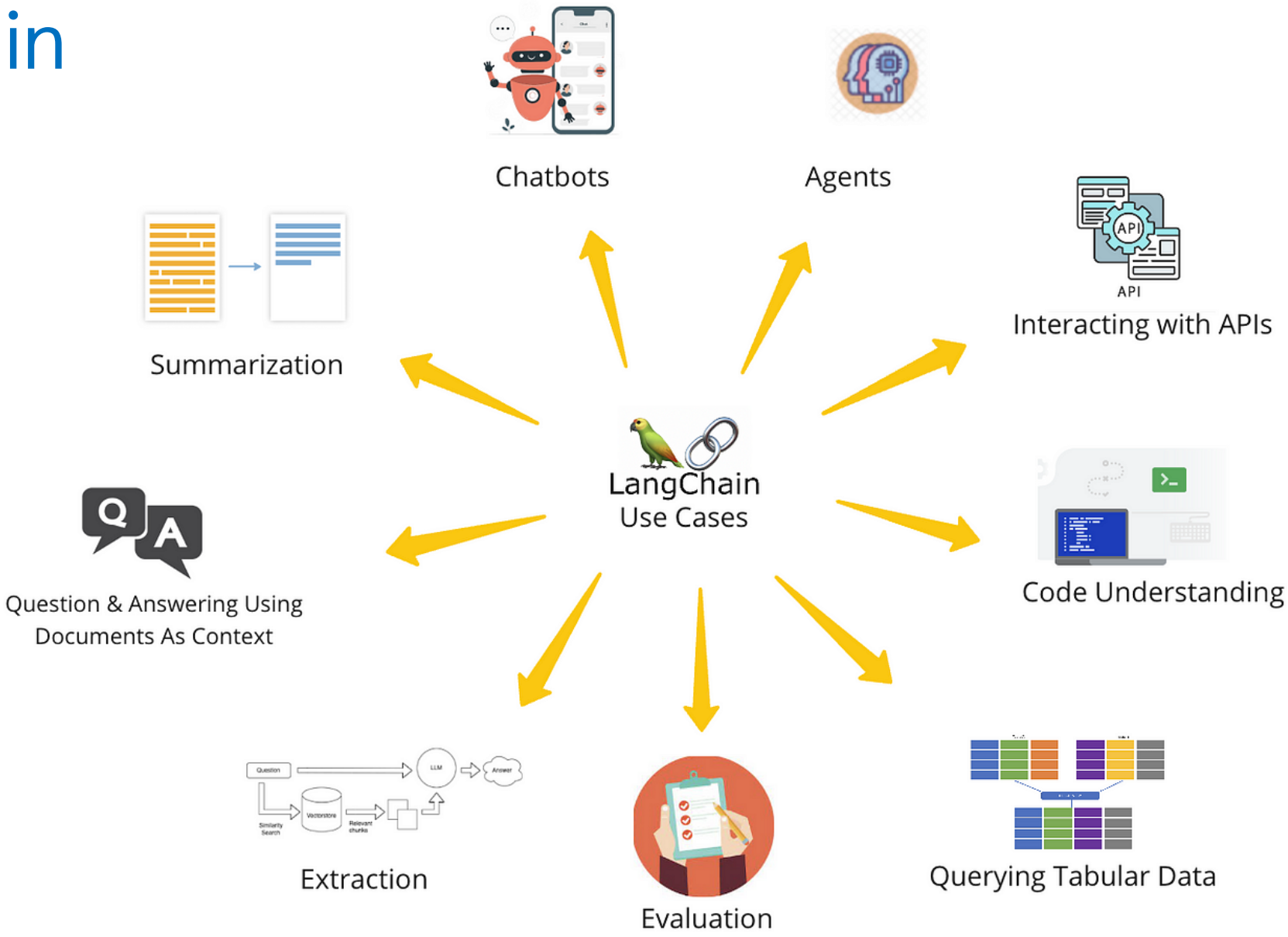
5. 퀵스타트

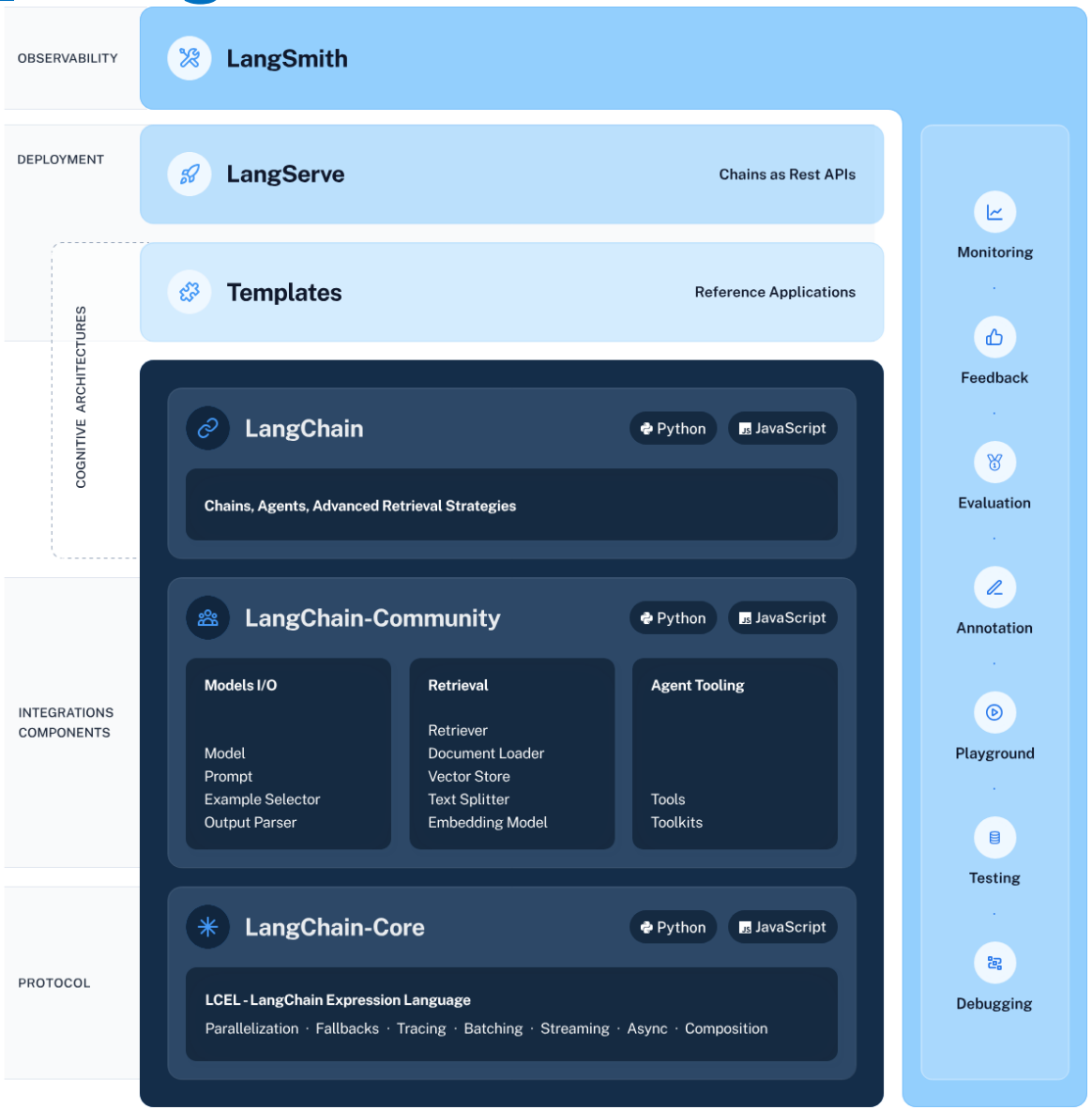
LangChain

박경규

LangChain


Chatbots
Agents
Summarization
API
Interacting with APIs
LangChain
Use Cases
Question & Answering Using
Documents As Context
Code Understanding
Question
LLM
Answer
Vectorstore
Similarity
Search
Relevant
chunks
