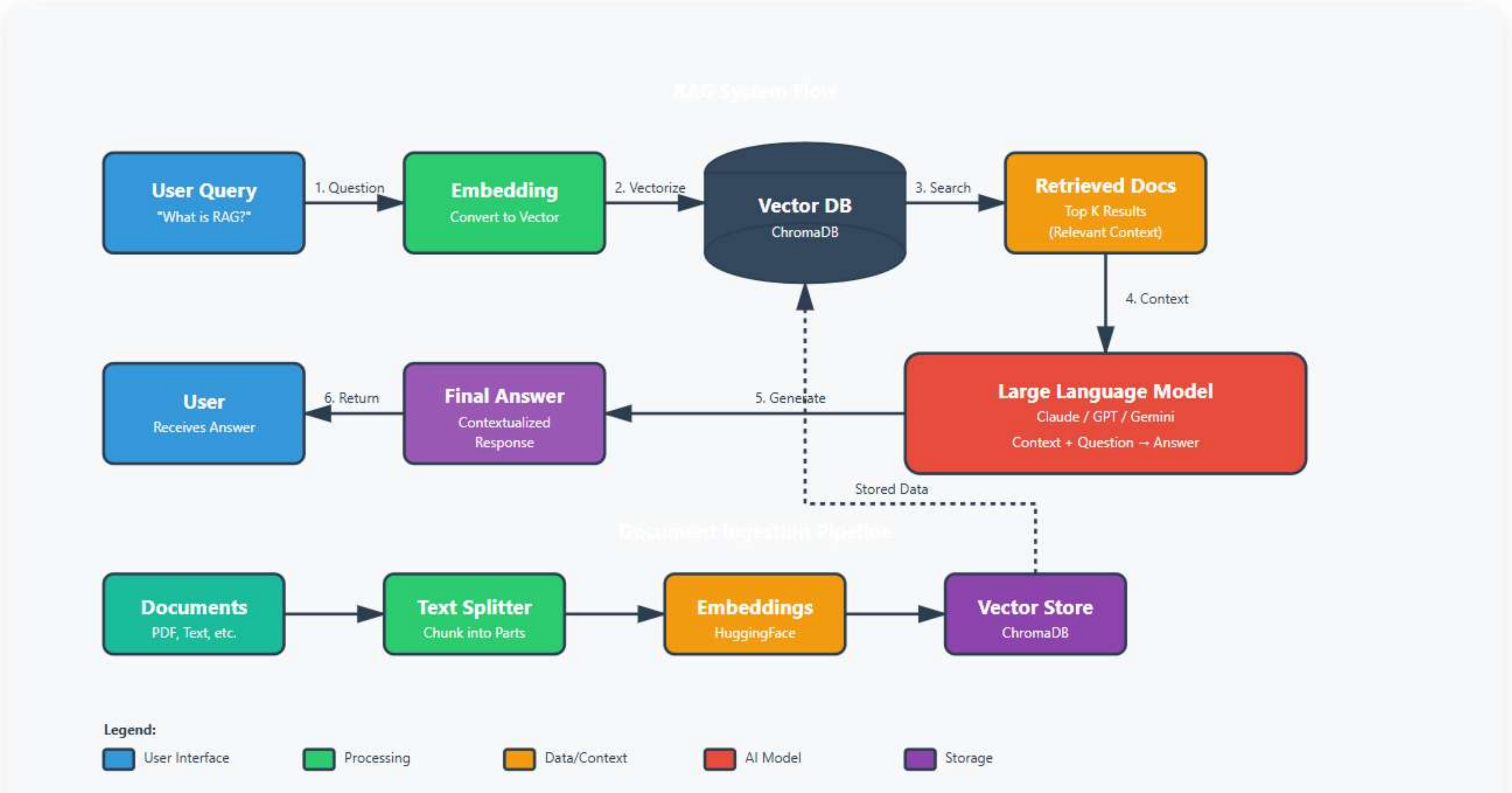
# RAG MCP Server

With ChromaDB, LangChain, Claude LLM

Lucas Kim

November 26, 2025

# 🔍 RAG (Retrieval-Augmented Generation) Architecture



## RAG Process Flow

1 **Document Ingestion (add_documents)**

Receive source documents → Split into chunks using RecursiveCharacterTextSplitter → Convert to vectors using HuggingFace embeddings → Store in ChromaDB with metadata

