Gradus Vol 12, No 2 (2025)
ISSN 2064-8014

MODEL CONTEXT PROTOCOL AS A POSSIBLE IMPLEMENTATION FOR THE OPERATION OF RETRIEVAL-AUGMENTED GENERATION

Kálmán Bolla [1*]

[1] Department of Information Technology, GAMF Faculty of Engineering and Computer Science, John von Neumann University, Hungary, https://orcid.org/0000-0002-4666-0990
https://doi.org/10.47833/2025.2.CSC.004

Keywords:
Artificial Intelligence
Generative AI
Large Language Model
Model Context Protocol
Retrieval-Augmented Generation

Article history:
Received 30 September 2025
Revised 6 November 2025
Accepted 15 November 2025

Abstract
This paper presents a system that extends large language models (LLMs) with information stored in external documents. I used the Retrieval-Augmented Generation (RAG) method for the solution, and the Model Context Protocol (MCP) was selected as the communication protocol. The article presents the high-level system architecture and a specific implementation. I verified the operation of the system using Generative AI and MCP Inspector tools.

1 Introduction

Two primary paradigms for the adaptation of Large Language Models (LLMs) have been identified: Retrieval-Augmented Generation (RAG) and model Fine-Tuning. While both approaches present inherent strengths and weaknesses, the selection of the most appropriate method is generally dictated by the specific demands of the given task.

Fine-tuning entails retraining a pre-trained large language model (LLM) with domain-specific knowledge – from documents, files, or other related materials – rather than relying on generic data. This yields the advantage that high accuracy can be achieved, but the major disadvantage is that training is costly, time-consuming, and resource-intensive.

In Retrieval-Augmented Generation (RAG), we retain the general capabilities of the LLM and supplement it with external data, typically one or more vector databases, the operation of which can be divided into three parts. The first step is retrieval, where relevant information is searched for in the database based on user input. The second step is for the user prompt to be merged with the information found in the database (augmentation). Finally, with the question and the new information, it generates a more accurate answer for the user (generation).

A huge advantage of RAG is that we can work with up-to-date information, unlike fine-tuning, where the model must be trained every time the information changes. Consequently, in case a new version of a document containing up-to-date information for users is created, it must simply be loaded into the vector database for the new information to become available for the LLM. This presents the difficulty of its implementation requiring the coordinated operation of multiple components, and the quality of the responses depends largely on the user prompt and the retrieval of relevant data.

An important component of RAG implementation is the vector database, which allows documents and queries to be converted into large-dimensional vectors, which will store the meaning and context of the texts. When a query arrives, it is to be converted to the same vector type and compared with the stored vectors, which allows us to search for relevant text fragments that are

* Corresponding author.
E-mail address: bolla.kalman@nje.hu

close to the query not only in terms of keywords but also in terms of meaning. These types of databases grant us the efficiency of storing large amounts of complex data.

In my article, I first present the theoretical background necessary for understanding, then I demonstrate the proposed RAG implementation. Finally, I detail the validation of the completed system and the results achieved.

2 Theoretical background

This chapter presents the theoretical knowledge essential to understand the working of the planned system.

2.1 Generative AI

In recent years, large language models have become the most exciting and most researched area of artificial intelligence. The real breakthrough in large language models came with the transformer architecture presented in Vaswani et al.'s paper [1], which forms the basis of contemporary LLMs. The transformer revolutionized natural language processing by enabling models to handle long-range dependencies and perform efficient parallel processing.

Devlin and his fellow researchers [2] achieved a major breakthrough with the development of the BERT model, which is capable of creating deep, bidirectional language representations from unlabeled text corpora. The BERT model showed significant performance advantages over the original transformer architecture in general language understanding tasks such as question-answering systems and text analysis.

Brown and his co-authors [3] presented the GPT-3 model, which already contained 175 billion parameters. Although GPT-3 retained the basic components of the transformer architecture, it brought revolutionary innovation through large-scale scaling and an autoregressive approach, especially in the field of creative applications. At the time of its release, it was the largest language model, proving that increasing the number of parameters dramatically improves the effectiveness of few-shot learning.

