이제 랭체인으로 기본적인 LLM 호출 방법을 살펴보자.

1.2 랭체인을 통한 LLM 호출

LLM은 대부분의 생성 AI 애플리케이션의 원동력이다. 랭체인은 대부분 LLM 제공업체와 상호작용하는 두 가지 인터페이스를 제공한다.

- 채팅 모델
- 기본 LLM

기본 LLM 인터페이스는 문자열 프롬프트를 입력받아 입력을 기본 LLM에 전송한 후, 다음에 이어질 표현을 예측해 출력으로 반환한다. 간단한 프롬프트를 사용해 모델을 실행하려면 다음과 같이 랭체인의 오픈AI LLM 래퍼를 가져와 invoke(호출)한다.

코드 1-1 기본 LLM 호출

Python

```
from langchain_openai.llms import OpenAI

model = OpenAI(model='gpt-3.5-turbo-instruct')

model.invoke('하늘이')
```

JavaScript

```
import { OpenAI } from '@langchain/openai';

const model = new OpenAI({ model: 'gpt-3.5-turbo-instruct' });

const response = await model.invoke('하늘이');
```

출력

```
푸릅니다!
```

TIP OpenAI에 매개변수 model을 전달했다. model은 사용할 기본 모델을 지정하는 매개변수로, LLM이나 채팅 모델 사용 시 필요한 가장 일반적인 매개변수다. 대부분의 제공업체는 성능과 비용 간 트레이드오프trade-off를 고려해 여러 모델을 제공한다(LLM은 성능이 뛰어나지만 비용이 높고 응답 속도가 느리다). 오픈AI가 제공하는 모델 목록(*https://oreil.ly/dM886*)을 확인하자. 대부분의 LLM은 다음 매개변수를 지원한다.

- temperature: 출력 생성에 사용하는 샘플링 알고리즘을 제어한다. 낮은 값(예: 0.1)은 보다 예측 가능한 결과를 만든다. 반면, 높은 값(예: 0.9)은 창의적이거나 예상치 못한 결과를 만들어낸다. 이 매개변수는 작업에 따라 서로 다른 값이 필요하다. 예를 들어, 구조화된 출력물을 생성할 때는 낮은 값을 사용하는 편이 좋다. 반면, 창의적인 글쓰기 같은 작업은 높은 값을 사용해야 더 나은 결과를 얻는다.
- max_tokens: 출력 크기와 비용을 제한한다. 낮은 값을 설정하면 LLM이 자연스러운 마무리에 도달하기 전에 출력을 중단한다.

이 외에도 모델마다 서로 다른 매개변수를 지원한다. 선택한 모델의 문서를 참고하길 권한다. 오픈AI는 매개변수를 정리한 문서(*https://oreil.ly/501RW*)를 제공한다.

기본 LLM과 달리 채팅 모델 인터페이스는 사용자와 모델이 양방향 대화를 주고받는다. 별도의 인터페이스를 제공하는 이유는 오픈AI 같이 사용자가 많은 LLM 제공업체가 메시지를 **user(사용자)**, **assistant(어시스턴트)**, **system(시스템)** 같은 역할로 구분하기 때문이다. 이때 role(역할)은 메시지의 콘텐츠 유형을 나타낸다.

- **system**: 사용자 질문에 답변하는 데 사용하는 지시 사항
- **user**: 사용자의 쿼리와 사용자가 생성한 그 밖의 모든 콘텐츠
- **assistant**: 채팅 모델이 생성한 콘텐츠

채팅 모델 인터페이스로 AI 챗봇 애플리케이션에서 구성을 쉽게 변환하고 관리할 수 있다. 다음은 랭체인의 ChatOpenAI 모델을 활용한 구현이다.[2]

코드 1-2 채팅 모델 호출

```python
from langchain_openai.chat_models import ChatOpenAI
```

2 랭체인은 80개 이상의 채팅 모델 패키지를 제공한다(2025년 4월). 지원하는 채팅 모델 목록은 *http://bit.ly/43wCzpB*에서 확인할 수 있다.

```python
from langchain_core.messages import HumanMessage

model = ChatOpenAI()
prompt = [HumanMessage('프랑스의 수도는 어디인가요?')]

model.invoke(prompt)
```

JavaScript

```javascript
import { ChatOpenAI } from '@langchain/openai';
import { HumanMessage } from '@langchain/core/messages';

const model = new ChatOpenAI();
const prompt = [new HumanMessage('프랑스의 수도는 어디인가요?')];

const response = await model.invoke(prompt);
```

출력

```
AIMessage(content='프랑스의 수도는 파리입니다.')
```

채팅 모델은 하나의 프롬프트 문자열 대신 앞서 언급한 각 역할에 따른 형태의 채팅 메시지 인터페이스를 이용한다.