2 **Vector Search (search_documents)**

Convert user query to embedding → Search ChromaDB using cosine similarity → Return top k documents (default: 4)

3 **Context Construction**

Merge retrieved documents into unified context → Maintain document order and source information → Generate prompt template

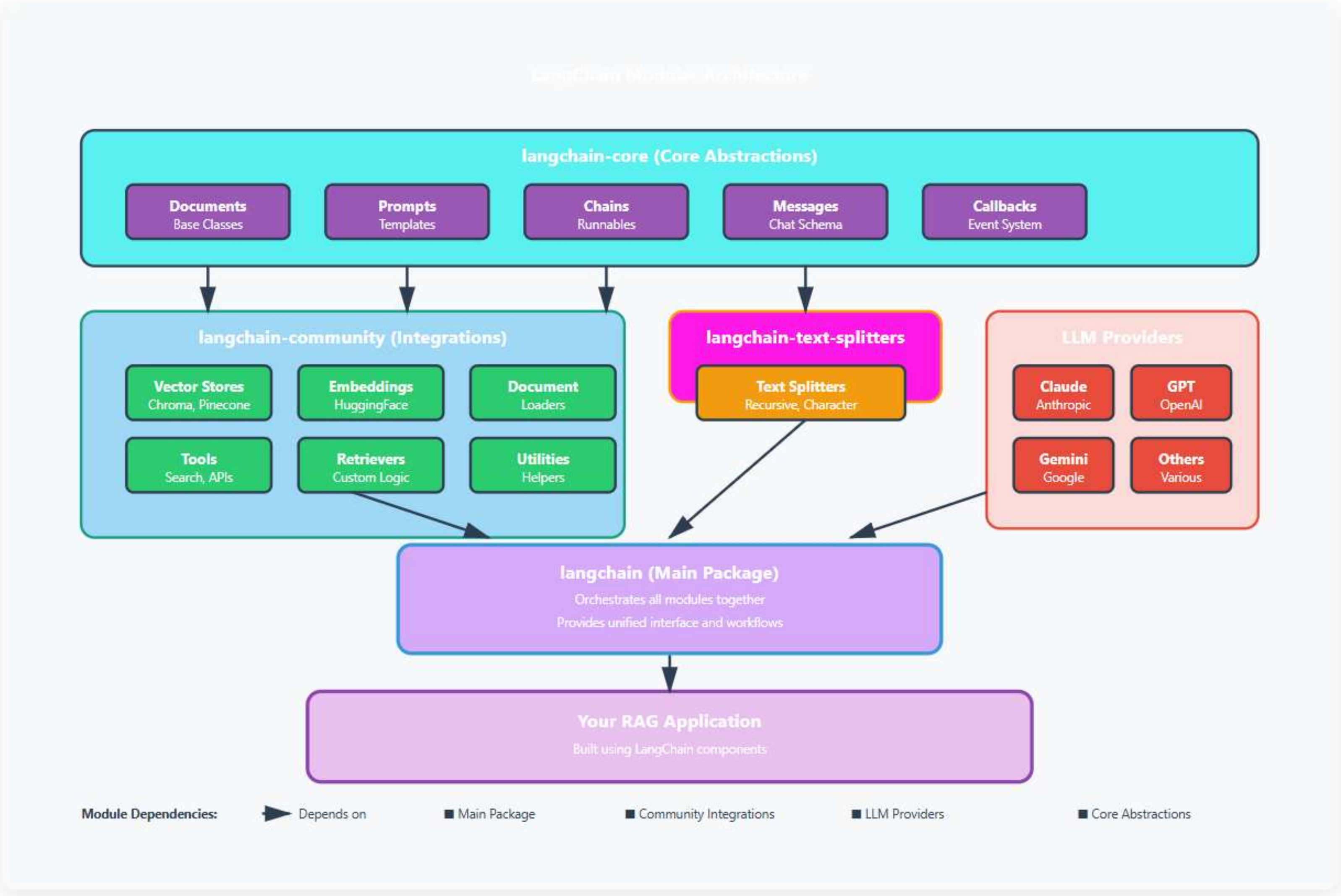
4 **AI Answer Generation (rag_query)**

Send context + question to Claude API → Claude Sonnet 4 model generates answer → Return result with source citations

