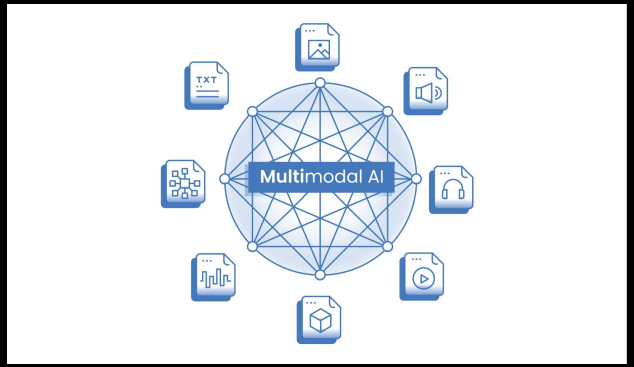
14장. MultiModal LLM

LLM을 활용한 실전 AI 애플리케이션 개발

실습 발표: 니콜

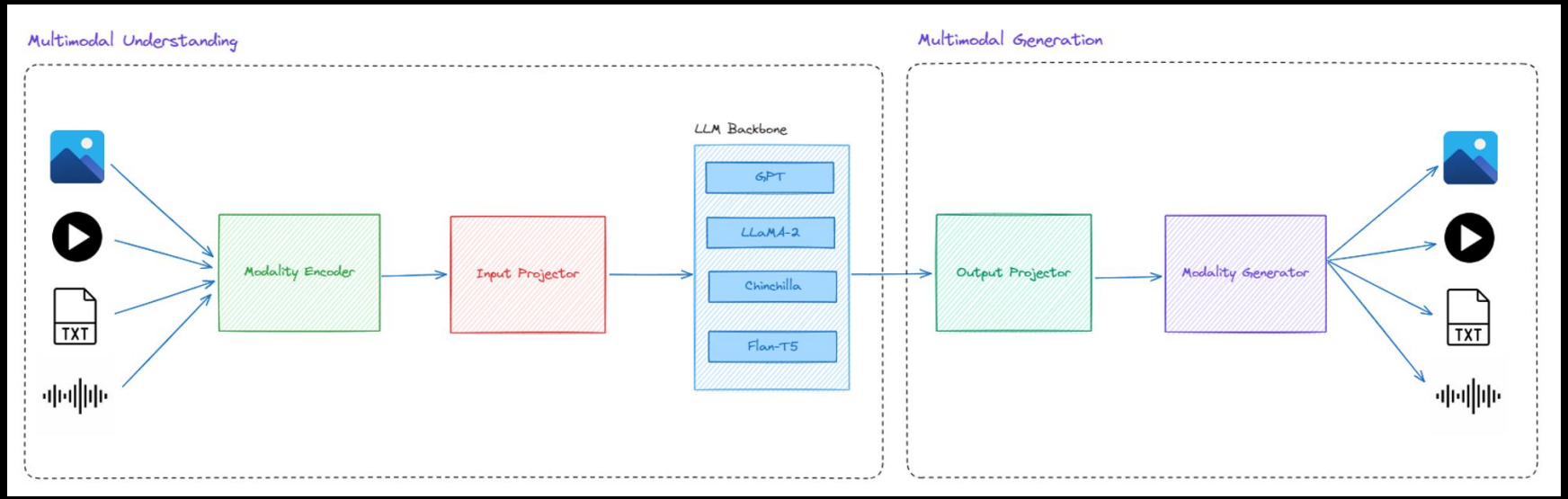
멀티 모달(MultiModal) LLM이란?



- 텍스트뿐만 아니라 이미지, 비디오, 오디오, 3D 등 다양한 형식의 데이터를 이해하고 생성할 수 있는 LLM

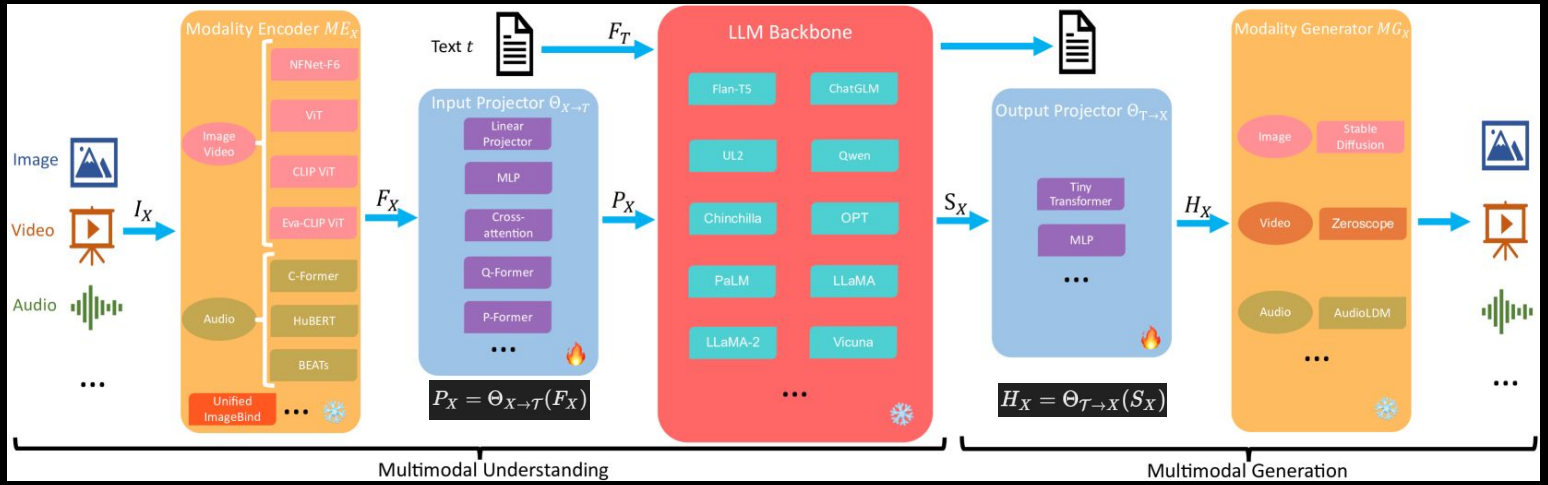
- (2024년 기준)
  - 텍스트와 이미지를 처리하는 멀티 모달 LLM이 가장 활발히 연구되고 있다.
  - 멀티 모달 생성보다는 멀티 모달 이해 성능을 높이기 위한 기술 개발에 집중

