LangChain Study 4주차 정리

category	LAB
상태	분류대기
완료	☐

1장. 대규모 언어 모델(LLM)의 개요

대규모 언어 모델(LLM, Large Language Model)은 방대한 양의 텍스트 데이터를 기반으로 학습된 인공지능 모델로, 자연어를 이해하고 생성하는 데 특화되어 있다. LLM은 인간의 언어 패턴, 의미 구조, 문법적 규칙을 학습함으로써 텍스트 요약, 번역, 질의응답, 코드 생성 등 다양한 응용 분야에서 활용된다. 최근에는 텍스트뿐 아니라 이미지, 오디오, 비디오 등 다양한 모달리티를 다루는 다중모달 모델(multimodal model)로 발전하고 있다.

LLM은 일반적으로 Transformer 아키텍처를 기반으로 하며, 셀프 어텐션(self-attention) 메커니즘을 통해 문맥을 이해하고 다음 단어를 예측하는 방식으로 작동한다. 주요 특징으로는 방대한 파라미터 수, 고차원 임베딩 표현, 사전 학습(pretraining)과 미세 조정(fine-tuning)의 두 단계 학습 과정, 그리고 인간의 피드백을 반영한 RLHF(Reinforcement Learning from Human Feedback) 기법의 사용 등이 있다.

다음은 Transformer의 핵심인 self-attention 연산을 단순화해 보여주는 예시이다.

```
import torch
import torch.nn.functional as F

query = torch.randn(1, 5, 8)
key = torch.randn(1, 5, 8)
value = torch.randn(1, 5, 8)

scores = torch.matmul(query, key.transpose(-2, -1)) / (8 ** 0.5)
weights = F.softmax(scores, dim=-1)
output = torch.matmul(weights, value)
print(output)
```

이 예시에서 각 토큰은 다른 모든 토큰과의 관계를 계산해, 문맥 정보를 반영한 새로운 표현을 생성한다. LLM은 이러한 연산을 수천 번 반복하여 문장의 전후 의미를 학습한다.

현재 LLM 생태계에는 OpenAI의 GPT 시리즈, Anthropic의 Claude, Google의 Gemini, Meta의 LLaMA, Upstage의 Solar, Cohere의 Command R, Perplexity의 Sonar, Xionic의 Xionic 모델 등 다양한 모델이 존재하며, 각기 다른 설계 철학과 장점을 바탕으로 발전하고 있다.

2장. 주요 LLM 모델별 특징

2.1 OpenAI GPT 모델

OpenAI의 GPT-4o 및 GPT-4-turbo 시리즈는 가장 대표적인 범용 LLM이다. GPT-4o는 멀티모달 입력(텍스트, 이미지, 오디오)을 모두 지원하며, 128,000토큰의 긴 문맥을 처리할 수 있다. `temperature`, `max_tokens`, `model_name` 등의 매개변수를 통해 생성의 다양성과 길이를 조절할 수 있다.

```
from langchain_openai import ChatOpenAI

llm = ChatOpenAI(model_name="gpt-4o", temperature=0.3)
response = llm.invoke("인공지능의 정의를 설명해줘.")
print(response.content)
```

이 코드에서 temperature 값을 0으로 낮추면 결정론적이고 일관된 응답을 얻고, 0.8로 높이면 창의적이지만 가변적인 결과를 얻는다. `stream()` 메서드를 사용하면 응답을 실시간으로 출력할 수도 있다.

2.2 Anthropic Claude

Anthropic의 Claude 모델은 안전성과 투명성을 강조하며, 대화의 맥락을 정교하게 유지하는 특성이 있다. Claude 3.5 Sonnet, Haiku, Opus 모델은 각각 속도와 성능의 균형을 다르게 설정한 버전이다.

```
from langchain_anthropic import ChatAnthropic

claude = ChatAnthropic(model_name="claude-3-5-sonnet-20241022")
response = claude.invoke("What are the advantages of reinforcement learn
```

```
ing?")
print(response.content)
```

Claude는 특히 문장 간 논리적 연결이 중요한 질의응답이나 요약 작업에서 강점을 보인다.

2.3 Perplexity

Perplexity는 검색 강화 생성(RAG)과 웹 실시간 검색을 결합한 모델이다. 최신 정보를 반영하면서 인용(citation) 출처를 함께 제공한다.

```
from langchain_perplexity import ChatPerplexity

pxy = ChatPerplexity(model="llama-3.1-sonar-large-32k-online")
response = pxy.invoke("What are the key differences between GPT-4 and Claude 3?")
print(response.content)
```

Perplexity는 실시간 검색 기능을 통해 뉴스나 기술 업데이트를 반영하는 응답을 생성한다.

