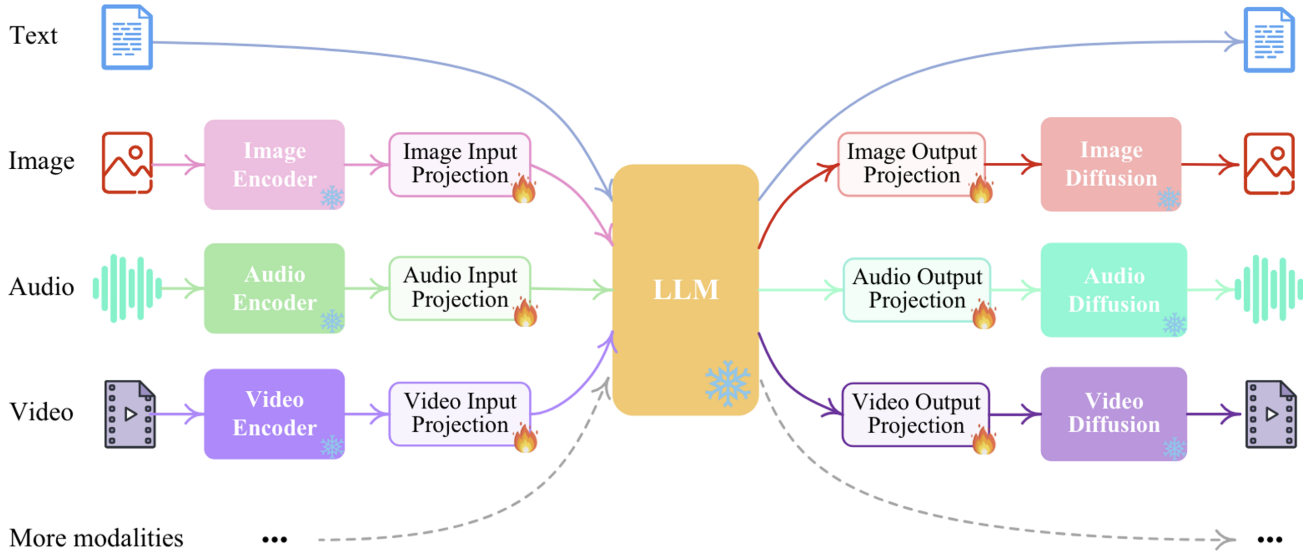
AI VIET NAM – AIO COURSE 2023

Exercise: Multimodal Large Language Models

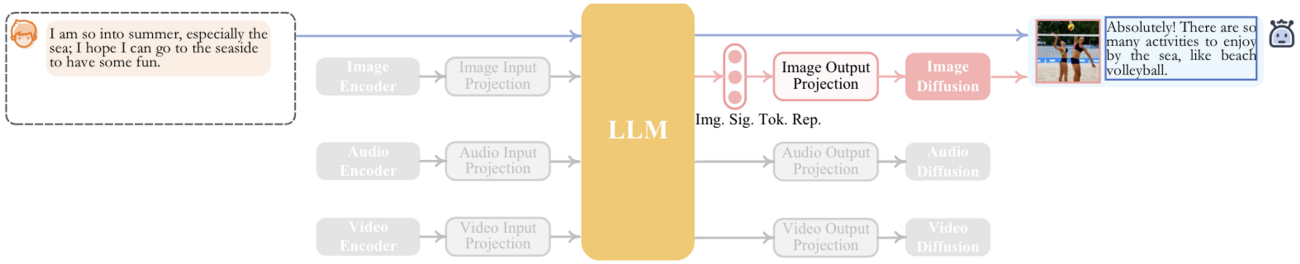
Quoc-Thai Nguyen và Duong-Thuan Nguyen

Ngày 23 tháng 4 năm 2024

Phần I. Giới thiệu



Hình 1: Giới thiệu các thành phần cơ bản trong mô hình MLLMs.



Hình 2: Ví dụ về sinh ảnh dựa vào văn bản sử dụng mô hình MLLMs.

