

# LGCNS Gen.AI 교육

LLM application 개발 입문

강사 김형욱  
([hukim@artiasolution.com](mailto:hukim@artiasolution.com))

# Contents

- LLM application 개론 ( 3 page )
- LangChain을 활용한 LLM 애플리케이션 입문 ( 26 page )
- RAG ( 36 page )
- AI 워크플로우 설계 ( 42 page )

# LLM Application 개론

# LLM Application 소개

- **LLM Application이란?**

- 특정 목적을 수행하기 위해 설계된 거대 언어 모델(LLM) 기반 애플리케이션

- 응용 사례

- Chatbot (AI 챗봇) – 24시간 고객 지원, 법률 상담, 의료 상담
- Search system – 문서 검색, FAQ 자동 응답
- Code Assistant – 코드 자동 생성 및 오류 수정 (예: GitHub Copilot)
- Content Creation – 기사, 블로그, 마케팅 카피 작성
- Autonomous Agent – 협업 에이전트 팀, 능동형 안드로이드

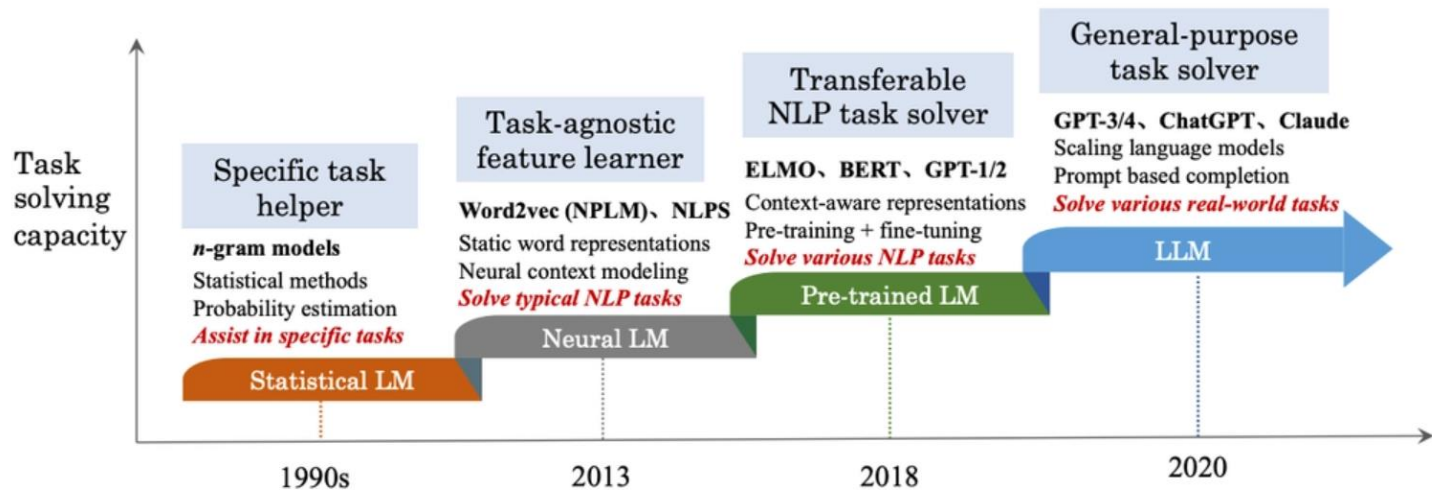
- LLM의 활용은 단순한 텍스트 생성 도구가 아니라 다양한 응용이 가능함.
- 기능별/복잡도별 다양한 활용 방식이 존재
  - 문서 요약, 데이터 추출 등과 같은 단순하지만, 노동집약적 작업에도 활용할 수 있음.
  - 반면, 자율성을 갖고 주어진 미션을 수행하는 에이전트는 높은 수준의 복잡도를 가짐.



Less complex ← ..... → More complex

# 언어모델 발전과 AI 솔루션 혁신

- 언어모델은 감정 분석, 분장 분류, 품사 Tagging 등 자연어처리를 위해 개발해왔음.
- 그러나 인터넷 자료를 방대하게 학습한 GPT-3이 일반화된 문제 해결 능력을 보임
  - 인터넷에 기록된 다양한 문제들을 거대모델이 메타학습 했기 때문
  - 모델의 구조(Transformer), 복잡도(Scaling Law), 학습 방법(Self-supervision)의 혁신이 합쳐진 결과.



# 언어모델(Language Models)의 정의

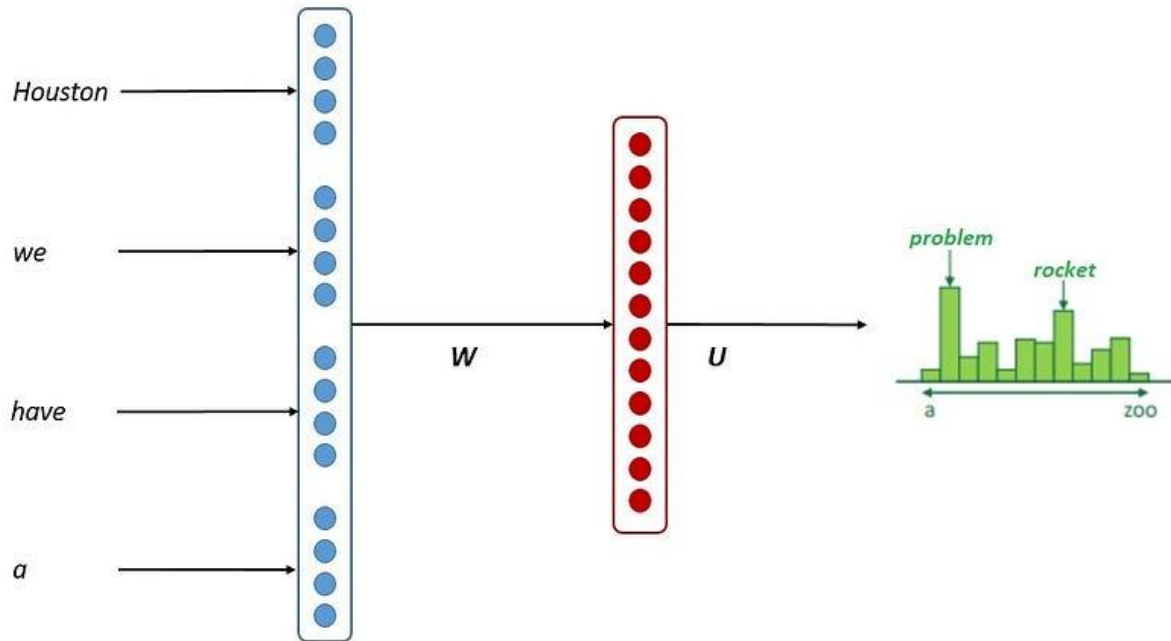
- 언어 모델(Language Model, LM)은 단어 시퀀스의 생성 확률을 모델링
  - 구체적으로, 주어진 문장에서 미래 토큰 or 누락된 토큰의 확률을 예측
    - ✓ 예를 들어, “The cat is”에 대한 확률 분포를 예측

$$f(\text{The, cat, is}) = \begin{pmatrix} P(\text{The}|\text{The, cat, is}) = .10 \\ P(\text{cat}|\text{The, cat, is}) = .03 \\ P(\text{is}|\text{The, cat, is}) = .01 \\ P(\text{blue}|\text{The, cat, is}) = .10 \\ \mathbf{P(\text{black}|\text{The, cat, is}) = .53} \\ P(\text{green}|\text{The, cat, is}) = .12 \\ P(\text{<end>}|\text{The, cat, is}) = .11 \end{pmatrix}$$

- 조건부 확률에 근거하여 미래 토큰을 결정하며, 이것이 텍스트 생성 방식임.
  - ✓ 확률은 문장이 자연스러움을 정량적으로 평가

$$(1) P(\text{Today is Wednesday}) = 0.001$$

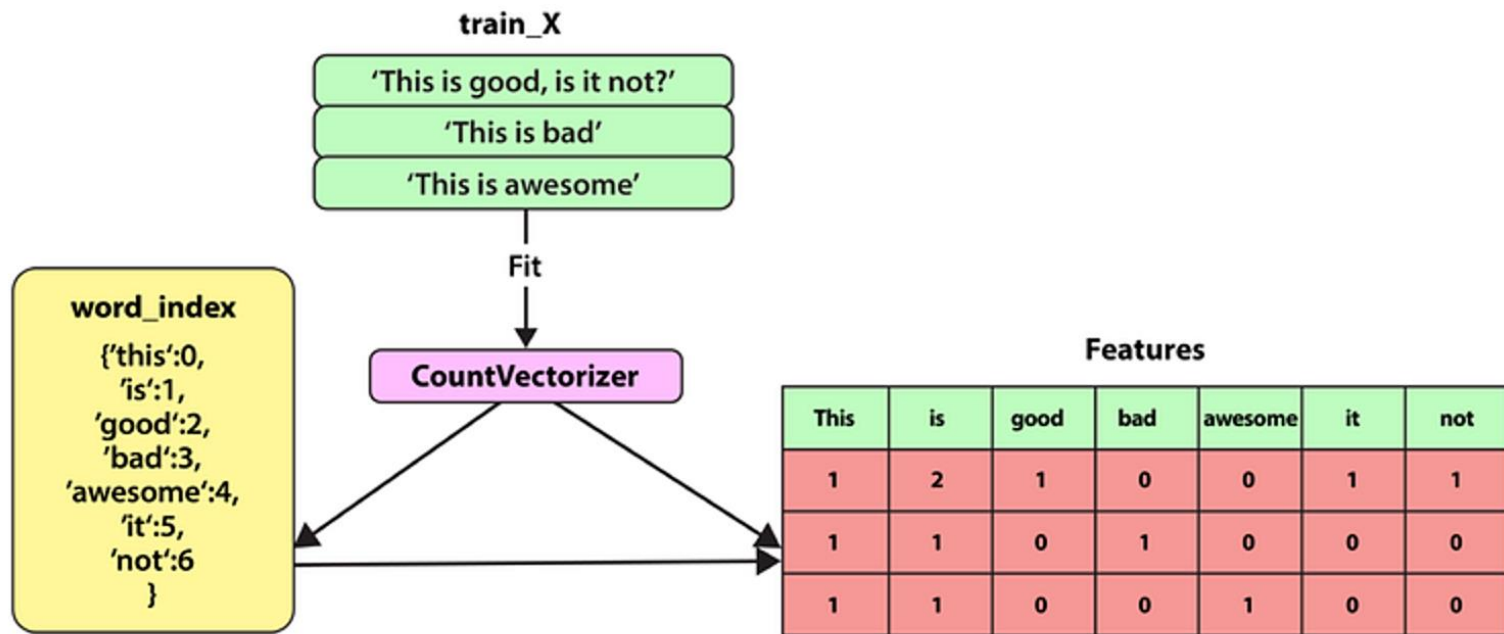
$$(2) P(\text{Today Wednesday is}) = 0.0000000001$$