2.4 Cohere

Cohere의 Command R+ 모델은 문서 검색과 요약 기능에 특화된 기업용 LLM이다. 다국어 지원이 뛰어나며, 실제 업무 자동화나 보고서 생성에도 활용된다.

```
from langchain_cohere import ChatCohere

co = ChatCohere(model="command-r-plus")
print(co.invoke("Summarize the recent AI trends in 5 bullet points.").content)
```

Cohere는 문서 검색, 회의 요약, 법률 문서 분석 등에서 효율적이다.

2.5 Upstage Solar

Upstage의 Solar 모델은 한국 스타트업이 개발한 LLM으로, OCR 기반 문서 처리와 검색형 문서 분석에 강점을 가진다.

```
from langchain_upstage import ChatUpstage

solar = ChatUpstage(model="solar-pro")
```

```
response = solar.invoke("서울의 주요 산업을 요약해줘.")
print(response.content)
```

Solar는 기업 환경에서 비정형 문서를 자동 요약하거나, PDF 보고서에서 주요 정보를 추출하는 데 활용된다.

2.6 Xionic

Xionic은 한국어 특화 모델로, 고품질의 자연스러운 한국어 응답을 생성한다.

```
from langchain_openai import ChatOpenAI

xionic = ChatOpenAI(base_url="https://api.xionic.ai/v1", model_name="xion
ic-2")
print(xionic.invoke("대규모 언어 모델의 학습 과정에 대해 설명해줘.").content)
```

Xionic은 한국어 데이터셋 기반으로 훈련되어, 기술 문서 번역이나 한국어 질의응답에 적합하다.

3장. 캐싱(Caching)과 모델 효율화

LLM 호출은 API 비용과 지연시간이 크기 때문에, 동일한 요청에 대해 캐싱을 적용하면 효율을 높일 수 있다. LangChain은 `InMemoryCache`와 `SQLiteCache` 두 가지 방식을 제공한다.

```
from langchain.globals import set_llm_cache
from langchain.cache import InMemoryCache, SQLiteCache
from langchain_openai import ChatOpenAI

# 인메모리 캐시 설정
set_llm_cache(InMemoryCache())
llm = ChatOpenAI(model_name="gpt-4o")
print(llm.invoke("What is LangChain?").content)

# SQLite 캐시로 변경
set_llm_cache(SQLiteCache(database_path=".langchain.db"))
print(llm.invoke("What is LangChain?").content)
```

첫 번째 호출은 실제 API 요청을 수행하지만, 동일한 요청은 이후 캐시에서 즉시 반환된다.

이 기능은 반복 질의가 많은 서비스에서 API 비용 절감과 응답 속도 향상에 매우 유용하다.

4장. 모델 직렬화(Serialization)

LangChain 모델 객체는 JSON 또는 Pickle 형태로 저장하여 재사용할 수 있다. 이는 배포 환경이나 세션 간 모델 설정을 공유할 때 특히 편리하다.

```
import pickle, json
from langchain_openai import ChatOpenAI

llm = ChatOpenAI(model_name="gpt-4o", temperature=0.5)

# JSON 직렬화
json_data = llm.dumps()
with open("llm_config.json", "w") as f:
    f.write(json_data)

# Pickle 저장
with open("llm_model.pkl", "wb") as f:
    pickle.dump(llm, f)

# 모델 복원
with open("llm_model.pkl", "rb") as f:
    loaded = pickle.load(f)
print(loaded.invoke("직렬화의 장점을 요약해줘.").content)
```

이 방식은 모델 재학습 없이 설정을 그대로 재활용할 수 있으며, 분산 서버 환경에서도 일관성을 유지한다.

5장. 토큰 사용량 추적

LangChain의 `get_openai_callback()` 은 모델 사용량과 비용을 실시간으로 측정할 수 있는 도구다.

```
from langchain.callbacks import get_openai_callback
from langchain_openai import ChatOpenAI
```

```
llm = ChatOpenAI(model_name="gpt-4o")

with get_openai_callback() as cb:
    llm.invoke("Explain overfitting in machine learning.")
    print(f"Prompt Tokens: {cb.prompt_tokens}, Completion Tokens: {cb.com
pletion_tokens}")
    print(f"Total Tokens: {cb.total_tokens}, Cost: ${cb.total_cost:.6f}")
```

이를 통해 서비스 규모 확장 시 비용 예측과 모델 선택 전략을 최적화할 수 있다.

