[GDGoC] AI-Langchain-Study

https://github.com/MelonChicken/5th_AI_langchain_study

Ch 01. Langchain Study
    Intro
    01. 설치 영상보고 따라하기
    03. LangSmith 추적 설정하기
        LangSmith 의 추적기능
        프로젝트 단위 추적
        1개의 실행에 대한 세부 단계별 추적
        LangSmith API Key 발급
        .env 에 LangSmith 키 설정
    Jupyter Notebook 혹은 코드에서 추적을 활성화 하기
    langchain-teddynote
        설치방법
        LangSmith 추적 설정
    04. OpenAI API 사용 **(GPT-4o 멀티모달)**
        ChatOpenAI
LogProb 활성화
    스트리밍 출력
    멀티모달 모델 (이미지 인식)
    05. LangChain Expression Language(LCEL)
        기본 예시: 프롬프트 + 모델 + 출력 파서
        프롬프트 템플릿의 활용
        Chain 생성
    06. LCEL : LangChain Expression **Language(LCEL)**
        Parallel: 병렬성
        배치에서의 병렬 처리
    07. Runnable
        데이터를 효과적으로 전달하는 방법
        RunnableParallel
        RunnableLambda

Ch 01. Langchain Study

> 개념을 학습하며 가졌던 의문
>
> 1. **LogProb 활성화 과정에서 왜 log를 써야하는가? (확률값 그대로 사용하지 않는 이유)**
> 2. 멀티모달을 사용하는 경우 이미지 인식 과정에서 제한된 정보에서의 추출을 하다보니 한계점이 보임 (인물 인식) → 텍스트 추출을 하는걸까? 한다면 어느 범위까지 제한이 걸려있을까?
> 3. 동기와 비동기의 퍼포먼스 차이는 존재하는가? → 당장은 없어보이는데 그렇다면 동기와 비동기는 언제 적용될 때 유의미한다?

Intro

> 💡 **Langchain**
> 언어 모델을 활용해 다양한 애플리케이션을 개발할 수 있는 프레임워크

pip 를 이용한 설치

```
pip install -r https://raw.githubusercontent.com/teddylee777/langchain-kr/main/requirements.txt
```

최소한의 기능만 설치하기 위한 **mini** 버전 (일부 패키지만 설치하는 경우)

```
pip install -r https://raw.githubusercontent.com/teddylee777/langchain-kr/main/requirements-mini.txt
```

Langchain 의 기능

1. 문맥을 인식하는 기능

   : 문맥(Context) 소스와 언어 모델 (GPT?)를 연결한다.

   예 : 프롬프트 지시사항, 소수의 예시, 응답에 근거한 내용 ...

2. 추론하는 기능

   : 언어 모델은 주어진 문맥을 바탕으로 어떠한 답변을 제공하거나, 어떤 조치를 취해야할지를 스스로 추론할 수 있다.

Langchain의 구성요소

1. LangChain 라이브러리 : Python 및 JavaScript 라이브러리
   - 다양한 컴포넌트의 인터페이스와 통합
   - 이러한 컴포넌트를 체인과 에이전트로 결합하는 기본 런타임
   - 즉시 사용 가능한 체인과 에이전트의 구현
2. **LangChain 템플릿**: 다양한 작업을 위한 쉽게 배포할 수 있는 참조 아키텍처 모음입니다.
3. **LangServe**: LangChain 체인을 REST API로 배포하기 위한 라이브러리입니다.
4. **LangSmith**: 어떤 LLM 프레임워크에도 구축된 체인을 디버그, 테스트, 평가, 모니터링할 수 있게 해주며 LangChain과 원활하게 통합되는 **개발자 플랫폼**입니다.
5. **LangGraph**: LLM을 사용한 상태유지가 가능한 다중 액터 애플리케이션을 구축하기 위한 라이브러리
   - LangChain 위에 구축되었으며 LangChain과 함께 사용하도록 설계.
   - 여러 계산 단계에서 다중 체인(또는 액터)을 순환 방식으로 조정할 수 있는 능력을 LangChain 표현 언어에 추가한다.

Langchain의 장점

1. 컴포넌트의 조립 및 통합
   - 언어 모델과의 작업을 위한 조립 가능한 도구 및 통합을 제공
   - 모듈식으로 설계됨. → 프레임워크로서의 활용도 보장

     뭉탱이 하나의 덩어리로 → 역할에 따라서 쪼개놓을 수도 잇고?
2. 즉시 사용 가능한 체인
   - 고수준 작업을 수행하기 위한 컴포넌트의 내장 조합 제공
   - 개발 과정을 간소화, 가속.