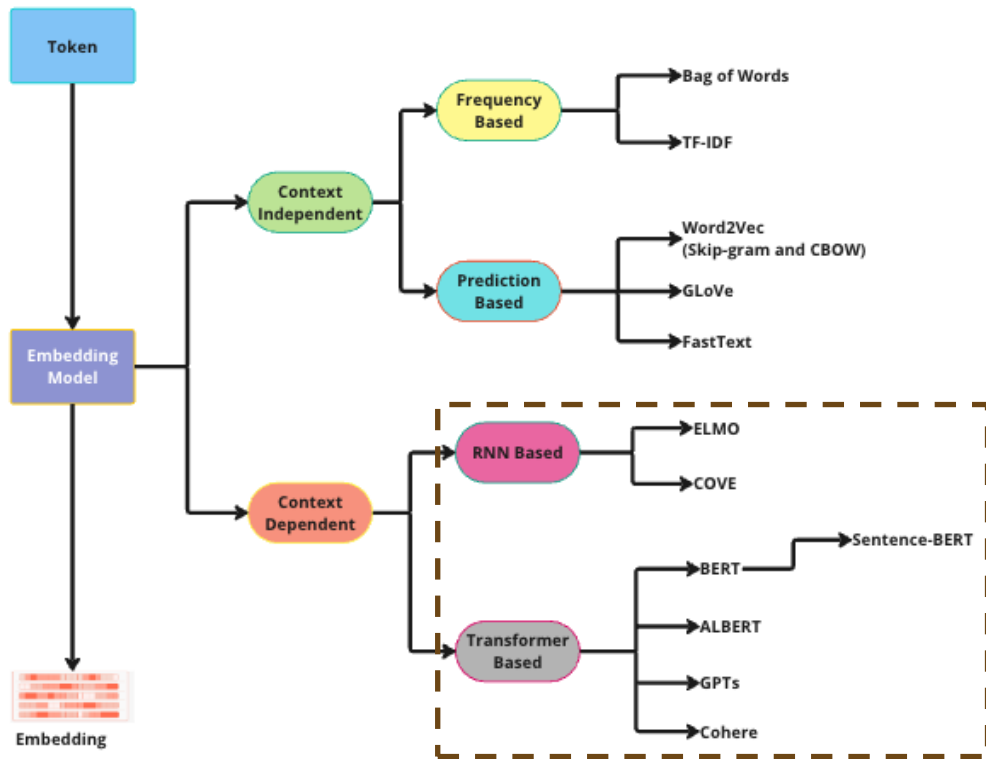
- 주어진 문장에서 미래 토큰을 예측하기 위해 머신러닝 모델 사용
  - 인간의 자연어를 기계가 처리하기 위해 데이터는 수치화되어 처리
    - ✓ (Tokenization) 문장은 단어나 음절 수준의 토큰으로 분해되어 수치화
    - ✓ (Embedding) 입력된 토큰들은 모델이 의미를 이해할 수 있는 벡터로 처리
    - ✓ (Classification) 임베딩된 벡터를 소프트맥스 분류기로 처리하여 다음 토큰의 확률 분포를 예측



# 고전적 임베딩 예시 : 단어 빈도수 기반의 BOW

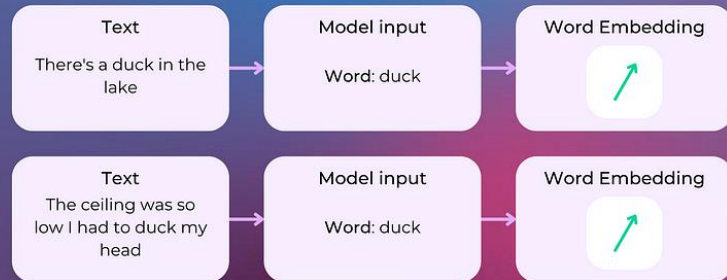


Bag-of-Words (through the CountVectorizer method) encodes the total number of times a document uses each word in the associated corpus.

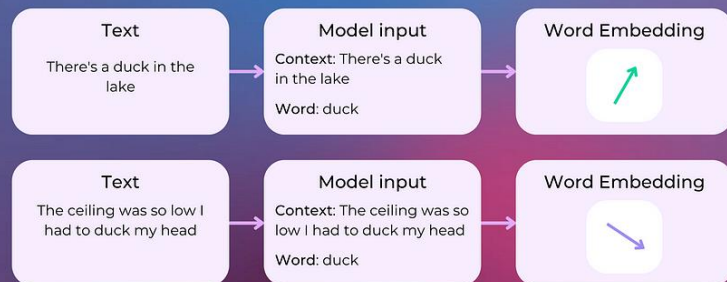


딥러닝 언어모델

## Context-Independent Word Embedding

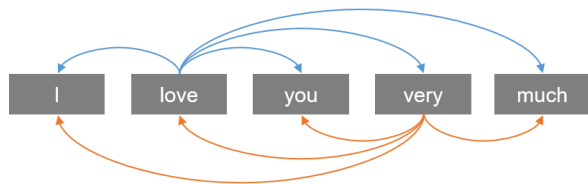


## Context-Dependent Word Embedding

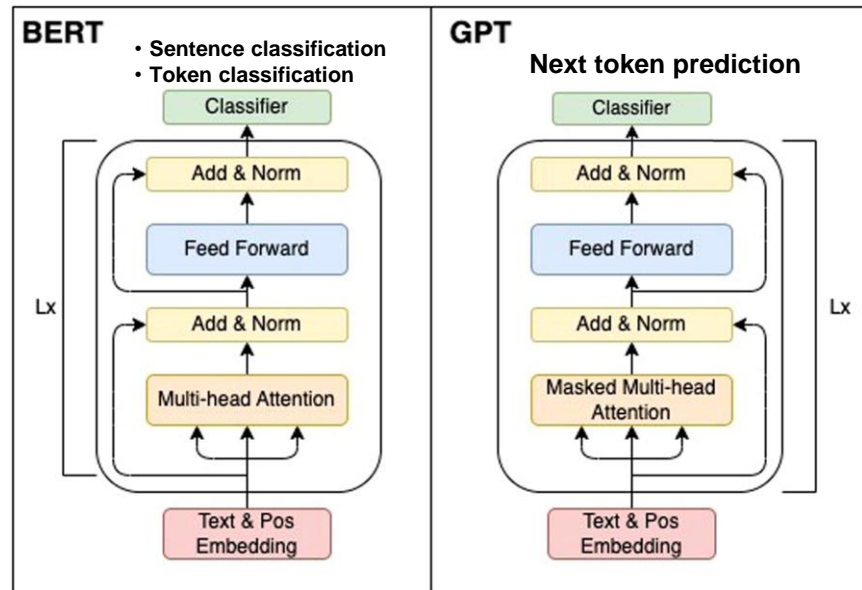


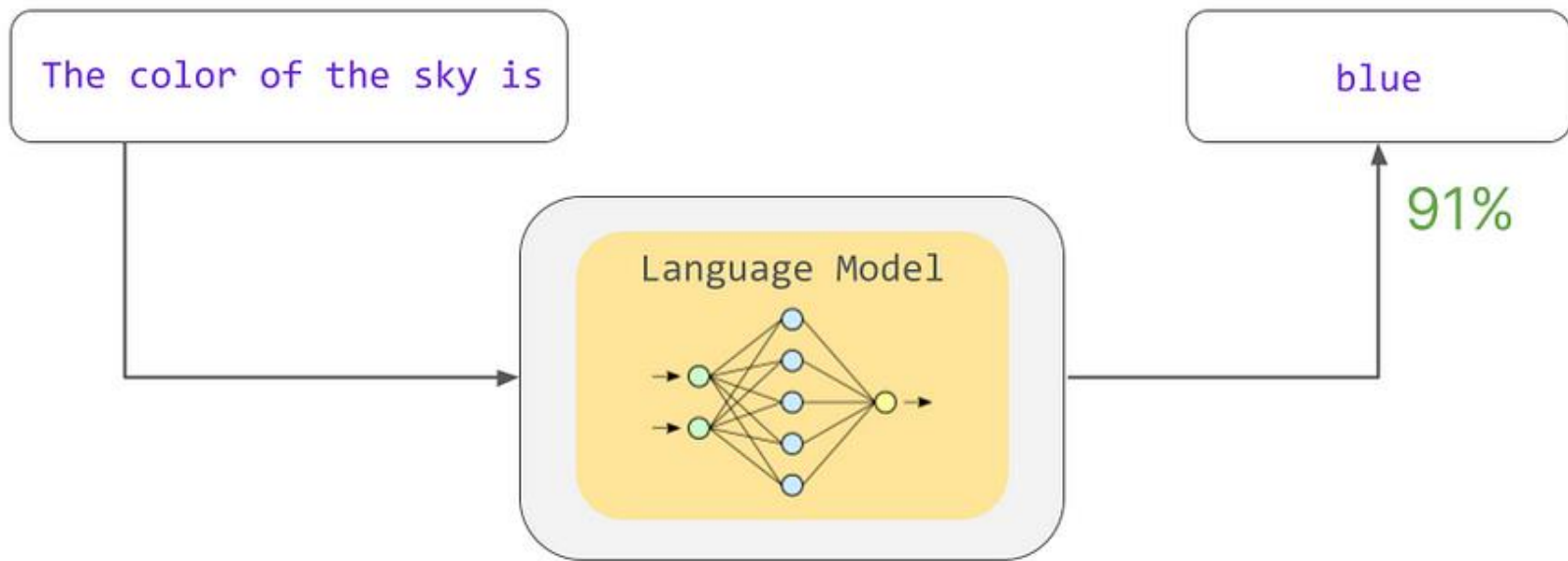
# 트랜스포머 언어모델

- 수 많은 Self Attention 계층을 쌓아 문장 내 복잡한 컨텍스트 분석



- 매우 복잡한 모델(최소 수십억 파라미터 이상)로 인간의 복잡한 언어패턴 이해
- 자기지도학습(Self-supervised learning) 방식을 적용해 human-label 없이도 방대한 자연어 데이터를 사전학습(pretraining)





