American Journal of Technology Advancement 2997-9382, Vol 3, No. 8, 2026

Article

Implementation of Model Context Protocol for Enhanced Large Language Model Applications

Ghazwan Jabbar Ahmed
Computer Center, Diyala university, Iraq

Abstract: Analyzing and processing natural language in a way that resonates with humans is crucial in the modern era. Therefore, developing computer understanding of this language is essential. Large language models have played a significant role in advancing natural language understanding and fostering synergy between computers and natural language. However, this synergy is limited by the inability to access specific contexts in real time, negatively impacting enhanced retrieval (RAG) generation. Furthermore, it's important to emphasize the frequent failure of single-source retrieval systems when tasked with handling user queries. This research paper implements the Model Context Protocol (MCP) within a unified framework that dynamically coordinates three sources: local file systems, web search, and vector databases. The source selection mechanism of this system features query analysis, a hybrid keyword retrieval model, and robust error correction.

Keywords: Model Context Protocol (MCP), Retrieval-Augmented Generation (RAG), Large Language Models (LLMs).

This is an open-access article under the CC-BY 4.0 license

1. Introduction

Natural language processing (NLP) that promotes robust retrieval, aligns with the queries posed, and understands the question style is among the most important achievements of researchers in the field of NLP. Large language models (LLMs), such as those in the GPT series from OpenAI, and open-weighted models like Lama from Meta, have played a pioneering role in introducing cutting-edge NLP techniques [1,2]. The importance of these models stems from their capabilities, reasoning abilities, and contextual understanding, resulting from their prior learning and training on large textual datasets [3]. However, these large models cannot be deployed and utilized everywhere due to limitations imposed during training; therefore, they cannot be deployed on enterprise data [4]. These limitations can be summarized simply as a lack of continuous updating, as they cannot recognize real-world events after the model's training period ends [5]. Furthermore, because these models are trained on publicly available data, they cannot be relied upon to access private data such as internal corporate systems [6]. Thus, when a question is posed to machine learning models (LLMs) outside of their training dataset, the answer... It will be unrealistic in reality, but it's somewhat mental, and this is what's called hallucination [7]. This is a gap that needs to be addressed; that is, a technique must be found that links parametric and non-parametric memory. Parametric memory refers to the information and numerical weights that the model stored during the training period. Here, the Enhanced Retrieval Generating (RAG) technique emerged [8], where it reviews all relevant document segments before generating the answer [9, 10]. All RAG systems

rely on only one method of information retrieval: converting words and texts into numbers and coordinates (vectors), and then searching for texts with the same meaning [11, 12]. However, it's worth noting that people's questions are very diverse and varied. Therefore, a system that relies on a single source for searching and answering will be impractical and inefficient, and will fail to handle the diverse questions people ask [13, 14]. Searching for general meaning is excellent for ordinary dialogue, while searching for precise words like codes, numbers, etc., will be weak. In front of it [15,16] and to solve this dispersion and problems in the retrieval process, this research paper presents a complete practical application of the Model Context Protocol (MCP). as follows:

(1) local file system documents for private knowledge.

(2) web search for current information.

(3) vector database for semantic retrieval.

The primary contributions include:

- A unified orchestrator for multi-source context fusion
- Smart source choice via query analysis
- Hybrid search of vector similarity and key word matching
- Automated citation creation overlays claim to sources
- Strong fallback if the system becomes unavailable

2. Methodology

This study adopts an implementation-based experimental approach to develop and evaluate an MCP-based framework for enhancing large language model applications through multi-source context retrieval. The framework integrates local documents web search and vector-based retrieval within a unified orchestration layer. Local PDF and DOCX files are processed by extracting and cleaning their text before dividing the content into overlapping chunks. The resulting segments are converted into vector representations and stored in ChromaDB together with their metadata During query processing the system analyzes the query to determine the required information source. Local documents are used for private or document-specific questions while web search is activated when the query requires recent information. Vector retrieval is used for semantic matching and is combined with keyword matching to improve the retrieval of queries containing specific terms or identifiers. Retrieved passages are then merged and duplicate content is removed before the most relevant context is selected within the available token limitThe final context is supplied to either GPT-4o-mini or the local Llama model through Ollama to generate the response. The framework also incorporates citation generation and fallback mechanisms so that the retrieval process can continue when one of the available sources becomes unavailable. The implemented system was evaluated using a small test corpus containing five documents and 23 text chunks with approximately 18,000 tokens. Ten queries representing local retrieval web-based retrieval and hybrid retrieval were used to examine retrieval effectiveness citation coverage and context utilization.