주요 모듈

- 모델 I/O
  - 프롬프트 관리 최적화 및 LLM과의 일반적인 인터페이스, 작업을 위한 유틸리티...
- 검색
  - 데이터 강화 생성에 초점. 외부 데이터 소스에서 가져오는 작업
- 에이전트
  - 언어 모델의 액션 의사결정, 실행 및 관찰, 반복 (전반적인 관리)

01. 설치 영상보고 따라하기

https://www.youtube.com/watch?v=mVu6Wj8Z7C0

- pyenv의 기능과 역할
- poetry의 기능과 역할

- pyproject.toml의 프로젝트 관리를 위한 것.

https://teddynote.com/10-RAG%EB%B9%84%EB%B2%95%EB%85%B8%ED%8A%B8/%ED%99%98%EA%B2%BD%20%EC%84%A4%EC%A0

03. LangSmith 추적 설정하기

> LangSmith는 **LLM 애플리케이션 개발, 모니터링 및 테스트** 를 위한 플랫폼

LangSmith 의 추적기능

추적은 LLM 애플리케이션의 동작을 이해하기 위한 도구

추적은 다음과 같은 문제를 추적하는 데 도움이 될 수 있습니다.

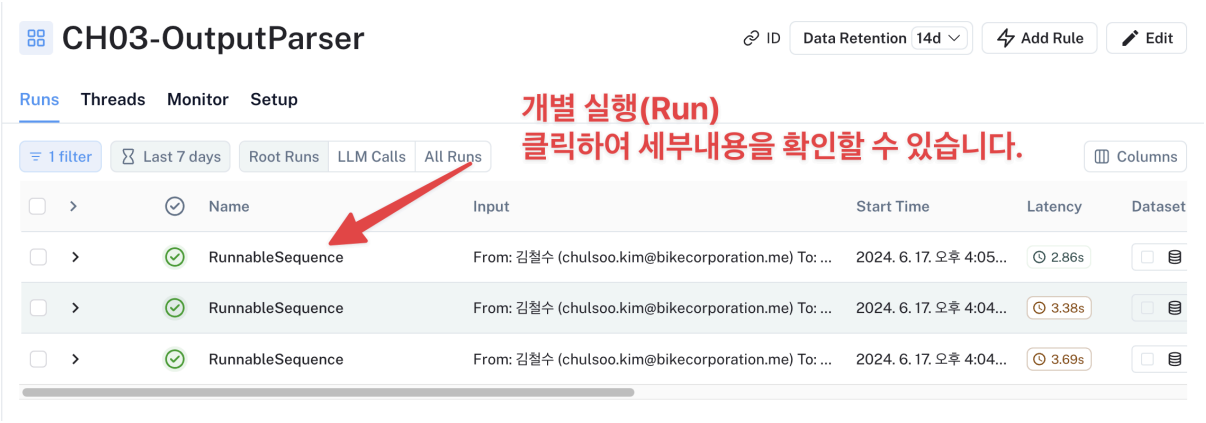
- 예상치 못한 최종 결과
- 에이전트가 루핑되는 이유
- 체인이 예상보다 느린 이유
- 에이전트가 각 단계에서 사용한 토큰 수

프로젝트 단위 추적

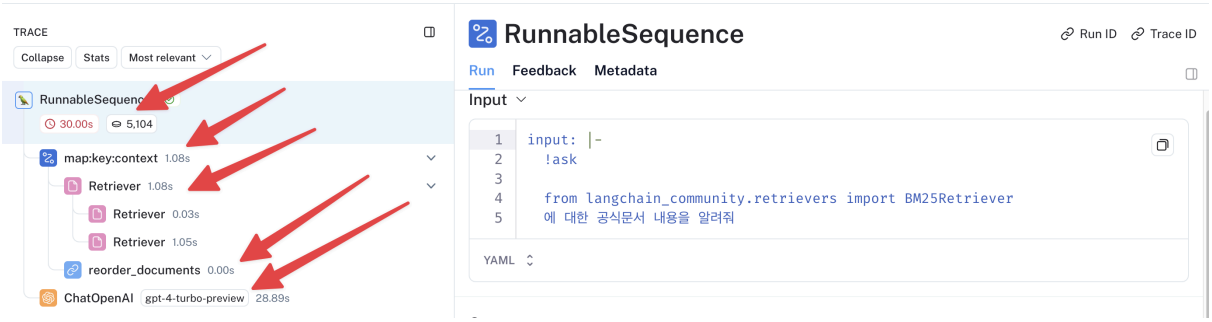
프로젝트 단위로 실행 카운트, Error 발생률, 토큰 사용량, 과금 정보등을 확인

Name	Feedback (7D)	Run Count (7D)	Error Rate (7D)	% Streaming (7D)	Total Tokens (7D)	Total Cost (7D)	P50 Latency (7D)
CH04-Models		22	0%	0%	6,501	$0.0091835	0.00s
CH01-Basic		48	0%	9%	189,876	$0.9631875	1.02s
llama3-agent		33	12%	94%	94,709		18.39s
CH03-OutputParser		3	0%	67%	3,332	$0.002268	3.38s
CH02-Prompt		29	3%	24%	17,389	$0.26302	3.30s

프로젝트를 클릭하면 실행된 모든 Run 이 나타난다.



