구글 코랩 무료 버전에서 Ollama 시작하기

환경 구축하기

- 1. GPU 설정하기
  - 런타임 -> 런타임 유형 변경 -> T4 GPU
- 2. 코랩에서 터미널 기능을 사용하기 위한 패키지 설치 및 colabxerm 확장을 로드

```
# 코랩 내에서 터미널 기능을 제공하는 패키지 설치
pip install colab-xterm

# 코랩 노트북에 colabxterm 확장을 로드.
# %load_ext는 IPython 명령어로 특정 확장을 활성화
%load_ext colabxterm
```

In [ ]:
```
!pip install colab-xterm
```

In [ ]:
```
%load_ext colabxterm
```

Ollama 는 무엇일까요?

- 오픈 소스 로컬 대규모 언어 모델(LLM) 실행 도구
- Ollama를 사용하면 개인 컴퓨터에서 LLM을 직접 실행 가능하다.
- 다양한 모델을 지원한다. Llama 2, 3, Mistral, Gemma 2, Qwen2.5 등 여러 오픈 소스 LLM 지원
- 모델을 쉽게 다운로드하고 실행이 가능하다.
- https://ollama.com/

Ollama 서비스 및 설치

- 구글 코랩 내에서 Xterm 실행
- Ollama 설치하기

  `curl https://ollama.ai/install.sh | sh`
  - Ollama 웹 사이트에서 설치 스크립트를 다운로드하여 실행.
  - 필요한 다운로드 및 설치를 하고 설치 프로세스를 자동으로 처리
- Ollama 서버 시작하기

  `ollama serve &`
  - ollama 한번 설치하면, 우리는 명령으로 서버를 시작할 수 있다.
  - ollama serve &
  - '&'는 백그라운드에서 실행한다. 터미널 계속 사용 가능.
- AI 모델 가져오기

    - Ollama 서버가 실행 중이므로 서버에 사용할 AI 모델을 가져올 수 있다. Mistral 모델을 예로 들어본다.

```
ollama pull mistral
```

- AI 모델 설치 확인

```
ollama --version
```

Model library

Model	Parameters	Size	Download Command
Llama 3.2	3B	2.0GB	`ollama run llama3.2`
Llama 3.2	1B	1.3GB	`ollama run llama3.2:1b`
Llama 3.1	8B	4.7GB	`ollama run llama3.1`
Llama 3.1	70B	40GB	`ollama run llama3.1:70b`
Llama 3.1	405B	231GB	`ollama run llama3.1:405b`
Phi 3 Mini	3.8B	2.3GB	`ollama run phi3`
Phi 3 Medium	14B	7.9GB	`ollama run phi3:medium`
Gemma 2	2B	1.6GB	`ollama run gemma2:2b`
Gemma 2	9B	5.5GB	`ollama run gemma2`
Gemma 2	27B	16GB	`ollama run gemma2:27b`
Mistral	7B	4.1GB	`ollama run mistral`
Moondream 2	1.4B	829MB	`ollama run moondream`
Neural Chat	7B	4.1GB	`ollama run neural-chat`
Starling	7B	4.1GB	`ollama run starling-lm`
Code Llama	7B	3.8GB	`ollama run codellama`
Llama 2 Uncensored	7B	3.8GB	`ollama run llama2-uncensored`
LLaVA	7B	4.5GB	`ollama run llava`
Solar	10.7B	6.1GB	`ollama run solar`

필수 라이브러리 설치

```
pip install langchain_community
pip install langchain_core
pip install -U langchain-ollama
pip install langchain_core
```

In [ ]:
```
!pip install langchain_core
```

xterm 띄우기

In [ ]:
```
%xterm
```

- 명령어 정리

```
curl https://ollama.ai/install.sh | sh
ollama serve &
ollama pull mistral
```

- LLM 모델 다운로드

다운 가능한 모델 리스트 : https://ollama.com/library

```
ollama pull llama3
ollama pull gemma2:9b
ollama pull llava:7b
ollama pull bakllava
```

- 설치된 LLM 모델 실행

```
ollama run gemma2:9b
ollama run llama3
```

- 두 개의 모델을 실행하면 2개의 모델이 함께 실행되므로 이를 멈추고, 하나만 실행하는 게 좋다.