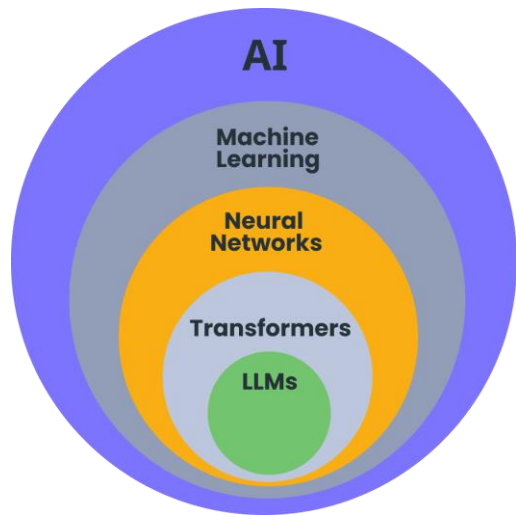
# LLM(Large Language Models)

- **LLM (Large Language Model)**

- 대량의 텍스트 데이터를 학습하여 사람과 유사한 수준의 자연어 이해 및 생성이 가능한 언어모델
- 일반적으로 100억 개 이상의 파라미터를 갖는 신경망 언어 모델

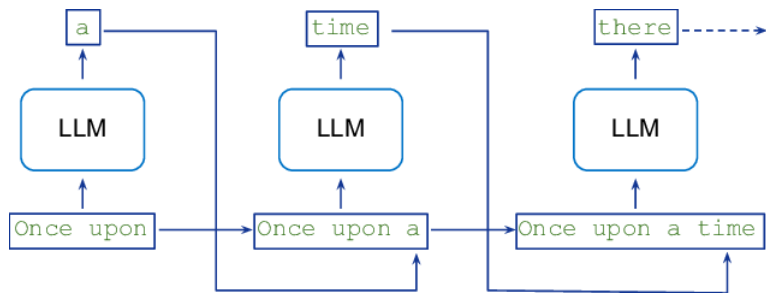
- **LLM의 핵심 특징**

- 트랜스포머 기반 : Transformer 기반의 Self-Attention 메커니즘 적용
- 대규모 데이터 사전학습 : 인터넷, 책, 논문 등 방대한 텍스트 데이터 사전학습
- 미세 조정(Fine-tuning) 및 강화학습 가능 : 특정 도메인 및 업무에 맞게 추가 학습
- 문맥 이해 : 단순한 단어 예측이 아닌 깊은 문맥적 이해 가능
- 범용성 : 챗봇, 번역, 문서 요약, 코드 생성, 검색, 분석 등 다양한 문제 해결 능력

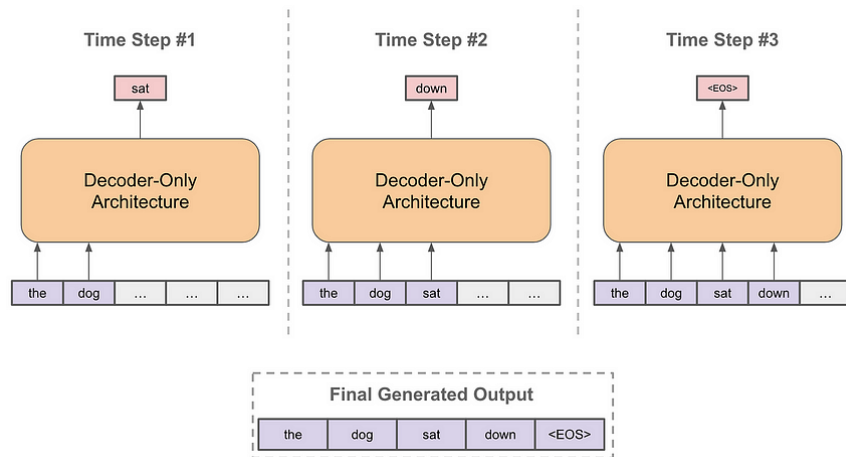


# LLM이 텍스트를 생성하는 방법

- LLM은 한 번에 문장을 완성하는 대신 다음 토큰 예측(Next Token Prediction)으로 순차적으로 문장을 만들어 감(Auto-regressive text generation)
  - 모델은 문장 시작 <SOS>에서부터, 문장 끝 <EOS>을 예측할 때까지 생성하도록 학습되어 있음.



<자기회귀적(auto-regressive) 문장 생성>



<EOS 토큰까지 순차적인 토큰 예측 과정>

# Chat LLM : 대화형으로 튜닝된 LLM

- 프롬프트에 “How to make pizza”를 LLM에 입력했다.
  - GPT의 예상 동작은...?
    - ✓ 1. 질문에 컨텍스트를 추가한다 : “for a family of six”
    - ✓ 2. 질문을 이어나간다 : “? What ingredients do I need? How much time would it take?”
    - ✓ 3. 피자 만드는 방법을 알려준다 : “To make pizza, start by preparing the dough. In a small bowl...”
  - 기본 LLM은 1,2 번 방식으로 응답하는 경향이 있음
    - ✓ 자연스러운 문장을 생성하는 것을 배웠을 뿐, 대화 형식을 배운게 아니기 때문.
    - ✓ 지시사항을 따르도록 하거나, 질문에 응답하도록 미세조정을 거친 모델이 Chat LLM.
    - ✓ LLM 애플리케이션 개발 시 대화형 모델의 수요가 많기 때문에 주로 Chat LLM을 활용함.

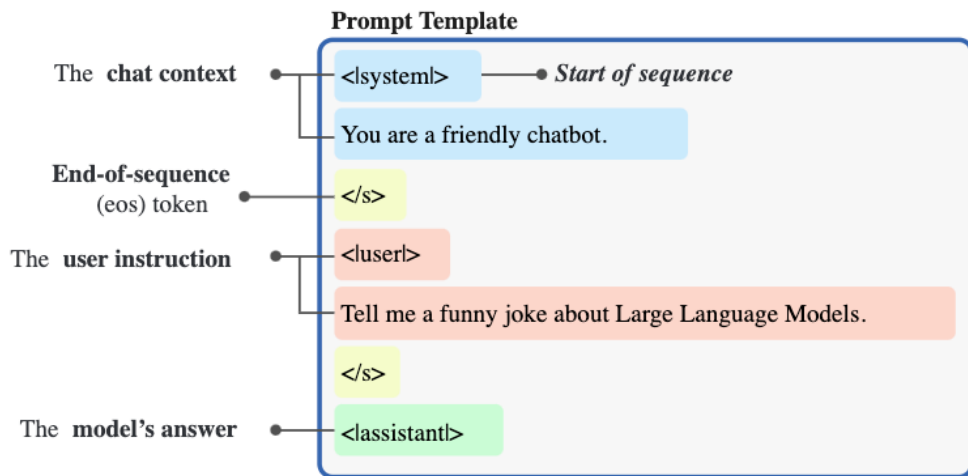
# 다중회차 대화(multi-turn conversation)

```
[  
  1 { "role": "system", "content": "\n당신은 은행 상담원입니다. 고객과 은행 업무에 대해 여러 번의 대화를  
    진행합니다.\n대화는 주어진 텍스트 정보에 기반해서 질의와 응답으로 이루어집니다.\n대출 상품에 대한 상세한 정보를  
    참고하여 고객의 질문에 정확히 답변하세요.\n" },  
  2 { "role": "user", "content": "내집마련 디딤돌대출을 신청할 때, 변동금리와 고정금리의 차이점은 무엇인가요?" },  
  3 { "role": "assistant", "content": "내집마련 디딤돌대출을 신청할 때, 변동금리와 고정금리의 차이점은 다음과  
    같습니다:\n\n- **고정금리**는 대출 실행 시 결정한 금리가 대출거래약정서에서 정한 기간 동안 동일하게 적용되는  
    금리입니다. 즉, 대출 기간 동안 금리가 변하지 않으므로 이자 부담이 일정하게 유지됩니다.\n\n- **변동금리**는 대출  
    약정 기간 내에 국토교통부 고시 기준 금리가 변경될 경우 해당 대출 금리가 변경되는 금리입니다. 따라서 기준금리가 인상될  
    경우 고객의 이자 부담이 증가할 수 있습니다.\n\n이 정보는 대출이자율 항목에서 찾았습니다." },  
  4 { "role": "user", "content": "변동금리로 내집마련 디딤돌대출을 선택할 경우, 금리 변경 주기는 어떻게 되나요?" },  
]
```



# Chat template

- Chat LLM은 발화자 구분을 위해 채팅에 맞는 템플릿의 형식으로 데이터를 학습함.
  - 반드시 추론 시에도 모델이 학습한 형태의 프롬프트 템플릿으로 전처리하여 입력해야 한다.
  - 프롬프트 템플릿은 모델 개발자가 모델과 함께 문서로 설명하거나, 데이터로 제공함.



The prompt template is automatically generated with the internal prompt template. Notice that there are different tags for differentiating between the user and the assistant.

- 다중회차 발화는 좌측과 같이 <|system|>, <|user|> 등과 같은 특수 문자를 사용해 구조화된 하나의 텍스트(프롬프트)로 합쳐진다.
- 모델은 Tokenization, Embedding을 거쳐 텍스트 생성을 통해 <|assistant|>의 답변 부분을 생성한다.
- 챗에서는 답변에 해당하는 문자열만 추출해 보여줌.