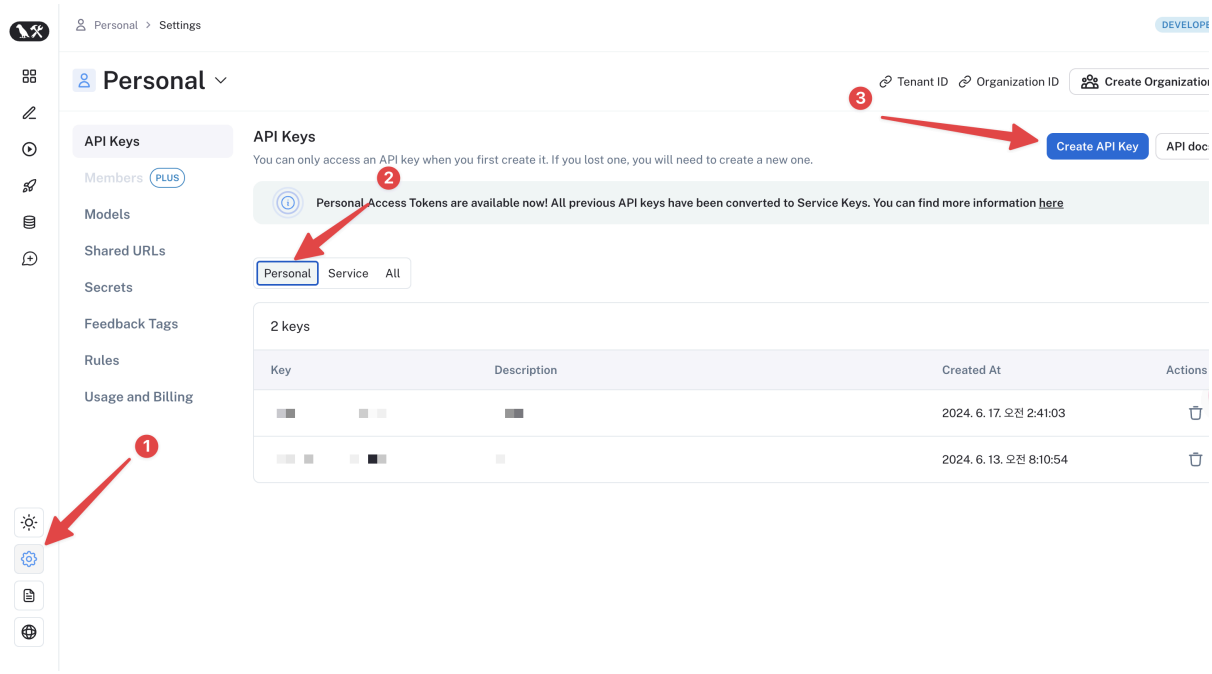
1개의 실행에 대한 세부 단계별 추적



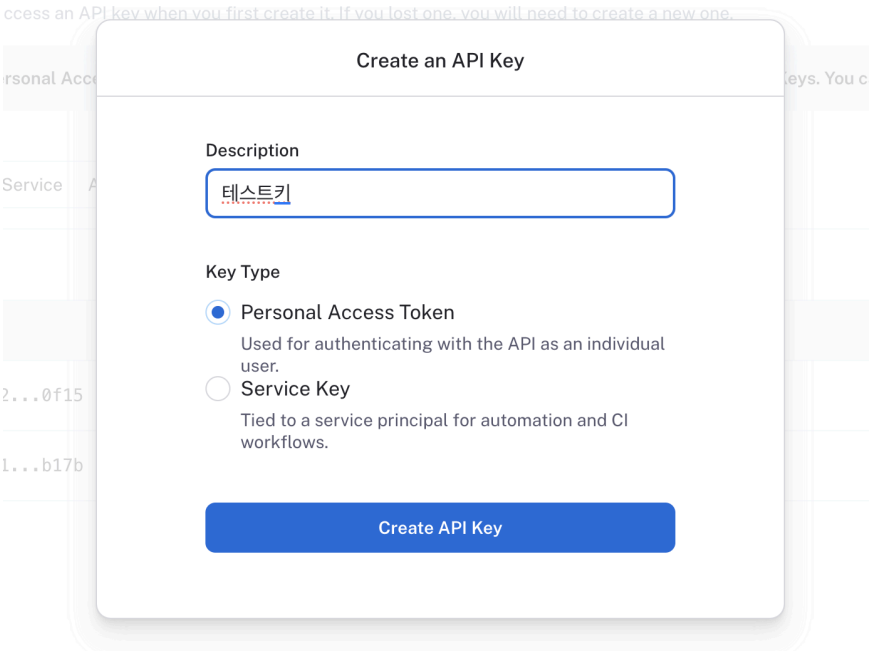
- GPT의 입출력 내역까지 자세하게 기록하기에 검색 알고리즘, 프롬프트의 영향에 대해서도 볼 수 있다.
- 상단에 실행 시간과 사용 토큰까지 표기가 되기 때문에 청구 금액까지 표기해준다.
- 이제 실제로 볼 수 있다.

