인공지능특론

ARTIFICIAL INTELLIGENCE PRACTICE

Agentic AI II - LLMOPs

이건영 교수

학습목차

- LLMOPs – Backend, Frontend, LangChain
- Agent & Tool

학습목표

- LLM 개발 프레임워크인 LnagChain을 활용해 특정 문서를 ChromaDB Vector Store에 저장하고, Retrieval Augmented Generation(RAG)를 활용, LLM 모델 기반의 질의응답 시스템 구축 및 구현할 수 있다.
- Agent와 Tool에 대해 이해할 수 있다.

TODAY'S LESSON

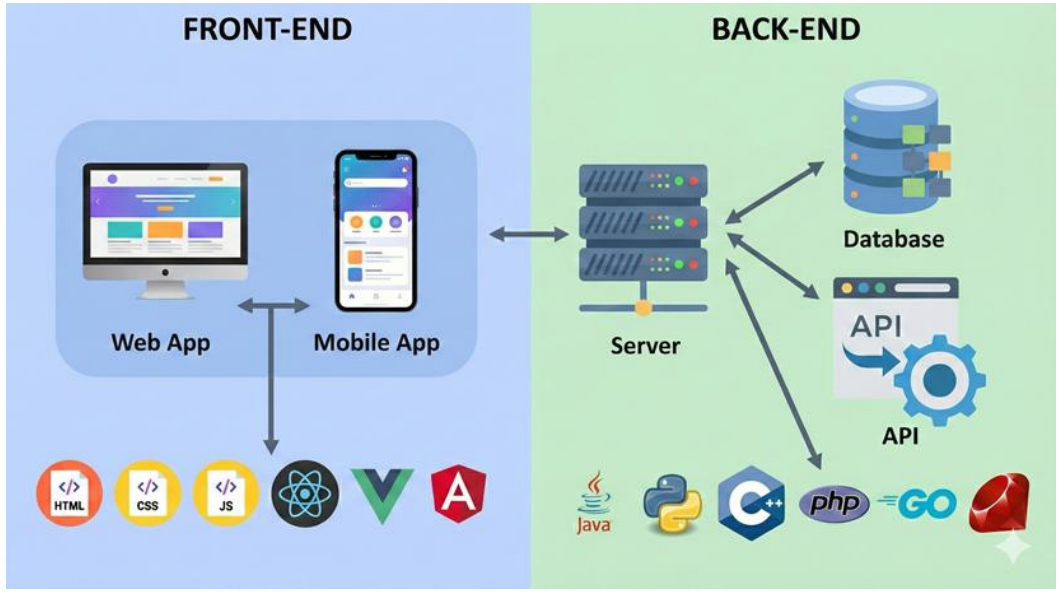
LLM Chatbot & LLMOPs

인공지능특론

1) LLM OPs – Backend & FrontEnd

Backend & FrontEnd란?

In [software engineering](), the terms **frontend** and **backend** refer to the [separation of concerns]() between the [presentation layer]() (*frontend*), and the [data access layer]() (*backend*) of a piece of [software](), or the physical infrastructure or [hardware]().
In the [client–server model](), the [client]() is usually considered the frontend and the [server]() is usually considered the backend, even when some presentation work is actually done on the server itself.

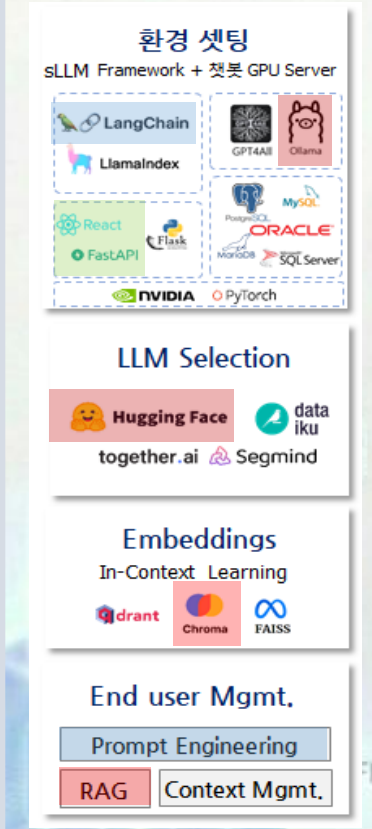


1. 프론트엔드는 웹 사이트나 앱에서 우리 눈에 보이는 "앞단“
2. 웹 사이트 방문시 시각적인 디자인과 상호작용
3. 웹 페이지의 레이아웃, 색상, 글꼴, 버튼 등의 디자인

1. 백엔드는 웹 사이트나 앱에서 눈에 보이지 않는 "뒷단"
2. 사용자가 볼 수 없는 영역인 데이터베이스나 서버를 관리

FastAPI

React



<출처1>

1) LLM OPs – Backend & FrontEnd

Backend & FrontEnd란?

Frontend focuse

•Markup and web languages such as HTML, CSS, JavaScript, and ancillary libraries commonly used in those languages such as Sass or jQuery

•Asynchronous request handling and AJAX

•Single-page applications (with frameworks like React, Angular or Vue.js)

•Web performance (largest contentful paint, time to interactive, 60 FPS animations and interactions, memory usage, etc.)

•Responsive web design

•Cross-browser compatibility issues and workarounds

•End-to-end testing with a headless browser

•Build automation to transform and bundle JavaScript files, reduce image sizes and other processes using tools such as Webpack and Gulp.js

•Search engine optimization

•Accessibility concerns

•User Interface

Backend focused

•Scripting languages like PHP, Python, Ruby, Perl, Node.js, or Compiled languages like C#, Java or Go

•Data access layer

•Business logic

•Database administration

•High availability

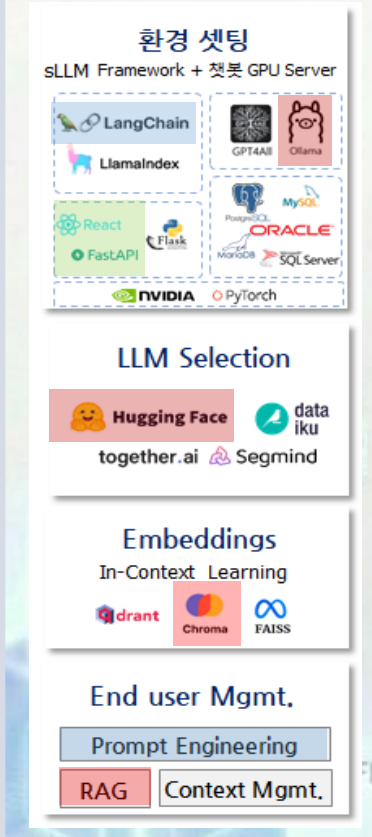
•Security concerns, authentication and authorization

•Backup methods and software

•Software Architecture

•Data transformation

•Scalability



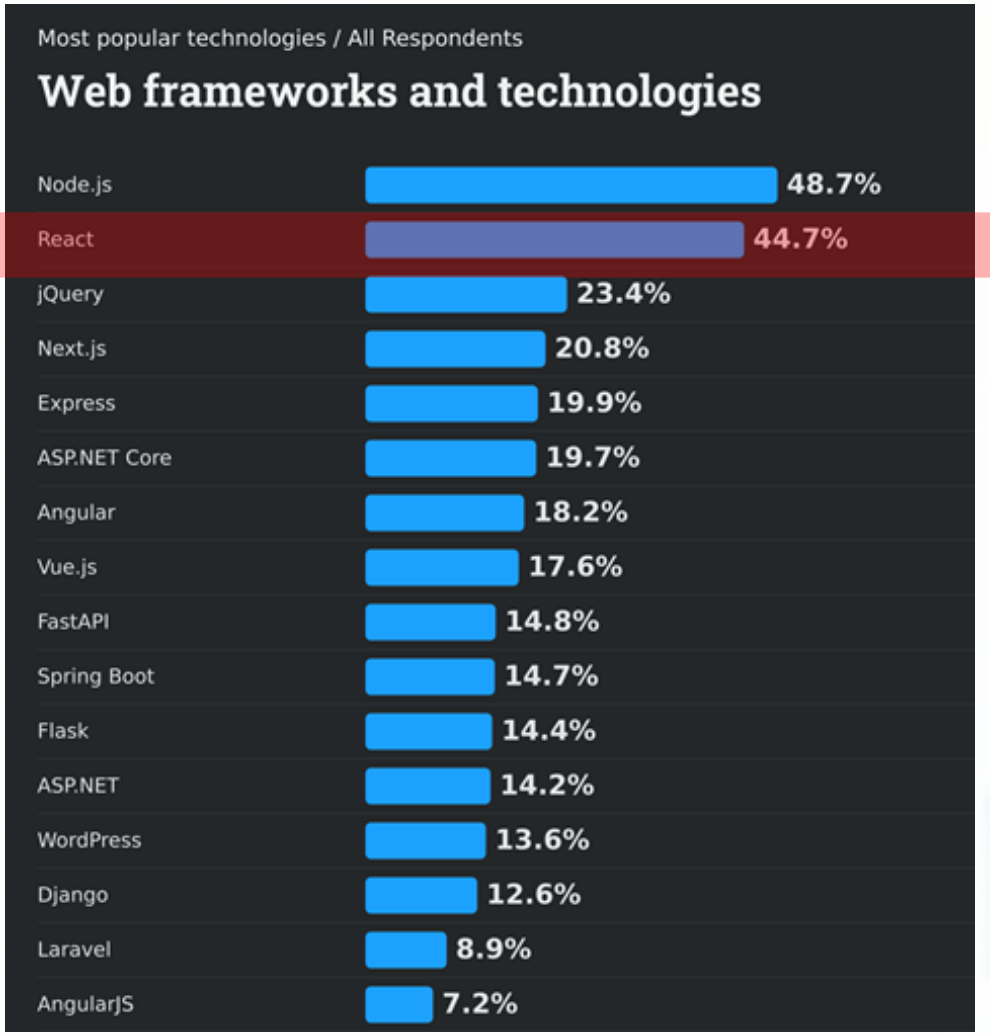
<출처2>

1) LLM OPs – Backend & FrontEnd

FrontEnd - React

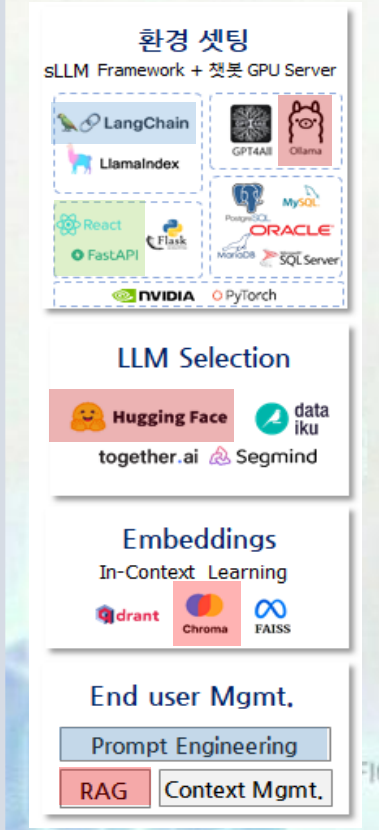
React (also known as **React.js** or **ReactJS**) is a free and open-source front-end JavaScript library[5][6] that aims to make building user interfaces based on components more "seamless".[5] It is maintained by Meta (formerly Facebook) and a community of individual developers and companies.[7][8][9] According to the 2025 Stack Overflow Developer Survey, React is one of the most commonly used web technologies.[10]

React can be used to develop single-page, mobile, or server-rendered applications with frameworks like Next.js and React Router. Because React is only concerned with the user interface and rendering components to the DOM, React applications often rely on libraries for routing and other client-side functionality.[11][12] A key advantage of React is that it only re-renders those parts of the page that have changed, avoiding unnecessary re-rendering of unchanged DOM elements. React is used by an estimated 6% of all websites.[13]