멀티 모달 LLM의 구성요소 (1)



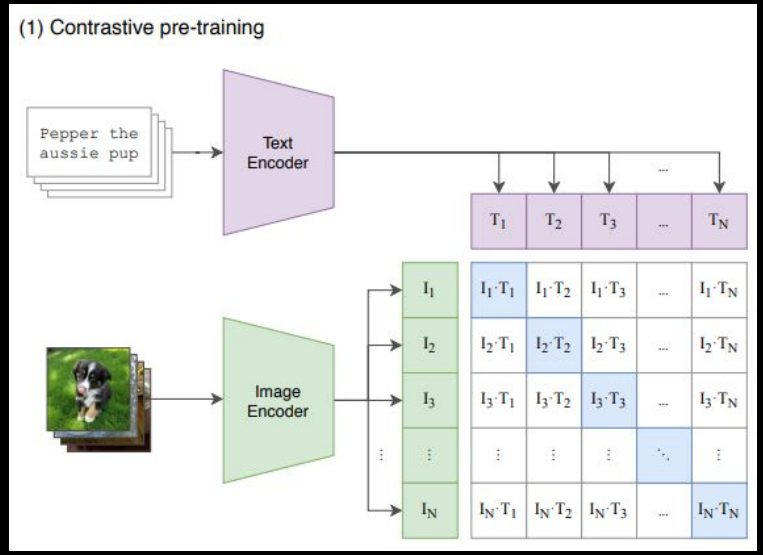
- 모달리티 인코더
- 입력 프로젝터
- LLM 백본
- 출력 프로젝터
- 모달리티 생성기

멀티 모달 LLM의 구성요소 (2)



- 모달리티 인코더: 이미지, 비디오, 오디오 등의 데이터 형식을 처리하기 위해 학습된 사전 학습 모델
  - 시각: NFNet-F6, ViT (Vision Transformer), CLIP ViT
  - 오디오: C-Former
  - 3D 포인트 클라우드(3D Point Cloud): ULIP-2
- 입력 프로젝터: 이미지 임베딩을 LLM 백본이 이해할 수 있는 텍스트로 변환
- LLM 백본: 다양한 모달리티에서 얻은 표현을 처리.
  - 의미 이해(semantic understanding), 추론(reasoning), 입력에 대한 결정(decision-making regarding inputs)을 내리는 과정에 참여.
- 출력 프로젝터: 모델이 내부적으로 생성한 정보를 사용자가 이해할 수 있는 형태로 변환하는 역할
- 모달리티 생성기: 모델이 특정한 형식(모달리티)의 데이터를 만들어내는데 사용

