arise from PDF layouts or line-wrapped text. Normalizing the text improves consistency and ensures that later processing steps behave predictably.

Because documents may be written in different languages, the system also detects the language of the extracted text with the help of the *langid* package (Lui & Baldwin, sd). If a document page is not written in English, it is translated before further processing, by asking a dedicated LLM to translate it. The LLM *zongwei/gemma3-translator:4b* (Zongwei, sd) is chosen for this task, which is a specialized model in translation. This guarantees that all documents stored in the system can be queried in the same language, which simplifies retrieval and improves answer quality.

Finally, metadata such as the document name and page number is attached to the converted text. This metadata is preserved throughout the ingestion, retrieval, and answer-generation stages and later enables transparent citations in the chatbot's responses. When an answer is returned to the user, the chatbot displays the generated response together with a list of citations derived from the metadata. Each

citation refers to the source document file name and the specific page from which the supporting information was retrieved. These citations are shown alongside or below the answer, allowing the user to see exactly which documents were used and to trace the response back to the original source material. By presenting citations at the document- and page-level, the system increases transparency and enables users to verify the correctness and origin of the generated answers.

Chunking

The bigger a text, the longer it takes for an LLM to process it. In addition, LLMs have a maximum amount of text they can handle at once. To efficiently manage large documents, the converted text from the previous step is therefore divided into smaller pieces called chunks. Chunking breaks long texts into manageable sections while preserving their meaning.

In this script, chunking is performed based on word count rather than characters, using the dedicated chunk-splitting function. This ensures that chunks align with natural language boundaries and remain readable. To prevent important information from being split across chunk boundaries, a small overlap is introduced between consecutive chunks. This overlap allows contextual information to be shared between chunks and reduces the risk of losing meaning at chunk borders.

The chunking workflow is executed during document ingestion. After text normalization, each page is passed to the higher-level chunking function, which applies word-based chunking, enforces maximum chunk size limits, and attaches metadata required for retrieval and citation.

Chunks are created within the boundaries of individual pages. This design choice ensures that chunks never mix content from different pages, which is essential for accurate referencing and citation later on. Each page is therefore processed independently, and chunking is reset at every page boundary. Therefore, paragraphs that span across a page break in the original document may be divided into separate chunks belonging to consecutive pages. While this can slightly reduce semantic continuity at page boundaries, it preserves precise page-level traceability and allows chunks from adjacent pages to be retrieved together during answer generation. Each chunk is assigned a unique identifier that includes the document name, page number, and chunk index. This identifier allows chunks to be tracked consistently throughout the workflow and ensures traceability from answers back to the original documents.

Chunking behavior is configurable through user-defined parameters. The maximum chunk size controls how many words are included in each chunk, directly affecting the balance between semantic completeness and retrieval precision. The overlap size determines how many words are shared between consecutive chunks, improving contextual continuity at the cost of additional storage and embedding overhead. By adjusting these parameters, the chunking strategy can be tuned to different document types and performance requirements.

Embedding

LLMs do not operate on raw text data, but instead rely on numerical representations of text, known as embeddings. These embeddings encode the semantic meaning of a text fragment in a vector space, enabling mathematical comparison and similarity search. Before text chunks can be used in the retrieval workflow, each chunk is converted into an embedding using a dedicated embedding model. In this work, the *snowflake-arctic-embed* (Snowflake, n.d.) model is chosen due to its specialization in generating high-quality text embeddings.

When a user submits a question to the chatbot, the question itself is also converted into an embedding using the same model. Using the same embedding model for both documents and queries ensures that meaningful similarity comparisons can be made.

Because embedding models have limitations on how much text they can process, chunk sizes are chosen conservatively. If a chunk exceeds these limits, the system automatically reduces its size to ensure successful embedding. Furthermore, to improve performance when processing many chunks, embeddings are created in batches. This significantly reduces the time required to ingest large document

collections.

Vector store

Once text chunks have been embedded, they must be stored in a way that allows efficient similarity-based search. For this purpose, the system uses a vector store, which is a type of database designed to store numerical embeddings and quickly compare them. In this work, *ChromaDB* (Chroma, sd) was selected to be used as the vector store database. In this script, embeddings are stored in the *ChromaDB* database together with their original text and associated metadata. Storing metadata alongside embeddings ensures that retrieved results remain interpretable and traceable.