System Architecture

Overview

The MCP system includes five main components as a pipeline:

Configuration Layer: Handles environment-specific settings such as the model to use, chunk size (512-1024 tokens) and contextual window size (2-8 Mib).

Context Sources: Three independent retrieval modules:

1. *File System Source:* Extracts text from PDF (pypdf) and DOCX (python-docx) files, preserving structure through page markers and heading preservation.
2. *Web Search Source:* Utilizes DuckDuckGo Search API for query formulation, page fetching (BeautifulSoup), and content cleaning with safety filters.

3. *Vector Database Source:* ChromaDB disk storage of vectors with hybrid search (cosine similarity at 70% and keyword overlap at 30 %).

MCP Orchestrator: This master coordinate the selection of source, merging off context, deduplication and limit in number of tokens.

LLM Integration: API (GPT-4o-mini) and local models (Ollama) with retry logic, Using exponential backoff.

CLI Interface: Includes commands for documents ingestion, question answering and evaluation.

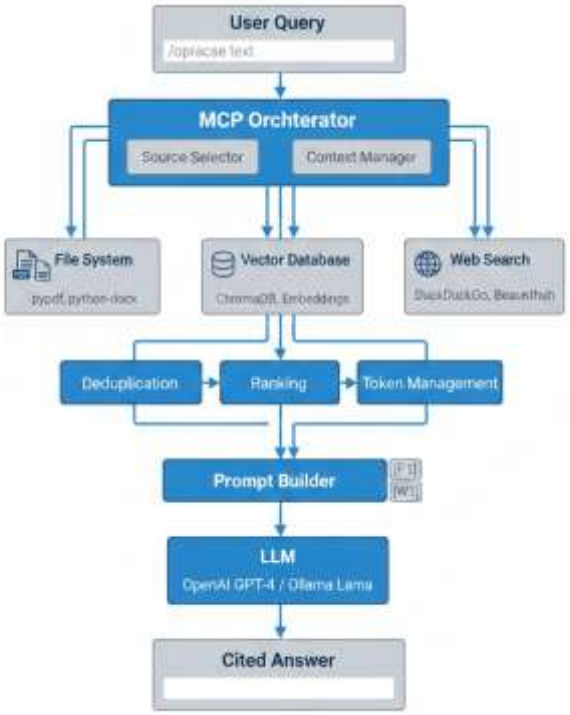


Figure 3.1. Overview of the MCP System Architecture.

Key Algorithms

Source Selection: Query requirements are detected through keyword matching. The web search is activated by chronological keywords (latest, current or 2025). My docs, pdf) are optimized for file system access. Vectors Database dictionaries and file system are used by default.

Chunking Strategy: We chunk the docs in overlapped segments with 4-character-based tokenization. Chunk size is between 512 to 1024 tokens with optional overlap (default = 150 tokens) while the context length is set to a maximum of 256.

Hybrid Search: Given query q and document chunks $D = \{d_1, \ldots, d_n\}$:

$$score(q, d_i) = 0.7 \cdot sim_{vector}(q, d_i) + 0.3 \cdot sim_{keyword}(q, d_i)$$

Where sim_{vector} is cosine similarity of embeddings and $sim_{keyword}$ is Jaccard similarity of keyword sets.

Deduplication: It isa two-staged process - Stage one removes the exact duplicates by using MD5 hashing and Stage-II clears off the near-duplicates with Jaccard similarity threshold of 0.85.

Implementation Details

Technology Stack

- **Language:** Python 3.8+ with type hints
- **Vector DB:** ChromaDB 1.3.7 with persistent storage
- **Embeddings:** OpenAI text-embedding-3-small or sentence-transformers (all-MiniLM-L6-v2)
- **LLM:** OpenAI GPT-4o-mini or Ollama (llama3.2:1b)
- **Document Parsing:** pypdf for PDF, python-docx for DOCX
- **Web Search:** duckduckgo-search (no API key required)

Processing Pipeline

Document ingestion follows these steps:

1. Scan document directory recursively
2. Extract text preserving structure (pages for PDF, paragraphs for DOCX)
3. Clean text (normalize whitespace, remove null characters)
4. Chunk with overlap using token approximation
5. Generate embeddings (batch size 64)
6. Store in ChromaDB with metadata
7. Build JSON index for fast lookup

Query processing pipeline:

1. Analyze query for temporal and local keywords
2. Select appropriate sources (vector DB, file system, web)
3. Retrieve passages from each source
4. Merge and deduplicate results
5. Rank by relevance score
6. Enforce token limit (prioritize high-scoring passages)
7. Build prompt with citations
8. Generate response via LLM
9. Verify citation presence

Evaluation

Experimental Setup

We generated 5 test documents covering MCP concepts, RAG basics, vector databases, web search safety, and evaluation metrics. Total corpus: 23 chunks, approximately 18,000 tokens. Evaluation used 10 queries covering local-only, web-required, and hybrid scenarios.