# 다중회차 대화(multi-turn conversation) 답변 추가

[

1

{ "role": "system", "content": "\n당신은 은행 상담원입니다. 고객과 은행 업무에 대해 여러 번의 대화를 진행합니다.\n대화는 주어진 텍스트 정보에 기반해서 질의와 응답으로 이루어집니다.\n대출 상품에 대한 상세한 정보를 참고하여 고객의 질문에 정확히 답변하세요.\n" },

2

{ "role": "user", "content": "내집마련 디딤돌대출을 신청할 때, 변동금리와 고정금리의 차이점은 무엇인가요?" },

3

{ "role": "assistant", "content": "내집마련 디딤돌대출을 신청할 때, 변동금리와 고정금리의 차이점은 다음과 같습니다:\n\n- \*\*고정금리\*\*는 대출 실행 시 결정한 금리가 대출거래약정서에서 정한 기간 동안 동일하게 적용되는 금리입니다. 즉, 대출 기간 동안 금리가 변하지 않으므로 이자 부담이 일정하게 유지됩니다.\n\n- \*\*변동금리\*\*는 대출 약정 기간 내에 국토교통부 고시 기준 금리가 변경될 경우 해당 대출 금리가 변경되는 금리입니다. 따라서 기준금리가 인상될 경우 고객의 이자 부담이 증가할 수 있습니다.\n\n이 정보는 대출이자율 항목에서 찾았습니다." },

4

{ "role": "user", "content": "변동금리로 내집마련 디딤돌대출을 선택할 경우, 금리 변경 주기는 어떻게 되나요?" },

5

{ "role": "assistant", "content": "변동금리로 내집마련 디딤돌대출을 선택할 경우, 금리 변경 주기는 대출 약정 기간 내에 국토교통부 고시 기준 금리가 변경될 때마다 변경됩니다. \n\n이 정보는 \"2. 대출이자율\" 항목에서 찾았습니다." }

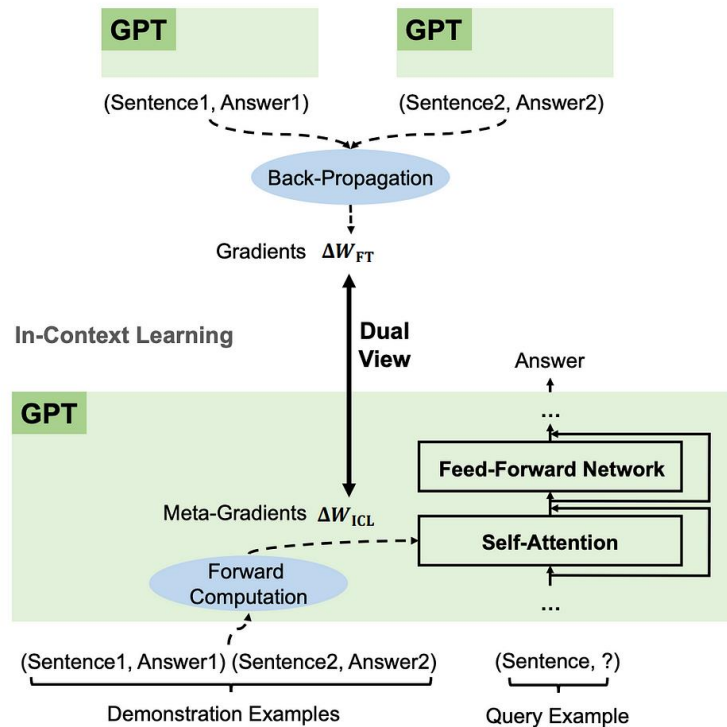
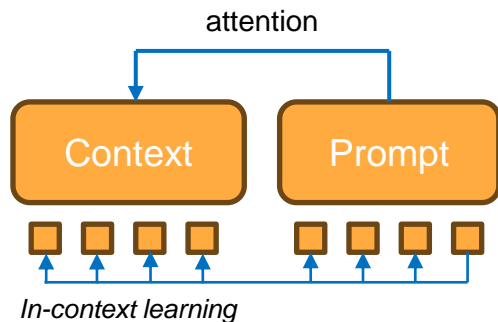
]

# LLM의 혁신: 컨텍스트 학습과 추론 능력

- 컨텍스트 학습 (Long-context Learning)
  - 긴 문장 및 문서 이해 가능 – 10K~100K 토큰 지원 (GPT-4 Turbo, Claude 2)
  - 이전 대화 맥락 유지 – 다중턴 대화에서 일관된 응답 제공
  - Few-shot Learning 활용 – 적은 예시만으로 새로운 작업 수행
- 논리적 추론 (Reasoning & Chain-of-Thought)
  - Chain-of-Thought (CoT) 기법 활용 – 단계별 사고 과정으로 복잡한 문제 해결
  - 수학 문제 풀이, 논리적 사고 적용 가능

# 컨텍스트 학습(Context learning)

- 트랜스포머는 기존 모델과 달리 가중치를 동적 계산
  - 예를 들어, CNN은 추론 상황에서 입력 이미지에 따라 필터 가중치가 달라지지 않음.
  - 반면 LLM은 학습 과정 중 자연스럽게 컨텍스트 활용 방법을 학습하고, 추론 시에도 활용 가능
    - ✓ 처음 접하는 문제도 맥락이 주어지면 해결 가능



# LLM의 논리적 추론

- 주어진 정보에서 합리적인 결론을 도출하는 과정
- LLM은 확률적 언어 모델이지만, 일정 수준의 논리적 패턴을 학습하여 추론이 가능
- LLM의 작동을 인간의 사고 방식과 유사하게 논리적 전개를 수행하도록 유도 가능함

## Standard Prompting

### Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

### Model Output

A: The answer is 27. ❌

## Chain-of-Thought Prompting

### Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls.  $5 + 6 = 11$ . The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

### Model Output

A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had  $23 - 20 = 3$ . They bought 6 more apples, so they have  $3 + 6 = 9$ . The answer is 9. ✅

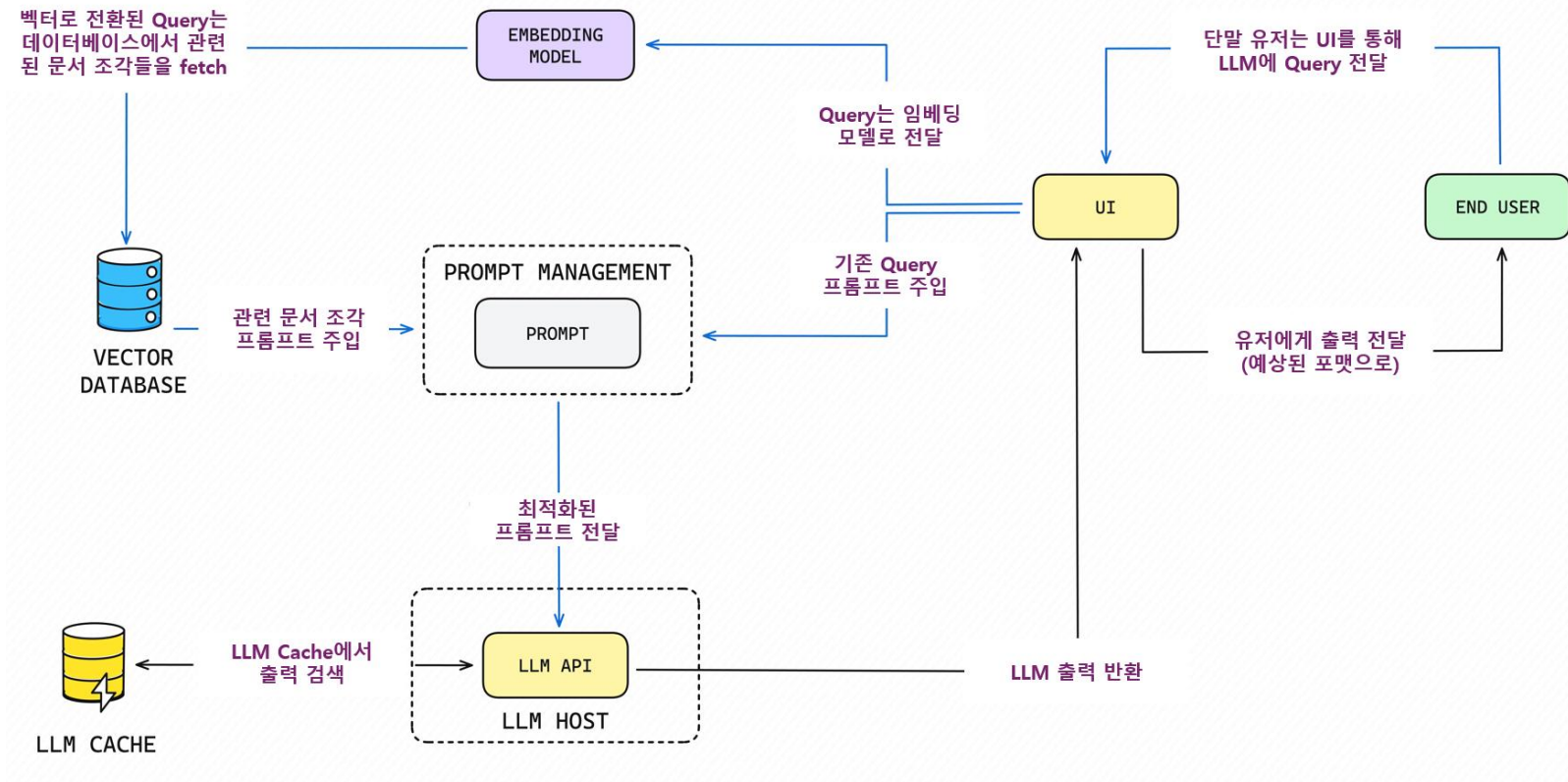
# LLM 기반 애플리케이션 설계 시 고려할 점

- 모델 선택 (상용 LLM vs. 오픈소스 LLM)
  - LLM API는 간편하지만, 제공자가 SLA(Service Level Agreement, 서비스 수준 계약)를 제공하지 않는 경우가 많음.
    - ✓ API 기반으로 빠르게 프로토타입을 만들되, 장기적으로는 자체 LLM 구축 고려 필요
- 프롬프트 엔지니어링이 핵심
  - 동일 LLM이더라도 프롬프트의 최적화에 따라 성능이 크게 달라짐.
  - 복잡한 로직을 적용할 시 오류가 누적될 가능성 높음
    - ✓ LLM 체인 길이를 최소화하고, 평가 메커니즘 도입
- 보안 & 신뢰성
  - 응답의 신뢰도 (Fact-checking, Hallucination 문제 해결)
    - ✓ 신뢰도 평가를 위한 체계적인 검증 방법을 고민해볼 필요가 있음
  - 개인정보 보호 및 데이터 보안
    - ✓ Watcher LLM(출력 감시) 도입이 필요할 수 있음.