The vector store uses a similarity measure to determine how closely related two embeddings are. In this system, similarity is computed by comparing the numerical embedding of a user's query with the embeddings of stored document chunks. Each embedding represents the semantic meaning of a text chunk as a numerical vector. Similarity between a user query and stored document chunks is computed by the *query()* function of the *ChromaDB* collection, which performs a cosine-distance-based nearest-neighbor search in the embedding space and returns a distance value that is converted into a similarity score.

$$similarity = 1 - distance$$

The *ChromaDB* package provides this functionality of retrieving relevant context. This similarity score is later used to identify which document chunks are most relevant to a user's question. To improve efficiency, the system tracks which documents have already been processed using a local manifest file. When documents are added, changed, or removed, only the affected data is updated in the vector store. This avoids unnecessary reprocessing and speeds up the ingestion process. The vector store step forms the backbone of the retrieval process and enables fast and scalable access to large document collections.

User-LLM interaction

When a user asks a question, the system follows a retrieval-augmented workflow to generate an answer. First, the user's question is converted into an embedding. This embedding is then used to search the vector store for document chunks that are most relevant to the question.

The vector store returns a set of candidate chunks based on similarity. Chunks that do not meet a minimum similarity threshold are discarded, ensuring that only relevant information is considered. A similarity score of 1 indicates identical prompt and chunk embeddings, while 0 means no similarity. In the case of the GEMINI digital twin RAG system implementation, the similarity threshold value was set to 0.1 so as many relevant candidates as possible were returned. From the remaining candidates, a smaller and more diverse subset of chunks is selected. This step prevents redundant information from dominating the context and helps cover different aspects of the question.

The selected chunks form the context that is provided to the LLM, together with the user's original question. The LLM is explicitly instructed to generate answers using only the information contained in this context. The application uses the Ollama framework (Ollama's new engine for multimodal models, 2026) to handle this interaction. Ollama provides a standardized interface for managing and running LLMs locally. Moreover, the Ollama framework allows different Ollama-compatible models such as LLaMA-based or Mistral-based models. If the provided context does not contain enough information to answer the question, the LLM is instructed to respond that it does not know. This constraint reduces the risk of generating unsupported or fabricated answers.

During document ingestion, Dutch content is translated to English so that all document chunks stored in the vector store are represented in a single language, enabling consistent similarity comparison. User prompts are expected to be provided in English. While prompts in other languages like Dutch may still produce an answer, response quality may be reduced, as the current system is not optimized for multilingual user prompts.

Once the answer is generated, it is returned to the user together with references to the source documents and pages from which the information was taken.

Cloud operation

The architecture of the RAG system is designed to integrate the open-source services of *Ollama* and *ChromaDB* in a system that can be set up with minimum efforts and resources. The system can be set up by running *ChromaDB* and *Ollama* locally or in Docker containers. These services are expected to run alongside the RAG application, and a connection to each component must be defined to ensure proper integration. Document ingestion and querying can be performed entirely on a local machine.

The implementation of the current RAG application consists of the Python backend, the *Ollama* application (or the connection to the Azure GPT models), and the *ChromaDB* vector store. In continuation of the GEMINI digital twin deployment method, the *Ollama* and *ChromaDB* components may be containerized with Docker or executed locally, provided that the RAG application can connect to them. For the purposes of the GEMINI digital twin, an additional interface is developed to allow user-LLM interactions (discussed in more detail in the next section).

As was briefly mentioned, the option utilizes Microsoft Azure services and specifically the GPT models that are integrated into Azure are supported. During the setting up of the system, it is possible to select between the Ollama compatible LLMs or the Azure compatible LLMs, only for the generation of the response. The document ingestion steps are always performed with the Ollama LLMs.

The deployment of the digital twin including Ollama and ChromaDB can be done on a local server or on a cloud server. Docker allows this flexibility as it ensures smooth execution and interaction of the containers. GEMINI digital twin has been tested on an Azure server. The Azure service enables easy access from any internet connection and at the same time requires little time to scale up with more GEMINI digital twin instances.