Extraction
Evaluation
Querying Tabular Data

LangChain 프레임워크 구성



LangChain은 언어모델로 구동되는 애플리케이션을 개발하기 위한 프레임워크로 다음과 같은 애플리케이션을 지원

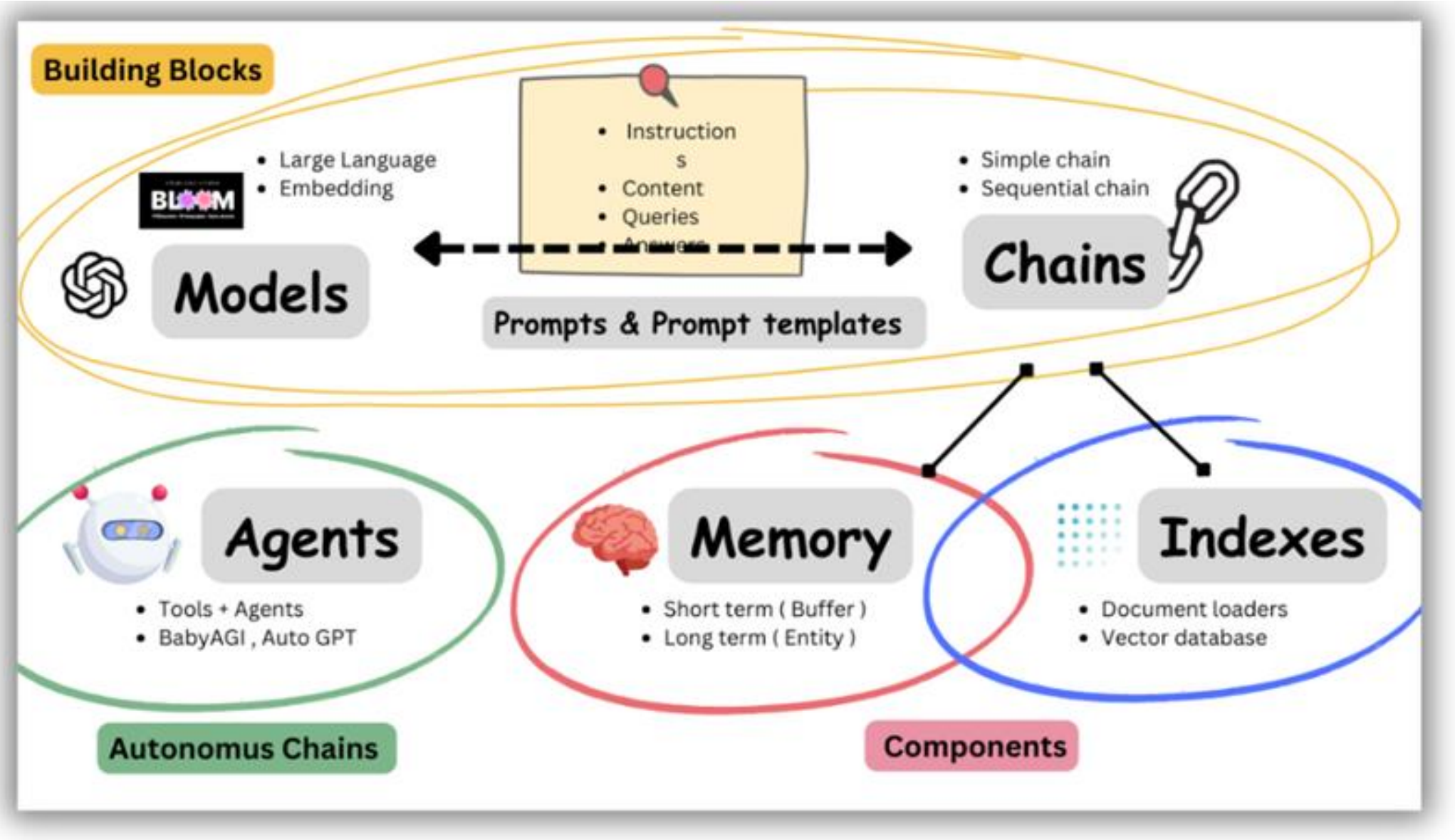
- **문맥 인식**(Are context-aware): 언어 모델을 프롬프트 지시, 퓨샷 예제, 응답의 근거가 되는 콘텐츠 등의 컨텍스트 소스에 연결
- **추론**(Reason): 언어 모델에 의존하여 제공된 컨텍스트에 따라 답변하는 방법, 취해야 할 조치 등을 추론

