CH04 모델(Model)

LLM의 성능은 RAG 시스템의 전체적인 성능과 사용자 만족도에 직접적으로 영향을 미

OpenAI `GPT-4o` 활용

```
# 단계 7: 언어모델(LLM) 생성
# OpenAI 의 GPT-4o 모델 을 생성합니다.
llm = ChatOpenAI(model_name="gpt-4o")
```

Anthropic `Claude3 Sonnet` 활용

```
from langchain_anthropicimport ChatAnthropic

# 단계 7: 언어모델(LLM) 생성
# Anthropic 의 Claude 모델 을 생성합니다.
llm = ChatAnthropic(model="claude-3-sonnet-20240229")
```

로컬 모델(`llama3-8b`) 활용

```
from langchain_community.chat_modelsimport ChatOllama

# 단계 7: 언어모델(LLM) 생성
# LangChain이 지원하는 Ollama(로컬) 모델을 사용합니다.
llm = ChatOllama(model="llama3:8b")
```

1. 다양한 LLM 모델 활용

OpenAI

옵션

`temperature`

- 샘플링 온도. 0과 2 사이
  - 높은 값(예: 0.8)은 출력을 더 무작위
  - 낮은 값(예: 0.2)은 출력을 더 집중되고 결정론적

`max_tokens`

- 채팅 완성에서 생성할 토큰의 최대 개수
    - 한 번에 생성할 수 있는 텍스트의 길이를 제어

`model_name`

- 적용 가능한 모델을 선택

모델명	설명
gpt-4o	GPT-4 터보보다 저렴하고 빠른 최신 다중모드 플래그십 모델
gpt-4-turbo	최신 GPT-4 터보 모델. 비전 기능, JSON 모드, 기능 호출 지원
gpt-4o-mini	GPT-3.5 터보보다 더 우수한 성능의 작은 모델
o1-preview	다양한 도메인의 어려운 문제 해결을 위한 추론 모델
o1-mini	코딩, 수학, 과학에 특화된 빠른 추론 모델
gpt-4o-realtime	실시간 API용 오디오 및 텍스트 입력 처리 모델 (베타)

Anthropic

- **Claude**

Perplexity

- Perplexity Pro 구독 시 **매월 $5 상당의 API 크레딧 제공**

매개변수

model
사용할 언어 모델

temperature
응답의 무작위성을 조절 (0.0-1.0), 0은 결정적, 1은 가장 무작위한 응답

top_p
토큰 샘플링의 확률 임계값 설정 (0.0-1.0), 높을수록 더 다양한 출력 허용

search_domain_filter
검색 결과를 지정된 도메인으로 제한, 리스트 형태로 제공 (예: ["perplexity.ai"])
특정 웹사이트에서만 정보를 찾도록 **범위를 지정**

return_images
응답에 이미지 포함 여부를 결정하는 불리언 플래그.

return_related_questions
관련 질문 제안 기능을 활성화/비활성화하는 불리언 플래그.

top_k
사용할 검색 결과의 수 제한 (0은 제한 없음을 의미).

streaming
응답을 스트리밍으로 받을지 완성된 형태로 받을지 결정하는 불리언 플래그.

presence_penalty
토큰(이미 사용했던 단어) 반복에 대한 페널티 (-2.0에서 2.0), 높을수록 재사용을 억제.
높은 값 =더 다양한 어휘

frequency_penalty
글에 **자주 쓰는 단어**가 나오면 **벌점**을 주고, **잘 안 쓰는 희귀한 단어**에 **가점**을 줍니다. 이는 글을 더 **독창적이고 다양하게** 만듭니다.

- **높은 값 (예: 2.0): 자주 쓰는 단어**의 사용을 **강력하게 억제**하고, **희귀한 단어**를 **선호**하도록 만듭니다. (글이 **창의적**이지만 **어색**해질 수 있음)
- **낮은 값 (예: -2.0): 자주 쓰는 단어**를 더 **편하게** 사용하도록 유도합니다. (글이 **자연스럽지만 평범**해질 수 있음)

`ChatPerplexity` 는 지식 정보의 출처를 `citations` 속성에 저장

Cohere

기업용 인공지능 솔루션을 제공하는 선도적인 AI 기업

- **Command R+:** 기업용으로 최적화된 Cohere의 최신 LLM
  - 10개 주요 비즈니스 언어 지원
- **Aya: 오픈소스 다국어 LLM**
  - 101개 언어 지원

Upstage

인공지능(AI) 기술, 특히 대규모 언어 모델(LLM)과 문서 AI 분야에 특화된 국내 스타트업

- **Solar LLM**
- **Document AI Pack**
  - OCR 기술을 기반으로 한 문서 처리 솔루션으로, 복잡한 문서에서 필요한 내용을 정확히 추출하고 디지털화
- **AskUp Seargest**

