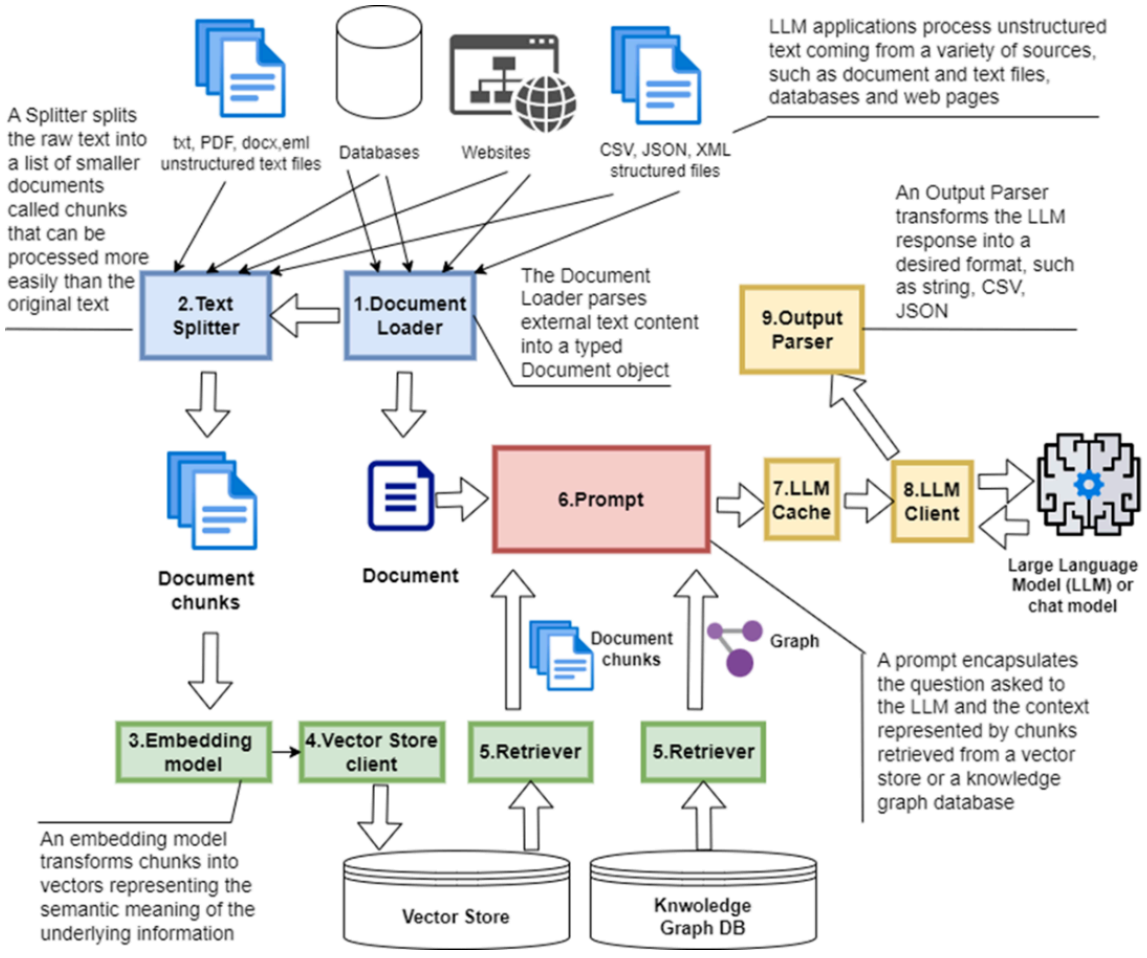
LangChain 프레임워크 활용 방안 연구

Lesson 1 랭체인을 활용한 LLM어플리케이션 개발

- LangChain 아키텍처
- 가장 대표적인 LLM기반 응용프로그램
- Langchain 프로그래밍 예제
- LLM 이란 무엇인가?

---

1. LangChain 아키텍처



LangChain 아키텍처

Langchain을 기반으로 만들어진 LLM 애플리케이션은 다양한 유형의 입력소스(txt, PDF, docx, eml(비정형 텍스트 파일), 데이터베이스, 웹사이트, CSV, JSON, XML 문서 및 텍스트 파일)를 효과적으로 처리할 수 있다.