- HumanMessage: 사용자 역할인 인간의 관점으로 작성한 메시지
- AIMessage: 어시스턴트 역할인 AI의 관점으로 작성한 메시지
- SystemMessage: 시스템 역할인 AI가 준수할 지침을 설정하는 메시지
- ChatMessage: 임의의 역할을 설정하는 메시지

예시에 SystemMessage로 느낌표 세 개와 함께 답변하라는 지시를 추가하겠다.

코드 1-3 시스템 메시지를 적용한 채팅 모델 호출

Python

```python
from langchain_core.messages import HumanMessage, SystemMessage
from langchain_openai.chat_models import ChatOpenAI

model = ChatOpenAI()
```

```
system_msg = SystemMessage(
    '''당신은 문장 끝에 느낌표를 세 개 붙여 대답하는 친절한 어시스턴트입니다.'''
)
human_msg = HumanMessage('프랑스의 수도는 어디인가요?')

model.invoke([system_msg, human_msg])
```

JavaScript

```
import { ChatOpenAI } from '@langchain/openai';
import { HumanMessage, SystemMessage } from '@langchain/core/messages';

const model = new ChatOpenAI();
const prompt = [
  new SystemMessage(
    '당신은 문장 끝에 느낌표를 세 개 붙여 대답하는 친절한 어시스턴트입니다.'
  ),
  new HumanMessage('프랑스의 수도는 어디인가요?'),
];

const response = await model.invoke(prompt);
```

출력

```
AIMessage('파리입니다!!!')
```

채팅 모델은 사용자 질문에는 포함하지 않았던 SystemMessage 내의 지시를 따른다. 사용자의 입력을 바탕으로 AI 애플리케이션이 비교적 예측 가능한 방식으로 응답하도록 미리 설정할 수 있다.

1.3 LLM 프롬프트 템플릿

앞서 프롬프트의 지시 사항이 출력에 미치는 영향을 확인했다. 프롬프트는 LLM이 컨텍스트를 이해하고, 질의에 적절한 답변을 생성하도록 유도한다. LLM 제공업체를 물어보는 프롬프트를 자세히 살펴보자.

컨텍스트(Context): 거대 언어 모델(LLM)은 자연어 처리(NLP) 분야의 최신 발전을 이끌고 있습니다. 거대 언어 모델은 더 작은 모델보다 우수한 성능을 보이며, NLP 기능을 갖춘 애플리케이션을 개발하는 개발자들에게 매우 중요한 도구가 되었습니다. 개발자들은 Hugging Face의 'transformers' 라이브러리를 활용하거나, 'openai' 및 'cohere' 라이브러리를 통해 OpenAI와 Cohere의 서비스를 이용하여 거대 언어 모델을 활용할 수 있습니다.

질문(Question): 거대 언어 모델은 어디서 제공하나요?

프롬프트가 단순한 문자열처럼 보이지만, LLM이 필요한 내용을 판단하고 입력에 따라 다른 결과를 내도록 만들어야 한다. 지금까지는 컨텍스트(Context)와 질문(Question)을 하드코딩했다. 그러나 이 값을 동적으로 전달할 수 있다면 어떨까?

다행히 랭체인은 동적dynamic 입력으로 프롬프트를 손쉽게 수정하는 프롬프트 템플릿 인터페이스를 제공한다. 다음과 같이 변수가 들어갈 위치를 중괄호로 표기해 템플릿을 작성하자.