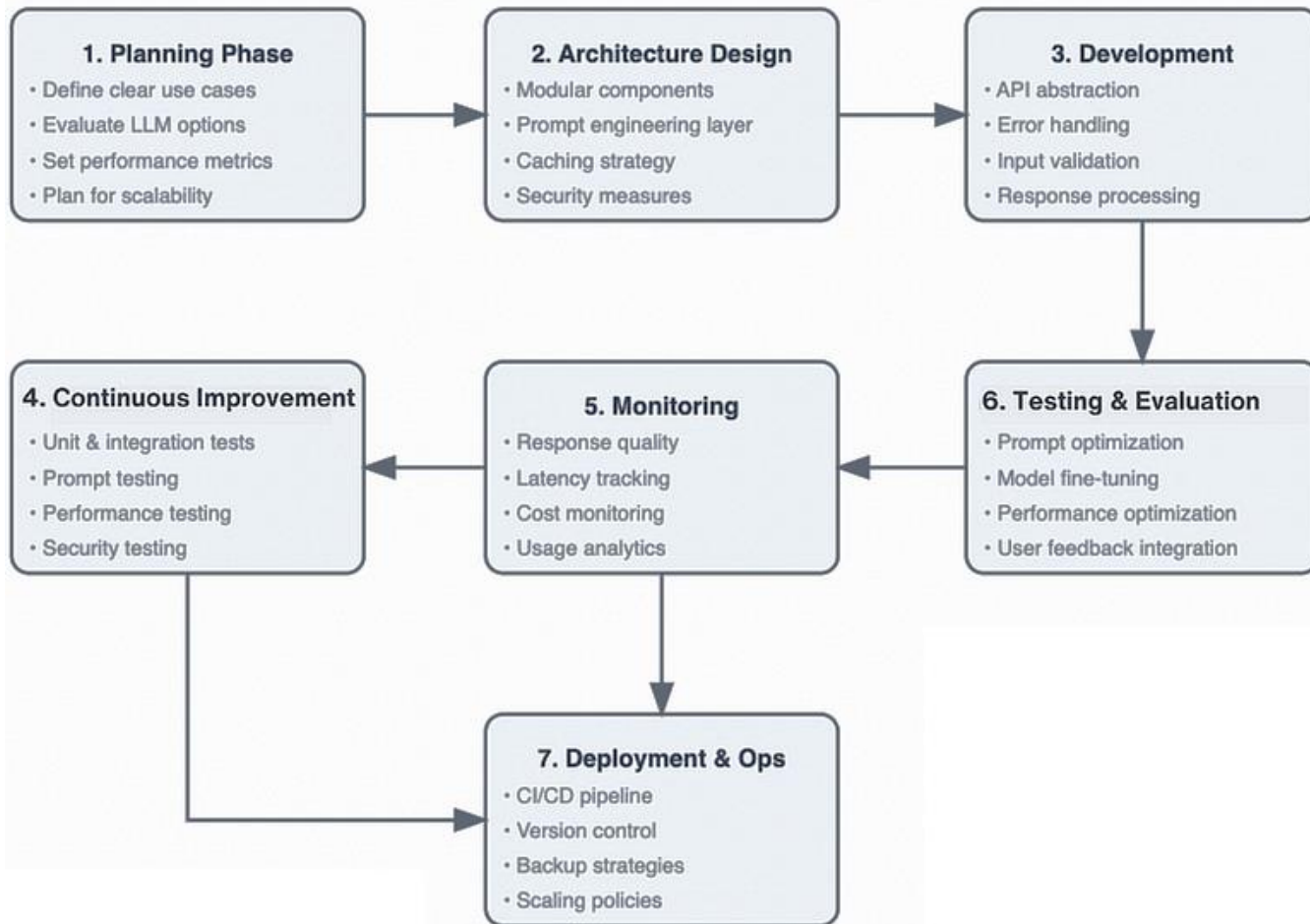
# LLM Application의 4단계 프로세스

- 입력 (Prompting & Retrieval)
  - 사용자 입력 (자연어 질문, 명령어, 문서)
  - 검색 기반 증강 (RAG) – 벡터DB(FAISS, Chroma) 활용
  - 프롬프트 엔지니어링 – Zero-shot, Few-shot 적용
- LLM 처리 (Generation & Reasoning)
  - 자연어 생성 (NLG)
  - 논리적 추론 (Reasoning) – CoT, ReAct 등 적용
- 후처리 (Post-processing)
  - 응답 필터링 – 신뢰성 확보, 안전성 검토
  - 포맷 변환 – JSON, XML 등 구조화된 출력
  - 리랭킹(Re-ranking) – 가중치 기반 최적 응답 선택
- 시스템 연계 (API & Memory)
  - 외부 API 호출 (검색 엔진, 금융 데이터, ERP 시스템)
  - 메모리 (단기 메모리, 장기 메모리)

# LLM Application architecture







# LangChain을 활용한 LLM 애플리케이션 입문

# LangChain: LLM Application 개발 프레임워크

- LangChain은 LLM 기반 애플리케이션을 쉽게 구축할 수 있도록 지원하는 프레임워크
  - LLM과 외부 데이터, API, 메모리, 검색 시스템을 결합하여 복잡한 AI 응용 프로그램 개발 가능
- LangChain의 핵심 역할
  - LLM과 데이터 소스(문서, 데이터베이스) 연동
  - 사용자 입력에 대한 프롬프트 제어 및 최적화
  - 응답을 가공하여 원하는 형식(JSON, 텍스트 등)으로 출력
  - 메모리를 활용하여 문맥을 유지하는 대화형 AI 구축

# LangChain Runnables

- LangChain의 컴포넌트들은 개별로 실행될 수 있도록 모듈화되어 있음.
  - LangChain에서는 개별 컴포넌트를 LangChain Runnable으로 지칭
  - 여러 개의 컴포넌트를 사슬처럼 하나로 연결하여 한 번에 실행할 수 있으며, 실행 단위를 Chain으로 지칭
- **핵심 컴포넌트**
  - Model (LLM 연결 및 실행) – OpenAI, Llama, Mistral 등 다양한 모델 지원
  - Prompt (프롬프트 엔지니어링 및 템플릿 관리) – 효과적인 입력 생성
  - 출력 파서 (Output Parser) – LLM의 출력을 원하는 형식으로 변환

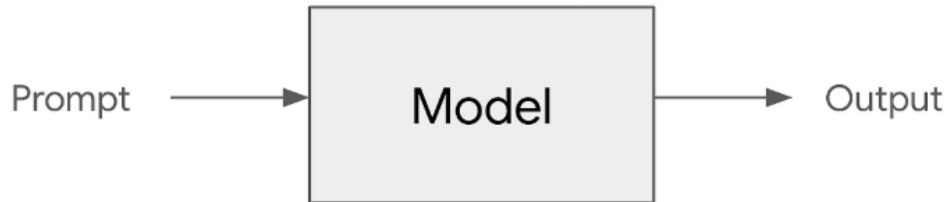
# LangChain에 LLM 모델 연결

- **상용 API 연결**

- OpenAI (GPT-4, GPT-4 Turbo)
- Google Gemini API
- ....

- **오픈소스 LLM 연결**

- Llama (Meta)
- Mistral, Falcon, Dolly 등
- QWEN
- ...



```
llm = ChatOpenAI(  
    temperature=0.1, # 창의성 (0.0 ~ 2.0)  
    model_name="gpt-4o", # 모델명  
)
```

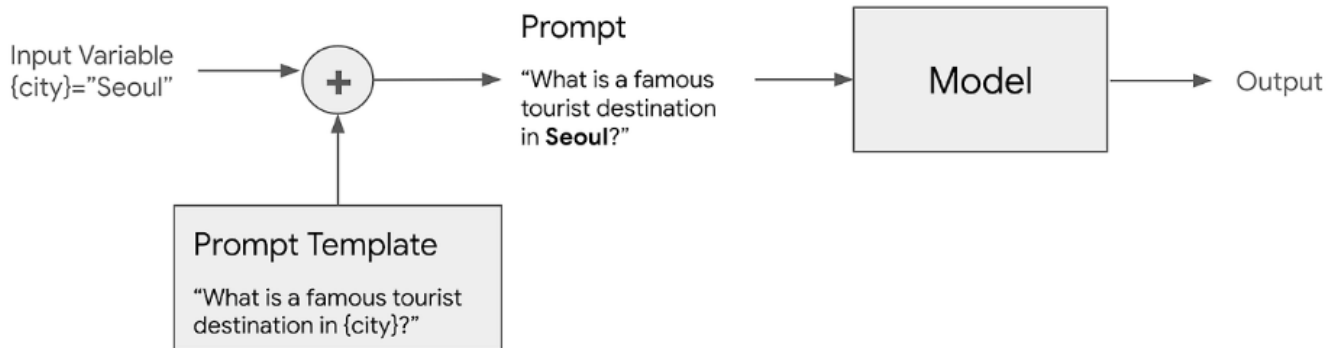
# 프롬프트 최적화하기

- PromptTemplate 활용
  - 변수 기반 동적 프롬프트 생성
  - 다양한 입력값을 조합하여 응답 최적화

```
from langchain.prompts import PromptTemplate
```

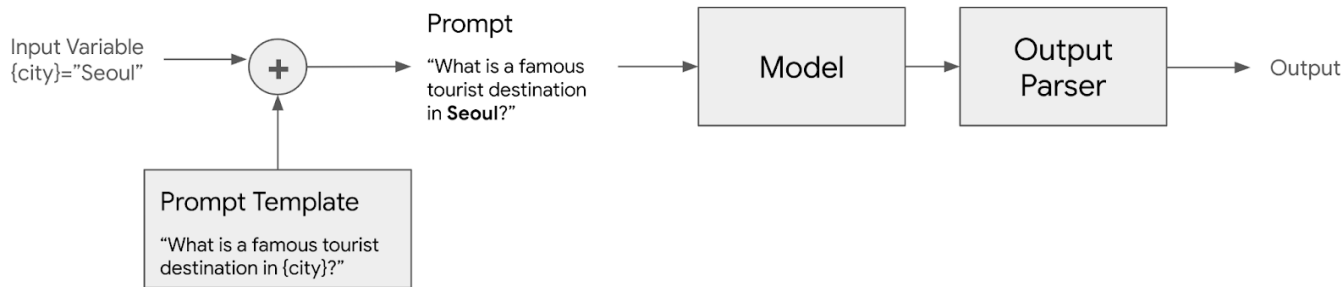
```
your_template = PromptTemplate.from_template(  
    "주어진 문장에 대해 설명하세요: {text}"
```

```
prompt = your_template.invoke({"text"}="LangChain은  
LLM을 쉽게 활용할 수 있는 프레임워크입니다.")
```



# OutputParser로 출력제어하기

- LangChain은 LLM의 출력을 구조화된 형식으로 변환 가능
- 주요 기능
  - JSON 형식 출력 – API 및 데이터 처리 연계
  - 키워드 추출 – 문장에서 핵심 개념 추출
  - 테이블 및 리스트 변환 – 응답을 구조화된 형태로 저장