주요 구성 요소:

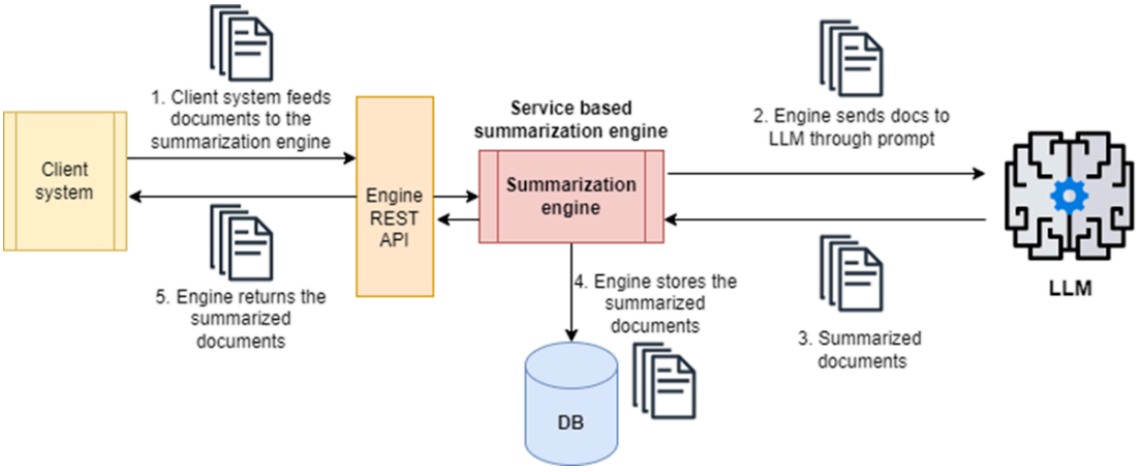
- **1) 문서 로더(Document Loader)**: 외부 텍스트 콘텐츠를 정형화된 문서 객체로 파싱
- **2) 텍스트 분할기(Text splitter)**: 원시 텍스트를 원본 텍스트보다 더 쉽게 처리할 수 있는 청크(chunks)라고 불리는 작은 문서 목록으로 분할하는 역할.
- **3) 임베딩 모델(Embedding Model)**: 청크를 기본 정보의 의미론적 의미를 나타내는 벡터로 변환함
- **4) 벡터 저장소 클라이언트 (Vector storage client)**: 임베딩된 벡터를 저장
- **5) 검색기(Retriever)**: 벡터 저장소나 지식 그래프 데이터베이스에서 관련 청크를 검색
- **6) 프롬프트(prompt)**: LLM에 묻는 질문과 벡터 저장소나 지식 그래프 데이터베이스에서 검색한 청크로 표현된 컨텍스트를 캡슐화 함.
- **7) LLM 캐시(LLM cache)**: 결과를 캐싱한다
- **8) LLM 클라이언트(LLM client)**: 대규모 언어 모델(LLM) 또는 채팅 모델과 통신
- **9) 출력 파서(Output parser)**: LLM 응답을 문자열, CSV, JSON과 같은 원하는 형식으로 변환
- **10) 체인 (Chain)**: LangChain의 처리 워크플로우를 안내하는 복합적인 구성으로, 특정 사용 사례에 맞게 사용자 정의되며 앞서 설명한 구성 요소들의 순차적 배열을 기반으로 작동함
- **11) 에이전트 (Agent):** 이 구성 요소는 순차적 체인을 확장하여 동적 워크플로우를 관리한다. 에이전트의 처리 과정은 유연하며 사용자 입력이나 구성 요소 출력에 따라 유연하게 적응할 수 있다. 동적 워크플로우에서 사용되는 리소스는 대부분의 프레임워크 문서에서 "도구(tools)" 또는 "플러그인(plugins)"이라고 불리며, 모든 도구의 모음을 "도구 모음(toolkit)"이라고 한다.

2. 가장 대표적인 LLM기반 응용프로그램

LangChain의 포괄적인 설계는 세 가지 주요 솔루션을 가능하게 한다 - **1) Summarization/query engine, 2) chatbot, and 3) Knowledge agent**

2-1. Summarization/query engine

LLM 기반 엔진은 다양한 컴퓨터 시스템을 위한 백엔드 도구로 작동하며, LLM에 대한 특정 요청을 처리한다. 예를 들어, 긴 텍스트 구절을 간결한 요약으로 압축하는 요약 엔진으로 기능할 수 있다. 이러한 요약은 즉시 클라이언트에게 반환 되거나 다른 애플리케이션이 사용할 수 있도록 데이터베이스에 저장될 수 있으며, 이는 아래 그림에 잘 요약돼 있다.