프로세스 확인

In [ ]:
```
!ps aux | grep ollama
```

In [ ]:
```
!ollama list
```

mistral 활용해 보기

- Mistral은 고성능의 대형 언어 모델로, 자연어 처리(NLP) 작업에 최적화.
- 프랑스의 언어모델. 23년 10월 매개변수가 73억개의 미스트랄 7B오픈
- 발표당시 매개변수 130억개의 Llama 2인 성능을 능가했다고 함.
- 미스트랄 Github : https://github.com/mistralai/mistral-inference

In [ ]:
```
%%time

from langchain_core.prompts import ChatPromptTemplate
from langchain_ollama.llms import OllamaLLM

template = """Question: {question}
Answer: Let's think step by step."""

# ChatPromptTemplate를 사용하여 템플릿을 생성합니다.
prompt = ChatPromptTemplate.from_template(template)

# 모델 초기화
```

```
model = OllamaLLM(model="mistral")

# 체인을 생성하여 질문과 답변을 연결합니다.
chain = prompt | model

# 체인을 실행하여 질문에 대한 답변을 얻습니다.
chain.invoke({"question": "LLM 모델에는 어떤 것이 있을까? 한글 답변 가능하니?"})
```

Gemma2를 이용한 번역기 만들기

In [ ]:

```
from langchain_community.llms import Ollama
from langchain.memory import ConversationSummaryBufferMemory
from langchain.schema.runnable import RunnablePassthrough
from langchain.prompts import ChatPromptTemplate, MessagesPlaceholder

# Ollama 모델 초기화
llm = Ollama(model="gemma2:9b")

# 대화 메모리 설정
memory = ConversationSummaryBufferMemory(
    llm=llm,
    max_token_limit=80,
    memory_key="chat_history",
    return_messages=True,
)

# 메모리 로드 함수
def load_memory(input):
    return memory.load_memory_variables({})["chat_history"]

# 프롬프트 템플릿 설정
prompt = ChatPromptTemplate.from_messages([
    ("system", f"너는 사용자 메시지를 한글을 영어로 바꾸는 훌륭한 번역가란다."),
    MessagesPlaceholder(variable_name="chat_history"),
    ("human", "{question}"),
])

# 체인 설정
chain = RunnablePassthrough.assign(chat_history=load_memory) | prompt | llm

# 체인 호출 함수
def invoke_chain(question):
    result = chain.invoke({"question": question})
    memory.save_context(
        {"input": question},
        {"output": result},
    )
    print("번역 결과:", result)

while True :
    print("저는 당신이 입력한 것을 잘 번역할 수 있어요.")
    userInput = input("입력해 주세요.(종료 : q or quit)")

    if userInput.lower() in ['q', 'quit']:
        print("프로그램을 종료합니다.")
        break

    invoke_chain(userInput)
```

```
ollama serve &
ollama list
ollama pull llava:7b
ollama pull bakllava
ollama list
```

멀티 모달 시스템

Bakllava 모델

- Bakllava는 최근의 멀티모달 언어 모델로, 텍스트와 이미지를 동시에 처리할 수 있도록 설계되었습니다. 이 모델은 다양한 자연어 처리(NLP)와 컴퓨터 비전 작업을 통합하여 수행할 수 있는 능력을 가지고 있습니다.
- 특징
  - 멀티모달 능력: 텍스트와 이미지 데이터를 동시에 처리하고 이해할 수 있습니다.
  - 대규모 데이터 학습: Bakllava 모델은 대규모 텍스트 및 이미지 데이터셋을 사용하여 학습되었기 때문에, 다양한 도메인에서 높은 성능을 발휘합니다.
  - 응용 분야: 이미지 캡션 생성, 비주얼 질문 응답, 이미지 설명 등 다양한 응용 분야에서 사용됩니다.