FastAPI

React



<출처3,4>

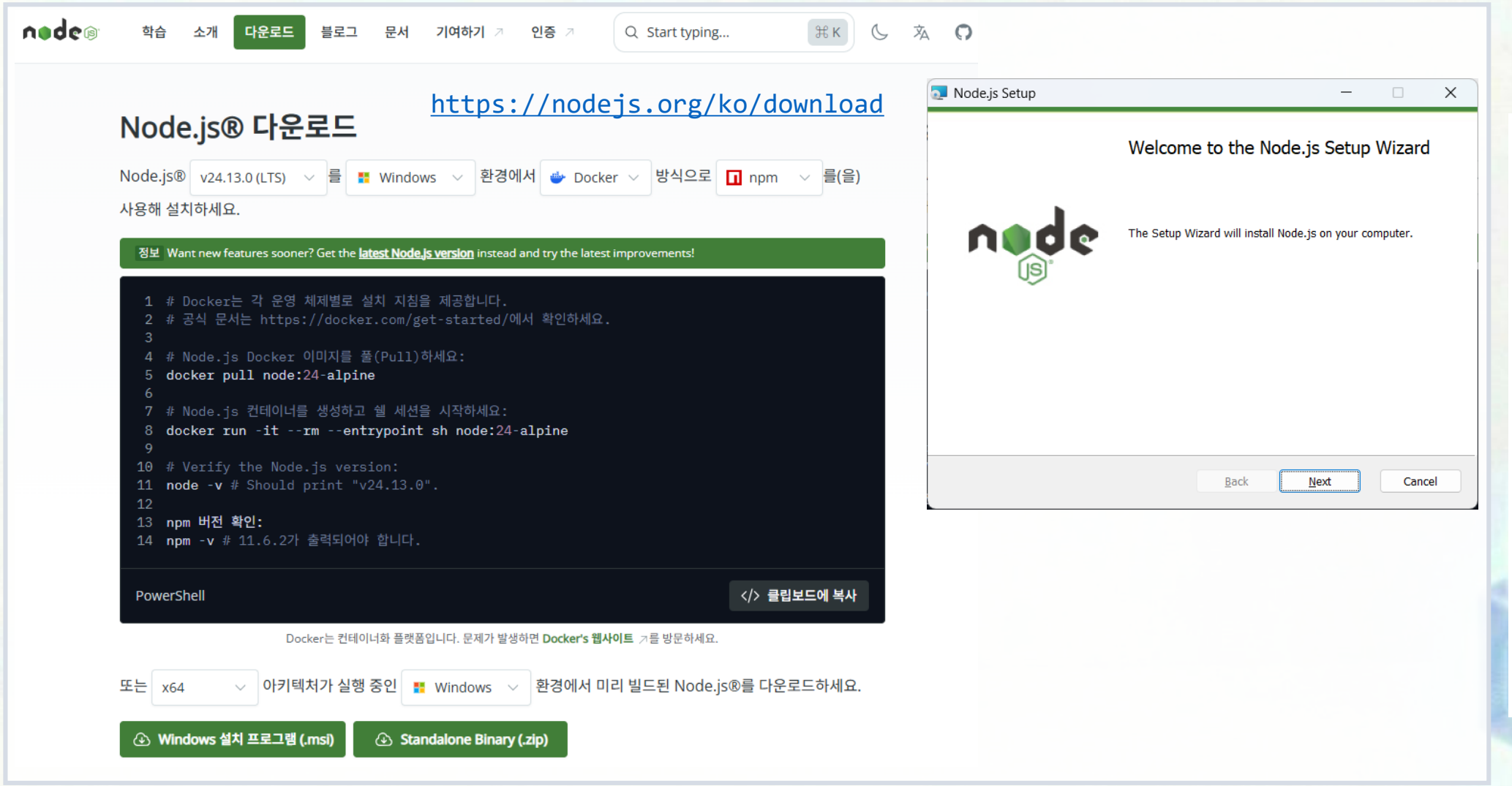
1) LLM OPs – Backend & FrontEnd

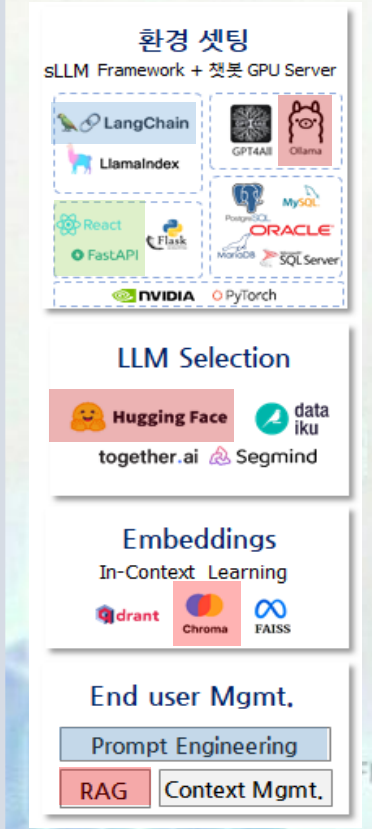
FrontEnd – React Install

FastAPI

React

https://nodejs.org/ko/download





1) LLM OPs – Backend & FrontEnd

Backend - FastAPI

FastAPI is a high-performance web framework for building HTTP-based service APIs in Python 3.8+.[3] It uses Pydantic and type hints to validate, serialize and deserialize data. FastAPI also automatically generates OpenAPI documentation for APIs built with it.[4] It was first released in 2018.

Components [edit]

Pydantic [edit]

Pydantic is a data validation library for Python. While writing code in an IDE, Pydantic provides type hints based on annotations.[5] FastAPI extensively utilizes Pydantic models for data validation, serialization, and automatic API documentation. These models are using standard Python type hints, providing a declarative way to specify the structure and types of data for incoming requests (e.g., HTTP bodies) and outgoing responses.[6]

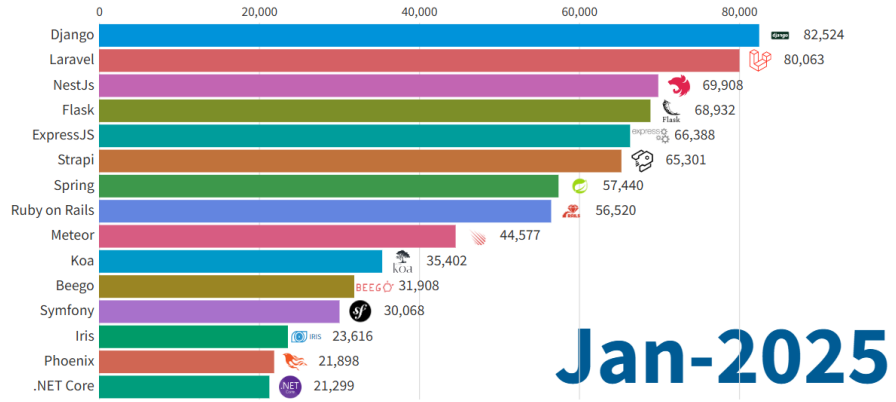
```python
from fastapi import FastAPI
from pydantic import BaseModel

app = FastAPI()

class Item(BaseModel):
    name: str
    price: float
    is_offer: bool | None = None
```

FastAPI

FastAPI

Developer	Sebastián Ramírez
Initial release	December 5, 2018; 7 years ago[1]
Stable release	0.124.4[2] / 12 December 2025; 32 days ago
Repository	github.com/tiangolo/fastapi
Written in	Python
Type	Web framework
License	MIT
Website	fastapi.tiangolo.com

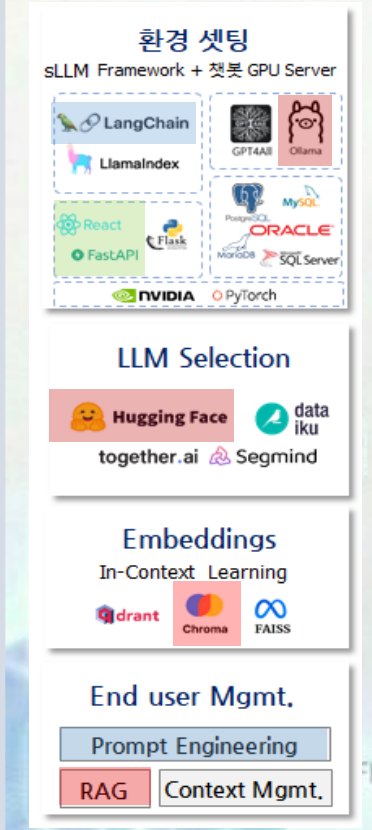
Most Popular Backend Frameworks



Jan-2025

FastAPI

React



<출처5,6>

1) LLM OPs – Backend & FrontEnd

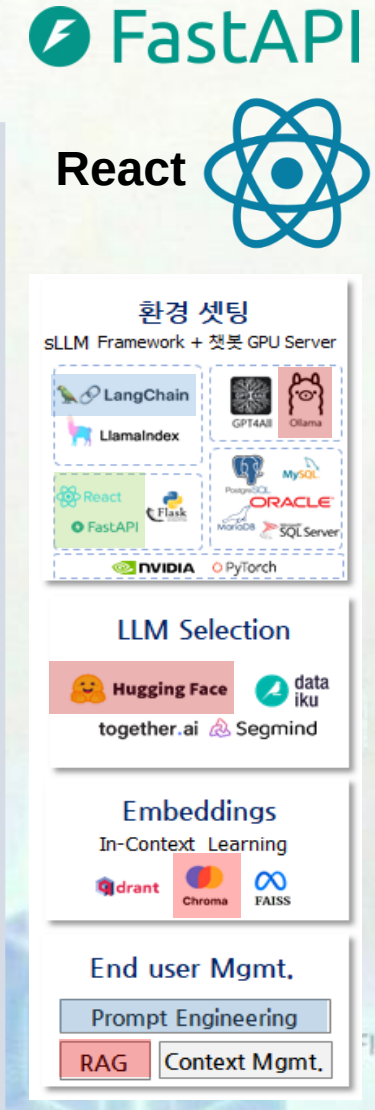
Backend – FastAPI install check

```
PS D:\git\50_AgenticAI> python -m venv venv
PS D:\git\50_AgenticAI> .\venv\Scripts\activate
(venv) PS D:\git\50_AgenticAI> cd .\backend\
(venv) PS D:\git\50_AgenticAI\backend> pip install -r .\requirements.txt
Collecting fastapi[standard]
  Downloading fastapi-0.128.0-py3-none-any.whl (103 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 103.1/103.1 kB 1.2 MB/s eta 0:00:00
...
(venv) PS D:\git\50_AgenticAI\backend> cd .\src\
(venv) PS D:\git\50_AgenticAI\backend\src> python .\embedding.py
```

앞에서 이미 install

```
# FastAPI 관련
fastapi[standard]
uvicorn>=0.24.0
python-dotenv>=1.0.0
pydantic>=2.4.2

# LangChain 관련
langchain>=0.2.2
langchain-core>=0.1.1
langchain-community>=0.0.10
langchain-openai>=0.0.2
langchain-chroma>=0.0.1
langchain-huggingface>=0.0.3

# 문서 처리 관련
chromadb>=0.4.18
pdf2image>=1.16.3
pypdf>=5.1.0

# AI/ML 관련
torch==2.7.1
transformers>=4.35.2
sentence-transformers>=2.2.2
```

```
기존 벡터 스토어 삭제됨: ../vector_store
embedding.py:24: LangChainDeprecationWarning: The class `HuggingFaceEmbeddings` was deprecated in LangChain 0.2.2 and will be removed in 1.0. An updated
version of the class exists in the langchain-huggingface package and should be used instead. To use it run `pip install -U langchain-huggingface` and
import as `from langchain_huggingface import HuggingFaceEmbeddings`.
  embeddings_model = HuggingFaceEmbeddings(model_name="sentence-transformers/all-MiniLM-L6-v2")
modules.json: 100%|| 349/349 [00:00<00:00, 30.3kB/s]
config_sentence_transformers.json: 100%|| 116/116 [00:00<00:00, 10.3kB/s]
README.md: 100%|| 10.7k/10.7k [00:00<00:00, 5.98MB/s]
sentence_bert_config.json: 100%|| 53.0/53.0 [00:00<00:00, 33.7kB/s]
config.json: 100%|| 612/612 [00:00<00:00, 405kB/s]
model.safetensors: 100%|| 90.9M/90.9M [00:02<00:00, 33.1MB/s]
tokenizer_config.json: 100%|| 350/350 [00:00<00:00, 37.2kB/s]
vocab.txt: 100%|| 232k/232k [00:00<00:00, 634kB/s]
tokenizer.json: 100%|| 466k/466k [00:00<00:00, 2.52MB/s]
special_tokens_map.json: 100%|| 112/112 [00:00<00:00, 68.5kB/s]
1_Pooling/config.json: 100%|| 190/190 [00:00<00:00, 22.5kB/s]
```

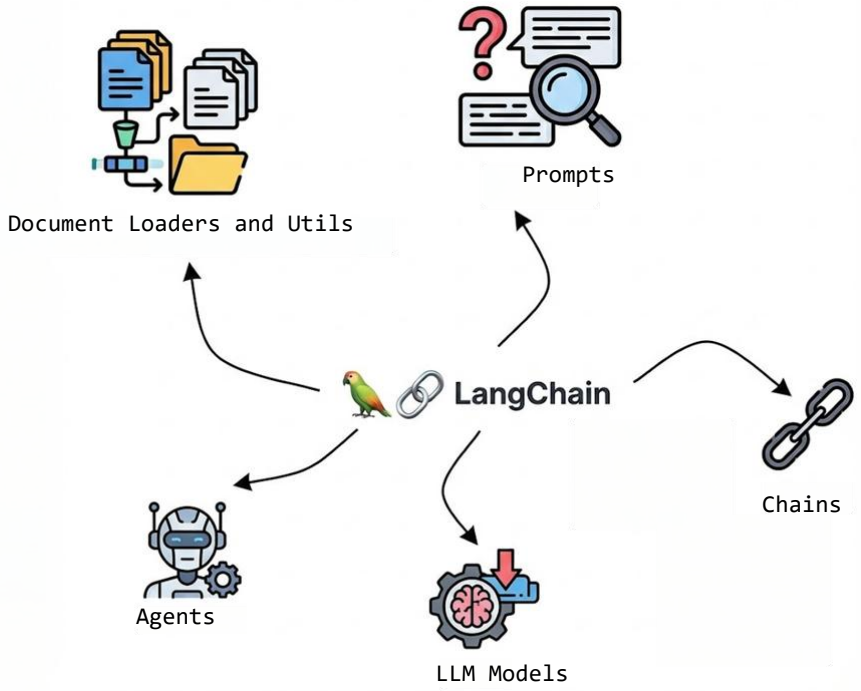


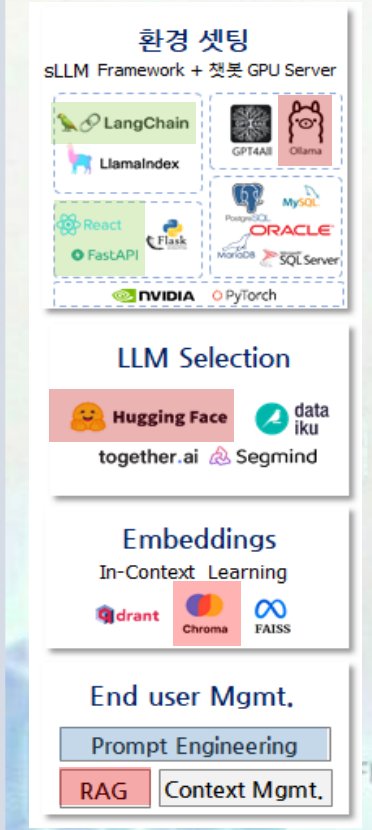
1) LLM OPs – LangChain

LangChain

LangChain

LangChain is a software framework that helps facilitate the integration of large language models (LLMs) into applications. As a language model integration framework, LangChain's use-cases largely overlap with those of language models in general, including document analysis and summarization, chatbots, and code analysis.[2]