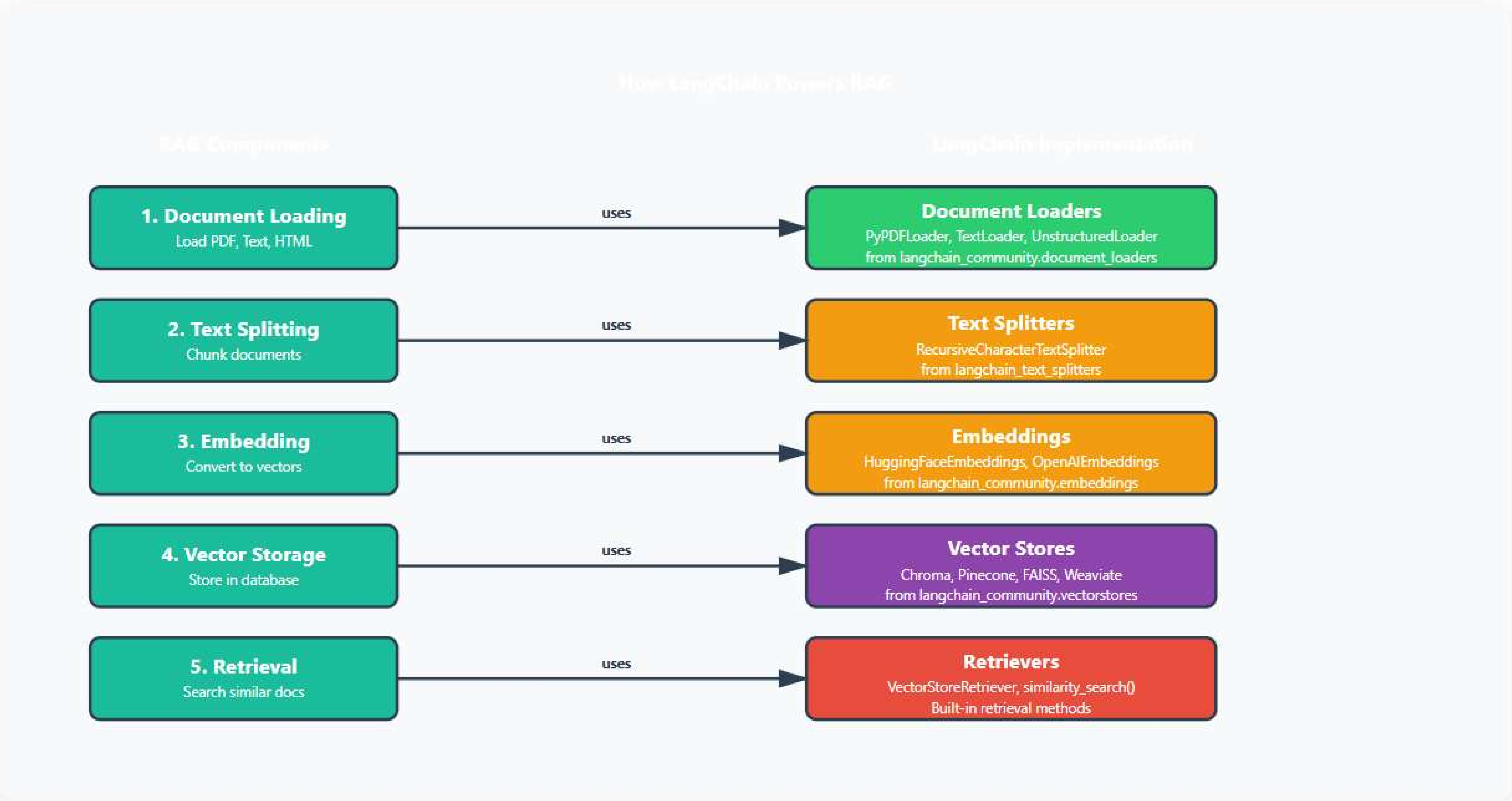
5 **Result Formatting & Return**

Format answer text + reference document info → Include metadata (source, chunk position, etc.) → Deliver final result to user