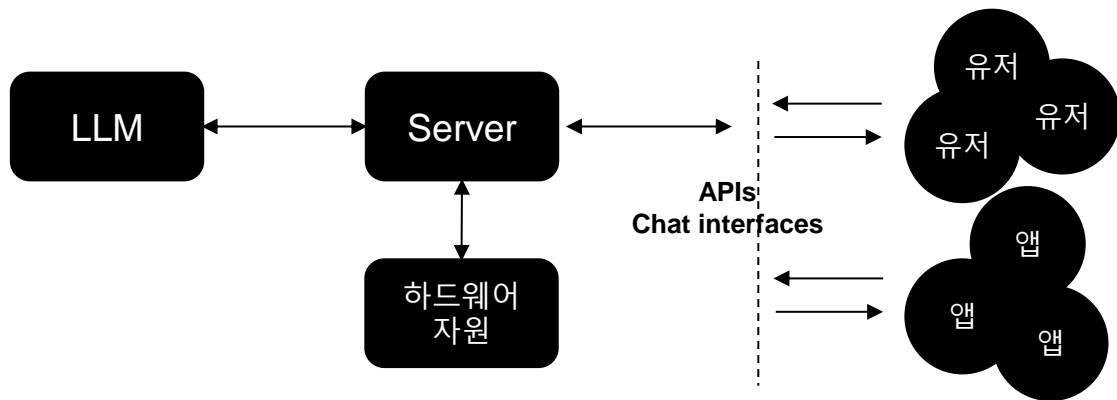
# LangChain으로 LLM Application 구축하기

- LangChain을 활용하면 LLM과 외부 시스템을 쉽게 연결 가능
- 프롬프트 엔지니어링과 Output Parsing으로 응답 최적화
- 다양한 응용 사례 (검색, 챗봇, 문서 요약 등)에 적용 가능

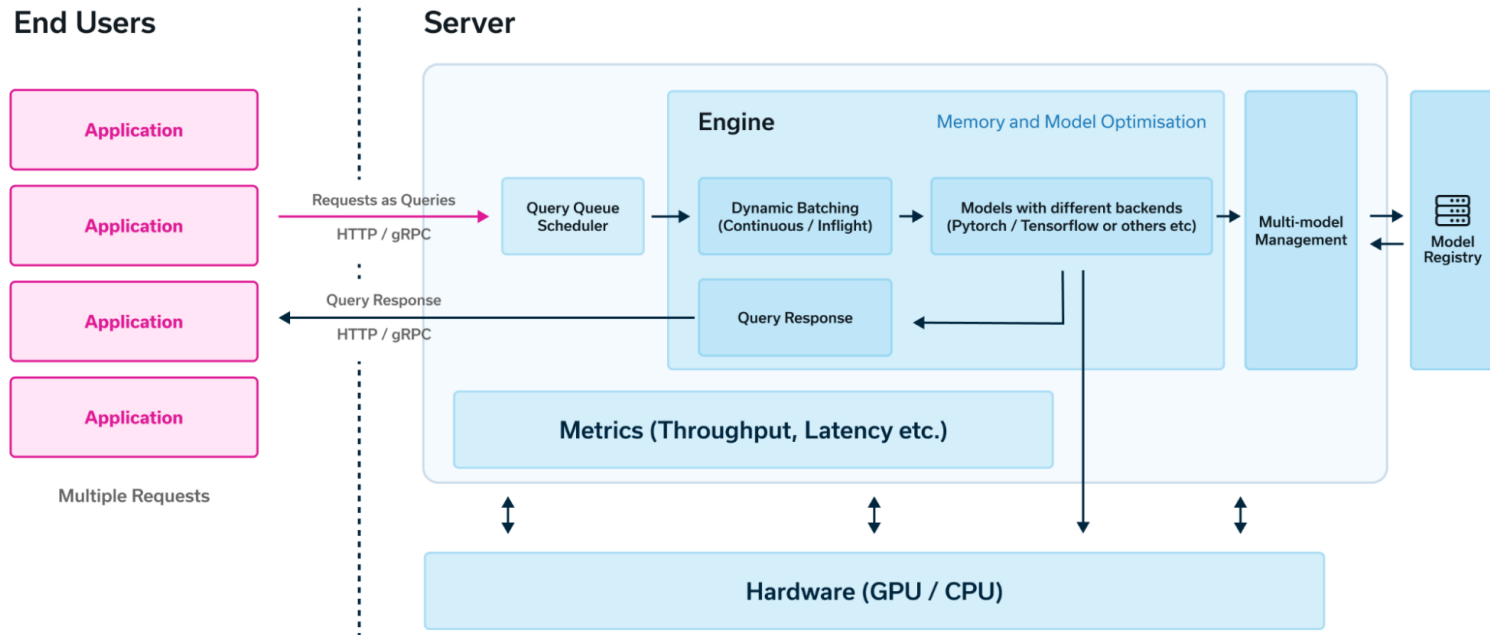


# LLM serving framework

- LLM 추론을 효율적으로 수행하기 위해, 트랜스포머와 텍스트 생성 알고리즘에 최적화된 연산 방식을 제공하는 서빙 프레임워크들이 있음.
  - 목적에 따라 두 분류로 나뉘어짐.
    - ✓ 소형화된 LLM을 개인용 로컬 환경에서 작동하기 위한 Ollama, llama.cpp 등
    - ✓ 고성능 컴퓨팅 환경을 최대한 활용하기 위한 vLLM, TGI, Ray Serve 등
- API 기능을 제공해 어플리케이션이 효율적으로 LLM과 상호작용하도록 지원



# LLM serving framework 구조



# LangChain과 오픈소스 LLM 연동

- LangChain은 다양한 서빙 프레임워크에 업로드된 모델과 쉽게 상호작용할 수 있는 인터페이스를 제공하고 있음.
  - 서빙 프레임워크의 API를 LangChain 인터페이스와 연결하여 사용하는 방식
  - 허깅페이스 모델을 연결할 수도 있지만 연산 속도가 매우 느리므로 유의 필요

# **RAG**

## **Retrieval-Augmented Generation**

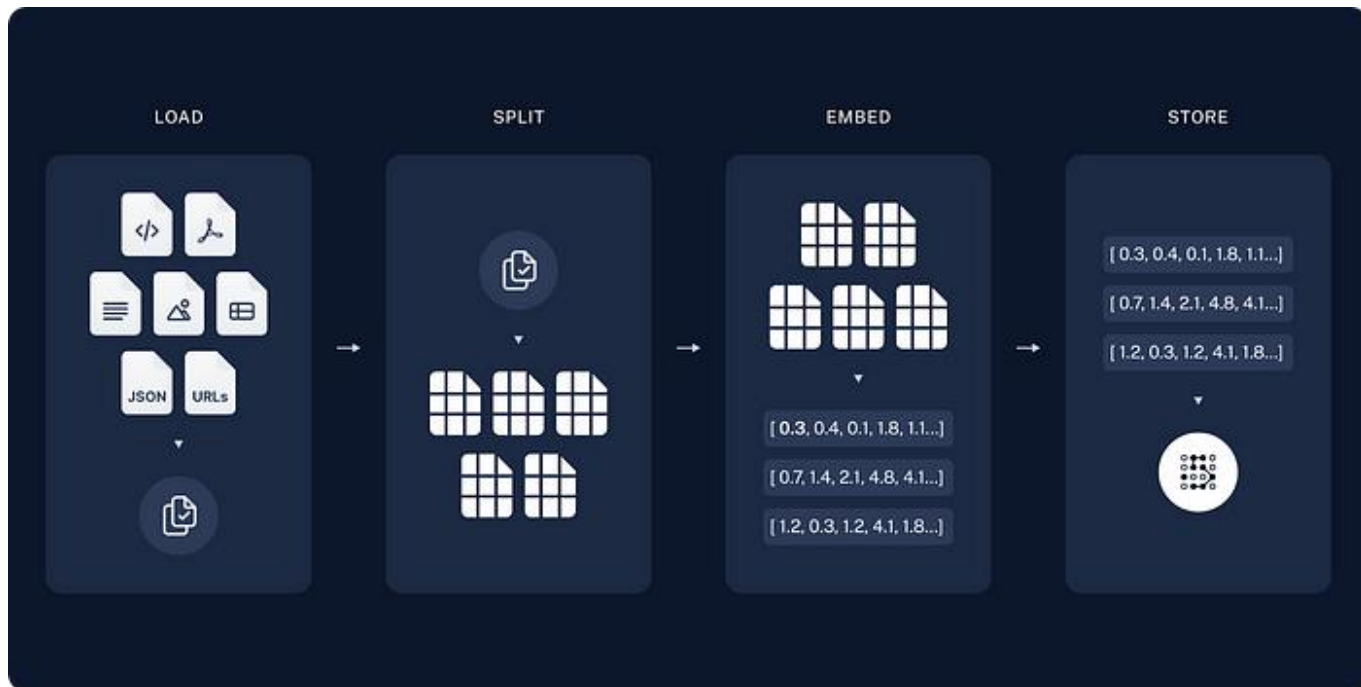
# RAG (Retrieval-Augmented Generation)란?