LangSmith API Key 발급

1. https://smith.langchain.com/ 으로 접속하여 회원가입을 진행합니다.
2. 가입후 이메일 인증하는 절차를 진행해야 합니다.
3. 왼쪽 톱니바퀴(Setting) - 가운데 "Personal" - "Create API Key" 를 눌러 API 키를 발급 받습니다.



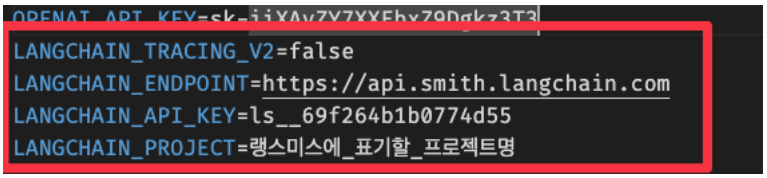
Description 에 본인이 알 수 있는 설명을 넣고 Create API Key 버튼을 클릭하여 생성합니다.



.env 에 LangSmith 키 설정

먼저, `.env` 파일에 LangSmith 에서 발급받은 키와 프로젝트 정보를 입력합니다.

- `LANGCHAIN_TRACING_V2` : **"true"** 로 설정하면 추적을 시작합니다.
- `LANGCHAIN_ENDPOINT` : `https://api.smith.langchain.com` 변경하지 않습니다.
- `LANGCHAIN_API_KEY` : 이전 단계에서 **발급받은 키** 를 입력합니다.
- `LANGCHAIN_PROJECT` : **프로젝트 명** 을 기입하면 해당 프로젝트 그룹으로 모든 실행(Run) 이 추적됩니다.



Jupyter Notebook 혹은 코드에서 추적을 활성화 하기

추적을 활성화 하는 방법은 매우 간단합니다. 환경 변수만 설정하면 됩니다.

.env 에 설정한 내용을 불러옵니다.

```
from dotenvimport load_dotenv

load_dotenv()
```

만약 설정한 추적이 활성화 되어 있고, API KEY 와 프로젝트 명이 제대로 설정되어 있다면, 이걸로도 충분합니다.

하지만, 프로젝트 명을 변경하거나, 추적을 변경하고 싶을 때는 아래의 코드로 변경할 수 있습니다.

```
import os

os.environ["LANGCHAIN_TRACING_V2"] = "true"
os.environ["LANGCHAIN_ENDPOINT"] = "https://api.smith.langchain.com"
os.environ["LANGCHAIN_PROJECT"] = "LangChain 프로젝트명"
os.environ["LANGCHAIN_API_KEY"] = "LangChain API KEY 입력"
```

langchain-teddynote

langchain 관련 기능을 보다 더 편리하게 사용하기 위한 목적으로 `langchain-teddynote` 패키지를 만들었습니다.

설치방법

설치코드 (터미널에서 실행 혹은 Jupyter Notebook 에서 실행)

```
pip install langchain-teddynote
```

LangSmith 추적 설정

`.env` 파일에 **LangSmith API 키가 설정** 되어 있어야 합니다.(`LANGCHAIN_API_KEY`)

```
from langchain_teddynoteimport logging

# 프로젝트 이름을 입력합니다.
logging.langsmith("원하는 프로젝트명")
```

출력예시

```
LangSmith 추적을 시작합니다.
[프로젝트명]
랭체인 튜토리얼 프로젝트
```

추적을 원하지 않을 때는 다음과 같이 **추적을 끌 수** 있습니다.

```
from langchain_teddynoteimport logging

# set_enable=False 로 지정하면 추적을 하지 않습니다.
logging.langsmith("랭체인 튜토리얼 프로젝트", set_enable=FalseOp)
```