    - 개인화된 검색 및 추천 서비스

Xionic

대한민국의 유망한 인공지능 스타트업으로, **기업용** 생성형 AI 솔루션을 개발

1. **STORM Platform**: 기업이 생성형 AI를 기술적 고민 없이 바로 적용할 수 있도록 하는 플랫폼
2. **STORM Answer**: 기업에 최적화된 생성형 AI 솔루션
3. **Xionic**: 상업적 이용이 가능한 라이센스의 한국어 AI 모델

LogicKor

한국어 언어 모델의 다분야 사고력을 평가하기 위해 만들어진 벤치마크 리더보드

- **실험 공간 제공**: 다양한 **한국어 LLM**을 가져와서 **LogicKor가 제시하는 테스트 문제**를 풀게 합니다.
- **성능 평가**: 모델들이 얼마나 **논리적**으로 생각하고 **다양한 분야의 문제**를 잘 해결하는지 **점수**를 매깁니다.
- **리더보드 (순위표)**: 이 점수를 바탕으로 모델들의 **순위**를 공개하여, **어떤 모델이 가장 성능이 좋은지** 한눈에 보여줍니다.

평가 영역

- 한국어 추론
- 수학
- 글쓰기
- 코딩
- 이해력

2. 캐싱(Cache)

InMemoryCache

인메모리 캐시를 사용하여 동일 질문에 대한 답변을 저장하고, 캐시에 저장된 답변을 반환합니다.

```
from langchain.globals import set_llm_cache
from langchain.cache import InMemoryCache

# 인메모리 캐시를 사용합니다.
set_llm_cache(InMemoryCache())
```

CPU times: user 112 ms, sys: 11.2 ms, total: 124 ms
Wall time: 3.51 s

CPU times: user 26.7 ms, sys: 3.63 ms, total: 30.3 ms
Wall time: 2.65 s → 훨씬 빨라짐!

SQLite Cache

```
from langchain_community.cache import SQLiteCache
from langchain_core.globals import set_llm_cache
import os

# 캐시 디렉토리를 생성합니다.
if not os.path.exists("cache"):
    os.makedirs("cache")

# SQLiteCache를 사용합니다.
set_llm_cache(SQLiteCache(database_path="cache/llm_cache.db"))
```

3. 모델 직렬화(Serialization) - 저장 및 불러오기

직렬화(Serialization) 란?
• 모델을 저장 가능한 형식으로 변환하는 과정

목적

- 모델 재사용 (재훈련 없이)
- **모델 배포** 및 공유 용이
- 계산 리소스 절약

장점

- 빠른 모델 로딩
- 버전 관리 가능

- 다양한 환경에서 사용 가능

`is_lc_serializable` 클래스 메서드로 실행하여 LangChain 클래스가 직렬화 가능한지 확인할 수 있습니다. class, llm 객체, chain의 직렬화 가능여부 확인가능!

체인(Chain) 직렬화

직렬화 가능한 모든 객체를 딕셔너리 (키-값)또는 JSON 문자열로 변환하는 과정

- `dumps` : 객체를 JSON 문자열로 직렬화 →type str
- `dumpd` : 객체를 딕셔너리로 직렬화 →dict

Pickle 파일

Python 객체를 바이너리 형태로 직렬화하는 포맷

- `pickle.dump()` : 객체를 파일에 저장
- `pickle.load()` : 파일에서 객체 로드

  pickle 파일로 저장합니다.

  ```
  import pickle

  # fuit_chain.pkl 파일로 직렬화된 체인을 저장합니다.
  with open("fruit_chain.pkl", "wb")as f:
      pickle.dump(dumpd_chain, f)
  ```

  JSON 형식으로 마찬가지로 저장할 수 있습니다.

  ```
  import json

  with open("fruit_chain.json", "w")as fp:
      json.dump(dumpd_chain, fp)
  ```

  load: 저장한 모델 불러오기

  먼저, 이전에 저장한 `pickle` 형식의 파일을 로드합니다.

  ```
  import pickle

  # pickle 파일을 로드합니다.
  ```

```
with open("fruit_chain.pkl", "rb")as f:
    loaded_chain = pickle.load(f)
```

로드한 json 파일을 `load` 메서드를 사용하여 로드합니다.

```
from langchain_core.loadimport load

# 체인을 로드합니다.
chain_from_file = load(loaded_chain)

# 체인을 실행합니다.
print(chain_from_file.invoke({"fruit": "사과"}))
```

4. 토큰 사용량 확인

특정 호출에 대한 토큰 사용량을 추적 - 현재 OpenAI API 에만 구현

`with get_openai_callback()` 구문안에서 실행되는 모든 토큰 사용량/요금이 추적

```
# callback을 사용하여 추적합니다.
with get_openai_callback() as cb:
    result = llm.invoke("대한민국의 수도는 어디야?")
    print(cb)
```