2.2 Retrieval-Augmented Generation

Retrieval-Augmented Generation (RAG) methods have revolutionized the application of large language models (LLMs) in recent years, as they combine retrieval techniques and generative models, thereby significantly increasing the accuracy and relevance of the generated text.

The roots of RAG principles date back to the 2020s, when Meta researchers introduced the RAG framework, enabling generative models to leverage external data sources (documents, knowledge bases) when responding. As a result, LLMs no longer relied solely on their training databases, but earned the capability of accessing current, domain-specific, or corporate data. This provided a solution for one of the biggest problems with models based on static knowledge: errors (hallucinations) resulting from a lack of up-to-date information and reliability. [4]

RAG architectures have continued to evolve over the years: for example, knowledge-intensive, multimodal, memory-augmented, and meta-learning RAG systems have emerged, all of which have further improved the quality of information retrieval, integration, and text generation. The latest RAG models are now capable of indexing and processing millions of scientific articles and integrating real-time data, while transcending domain-specific limitations and striving to serve professional and institutional needs. [4] [5]

2.3 Model Context Protocol

The Model Context Protocol (MCP) [6] was introduced by Anthropic in November 2024 as an open standard and communication protocol, which aims at providing a unified framework for establishing connections between AI applications (such as large language models) and external data sources and tools. MCP enables the creation of a client-server architecture, in which the AI system acts as the client, while the MCP server functions to make data sources available. The protocol provides a unified interface that greatly simplifies communication between AI systems and various external resources.

3 High-level system architecture

In this chapter, I will introduce the elements used for implementation, document loading, and information extraction.

The proposed solution is not a classic RAG implementation as a new component, the MCP server, appears between the vector database and the large language model. As mentioned in the theoretical background, MCP is a new standard that makes it easier for AI agents to access documents stored in the vector database.

The data will be loaded by a custom document loader application. The data loader is responsible for initializing the vector database collections and splitting and loading documents (e.g., docx, pdf) into the database. Splitting the entire documents into smaller text fragments (chunk) for data loading is highly advisable, and care must be taken to ensure that there is sufficient overlap between the individual fragments [8]. In practice, this will mean that some of the text fragments will be duplicated in the database. In this paper 300 token chunk size and 80 token overlap is used. For better search results, metadata should also be associated with the text [7]. The metadata collection used for this solution is as follows:

- source: file name
- title: document title
- upload date (date and time): the timestamp of the document uploading
- validity period (optional)
- page number
- tags (list): important tags in the text

Metadata also plays an important role (beyond providing accurate and relevant information) in reducing so-called hallucinations, because it is possible to identify which text fragment on which page of which document was used to generate the response (it becomes traceable).

The figure below shows the high-level system architecture:

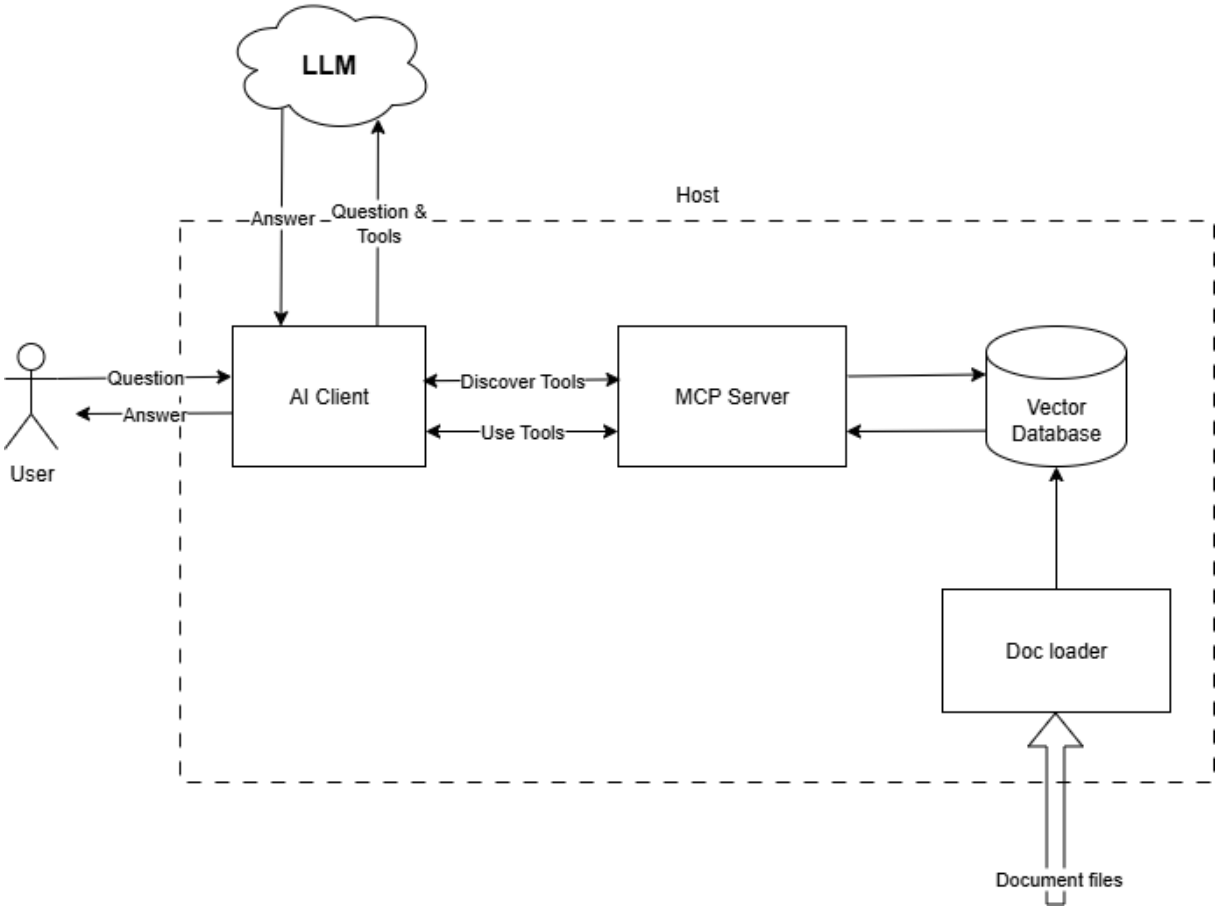

Figure 1. RAG system architecture using MCP protocol

3.1 Vector database

Qdrant [*]has been chosen as a vector database being a high-performance, open-source vector database and similarity search engine developed for modern artificial intelligence (AI) and machine learning applications, especially when fast search and storage of large, high-dimensional vectors is required. Its key features include fast search capabilities, the ability to filter data based on complex filter conditions, and an easy-to-use API for database management.

3.2 Generative AI

Claude AI has been selected as an advanced generative artificial intelligence chatbot developed by Anthropic. The purpose of this system is to enable natural, human-like text and multimodal (text, image, and voice-based) interactions. Two of its main features have been applied: the large context window[†] (1 million tokens) and the customizable integration options[‡]. The advantage of the former is that it can process a significant amount of information simultaneously, while the integration options also support MCP server connection. The Claude Sonnet 4 model has been used for testing the system.

3.3 AI client

Claude Desktop acts as the AI client in the system. This is also a desktop application developed by Anthropic, which allows direct access to Claude AI models running in the cloud. The application is not only capable of text-based dialogue with the user but can also control the host computer. For us, it is a valuable solution primarily because of its customizability and integration with local systems. In the application's configuration file (*claude_desktop_config.json*), multiple MCP servers can be registered, so the general knowledge provided by Claude can be supplemented with information from our own data, such as a vector database.

3.4 MCP server

The integration between the large language model and the Qdrant vector database was completed by using the official Qdrant MCP server[§]. The official Qdrant MCP server is an open source backend application developed in Python, which complements the Qdrant vector database supporting the Model Context Protocol (MCP). Principally, it aims at enabling various LLM (large language model) applications to conveniently and standardly store and retrieve relevant information and "memories" from the Qdrant system—for example, code snippets, documentation, or any textual knowledge.

The official MCP server provides only two tools, *qdrant-store* and *qdrant-find*. The *qdrant-find* tool responsibe for querying on the saved documentum chucks, while the *qdrant-store* save text fragments with metadata.