The RAG application can be operated by executing simple steps. Within the GEMINI digital twin, the following workflow was implemented to integrate the RAG application:

- The user uploads documents from the GEMINI digital twin interface.
- The RAG application automatically ingests the documents. This means that documents are processed and stored ahead of time, allowing user queries to be answered quickly without repeating expensive preprocessing steps. Document ingestion is performed independently of user interaction.
- When a user asks a question to the RAG application, the process of retrieving and generating is executed. First, the relevant context is retrieved via an interaction with the ChromaDB container. After that, the generation is executed with a second interaction with the Ollama container. Due to the longer processing period that the generation requires, the execution of the request to the LLM is handled by Celery, the workflow manager implemented in GEMINI digital twin.

The result of the current RAG system is a framework which can be installed and operated with low effort, while it utilizes open-source software. At the same time, it provides excellent performance in a usable package.

Using the GEMINI RAG implementation

The RAG workflow within the GEMINI digital twin has been implemented through the use of two applications: the report management application (Figure 2), and the chat window (Figure 3). The former application allows a user to upload documents related to the system for which the digital twin has been set up (such as well intervention end-of-job reports, completion reports, fluid sample analysis results, etc.). By checking the “document is included for search” box, the document will be processed and stored according to the workflow described in the prior sections. Processing is only done the first time a document is uploaded. In case a report is removed from the management system, its embeddings are also removed from the ChromaDB instance.

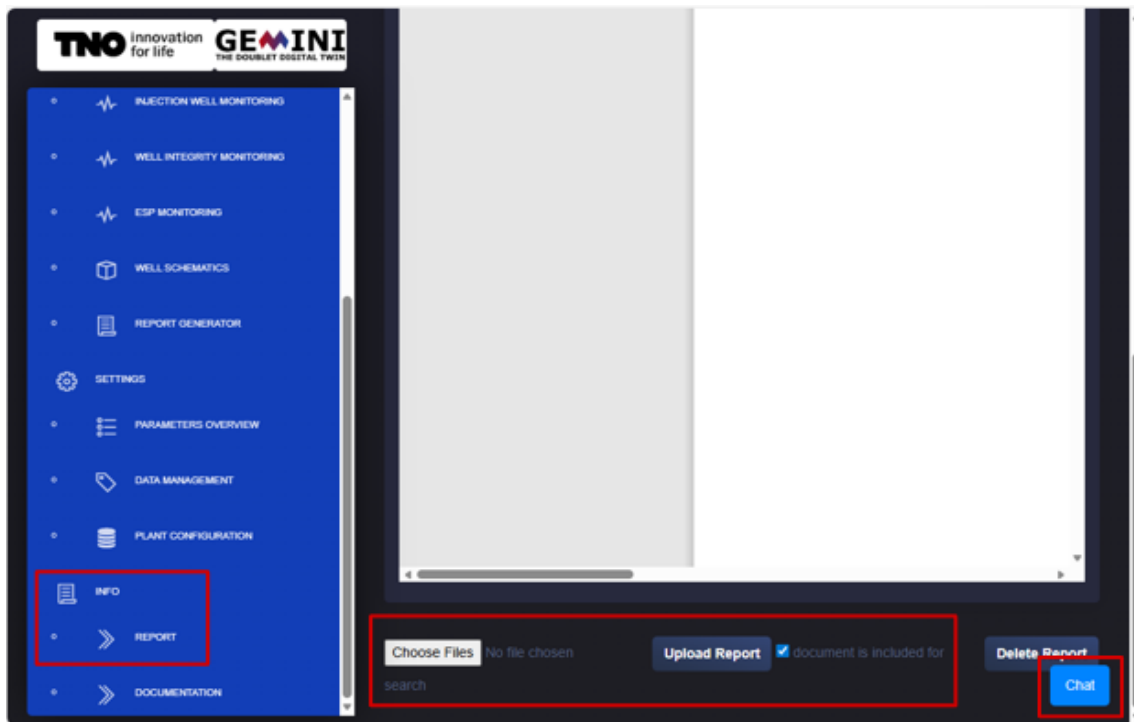


Figure 2 Screenshot of the report management application within GEMINI.

Using the developed RAG workflow is done through the chat interface as shown in Figure 3. By clicking the button on the bottom right of Figure 2, the chat window pops up and the user can interact with it. This window is developed as an overlay on top of the general user interface and is thus accessible from within every application or menu in GEMINI (not only the document management application). The top view of this window shows an overview of the chat history, while in the bottom field, the user can ask the LLM their question. By clicking the “Send” button or pressing “enter” on the keyboard, the question is sent through to the back-end workflow that will go through the steps detailed in the previous sections.