6장. 구글 Gemini 모델 활용

Google Gemini는 텍스트, 이미지, 오디오를 동시에 처리할 수 있는 다중모달 모델이다. LangChain에서는 `ChatGoogleGenerativeAI` 클래스로 접근한다.

```
from langchain_google_genai import ChatGoogleGenerativeAI

gemini = ChatGoogleGenerativeAI(model="gemini-1.5-pro-latest")
result = gemini.invoke("이 이미지를 요약해줘.", images=["image.jpg"])
print(result.content)
```

또한 비디오 입력을 통해 특정 장면의 내용을 분석할 수 있으며, 여러 요청을 병렬 처리하는 `batch()` 메서드도 지원한다.

Gemini는 시각적 데이터가 포함된 검색, 이미지 캡션 생성, 비디오 장면 요약 등에서 높은 효율을 보인다.

7장. Hugging Face와의 통합

Hugging Face는 오픈소스 AI 모델 허브로, LangChain과 결합해 로컬 또는 클라우드 기반 추론을 쉽게 수행할 수 있다.

```
from langchain_huggingface import HuggingFacePipeline

hf = HuggingFacePipeline.from_model_id(
    model_id="beomi/llama-2-ko-7b",
    task="text-generation",
    model_kwargs={"temperature": 0.7, "max_length": 128}
```

```
)
print(hf.invoke("서울의 역사적 특징을 요약해줘.").content)
```

이 방식은 GPU 환경에서 로컬 실행이 가능하며, 사내 인프라에서 개인정보가 포함된 데이터를 안전하게 처리할 수 있다는 장점이 있다.

8장. Ollama를 통한 로컬 모델 실행

Ollama는 LLaMA, Gemma, Mistral 등의 모델을 로컬 환경에서 구동할 수 있도록 지원한다.

MacOS, Linux, Windows 환경에서 작동하며, LangChain과의 연동이 간단하다.

```
from langchain_community.llms import Ollama

ollama = Ollama(model="llama3")
response = ollama.invoke("What is reinforcement learning?")
print(response.content)
```

로컬 실행의 장점은 데이터 유출 방지, 지연시간 최소화, 네트워크 의존성 제거에 있다.

특히 기업 내부망 환경에서 민감한 데이터를 다룰 때 적합하다.

9장. GPT4All과 오픈소스 LLM 생태계

GPT4All은 nomic-ai가 개발한 완전 오픈소스 LLM 플랫폼으로, 로컬 CPU/GPU 환경에서 독립 실행이 가능하다.

```
from langchain_community.llms import GPT4All

gpt4all = GPT4All(model="ggml-model-gpt4all-falcon-q4_0.gguf")
print(gpt4all.invoke("Compare AI and traditional programming.").content)
```

이 방식은 인터넷 연결이 불가능한 환경에서도 모델을 운용할 수 있으며, 오프라인 연구나 학습용 프로젝트에서 유용하다.

또한 “EEVE-Korean-Instruct-10.8B”와 같은 한국어 특화 모델도 지원한다.

10장. 비디오 질의응답과 멀티모달 응용

Gemini API는 비디오 분석 기능을 통해 영상의 요약, 장면 추출, 특정 키워드 탐지를 수행한다.

```
import google.generativeai as genai

genai.configure(api_key="YOUR_GEMINI_API_KEY")
video = genai.upload_file("lecture.mp4")

model = genai.GenerativeModel("gemini-1.5-pro-latest")
response = model.generate_content(["영상에서 AI 관련 언급이 있는 구간을 찾아줘.", video])
print(response.text)

genai.delete_file(video.name)
```

Gemini는 업로드된 영상을 자동으로 분석하고, 특정 주제나 인물 언급 구간을 타임스탬프와 함께 반환한다.

이 기능은 교육 콘텐츠 분석, 회의록 생성, 영상 요약 등 실무적 응용이 가능하다.

결론

3주차에서 저자는 LangChain 프레임워크를 중심으로 OpenAI, Anthropic, Perplexity, Cohere, Upstage, Hugging Face, Ollama, GPT4All, Google Gemini 등 다양한 LLM을 통합적으로 사용하는 방법을 다루었다.

이러한 통합은 연구 및 산업 현장에서 모델 간 비교, 비용 최적화, 응용 도메인별 맞춤형 아키텍처 설계에 필수적이다. 또한, 캐싱·직렬화·토큰 모니터링과 같은 효율화 기법을 통해 안정적인 서비스 운영이 가능하다.

향후 LLM 환경은 단일 모델 중심에서 벗어나, 목적과 맥락에 따라 다양한 모델을 조합하고 관리하는 멀티모델(multimodel) 생태계로 진화할 것이다. LangChain과 같은 통합형 프레임워크는 이러한 변화의 핵심 인프라로 자리 잡고 있으며, 대규모 언어 모델의 응용 가능성을 극대화하고 있다.