- LLM의 한계를 극복하기 위해 외부 정보를 검색(Retrieval)하여 응답을 생성하는 기법
  - LLM 단독 응답이 아닌, 검색 시스템과 결합하여 신뢰도 높은 정보를 제공
- RAG의 핵심 아이디어
  - LLM이 학습한 정적 데이터에 의존하지 않고, **최신 정보 및 외부 문서를 검색**하여 응답 생성
  - 웹 문서, PDF, 데이터베이스 등 다양한 소스를 활용 가능

# RAG가 필요한 이유

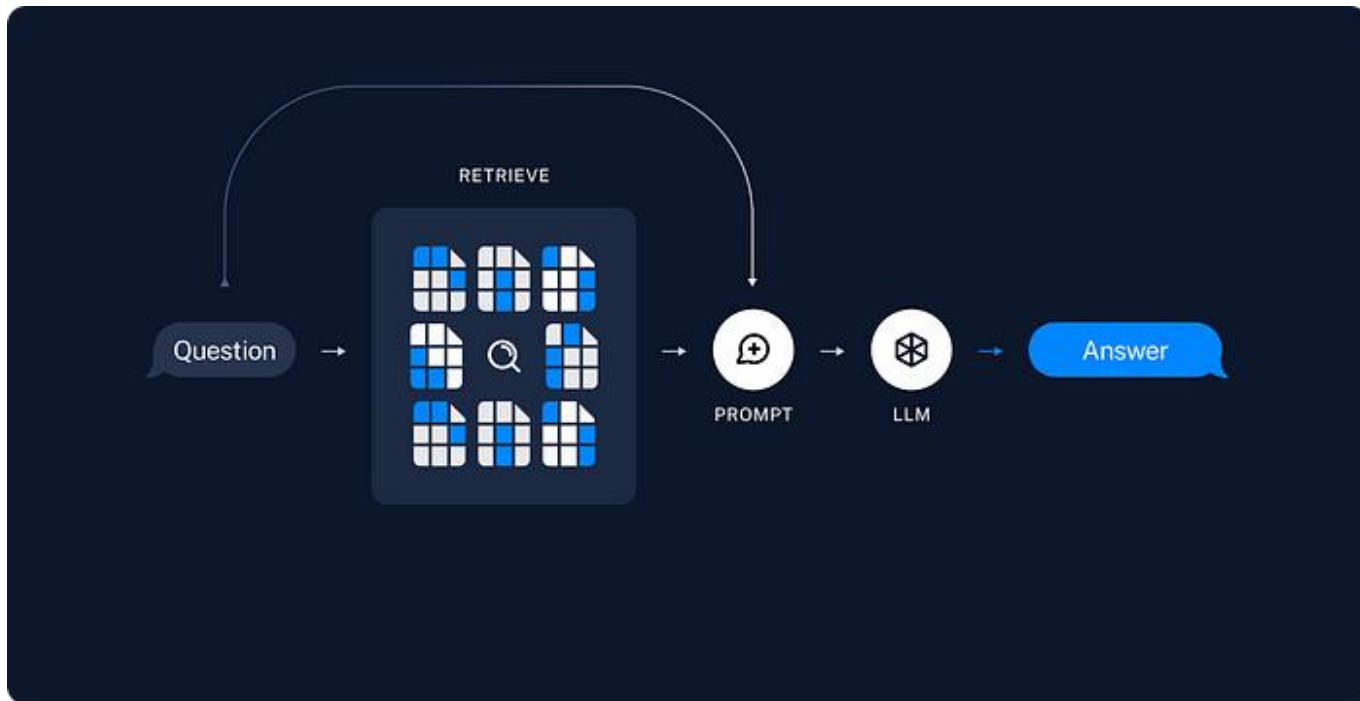
- LLM의 한계
  - 학습된 데이터까지만 정보를 제공 (최신 정보 부족)
  - 사실과 다른 응답 생성 가능 (Hallucination 문제)
  - 참고할 긴 문서에서 필요한 정보만 뽑아내 활용하기 어려움
- RAG를 활용하면?
  - 최신 데이터 검색 가능 (실시간 정보 반영)
  - LLM의 신뢰성 향상 (검색된 근거 기반 응답).
    - ✓ 프롬프트에 “RAG 문서에서 찾은 컨텍스트를 활용해 답변합니다. 적합한 정보가 없을 경우, 지어내지 말고 관련 정보를 찾지 못 했다고 답변해주세요.”를 추가
  - 검색을 통해 긴 문서에서 핵심 정보 추출 가능

# RAG 준비 프로세스



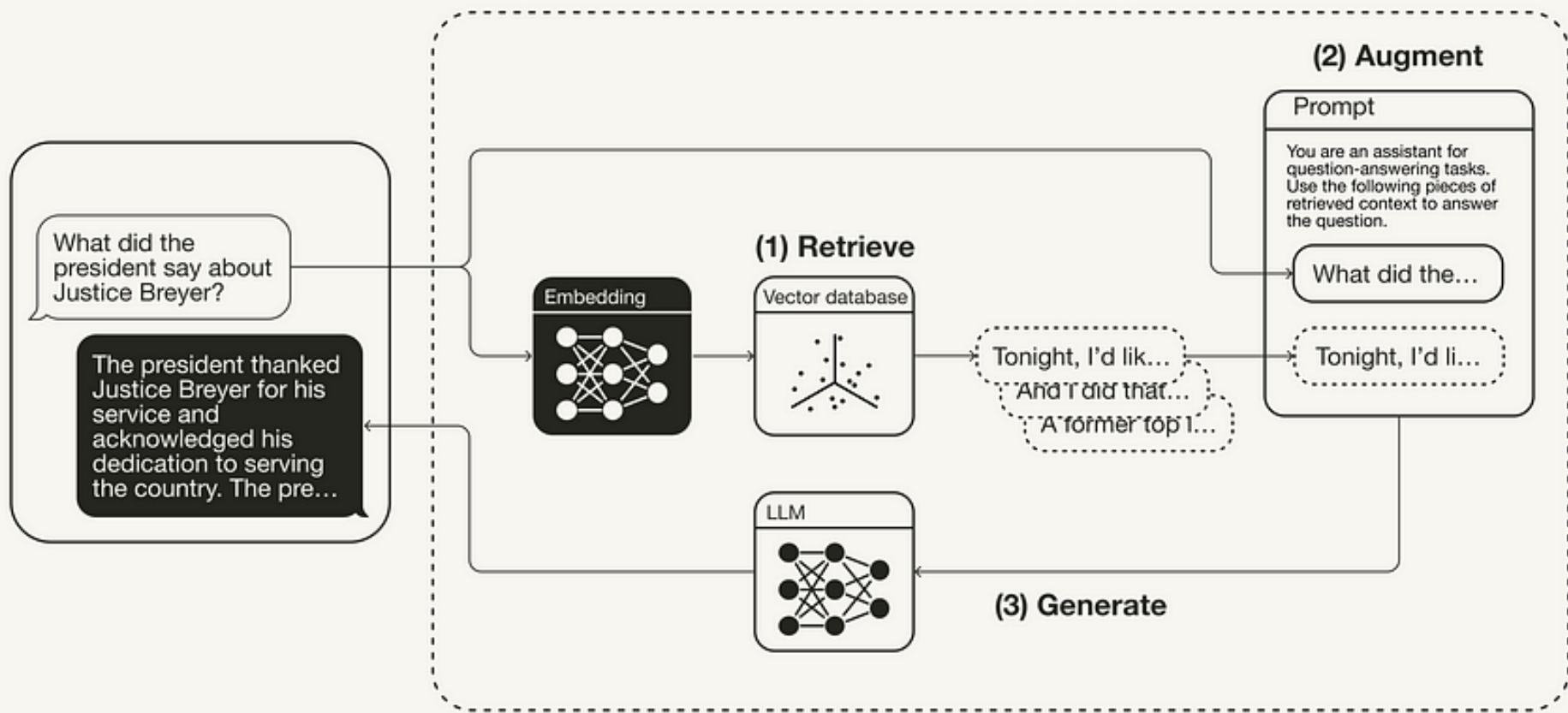
- 문서 분할 (SPLIT) – 문서를 작은 단위(Chunk)로 나누어 검색 최적화
- 임베딩 변환 (Embedding) – 텍스트를 벡터화하여 검색 가능하게 변환
- 벡터 저장소 (Vector Store) – FAISS, ChromaDB 등에 벡터화된 문서 저장

# RAG 실행 프로세스



- 유사도 검색 (Similarity Search) – 입력 질의와 가장 유사한 문서 검색
- LLM 결합 (Generation) – 검색된 문서를 프롬프트에 결합해 LLM 응답 생성

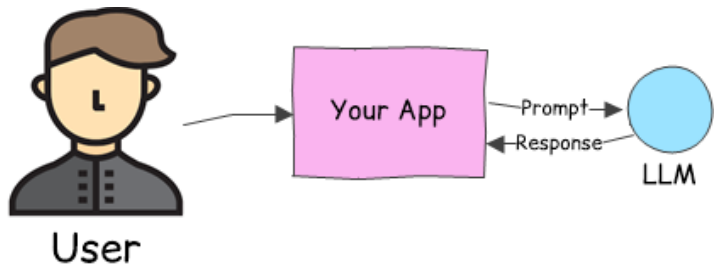




# AI 워크플로우 설계

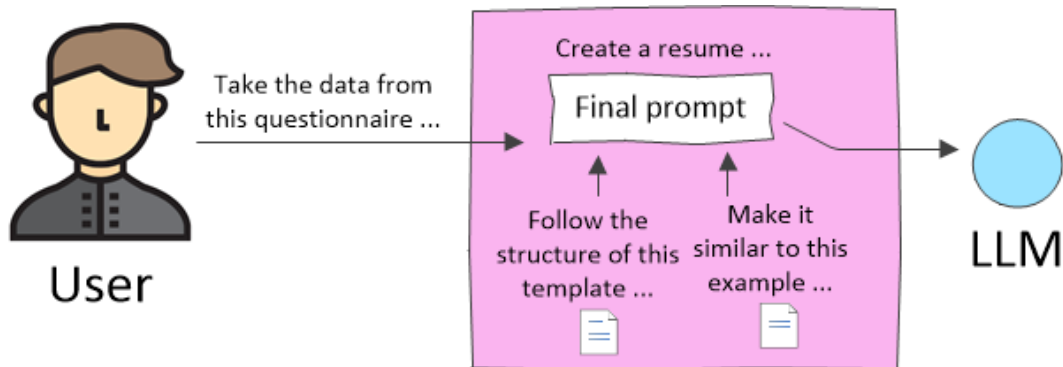
# AI 워크플로우란?