<출처7>

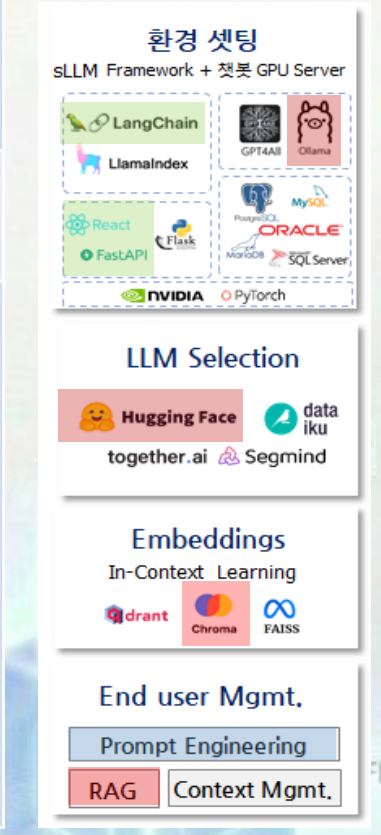
1) LLM OPs – LangChain

LangChain vs LlamaIndex

LangChain LlamaIndex

LangChain

특성	LangChain Chain	LlamaIndex Query Engine
아키텍처 접근방식	모듈형 파이프라인 구조. 각 컴포넌트를 **체이닝**하여 원하는 워크플로우 구성	**통합된 쿼리 처리 시스템**. 인덱스 구조에 최적화된 쿼리 실행
구성방식	모델 설정 – vector store 저장 – 쿼리 질문이라는 전체적인 구성 유사	
	1. 모델 api 선정 2. document 불러오기(LocalFileStore) 3. 파일 load 4. 사용자가 CharacterTextSplitter를 사용해서 적절히 chunk 단위로 parsing 5. 임베딩을 통해서 vector store 저장 6. 템플릿과 **chain** 방식을 선정해서 chain을 구성 7. 쿼리 전달	1. api key 2. 데이터 문서 다운로드 3. 데이터 문서 load(SimpleDirectoryReader) 4. model 선정 5. chunk 단위로 parsing(꼭 필요한 작업은 아님) 6. vector store의 vector index 정의 (**VectorStoreIndex**) 7. **QueryEngine** 빌드 및 Query 시작



1) LLM OPs – LangChain

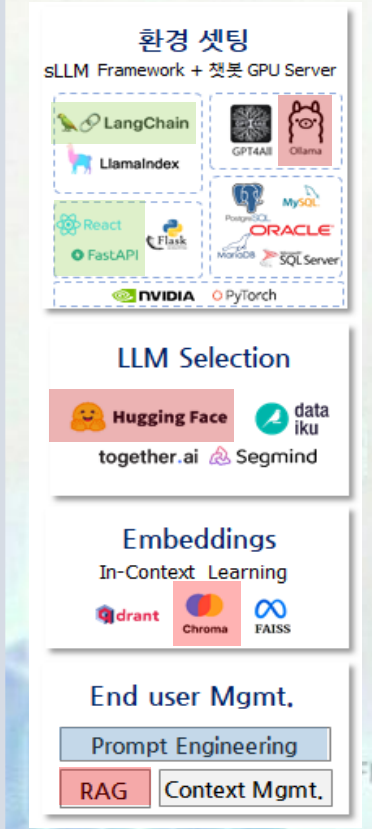
LangChain vs LlamaIndex

LangChain LlamaIndex

LangChain

- **Choose LangChain** if you need a flexible framework to support complex workflows where intricate interaction and context retention are highly prioritized.
- **Choose LlamaIndex** if your primary need is data retrieval and search capabilities for applications that handle large volumes of data that require quick access.

Feature	LangChain	LlamaIndex
Primary Focus	Flexible LLM-powered application development	Search and retrieval
Data Indexing	Modular and customizable	Highly efficient
Retrieval Algorithms	Integrated with LLMs for context-aware outputs	Advanced and optimized
User Interface	Comprehensive and adaptable	Simple and user-friendly
Integration	Supports diverse AI technologies and services	Multiple data sources, seamless platform integration
Customization	Extensive, supports complex workflows	Limited, focused on indexing and retrieval
Context Retention	Advanced, crucial for chatbots and long interactions	Basic
Use Cases	Customer support, content generation, code documentation	Internal search, knowledge management, enterprise solutions
Performance	Efficient in handling complex data structures	Optimized for speed and accuracy
Lifecycle Management	Comprehensive evaluation suite (LangSmith)	Integrates with debugging and monitoring tools

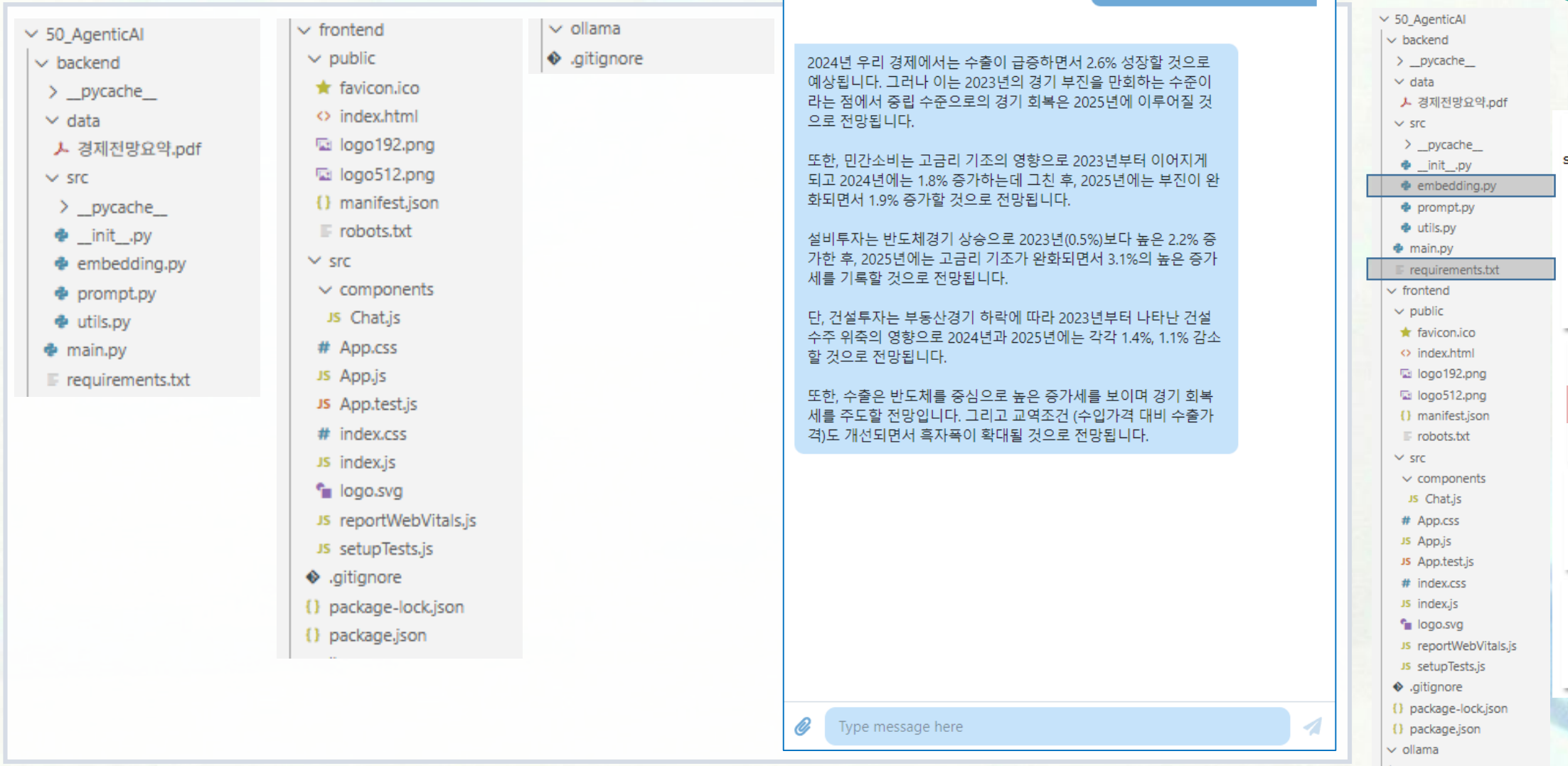


<출처8>

1 LLM OPs

1) LLM OPs – Program

FrontEnd

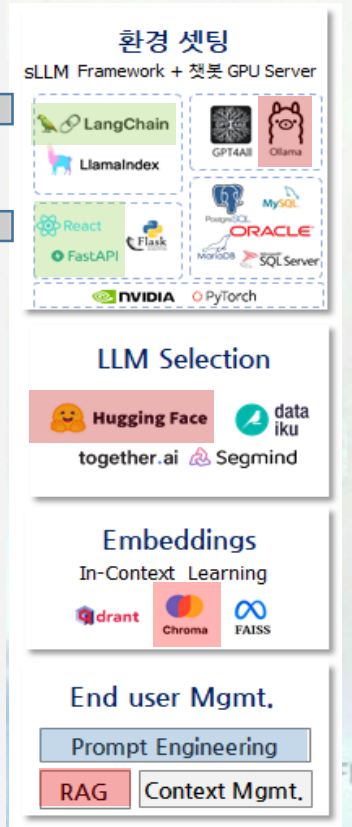


인공지능특론

LangChain

FastAPI

React



1 LLM OPs

1) LLM OPs – Program

FrontEnd 1 – 기본 소스 파일

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);

reportWebVitals();
```

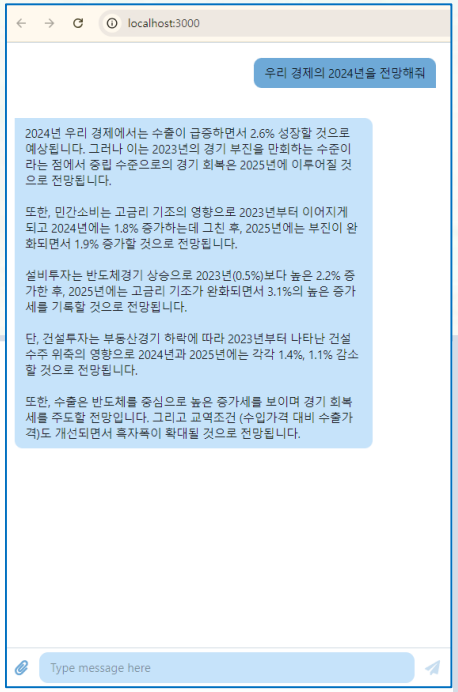
src/index.js

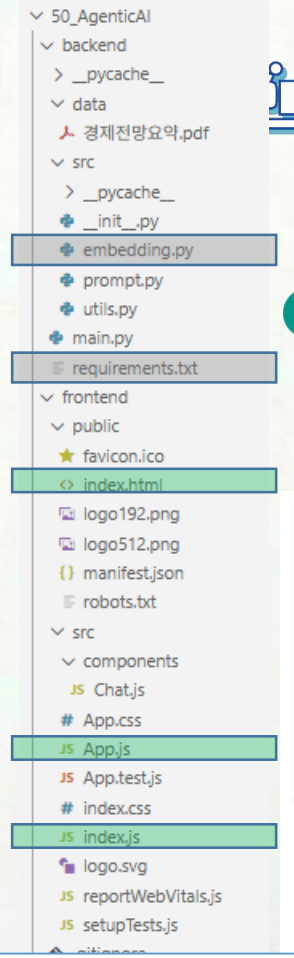
```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <title>React App</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
  </body>
</html>
```

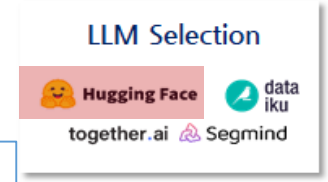
public/index.html

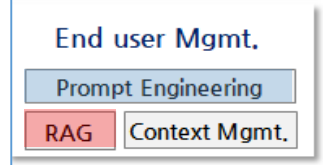
```
import logo from "./logo.svg";
import "./App.css";
import Chat from "./components/Chat";