MLLMs - Multimodal Large Language Models ngày càng được phát triển rộng rãi với mục tiêu xây dựng một mô hình ngôn ngữ lớn có thể xử lý cho các kiểu dữ liệu khác nhau như: văn bản, hình ảnh, âm thanh, video.

Với các mô hình MLLMs đầu vào có thể là văn bản, hình ảnh, video, âm thanh hoặc kết hợp của các kiểu dữ liệu trên. Đầu ra có thể là văn bản, hình ảnh, video, âm thanh. Ví dụ về sinh hình ảnh dựa vào văn bản được mô tả trong Hình 2.

Mô hình MLLMs bao gồm các thành phần chính sau:

- Modality Encoder: Bao gồm các module có thể từ các dữ liệu đầu vào như hình ảnh, âm thanh, video biểu diễn thành các đặc trưng. Ví dụ, với hình ảnh / video chúng ta có các mô hình như ViT, CLIP, Eva-CLIP ViT,... Với âm thanh chúng ta có các mô hình như: HuBERT, BEATs,... Hoặc là ImageBind - mô hình chung được sử dụng để encode các kiểu dữ liệu trên.
- Input Projector: phần này sẽ học cách kết nối các đặc trưng từ Encoder với văn bản đẩy vào LLMs. Các phương pháp bao gồm: Linear, MLP, Cross-Attention, Q-Former, P-former,...
- LLMs: các kiến trúc mô hình LLMs có thể được sử dụng như: GPT, LLaMA, Vicuna, Flan-T5,...
- Output Projector: bao gồm các phương pháp có thể ánh xạ từ đặc trưng của mô hình ngôn ngữ sang đặc trưng của kiểu dữ liệu như hình ảnh, video, âm thanh. Ví dụ: MLP, Tiny Transformer,...
- Modality Generator: bao gồm các module nhận đặc trưng của từng kiểu dữ liệu để sinh ra dữ liệu mong muốn. Ví dụ như Stable Diffusion cho ảnh, Zeroscope cho video, AudioLDM cho âm thanh,...

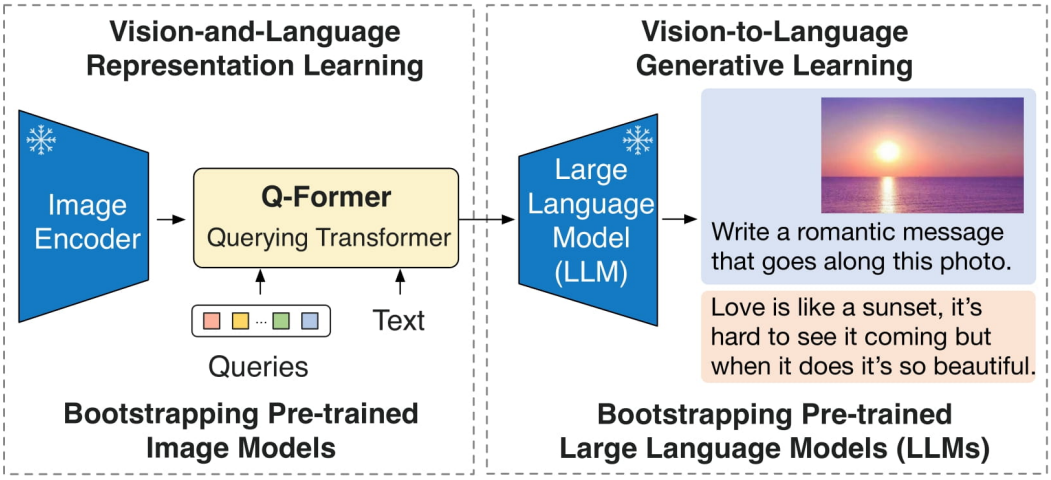
Dưới đây là một số mô hình MLLMs điển hình hiện nay. Trong đó I: Image (hình ảnh), T: Text (văn bản), V: Video, A: Audio (âm thanh).