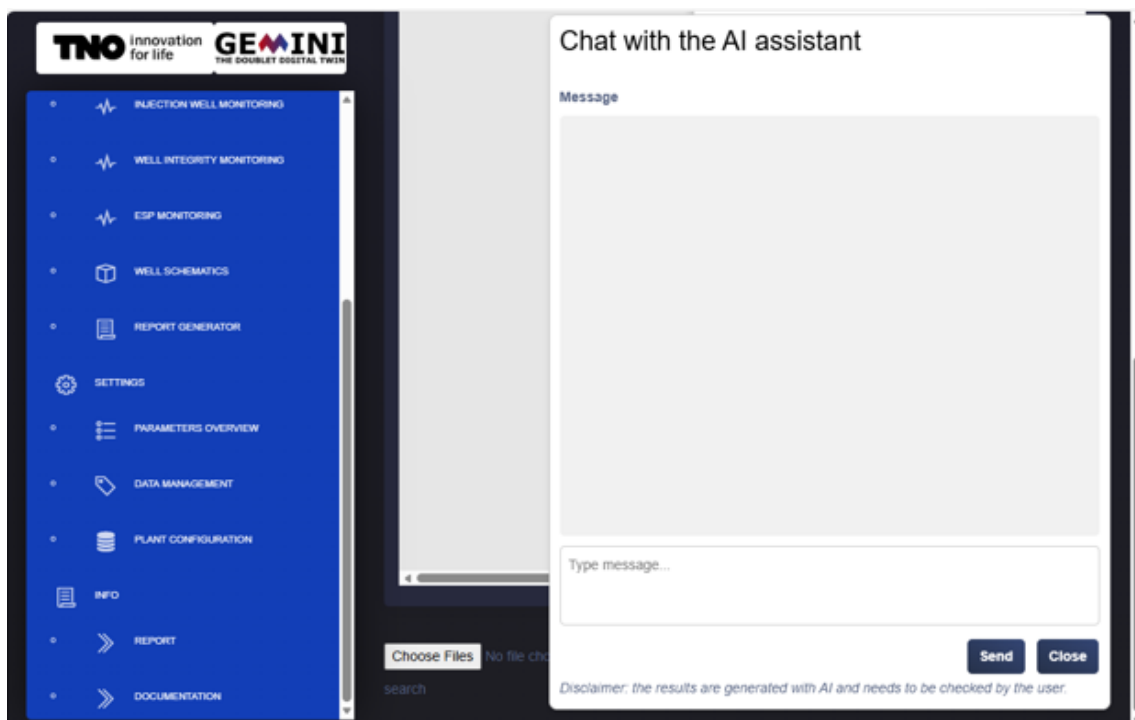


Figure 3 Screenshot of the chat window implementation within the GEMINI digital twin (after clicking the "chat" button in the bottom right of the user interface; see the highlighted button in Figure 2).

After a user asks a question, the chatbot will indicate that it is processing the query, and once finished, will provide the answer back to the user (Figure 4). In addition to providing the response to the user's question, the chat interface also gives the references that it used to form the answer to the question. If needed, these references can be used to double-check the answer provided by the chatbot. In future versions of the GEMINI digital twin, these links will be clickable and directly open the original source document at the location where the information was found.

By clicking the "Close" button on the bottom right of the window, it will disappear and return to the view as seen in Figure 2. When the chat window is closed, its processes will persist in the background. Thus, the user can pose a question, close the window and continue with other functionalities within GEMINI, and when reopening the chat window, the user can continue directly where they left off. In addition, all the chat history of the current session will also still be available. At the moment, chat history is not saved between sessions. Therefore, when a user logs out and logs back in at a later moment or refreshes the webpage, they can unfortunately not return to the discussion they were in during their previous session they logged out off.