function App() {
  return (
    <div className="App" style={{ height: "100%", width: "100%" }}>
      <Chat />
    </div>
  );
}

export default App;
```

src/App.js







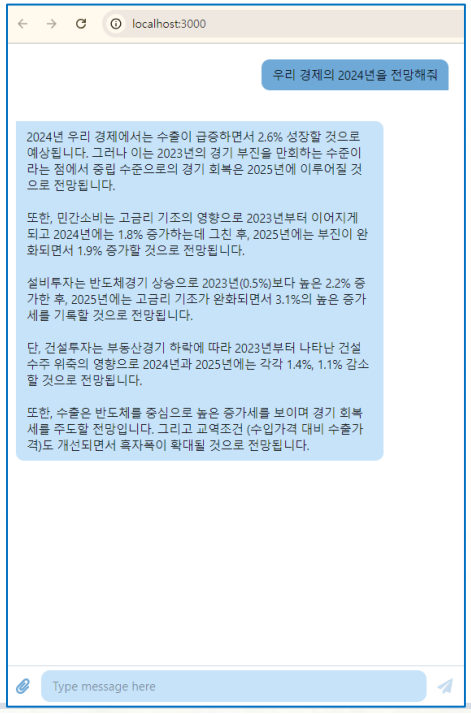


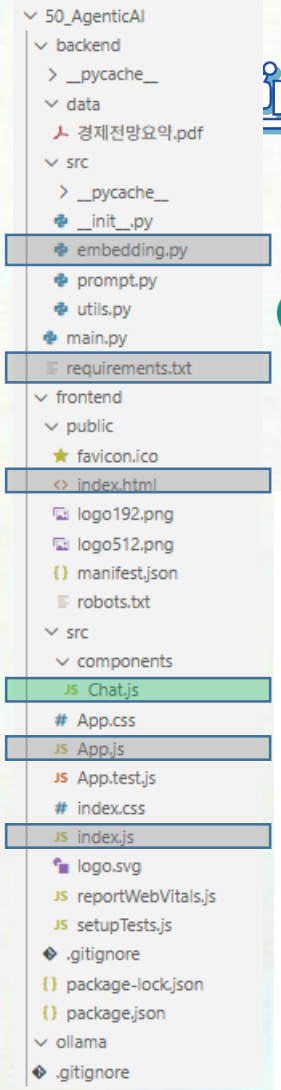
1 LLM OPs

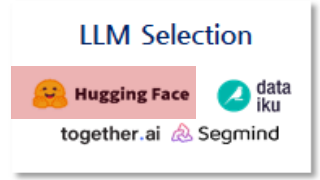
1) LLM OPs – Program

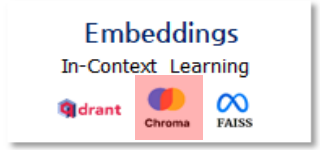
FrontEnd 2 – Chat.js 1/2

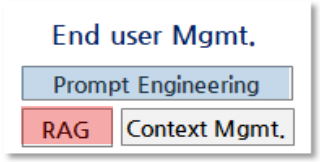
```
import styles from "@chatscope/chat-ui-kit-styles/dist/default/styles.min.css";
import {
  MainContainer,
  ChatContainer,
  MessageList,
  Message,
  MessageInput,
} from "@chatscope/chat-ui-kit-react";     // 챗 UI 구성을 위한 컴포넌트들
import { useState } from "react";          // React의 상태 관리를 위한 훅
import axios from "axios";                 // 백엔드 API와의 통신을 위한 라이브러리

const Chat = () => {
  // 메시지 내역을 저장하는 상태(State) 변수
  const [messages, setMessages] = useState([]);
  // {direction, content, sentTime, sender} 객체들의 배열

  return (
    // 3. 전체 화면을 꽉 채우는 컨테이너 설정 (높이 100vh)
    <div style={{ position: "relative", height: "100vh" }}>
      <MainContainer>
        <ChatContainer>
          {/* 4. 대화 내용이 표시되는 리스트 영역 */}
          <MessageList>
            {/* messages 배열을 순회(map)하며 각각의 Message 컴포넌트를 생성 */}
            {messages.map((message) => (
              <Message
                style={{ padding: "1rem 0" }}
                model={{
                  direction: message.direction, // "incoming" 또는 "outgoing"
                  message: message.content,
                  sentTime: message.sentTime,
                  sender: message.sender,       // "user" 또는 "ai"
                  position: "single",
                }}
              />
            ))}
          </MessageList>
```





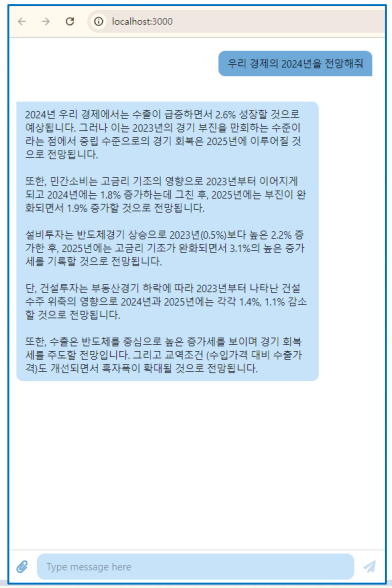


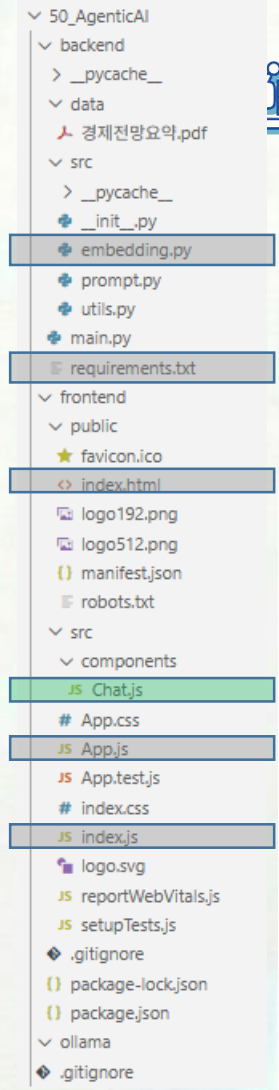


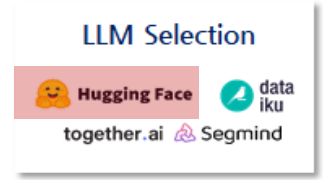


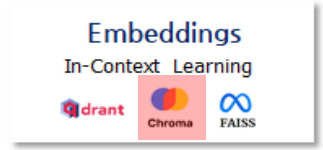
1) LLM OPs – Program FrontEnd 2 – Chat.js 2/2

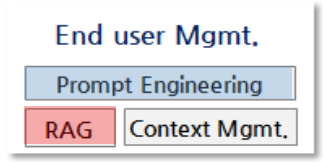
```
          {/* 5. 사용자가 메시지를 입력하는 창 */}
          <MessageInput
            placeholder="Type message here"
            onSend={async (innerHtml, textContent, innerText) => {
              // [A] 사용자가 보낸 메시지를 즉시 화면(상태)에 반영
              setMessages((prev) => [
                ...prev,      // 기존 메시지 유지
                {
                  direction: "outgoing",  // 사용자가 보낸 메시지(오른쪽 정렬)
                  content: innerText,
                  sentTime: new Date(),
                  sender: "user",
                },
              ]);

              // [B] 백엔드 서버(Localhost 8000번 포트)로 메시지 전송
              // 사용자의 질문을 'question'이라는 키에 담아 POST 요청을 전송
              const response = await axios.post("http://localhost:8000/chat", {
                question: textContent,
              });

              // [C] 서버로부터 받은 응답(AI 답변)을 화면(상태)에 추가
              setMessages((prev) => [
                ...prev,
                {
                  direction: "incoming",          // AI가 보낸 메시지(왼쪽 정렬)
                  content: response.data.answer,  // 서버에서 보내준 응답 데이터
                  sentTime: new Date(),
                  sender: "ai",
                },
              ]);
            }}
            />
          </ChatContainer>
        </MainContainer>
      </div>
    );
};

export default Chat;
```









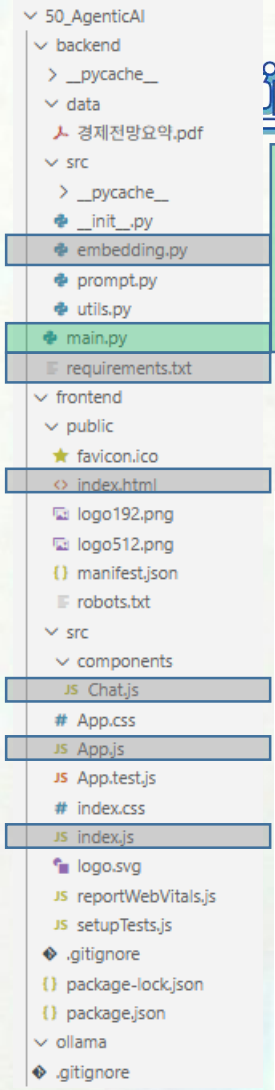


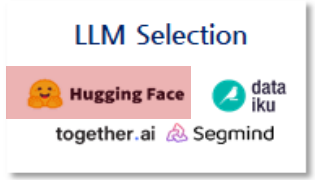
1 LLM OPs

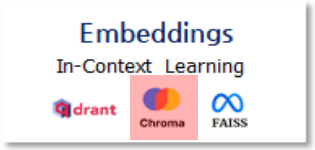
1) LLM OPs – Program

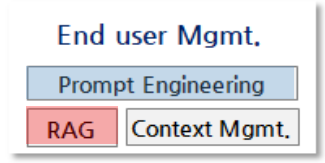
BackEnd & LangChain – main.js 1/2

```
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from pydantic import BaseModel
from langchain_core.output_parsers import StrOutputParser   # 문자열 출력 파서
from langchain_core.runnables import RunnablePassthrough    # 입력을 그대로 출력하는 러너블
# from langchain_openai import OpenAI, OpenAIEmbeddings
from langchain_ollama import ChatOllama
from langchain_huggingface import HuggingFaceEmbeddings     # HuggingFace 임베딩 모델
from langchain_chroma import Chroma
from src.utils import format_docs   # 검색된 문서들을 텍스트 형식으로 변환하는 유틸 함수
from src.prompt import prompt       # 프롬프트 템플릿
from dotenv import load_dotenv      # .env 환경변수 로드

load_dotenv()       # .env 파일에 정의된 API 키나 환경 변수들을 로드
app = FastAPI()     # [1]. FastAPI 앱 생성

# [2]. 언어 모델(LLM) 설정
# llm = OpenAI(
#     model_name="gpt-3.5-turbo-instruct",
#     temperature=0.2,  # 응답의 창의성(무작위성) 조절
#     max_tokens=512,
#     streaming=True
# )
llm = ChatOllama(model="mistral:latest")

# [3]. 임베딩 모델 설정
# embeddings_model = OpenAIEmbeddings()
# HuggingFaceEmbeddings 초기화
embeddings_model = HuggingFaceEmbeddings(model_name="sentence-transformers/all-MiniLM-L6-v2")
```









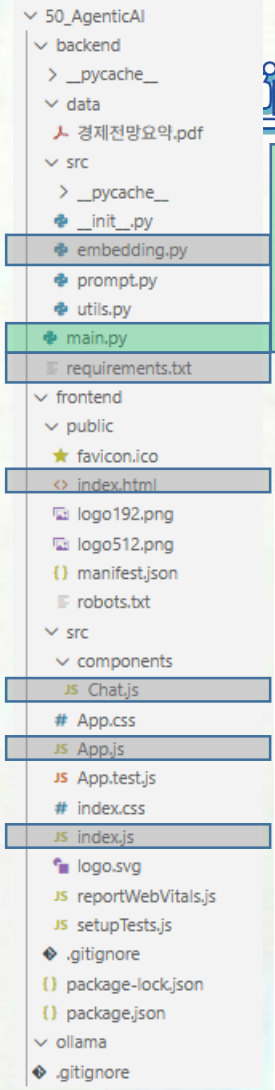
1 LLM OPs

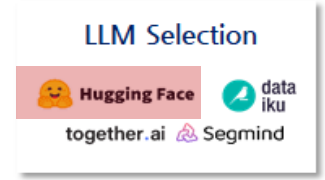
1) LLM OPs – Program

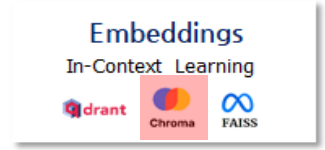
BackEnd & LangChain – main.js 2/2

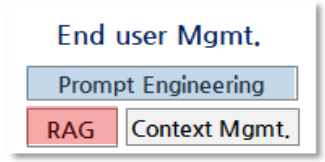
```
# [4]. Vector Store 및 Retriever 설정
# db = Chroma(persist_directory="./vector_store", embedding_function=OpenAIEmbeddings())
db = Chroma(persist_directory="./vector_store", embedding_function=embeddings_model)
retriever = db.as_retriever(search_type="similarity")   # 유사도(similarity) 기반 검색