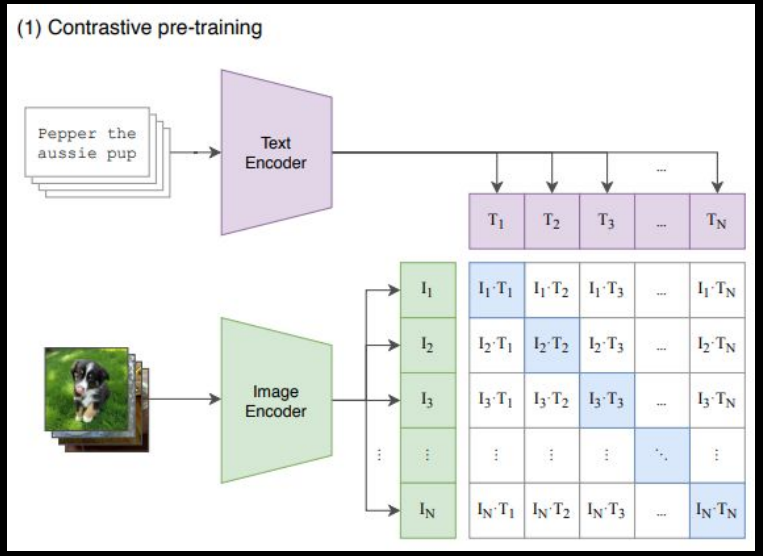
CLIP

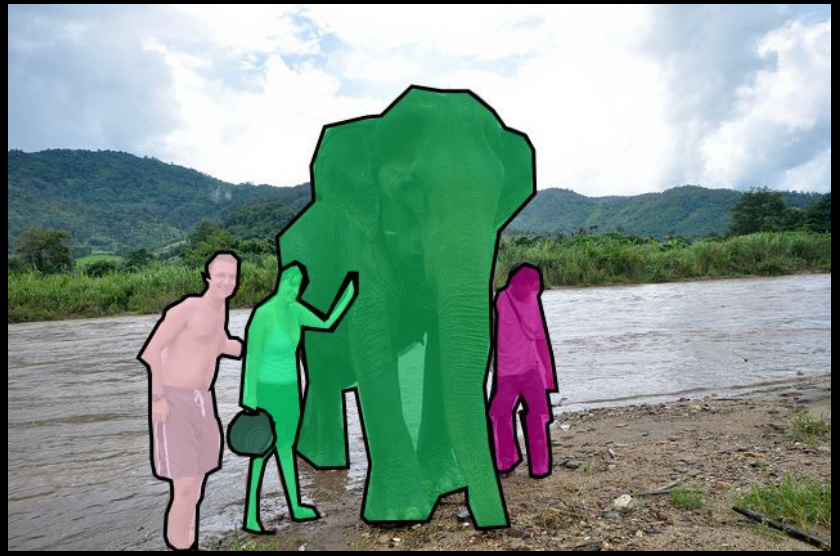


- 텍스트 데이터와 이미지 데이터의 관계를 계산할 수 있도록 텍스트 모델과 이미지 모델을 함께 학습시킨 모델

- Clip은 텍스트 모델과 이미지 모델 2개로 구성
- 텍스트와 이미지 데이터의 관계를 계산 -> 이미지와 텍스트의 유사성 계산, 검색/분류에 활용

CLIP 모델의 학습 방법 (1)

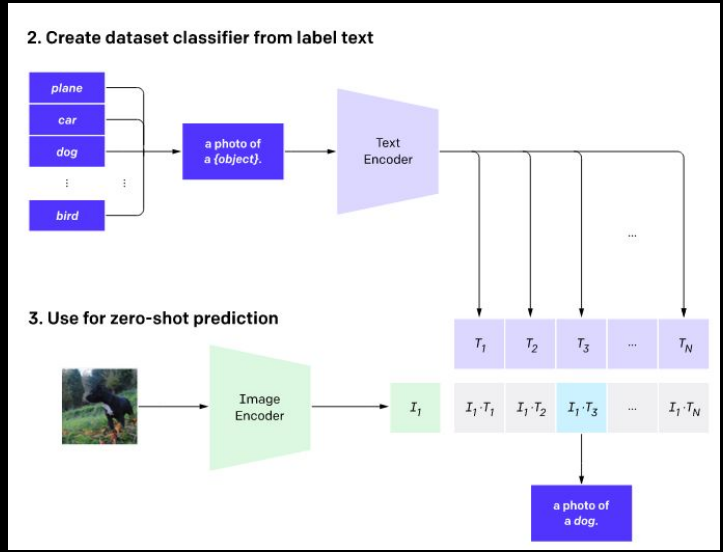




- 대조 학습(Constrastive pre-training)을 통해 모델을 학습시킨다.
  - 매칭되는 쌍의 유사도는 커지고, 그렇지 않은 쌍의 유사도는 작아지도록 학습

- 텍스트 인코더로는 트랜스포머 모델을 활용하고,
  이미지 인코더로는 ViT(Visual Transformer), ResNet 등의 이미지 모델을 활용.

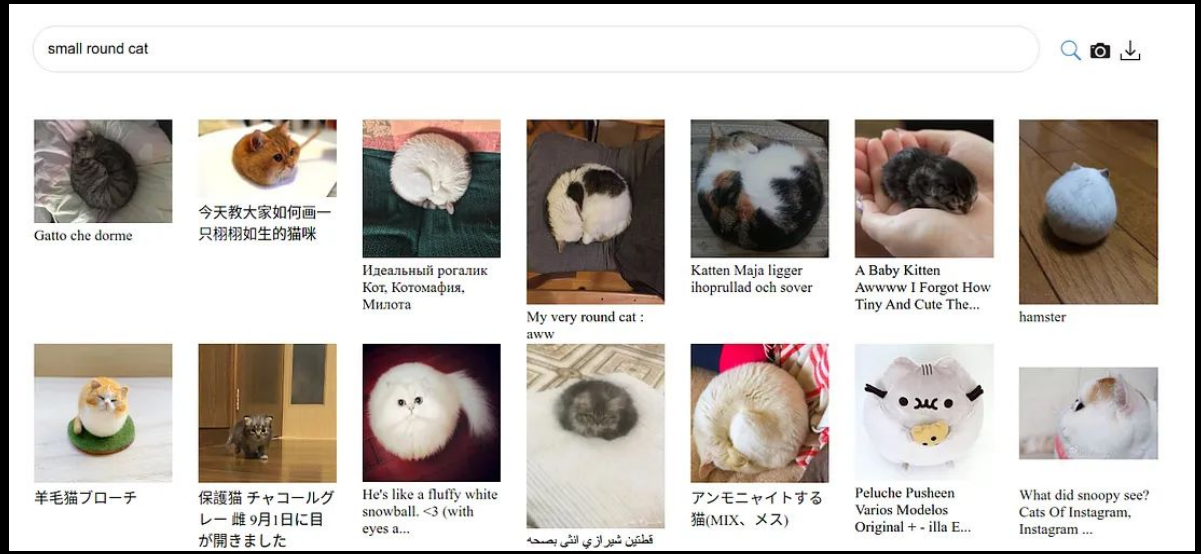
CLIP 모델의 학습 방법 (2)



- 제로샷 추론(zero-shot prediction)을 수행
  - 사전 학습 데이터 이외에 특정 작업을 위한 데이터로 미세 조정하지 않은 상태에서 추론을 수행

