

Using TMS AI Studio TTSMCPCloudAI for Retrieval-Augmented Generation in Delphi with Text and PDF Inputs

Retrieval-Augmented Generation (RAG) has become a key technique for combining Large Language Models (LLMs) with user-provided knowledge sources. By supplying relevant text passages to an AI model along with a prompt, applications can generate answers grounded in specific documents, making the results accurate, explainable, and domain-aware.

In this article, we explore a practical Delphi implementation of RAG using the **TTSMCPCloudAI** component from **TMS AI Studio** <https://www.tmssoftware.com/site/tmsaistudio.asp>. Make sure you have it installed in your Delphi IDE. The project allows users to select either a **plain text (.TXT)** file or a **PDF (.PDF)** document. In the case of PDF input, the software automatically extracts raw text using the **PDFium** open-source rendering library (<https://github.com/chromium/pdfium>). This extracted (or native) text is then fed into the TTSMCPCloudAI component along with the user's question, and the AI model returns answers focused specifically on the provided document content. The reason for extracting the plain text from the PDF before is because not all LLMs have built-in the ability to deal directly with PDF files.

The result is a clean and efficient Delphi application that demonstrates how RAG can be integrated into real-world desktop applications with minimal effort.

1. Overview of the Application

The goal of the application is simple:

1. Allow the user to load a **.TXT** or **.PDF** file.
2. Ensure that for PDF files, the contents are fully converted into plain text.
3. Send this text together with the user's prompt to the TTSMCPCloudAI component.
4. Display the AI-generated result inside a memo.

This combination of automated text extraction and AI-assisted response generation makes it possible to ask any question about the document—summaries, clarifications, translations, explanations, and more—while the model remains grounded in the supplied text.

2. Extracting Text from PDF Using PDFium

PDF is not a text-native format; therefore, retrieving usable text requires a rendering or extraction engine. To avoid complexity and licensing issues, the project uses the **free PDFium DLL**, a robust and widely used library capable of rendering, rasterizing, and extracting text from PDF documents.

To interface with PDFium, a small custom Delphi unit was written. This wrapper handles:

- Loading the PDFium DLL.
- Opening a PDF file.

- Iterating through pages.
- Extracting plain text from each page.
- Concatenating the results into a single UTF-8 text string.

The unit hides all low-level details, exposing only a clean function such as:

```
function ExtractTextFromPDF(const FileName: string): string;
```

This makes text extraction seamless. When the user selects a PDF file, the application simply passes it to the extractor, and the result behaves exactly like a normal .TXT file.

Because the extracted output is pure text, it integrates perfectly with the RAG process used by TTSMCPCloudAI.

3. Preparing the Text for the RAG Pipeline

Once the application has a block of text—either directly from a .TXT file or extracted from a PDF—the next step is to prepare it for use with the AI model.

The TTSMCPCloudAI component provides a very convenient interface for sending prompts, system instructions, and auxiliary data to various supported LLMs with a single code base. The RAG process in this example works by:

- Treating the file's text as **context**.
- Sending it along with the user's question in a single request.
- Allowing the AI model to generate answers grounded in the provided document.

Because the text is bundled into the input, no vector database or pre-processing pipeline is necessary. While more advanced RAG systems use embeddings and semantic search, this example focuses on **direct context injection**, which is often sufficient for shorter and medium-sized documents.

4. Sending the Request Through TTSMCPCloudAI

The TTSMCPCloudAI component handles communication with the configured AI provider (OpenAI, Azure OpenAI, local models, or any supported endpoint). Once the combined prompt is ready, it is passed to the component's request mechanism.

A simplified example may look like this:

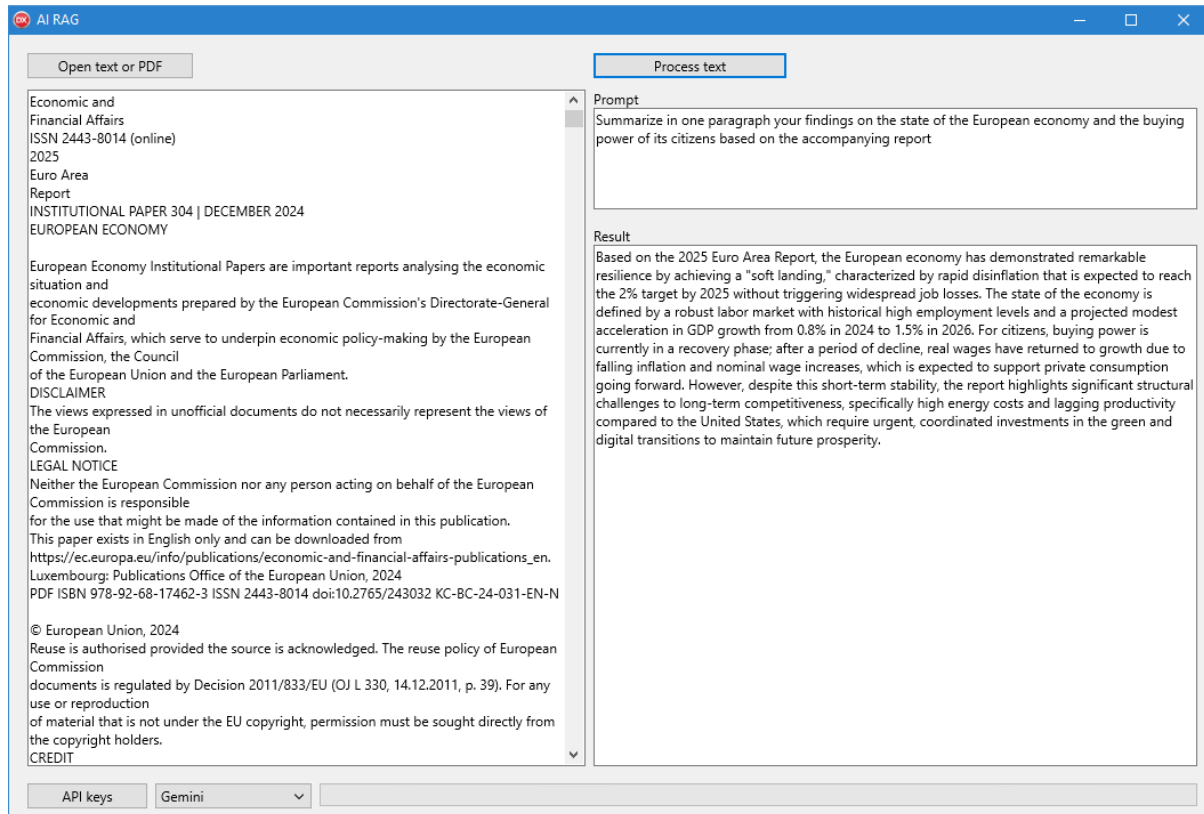
```
AIRequest := TTSMCPCloudAI.Create;  
AIRequest.AddFile(filename, aiftText);  
AIRequest.Context.Text := prompt;  
AIRequest.Execute;
```