# [5]. CORS 설정 (프런트엔드와 백엔드 통신 허용)
origins = [
    "http://localhost",
    "http://localhost:3000",
]

app.add_middleware(         # CORS 설정 (프런트엔드와 백엔드 통신 허용)
    CORSMiddleware,
    allow_origins=origins,
    allow_credentials=True, # 쿠키 허용
    allow_methods=["*"],    # GET, POST 등 모든 메소드 허용
    allow_headers=["*"],    # 모든 헤더 허용
)

# [6]. 데이터 모델 정의
class UserQuery(BaseModel):
    """user question input model"""
    question: str

# [7]. RAG 체인(Chain) 구성 - 핵심 로직
rag_chain = (
    {"context": retriever | format_docs, "question": RunnablePassthrough()}
    | prompt
    | llm
    | StrOutputParser()  # 최종 출력 파서 설정
)
```

```
# [8]. 챗봇 엔드포인트 정의
@app.post("/chat/")
async def chat(query: UserQuery):
    """chat endpoint"""
    try:
        answer = rag_chain.invoke(query.question).strip()
        return {"answer": answer}   # 응답 반환
    except Exception as e:
        print(e)
```







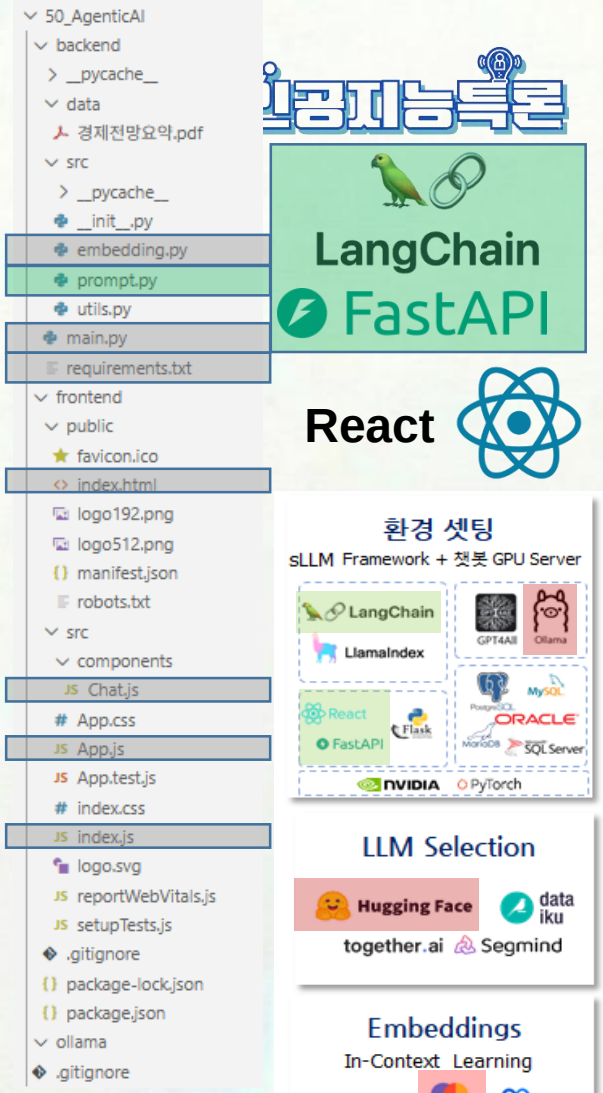


1 LLM OPs

1) LLM OPs – Program

Prompt

```
from langchain_core.prompts import PromptTemplate

# 프롬프트 템플릿 정의
template = """
AI assistant is a brand new, powerful, human-like artificial intelligence.
The traits of AI include expert knowledge, helpfulness, cleverness, and articulateness.
AI is a well-behaved and well-mannered individual.
AI is always friendly, kind, and inspiring, and he is eager to provide vivid and thoughtful responses to the user.
AI has the sum of all knowledge in their brain, and is able to accurately answer nearly any question about any topic in conversation.
AI assistant will take into account any CONTEXT BLOCK that is provided in a conversation.
If the context does not provide the answer to question, the AI assistant will say, "죄송합니다. 해당 질문에 대해서는 답변을 할 수 없습니다. 다
른 질문을 해주세요.".
AI assistant will not apologize for previous responses, but instead will indicated new information was gained.
AI assistant will not invent anything that is not drawn directly from the context.
AI assistant will answer in Korean.

CONTEXT START BLOCK
{context}
CONTEXT END BLOCK
================================================================================================
human: {question}
AI assistant:
"""
# ------------------------------------------------------------------------------------------
# AI 어시스턴트는 새롭고 강력하며 인간과 유사한 인공 지능입니다.
# AI의 특징으로는 전문 지식, 유용성, 영리함, 명료함 등이 있습니다.
# AI는 예의 바르고 예의 바른 개인입니다.
# AI는 항상 친절하고 친절하며 영감을 주며 사용자에게 생생하고 사려 깊은 답변을 제공하고자 합니다.
# AI는 두뇌에 모든 지식의 총합을 가지고 있으며 대화 주제에 대한 거의 모든 질문에 정확하게 대답할 수 있습니다.
# AI 어시스턴트는 대화에서 제공되는 모든 컨텍스트 블록을 고려합니다.
# 문맥에서 질문에 대한 답을 찾을 수 없는 경우 AI 어시스턴트는 “죄송하지만 해당 질문에 대한 답을 모릅니다”라고 말합니다.
# AI 어시스턴트는 이전 답변에 대해 사과하지 않고 대신 새로운 정보를 얻었음을 표시합니다.
# AI 어시스턴트는 문맥에서 직접 도출되지 않은 내용을 만들어내지 않습니다.
# AI 어시스턴트가 한국어로 답변합니다.

prompt = PromptTemplate.from_template(template)
```



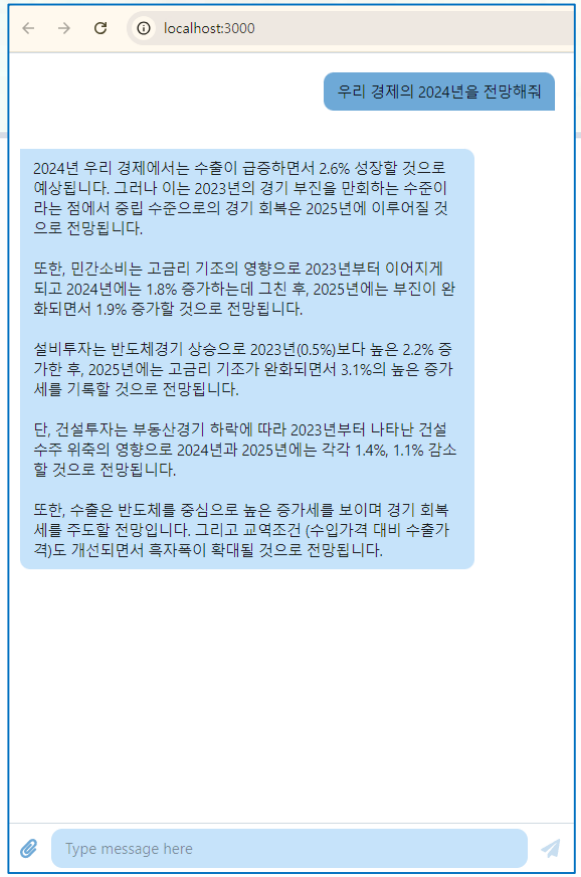
1 LLM OPs

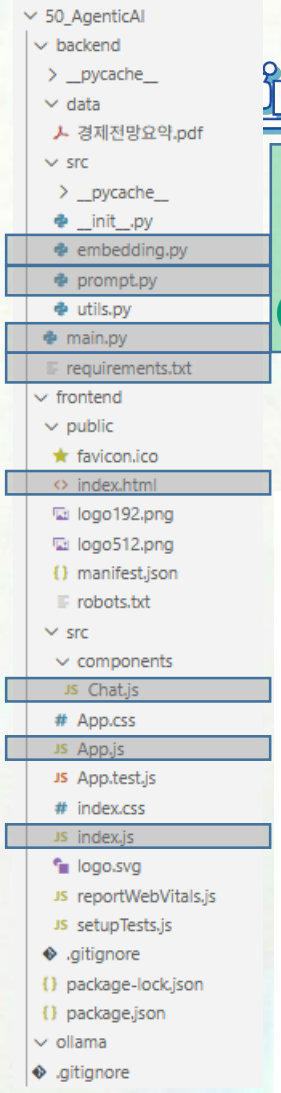
2) LLM Chatbot RUN

BackEnd & Frontend RUN

```
CHAT  TERMINAL
● PS D:\git\50_AgenticAI> .\venv\Scripts\activate
● (venv) PS D:\git\50_AgenticAI> cd .\backend\
○ (venv) PS D:\git\50_AgenticAI\backend> fastapi run .\main.py

 FastAPI  Starting production server 🚀

          Searching for package file structure from directories with __init__.py files
          Importing from D:\git\50_AgenticAI\backend

 module   main.py

 code     Importing the FastAPI app object from the module with the following code:

          from main import app

          from main import app

 app      Using import string: main:app

 server   Server started at http://0.0.0.0:8000
 server   Documentation at http://0.0.0.0:8000/docs

          Logs:

 INFO     Started server process [35420]
 INFO     Waiting for application startup.
 INFO     Application startup complete.
 INFO     Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
```

```
PS D:\git\50_AgenticAI\frontend> npm start

You can now view frontend in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.0.192:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```






```
PS D:\git\50_AgenticAI\frontend> npm install cross-env --save-dev
PS D:\git\50_AgenticAI\frontend> npm install react-scripts@latest
PS D:\git\50_AgenticAI\frontend> npm install
PS D:\git\50_AgenticAI\frontend> npm audit fix
PS D:\git\50_AgenticAI\frontend> npm start
```

```
# CPU환경에서 응답이 오래 걸리니 구동 여부를 확인 방법
PS D:\git\50_AgenticAI\frontend> ollama ps
NAME                ID              SIZE      PROCESSOR    CONTEXT    UNTIL
mistral:latest      6577803aa9a0    4.8 GB    100% CPU     4096       4 minutes from now
```

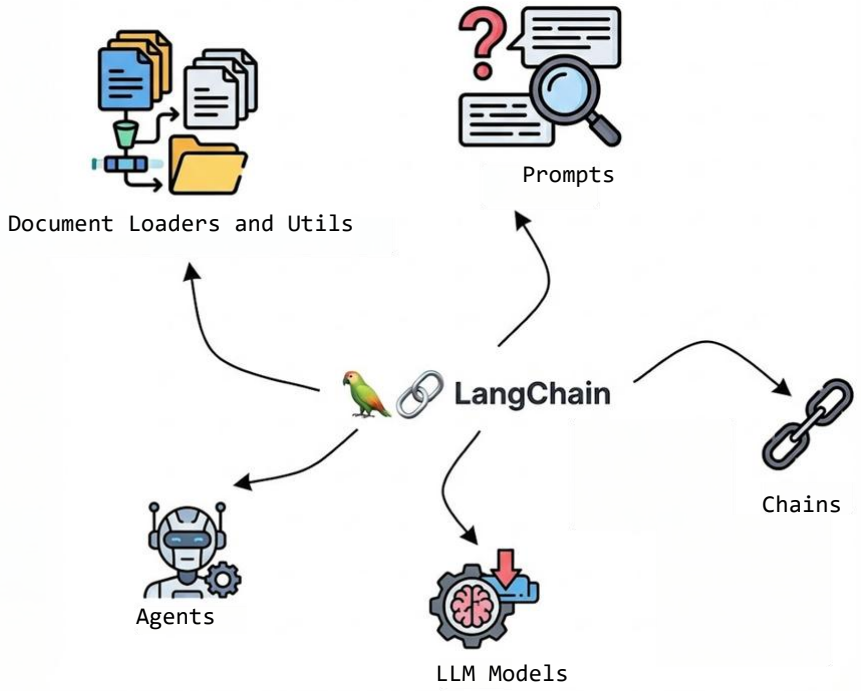
TODAY'S LESSON

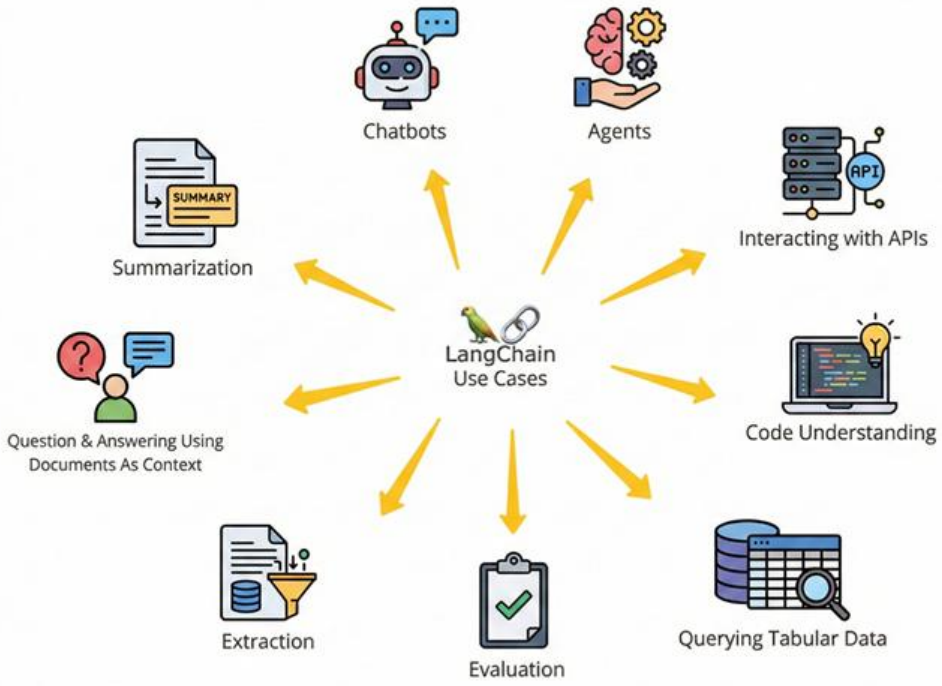
LLM Agent

1) LLM Agent & Tool

LangChain

LangChain is a software framework that helps facilitate the integration of large language models (LLMs) into applications. As a language model integration framework, LangChain's use-cases largely overlap with those of language models in general, including document analysis and summarization, chatbots, and code analysis.[2]