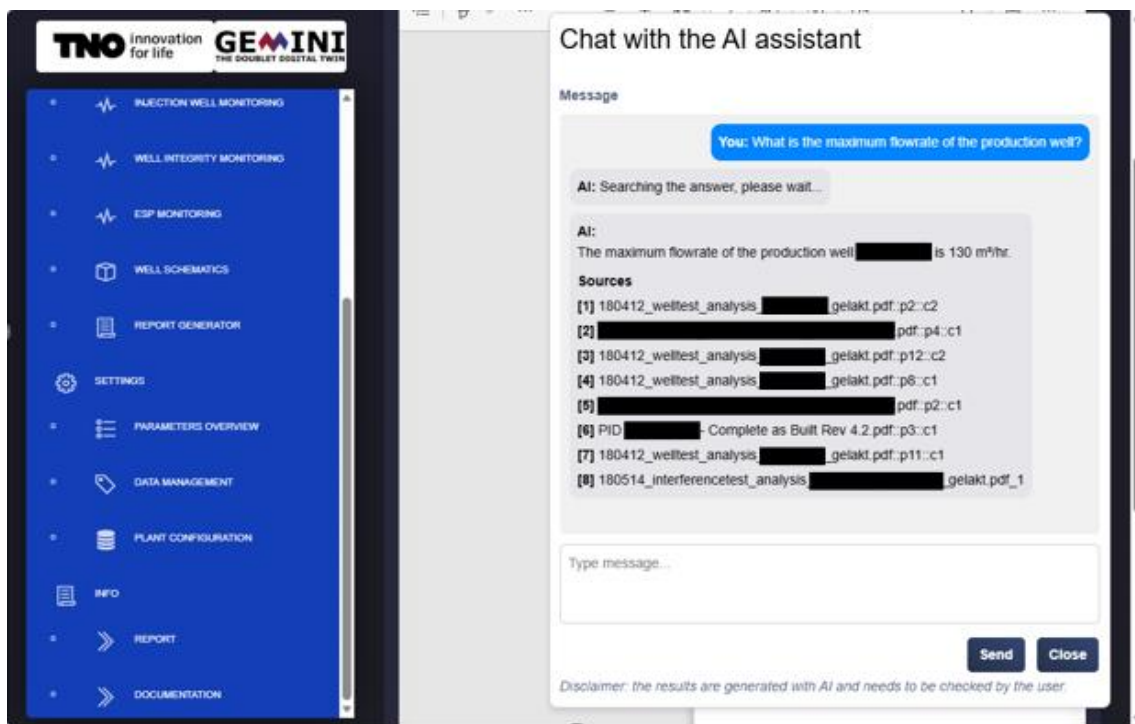


Figure 4 Screenshot of the chat window in GEMINI after posing and responding to a question (sensitive details censored).

Limitation on the current implementation

In Retrieval-Augmented Generation (RAG) applications, there is an inherent trade-off between response speed and answer accuracy. Larger language models typically provide more accurate responses but have higher latency, while smaller models respond faster with potentially lower accuracy. In addition, the hardware the models are run on also has a high impact on its response speed, as with more computing power, more complex models can be run more quickly. The current implementation of the RAG workflow in GEMINI provides two methods for operationalizing its Large Language Model: an open-source solution making use of the Ollama framework, and an option to connect to OpenAI's GPT models through the Azure Cloud service. While the former comes with the benefits of its free and open-source sovereign use, running it as a local solution within GEMINI requires a large amount of computing power to maintain its speed. As computing power comes with its costs, this option can be more expensive due to the required investments needed. On the other hand, the Azure Cloud solution, while giving access to powerful LLM models and computing power at less direct investment costs, does set limits on the user in their freedom to use the provided model as it is proprietary to OpenAI. As such, when comparing the current implementation of the fully open-source RAG implementation running on basic hardware needed for the GEMINI digital twin against the Azure Cloud-connected implementation, the former is significantly slower, taking minutes to provide a result, while the latter responds in seconds.

In the introduction to the RAG workflow, the use of multi-modal LLM's was briefly mentioned as a method for allowing the processing and interpretation of not only text, but also images and plots. In its current form, the GEMINI RAG workflow does not employ any multi-modal models and thus does not support image analysis. However, the Ollama framework used to operationalize the open-source LLM's does contain a number of multi-modal models (Ollama's new engine for multimodal models, 2026), so in theory GEMINI (with some additional development) could in the future also provide multi-modal analysis functionalities.

Finally, it should be noted that in general answering questions accurately is a non-trivial task, and the quality of an answer depends on a number of factors:

- The clarity and specificity of the user's question.
- The quality, structure, and formatting of the uploaded reports.
- The ability of the language model to interpret and reason over the retrieved content.