04. OpenAI API 사용 (GPT-4o 멀티모달)

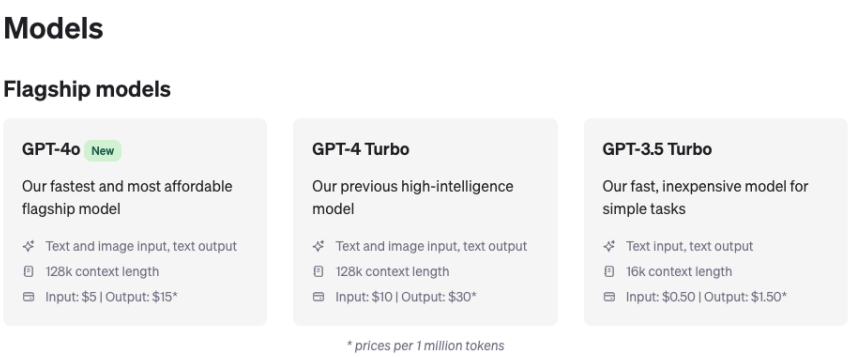
ChatOpenAI

OpenAI 사의 채팅 전용 LLM

옵션

- `temperature` : 사용할 샘플링 온도는 0~2 사이에서 선택 가능
  - 0.8과 같은 높은 값은 출력을 더 **무작위**하게 만들고
  - 0.2와 같은 낮은 값은 출력을 더 집중되고 **결정론**적으로 만듭니다.
  - transformation과정에서 softmax function
  - embedding (Natural Language Process)
    - 단어의 벡터화 (범위를 제한할 수 없다, weight?) →
    - I make bread for _____ . ____?
    - her : 35, me :20, noodle : 0.123131 → **확률분포 (0~1)** 로 그중 높은 값을 취한다
    - 어느값 → 0~1 값으로 환산 (softmax)
      - 확률분포가 정규분포, uniform, ... → 분산을 어떻게?
      - 분산 낮으면 → narrow → temperature가 낮다.**her : 99**, me :10, noodle : 0.001
        - 결정론적으로 일관되게!
      - 분산 높으면 → **Disperse → temperature가 높자 :.her : 30**, me :30, noodle : 10
        - 창의성!
- `max_tokens` : 채팅 완성에서 생성할 토큰의 최대 개수
- `model_name` : 적용 가능한 모델 리스트 - `gpt-3.5-turbo` - `gpt-4-turbo` - `gpt-4o`

https://platform.openai.com/docs/models




```
from langchain_openai import ChatOpenAI

# 객체 생성
llm = ChatOpenAI(
    temperature=0.1,  # 창의성 결정론적 vs. 산발적 (0.0 ~ 2.0)
    model_name="gpt-4o",  # 모델명
)

# 질의내용
question = "대한민국의 수도는 어디인가요?"

# 질의
print(f"[답변]: {llm.invoke(question)}")
```

> [답변]: content='대한민국의 수도는 서울입니다.' additional_kwargs={'refusal': None} response_metadata={'token_usage': {'completion_tokens': 8, 'prompt_tokens': 16, 'total_tokens': 24, 'completion_tokens_details': {'accepted_prediction_tokens': 0, 'audio_tokens': 0, 'reasoning_tokens': 0, 'rejected_prediction_tokens': 0}, 'prompt_tokens_details': {'audio_tokens': 0, 'cached_tokens': 0}}, 'model_name': 'gpt-4o-2024-08-06', 'system_fingerprint': 'fp_f33640a400', 'id': 'chatcmpl-CMZctjdzf61hUh9QnLKEdqrEjU8xQ', 'service_tier': 'default', 'finish_reason': 'stop', 'logprobs': None} id='run--327ef2e7-fbbc-4750-b76e-b105b7347fae-0' usage_metadata={'input_tokens': 16, 'output_tokens': 8, 'total_tokens': 24, 'input_token_details': {'audio': 0, 'cache_read': 0}, 'output_token_details': {'audio': 0, 'reasoning': 0}}

LogProb 활성화

주어진 텍스트에 대한 모델의 **토큰 확률의 로그 값**

토큰이란 문장을 구성하는 개별 단어나 문자 등의 요소를 의미하고, 확률은 **모델이 그 토큰을 예측할 확률**을 나타냅니다.

→ 왜 log를 써야하는가?