# ⛓ LangChain 1.0 Architecture



# 🔗 RAG + LangChain Integration



How LangChain Powers RAG

RAG Components

LangChain Implementation

**1. Document Loading**
Load PDF, Text, HTML

uses

**Document Loaders**
PyPDFLoader, TextLoader, UnstructuredLoader
from langchain_community.document_loaders

**2. Text Splitting**
Chunk documents

uses

**Text Splitters**
RecursiveCharacterTextSplitter
from langchain_text_splitters

**3. Embedding**
Convert to vectors

uses

**Embeddings**
HuggingFaceEmbeddings, OpenAIEmbeddings
from langchain_community.embeddings

**4. Vector Storage**
Store in database

uses

**Vector Stores**
Chroma, Pinecone, FAISS, Weaviate
from langchain_community.vectorstores

**5. Retrieval**
Search similar docs

uses

**Retrievers**
VectorStoreRetriever, similarity_search()
Built-in retrieval methods

## 📐 System Architecture

### 🎨 Client Layer

- Claude Desktop App
- MCP Client
- User Interface

### ⚙️ Server Layer

- MCP Server (http)
- Tool Handlers
- LangChain Integration
- Claude API Client

### 💾 Storage Layer

- ChromaDB
- Vector Store
- HuggingFace Embeddings
- Persistent Storage

# Download source

```
#Download source
$> git clone https://github.com/hjkim7796/RAG-CHROMA-MCP-SERVER.git

$> dir RAG-CHROMA-MCP-SERVER

Directory: C:₩hjkim₩PythonProject₩RAG-CHROMA-MCP-SERVER


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        11/21/2025  12:06 PM                .venv
d-----        11/21/2025  11:58 AM                docs
-a----        11/21/2025  11:58 AM             68 .gitignore
-a----        11/21/2025  11:58 AM         227607 abc.pdf
-a----        11/21/2025  11:58 AM          16847 add_pdf_to_mcp.py
-a----        11/21/2025  11:58 AM           5657 mcp-http-proxy.py
-a----        11/21/2025  11:58 AM            383 query-chromaDB.py
-a----        11/21/2025  11:58 AM          20699 rag_mcp_http_server.py
-a----        11/21/2025  11:58 AM           7821 README_HTTP.md
-a----        11/21/2025  11:58 AM            347 requirements.txt
-a----        11/21/2025  11:58 AM           6936 requirements_freeze.txt
-a----        11/21/2025  11:58 AM          11420 test_server.py
-a----        11/21/2025  11:58 AM           2034 test_server_with_proxy.py
```

# Run rag-mcp-server

```
#Go to the source folder
$> cd <user-project-folder>
```

```
#Config a virtual environment (venv) in VS Code for Python to ensure your project uses the correct isolated environment.
$> python -m venv .venv
$> .venv/Scripts/activate.ps1
```

```
#Install all python packages
(.venv)> pip install -r requirements.txt
```

```
#Run rag mcp server
(.venv)> python rag_mcp_http_server.py
```