LLM 서비스 기반 요약 엔진 Workflow 설명

1) 클라이언트 시스템이 요약 엔진에 문서를 전달함

2) 엔진이 프롬프트를 통해 LLM에 문서를 전송함

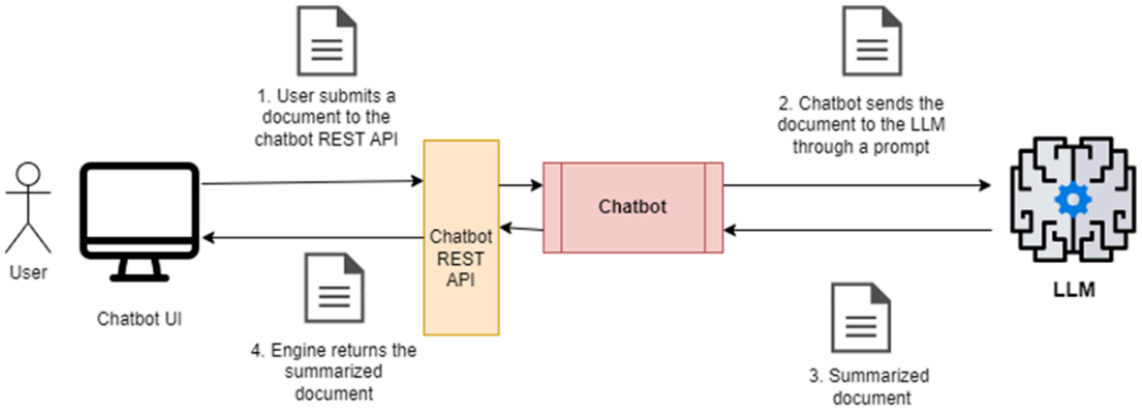
3) 요약된 문서

4) 엔진이 요약된 문서를 저장함

5) 엔진이 요약된 문서를 반환함

2-2. LLM-based Chatbots

다양한 유형의 챗봇들은 다음과 같은 일반적인 작업에 특화되어 있다: 텍스트 요약, 질문 답변, 언어 번역. 챗봇은 이미 가지고 있는 지식(예: 벡터 저장소에 위치한)을 사용하거나 사용자 인터페이스를 통해 입력을 받아 작업을 수행하는 데 도움을 준다.



1. 사용자가 챗봇 REST API로 문서를 제출
2. 챗봇이 프롬프트를 통해 LLM에 문서를 전송
3. 요약된 문서
4. 엔진이 요약된 문서를 반환

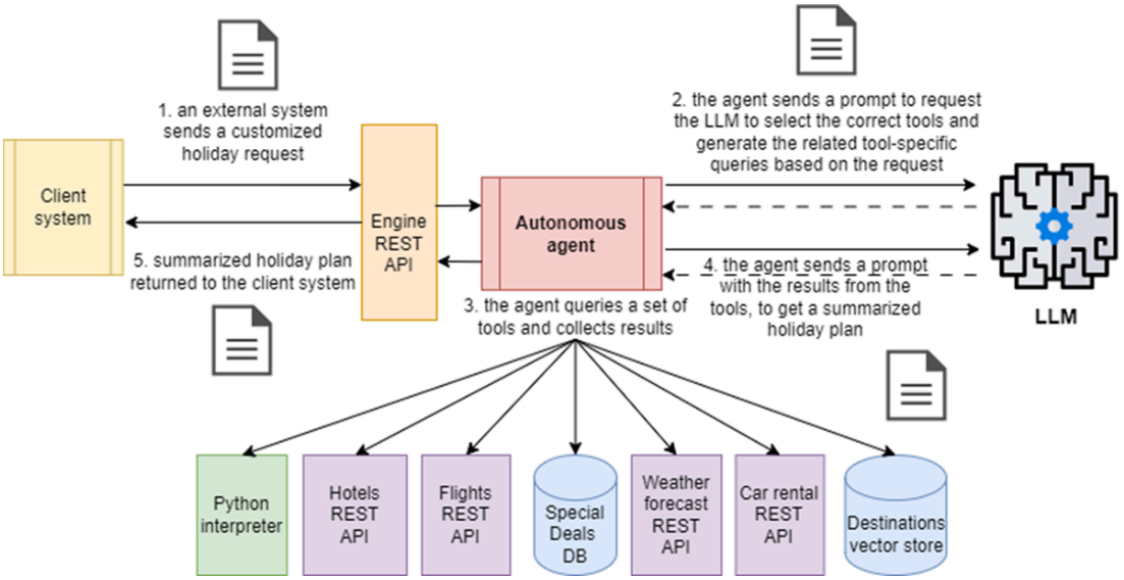
요약 챗봇은 요약 엔진과 몇 가지 유사한 점이 있지만, LLM과 사용자가 함께 결과를 미세조정하고 개선할 수 있는 대화형(상호작용) 경험을 제공한다.