LangChain 프레임워크구성

- **LangChain Libraries:** Python 및 JavaScript 라이브러리.
  수많은 컴포넌트에 대한 인터페이스와 통합,
  이러한 컴포넌트를 체인 및 에이전트로 결합하는 기본 런타임,
  체인 및 에이전트의 즉시 사용 가능한 구현을 포함
- **LangChain Templates:** 다양한 작업을 위해 쉽게 배포할 수 있는 참조 아키텍처 모음
- **LangServe:** LangChain 체인을 REST API로 배포하기 위한 라이브러리
- **LangSmith:** 모든 LLM 프레임워크에 구축된 체인을 디버그, 테스트, 평가, 모니터링할 수 있는 개발자 플랫폼으로, LangChain과 원활하게 통합

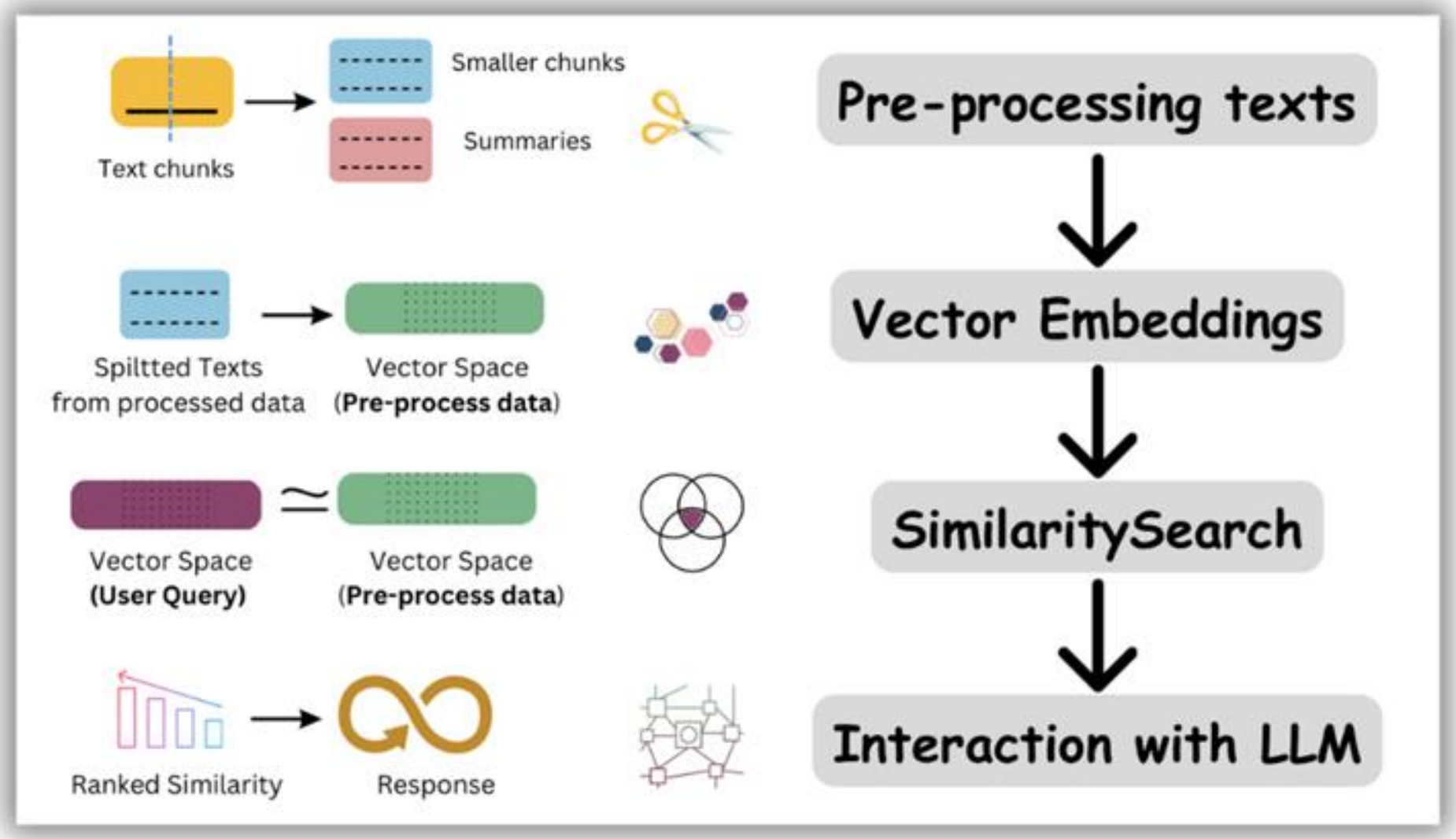
https://python.langchain.com/docs/get_started/introduction

LangChain 주요 빌딩 블록



https://www.packtpub.com/article-hub/using-langchain-for-large-language-model-powered-applications

LangChain 주요 워크플로



https://www.packtpub.com/article-hub/using-langchain-for-large-language-model-powered-applications

LangChain 퀵스타트 LangChain_QuickStart.ipynb

LangChain 설치

[1]:
```
%pip install langchain langchain-community langchain-openai
```

1. LLM Chain

[2]:
```
import os

os.environ["OPENAI_API_KEY"] = "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"
```

[3]:
```
from langchain_openai import ChatOpenAI

llm = ChatOpenAI()
```

[4]:
```
llm.invoke("langsmith가 어떻게 테스트에 도움을 줄 수 있나요?")
```

[4]:
```
AIMessage(content='Langsmith는 다음과 같은 방법으로 테스트에 도움을 줄 수 있습니다:\n\n1. 자동화된 테스트: Langsmith는 테스트를 자동화하여 효율적인 테스트를 수행할 수 있습니다. 이는 개발자가 코드 변경 사항을 반복적으로 테스트하는 데 도움이 됩니다. 자동화된 테스트를 통해 코드의 기능적인 측면과 예상되는 결과를 확인할 수 있습니다.\n\n2. 다국어 테스트: Langsmith는 다양한 언어에 대한 테스트를 지원할 수 있습니다. 이는 다
```

