

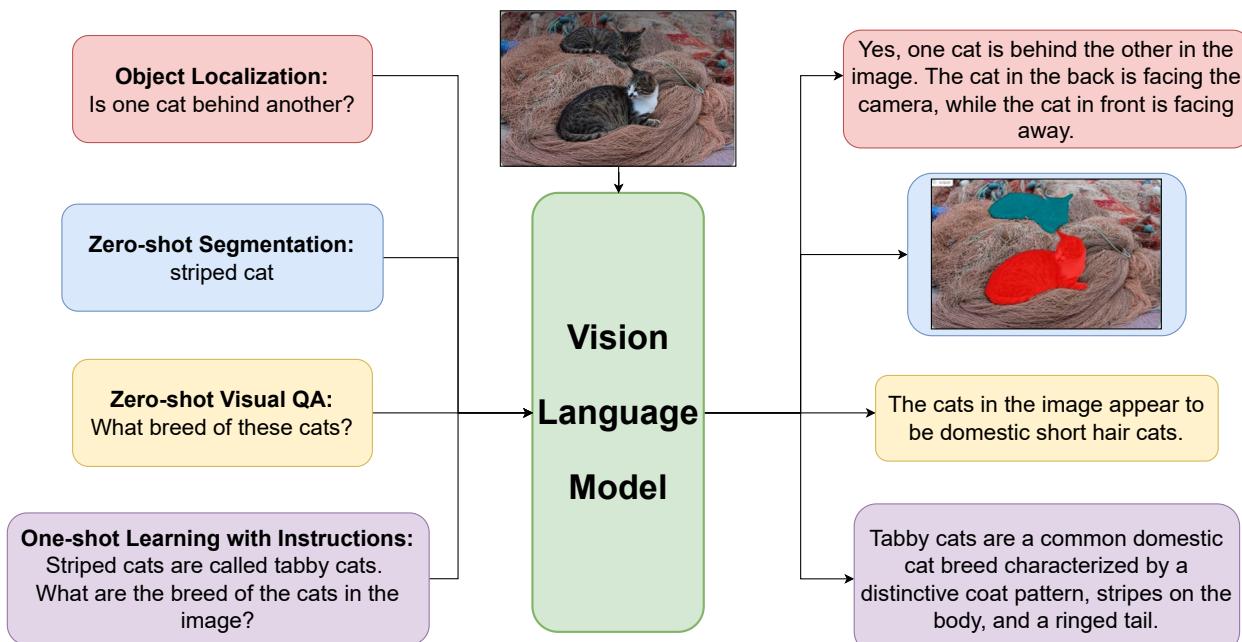
AI VIET NAM – AI COURSE 2024

Tutorial: Supervised Fine-tuning a Large Vision-Language Model

Dinh-Thang Duong, Nguyen-Thuan Duong, Phuc-Thinh Nguyen,
Quang-Vinh Dinh

I. Giới thiệu

Large Vision-Language Models (LVLMs) (Tạm dịch: Mô hình ngôn ngữ thị giác lớn) là một dạng mô hình Deep Learning với khả năng thực hiện hầu hết các tác vụ liên quan đến hình ảnh cũng như ngôn ngữ tự nhiên. Tận dụng sức mạnh của các mô hình ngôn ngữ lớn (LLMs), loại mô hình này được thiết kế để xử lý và hiểu cả hai dạng dữ liệu (thị giác và văn bản) trong một hệ thống duy nhất, mở ra những tiềm năng thiết kế ứng dụng về AI đa dạng hơn cũng như đặt nền móng cho sự phát triển về một hệ thống AGI (Artificial General Intelligence) có khả năng thực hiện bất cứ tác vụ nào mà con người làm được và hơn thế nữa.



Hình 1: Ảnh minh họa cho khả năng xử lý đa tác vụ của Large Vision-Language Models.

Về mặt lý thuyết, một mô hình LVLM có khả năng thực hiện được rất nhiều các tác vụ khác nhau liên quan đến thị giác máy tính cũng như xử lý ngôn ngữ tự nhiên nếu được huấn luyện trên dữ liệu tương ứng.

Song, khi sử dụng mô hình trên những tác vụ mà nội dung không thuộc trong bộ dữ liệu gốc mà mô hình được huấn luyện trước, LVLMs thường không đạt được độ chính xác mong muốn do thiếu kiến thức về ngữ cảnh và các khái niệm đặc thù trong dữ liệu mới. Vì vậy, trong bài viết này, chúng ta sẽ cùng tìm hiểu cách thức cải thiện độ chính xác của một mô hình LVLM thông qua việc huấn luyện tiếp tục có giám sát (supervised fine-tuning) trên bộ dữ liệu thuộc tác vụ mà chúng ta muốn LVLM đạt hiệu suất tốt hơn.