2-3. LLM-based autonomous agents

LLM 기반 자율 에이전트는 다양한 데이터 소스와 분기 워크플로우(Branching workflow)를 다룰 때 LLM과 협력하여 복잡한 워크플로우를 처리하도록 설계된 정교한 도구다.

이 자율 에이전트는 다양한 구조화된 데이터 소스와 비구조화된 데이터 소스에 연결하여 복잡한 작업을 수행하도록 설계되었다. LLM과 상호작용할 때 워크플로우의 각 단계에서 높은 수준의 독립성을 보여주며, 최종적으로 포괄적인 결과를 제공할 수 있다.

예를 들어, 여행사가 외부 클라이언트 시스템(예: agoda, 야놀자 등)에서 오는 자연어 요청을 기반으로 휴일 패키지를 만들기 위해 자율 에이전트를 사용하는 시나리오늘 생각해보자. 아래 그림에 나온 것처럼, 자율 에이전트는 LLM에 프롬프트를 보내 요청에 맞는 도구(예: 항공, 호텔, 렌터카 제공업체, 날씨 예보 서비스, 내부 휴일 할인 데이터베이스)를 선택하도록 요청할 수 있다. LLM은 개발자가 잘 설계한 프롬프트에 따라 관련 도구를 선택하고, 휴일 할인 데이터베이스용 SQL 쿼리나 제공 업체용 REST API 요청 같은 필요한 쿼리를 생성한다. 에이전트는 각 쿼리를 실행하고 결과를 수집한 뒤, 원래 휴일 요청과 쿼리된 모든 도구의 결과를 포함한 컨텍스트와 함께 LLM에 또 다른 프롬프트를 보낸다. LLM은 모든 예약을 포함한 요약된 휴일 계획으로 응답하고, 에이전트는 이를 외부 클라이언트 시스템에 반환할 수 있다.



실제 어플리케이션에서의 워크플로우는 위 그림보다 더 복잡할 수 있다. 예를 들어, 최종 휴일 계획이 생성되기 전에 에이전트와 LLM간에 여러 번의 Loop을 통한 미세조정이 필요할 수도 있으며, 금융이나 의료 군사 등과 같은 중요한 응용 분야에서는 에이전트에 “사람의 개입(Human-in-the loop)” 단계를 포함할 수도있다.

3. Langchain 프로그래밍 예제

3-1. 문장 완성 예시 (sentence completion)

```
from langchain_openai import ChatOpenAI
import getpass

OPENAI_API_KEY = getpass.getpass('Enter your OPENAI_API_KEY')

llm = ChatOpenAI(openai_api_key=OPENAI_API_KEY,
                 model_name="gpt-4o-mini")

llm.invoke("It's a hot day, I would like to go to the...")
```

위 명령어를 입력하면 아래와 같은 출력값을 볼 수 있다.

```
AIMessage(content="...beach to cool off and relax in the refreshing water. The sound of the waves crashing against the shore and the feeling of the warm sand beneath my feet is exactly what I need to unwind and escape from the heat.", response_metadata={'finish_reason': 'stop', 'logprobs': None})
```

3-1. Prompt Engineering 예시

프롬프트는 LLM에게 작업을 수행하고 응답을 생성하도록 지시하는 명령을 의미한다. 이는 LLM 응용 프로그램의 핵심 부분으로, LLM 응용 프로그램 개발 시 종종 시행착오를 통해 프롬프트를 설계하고 다듬는 데 많은 시간을 투

자한다. 프롬프트를 둘러싼 다양한 패턴과 기술이 이미 형성되고 있으며, 이를 통해 프롬프트 엔지니어링이라는 학문이 등장하고 있다. 이 분야는 최상의 답변을 얻을 수 있는 프롬프트를 만드는 데 초점을 맞춘다.