LangChain 퀵스타트

Prompt template : 사용자 입력을 LLM에 더 나은 입력으로 변환하는 데 사용

[5]:
```python
from langchain_core.prompts import ChatPromptTemplate

prompt = ChatPromptTemplate.from_messages([
    ("system", "당신은 세계적인 수준의 기술 문서 작성자입니다."),
    ("user", "{input}")
])
```

llm과 prompt를 간단한 LLM 체인으로 결합

[6]:
```python
chain = prompt | llm
```

[7]:
```python
chain.invoke({"input": "langsmith가 어떻게 테스트에 도움을 줄 수 있나요?"})
```

[7]:
```
AIMessage(content='langsmith는 테스트에 다양한 방법으로 도움을 줄 수 있습니다. \n\n첫째로, langsmith는 테스트 케이스를 작성하는 데 도움을 줄 수 있습니다. 테스트 케이스는 특정 기능이나 시스템의 동작을 확인하기 위해 작성되는 입력, 예상 결과 및 실행 조건의 조합입니다. langsmith는 테스트 케이스를 자동으로 생성하거나 테스트 케이스 작성을 돕는 도구를 제공할 수 있습니다.\n\n둘째로, langsmith는 테스트 실행 및 결과 분석에 도
```

LangChain 퀵스타트

채팅 Message를 문자열로 변환하는 간단한 OutputParser를 추가

[8]:
```
from langchain_core.output_parsers import StrOutputParser

output_parser = StrOutputParser()
```

[9]:
```
chain = prompt | llm | output_parser
chain.invoke({"input": "langsmith가 어떻게 테스트에 도움을 줄 수 있나요?"})
```

[9]:
```
'langsmith는 테스트에 여러 가지 방식으로 도움을 줄 수 있습니다. 다음은 그 예시입니다:\n\n1. 테스트 계획 및 전략 개발: langsmith는 테스트 전략을 개발하고 테스트 계획을 수립하는 데 도움을 줄 수 있습니다. 이를 통해 테스트 범위, 우선순위 및 리소스 할당 등을 효율적으로 관리할 수 있습니다.\n\n2. 테스트 케이스 작성: langsmith는 테스트 케이스를 작성하는 데 도움을 줄 수 있습니다. 효율적인 테스트 케이스는 모든 측면을 포괄하고 오류를 식별하는 데 도움이 되며, langsmith는 문법, 스타일 및 일관성에 대한 지식을 제공할 수 있습니다.\n\n3. 테스트 자동화: langsmith는 테스트 자동화에도 도움을 줄 수 있습니다. 테스트 자동화는 반복적이고 시간 소모적인 테스트 작업을 자동화하여 효율성을 향상시키고 오류를 신속하게 찾을 수 있게 합니다. langsmith는 자동화 도구와 언어의 지식을 활용하여 이를 지원할 수 있습니다.\n\n4. 테스트 결과 분석 및 보고서 작성: langsmith는 테스트 결과를 분석하고 관련된 보고서를 작성하는 데 도움을 줄 수 있습니다. 이를 통해 테스트의 문제점과 개선 사항을 식별하고 향후 테스트 전략에 반영할 수 있습니다.\n\n5. 테스트 문서화: langsmith는 테스트 관련
```

LangChain 퀵스타트

2. Retrieval Chain

- 질문에 제대로 답하려면 추가 컨텍스트를 제공해야 합니다. 검색을 통해 이를 수행할 수 있습니다.
- Retrieval(검색)은 LLM에 직접 전달하기에는 너무 많은 데이터가 있을 때 유용합니다.
- 그런 다음 검색기를 사용하여 가장 관련성이 높은 부분만 가져와서 전달할 수 있습니다.
- 이 과정에서 Retriever에서 관련 문서를 조회한 다음 프롬프트에 전달합니다.
- Retriever는 SQL 테이블, 인터넷 등 무엇이든 뒷받침할 수 있지만, 여기서는 벡터 스토어를 채우고 이를 Retriever로 사용하겠습니다.

BeautifulSoup을 설치하고, WebBaseLoader로 웹페이지 데이터 로드

[10]:
```
%pip install beautifulsoup4
```

[11]:
```
from langchain_community.document_loaders import WebBaseLoader
loader = WebBaseLoader("https://docs.smith.langchain.com/overview")

docs = loader.load()
```

LangChain 퀵스타트

데이터를 임베딩벡터로 변환하여서 벡터스토어로 색인화

[12]:
```
%pip install faiss-cpu
```

[13]:
```
from langchain_openai import OpenAIEmbeddings

embeddings = OpenAIEmbeddings()
```

[14]:
```
from langchain_community.vectorstores import FAISS
from langchain.text_splitter import RecursiveCharacterTextSplitter



text_splitter = RecursiveCharacterTextSplitter()
documents = text_splitter.split_documents(docs)
vector = FAISS.from_documents(documents, embeddings)
```

LangChain 퀵스타트

retrieval chain 생성

[15]:
```python
from langchain.chains.combine_documents import create_stuff_documents_chain

prompt = ChatPromptTemplate.from_template("""제공된 context에만 근거하여 다음 질문에 답하세요.:

<context>
{context}
</context>

Question: {input}""")

document_chain = create_stuff_documents_chain(llm, prompt)
```

[16]:
```python
from langchain_core.documents import Document

document_chain.invoke({
    "input": "langsmith가 어떻게 테스트에 도움을 줄 수 있나요?",
    "context": [Document(page_content="langsmith를 사용하면 테스트 결과를 시각화할 수 있습니다.")]
})
```

[16]:
```
'langsmith를 사용하면 테스트 결과를 시각화할 수 있습니다. 이는 테스트 과정에서 얻은 데이터를 그래프나 차
```

LangChain 퀵스타트

[17]:
```python
from langchain.chains import create_retrieval_chain

retriever = vector.as_retriever()
retrieval_chain = create_retrieval_chain(retriever, document_chain)
```

[18]:
```python
response = retrieval_chain.invoke({"input": "LangSmith가 어떻게 테스트에 도움을 줄 수 있나요?"})
print(response["answer"])
```

```
LangSmith는 다음과 같은 방법으로 테스트에 도움을 줄 수 있습니다:

1. 데이터셋 생성: LangSmith를 사용하여 테스트에 사용할 데이터셋을 생성할 수 있습니다. 이 데이터셋은 테스트에 사용되는 프롬프트나 체인의 변경사항을 검증하는 데 사용될 수 있습니다.

2. 체인 실행 및 결과 확인: 생성된 데이터셋을 사용하여 체인을 실행하고 결과를 시각화하여 확인할 수 있습니다. 결과를 시각적으로 확인하는 것은 여전히 중요하며, LangSmith를 사용하면 이 작업을 쉽게 수행할 수 있습니다.

3. 피드백과 평가: 실행된 체인에 대한 피드백을 추가하고 평가할 수 있습니다. 피드백은 실행 결과와 연결되어 향후 분석이나 필터링에 사용될 수 있습니다. 또한 LangSmith에는 사람에 의한 평가를 위한 기능도 제공되며, 이를 통해 모델의 품질과 신뢰성을 평가할 수 있습니다.
```

LangChain 퀵스타트

3. Conversation Retrieval Chain

- LLM 애플리케이션의 주요 유형인 채팅봇 구현은 전체 히스토리를 고려해야 합니다.
- create_retrieval_chain 함수를 사용할 수 있지만 두 가지를 변경해야 합니다:
- 검색 방법은 이제 가장 최근의 입력만 처리하는 것이 아니라 전체 히스토리를 고려해야 합니다.
- 최종 LLM 체인도 마찬가지로 전체 히스토리를 고려해야 합니다.