Model	I/O	Modality Encoder	Input Projector	LLM	Output Projector	Modality Generator
BLIP-2	IT => T	CLIP ViT	Q-Former Linear	Flan-T5 OPT	-	-
LLaVA	IT => T	CLIP ViT	Linear	Vicuna	-	-
miniGPT-4	IT => T	Eva-CLIP ViT	Q-Former Linear	Vicuna	-	-
InstructBLIP	IVT => T	ViT	Q-Former Linear	Flan-T5 Vicuna	-	-
Next-GPT	IVAT => IVAT	ImageBlind	Linear	Vicuna	Tiny Transformer	Stable Diffusion Model
ModaVerse	IVAT => IVAT	ImageBlind	Linear	LLaMA2	MLP	Stable Diffusion Model

Hình 3: Một số mô hình MLLMs điển hình hiện nay.

Trong phần tiếp theo, chúng ta sẽ huấn luyện mô hình cơ bản BLIP-2 trên bài toán VQA - Visual Question Answering.

Phần II. Visual Question Answering using BLIP-2



Hình 4: Kiến trúc mô hình BLIP-2.

Kiến trúc mô hình BLIP-2 được mô tả trong Hình 4. Đầu vào mô hình là câu hỏi và hình ảnh. Đầu ra của mô hình là câu trả lời dựa vào câu hỏi và ngữ cảnh là hình ảnh. Mô hình BLIP-2 bao gồm các thành phần:

- Image Encoder: CLIP ViT.
- Input Projector: Q-Former.
- LLMs: Flan-T5 / OPT.

Trong phần tiếp theo, chúng ta sẽ huấn luyện mô hình BLIP-2 trên bộ dữ liệu VQA dựa vào thư viện transformers. Bộ dữ liệu VQA có 14,550 samples. Mỗi sample bao gồm đầu vào: Image + Question và đầu ra: Answer. Bộ dữ liệu VQA có thể được tải về: tại đây.

2.1. Build Dataset

```
# install libs
!pip install -q peft transformers bitsandbytes datasets

import os
import torch
from PIL import Image

class VQADataset(torch.utils.data.Dataset):
    """VQA (v2) dataset."""

    def __init__(self, dataset, processor, data_path):
        self.dataset = dataset
        self.processor = processor
        self.data_path = data_path

    def __len__(self):
        return len(self.dataset)
```

```
    def __getitem__(self, idx):
        # get image + text
        question = self.dataset[idx]['question']
        answer = self.dataset[idx]['answer']
        image_id = self.dataset[idx]['pid']
        image_path = os.path.join(self.data_path, f"train_fill_in_blank/train_fill_in_blank/{image_id}/image.png")
        image = Image.open(image_path).convert("RGB")
        text = question

        encoding = self.processor(image, text, padding="max_length", truncation=True, return_tensors="pt")
        labels = self.processor.tokenizer.encode(
            answer, max_length= 8, pad_to_max_length=True, return_tensors='pt'
        )
        encoding["labels"] = labels
        for k,v in encoding.items():  encoding[k] = v.squeeze()
        return encoding
```

2.2. Load Model

```
import torch
from peft import LoraConfig, get_peft_model
from transformers import BlipProcessor, BlipForQuestionAnswering

processor = BlipProcessor.from_pretrained("Salesforce/blip-vqa-base")
model = BlipForQuestionAnswering.from_pretrained("Salesforce/blip-vqa-base")

config = LoraConfig(
    r=16,
    lora_alpha=32,
    lora_dropout=0.05,
    bias="none",
    target_modules=["query", "key"]
)

model = get_peft_model(model, config)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
model.print_trainable_parameters()
```

2.3. Create Dataloader

```
from datasets import load_dataset
from torch.utils.data import DataLoader

data_path = './IconDomainVQAData'
ds = load_dataset("json", data_files=f"{data_path}/train.jsonl")

train_dataset = VQADataset(
    dataset=ds['test'],
    processor=processor,
    data_path=data_path
)
valid_dataset = VQADataset(
    dataset=ds['test'],
    processor=processor,
    data_path=data_path
)
```

```

batch_size = 8
train_dataloader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True, pin_memory=True)
valid_dataloader = DataLoader(valid_dataset, batch_size=batch_size, shuffle=False, pin_memory=True)
```