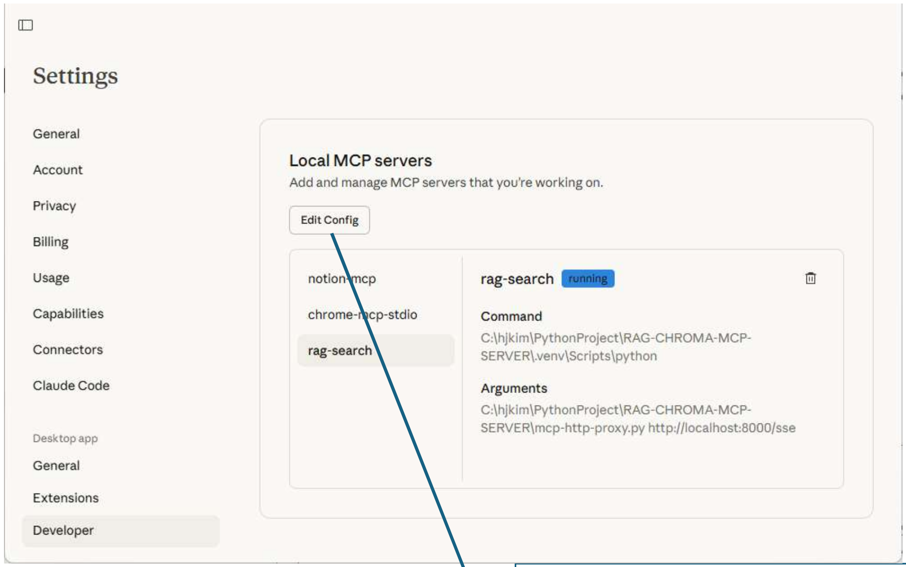
# Test rag-mcp-server

```
#Go to the source folder
(.venv)> cd <user-project-folder>
```

```
#Check server is working properly
(.venv)> python test_server.py
```

```
#Check proxy server is working properly
(.venv)> python test_server_with_proxy.py
```

# Config for Claude Desktop integration



**Local MCP servers**

Add and manage MCP servers that you're working on.

**Command**

C:\hjkim\PythonProject\RAG-CHROMA-MCP-SERVER\.venv\Scripts\python

**Arguments**

C:\hjkim\PythonProject\RAG-CHROMA-MCP-SERVER\mcp-http-proxy.py http://localhost:8000/sse

File location: %APPDATA%₩Claude₩claude_desktop_config.json

```
{
  "mcpServers": {
    "rag-search": {
      "command": "C:₩₩hjkim₩₩PythonProject₩₩RAG-CHROMA-MCP-SERVER₩₩.venv₩₩Scripts₩₩python",
      "args": ["C:₩₩hjkim₩₩PythonProject₩₩RAG-CHROMA-MCP-SERVER ₩₩rag_mcp_server.py"]
  }
}
```

# Test rag-mcp-server

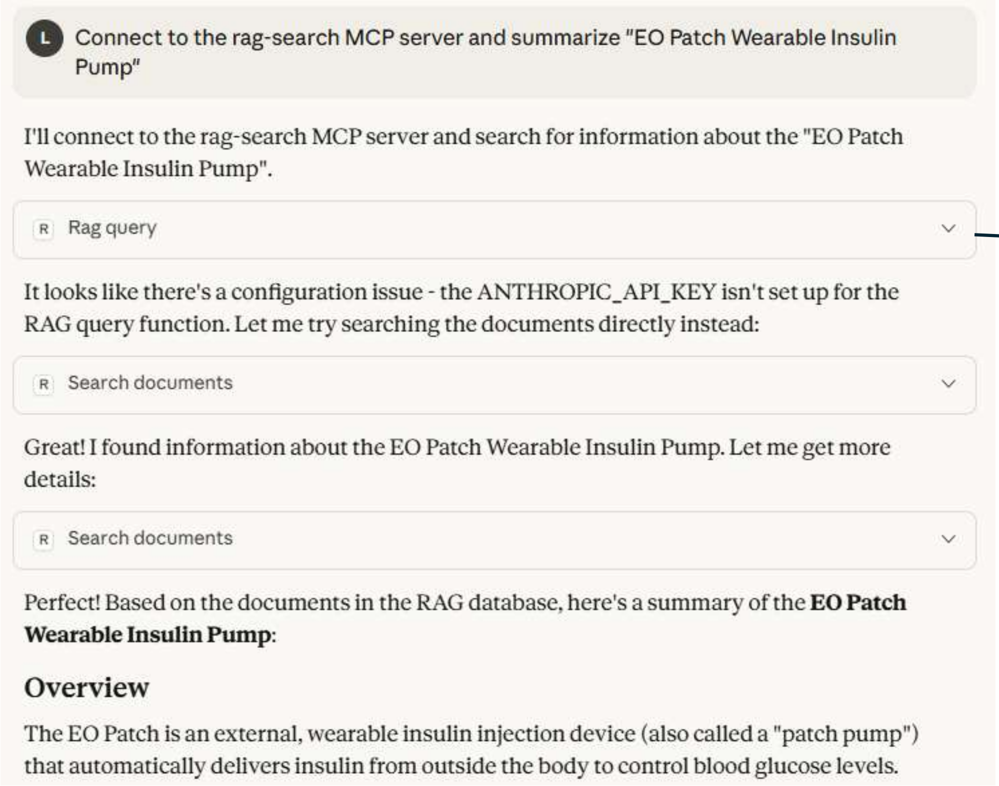
```
#Go to the source folder
(.venv)> cd <user-project-folder>

#Check server is working properly
(.venv)> python test_server.py

#Check proxy server is working properly
(.venv)> python test_server_with_proxy.py
```