Tokens Used: 55

    Prompt Tokens: 15
    Completion Tokens: 40

Successful Requests: 1
Total Cost (USD): $0.0006749999999999999

5. 구글 생성 AI

```
!pip install -qU langchain-google-genai
```

Safety Settings

모델이 생성하는 콘텐츠의 안전성과 적절성을 보장하고, 사용자에게 유해하거나 부적절한 내용이 노출되는 것을 방지

콘텐츠 필터링 및 안전 설정과 관련된 다양한 카테고리와 해당 임계값이 정의된 문

```
from langchain_google_genai import (
    ChatGoogleGenerativeAI,
    HarmBlockThreshold,
    HarmCategory,
)

llm = ChatGoogleGenerativeAI(
    # 사용할 모델을 "gemini-pro"로 지정합니다.
    model="gemini-1.5-pro-latest",
    safety_settings={
        # 위험한 콘텐츠에 대한 차단 임계값을 설정합니다.
        # 이 경우 위험한 콘텐츠를 차단하지 않도록 설정되어 있습니다. (그럼에도 기본적인 차단이 있을 수 있습니다.)
        HarmCategory.HARM_CATEGORY_SEXUALLY_EXPLICIT: HarmBlockThreshold.BLOCK_NONE,
        HarmCategory.HARM_CATEGORY_HATE_SPEECH: HarmBlockThreshold.BLOCK_NONE,
        HarmCategory.HARM_CATEGORY_HARASSMENT: HarmBlockThreshold.BLOCK_NONE,
        HarmCategory.HARM_CATEGORY_DANGEROUS_CONTENT: HarmBlockThreshold.BLOCK_NONE,
    },
)
```

Batch 단위 실행

여러 개의 작업이나 데이터를 **하나의 묶음(Batch)**으로 만들어서 **한 번에** 처리하는 방식

💡 왜 배치를 사용할까? **효율성**을 극대화하기 위해

네트워크 통신 비용과 서버 부하를 크게 줄일 수
병렬 처리 능력을 최대한 활용

```
from langchain_google_genai import ChatGoogleGenerativeAI
```

```
llm = ChatGoogleGenerativeAI(
    # 사용할 모델을 "gemini-pro"로 지정합니다.
    model="gemini-1.5-pro-latest",
)

results = llm.batch(
    [
        "대한민국의 수도는?",
        "대한민국의 주요 관광지 5곳을 나열하세요",
    ]
)

for res in results:
    # 각 결과의 내용을 출력합니다.
    print(res.content)
```

Multimodal 모델

멀티모달 AI (또는 LLM)는 **두 가지 이상의 모달리티**를 동시에 입력으로 받아들이거나, 또는 한 가지 모달리티를 입력받아 다른 모달리티로 출력할 수 있는 AI 시스템입니다.

‘모달리티(Modality): 정보가 존재하는 형태'의 종류

- **텍스트 (Text):** 글자, 단어, 문장
- **이미지 (Image):** 사진, 그림
- **오디오 (Audio):** 소리, 음성
- **비디오 (Video):** 영상
- **기타:** 센서 데이터, 3D 모델 등

```
from langchain_teddynote.models import MultiModal
from langchain_teddynote.messages import stream_response

# 객체 생성
gemini = ChatGoogleGenerativeAI(model="gemini-1.5-pro-latest")

system_prompt = (
    "당신은 시인입니다. 당신의 임무는 주어진 이미지를 가지고 시를 작성하는 것입니다."
```

```
)

user_prompt = "다음의 이미지에 대한 시를 작성해주세요."

# 멀티모달 객체 생성
multimodal_gemini = MultiModal(
    llm, system_prompt=system_prompt, user_prompt=user_prompt
)
# 샘플 이미지 경로(파일의 경로, URL 등)를 지정합니다.
IMAGE_URL = "images/jeju-beach.jpg"

# 이미지 파일로 부터 질의
answer = multimodal_gemini.stream(IMAGE_URL)

# 스트리밍 방식으로 각 토큰을 출력합니다. (실시간 출력)
stream_response(answer)
```

텍스트와 **이미지**를 **동시에** 모델에게 전달 →Multimodal

6. HuggingFace Endpoints

Hugging Face Hub: **머신러닝 모델**을 **클라우드 환경**에서 **손쉽게 배포**하고 **운영.** 이 온라인 플랫폼에서 사람들은 쉽게 협업하고 함께 머신러닝을 구축

참고 모델 리스트

- 허깅페이스 LLM 리더보드: https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard
- 모델 리스트: https://huggingface.co/models?pipeline_tag=text-generation&sort=downloads
- LogicKor 리더보드: https://lk.instruct.kr/

