# Introduction

Those notes are about IT tools used for building AI agents. Related notes about machine learning theory (NLP) can be found in the ‘machine learning > NLP > NLP notes’ file.

# MCP

MCP is a protocol which enables LLMs to use different tools.

There are two components of MCP:

* MCP server – App where we define different tools which can be used by LLMs.
* MCP client – App where LLM is using tools from a MCP server

Both components are explained in more detail in sections below.

## MCP server

MCP server is an app which enables using different tools (like accessing a database or using different functions). For example:

A screen shot of a computer program

AI-generated content may be incorrect.

It is similar to Rest API. It is a long running program (it stops when we stop it) and when other app (a MCP client) sends a request, it sends a response.

In a request we tell what tool we want to use and send parameters for this tool.

As a response we receive an output from this tool for those parameters.

## MCP client

MCP client is an app which uses tools defined in a MCP server.

For example here is a function which uses a tool from a MCP server:

A computer screen shot of a program code

AI-generated content may be incorrect.

In this example, in order to make this function work, the MCP server must be running on the same server where we run this function.

# Agent deciding which MCP tool to use

We can create an app where LLM take a request from a user and decides whether or not to use some tool available in the MCP server, which one and what parameters to use.

For example here we have:

* call\_tool function for using a tool from a MCP server
* agent\_loop function which:
  + Takes user question as an input
  + LLM decides whether or not to use the tool
  + If LLM decides to use the tool, it provides parameters to use in this tool
  + If LLM decides not to use the tool, it generates the final answer

A screen shot of a computer program

AI-generated content may be incorrect.

# Multi agent orchestration

Here is an example using LangGraph to orchestrate two agents:

* Retriever Agent → embeds the query and retrieves documents from a vector DB
* Generator Agent → generates a final answer based on the retrieved documents

Here we use FAISS to create a vector db which stores in memory embeddings of provided docs:

A screenshot of a computer

AI-generated content may be incorrect.

Then we create two agents:

* Retriever
  + Takes a question as an input
  + Embeds the question
  + Retrieves the docs related to the question
* Generator
  + Takes a question and retrieved docs as an input
  + Generates an answer

A screenshot of a computer screen

AI-generated content may be incorrect.

Then we build the graph which specifies how data flows between agents. Here we specify that the retriever’s output is the generator’s input:

A screenshot of a computer program

AI-generated content may be incorrect.

# Semantic search

Semantic search is about finding text with a similar meaning to the given, another text.

It is done by converting sentences into vectors (embedding vectors) which represent their meaning, and then comparing how similar are those vectors.

Sentences with similar embedding vectors have similar meaning.

Below are described different techniques for performing semantic search.

## Encoding a question and searching a vector db

In this method for performing semantic search we have two steps:

* Create a vector db
* Find documents in a vector db similar to the given question

Here is more detailed breakdown of each step:

**Create a vector db**

* Convert documents into vector embeddings using LLM
* Add embeddings to a vector db

**Find documents in a vector db similar to the given question**

* Convert a question into a vector embedding using LLM
* Find in a vector db docs with vectors similar the question’s vector

## Vector dbs

Here is a short description of vector databases. More detailed information can be found in the ‘ml engineering > vector databases’ folder in this knowledge base.

There are two ways how vector dbs can store their data:

* In memory
* On disk

Example tools for creating vector dbs:

* FAISS – Stores data only in memory
* Chroma – Can store data in both memory and on disk. It also provides a Rest API (using the Chroma Server mode) for interacting with vector db.

## Serving searching vector dbs – Rest API / MCP

If our vector db stores data in memory, then creating it for every question is not practical.

Instead we need to create a long running service which builds a vector db once, and enables other apps to search through this db.

We can create for example a Rest API or MCP server to which other apps can send a question and it will send similar docs from a vector db as a response.

We can create this Rest API on our own or use a vector db which provides such an API, for example Chroma.

Here is an example script of creating our own Rest API where we use FAISS as a vector db. It consists of two parts.

In the first part we:

* Convert docs into vector embeddings
* Save embeddings in a vector db
* Save vector db in a file

A screen shot of a computer program

AI-generated content may be incorrect.

In the second part we:

* Load the vector db from a file
* Create a Rest API where we find docs similar to a question

Finding similar docs is done in the following way:

* Convert a question into a vector embedding
* Find similar vectors in a vector db

A computer screen shot of a program

AI-generated content may be incorrect.

There are also tools for building vector dbs which provides Rest APIs and which can store data on a disk.