When the response arrives, the component's events OnExecuted returns the LLM response, which is then displayed inside a memo on the form.

This allows the user to interactively ask new questions without re-loading the file unless desired.

5. User Interface Workflow

The application's workflow is intuitive:



1. Select Document

A button triggers an TOpenDialog allowing selection of a .TXT or .PDF file.

2. Automatic Text Handling

- For .TXT: the file is read directly.
- For .PDF: the text is extracted using the PDFium-based Delphi unit.

3. Enter Question

The user types any question related to the document.

4. Submit to AI

The TTSMCPCloudAI component sends the prompt and the text.

5. Display Result

The returned response is shown in a memo for easy review.

This framework makes it extremely simple to add document-aware intelligence to Delphi applications without needing heavy infrastructure.

6. Advantages of This RAG Integration

Simple and Lightweight

No database, server, or indexing pipeline is required. All processing happens locally except for the AI model inference.

Flexible Input Handling

With PDF text extraction via PDFium, users can supply almost any document.

Grounded and Accurate Responses

The AI model is constrained to the provided text, avoiding hallucinations and ensuring answers stay fact-based.

Component-Driven Architecture

TTSMCPCloudAI abstracts networking, authentication, request formatting, and response handling, letting the developer focus on application logic.

Extendable Design

Future enhancements could include:

- Chunking long documents.
- Embedding-based semantic search.
- Storing previously analyzed documents.
- Adding UI features like syntax highlighting or source reference tracking.

7. Token Size Considerations When Sending Plain Text to the LLM

Because the application sends both the user's prompt and the full extracted text to the LLM, the **combined size contributes to the total number of tokens** consumed in the request. Every model has a maximum context window—meaning the total number of tokens it can process at once, including both input *and* output. If the extracted file text is large, it may exceed the model's capacity, preventing the request from succeeding or forcing truncation. Developers should therefore be aware of approximate token limits of common models and choose appropriately depending on the expected document size. Below is an overview of the typical maximum context windows for popular LLM families.

Approximate Token Limits of Popular LLMs (2024–2025)

Model Family	Common Context Sizes	Notes
OpenAI ChatGPT (GPT-4 / GPT-4o / GPT-4.1)	128k – 200k tokens	Most recent GPT-4.1 and GPT-4o models support ~200k tokens; older GPT-4 Turbo ~128k.
Anthropic Claude	200k – 1,000,000 tokens	Claude 3.5 Sonnet supports up to 200k; Claude 3.5 Opus and Haiku support 1M-token context windows.
Google Gemini	128k – 1,000,000+ tokens	Gemini 1.5 models support extremely large contexts (1M tokens), making them ideal for large document RAG.
xAI Grok	~128k tokens	Grok-1 models provide large context windows, typically around 128k depending on version.
Meta Llama-3 / Llama-3.1	8k – 128k tokens	Base models often 8k–32k; extended-context versions reach up to 128k.
Mistral / Mixtral	32k – 128k tokens	Open-weights models vary; many support 32k, with some extended versions reaching 128k.
DeepSeek	64k – 128k tokens	DeepSeek-R1 and DeepSeek-V3 typically offer high context windows suitable for large-text RAG.

8. Conclusion

This Delphi project demonstrates how easy it is to integrate Retrieval-Augmented Generation into a desktop application using **TTSMCPCloudAI**. By combining the convenience of component-based AI access with simple but effective PDF text extraction through **PDFium**, developers can empower their applications with intelligent, document-aware capabilities.

Whether the goal is to summarize files, analyze reports, extract insights, or build interactive assistance tools, this architecture provides a solid foundation. With only a small amount of code, Delphi developers can unlock powerful AI features and offer users a remarkably intuitive way to explore and understand the content of their documents.