Consistent document types and simple document formats can make the retrieval of information easier for the system. For instance, older documents with scanned pages or even handwritten text can form a great challenge to properly embed. Likewise, clear or simple questions will lead to accurate and clear answers. If the answer to a question does not initially contain the sought after information, users are encouraged to:

- Ask simple and specific questions.
- Use terminology that is likely to appear in the uploaded reports.
- Break down complex queries into smaller, focused questions when possible.

Conclusions

The advent of Large Language Models has made it possible to effectively and efficiently interact with, and extract data or information from, large collections of documents. Within the Nieuwe Warmte Nu! project, the GEMINI digital twin was extended with Large Language Model-based Retrieval Augmented Generation (RAG) capabilities, allowing operators to more efficiently retrieve pertinent information about their systems from reports without having to search for it themselves.

The implemented RAG workflow combines document ingestion, text processing, embedding, vector-based retrieval, and Large Language Model generation and interaction, to provide users with an intuitive and powerful interface for searching through and retrieving information from technical reports, operational logs, fluid sample results, and other text-based documentation. By limiting the Large Language Model to only generate answers to user questions based on information found in these reports, the RAG workflow prevents hallucinations, making sure only factual responses are provided back to the user.

In line with the project goals of transparency and open-source-based solutions, the RAG workflow within the GEMINI digital twin was developed using entirely open-source components, such as the Ollama framework for accessing open-source LLM models, the ChromaDB database format for storing embedded text chunks, all containerized and deployed using Docker. As an alternative, an optional connection to Azure-hosted OpenAI GPT models has also been provided, offering potentially higher performance and faster response times, albeit using proprietary models and at additional usage fees. The latter option is useful in case appropriate infrastructure is not in place for running the open-source option.

The developed workflow provides the following functionalities within the GEMINI digital twin framework:

- It provides a structural approach to processing and ingesting text-based documents.
- It improves the efficiency of finding relevant information in a large collection of documents.
- It enforces that responses are factual by only allowing relevant information found in the document database to be used to generate responses to user questions.
- It offers transparency in its answers by providing references to the source documents along with its responses.
- It provides intuitive interaction with the workflow by using a chatbot interface through which the user can ask their questions to the Large Language Model.

However, there are a few limitations to the current implementation of the RAG workflow. For one, the RAG system does not make use of any multi-modal models for processing data, and therefore operates exclusively on text-based data, and is unable to (reliably) interpret figures, diagrams, scanned pages, and hand-written notes. In addition, performance and speed of the implementation depend heavily on the hardware it is used on. Thus, while the provided open-source solutions provide flexibility and (data) sovereignty, they can require significant hardware investments to match the speed and accuracy of proprietary software-as-a-service solutions (such as the provided Azure GPT connection). Finally, quality of the generated responses also depends on a lot of factors, including quality of the original source documents, and clarity and completeness of the posed question.

That said, the current development included in the GEMINI digital twin provides accurate and flexible first implementation of a fully open-source RAG workflow for use in geothermal applications. It has been developed using a flexible framework that allows for future extensions, including the use of multi-modal models, improved document processing and chunking, and enhanced interactivity including direct links between references in the responses and their associated locations within source documents.

References

Chroma. (n.d.). ChromaDB, 1.5. Retrieved 2 13, 2026, from Chroma Documentation: <https://docs.trychroma.com/docs/overview/getting-started>

Community, L. (n.d.). langchain_community Documentation, 0.3.31. Retrieved 2 13, 2026, from LangChain: <https://python.langchain.com/docs/integrations/>

Lui, M., & Baldwin, T. (n.d.). langid.py, 1.1.6. Retrieved 2 13, 2026, from GitHub: <https://github.com/saffsd/langid.py>

Ollama's new engine for multimodal models. (2026, February 7). Retrieved from ollama.com: <https://ollama.com/blog/multimodal-models>

Snowflake. (n.d.). snowflake-arctic-embed, 335m. Retrieved 02 13, 2026, from Ollama Model Library: <https://ollama.com/library/snowflake-arctic-embed>

Zongwei. (n.d.). gemma3-translator, version 4b. Retrieved 2 13, 2026, from Ollama Model Library: <https://ollama.com/zongwei/gemma3-translator>