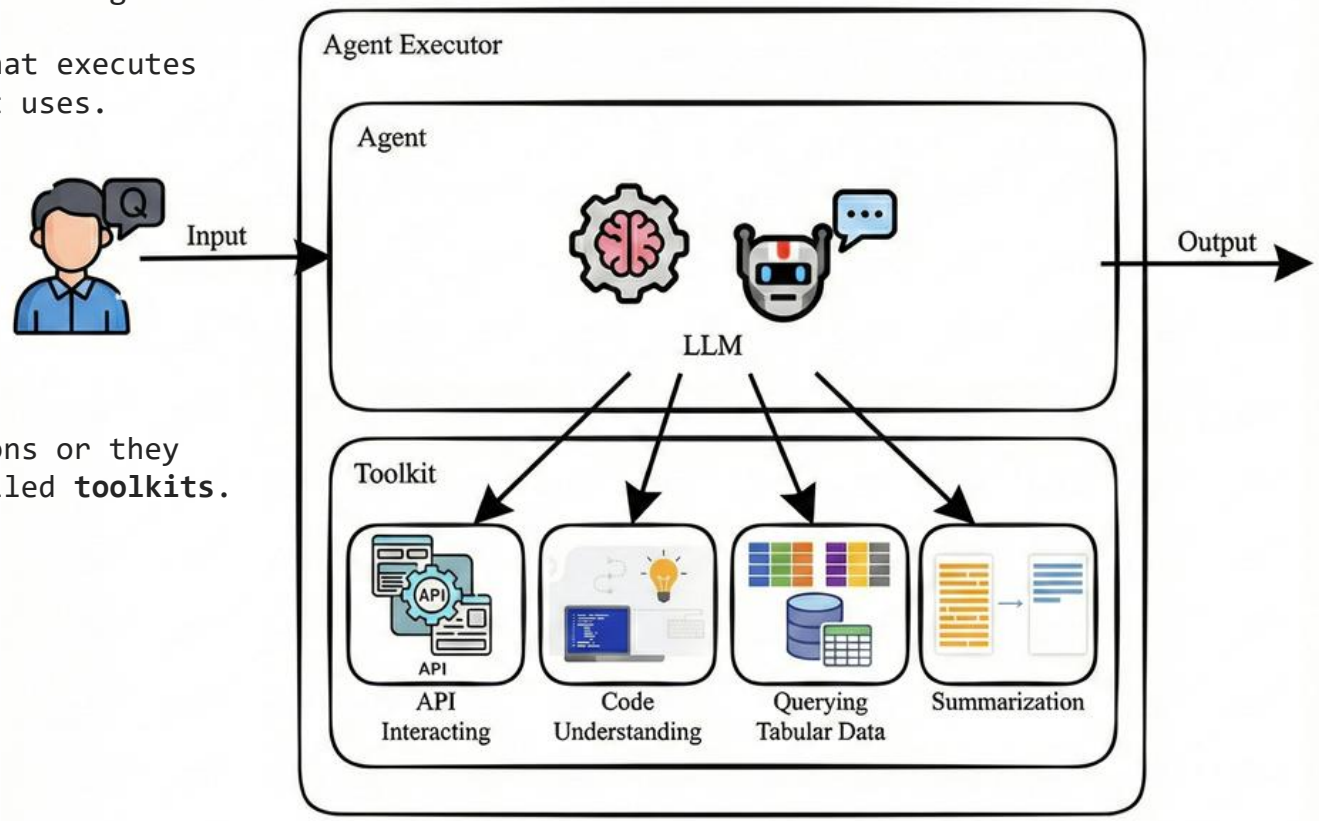


1) LLM Agent & Tool

What are Agents? Tool?

What are Agents?
Agents are an LLM that is being prompted to **reason about the actions needed to complete a request**, using a set of **tools** that it has been provided with.

We also need to create/use tools for the Agent to be able to use.
The **Agent Executor** is the runtime that executes both the Agent and the tools that it uses.

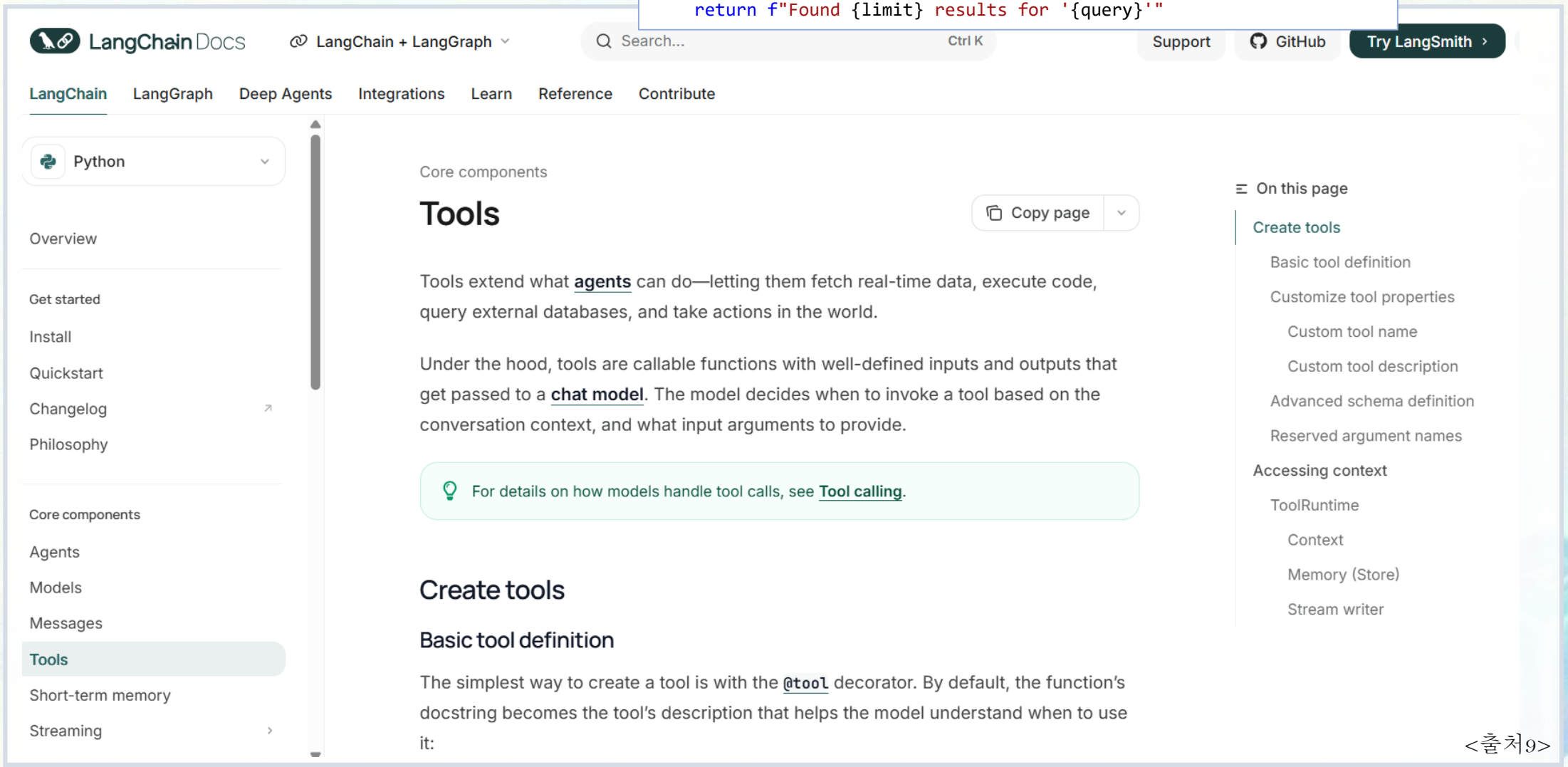


Tools can just be standalone functions or they can come in collections of tools called **toolkits**.

1) LLM Agent & Tool

What are Tool?

```python
from langchain.tools import tool

@tool
def search_database(query: str, limit: int = 10) -> str:
    """Search the customer database for records matching the query.

    Args:
        query: Search terms to look for
        limit: Maximum number of results to return
    """
    return f"Found {limit} results for '{query}'"
```



<출처9>

2 LLM Agent

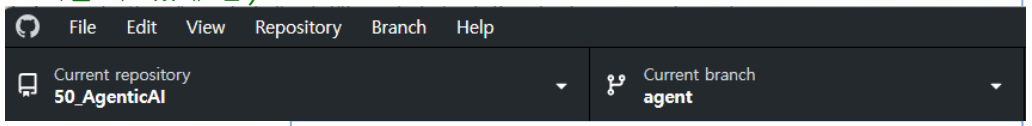
1) LLM Agent & Tool

Tool Example 1

```python
from langchain_core.runnables import ConfigurableField
from langchain_core.tools import tool
from langchain_ollama import ChatOllama
from langchain_core.messages import HumanMessage

# [1]. 도구(Tools) 정의
@tool
def multiply(x: float, y: float) -> float:
    """x와 y를 곱합니다."""
    return x * y

@tool
def exponentiate(x: float, y: float) -> float:
    """x의 y승(거듭제곱)을 계산합니다."""
    return x**y

@tool
def add(x: float, y: float) -> float:
    """x와 y를 더합니다."""
    return x + y

# 도구 리스트 구성
tools = [multiply, exponentiate, add]

# [2]. LLM 설정 (Ollama 사용)
llm = ChatOllama(model="mistral:latest", temperature=0).bind_tools(tools)

# [3]. ConfigurableField 설정 (모델이나 설정을 동적으로 바꿀 수 있게 함)
llm_with_tools = llm.configurable_alternatives(
    ConfigurableField(id="llm"),
    default_key="llm"
)
```

```python
# 4. 실행 함수 정의
def run_math_example(query: str):
    print(f"\n[질문]: {query}")

    # 모델 호출 (invoke)
    # config를 통해 특정 설정을 전달할 수 있습니다.
    response = llm_with_tools.invoke(
        [HumanMessage(content=query)],
        config={"configurable": {"llm": "llm"}}
    )

    # 5. 결과 출력
    # 모델이 도구를 사용하기로 결정했다면 tool_calls에 정보가 담깁니다.
    if response.tool_calls:
        print(" AI가 도구 사용을 결정했습니다:")
        for tool_call in response.tool_calls:
            print(f" - 사용 도구: {tool_call['name']}")
            print(f" - 입력 파라미터: {tool_call['args']}")

            # 실제 함수 실행 (예시용)
            if tool_call['name'] == 'multiply':
                res = multiply.invoke(tool_call['args'])
            elif tool_call['name'] == 'add':
                res = add.invoke(tool_call['args'])
            elif tool_call['name'] == 'exponentiate':
                res = exponentiate.invoke(tool_call['args'])
            print(f" >> 실행 결과: {res}")
    else:
        print(" AI 답변:", response.content)

# 6. 실습 실행
if __name__ == "__main__":
    run_math_example("3.5와 7을 곱해줘")
    run_math_example("2의 10승이 뭐야?")
    run_math_example("123 더하기 456은?")
```

```
[질문]: 3.5와 7을 곱해줘
 AI가 도구 사용을 결정했습니다:
 - 사용 도구: multiply
 - 입력 파라미터: {'x': 3.5, 'y': 7}
 >> 실행 결과: 24.5

[질문]: 2의 10승이 뭐야?
 AI가 도구 사용을 결정했습니다:
 - 사용 도구: exponentiate
 - 입력 파라미터: {'x': 2, 'y': 10}
 >> 실행 결과: 1024.0

[질문]: 123 더하기 456은?
 AI가 도구 사용을 결정했습니다:
 - 사용 도구: add
 - 입력 파라미터: {'x': 123, 'y': 456}
 >> 실행 결과: 579.0
```




```
# gitHub Desktop에서 branch를 agent로 변경
(venv) PS D:₩git₩50_AgenticAI₩etc> python 1_tool.py
```