1. 확률의 값은 너무 작다 → 로그를 통해 조금 더 구별되게, 크기를 키운다
2. 확률을 구할 때 곱셈이 아니라 덧셈으로 해석할 수 있다.
3. 비교할 때 상대적 크기를 쉽게 알 수 있어 (로그는 단조 증가, 따라서 x가 크면 확률도 커진다) 값만 비교해도 순위를 알 수 있다
4. 머신러닝과 통계에서 표준으로 사용하는 log-likelihood

> LogProb의 유용성
>
> 1. 모델의 응답의 신뢰도
>
>   → 토큰에 대해 얼마나 확신을 가지는가, 모호한가? 확신이 있는 선택인가 → temperature?
> 2. 후보 토큰 (Top-k) 확인
>
>   다른 가능성 있는 응답 후보를 볼 수 있다.

3. 후처리 및 커스터마이징

   threshold (임계점) 기반 필터링이나 재랭킹을 할 수 있다

   확률 기반의 의사결정 시스템 응용 가능

4. 디버깅 및 평가에 활용

```
# 객체 생성
llm_with_logprob = ChatOpenAI(
    temperature=0.1,  # 창의성 (0.0 ~ 2.0)
    max_tokens=2048,  # 최대 토큰수
    model_name="gpt-3.5-turbo",  # 모델명
).bind(logprobs=True)

# 질의내용
question = "대한민국의 수도는 어디인가요?"

# 질의
response = llm_with_logprob.invoke(question)

# 결과 출력
response.response_metadata
```


{'token_usage': {'completion_tokens': 15,
'prompt_tokens': 24,
'total_tokens': 39,
'completion_tokens_details': {'accepted_prediction_tokens': 0,
'audio_tokens': 0,
'reasoning_tokens': 0,
'rejected_prediction_tokens': 0},
'prompt_tokens_details': {'audio_tokens': 0, 'cached_tokens': 0}},
'model_name': 'gpt-3.5-turbo-0125',
'system_fingerprint': None,
'id': 'chatcmpl-CMZoNWIKBR02O4dWHQOBSVIRJXThr',
'service_tier': 'default',
'finish_reason': 'stop',
'logprobs': {'content': [{'token': '대',
'bytes': [235, 140, 128],
'logprob': -0.047983073,
'top_logprobs': []},
{'token': '한',
'bytes': [237, 149, 156],
'logprob': -1.9361265e-07,
'top_logprobs': []},
{'token': '\\xeb\\xaf',
'bytes': [235, 175],
'logprob': -1.6240566e-06,
'top_logprobs': []},
{'token': '\\xbc', 'bytes': [188], 'logprob': 0.0, 'top_logprobs': []},
{'token': '\\xea\\xb5',
'bytes': [234, 181],
'logprob': -3.1281633e-07,
'top_logprobs': []},
{'token': '\\xad', 'bytes': [173], 'logprob': 0.0, 'top_logprobs': []},
{'token': '의',
'bytes': [236, 157, 152],
'logprob': -4.723352e-06,
'top_logprobs': []},

{'token': ' 수',
'bytes': [32, 236, 136, 152],
'logprob': -6.704273e-07,
'top_logprobs': []},
{'token': '도',
'bytes': [235, 143, 132],
'logprob': -5.5122365e-07,
'top_logprobs': []},
{'token': '는',
'bytes': [235, 138, 148],
'logprob': -1.9361265e-07,
'top_logprobs': []},
{'token': ' 서',
'bytes': [32, 236, 132, 156],
'logprob': -3.0545007e-06,
'top_logprobs': []},
{'token': '\\xec\\x9a',
'bytes': [236, 154],
'logprob': 0.0,
'top_logprobs': []},
{'token': '\\xb8', 'bytes': [184], 'logprob': 0.0, 'top_logprobs': []},
{'token': '입니다',
'bytes': [236, 158, 133, 235, 139, 136, 235, 139, 164],
'logprob': -0.13991028,
'top_logprobs': []},
{'token': '.',
'bytes': [46],
'logprob': -2.220075e-06,
'top_logprobs': []}],
'refusal': None}}

스트리밍 출력

스트리밍 옵션은 질의에 대한 답변을 실시간으로 받을 때 유용하다.

→ 현재 ChatGPT나 Gemini 등 우리가 사용하는 서비스에서 출력하는 ㅂ아식

```
# 스트림 방식으로 질의
# answer 에 스트리밍 답변의 결과를 받습니다.
answer = llm.stream("대한민국의 아름다운 관광지 10곳과 주소를 알려주세요!")
```

```
# 스트리밍 방식으로 각 토큰을 출력합니다. (실시간 출력)
for tokenin answer:
    print(token.content, end="", flush=True)
```

멀티모달 모델 (이미지 인식)

여러 가지 형태의 정보(모달)를 통합하여 처리하는 기술

gpt-4o나 gpt-4-turbo 모델은 이미지 인식 기능 (vision) 이 추가되어있다.