Updating Retrieval

- 검색을 업데이트하기 위해 새 체인을 생성합니다.
- 이 체인은 가장 최근의 입력(input)과 대화 기록(chat_history)을 받아 LLM을 사용하여 검색 쿼리를 생성합니다.

LangChain 퀵스타트

[19]:
```
from langchain.chains import create_history_aware_retriever
from langchain_core.prompts import MessagesPlaceholder

# First we need a prompt that we can pass into an LLM to generate this search query

prompt = ChatPromptTemplate.from_messages(
    [
        MessagesPlaceholder(variable_name="chat_history"),
        ("user", "{input}"),
        (
            "user",
            "Given the above conversation, generate a search query to look up in order to get informat
        ),
    ]
)
retriever_chain = create_history_aware_retriever(llm, retriever, prompt)
```

LangChain 퀵스타트

사용자가 후속 질문을 하는 인스턴스를 전달하여 이를 테스트해 볼 수 있습니다.

[20]:
```python
from langchain_core.messages import HumanMessage, AIMessage

chat_history = [
    HumanMessage(content="LangSmith가 LLM 애플리케이션 테스트를 도와줄 수 있나요?"),
    AIMessage(content="예!"),
]
retriever_chain.invoke({"chat_history": chat_history, "input": "방법을 알려주세요."})
```

[20]:
```
[Document(page_content='LangSmith Overview and User Guide | 🦜🛠️ LangSmith', metadata={'source': 'https://docs.smith.langchain.com/overview', 'title': 'LangSmith Overview and User Guide | 🦜🛠️ LangSmith', 'description': 'Building reliable LLM applications can be challenging. LangChain simplifies the initial setup, but there is still work needed to bring the performance of prompts, chains and agents
```

- 이렇게 하면 LangSmith에서 테스트에 대한 문서가 반환되는 것을 볼 수 있습니다.
- 이는 LLM이 채팅 기록과 후속 질문을 결합하여 새로운 쿼리를 생성했기 때문입니다.
- 이제 새로운 retriever가 생겼으니 검색된 문서를 염두에 두고 대화를 계속할 수 있는 새로운 체인을 만들 수 있습니다.

LangChain 퀵스타트

[21]:
```
prompt = ChatPromptTemplate.from_messages(
    [
        (
            "system",
            "Answer the user's questions based on the below context:\n\n{context}",
        ),
        MessagesPlaceholder(variable_name="chat_history"),
        ("user", "{input}"),
    ]
)
document_chain = create_stuff_documents_chain(llm, prompt)

retrieval_chain = create_retrieval_chain(retriever_chain, document_chain)
```

[22]:
```
chat_history = [
    HumanMessage(content="LangSmith가 LLM 애플리케이션 테스트를 도와줄 수 있나요?"),
    AIMessage(content="Yes!"),
]
retrieval_chain.invoke({"chat_history": chat_history, "input": "방법을 알려주세요."})
```

LangChain 퀵스타트

[22]:

```
{'chat_history': [HumanMessage(content='LangSmith가 LLM 애플리케이션 테스트를 도와줄 수 있나요?'),
  AIMessage(content='Yes!')],
 'input': '방법을 알려주세요.',
 'context': [Document(page_content='LangSmith Overview and User Guide | 🦜🛠️ LangSmith', metadata=
{'source': 'https://docs.smith.langchain.com/overview', 'title': 'LangSmith Overview and User Guide |
🦜🛠️ LangSmith', 'description': 'Building reliable LLM applications can be challenging. LangChain si
mplifies the initial setup, but there is still work needed to bring the performance of prompts, chain
s and agents up the level where they are reliable enough to be used in production.', 'language': 'e
n'}),
  Document(page_content="Skip to main content🦜🛠️ LangSmith DocsPython DocsJS/TS DocsSearchGo to App
LangSmithOverviewTracingTesting & EvaluationOrganizationsHubLangSmith CookbookRelease NotesOverviewOn
this pageLangSmith Overview and User GuideBuilding reliable LLM applications can be challenging. Lang
Chain simplifies the initial setup, but there is still work needed to bring the performance of prompt
s, chains and agents up the level where they are reliable enough to be used in production.Over the pa
st two months, we at LangChain have been building and using LangSmith with the goal of bridging this
gap. This is our tactical user guide to outline effective ways to use LangSmith and maximize its bene
fits.On by default\u200bAt LangChain, all of us have LangSmith's tracing running in the background by
default. On the Python side, this is achieved by setting environment variables, which we establish wh
```

LangChain 퀵스타트

4. Agent

- Agent는 LLM이 수행할 단계를 결정합니다.
- Agent를 만들 때 Agent가 액세스할 수 있는 도구를 정합니다.
- 이 예에서는 에이전트에게 두 가지 툴에 대한 액세스 권한을 부여하겠습니다:

1. 방금 만든 Retriever : LangSmith에 대한 질문에 쉽게 답변
2. Search tool : 최신 정보가 필요한 질문에 답변

Retriever를 위한 툴을 설정

[23]:
```
from langchain.tools.retriever import create_retriever_tool

retriever_tool = create_retriever_tool(
    retriever,
    "langsmith_search",
    "Search for information about LangSmith. For any questions about LangSmith, you must use this tool
)
```

LangChain 퀵스타트

Search를 위한 툴 설정

- Search 엔진으로" Tavily 사용 : API Key - https://app.tavily.com/
- export TAVILY_API_KEY=...