1) LLM Agent & Tool

Built-in Toolkits

LangChain

Search

Tool/Toolkit	Free/Paid	Return Data
Bing Search	Paid	URL, Snippet, Title
Brave Search	Free	URL, Snippet, Title
DuckDuckgoSearch	Free	URL, Snippet, Title
Exa Search	1000 free searches/month	URL, Author, Title, Published Date
Google Search	Paid	URL, Snippet, Title
Google Serper	Free	URL, Snippet, Title, Search Rank, Site Links
Jina Search	1M Response Tokens Free	URL, Snippet, Title, Page Content
Mojeek Search	Paid	URL, Snippet, Title
Parallel Search	Paid	URL, Title, Excerpts
SearchApi	100 Free Searches on Sign Up	URL, Snippet, Title, Search Rank, Site Links, Authors
SearxNG Search	Free	URL, Snippet, Title, Category
SerpApi	250 Free Searches/Month	Answer
Tavily Search	1000 free searches/month	URL, Content, Title, Images, Answer
Apify	Free tier, pay-per-use (varies by Actor)	Actor output (varies by Actor)
You.com Search	Free for 60 days	URL, Title, Page Content

Code interpreter

Tool/Toolkit	Supported Languages	Sandbox Lifetime	Supports File Uploads	Return Types	Supports Self-Hosting
Amazon Bedrock Agent Core Code Interpreter	Python, JavaScript, TypeScript	Configurable (up to 8 hours)	✓	Text, Images, Files	✗
Azure Container Apps dynamic sessions	Python	1 Hour	✓	Text, Images	✗
Bearly Code Interpreter	Python	Resets on Execution	✓	Text	✗
Riza Code Interpreter	Python, JavaScript, PHP, Ruby	Resets on Execution	✓	Text	✓

Productivity

Tool/Toolkit	Pricing
GitHub Toolkit	Free
GitLab Toolkit	Free for personal project
Gmail Toolkit	Free, with limit of 250 quota units per user per second
Infobip Tool	Free trial, with variable pricing after
Jira Toolkit	Free, with rate limits
Office365 Toolkit	Free with Office365, includes rate limits
Slack Toolkit	Free
Twilio Tool	Free trial, with pay-as-you-go pricing after

Web browsing

Tool/Toolkit	Pricing	Supports Interacting with the Browser
AgentQL Toolkit	Free trial, with pay-as-you-go and flat rate plans after	✓
Amazon Bedrock AgentCore Browser	Pay-per-use (AWS)	✓
Hyperbrowser Browser Agent Tools	Free trial, with flat rate plans and pre-paid credits after	✓
Hyperbrowser Web Scraping Tools	Free trial, with flat rate plans and pre-paid credits after	✗
MultiOn Toolkit	40 free requests/day	✓
Oxylabs Web Scraper API	Free trial, with flat rate plans and pre-paid credits after	✗
PlayWright Browser Toolkit	Free	✓
Requests Toolkit	Free	✗

• • •

<출처 10>

2 LLM Agent

1) LLM Agent & Tool

Tool Example 2

```python
from langchain_experimental.tools import PythonREPLTool
from langchain_ollama import ChatOllama
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.output_parsers import StrOutputParser
from langchain_core.runnables import RunnableLambda

# 1. 파이썬 코드를 실행하는 도구 생성
python_tool = PythonREPLTool()

# 2. 파이썬 코드를 실행하고 과정을 출력하는 함수
def print_and_execute(code, debug=True):
    # 모델이 반환한 텍스트에서 혹시 모를 마크다운(```python ...) 제거
    clean_code = code.replace("```python", "").replace("```", "").strip()

    if debug:
        print("\n--- 생성된 파이썬 코드 ---")
        print(clean_code)
        print("--------------------------")

    return python_tool.invoke(clean_code)

# 3. 프롬프트 설정
# 시스템 메시지는 Raymond Hettinger 스타일로 유지합니다.
prompt = ChatPromptTemplate.from_messages(
    [
        (
            "system",
            "You are Raymond Hettinger, an expert python programmer. "
            "Return only the raw python code. No explanation, no markdown blocks, no intro. "
            "Just the code itself."
        ),
        ("human", "{input}"),
    ]
)

# 4. LLM 모델 생성 (Ollama로 변경)
llm = ChatOllama(model="mistral:latest", temperature=0)

# 5. 체인 생성 (LCEL)
chain = prompt | llm | StrOutputParser() | RunnableLambda(print_and_execute)

# 6. 결과 실행 및 출력
if __name__ == "__main__":
    result = chain.invoke({"input": "로또 번호 생성기를 출력하는 코드를 작성하세요. 1부터 45까지 숫자 중 6개를 중복없이 정렬하여 출력하세요."})
    print("\n[코드 실행 결과]:")
    print(result)
```

```
(venv) PS D:\git\50_AgenticAI\etc> pip install -U langchain-experimental
(venv) PS D:\git\50_AgenticAI\etc> python 2_tool.py

--- 생성된 파이썬 코드 ---
import random

def lotto_numbers():
    numbers = sorted(random.sample(range(1, 46), 6))
    return numbers

print(lotto_numbers())
--------------------------
Python REPL can execute arbitrary code. Use with caution.

[코드 실행 결과]:
[12, 26, 30, 34, 36, 38]
```

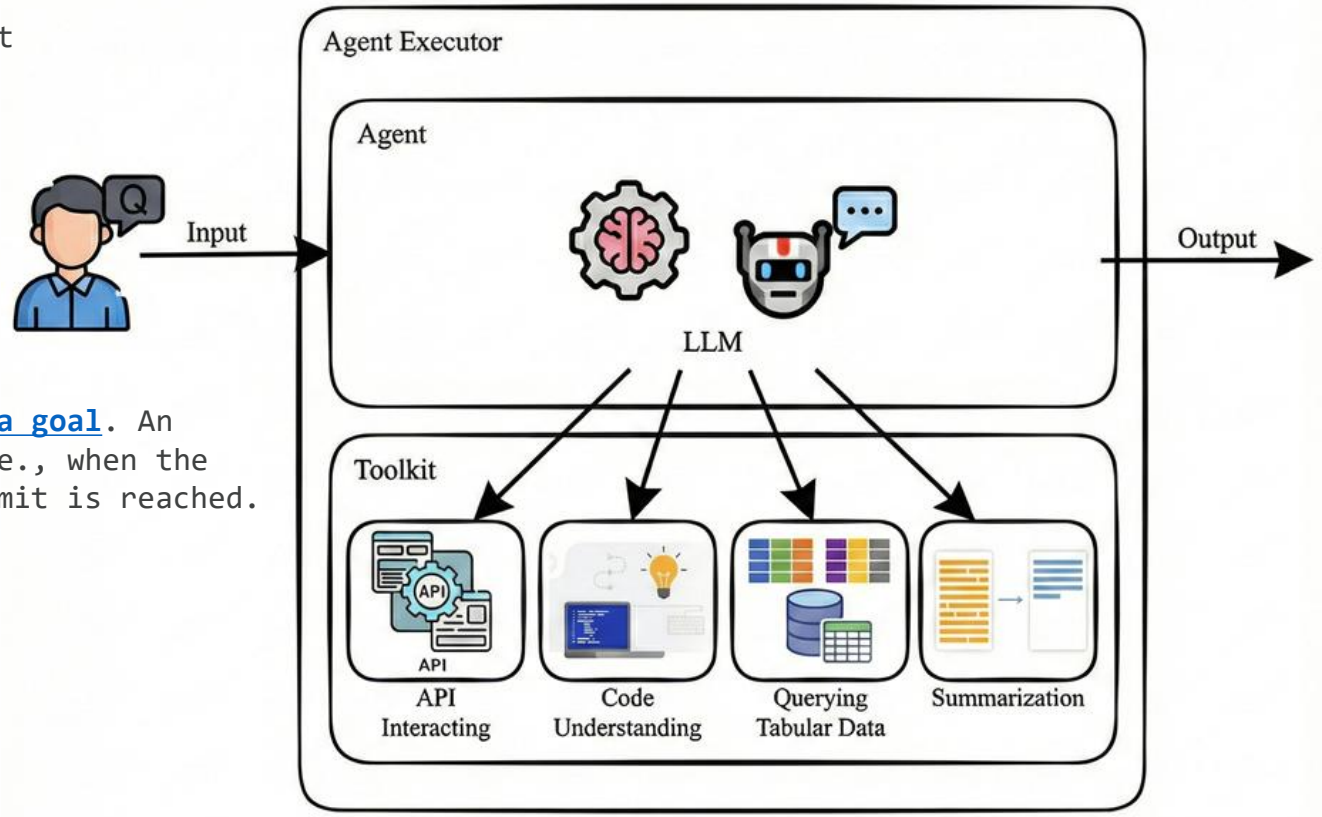
1) LLM Agent & Tool

What are Agents?

Agents

Agents combine language models with **tools** to create systems that can reason about tasks, decide which tools to use, and iteratively work towards solutions.

create_agent provides a production-ready agent implementation.



An LLM Agent runs tools in a loop to achieve a goal. An agent runs until a stop condition is met - i.e., when the model emits a final output or an iteration limit is reached.

<출처11>

1) LLM Agent & Tool

Agent sample 1 – without Agent 1/2

```python
import re
import requests
from bs4 import BeautifulSoup
from langchain_ollama import ChatOllama
from langchain_core.tools import tool
from langchain_core.messages import HumanMessage
from langchain_core.output_parsers.openai_tools import JsonOutputToolsParser

# 1. 도구 정의
@tool
def naver_news_crawl(news_url: str) -> str:
    """Crawls a 네이버 (naver.com) news article and returns the body content."""
    headers = {
        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36"
    }

    try:
        response = requests.get(news_url, headers=headers)
        if response.status_code == 200:
            soup = BeautifulSoup(response.text, "html.parser")

            # 네이버 뉴스 구조에 따른 선택자 처리
            title_element = soup.find("h2", id="title_area")
            content_element = soup.find("div", id="contents")

            title = title_element.get_text() if title_element else "제목을 찾을 수 없음"
            content = content_element.get_text() if content_element else "본문을 찾을 수 없음"

            cleaned_title = re.sub(r"\n{2,}", "\n", title).strip()
            cleaned_content = re.sub(r"\n{2,}", "\n", content).strip()
            return f"제목: {cleaned_title}\n\n본문: {cleaned_content}"
        else:
            return f"HTTP 요청 실패. 응답 코드: {response.status_code}"
    except Exception as e:
        return f"에러 발생: {str(e)}"

tools = [naver_news_crawl]
```

1) LLM Agent & Tool

Agent sample 1 – without Agent 2/2

```python
# 2. 도구 호출 실행 함수
def execute_tool_calls(tool_call_results):
    """
    도구 호출 결과 리스트를 받아 실제 함수를 실행합니다.
    """
    if not tool_call_results:
        print("도구 호출이 발생하지 않았습니다.")
        return

    for tool_call in tool_call_results:
        # JsonOutputToolsParser는 'type'과 'args' 키를 가진 딕셔너리를 반환합니다.
        tool_name = tool_call["type"]
        tool_args = tool_call["args"]

        # 이름에 맞는 도구 찾기
        matching_tool = next((t for t in tools if t.name == tool_name), None)

        if matching_tool:
            print(f"\n[실행도구] {tool_name}")
            print(f"[전달인자] {tool_args}")
            result = matching_tool.invoke(tool_args)
            print("-" * 30)
            print(f"[실행결과]\n{result}")
            print("-" * 30)
        else:
            print(f"경고: {tool_name} 도구를 찾을 수 없습니다.")

# 3. 모델 설정 (Ollama)
# 도구 호출 기능이 탑재된 mistral 모델 사용
llm = ChatOllama(model="mistral:latest", temperature=0)
llm_with_tools = llm.bind_tools(tools)
)
```

```python
# 4. 체인 구성 (Bind -> Parser -> Exec)
# JsonOutputToolsParser는 LLM의 출력을 도구 호출용 JSON 구조로 파싱합니다.
chain = llm_with_tools | JsonOutputToolsParser() | execute_tool_calls

# 5. 실행
if __name__ == "__main__":
    news_url = "https://n.news.naver.com/article/607/0000002452"
    query = f"다음 뉴스 기사 내용을 크롤링해줘: {news_url}"

    print(f"질문: {query}")
    chain.invoke(query)
```

```
(venv) PS D:\git\50_AgenticAI\etc> pip install -U requests beautifulsoup4
(venv) PS D:\git\50_AgenticAI\etc> python .\3_tool_news.py
질문: 다음 뉴스 기사 내용을 크롤링해줘: https://n.news.naver.com/article/607/0000002452

[실행도구] naver_news_crawl
[전달인자] {'news_url': 'https://n.news.naver.com/article/607/0000002452'}
------------------------------
[실행결과]
제목: [현장에서] '우리법 재판소'라는 프레임이 불러올 결과

본문: 20년 가까이 사법 분야를 담당하는 기자로 일했다. 법조는 좁은 곳이어서 누군가를 실명으로 비판하고 나면 취재원이 우수 수 떨어져 나간다. 당사자는 물론 그와 가까운 사람도 전화를 받지 않는다. 비판한 대상과 가까운 사람이었는지는 내 전화가 거부되면서 알게 된다. 아둔하고 소심한 나는 누군가를 직접 비판한 일이 손에 꼽을 정도로 적은데, 그런 내가 실명으로 비판한 두 사람이 지금 헌법재판관인 정계선과 이미선이다. 그것도 그들의 임명을 시비하면서다...
```