LLaVa 모델

- Stanford University의 연구팀에 의해 개발에서 개발된 멀티모달 모델로, 텍스트와 이미지를 결합하여 복잡한 질문에 대답할 수 있는 능력

In [ ]:
```python
# 필요한 라이브러리 임포트
import base64
from io import BytesIO
from IPython.display import HTML, display
from PIL import Image
from langchain_core.output_parsers import StrOutputParser
from langchain_community.chat_models import ChatOllama
from langchain_core.messages import HumanMessage

def image_to_base64(img_path):
    """
    이미지 파일을 Base64 인코딩된 문자열로 변환하는 함수

    :param img_path: 이미지 파일 경로
    :return: Base64로 인코딩된 이미지 문자열
    """
    with Image.open(img_path) as img:
        buffer = BytesIO()  # 메모리 내 바이트 스트림 생성
        img.save(buffer, format="JPEG")  # 이미지를 JPEG 형식으로 버퍼에 저장
        return base64.b64encode(buffer.getvalue()).decode("utf-8")  # Base64로 인

def display_base64_image(b64_str):
    """
    Base64로 인코딩된 이미지를 IPython 환경에서 표시하는 함수
```

```
    :param b64_str: Base64로 인코딩된 이미지 문자열
    """
    display(HTML(f'<img src="{b64_str}" />'))  # HTML 태그

def create_prompt(input_data):
    """
    LLM에 전달할 프롬프트를 생성하는 함수

    :param input_data: 이미지와 텍스트 정보를 포함한 딕셔너리
    :return: LLM에 전달할 메시지 객체 리스트
    """
    image_part = {
        "type": "image_url",
        "image_url": f"{input_data['image']}",  # Base64
    }
    text_part = {"type": "text", "text": input_data["text"]}  # 텍스트 부분
    return [HumanMessage(content=[image_part, text_part])]  # 이미지와 텍스트를

# 이미지 처리 및 표시
image_path = "./Dog_rawPixel01.jpg"  # 처리할 이미지 파일 경로
base64_image = image_to_base64(image_path)  # 이미지를 Base64로 인코딩
display_base64_image(base64_image)  # 인코딩된 이미지 표시
```

In [ ]:
```
import base64
from io import BytesIO
from IPython.display import HTML, display
from PIL import Image
from langchain_core.output_parsers import StrOutputParser
from langchain_community.chat_models import ChatOllama
from langchain_core.messages import HumanMessage

def image_to_base64(img_path):
    with Image.open(img_path) as img:
        buffer = BytesIO()
        img.save(buffer, format="JPEG")
        return base64.b64encode(buffer.getvalue()).decode("utf-8")

def display_base64_image(b64_str):
    display(HTML(f'<img src="{b64_str}" />'))

def create_prompt(input_data):
    image_part = {
        "type": "image_url",
        "image_url": f"{input_data['image']}",
    }
    text_part = {"type": "text", "text": input_data["text"]}
    return [HumanMessage(content=[image_part, text_part])]

# 이미지 처리
image_path = "./Dog_rawPixel01.jpg"
base64_image = image_to_base64(image_path)
display_base64_image(base64_image)
```

In [ ]:
```
from langchain_core.output_parsers import StrOutputParser
from langchain_community.chat_models import ChatOllama
from langchain_core.messages import HumanMessage

# ChatOllama 멀티모달 언어 모델을 불러옵니다.
```

```
multimodal_model = ChatOllama(model="llava:latest", temperature=0)

# 프롬프트 함수, 언어 모델, 출력 파서를 연결하여 체인을 생성합니다.
processing_chain = create_prompt | multimodal_model | StrOutputParser()

# 체인을 호출하여 쿼리를 실행합니다.
result = processing_chain.invoke(
    # 텍스트와 이미지를 전달합니다.
    {"text": "Describe the image", "image": base64_image}
)

print(result)  # 쿼리 결과를 출력합니다.
```

REF

- https://medium.com/@abonia/running-ollama-in-google-colab-free-tier-545609258453