가장 간단한 프롬프트 예시는 다음과 같다.

```
prompt_input = """Write a short message to remind users to be
vigilant about phishing attacks."""
response = llm.invoke(prompt_input)
print(response.content)
```

결과:

```
Just a friendly reminder to stay vigilant against phishing attacks. Be cautious of any
suspicious emails, messages, or requests for personal information. Stay safe online!
```

3-2. Prompt 템플릿 예시

프롬프트 템플릿은 같은 주제의 다양한 버전을 실행할 수 있도록 구조화된 프롬프트를 의미한다. LangChain은 이를 위해 PromptTemplate이라는 클래스를 제공한다. 이 클래스의 역할은 템플릿 구조(template structure)와 입력 매개변수(input parameters)를 이용하여 프롬프트를 생성하는 것이다. 아래는 템플릿에서 프롬프트를 생성하고 실행하는 예제이다.

```
from langchain_core.prompts import PromptTemplate

segovia_aqueduct_text = """The Aqueduct of Segovia (Spanish: Acueducto de Segovia) is a
Roman aqueduct in Segovia, Spain. It was built around the first century AD to channel
water from springs in the mountains 17 kilometres (11 mi) away to the city's fountains,
public baths and private houses, and was in use until 1973. Its elevated section, with
its complete arcade of 167 ...."""

prompt_template = PromptTemplate.from_template("You are an experienced copywriter.
Write a {num_words} summary of the following text, using a {tone} tone:{text}")

prompt_input = prompt_template.format(
   text=segovia_aqueduct_text,
   num_words=20,
   tone="knowledgeable and engaging")

response = llm.invoke(prompt_input)
print(response.content)
```

위 코드를 실행하면 다믕과 유사한 결과가 나온다:

```
The Aqueduct of Segovia, a Roman marvel in Spain, dates back to the 1st century AD,
channeling water for centuries.
```

3-3. LangChain의 LCEL(LangChain Expression Language) Chain 기능을 활용한 코드 실행

LangChain을 사용하는 이점 중 하나는 "체인(chain)"이라는 개념을 중심으로 구축된 처리 기술이다. 체인은 특정 결과를 달성하기 위해 구성 요소들을 연결한 파이프라인이다. 예를 들어, (이 코드 조각을 실행하지는 말고) 웹사이트에서 최신 뉴스를 스크랩하고, 이를 요약한 뒤, 누군가에게 이메일로 보내는 체인을 다음과 같이 만들 수 있다:

```
chain = web_scraping | prompt | llm_mode | email_text
```

다음은 LCEL을 활용한 또다른 코드 예시이다:

```
prompt_template = PromptTempalte.from_template("You are an experienced copywriter.
Write a {num_words} words summary of the following text, using a {tone} tone: {text}"

llm = ChatOpenAI(openai_api_key = OPENAI_API_KEY, model_name="gpt-4o-mini"

chain = prompt_template | llm

response = chain.invoke({"text": segovia_aqueduct_text,
                         "num_words": 20,
                         "tone": "knowledgeable and engaging"})
print(response.content)
```

위 코드 실행 시 다음과 같은 결과 값을 얻을 수 있다.

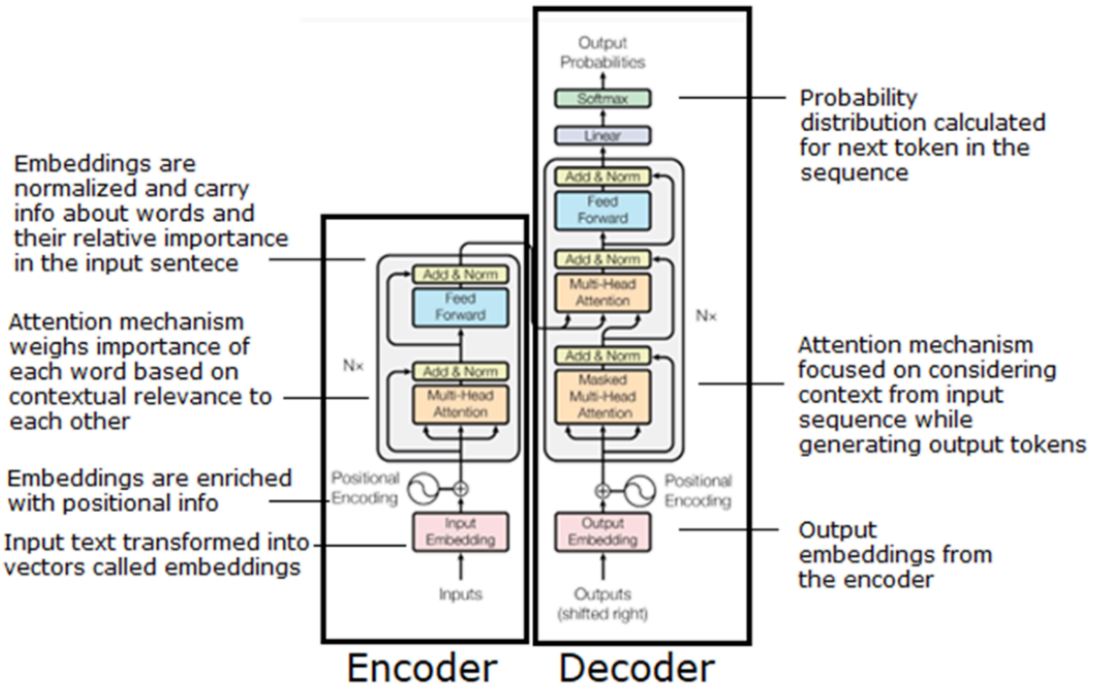
```
The Aqueduct of Segovia: A Roman marvel channeling water to the city, adorned with 167
arches, symbolizing Segovia's rich history.
```