- AI 애플리케이션이 응답을 처리하는 구조 의미
  - AI 기반 애플리케이션은 입력 → 처리 → 응답의 단계로 구성
  - 단순 LLM 호출이 아니라 다양한 데이터 소스와 API를 결합하여 응답을 최적화
- AI 워크플로우의 주요 요소
  - 데이터 흐름 제어 – 사용자 입력을 처리하고 적절한 응답을 생성
  - 다중 스텝 태스크 처리 – 여러 개의 AI 모델, API, 데이터 연계를 통한 복합 워크플로우
  - 상태(State) 관리 – 대화 이력, 사용자 정보 저장



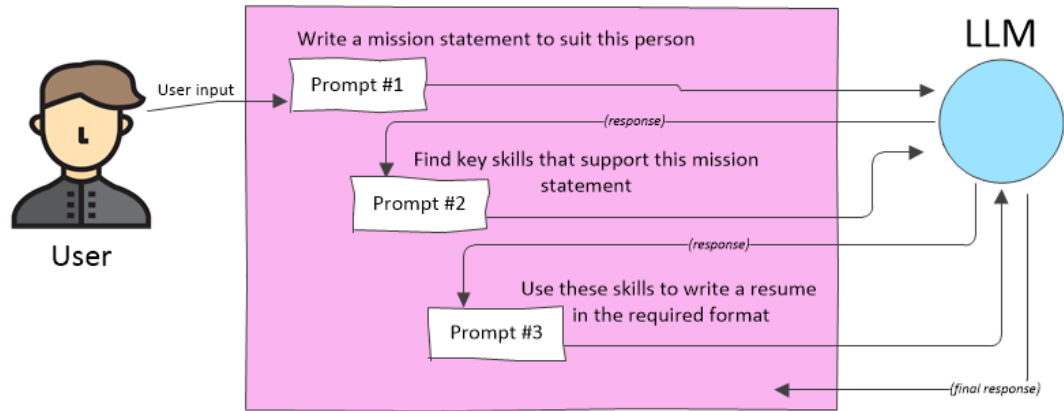
- LLM 프록시 방식

- 가장 단순한 형태의 LLM 애플리케이션
- 사용자가 입력한 요청을 LLM에 그대로 전달
- LLM의 응답을 그대로 사용자에게 반환



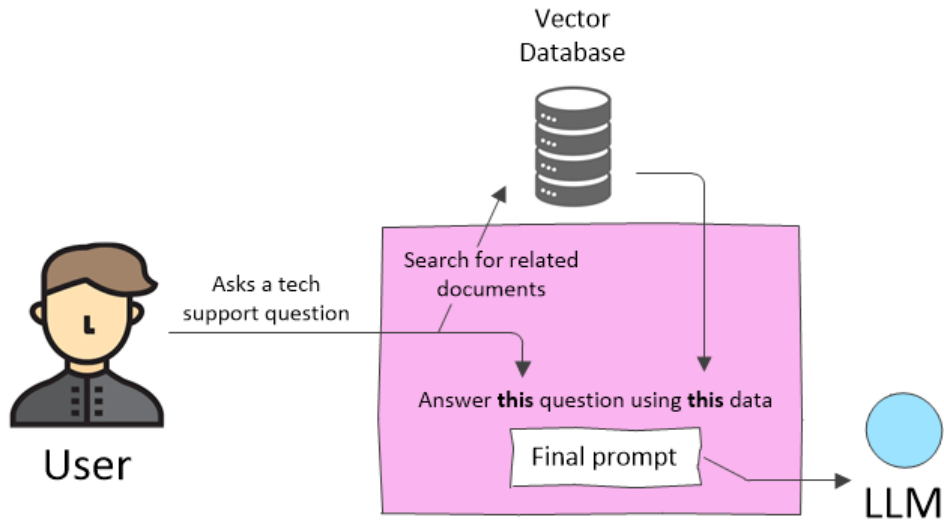
- 프롬프트 엔지니어링을 통한 개선

- 사용자의 입력과 애플리케이션의 목적을 결합하여 하나의 프롬프트를 생성
- LLM에게 더 명확하고 구체적인 요청 전달
- 예를 들어, 단순한 질문을 LLM에 전달하는 것이 아니라, 사용자의 입력과 사전 정의된 가이드를 포함한 구조화된 문서 형태로 요청



## 프롬프트 체이닝 (Prompt Chaining)

- 사용자의 요청을 분석하여 수행할 작업을 추론
- 여러 개의 LLM 호출을 연결하여 최종 응답을 생성
  - ✓ 경우에 따라 분기를 만들 수도 있음.
- 예를 들어, 첫 번째 LLM 요청에서 데이터를 추출한 후, 두 번째 요청에서 해당 데이터를 기반으로 더 구체적인 답변 생성



## RAG (Retrieval-Augmented Generation)

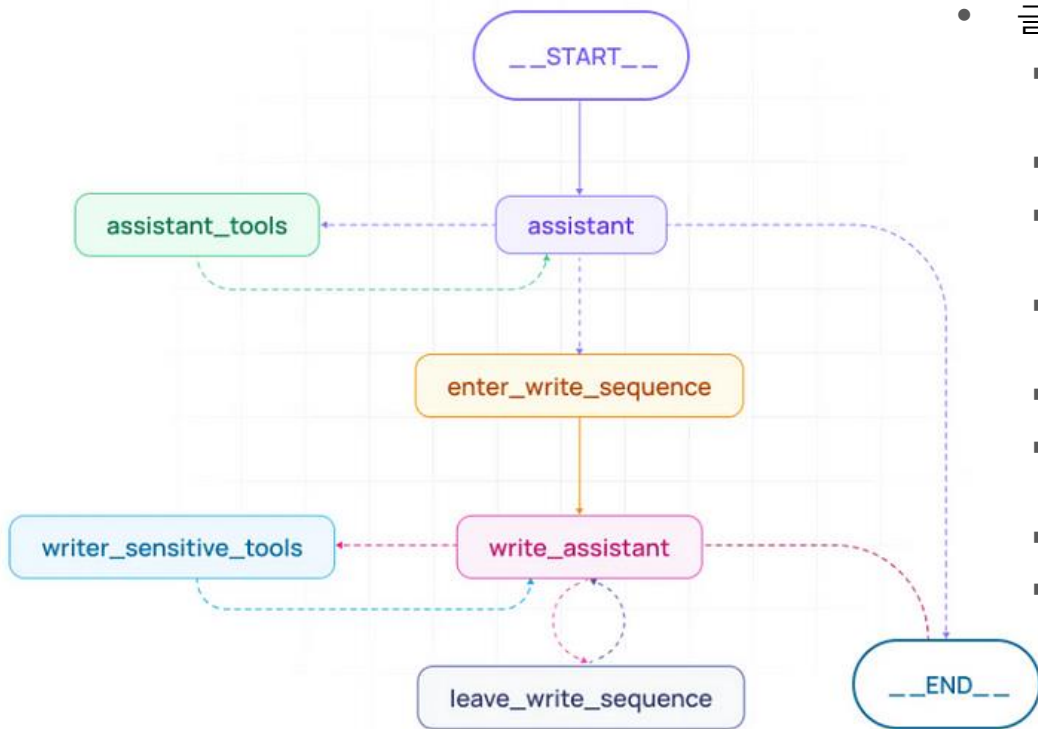
- 외부 데이터베이스에서 관련 정보를 검색하여 프롬프트에 포함하는 방식
- 그 외 외부 API를 사용해 외부에서 필요한 정보를 검색하여 프롬프트에 포함 가능

# LLM을 활용한 AI 워크플로우의 필요성

- 단순한 LLM 호출의 한계
  - 모든 질문을 하나의 경로로 처리하면, 정교한 응답 제공 불가능
  - 사용자의 의도를 파악하고 상황별 적절한 응답을 제공해야 함
  - 외부 데이터(API, 검색 시스템, DB)와 연동이 필요할 수도 있음
- AI 워크플로우에서 분기(Conditional Node)의 역할
  - 사용자의 입력을 분석하여 최적의 경로로 연결
  - 필요한 경우 외부 API 호출 및 DB 조회로 연결
  - 각 상황별 맞춤형 응답 제공

# LangGraph

- LangGraph는 LLM 기반 애플리케이션을 그래프 구조로 구성하는 프레임워크
  - 복잡한 AI 워크플로우를 직관적으로 설계하고 실행 가능
  - 노드(Node)와 엣지(Edge)를 활용하여 다중 단계 응답 시스템 구성 가능
  - 상태(State)를 관리하며 복잡한 AI 흐름을 설계하는 데 최적화됨
- LangGraph의 핵심 개념
  - 노드(Node) – 특정 기능을 수행하는 단위 (LLM 호출, API 연결, 데이터 검색)
  - 엣지(Edge) – 노드 간 연결, 사용자의 입력에 따라 경로 변경(Conditional Edge) 가능
  - State Management – 대화 이력 및 세션 유지 가능



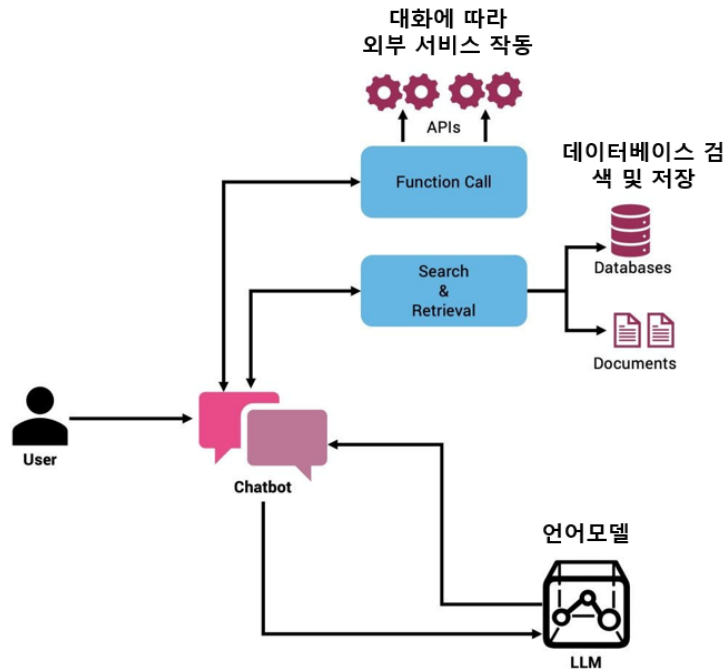
## • 글쓰기 작업 에이전트의 워크플로우

- 시작 노드 (\_\_START\_\_) → 전체 워크플로우의 시작점을 나타냄
- assistant 노드 → 기본적인 어시스턴트 기능을 처리하는 단계
- assistant\_tools 노드 → 어시스턴트가 특정 도구(예: API 호출, 검색 시스템)를 활용하는 단계
- enter\_write\_sequence 노드 → 작성(Write) 작업을 시작하는 트리거
- write\_assistant 노드 → 실제로 AI가 글을 작성하는 단계
- writer\_sensitive\_tools 노드 → 글쓰기 도중 민감한 데이터를 다루는 도구 활용 가능성
- leave\_write\_sequence 노드 → 글쓰기 작업이 완료되는 단계
- 종료 노드 (\_\_END\_\_) → 워크플로우 종료



# Function Call을 활용한 외부 API 연동

- Function Calling 개념
  - LLM이 외부 API를 직접 호출하여 최신 정보 활용 가능
  - 금융 데이터, 뉴스, 날씨 정보 등과 결합 가능
  - Function Call을 활용한 AI 시스템 예시
    - ✓ 사용자: "비트코인 가격 알려줘"
    - ✓ AI: "실시간 가격을 조회하는 중..."
    - ✓ API 호출 후 응답 제공



**Thank you!**