- 텍스트 : 글자
- 이미지 : 시각
- 오디오 : 청각
- 비디오 : 시각 + 청각

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

```
# 객체 생성
llm = ChatOpenAI(
    temperature=0.1,  # 창의성 (0.0 ~ 2.0)
    max_tokens=2048,  # 최대 토큰수
    model_name="gpt-4o",  # 모델명
)

# 멀티모달 객체 생성
multimodal_llm = MultiModal(llm)

# 샘플 이미지 주소(웹사이트로 부터 바로 인식)
IMAGE_URL = "https://t3.ftcdn.net/jpg/03/77/33/96/360_F_377339633_Rtv9I77sSmSNcev8bEcnVxTHrXB4nRJ5.jpg"

# 이미지 파일로 부터 질의
answer = multimodal_llm.stream(IMAGE_URL)
# 스트리밍 방식으로 각 토큰을 출력합니다. (실시간 출력)
stream_response(answer)
```

> 다른 사진으로 바뀌었을때 오류가 발생한다 (다운로드가 안되는 경우 오류 발생)
>
> ```
> from langchain_teddynote.models import MultiModal
> from langchain_teddynote.messages import stream_response
>
> # 객체 생성
> llm = ChatOpenAI(
>     temperature=0.1,  # 창의성 (0.0 ~ 2.0)
>     max_tokens=2048,  # 최대 토큰수
>     model_name="gpt-4o",  # 모델명
> )
>
> # 멀티모달 객체 생성
> multimodal_llm = MultiModal(llm)
>
> # 샘플 이미지 주소(웹사이트로 부터 바로 인식)
> IMAGE_URL = "https://upload.wikimedia.org/wikipedia/commons/thumb/5/54/G%C3%BCnther_Steiner_at_the_2022_Austrian_Grand_Prix.jpg/500px-G%C3%BCnther_Steiner_at_the_2022_Austrian_Grand_Prix.jpg"
>
> # 이미지 파일로 부터 질의
> answer = multimodal_llm.stream(IMAGE_URL)
> # 스트리밍 방식으로 각 토큰을 출력합니다. (실시간 출력)
> stream_response(answer)
> ```

→ 실제로 해보면 웹검색을 한 것 처럼 보인다 (배경에 대한 사전지식이 없는 상태일 것 같은 분야에도 학습한 것 같은 흔적이 보임)

→ 하지만 결국 제한된 정보에서의 추출이다보니 한계점이 보임 (인물 인식) → 텍스트 추출을 하는걸까?

> 이 이미지는 Haas F1 팀의 피트월에서 일하는 직원의 모습입니다. Haas F1 팀은 2016년에 F1에 데뷔한 미국 기반의 팀으로, 주로 중위권에서 경쟁하고 있습니다. 팀은 강력한 기술 파트너십과 효율적인 운영으로 주목받고 있으며, 드라이버와의 협력도 중요하게 여깁니다. 팀의 목표는 꾸준한 성적 향상과 포인트 획득입니다.

05. LangChain Expression Language(LCEL)

기본 예시: 프롬프트 + 모델 + 출력 파서

프롬프트 템플릿의 활용

PromptTemplate

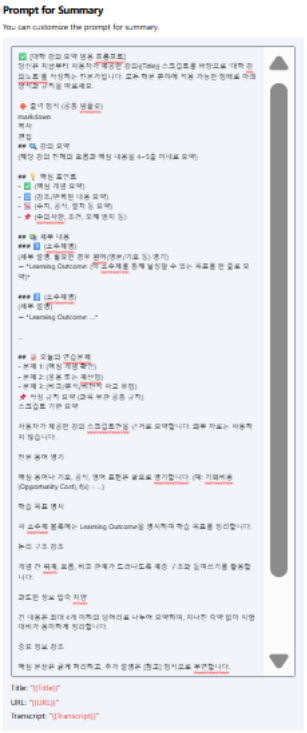
- 사용자의 입력 변수를 사용하여 완전한 프롬프트 문자열을 만드는 데 사용되는 템플릿

사용법

- `template` : 템플릿 문자열입니다. 이 문자열 내에서 중괄호 `{}` 는 변수를 나타냅니다.

> ChatGPT관련 응용프로그램에서 많이 보는 구조.

> PromptTemplate(input_variables=['country'], input_types={}, partial_variables={}, template='{country}의 수도는 어디인가요?')