4. LLM(Large Language Model)이란 무엇인가?

대규모 언어 모델(LLM)은 인간 언어와 관련된 작업, 예를 들어 텍스트 이해, 생성, 요약, 번역 등을 위해 설계된 딥러닝 신경망이다. LLM은 기사, 책, 소셜 미디어, 웹사이트 같은 소스에서 얻은 엄청난 양의 텍스트 데이터로 훈련된다. 이 모델들은 언어를 "토큰"이라는 단위로 처리하는데, 토큰은 대략 단어와 비슷하지만 일부 단어는 여러 토큰으로 나뉠 수도 있다. 가장 큰 LLM은 수조 개의 토큰으로 훈련돼서 인간 언어 능력을 흉내낼 수 있다.이런 광범위한 훈련 덕분에 LLM은 문법, 문장 구조, 맥락을 잘 이해한다. 인간처럼 텍스트를 해석하고 생성할 수 있어서 "생성형 AI"로 분류된다. (이 카테고리에는 오디오나 비디오 같은 콘텐츠를 만드는 기술도 포함된다.) 예를 들어, LLM은 원시 제품 데이터를 바탕으로 온라인 상점의 제품 설명을 자동으로 만들어낼 수 있다. LLM은 자연어 기능을 애플리케이션에 탑재하여 사용자 상호작용을 더 매끄럽게 할 수 있다. 사용자는 "프롬프트(prompt)"를 통해 LLM과 상호작용하는데, 프롬프트에는 작업에 대한 "지시(instruction)"와 작업을 이해하는데 필요한 텍스트인 "맥락(context)"이 포함된다. 프롬프트는 채팅 대화 형식일 수도 있고, REST API를 통해 프로그램적으로 전달될 수도 있다.

LLM은 문장에서 다음 단어나 토큰을 계속 예측하도록 설계되었다. LLM은 프롬프트가 주어지면 이를 분석해서 이전에 예측한 단어들을 고려하며 다음 단어를 예측할 수 있다. 예를 들어, "It’s a hot day, I would like to go to the,"라는 입력이 있으면 LLM은 다음 단어로 "beach"을 예측할 수 있는데, 이를 기술적으로 "완성(completion)"이라고 부른다.

LLM은 훨씬 더 인상적이고 다양한 작업을 처리할 수 있다. 주어진 문장에서 다음 단어를 예측하는 것을 넘어, 최소한의 초기 입력만으로 전체 문장, 단락, 심지어 긴 에세이나 소설까지 작성할 수 있다. 이는 주어진 프롬프트와 이전 출력에 기반해 텍스트를 계속 생성함으로써 이루어진다. LLM이 문장에서 다음 단어를 예측하고 단락이나 에세이 같은 전체 텍스트를 생성하는 놀라운 능력은 "트랜스포머 구조"에 그 뿌리를 두고 있다. 이 구조는 2017년 구글 브레인 팀, 구글 리서치, 그리고 토론토 대학교가 "Attention is All You Need"라는 논문 (https://arxiv.org/pdf/1706.03762.pdf)에서 소개했다. 이 구조는 단어들이 서로 어떻게 관련되는지를 세심하게 분석하며 언어 이해를 깊이 탐구한다. 구글 브레인의 논문에 나오는 아래 그림은 트랜스포머가 어떻게 작동하는지 개략적으로 보여준다.



위 그림에서 설명된 Transformer 아키텍처의 작동 원리는 간략하게 다음과 같다.