3.5 Document loader

The document loader is a custom application that I developed in Python. I have already defined its general tasks at the beginning of this chapter, so in this subsection I will only share information related to its specific implementation.

During implementation, it proved significant to attend to the fact that a single instance of the official Qdrant MCP server only supports one collection, so the document loader imports all uploaded documents to the same collection. If the collection does not yet exist, the application can create it, but there are some important criteria to keep in mind. The official Qdrant MCP server only supports *fastembed* embedding providers and *sentence-transformers/all-MiniLM-L6-v2* embedding models. If these requirements are not presented, reading the information through the server will prove to be impossible. The next restriction is that *qdrant-find* assumes that the information is stored in the *document* and the metadata is stored in the *metadata* attribute in json format.

[*] There are other vector database alternatives, such as ChromaDB, Milvus, etc.
[†] https://claude.com/blog/1m-context
[‡] https://claude.com/platform/api
[§] https://github.com/qdrant/mcp-server-qdrant

After creating the collection, the application reads the files one by one within a given directory. It can handle docx and pdf formats. The source metadata includes the path to the source file, and the validity period and page number are automatically filled in. The application asks the user for the document title, tags, and validity period. I set this metadata for each text excerpt when it is loaded, so if multiple documents are saved to the same collection, the source of the information will become apparent, and the tags will help us search faster.

4 Validation

After setting up the system, I checked its correct operation in two ways. First, I checked the connection between the MCP server and the Qdrant vector database using the MCP inspector [*]application. Once I was convinced that the integration between the two components was working, I was able to establish a connection between Generative AI and the MCP server and validate the system.

As a sample document, I used the dean's regulation titled "*A záróvizsgák lebonyolításának rendje*" from John von Neumann University, which I imported into the Qdrant database using a document loader before starting the testing. During testing, I was curious to see if all relevant information on the topic of final exams could be extracted from this specific document without Generative AI relying on its own general knowledge or hallucinating (giving incorrect answers).

4.1 MCP inspector test

MCP Inspector is a developer tool that provides a visual interface for testing and debugging MCP (Model Context Protocol) servers.

I solved the MCP server communication with an STDIO connection. In the case of an STDIO connection, MCP Inspector starts the Qdrant MCP server with the uvx command. For proper operation, three additional environment variables need to be set:

Table 1. MCP inspector parameters for STDIO connection

Parameter	Description	Value
QDRANT_URL	The access address of the Qdrant database instance	https://localhost:6333
COLLECTION_NAME	Name of the collection	nje_collection
EMBEDDING_MODEL	Name of preprocessor embedding model	sentence-transformers/all-MiniLM-L6-v2

The MCP inspector enables the connection to the MCP server via a web interface. *qdrant-find* tool is used to query on the vector database, which expects a text-type input query, and returns a JSON-formatted response with all relevant results from the database.