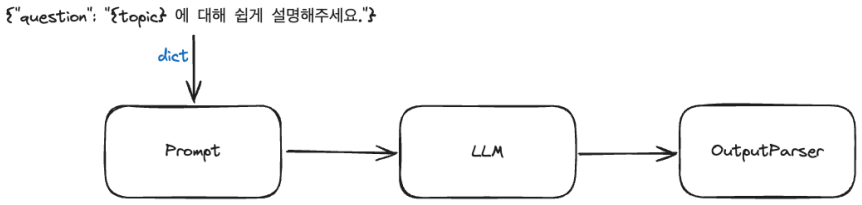
- `input_variables` : 중괄호 안에 들어갈 변수의 이름을 리스트로 정의한다.

`input_variables`

- input_variables는 PromptTemplate에서 사용되는 변수의 이름을 정의하는 리스트

Chain 생성

chain이란 무엇인가?



여기서 우리는 LCEL을 사용하여 다양한 구성 요소를 단일 체인으로 결합합니다

```
chain = prompt | model | output_parser
```

`|` 기호는 unix 파이프 연산자와 유사하며, 서로 다른 구성 요소를 연결하고 한 구성 요소의 출력을 다음 구성 요소의 입력으로 전달한다.

06. LCEL : LangChain Expression **Language(LCEL)**

사용자 정의 체인을 가능한 쉽게 만들 수 있도록, `Runnable` 프로토콜을 구현했습니다.

> Runnable 프로토콜이란?
>
> 표준 인터페이스
>
> 사용자정의 체인을 정의하고 표준 방식으로 호출하는것을 돕는다.
>
> 포함하고 있는 것
>
> - `stream`: 응답의 청크를 스트리밍합니다.
> - `invoke`: 입력에 대해 체인을 호출합니다.
> - `batch`: 입력 목록에 대해 체인을 호출합니다.
>
> 포함된 비동기 메소드
>
> - `astream`: 비동기적으로 응답의 청크를 스트리밍합니다.
> - `ainvoke`: 비동기적으로 입력에 대해 체인을 호출합니다.
> - `abatch`: 비동기적으로 입력 목록에 대해 체인을 호출합니다.
> - `astream_log`: 최종 응답뿐만 아니라 발생하는 중간 단계를 스트리밍합니다.

> 동일 주제에 대하여 stream과 invoke 메소드로 각각 출력해본 결과 드라마틱하지는 않지만 stream이 미약하게 더 빠르다

함수 `chain.astream`은 비동기 스트림을 생성하며, 주어진 토픽에 대한 메시지를 비동기적으로 처리한다.

비동기 for 루프(`async for`)를 사용하여 스트림에서 메시지를 순차적으로 받아오고, `print` 함수를 통해 메시지의 내용(`s.content`)을 즉시 출력합니다. `end=""`는 출력 후 줄바꿈을 하지 않도록 설정하며, `flush=True`는 출력 버퍼를 강제로 비워 즉시 출력되도록 한다.

> 동기와 비동기의 퍼포먼스 차이는 존재하는가? → 당장은 없어보임

Parallel: 병렬성

개별로 돌릴때랑 병렬로 돌릴때의 비용적인 차이가 유의미하게 존재하는가? 속도는 병렬로 한번에 처리하는데 두번에 나누어서 처리하는 것에 비해 확실히 빠름

배치에서의 병렬 처리

→ 병렬성을 배치로 적용해서 한번에 여러 주제에 대한 여러 작업을 수행할 수 있다.

07. Runnable

데이터를 효과적으로 전달하는 방법

- `RunnablePassthrough`는 입력을 변경하지 않거나 추가 키를 더하여 전달할 수 있다.
- `RunnablePassthrough()`가 단독으로 호출되면, 단순히 입력을 받아 그대로 전달한다.
- `RunnablePassthrough.assign(...)` 방식으로 호출되면, 입력을 받아 assign 함수에 전달된 추가 인수를 추가한다.
  - 이제는 단순히 데이터를 | 로 전달하는것이 아니라 중간에 인수를 삽입할 수 있다.

chain 을 `invoke()` 하여 실행할 때는 입력 데이터의 타입이 딕셔너리이어야 한다 (invoke에서 key값을 활용한 작업을 수행하기 때문).
`chain.invoke({"num": 5})`

→ 하지만 langchain 라이브러리가 업데이트됨에 따라 템플릿이 한개 이상 있으면 값만 전달하는 것도 가능하다

RunnableParallel

Chain 도 RunnableParallel 적용할 수 있다 → 이제 간편하게 값만 넣어도 된다.

RunnableLambda

RunnableLambda 를 사용하여 사용자 정의 함수를 맵핑하여 더 자유로운 커스텀이 가능해졌다.