LLM은 일반적으로 인코더와 디코더라는 두 가지 주요 구성 요소를 가진다(단, GPT 계열 같은 일부 모델은 디코더만 사용한다). 인코더는 입력 텍스트를 "임베딩"이라는 벡터로 변환하는데, 이는 단어의 의미와 맥락을 포착한다. 이 임베딩에는 각 단어나 토큰의 위치 및 맥락적 세부 정보가 포함된다. 디코더는 인코더의 임베딩과 이전 단어들을 사용해 시퀀스의 다음 단어를 예측함으로써 출력을 생성한다. "어텐션" 메커니즘은 모델이 입력과 출력의 핵심 부분에 집중하도록 도와주며, 텍스트에서 단어들이 멀리 떨어져 있더라도 그들 간의 관계를 이해할 수 있게 한다. 작동 원리에 대한 더 자세한 설명은 나중에 하도록 하겠다. 여기서는 기본 작동 원리 정도만 간략하게 이해하고 넘어가도 된다.

4-1. LLM은 주로 어떻게 사용되는가?

LLM은 텍스트 분류, 코드 생성, 논리적 추론과 같은 다양한 작업에 사용된다. 다음은 가장 일반적인 실제 사용 사례들이다:

- **텍스트 분류 및 감정 분석(Text classification & Sentiment analysis)**: LLM은 뉴스 기사를 카테고리로 분류하거나 감정 분석을 기반으로 주식을 추천하는 등의 기능을 수행할 수 있다.
- **자연어 이해 및 생성(Natural language understanding ad generation)**: LLM은 텍스트의 주요 주제를 파악하거나 길이, 톤, 용어 선호도에 따라 요약을 생성할 수 있다. Duolingo는 AI를 사용해 수업 준비 시간을

혁신적으로 단축하고 있으며 관련 내용은 블로그 포스트 “How Duolingo Uses AI to Create Lessons Faster”에 자세히 설명되어 있다.

- **의미검색(semantic search)**: LLM은 단순 키워드가 아닌 질문의 맥락과 의도를 사용해 데이터베이스를 조회할 수 있다. Picnic 슈퍼마켓 앱은 레시피 검색을 개선하기 위해 LLM을 사용하며, 이는 Medium 포스트 “Enhancing Search Retrieval with Large Language Models (LLMs)”에 설명되어 있다.
- **자율추론 및 워크플로우 실행(Autonomous reasoning and workflow execution)**: LLM은 사용자의 요청을 이해하고 워크플로우 전반에 걸쳐 입력과 출력을 처리해 완전한 휴가 패키지 계획/개발과 같은 작업을 효과적으로 처리할 수 있다(2-3 LLM-based autonomous agents 참고).
- **구조화된 데이터 추출(Structured data extraction)**: LLM은 뉴스 기사나 재무 보고서에서 엔터티와 그 관계를 추출하는 등의 비정형 텍스트에서 구조화된 데이터를 추출할 수 있다.
- **코드 이해 및 생성(Code understanding and generation)**: LLM은 프로그래머가 작성한 코드에서 문제를 발견하거나, 사용자의 지시에 따라 함수, 클래스, 심지어 전체 애플리케이션을 생성하는 등의 작업에도 활용될 수 있다.

4-2. LLM을 사용자의 필요/요구에 맞춰 조정하는 방법

LLM은 특정 도메인에 대한 사전 지식이나 추가 훈련 없이도 사용자의 요청과 의도에 맞춰 응답하는 능력을 향상시킬 수 있는 몇가지 핵심 기술들을 가지고 있다. 여기에는 주로 다음 기술들이 포함된다:

- Prompt Engineering
- Retrieval Augmented Generation (RAG)
- Fine-tuning

Prompt Engineering:

LLM에 주어지는 프롬프트는 간단한 요청부터 추가 데이터나 맥락이 포함된 복잡한 지시까지 다양하다. 프롬프트 엔지니어링은 LLM이 사용자의 의도를 효과적으로 이해하고 응답할 수 있도록 하는데 그 목적이 있다. 이 기법을 활용하면 LLM은 추가적인 훈련 없이도 특정 분야나 도메인에 관련된 사용자의 지시를 효과적으로 이해하고 처리할 수 있으며, 이는 흔히 "**In-context learning**" 기술로 알려져 있다.