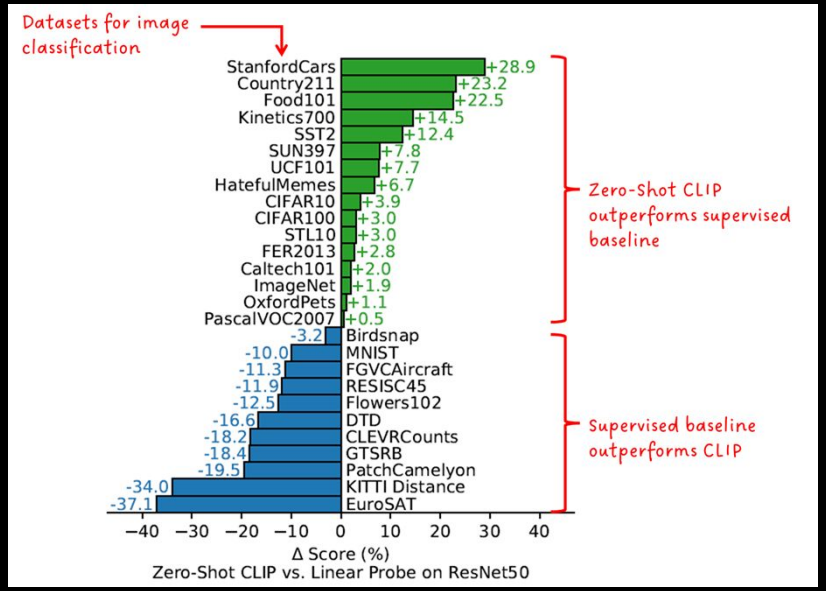
1. 레이블이 있는 데이터셋에서 레이블을 “A photo of a {객체}”로 변경하는 프롬프트 엔지니어링을 수행
2. 입력 텍스트는 학습된 텍스트 인코더를 사용해 텍스트 임베딩으로 만들고 이미지는 학습한 이미지 인코더를 사용해 이미지 임베딩으로 만든다.
3. 임베딩 사이의 유사도가 가장 큰 인덱스가 추론 결과가 된다.

CLIP 모델을 활용한 이미지 검색



- 이미지와 텍스트 데이터의 유사도 계산을 활용해서 이미지 검색에도 활용할 수 있는데요.
  - 이미지 검색: 이미지와 텍스트의 유사도를 기반으로 텍스트를 입력했을 때 유사한 이미지를 찾는 기능

CLIP 모델의 제로샷 추론 성능은?

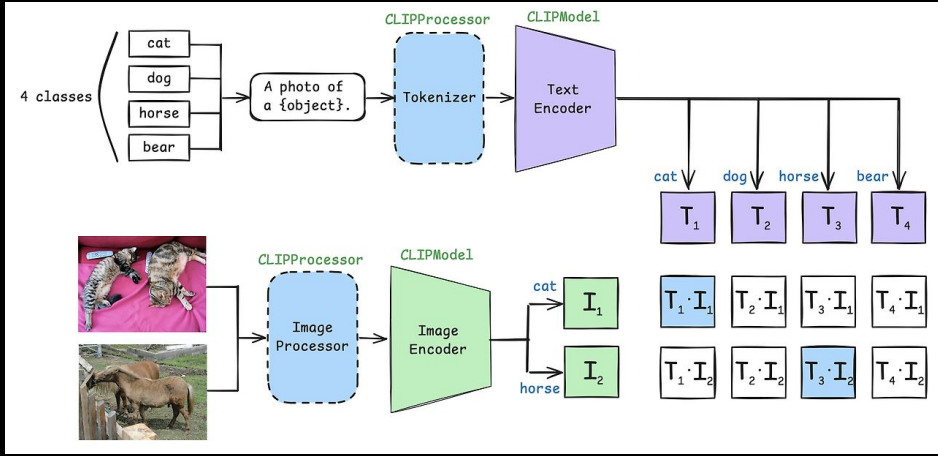


27개의 데이터셋에 대해 지도학습한 ResNet50 모델과 성능 비교

- 16개의 데이터셋에서 해당 데이터셋을 전혀 학습하지 않은 CLIP 모델의 성능이 더 높았다.
- STL10 데이터셋에서는 SOTA 성능을 달성했다.

CLIP 모델 활용법 (1)

이미지 분류 (Image Classification)



각 component 위에 있는 초록색 글씨는 이를 처리하는 class

- 4개의 class {cat, dog, horse, bear} 에 대해 분류를 하는 task로
  각 class에 대한 text와 이미지의 embedding 값을 사용하여 이미지 분류를 진행한다.

- '고양이'와 '말' 이미지를 Image Encoder의 입력으로 넣기 때문에,
  I1에 대해서는 T1·I1의 값이 가장 높게 나와야하고, I2에 대해서는 T3·I2의 값이 가장 높게 나와야 한다.

CLIP 모델 활용법 (2)

이미지 분류 (Image Classification)

1. Load pre-trained model

```python
from transformers import CLIPModel, CLIPProcessor

model = CLIPModel.from_pretrained("openai/clip-vit-base-patch32")
processor = CLIPProcessor.from_pretrained("openai/clip-vit-base-patch32")
```

- HuggingFace Transformer 라이브러리를 통해 CLIP 모델을 활용.