코드 1-4 프롬프트 템플릿 적용

Python

```
from langchain_core.prompts import PromptTemplate

template = PromptTemplate.from_template('''아래 작성한 컨텍스트(Context)를 기반으로
    질문(Question)에 대답하세요. 제공된 정보로 대답할 수 없는 질문이라면 "모르겠어요." 라
    고 답하세요.

Context: {context}

Question: {question}

Answer: ''')

template.invoke({
    'context': '''거대 언어 모델(LLM)은 자연어 처리(NLP) 분야의 최신 발전을 이끌고 있습니다.
    거대 언어 모델은 더 작은 모델보다 우수한 성능을 보이며,
    NLP 기능을 갖춘 애플리케이션을 개발하는 개발자들에게
    매우 중요한 도구가 되었습니다. 개발자들은 Hugging Face의 'transformers' 라이브러리를
    활용하거나, 'openai' 및 'cohere' 라이브러리를 통해 OpenAI와 Cohere의 서비스를 이용하여
    거대 언어 모델을 활용할 수 있습니다.
```

```
    ''',
    'question': '거대 언어 모델은 어디서 제공하나요?'
})
```

JavaScript

```
import { PromptTemplate } from '@langchain/core/prompts';

const template =
  PromptTemplate.fromTemplate('아래 작성한 컨텍스트(Context)를 기반으로 질문(Question)
  에 대답하세요. 제공된 정보로 대답할 수 없는 질문이라면 "모르겠어요." 라고 답하세요.

Context: {context}

Question: {question}

Answer: ');

const response = await template.invoke({
  context:
    '거대 언어 모델(LLM)은 자연어 처리(NLP) 분야의 최신 발전을 이끌고 있습니다. 거대 언어
    모델은 더 작은 모델보다 우수한 성능을 보이며, NLP 기능을 갖춘 애플리케이션을 개발하는
    개발자들에게 매우 중요한 도구가 되었습니다. 개발자들은 Hugging Face의 'transformers'
    라이브러리를 활용하거나, 'openai' 및 'cohere' 라이브러리를 통해 OpenAI와 Cohere의 서
    비스를 이용하여 거대 언어 모델을 활용할 수 있습니다.',
  question: '거대 언어 모델은 어디서 제공하나요?',
});
```

출력

```
StringPromptValue(text= '아래 작성한 컨텍스트(Context)를 기반으로
    질문(Question)에 대답하세요. 제공된 정보로 대답할 수 없는 질문이라면 "모르겠어요."
    라고 답하세요.

Context: 거대 언어 모델(LLM)은 자연어 처리(NLP) 분야의 최신 발전을 이끌고 있습니다.
    거대 언어 모델은 더 작은 모델보다 우수한 성능을 보이며,
    NLP 기능을 갖춘 애플리케이션을 개발하는 개발자들에게
    매우 중요한 도구가 되었습니다. 개발자들은 Hugging Face의 'transformers' 라이브러
    리를 활용하거나, 'openai' 및 'cohere' 라이브러리를 통해 OpenAI와 Cohere의 서비스
    를 이용하여 거대 언어 모델을 활용할 수 있습니다.
```

```
Question: 거대 언어 모델은 어디서 제공하나요?

Answer: ')
```

이 예시는 앞서 사용한 정적static 프롬프트를 동적 프롬프트로 만들었다. template은 최종 프롬프트의 구조와 동적 입력이 삽입될 위치에 대한 정의로 구성된다. 따라서 템플릿은 여러 개의 정적이고 구체적인 프롬프트를 제작하는 재료로 사용할 수 있다. 특정 값을 사용해 프롬프트를 채워 LLM에 입력할 정적 프롬프트를 생성할 수 있다. 지금 같은 경우는 context와 question에 값이 추가된 정적 프롬프트가 생성된다.

보다시피, invoke 메서드를 통해 question 인자를 동적으로 전달한다. 랭체인 프롬프트는 파이썬의 f-string 문법으로 동적 프롬프트를 구성한다. 중괄호로 둘러싸인 변수(예: {question})를 런타임에 전달되는 값으로 대체할 플레이스홀더로 사용한다. 이전 예시에서, {question}는 "거대 언어 모델은 어디서 제공하나요?"로 바뀐다. 랭체인을 이용해 오픈AI의 LLM에 동적 프롬프트를 입력하는 방법을 살펴보자.

코드 1-5 동적 프롬프트

Python

```
from langchain_openai.llms import OpenAI
from langchain_core.prompts import PromptTemplate

# 'template'과 'model'은 언제든 다시 쓸 수 있다

template = PromptTemplate.from_template('''아래 작성한 컨텍스트(Context)를 기반으로
    질문(Question)에 대답하세요. 제공된 정보로 대답할 수 없는 질문이라면 "모르겠어요." 라
    고 답하세요.

Context: {context}

Question: {question}

Answer: ''')

model = OpenAI()
```