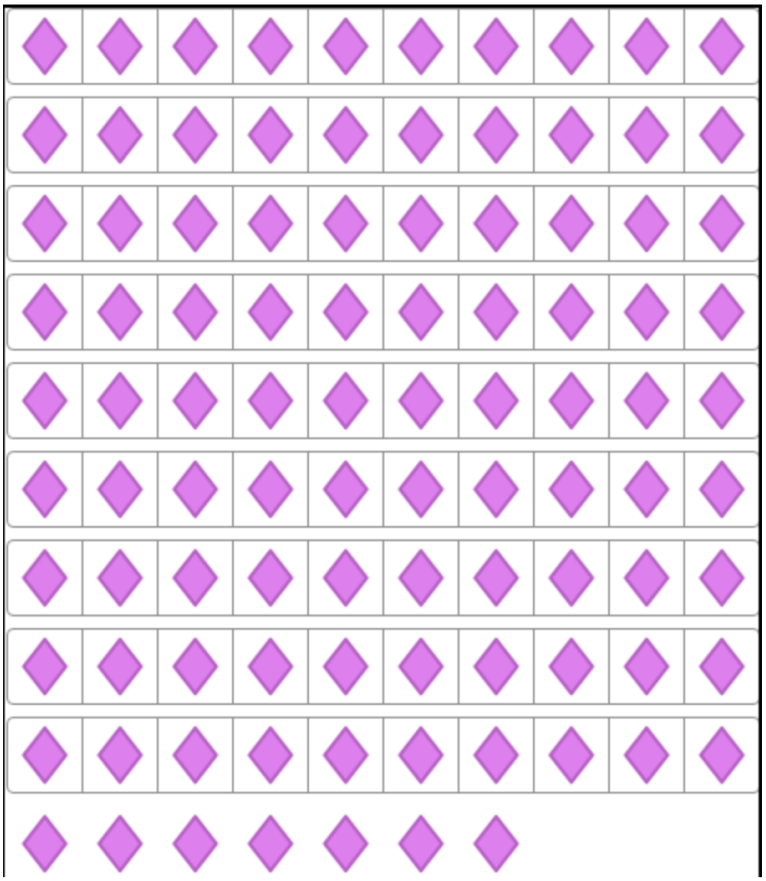
2.4. Train

```
from tqdm import tqdm

optimizer = torch.optim.AdamW(model.parameters(), lr=4e-5)
scheduler = torch.optim.lr_scheduler.ExponentialLR(optimizer, gamma=0.9, last_epoch=-1, verbose=False)

num_epochs = 1
min_eval_loss = float("inf")
scaler = torch.cuda.amp.GradScaler()

for epoch in range(num_epochs):
    epoch_loss = []
    model.train()
    for idx, batch in zip(tqdm(range(len(train_dataloader)), desc=f'Training batch: {epoch+1}'), train_dataloader):
        input_ids = batch.pop('input_ids').to(device)
        pixel_values = batch.pop('pixel_values').to(device)
        attention_masked = batch.pop('attention_mask').to(device)
        labels = batch.pop('labels').to(device)

        with torch.amp.autocast(device_type='cuda', dtype=torch.float16):
            outputs = model(
                input_ids=input_ids,
                pixel_values=pixel_values,
                labels=labels
            )

        loss = outputs.loss
        epoch_loss.append(loss.item())

        optimizer.zero_grad()
        scaler.scale(loss).backward()
        scaler.step(optimizer)
        scaler.update()

    model.eval()
    valid_loss = []
    for idx, batch in zip(tqdm(range(len(valid_dataloader)), desc=f'Validating batch: {epoch+1}'), valid_dataloader):
        input_ids = batch.pop('input_ids').to(device)
        pixel_values = batch.pop('pixel_values').to(device)
        attention_masked = batch.pop('attention_mask').to(device)
        labels = batch.pop('labels').to(device)

        with torch.amp.autocast(device_type='cuda', dtype=torch.float16):
            outputs = model(
                input_ids=input_ids,
                pixel_values=pixel_values,
                attention_mask=attention_masked,
                labels=labels
            )
```

```
        loss = outputs.loss
        valid_loss.append(loss.item())
    print("Epoch: {} - Training loss: {} - Eval Loss: {} - LR: {}".format(epoch+1, sum(epoch_loss)/len(epoch_loss), sum(valid_loss)/len(valid_loss), optimizer.param_groups[0]["lr"]))
    scheduler.step()
    avg_loss = sum(valid_loss)/len(valid_loss)
    if avg_loss < min_eval_loss:
        model.save_pretrained("./save_model", from_pt=True)
        print("Saved model to ./save_model")
        min_eval_loss = valid_loss
processor.save_pretrained("./save_model", from_pt=True)
```