- 모델과 데이터 처리 프로세서를 불러온다.
  - CLIPProcessor: 입력 데이터의 전처리를 담당
    - CLIPImageProcessor와 CLIPTokenizer를 wrapping한 class
    - CLIPImageProcessor는 이미지 처리(e.g. resize, normalization)를 담당한다.
    - CLIPTokenizer은 text tokenizing을 담당한다.
    - 이에 대응하는 image_processor, tokenizer 필드를 갖고있다.
  - CLIPModel: Text Encoder와 Image Encoder를 갖는 임베딩 모델
    - 이에 대응하는 text_model, vision_model 필드를 갖고 있다.

CLIP 모델 활용법 (3)

2. Load image

```python
from PIL import Image
import requests

def get_image(url):
    return Image.open(requests.get(url, stream=True).raw)

cat_image = get_image("http://images.cocodataset.org/val2017/000000039769.jpg")
horse_image = get_image("https://farm6.staticflickr.com/5465/8929343165_e34cf36bce_z.jpg")
```

- 이미지 URL을 통해 가져와 모델에 입력으로 넣어주고, requests 라이브러리를 활용해서 url 주소의 이미지를 가져오고 PIL 라이브러리를 통해 이미지를 읽는다.

CLIP 모델 활용법 (4)

3. Get model inputs

```python
text = ["a photo of a cat", "a photo of a dog", "a photo of a horse", "a photo of a bear"]
inputs = processor(text=text,
                   images=[cat_image, horse_image],
                   return_tensors="pt",
                   padding=True)

print("inputs.keys()")
print(inputs.keys())

print('\ninputs["input_ids"]')
print(inputs["input_ids"])

print('\ninputs["attention_mask"]')
print(inputs["attention_mask"])

print('\ninputs["pixel_values"].shape')
print(inputs["pixel_values"].shape) # 원본 이미지 크기는 (640, 480)
```

```
inputs.keys()
dict_keys(['input_ids', 'attention_mask', 'pixel_values'])

inputs["input_ids"]
tensor([[49406,   320,  1125,   539,   320,  2368,   269, 49407],
        [49406,   320,  1125,   539,   320,  1929,   269, 49407],
        [49406,   320,  1125,   539,   320,  4558,   269, 49407],
        [49406,   320,  1125,   539,   320,  4298,   269, 49407]])

inputs["attention_mask"]
tensor([[1, 1, 1, 1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1, 1, 1, 1]])

inputs["pixel_values"].shape
torch.Size([2, 3, 224, 224])
```

- 다음으로 CLIP 모델의 입력을 만들기 위해 CLIPProcessor를 사용한다.
- 4개의 class에 대한 text와 앞서 가져온 이미지를 CLIPProcessor의 입력으로써 사용한다.

- CLIPProcessor의 출력값은 CLIPModel의 입력값이 된다.
- 이 값은 dict로 다음과 같은 값이 들어있다.
  - input_ids: tokenized text
  - attention_mask: Mask to avoid performing attention on padding token indices
  - pixel_values: resized & normalized image

CLIP 모델 활용법 (5)

4. Inference (추론)

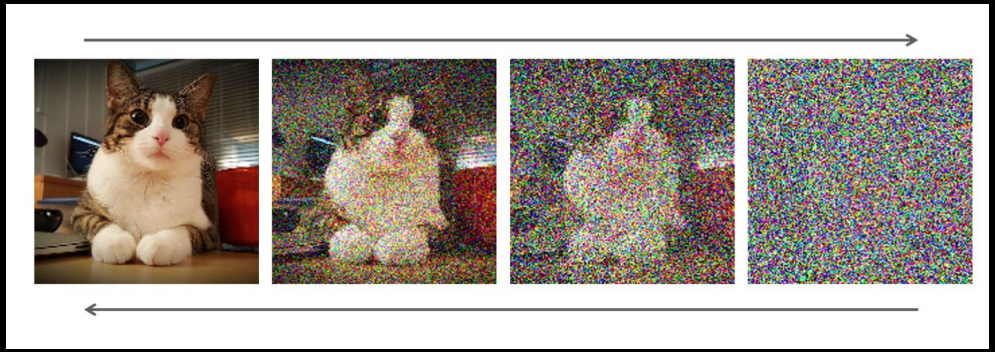
```python
import torch

with torch.no_grad():
    outputs = model(**inputs)

print("\nimage-text similarity score:")
logits_per_image = outputs.logits_per_image  # image-text similarity score
print(logits_per_image.numpy())

probs = logits_per_image.softmax(dim=1)  # take the softmax to get the label probabilities
print("\nlabel probability(softmax):")
print(probs.numpy())

print("\npred labels:")
print(probs.argmax(dim=1).numpy())
```

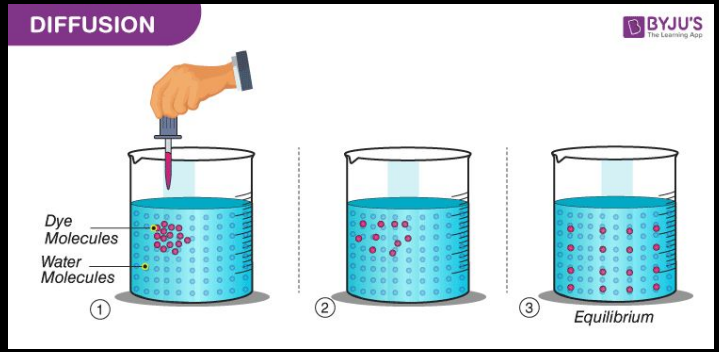
```
image-text similarity score:
[[24.884726 19.275553 18.02513  17.776133]
 [21.128246 23.476625 30.162859 22.668709]]

label probability(softmax):
[[9.9449903e-01 3.6439428e-03 1.0435652e-03 8.1354514e-04]
 [1.1898247e-04 1.2455784e-03 9.9808013e-01 5.5526127e-04]]

pred labels:
[0 2]
```

- 모델의 입력값을 사용하여 classification을 진행한다.
- 출력된 값의 logit을 통해 해당 이미지의 class를 예측할 수 있다.
- softmax를 활용해서 모델의 출력값을 비교.