[24]:
```
import os

os.environ["TAVILY_API_KEY"] = "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"
```

[25]:
```
from langchain_community.tools.tavily_search import TavilySearchResults

search = TavilySearchResults()
```

tool list

[26]:
```
tools = [retriever_tool, search]
```

미리 정의된 프롬프트를 사용하기 위해 langchain hub 설치

[27]:
```
%pip install langchainhub
```

LangChain 퀵스타트

[28]:
```python
from langchain_openai import ChatOpenAI
from langchain import hub
from langchain.agents import create_openai_functions_agent
from langchain.agents import AgentExecutor

# Get the prompt to use - you can modify this!
prompt = hub.pull("hwchase17/openai-functions-agent")
llm = ChatOpenAI(model="gpt-3.5-turbo", temperature=0)
agent = create_openai_functions_agent(llm, tools, prompt)
agent_executor = AgentExecutor(agent=agent, tools=tools, verbose=True)
```

이제 agent를 호출하고 agent의 응답을 확인할 수 있습니다.

[29]:
```python
agent_executor.invoke({"input": "LangSmith가 테스트에 어떤 도움을 줄 수 있나요?"})
```

LangChain 퀵스타트

```
> Entering new AgentExecutor chain...
LangSmith는 다음과 같은 도움을 제공할 수 있습니다:

1. 정보 검색: LangSmith를 사용하여 원하는 주제에 대한 정보를 검색할 수 있습니다. LangSmith는 신뢰할 수 있는 소스에서 정보를 가져와 정확하고 포괄적인 결과를 제공합니다.

2. 질문에 대한 답변: LangSmith는 다양한 주제에 대한 질문에 대한 답변을 제공할 수 있습니다. 예를 들어, 역사적 사건, 과학적 개념, 문화적 이벤트 등에 대한 질문에 대한 답변을 얻을 수 있습니다.

3. 최신 뉴스 및 현재 이벤트: LangSmith는 신문 기사, 블로그 게시물, 소셜 미디어 게시물 등을 검색하여 최신 뉴스 및 현재 이벤트에 대한 정보를 제공할 수 있습니다.

4. 언어 지원: LangSmith는 다양한 언어를 지원하며, 다른 언어로 된 문서를 번역하거나 다른 언어로 된 질문에 대한 답변을 제공할 수 있습니다.

LangSmith는 다양한 분야에서 도움을 제공하는 다재다능한 도구입니다. 어떤 도움이 필요한지에 따라 LangSmith를 활용할 수 있습니다.

> Finished chain.
```

[29]:
```
{'input': 'LangSmith가 테스트에 어떤 도움을 줄 수 있나요?',
 'output': 'LangSmith는 다음과 같은 도움을 제공할 수 있습니다:\n\n1. 정보 검색: LangSmith를 사용하여 원하는 주제에 대한 정보를 검색할 수 있습니다. LangSmith는 신뢰할 수 있는 소스에서 정보를 가져와 정확하고 포
```

LangChain 퀵스타트

[30]:
```
agent_executor.invoke({"input": "오늘 서울 날씨는?"})
```

[30]:
```
> Entering new AgentExecutor chain...

Invoking: `tavily_search_results_json` with `{'query': '오늘 서울 날씨'}`


[{'url': 'https://www.newsis.com/view/?id=NISX20240209_0002623004', 'content': '2024.02.10 (토) 서울 6℃ 사회 구름 많고 흐린 설날…미세먼지 \'나쁨\'[오늘날씨] 등록 2024.02.10 04:00:00수정 2024.02.10 06:15:29 전국 가끔 구름 많음…오후엔 \'흐림\'  [서울=뉴시스] 김명년 기자 = 서울지역 초미세먼지 농도가 나쁨 수준을 보인 지난달 30일 오전 서울 용산구 국립중앙박물관에서 바라본 남산타워가 뿌옇다. 2024.01.30. kmn@newsis.com  내연녀 외도의심 흉기 휘두른 50대, 징역 5년 짧은 설 연휴, \'서울 나들이\' 어때요…가볼 만한 곳은 인천, 구름 많고 큰 일교차…미세먼지 \'나쁨\'  아침 최저 -7~1도, 낮 최고 5~11도2024.02.10 (토) 서울 -0℃ ... 6구름 많고 흐린 설날…미세먼지 \'나쁨\'[오늘날씨] 7제주, 설날 낮 동안 가끔 비…최고 11~12도1"거대 ...'}]오늘 서울의 날씨는 구름이 많고 흐린 상태이며, 기온은 6도입니다. 또한, 미세먼지 농도는 '나쁨' 수준입니다. 자세한 내용은 [여기](https://www.newsis.com/view/?id=NISX20240209_0002623004)에서 확인하실 수 있습니다.

> Finished chain.
```

LangChain 퀵스타트

[31]:
```
chat_history = [
    HumanMessage(content="LangSmith가 LLM 애플리케이션 테스트를 도와줄 수 있나요?"),
    AIMessage(content="예!"),
]
agent_executor.invoke({"chat_history": chat_history, "input": "방법을 알려주세요."})
```

```
> Entering new AgentExecutor chain...
LLM 애플리케이션 테스트를 도와드리기 위해 다음과 같은 방법을 제안합니다:

1. LLM 애플리케이션 요구사항 이해: LLM 애플리케이션의 요구사항을 자세히 이해하고, 어떤 테스트가 필요한지 파악합니다.

2. 테스트 계획 수립: 테스트 목표와 범위를 설정하고, 테스트 계획을 수립합니다. 이 단계에서는 어떤 테스트를 수행할 것인지, 어떤 테스트 케이스를 작성할 것인지 등을 결정합니다.

3. 테스트 환경 설정: LLM 애플리케이션을 테스트하기 위한 환경을 설정합니다. 이는 테스트용 서버, 데이터베이스, 테스트 데이터 등을 포함할 수 있습니다.
```