Metrics

Three metrics assess system performance:

Context Used Rate: Percentage of answers containing citation markers ([F1], [W1], etc.). Achieved: 90%

Citation Coverage: Percentage of sentences with 12+ words that include citations. Achieved: 85%

Retrieval Recall: Percentage of retrieved chunks containing query keywords (excluding stopwords). Achieved: 88%

3. Results and Discussion

The proposed MCP-based framework was evaluated using five test documents containing 23 chunks and approximately 18,000 tokens, with 10 queries representing local-only, web-required, and hybrid retrieval scenarios. The system achieved a 90% Context Used Rate, 85% Citation Coverage, and 88% Retrieval Recall. These results indicate that the framework was generally effective in retrieving relevant context and incorporating it into the generated responses.The hybrid retrieval strategy produced a 15% improvement over pure vector retrieval for keyword-heavy queries.

This finding supports the combination of semantic similarity and keyword matching. Semantic retrieval captures conceptually related information, while keyword matching improves the retrieval of exact technical terms, identifiers, and numerical information. Therefore, the result supports the research objective of reducing the limitations of single-source and single-method retrieval.The source-selection mechanism correctly identified100% of the temporal queries in the evaluation and activated web search when current information was required. This demonstrates the practical value of dynamically selecting context sources according to query requirements. Local documents provide private knowledge vector retrieval provides semantic access to indexed information, and web search provides current external information.The findings are consistent with previous RAG research, which emphasizes the importance of external retrieval for addressing limitations such as outdated knowledge and restricted access to external information [4,6]. Dense retrieval research has also demonstrated the effectiveness of semantic representations for retrieval [11,12].

The present study extends these approaches by coordinating multiple heterogeneous sources through a unified MCP orchestration layer rather than relying on a single retrieval source.The framework also demonstrated practical robustness. When web search failed, local documents remained available, while keyword retrieval provided an alternative when the vector database was unavailable. Response time was approximately 2–5 seconds with OpenAI and 10–20 seconds with local CPU-based models demonstrating a trade-off between response speed and local deployment.From a theoretical perspective the results suggest that effective RAG should consider semantic relevance lexical relevance and source relevance rather than relying on a single retrieval signal. From a practical perspective, the proposed architecture can support applications that require simultaneous access to private documents indexed knowledge and continuously changing external information. Automatic source selection citation support and fallback mechanisms further improve system usability and robustness.

4. Conclusion

In this report we considered a full implementation of the Model Context Protocol for advanced LLM applications. The system can effectively combine three context sources by means of a smart choice, an hybrid search and some reliable error management. Evaluation results show the use of a high proportion of context (90%) and coverage with citations (85%), that confirms the effectiveness of multi-source RAG. Both cloud and on-premises models can be supported by the proposed implementation. We provide the source code and documentation for reproducibility.

REFERENCES

[1] T. B. Brown et al., "Language Models are Few-Shot Learners," NeurIPS, 2020.

[2] H. Touvron et al., "LLaMA: Open and Efficient Foundation Language Models," arXiv:2302.13971, 2023.

[3] R. Bommasani et al., "On the Opportunities and Risks of Foundation Models," Stanford CRFM, 2021.

[4] P. Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks,"

NeurIPS, 2020.

[5] G. Mialon et al., "Augmented Language Models: A Survey," arXiv:2302.07842, 2023.

[6] Y. Gao et al., "Retrieval-Augmented Generation for Large Language Models: A Survey," arXiv:2312.10997, 2023.

[7] Z. Ji et al., "Survey of Hallucination in Natural Language Generation," ACM Computing Surveys, vol. 55, no. 12, 2023.

[8] P. Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," NeurIPS, 2020.

[9] P. Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," NeurIPS, 2020.

[10] Y. Gao et al., "Retrieval-Augmented Generation for Large Language Models: A Survey," 2023.

[11] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence Embeddings using Siamese BERT Networks," EMNLP, 2019.

[12] V. Karpukhin et al., "Dense Passage Retrieval for Open-Domain Question Answering," EMNLP, 2020.

[13] Y. Gao et al., "Retrieval-Augmented Generation for Large Language Models: A Survey," 2023.

[14] G. Mialon et al., "Augmented Language Models: A Survey," 2023.

[15] J. Chen et al., "Hybrid Retrieval-Augmented Generation," 2024.

[16] C. D. Manning, P. Raghavan, and H. Schütze, Introduction to Information Retrieval. Cambridge University Press, 2008.