- 결과
- (2x4) 차원을 갖는 것을 확인할 수 있다.
- 높은 정확도로 class를 예측하는 것을 확인할 수 있다.

Diffusion 모델



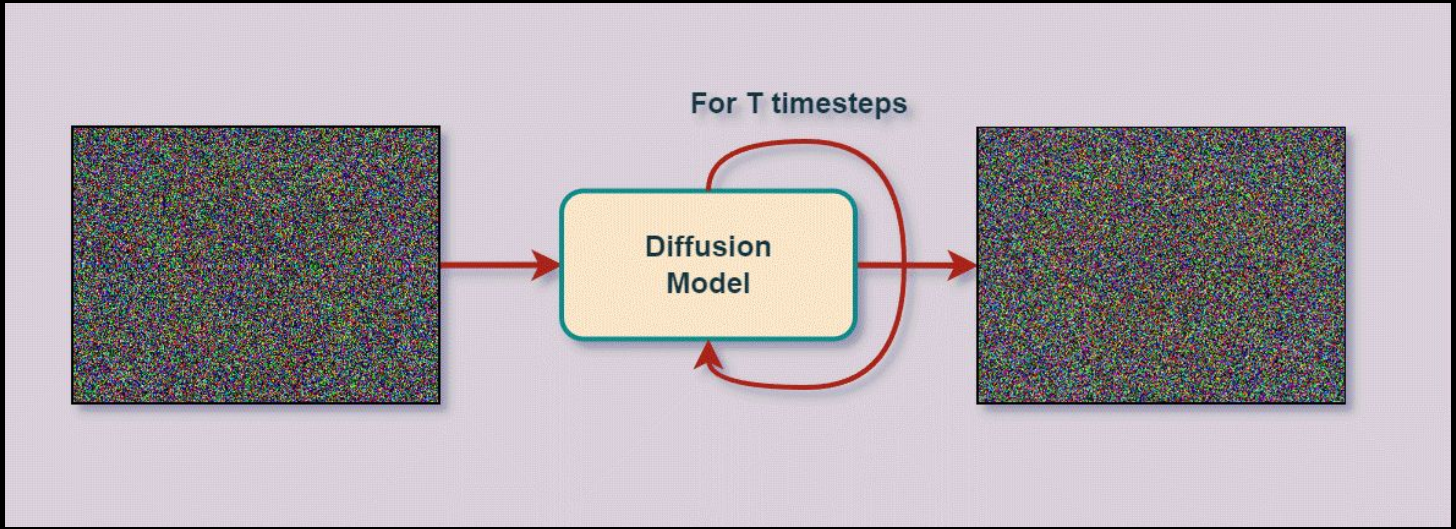


- 확산 현상에서 영감을 받아 만들어진 생성 모델
  - 확산: 물질이 농도가 높은 곳에서 낮은 곳으로 이동하는 현상

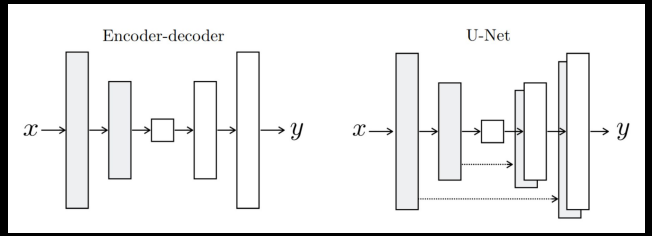
- 물에 떨어뜨린 물감이 퍼지는 것과 같이 분자가 확산되는 과정에서 아이디어를 얻은 개념
- 이미지의 '입자'들이 흩어지는 과정을 모델링할 수 있다면,
  그 반대로 노이즈로부터 이미지를 생성하는 것도 가능하겠다는 생각에 착안해서 만들어진 모델