# Add a pdf document

```
#Go to the source folder
(.venv)> cd <user-project-folder>

#Add pdf document to RAG system
(.venv)> python add_pdf_to_mcp.py abc.pdf

...

============================================================
✅ PDF successfully added to RAG system!
============================================================
```

# Test query on claude desktop

Query: **Connect to the rag-search MCP server and summarize "EO Patch Wearable Insulin Pump"**



Connect to the rag-search MCP server and summarize "EO Patch Wearable Insulin Pump"

I'll connect to the rag-search MCP server and search for information about the "EO Patch Wearable Insulin Pump".

It looks like there's a configuration issue - the ANTHROPIC_API_KEY isn't set up for the RAG query function. Let me try searching the documents directly instead:

Great! I found information about the EO Patch Wearable Insulin Pump. Let me get more details:

Perfect! Based on the documents in the RAG database, here's a summary of the **EO Patch Wearable Insulin Pump**:

**Overview**

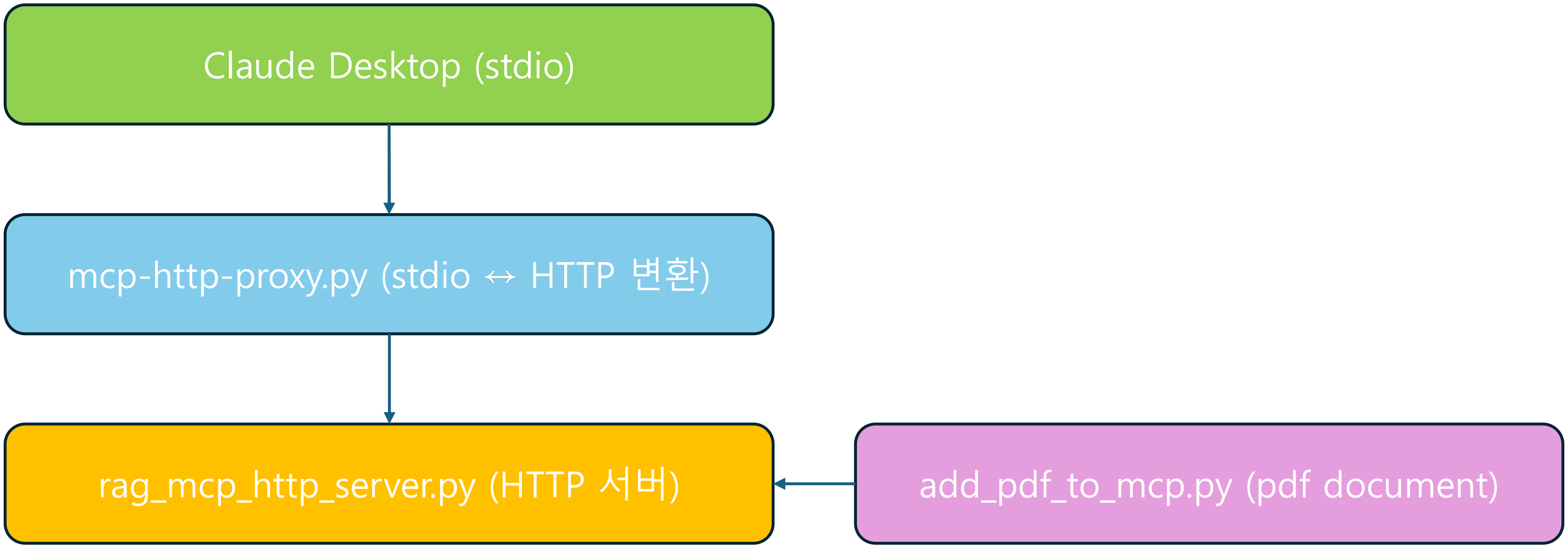
The EO Patch is an external, wearable insulin injection device (also called a "patch pump") that automatically delivers insulin from outside the body to control blood glucose levels.