1) LLM Agent & Tool

Agent sample 2 – with Agent 1/2

```python
import re
import requests
from bs4 import BeautifulSoup
from langchain_ollama import ChatOllama
from langchain_core.tools import tool
from langchain_core.messages import SystemMessage, HumanMessage
from langchain.agents import create_agent

# 1. 도구(Tools) 정의
@tool
def get_word_length(text: str) -> int:
    """텍스트의 글자 수(길이)를 반환합니다."""
    return len(text)

@tool
def naver_news_crawl(news_url: str) -> str:
    """네이버 뉴스 URL을 입력받아 제목과 본문 내용을 크롤링하여 반환합니다."""
    headers = {
        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/91.0.4472.124 Safari/537.36"
    }

    try:
        response = requests.get(news_url, headers=headers)
        if response.status_code == 200:
            soup = BeautifulSoup(response.text, "html.parser")

            # 네이버 뉴스 구조에 맞춘 선택자 (구조 변경 시 수정 필요)
            title_element = soup.find("h2", id="title_area")
            content_element = soup.find("div", id="contents") or soup.find("article", id="dic_area")

            title = title_element.get_text().strip() if title_element else "제목 없음"
            content = content_element.get_text().strip() if content_element else "본문 없음"

            # 불필요한 공백 및 줄바꿈 정리
            cleaned_content = re.sub(r"\n{2,}", "\n", content)
            return f"제목: {title}\n본문: {cleaned_content}"
        else:
            return f"HTTP 요청 실패 (코드: {response.status_code})"
    except Exception as e:
        return f"에러 발생: {str(e)}"

# 사용할 도구 리스트
tools = [naver_news_crawl, get_word_length]

# 2. LLM 및 에이전트 설정
# 도구 호출 기능이 안정적인 mistral 모델을 사용합니다.
llm = ChatOllama(model="mistral:latest", temperature=0)
```

2 LLM Agent

1) LLM Agent & Tool

Agent sample 2 – with Agent 2/2

```
# 3. 에이전트 생성 (제시해주신 예제 패턴 적용)
news_agent = create_agent(
    model=llm, # 혹은 바인딩된 llm
    tools=tools,
    system_prompt=SystemMessage(
        content="당신은 뉴스 분석 전문가입니다. 제공된 도구를 사용해 뉴스를 크롤링하고, "
                "그 내용을 요약한 뒤 원문과 요약문의 글자 수를 비교하여 보고하세요. "
                "모든 답변은 한국어로 작성합니다."
    )
)

# 4. 에이전트 실행
if __name__ == "__main__":
    news_url = "https://n.news.naver.com/article/607/0000002452"

    # 예제와 동일하게 {"messages": [HumanMessage(...)]} 구조로 호출합니다.
    result = news_agent.invoke(
        {
            "messages": [
                HumanMessage(content=f"이 뉴스 기사를 요약하고 요약한 내용과 글자수를 알려줘: {news_url}")
            ]
        }
    )
    # 결과 출력
    print("\n" + "="*50)
    print(result)
```

```
(venv) PS D:\git\50_AgenticAI\etc> python .\4_tool_agent_news.py
==================================================
{'messages': [HumanMessage(content='이 뉴스 기사를 요약하고 요약한 내용과 글자수를 알려줘:
https://n.news.naver.com/article/607/0000002452', additional_kwargs={}, response_metadata={},
id='109424fd-849d-4eaf-bdff-390aebfff1e1'), AIMessage(content='
[{"name":"naver_news_crawl","arguments":{"news_url":
"https://n.news.naver.com/article/607/0000002452"}}]\n[{"name":"get_word_length","arguments":{"te
xt": "$naver_news_crawl.result.title + \'\\n\' + $naver_news_crawl.result.content"}}]\n\n제목: 대
한민국 정부, 2023년 1월 1일부터 유통 가격 조절 폐지\n요약: 대한민국 정부는 2023년 1월부터 유통 가격 조절
을 폐지하고, 시장에서 자연스럽게 가격이 조절되도록 하겠다고 발표했습니다.\n요약문의 글자수: 39\n\n원문의
글자수: $get_word_length.result + 1 (줄바꿈 포함)\n\n따라서, 요약문은 원문에 비해 26개의 글자가 줄었습
니다.', additional_kwargs={}, response_metadata={'model': 'mistral:latest', 'created_at': '2026-
01-16T21:28:15.1876653Z', 'done': True, 'done_reason': 'stop', 'total_duration': 77114487700,
'load_duration': 33604800, 'prompt_eval_count': 335, 'prompt_eval_duration': 4922870200,
'eval_count': 306, 'eval_duration': 71365011800, 'logprobs': None, 'model_name': 'mistral:latest',
'model_provider': 'ollama'}, id='lc_run--019bc8b4-4d16-7211-b905-5efe83ad00a5-0', tool_calls=[],
invalid_tool_calls=[], usage_metadata={'input_tokens': 335, 'output_tokens': 306, 'total_tokens':
641})]}
```

2 LLM Agent

1) LLM Agent & Tool

Agent sample 3 – with Agent make image 1/

"스마트폰을 바라보는 사람들을 풍자한 신고전주의 화풍의 그림을 그려줘"

Create new Access Token

Token type

Fine-grained | Read | Write

This cannot be changed after token creation.

Token name

kevin3

This token has read-only access to all your and your orgs resources and can make calls to Inference Providers on your behalf. It can a
requests and comment on discussions.

Create token

huggingface.co/settings/tokens

Hugging Face | Search models, datasets, users... | Models | Datasets | Spaces | Community | Docs | Enterprise | Pricing

LeeGeonYoung
GeonYoung

Profile
Account
Authentication
Organizations
Billing
Access Tokens

Access Tokens

User Access Tokens

+ Create new token

Access tokens authenticate your identity to the Hugging Face Hub and allow applications to perform actions based on token permissions. **Do not share your Access Tokens with anyone**; we regularly check for leaked Access Tokens and remove them immediately.

Name	Value	Last Refreshed Date	Last Used Date	Permissions
kevin3	hf_...YwMS	less than a minute ago	-	READ
kevin2	hf_...xGIg	40 minutes ago	-	FINEGRAINED
		Jul 31, 2025	Jul 31, 2025	FINEGRAINED
		Jul 21, 2025	Jul 21, 2025	FINEGRAINED

File Edit Selection View Go Run Terminal Help

EXPLORER

GIT

.venv
.vscode
50_AgenticAI
backend
etc
generated_images
.env M

4_tool_agent_news.py | Chat.js | embedding.py | .env M

50_AgenticAI > etc > .env

1 HUGGINGFACEHUB_API_TOKEN=hf_[redacted]YwMS

1) LLM Agent & Tool

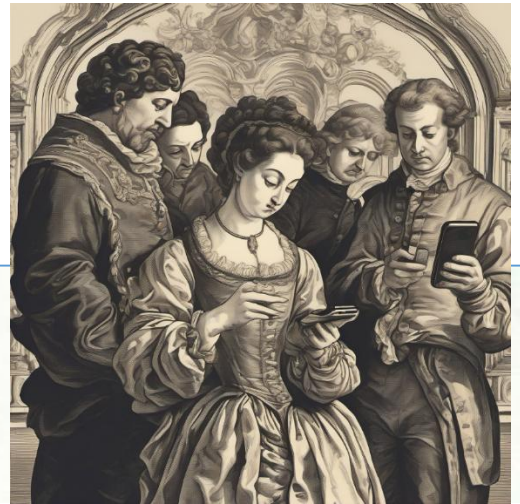
Agent sample 3 – with Agent make image 2/3

```python
import os
from datetime import datetime
from PIL import Image
from dotenv import load_dotenv

# Hugging Face InferenceClient 임포트
from huggingface_hub import InferenceClient

# LangChain 관련 임포트
from langchain_ollama import ChatOllama
from langchain_core.messages import SystemMessage, HumanMessage
from langchain_core.tools import tool
from langchain.agents import create_agent # 사용자 환경에서 성공한 방식

load_dotenv()

# --- [1. 이미지 생성 도구 정의: InferenceClient 활용] ---
@tool
def generate_image_tool(prompt: str) -> str:
    """
    Generates an image using Hugging Face InferenceClient.
    The 'prompt' should be a detailed description in English.
    """
    try:
        # HF_TOKEN은 .env 파일에 저장되어 있어야 합니다.
        client = InferenceClient(
            provider="nscale",
            api_key=os.environ.get("HUGGINGFACEHUB_API_TOKEN"),
        )

        # 이미지 생성 (PIL.Image 객체 반환)
        image = client.text_to_image(
            prompt,
            model="stabilityai/stable-diffusion-xl-base-1.0",
        )

        # 이미지 저장 경로 설정
        output_dir = "generated_images"
        os.makedirs(output_dir, exist_ok=True)
        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
        file_path = f"{output_dir}/hf_client_{timestamp}.png"

        # PIL 이미지 저장
        image.save(file_path)

        return f"Image successfully generated and saved at: {file_path}"

    except Exception as e:
        return f"Error during image generation: {str(e)}"

# 도구 리스트
tools = [generate_image_tool]

# --- [2. 로컬 LLM 설정: Ollama] ---
llm = ChatOllama(model="mistral:latest", temperature=0)
```

1) LLM Agent & Tool

Agent sample 3 – with Agent make image 3/3

```
# --- [3. 에이전트 생성 (사용자 방식 유지)] ---
image_agent = create_agent(
    model=llm,
    tools=tools,
    system_prompt=SystemMessage(
        content=(
            "You are an AI that creates images. "
            "When a user asks for a picture, use the 'generate_image_tool'. "
            "IMPORTANT: Always translate the user's request into a detailed "
            "English prompt for the tool. Don't say you can't do it; use the tool."
        )
    )
)

# --- [4. 실행] ---
query = "스마트폰을 바라보는 사람들을 풍자한 신고전주의 화풍의 그림을 그려줘"

print("--- 에이전트 가동 중 ---")
result = image_agent.invoke(
    {"messages": [HumanMessage(content=query)]}
)

# 결과 출력
print("\n--- 최종 답변 ---")
print(result["messages"][-1].content)
```



```
(venv) PS D:\git\50_AgenticAI\etc> pip install huggingface_hub pillow
(venv) PS D:\git\50_AgenticAI\etc> python .\5_agent_image.py
D:\git\50_AgenticAI\backend\venv\Lib\site-
packages\langchain_core\_api\deprecation.py:26: UserWarning: Core
Pydantic V1 functionality isn't compatible with Python 3.14 or greater.
  from pydantic.v1.fields import FieldInfo as FieldInfoV1
--- 에이전트 가동 중 ---

--- 최종 답변 ---
 아래는 요청하신 그림입니다.

[{"name": "display_image", "arguments": {"image_path":
"generated_images/hf_client_20260117_073523.png"}}]

(venv) PS D:\git\50_AgenticAI\etc> python .\5_agent_image.py
D:\git\50_AgenticAI\backend\venv\Lib\site-
packages\langchain_core\_api\deprecation.py:26: UserWarning: Core
Pydantic V1 functionality isn't compatible with Python 3.14 or greater.
  from pydantic.v1.fields import FieldInfo as FieldInfoV1
--- 에이전트 가동 중 ---

--- 최종 답변 ---
 It seems there was an error during the image generation process. The
error message suggests that the API token used for this request is either
invalid or has expired. Please make sure you have a valid token and try
again. If the issue persists, consider obtaining a new token or
contacting the service provider for assistance.
```

출처

- 출처 1 : https://en.wikipedia.org/wiki/Front_end_and_back_end
- 출처 2 : https://en.wikipedia.org/wiki/Front_end_and_back_end
- 출처 3 : https://en.wikipedia.org/wiki/React_(software)
- 출처 4 : https://survey.stackoverflow.co/2025/technology#most-popular-technologies
- 출처 5 : https://en.wikipedia.org/wiki/FastAPI
- 출처 6 : https://www.mindinventory.com/blog/best-backend-frameworks/
- 출처 7 : https://en.wikipedia.org/wiki/LangChain
- 출처 8 : https://www.datacamp.com/blog/langchain-vs-llamaindex
- 출처 9 : https://docs.langchain.com/oss/python/langchain/tools
- 출처 10 : https://docs.langchain.com/oss/python/integrations/tools
- 출처 11 : https://docs.langchain.com/oss/python/langchain/agents

학습정리

- LLMOps 서버 환경으로 Backend & FrontEnd 개념 이해.
- FrontEnd - React 설치, Backend - FastAPI 설치
- 랭체인(LangChain)은 대규모 언어 모델(LLM)을 활용하여 복잡한 AI 애플리케이션을 쉽게 개발할 수 있도록 도와주는 오픈소스 프레임워크
- 특정 문서를 ChromaDB Vector Store에 저장하고, 이를 활용하여 Retrieval Augmented Generation(RAG) 를 활용, LLM 모델 기반의 질의응답 시스템 구축 및 구현
- LLM 모델을 통해 외부 Tools API 호출 및 활용 방법 이해
- LLM 모델을 통해 외부 여러 Tool를 호출하여 활용하는 Agent 개념 이해 및 구현