프롬프트 엔지니어링은 종종 특정 작업에 맞춘 지시와 맥락이 포함된 프롬프트 템플릿을 설계하는 것을 포함한다. 이 템플릿은 고정된 지시와 재사용을 위해 사용자 정의 가능한 맥락 매개변수로 표준화 할 수 있다. 이 방법의 단순성과 효율성은 LLM이 낯선 주제에서도 관련성 있는 응답을 제공할 수 있도록 도움을 준다. 맞춤형 챗봇은 대화 기록을 통해 대화 맥락을 유지함으로써 상호작용을 개선하기 위해 이러한 기법들을 활용한다. 그러나 프롬프트 엔지니어링만으로는 사용자가 콘텐츠나 엔터프라이즈 데이터와 상호작용해야 할 때 애플리케이션에 필요한 기능을 충분히 제공하지 못하는 경우가 많다. 이런 경우 우리는 검색증강생성 기법으로 알려진 RAG 아키텍처를 사용하는 것을 고려할 수 있다.

Retrieval Augemnted Generation:

LangChain은 임베딩과 원문 텍스트 조각을 벡터 스토어(vector store)에 저장하여, 빠른 정보 검색을 위한 지식 데이터베이스를 생성하고 활용할 수 있다. 사용자가 자연어로 벡터 데이터베이스에 질문을 하면, LangChain Retriever가 질문을 임베딩으로 변환해 저장된 임베딩과 비교하고 가장 유사한 텍스트 조각을 검색한다. 그 후 리트리버는 가장 유사한 임베딩과 해당 텍스트 조각만 반환한다. 이렇게 로컬 지식 베이스를 활용하면 가장 관련성이 높은 텍스트만 검색되어 LLM에 전송되므로 맥락의 양(context-window)을 크게 줄일 수 있는 이점이 있다. 이는 LLM의 성능을 향상 시킬 뿐 아니라, 많은 LLM이 처리된 토큰 수에 따라 요금을 부과하므로 비용 또한 절감할 수 있다.마지막으로, RAG를 활용한 로컬 지식 베이스를 활용하면 LLM 응답이 검증된 소스에 기반을 두므로, "환각(hallucination)"이나 부정확한 응답을 최소화할 수 있다.

Fine-tuning:

파인튜닝은 사전 훈련된 LLM을 특정 작업에 특화되도록 조정하는 과정이다. 이는 모델을 특정 기능을 처리하는 방법을 가르치는 예제 데이터셋으로 훈련시키는 과정을 포함한다. 과거에는 전문 프로그래밍 라이브러리를 사용해야만 수행할 수 있지만, 이제는 데이터셋과 최소한의 설정 만으로 파인튜닝을 가능하게 하는 "제로 코드" 또는 "로우 코드" 도구가 많다.

파인튜닝의 장점은 LLM에 도메인별 지식을 부여해 전문화된 프롬프트를 반복적으로 제공할 필요를 줄일 수 있다는 것이다. 하지만 데이터셋을 준비하고 정제하는 데 시간이 많이 걸리고, 고성능 GPU를 구매하거나 대여하는 데 비용이 많이 든다는 단점도 있다 .파인튜닝의 필요성에 대해서는 여러 논란이 있다. 일부 전문가는 일반 LLM이 추가 훈련 없이도 다양한 도메인에서 효과적이라고 믿는다. 하지만 고유한 전문 용어가 필요한 특화된 분야에서는 파인튜닝이 필수적이라고 주장하는 학자들도 있다.

파인튜닝된 모델의 예로는 생물학을 위한 BioMistral, 법률을 위한 LexisNexis LLM, 금융을 위한 BloombergGPT, 프로그래밍을 위한 CodeGPT 등이 있다. Heydar Soudani, Evangelos Kanoulas, Faegheh Hasibi와 같은 연구자들은 "Fine-Tuning vs. Retrieval-Augmented Generation for Less Popular Knowledge" 논문에서 검색 증강 생성(RAG)이 파인튜닝보다 더 효과적일 수 있다고 주장하기도 한다. RAG는 프롬프트를 사용해 모델에 지식을 제공하며, 사전 훈련된 모델과 프롬프트 엔지니어링을 활용해 정확도를 높이고 비용을 줄일 수 있다. 요약하자면, 파인튜닝은 LLM을 특정 도메인에 맞춰 성능을 개선하는 역할을 한다. 그 중요성은 도메인 용어의 복잡성에 따라 다르다고 할 수 있다. 개발자는 효과적인 이해와 응답 생성에 초점을 맞춰 파인튜닝의 필요성에 대해 판단할 필요가 있다.