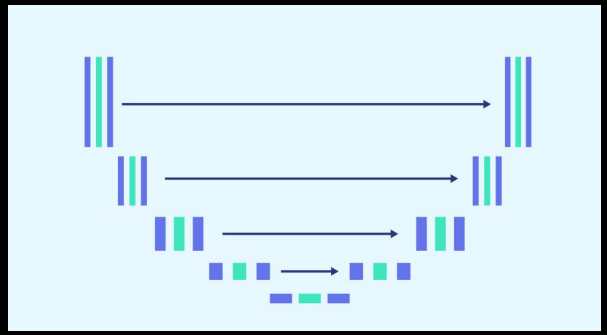
- U-Net이라는 인코더 디코더 모델을 많이 활용

Diffusion 모델



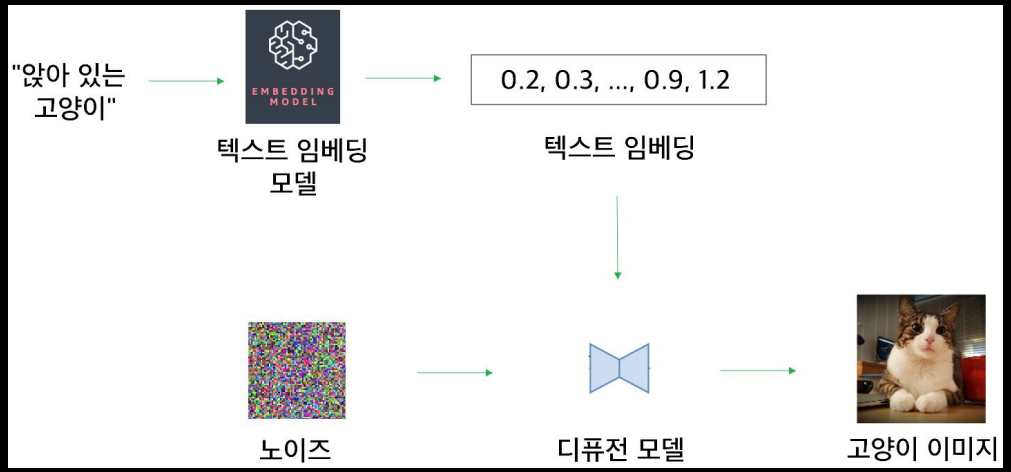
U-Net 인코더-디코더





- 인코더 디코더: 입력 데이터의 차원을 낮추는 인코딩 단계와
  차원을 높이는 디코딩 단계를 통해 데이터의 의미를 압축하기 위해 사용되는 모델 구조
- 장점: 이미지의 위치 정보 손실 방지 가능
- 디퓨전 모델로 많이 활용됨

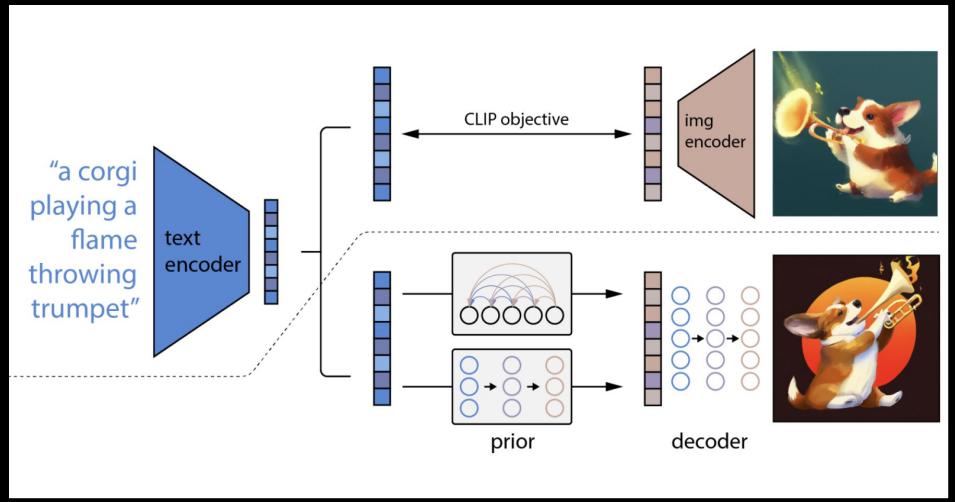
Diffusion 모델로 원하는 이미지 생성



- 디퓨전 모델에 노이즈를 넣어주면서 원하는 결과물의 형태를 텍스트 임베딩으로 변환해서 디퓨전 모델에 함께 입력으로 넣어줌.

  -> 디퓨전 모델이 텍스트 임베딩을 참고해서 원하는 이미지를 생성

DALL-E 모델



- CLIP 모델을 활용해서 텍스트 임베딩을 만든다.
- 텍스트 임베딩을 활용해서 두 단계를 거쳐서 이미지를 생성.

1. CLIP의 텍스트 인코더를 사용해서 입력한 텍스트를 텍스트 임베딩으로 만든다.
2. Prior 모델을 통해 CLIP 이미지 임베딩을 만들고 디코더를 사용해서 이미지를 생성한다.

    - Prior 모델: 텍스트 임베딩을 입력으로 받아 이미지 임베딩을 예측하는 디퓨전 모델
    - 디코더 모델: 이미지 임베딩을 참조해 이미지를 생성하는 디퓨전 모델

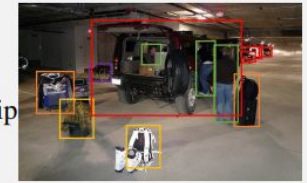
LLaVA

- 이미지를 인식하는 CLIP 모델과 LLM을 결합해 모델이 이미지를 인식하고 그 이미지에 대한 텍스트를 생성
- CLIP 모델과 DALL-E 모델의 한계 극복

LLaVA의 학습 데이터

Context type 1: Captions

A group of people standing outside of a black vehicle with various luggage.
Luggage surrounds a vehicle in an underground parking area
People try to fit all of their luggage in an SUV.
The sport utility vehicle is parked in the public garage, being packed for a trip
Some people with luggage near a van that is transporting it.