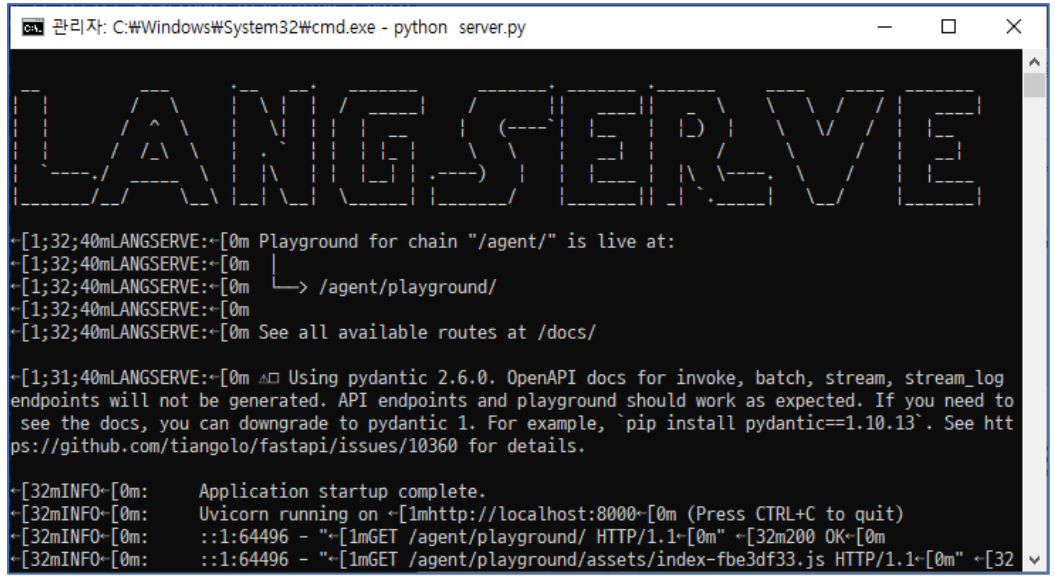
LangChain 퀵스타트

5. Serving with LangServe

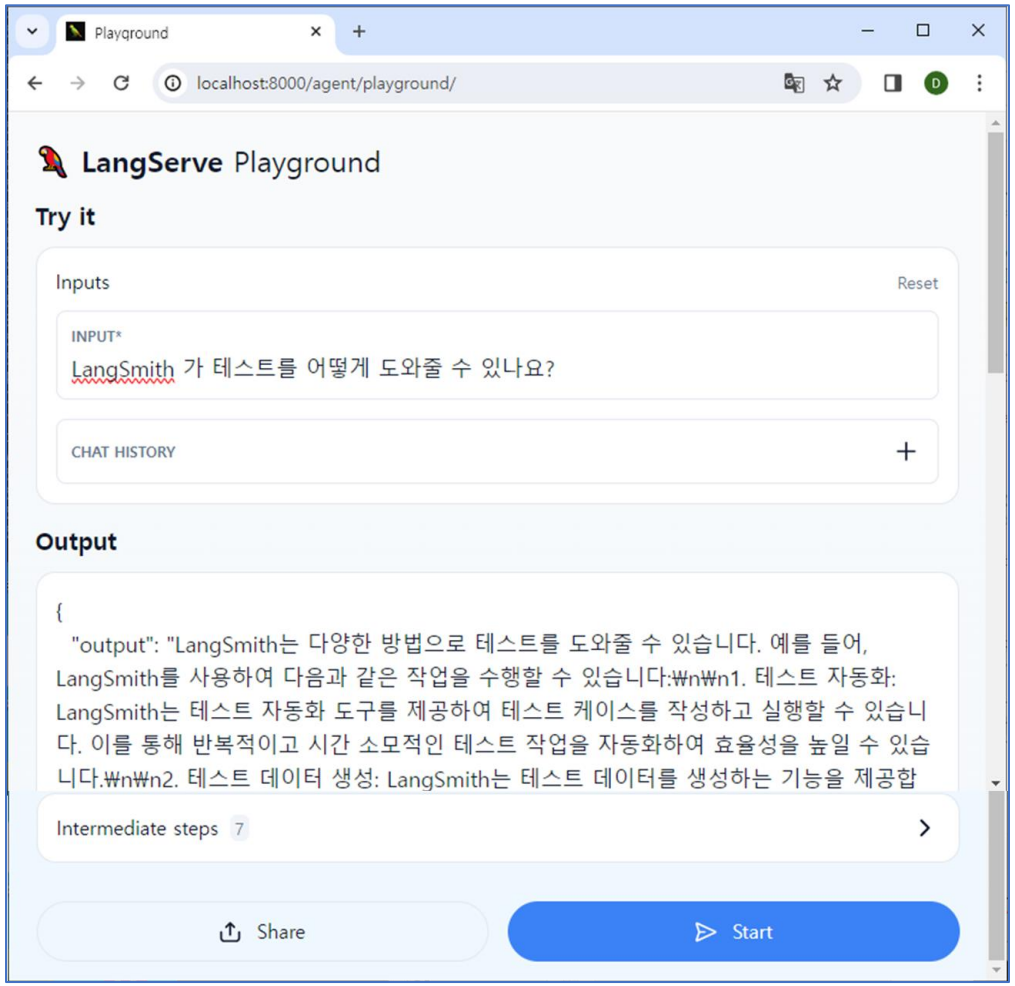
- langserve, fastapi를 설치하고 로컬에서 실행 :

[32]:
```
%pip install langserve fastapi sse_starlette
```

서버 실행 python server.py



Playground http://localhost:8000/agent/playground/



LangChain 퀵스타트

Client

```
from langserve import RemoteRunnable

remote_chain = RemoteRunnable("http://localhost:8000/agent/")
remote_chain.invoke(
    {
        "input": "LangSmith가 테스트에 어떤 도움을 줄 수 있나요?",
        "chat_history": [],  # Providing an empty list as this is the first call
    }
)
```

```
{'output': 'LangSmith는 다음과 같은 도움을 제공할 수 있습니다:\n\n1. 정보 검색: LangSmith를 사용하여 원하는 주제에 대한 정보를 검색할 수 있습니다. LangSmith는 신뢰할 수 있는 소스에서 정보를 가져와 정확하고 포괄적인 결과를 제공합니다.\n\n2. 질문에 대한 답변: LangSmith는 다양한 주제에 대한 질문에 대한 답변을 제공할 수 있습니다. 예를 들어, 역사적 사건, 과학적 개념, 문화적 이벤트 등에 대한 질문에 대한 답변을 얻을 수 있습니다.\n\n3. 최신 뉴스 및 현재 이벤트: LangSmith는 신문 기사, 블로그 게시물, 소셜 미디어 게시물 등을 검색하여 최신 뉴스 및 현재 이벤트에 대한 정보를 제공할 수 있습니다.\n\n4. 언어 번역: LangSmith는 다양한 언어 간의 번역을 지원합니다. 따라서 다른 언어로 작성된 문서를 번역하거나 다른 언어로 대화하는 데 도움을 줄 수 있습니다.\n\n5. 문서 요약: LangSmith는 긴 문서를 요약하여 핵심 내용을 추출할 수 있습니다. 이를 통해 시간을 절약하고 중요한 정보를 빠르게 파악할 수 있습니다.\n\nLangSmith는 다양한 분야에서 도움을 줄 수 있으
```

LCEL (LangChain Expression Language)

LCEL은 체인을 쉽게 구성할 수 있는 선언적 방법입니다.