[*] https://github.com/modelcontextprotocol/inspector

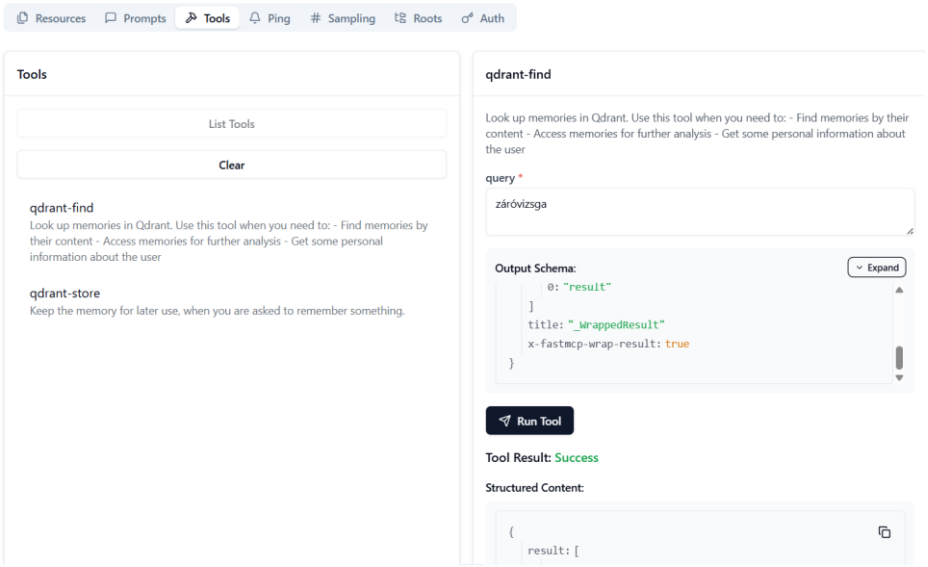

Figure 2. MCP inspector web interface and qdrant-find tool usage example

Several queries have been run with MCP Inspector and I got the correct text snippet back every time.

4.2 Claude AI validation

In order to validate the system's operation, 10 questions were created covering the content of the 9-page document used for testing and the questions were extended with the text "Use the local Qdrant database" to ensure that the system would always search the local database rather than relying on its general knowledge. Prompts used to validate the system:

1. *Summarize the contents of the dean's regulation on the procedure for conducting final exams.*
2. *Who are the members of the final exam committee?*
3. *How can I register for the final exam?*
4. *What are the requirements for registering for the final exam?*
5. *Who decides on the conduct of final exams in cases of force majeure?*
6. *How are final exam results calculated? Provide an example where ZV1=4, ZV2=4, and SZDG=5.*
7. *What will be the textual qualification on the diploma if the final exam result is 4.81?*
8. *Who organizes the final exam?*
9. *What is the clerk's role in the final exam?*
10. *How many members must be present at the final exam at the same time?*

The prompts formulated above were handled appropriately using Claude Sonnet 4 model MCP tools (*qdrant-find* and *qdrant-store*), with all questions answered based on data found in the vector database on the host machine. Only question 7 could not be processed due the fact that the information is stored in a table format inside the document and Claude therefore provided an incorrect answer.

5 Conclusion

In my article, I presented a Generative AI-based system that allowed me to access information from a local vector database, supplementing Generative AI's general knowledge with organization-specific knowledge. The system is powered by the MCP protocol presented by Anthropic, which greatly simplifies communication between participants. I uploaded the vector database using a proprietary application, taking into account the restrictions required by the MCP server.

References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, I. Polosukhin, “Attention Is All You Need,” 31st Conference on Neural Information Processing Systems (NIPS), Advances in Neural Information Processing Systems, Vol. 30., 2017, doi: 10.48550/arXiv.1706.03762

[2] J. Devlin, M. Chang, K. Lee, K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” *Proceedings of NAACL-HLT 2019 (Minneapolis, Minnesota)*, pp. 4171–4186, 2019, doi: 10.18653/v1/N19-1423

[3] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam et. al., Language Models are Few-Shot Learners, *Proceedings of NAACL-HLT* 2020, 2020, arXiv:2005.14165

[4] S. Gupta, R. Ranjan, S. N. Singh, A Comprehensive Survey of Retrieval-Augmented Generation (RAG): Evolution, Current Landscape and Future Directions, arXiv preprint, 2024, 10.48550/arXiv.2410.12837

[5] Ozan Gokdemir et al., HiPerRAG: High-Performance Retrieval Augmented Generation for Scientific Insights, arXiv preprint, 2025, 10.48550/arXiv.2505.04846

[6] Introducing the Model Context Protocol, https://www.anthropic.com/news/model-context-protocol, 2024 november 25

[7] Agada Joseph Oche, Ademola Glory Folashade, Tirthankar Ghosal, Arpan Biswas, A Systematic Review of Key Retrieval-Augmented Generation (RAG) Systems: Progress, Gaps, and Future Directions, arXiv preprint, 2025, 10.48550/arXiv.2507.18910

[8] Mahmoud Amiri, Thomas Bocklitz, Chunk Twice, Embed Once: A Systematic Study of Segmentation and Representation Trade-offs in Chemistry-Aware Retrieval-Augmented Generation, arXiv preprint, 2024, 10.48550/arXiv.2506.17277