2.5. Predict

Sau khi huấn luyện mô hình xong, trong phần này chúng ta sẽ load mô hình tốt nhất và dự đoán. Một ví dụ sử dụng đánh giá mô hình được mô tả như Hình 5.



Hình 5: Hình ảnh ví dụ với câu hỏi: "How many diamonds are there?".

```
import json
processor = BlipProcessor.from_pretrained("./save_model")
model = BlipForQuestionAnswering.from_pretrained("./save_model").to(device)
test_data_dir = f"{data_path}/test_data/test_data"
samples = os.listdir(test_data_dir)
sample_path = f"{data_path}/test_data/test_data/{samples[0]}"

# read question, image id
```

```
json_path = os.path.join(sample_path, "data.json")
with open(json_path, "r") as json_file:
    data = json.load(json_file)
    question = data["question"] # How many diamonds are there?
    image_id = data["id"]

# load image
image_path = os.path.join(test_data_dir, f"{image_id}", "image.png")
image = Image.open(image_path).convert("RGB")

encoding = processor(image, question, return_tensors="pt").to(device, torch.float16)

out = model.generate(**encoding)
generated_text = processor.decode(out[0], skip_special_tokens=True)

generated_text
```

Phần 4. Câu hỏi trắc nghiệm

Câu hỏi 1 Mô hình nào sau đây không phải là mô hình ngôn ngữ lớn (LLMs)?
a) Vicuna
b) LLaMA
c) PaLM
d) ResNet

Câu hỏi 2 Thành phần nào sau đây không có trong các thành phần chính của mô hình MLLMs?
a) Contrastive Search
b) Modality Encoder
c) LLMs
d) Input Projector

Câu hỏi 3 Mô hình được sử dụng để encode âm thanh là?
a) ViT
b) CLIP
c) BERT
d) BEATs

Câu hỏi 4 Phương pháp không được sử dụng trong bước Input Projector là?
a) Linear Projector
b) MLP
c) Q-Former
d) Cosine Similarity

Câu hỏi 5 Phương pháp nào được sử dụng làm khối Generator cho audio?
a) Zeroscope
b) AudioLDM
c) HuBERT
d) WavLM

Câu hỏi 6 Mô hình LLaVA sử dụng LLM nào?
a) Flan-T5
b) Vicuna
c) GPT3
d) PaLM

Câu hỏi 7 Mô hình BLIP-2 không xử lý được bài toán nào sau đây?
a) Image Captioning
b) Prompted Image Captioning

c) Visual Question Answering
d) Audio Generation

Câu hỏi 8 Mô hình nào sau đây có thể encode cho hình ảnh, văn bản, video?
a) CLIP
b) ViT
c) Eva-CLIP ViT
d) ImageBlind

Câu hỏi 9 Mô hình NExT-GPT sử dụng mô hình nào cho bước Output Generator?
a) MLP
b) Linear Projector
c) Tiny Transformer
d) Cross-Attention

Câu hỏi 10 Mô hình NExT-GPT sử dụng LLM nào?
a) Flan-T5
b) Vicuna
c) GPT3
d) PaLM

- Hết -