Context type 2: Boxes

person: [0.681, 0.242, 0.774, 0.694], person: [0.63, 0.222, 0.686, 0.516], person: [0.444, 0.233, 0.487, 0.34], backpack: [0.384, 0.696, 0.485, 0.914], backpack: [0.755, 0.413, 0.846, 0.692], suitcase: [0.758, 0.413, 0.845, 0.69], suitcase: [0.1, 0.497, 0.173, 0.579], bicycle: [0.282, 0.363, 0.327, 0.442], car: [0.786, 0.25, 0.848, 0.322], car: [0.783, 0.27, 0.827, 0.335], car: [0.86, 0.254, 0.891, 0.3], car: [0.261, 0.101, 0.787, 0.626]

- ChatGPT와 GPT-4를 활용해서 데이터셋을 생성해서 데이터셋 부족 문제 해결
  - GPT-4는 이미지에 대한 설명과 위치 정보를 통해서 이미지를 인식

- GPT-4가 생성하는 3가지 유형의 텍스트

1. 대화: 이미지를 보고 답변하는 형식의 데이터 (약 58,000개)
2. 자세한 설명: 이미지 설명을 읽고 이미지에 대해 자세히 설명 (약 23,000개)
3. 복잡한 추론: 답변을 위해 단계별 추론이 필요한 어려운 질문을 생성하고 답변 (약 77,000개)

LLaVA 모델 구조

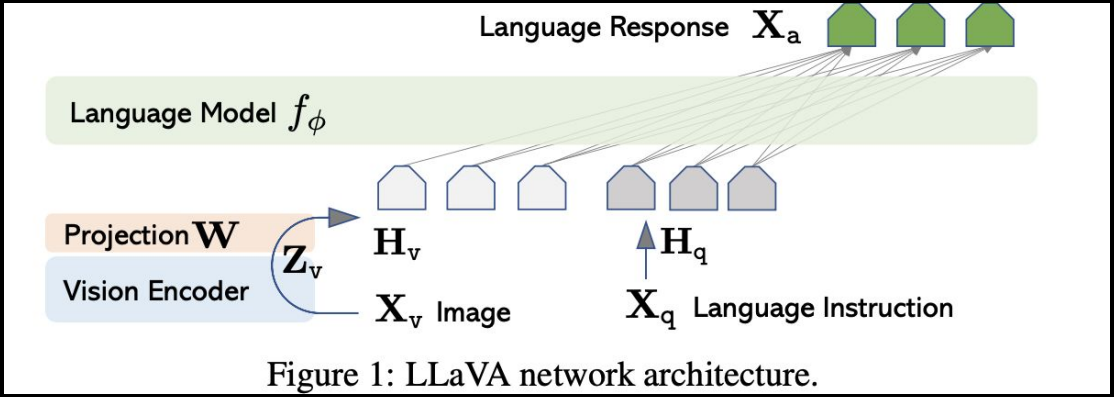
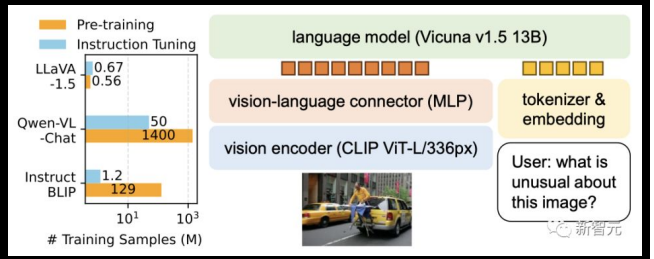
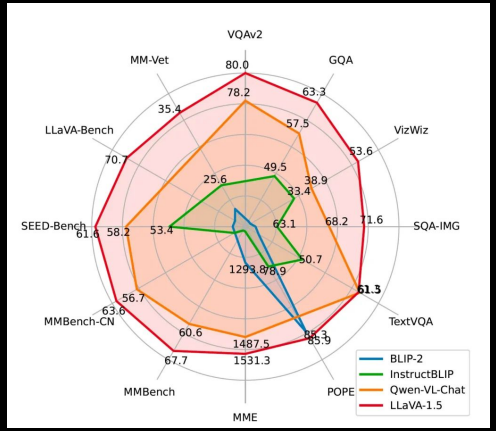


Figure 1: LLaVA network architecture.

1. 입력 이미지(Xv)를 CLIP의 이미지 인코더(Vision Encoder)를 통해 이미지 임베딩(Zv)으로 만들고
2. 간단한 선형 층(Projection W)을 통과해서 LLM에 입력할 임베딩 토큰(Hv)으로 만듭니다.
3. 텍스트 지시사항은 토큰 임베딩(Hq)으로 변환해서 함께 입력으로 넣고 결과 Xa를 생성합니다.

LLaVA를 발전시킬 수 있는 방법이 있을까?





Data (PT)	Data (IT)	Model	MMMU (val)	Math-Vista	MMB-ENG	SEED-IMG
N/A	N/A	GPT-4V	56.8	49.9	75.8	71.6
N/A	N/A	Gemini Ultra	59.4	53	-	-
N/A	N/A	Gemini Pro	47.9	45.2	73.6	70.7
1.4B	50M	Qwen-VL-Plus	45.2	43.3	-	65.7
1.5B	5.12M	CogVLM-30B	32.1	-	-	-
125M	~1M	Yi-VL-34B	45.9	-	-	-
558K	665K	LLaVA-1.5-13B	36.4	27.6	67.8	68.2
558K	760K	LLaVA-NeXT-34B	51.1	46.5	79.3	75.9

LLaVA-1.5

- 11개의 데이터셋에서 가장 뛰어난 성능
  - 다른 모델보다 훨씬 적은 데이터로 학습!

LLaVA-NeXT

- 입력 이미지의 해상도가 4배 높아짐
- 고품질의 지시 데이터셋을 구축해서 시각적 추론 능력과 OCR 성능이 개선
- 더 많은 시나리오에서 응답할 수 있어 다양한 애플리케이션에 활용
- SGLang 프레임워크를 사용해 추론 성능이 높아짐

감사합니다