주요 기능

- 스트리밍 지원 : 체인의 첫 번째 토큰을 빠르게 받을 수 있도록 함.
- 비동기 지원 : 동기 및 비동기 API 모두에서 체인 호출 가능.
- 최적화된 병렬 실행 : 병렬 실행 가능한 단계는 자동으로 최적화.
  예를 들어 여러 검색기에서 문서를 동시에 가져옴.
- 재시도 및 폴백 : 모든 체인 부분에 대해 재시도 및 폴백 구성 가능.
  대규모로 체인을 안정적으로 만드는 데 유용.
- 중간 결과 액세스 : 복잡한 체인에서 중간 단계의 결과에 액세스 가능. 사용자에게 상태 알림, 디버깅에 유용.
- LangSmith 추적 통합 : 모든 단계에서 발생하는 일을 LangSmith에 자동으로 기록.
  관찰 가능성과 디버깅 가능성 극대화.
- LangServe 배포 통합 : LCEL로 생성된 모든 체인은 LangServe를 통해 쉽게 배포 가능.
  LangServe 활용으로 체인 관리와 배포 간소화.

https://python.langchain.com/docs/expression_language/

LCEL (LangChain Expression Language)

Invoke

Without LCEL

```
from typing import List

import openai


prompt_template = "Tell me a short joke about {topic}"
client = openai.OpenAI()

def call_chat_model(messages: List[dict]) -> str:
    response = client.chat.completions.create(
        model="gpt-3.5-turbo",
        messages=messages,
    )
    return response.choices[0].message.content

def invoke_chain(topic: str) -> str:
    prompt_value = prompt_template.format(topic=topic)
    messages = [{"role": "user", "content": prompt_value}]
    return call_chat_model(messages)

invoke_chain("ice cream")
```

LCEL

```
from langchain_core.runnables import RunnablePassthrough


prompt = ChatPromptTemplate.from_template(
    "Tell me a short joke about {topic}"
)
output_parser = StrOutputParser()
model = ChatOpenAI(model="gpt-3.5-turbo")
chain = (
    {"topic": RunnablePassthrough()}
    | prompt
    | model
    | output_parser
)

chain.invoke("ice cream")
```

https://python.langchain.com/docs/expression_language/why

LCEL (LangChain Expression Language)

Stream

Without LCEL

```
from typing import Iterator


def stream_chat_model(messages: List[dict]) -> Iterator[str]:
    stream = client.chat.completions.create(
        model="gpt-3.5-turbo",
        messages=messages,
        stream=True,
    )
    for response in stream:
        content = response.choices[0].delta.content
        if content is not None:
            yield content

def stream_chain(topic: str) -> Iterator[str]:
    prompt_value = prompt.format(topic=topic)
    return stream_chat_model([{"role": "user", "content": prom


for chunk in stream_chain("ice cream"):
    print(chunk, end="", flush=True)
```

LCEL

```
for chunk in chain.stream("ice cream"):
    print(chunk, end="", flush=True)
```

https://python.langchain.com/docs/expression_language/why

LCEL (LangChain Expression Language)

Batch

Without LCEL

```
from concurrent.futures import ThreadPoolExecutor


def batch_chain(topics: list) -> list:
    with ThreadPoolExecutor(max_workers=5) as executor:
        return list(executor.map(invoke_chain, topics))

batch_chain(["ice cream", "spaghetti", "dumplings"])
```

LCEL

```
chain.batch(["ice cream", "spaghetti", "dumplings"])
```

https://python.langchain.com/docs/expression_language/why

LCEL (LangChain Expression Language)

Async

Without LCEL

```
async_client = openai.AsyncOpenAI()

async def acall_chat_model(messages: List[dict]) -> str:
    response = await async_client.chat.completions.create(
        model="gpt-3.5-turbo",
        messages=messages,
    )
    return response.choices[0].message.content

async def ainvoke_chain(topic: str) -> str:
    prompt_value = prompt_template.format(topic=topic)
    messages = [{"role": "user", "content": prompt_value}]
    return await acall_chat_model(messages)
```

```
await ainvoke_chain("ice cream")
```

LCEL

```
chain.ainvoke("ice cream")
```

https://python.langchain.com/docs/expression_language/why

LCEL (LangChain Expression Language)

LLM instead of chat model

Without LCEL

```
def call_llm(prompt_value: str) -> str:
    response = client.completions.create(
        model="gpt-3.5-turbo-instruct",
        prompt=prompt_value,
    )
    return response.choices[0].text

def invoke_llm_chain(topic: str) -> str:
    prompt_value = prompt_template.format(
    return call_llm(prompt_value)

invoke_llm_chain("ice cream")
```

LCEL

```
from langchain_openai import OpenAI

llm = OpenAI(model="gpt-3.5-turbo-instruct
llm_chain = (
    {"topic": RunnablePassthrough()}
    | prompt
    | llm
    | output_parser
)

llm_chain.invoke("ice cream")
```

https://python.langchain.com/docs/expression_language/why

LCEL (LangChain Expression Language)

Different model provider

Without LCEL

```
import anthropic

anthropic_template = f"Human:\n\n{prompt_t
anthropic_client = anthropic.Anthropic()

def call_anthropic(prompt_value: str) -> s
    response = anthropic_client.completion
        model="claude-2",
        prompt=prompt_value,
        max_tokens_to_sample=256,
    )
    return response.completion

def invoke_anthropic_chain(topic: str) ->
    prompt_value = anthropic_template.form
    return call_anthropic(prompt_value)

invoke_anthropic_chain("ice cream")
```

LCEL

```
from langchain_community.chat_models impor

anthropic = ChatAnthropic(model="claude-2"
anthropic_chain = (
    {"topic": RunnablePassthrough()}
    | prompt
    | anthropic
    | output_parser
)

anthropic_chain.invoke("ice cream")
```

https://python.langchain.com/docs/expression_language/why

Thank you

kgpark88@gmail.com