To enable Rag queries, create a .env file and add the following:
ANTHROPIC_API_KEY=your-api-key-here

# 📦 File Description

- rag_mcp_server.py - Main RAG MCP server
- add_pdf_to_mcp.py – Example for adding a PDF document to RAG system
- query-chromaDB.py – Example for querying to chromaDB
- test_server.py - Example for testing the RAG MCP server
- test_server_with_proxy.py - Example for testing the Proxy server
- requirements.txt – All python package list
- README_HTTP.md - Detailed User Guide

# How it works



Claude Desktop (stdio)

mcp-http-proxy.py (stdio ↔ HTTP 변환)

rag_mcp_http_server.py (HTTP 서버)

add_pdf_to_mcp.py (pdf document)

# 🚀 Key Features

- **add_documents** - Adding documents to the vector DB
- **search_documents** - Similarity-based search
- **rag_query** - Query via RAG (using Claude), Need API key
- **delete_collection** - Delete collection

## 🛠️ Available MCP Tools

### ① add_documents

Add documents to the vector database. Automatically handles chunk splitting and embedding.

**Parameters:**

- texts: Array of document texts
- metadatas: Array of metadata objects
- chunk_size: Chunk size (default: 1000)
- chunk_overlap: Overlap size (default: 200)

### ② search_documents

Search for similar documents in the vector database using cosine similarity.

**Parameters:**

- query: Search query string
- k: Number of results (default: 4)
- filter: Optional metadata filter

### ③ rag_query

Answer questions using RAG. Retrieves relevant documents and generates answers with Claude.

**Parameters:**

- question: Question text
- k: Number of docs to search (default: 4)
- language: Answer language (ko/en)

### ④ get_collection_info

Retrieve current collection information including document count and storage location.

**Parameters:**

- None (no parameters required)

### ⑤ delete_collection

Delete entire collection. All documents and embeddings will be permanently removed.

**Parameters:**

- confirm: Confirmation flag (must be true)

## ✨ Key Features

### 🔤 Automatic Text Chunking

Documents are automatically split into manageable chunks (1000 chars with 200 char overlap) while preserving semantic coherence.

### 🧠 Smart Embeddings

Uses sentence-transformers/all-MiniLM-L6-v2 model for high-quality embeddings. Runs locally without API keys.

### 💾 Persistent Storage

ChromaDB stores vectors persistently on disk. Data survives server restarts and can be queried anytime.

### 🎯 Similarity Search

Cosine similarity-based vector search efficiently finds the most relevant documents for any query.

### 🏷️ Metadata Management

Store and filter documents by custom metadata (source, category, tags, etc.) for precise retrieval.

### 🌍 Multilingual Support

Supports multiple languages through HuggingFace embeddings and Claude's multilingual capabilities.

## 💻 Technology Stack

| | |
|---|---|
| **LangChain** | v1.0.7 |
| **ChromaDB** | v1.3.5 |
| **Claude AI** | Sonnet 4 |
| **HuggingFace** | Transformers v5.1.2 |
| **MCP** | v1.21.2 |
| **Python** | v3.12 |