Serverless Endpoints

클라우드 환경에서 **모델 배포 및 실행**을 위한 방식 중 하나로, 사용자가 **서버 인프라를 직접 관리할 필요가 전혀 없는** 배포 형태

클라우드 서비스 제공자(AWS, Google Cloud, Hugging Face 등)가 **모델 실행 환경**을 완전히 관리해주는 방식

- **비용 효율성:** 모델을 사용하지 않을 때는 **자동으로 자원을 해제**하고 요금을 부과하지 않아, **운영 비용을 크게 절감**할 수 있습니다.
- **운영 단순화:** 모델만 배포하면 되므로, 개발팀이 인프라 관리 대신 **모델 개발과 개선**에만 집중할 수 있습니다.
- **즉각적인 확장:** 트래픽이 급증해도 시스템이 자동으로 처리량을 늘려주기 때문에 **안정적인 서비스** 제공이 가능합니다.

하지만 로드가 다른 요청과 공유되기 때문에 대용량 사용 사례에서는 속도 제한이 있을 수 있음
→ 전용 엔드포인트!

전용 엔드포인트(Dedicated Endpoint)

서버를 여러 고객이 **공유하지 않고**, **오직 한 고객**만을 위해 **24시간 전담 대기**

특징	전용 엔드포인트 (Dedicated)	서버리스 엔드포인트 (Serverless)
자원 할당	**항상** 할당되어 켜져 있음 (**Warm**)	요청 시에만 할당되고 사용 없으면 꺼짐 (**Scale-to-Zero**)
속도 (응답 시간)	매우 빠름 (지연 시간 거의 없음)	첫 요청 시 지연 시간 (**Cold Start**) 발생 가능
비용 청구	**시간 단위**로 서버 전체 비용 청구	**사용한 만큼**의 자원만 청구
보안/격리	단일 고객 전용으로 **격리 수준이 높음**	자원을 공유하지만 논리적으로 격리

결론적으로, 전용 엔드포인트는 비용이 더 들더라도 **안정적인 성능**과 **최소한의 지연 시간**, **최고의 격리 수준**이 필요한 핵심 서비스에 사용됩니다.

Enterprise Workload: 회사가 돈을 벌거나 운영을 지속하는 데 반드시 필요하고 중요한 모든 IT 업무

7. 허깅페이스 로컬(HuggingFace Local)

허깅페이스 생태계의 **도구, 모델, 데이터셋** 등을 **자신의 **로컬 컴퓨터 환경(**개인 PC, 사내 서버 등)**에서 직접 사용하고 실행하는 방식을 총칭

'로컬' 사용의 장점

- **비용 절감:** 클라우드 컴퓨팅 자원(GPU 인스턴스) 사용료를 지불하지 않고, 자신의 하드웨어 자원을 사용합니다.

- **완전한 통제권:** 데이터와 모델을 외부 네트워크에 전송할 필요 없이 **보안이 강화된 내부 환경**에서 작업할 수 있습니다.
- **맞춤형 환경:** 원하는 라이브러리 버전과 운영체제 환경을 자유롭게 설정하여 사용할 수 있습니다.

모델 캐시 경로 설정

```
os.environ["TRANSFORMERS_CACHE"] = "./cache/"
os.environ["HF_HOME"] = "./cache/"
```

`TRANSFORMERS_CACHE` & `HF_HOME`: 허깅페이스 라이브러리(`transformers` 등)가 **모델 파일, 토크나이저 파일** 등을 다운로드받아 저장할 때 **기본 경로**를 지정합니다. 여기서는 현재 실행 디렉터리 아래의 `./cache/` 폴더에 저장하도록 설정했습니다.

HuggingFace 모델 로드 및 설정

```
from langchain_huggingface import HuggingFacePipeline

llm = HuggingFacePipeline.from_model_id(
    model_id="microsoft/Phi-3-mini-4k-instruct",
    task="text-generation",
    pipeline_kwargs={
        "max_new_tokens": 256,
        "top_k": 50,
        "temperature": 0.1,
    },
)

question = "Hugging Face is"
response = llm.invoke(question)
print(response)
```

9. Ollama

Get up and running with large language models.

Run Llama 3, Phi 3, Mistral, Gemma 2, and other models. Customize and create your own.

Download ↓

Available for macOS, Linux, and Windows (preview)

Ollama는 모델에 필요한 모든 것을 **`Modelfile`**이라는 **하나의 묶음**으로 만들고, **GPU 사용** 등을 포함한 **실행 설정을 자동으로 최적화**해줍니다.
→ 오픈 소스 대규모 언어 모델을 로컬에서 실행 가능!