Trong bài viết này, hai bài toán gồm Object Detection và Visual Question Answering (VQA) sẽ được chọn làm các bài toán mà LVLMs cần giải. Theo đó, bài viết được bối cảnh như sau:

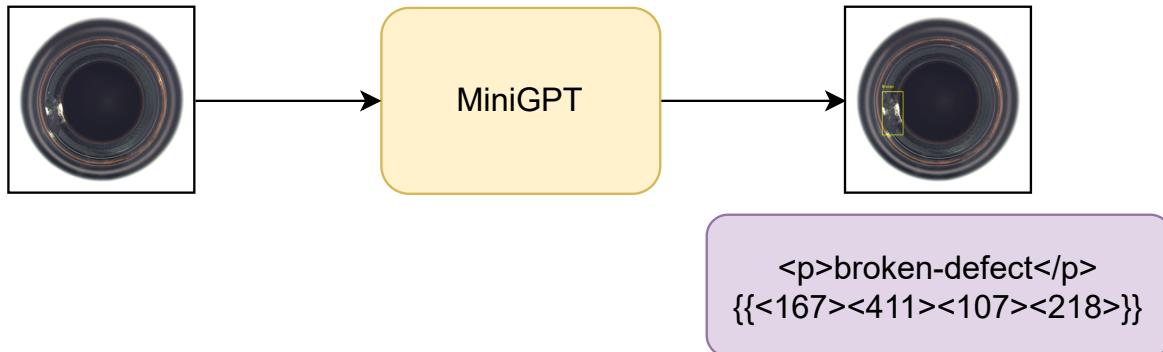
- **Phần I:** Giới thiệu về nội dung bài viết.
- **Phần II:** Tóm tắt về bài toán Object Detection và Visual Question Answering.
- **Phần III:** Tóm tắt về mô hình MiniGPT-4V.
- **Phần IV:** Hướng dẫn cài đặt, huấn luyện mô hình và đánh giá.
- **Phần V:** Tài liệu tham khảo.

II. Mô tả bài toán

1. Anomaly Detection: Phát hiện bất thường hay phát hiện ngoại lai (Outlier Detection) là quá trình xác định các điểm dữ liệu khác biệt so với phần còn lại, hoặc không tuân theo mẫu dự kiến trong một tập dữ liệu ảnh. Theo đó, đầu vào và đầu ra của mô hình như sau:

- **Input:** Hình ảnh của một vật thể.
- **Output:** Bounding box cho thành phần của vật thể trong ảnh bị lỗi, đồng thời loại lỗi của thành phần đó. Nếu như ảnh truyền vào không tồn tại lỗi, bounding box sẽ bao quanh toàn bộ vật thể.

Trong nội dung của bài viết này, ta sẽ xác định điểm bất thường với các vật dụng trong công nghiệp như: đinh ốc, sợi cáp, gạch lát nền, ... qua tập dữ liệu **MVTec AD** [1]. Với mỗi vật dụng, có thể có một hoặc nhiều lỗi khác nhau tồn tại như ví dụ ở hình sau.



Hình 2: Minh họa việc dụng mô hình MiniGPT trong việc phát hiện vật thể bất thường.

2. Vietnamese Visual Question Answering (ViVQA): Bài toán Trả lời Câu hỏi Hình ảnh (VQA) trong ngữ cảnh tiếng Việt. Mục tiêu là xây dựng hệ thống có khả năng nhận diện và phân tích nội dung hình ảnh, đồng thời hiểu và trả lời các câu hỏi được đặt ra bằng tiếng Việt. Điều này đòi hỏi mô hình kết hợp các phương pháp xử lý ngôn ngữ tự nhiên (NLP) và thị giác máy tính (CV) trong bối cảnh ngôn ngữ tiếng Việt, cụ thể như sau:

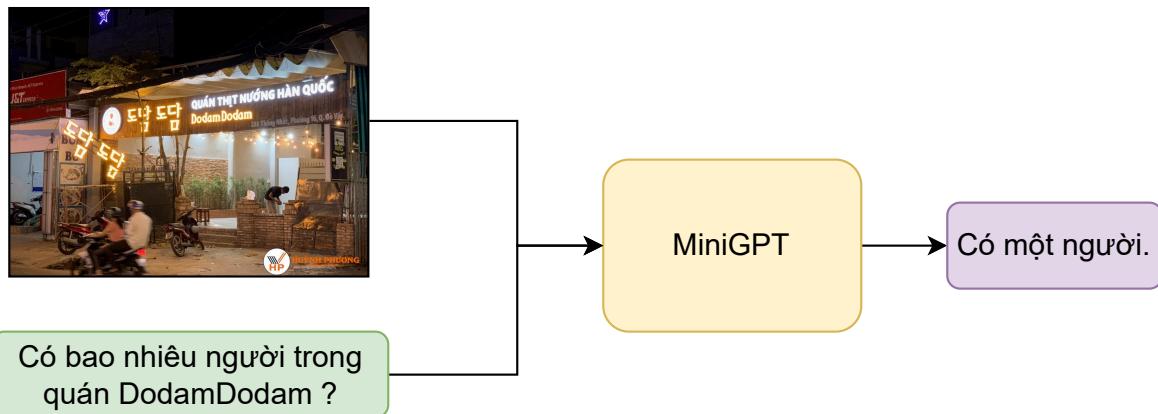
- **Input:** Một cặp dữ liệu gồm:
 - **Hình ảnh:** Chứa các thông tin thị giác (cảnh vật, đối tượng, văn bản, v.v.).
 - **Câu hỏi tiếng Việt:** Được đặt ra xoay quanh nội dung của hình ảnh.
- **Output:** Câu trả lời bằng tiếng Việt, dựa trên việc phân tích nội dung hình ảnh, nhận diện đối tượng, văn bản (nếu có), và hiểu ngữ cảnh câu hỏi.

Đối với đặc thù của bài toán này, tập dữ liệu **ViVQA** [2] sẽ được sử dụng. Đây là một bộ dữ liệu VQA tập trung vào ngôn ngữ tiếng Việt, yêu cầu mô hình:

- (a) Phân tích ngữ nghĩa câu hỏi bằng tiếng Việt.

- (b) Nhận diện đối tượng, văn bản (nếu cần) và ngữ cảnh trong hình ảnh.
- (c) Tổng hợp thông tin để đưa ra câu trả lời chính xác, phù hợp ngôn ngữ và văn phong tiếng Việt.

Thông qua ViVQA, ta hướng đến việc cải thiện khả năng đọc hiểu và suy luận trên hình ảnh trong môi trường ngôn ngữ tiếng Việt, cũng như tối ưu hiệu suất của mô hình trên các tác vụ VQA đa ngôn ngữ.



Hình 3: Minh họa về cách mô hình có thể trả lời câu hỏi bằng tiếng Việt dựa trên nội dung ảnh.

III. Mô hình MiniGPT-4V

III.1. Giới thiệu về MiniGPT-4V

MiniGPT-4V [3] là một trong những mô hình LVLM với những bước tiến mới trong việc kết hợp khả năng hiểu ngôn ngữ tự nhiên của các mô hình LLM với khả năng nhận diện và xử lý hình ảnh. Mục tiêu của mô hình là tạo ra một giao diện thống nhất (unified interface) cho cả các nhiệm vụ thị giác và ngôn ngữ, giúp xử lý các tác vụ như trả lời câu hỏi dựa trên hình ảnh, tạo chú thích ảnh và các ứng dụng đa nhiệm khác trở nên tốt hơn.

III.2. Kiến trúc của MiniGPT-4V

Mô hình MiniGPT-4 gồm ba thành phần chính:

1. Image Encoding Module (Module encoding hình ảnh).
2. Projection Module.
3. LLM Module.

Với mỗi thành phần trên sẽ có các mô tả chi tiết cụ thể như sau:

III.2.1. Module encoding hình ảnh

Tương tự như các LVLM khác, một Image Encoder sẽ được sử dụng nhằm giúp trích xuất những đặc trưng trực quan từ ảnh. Các kiến trúc như Vision Transformer (ViT) [4] hay ResNet [5] được sử dụng để chuyển đổi hình ảnh thành các vector đặc trưng. Kết quả thu được là một vector biểu diễn các thông tin quan trọng từ hình ảnh. Module này sẽ được freeze lại trong lúc huấn luyện mô hình.

III.2.2. Module projection

Sau khi thu được vector đặc trưng từ hình ảnh, module projection sẽ chuyển đổi vector đó sang một không gian biểu diễn tương thích với không gian embedding có sẵn của mô hình ngôn ngữ. Quá trình này giúp đưa các thông tin thị giác vào cùng với chuỗi embedding mà mô hình LLM có thể hiểu và xử lý đồng thời. Nếu gọi vector đặc trưng của ảnh là \mathbf{v} , thì sau khi chiếu (project) ta nhận được:

$$\mathbf{z} = f_{\text{proj}}(\mathbf{v}), \text{with } z \in R^{d_e}$$

với R^{d_e} là kích thước của các token embedding trong mô hình ngôn ngữ. Ta sẽ xem các thành phần từ ảnh cũng như là các token trong một câu.

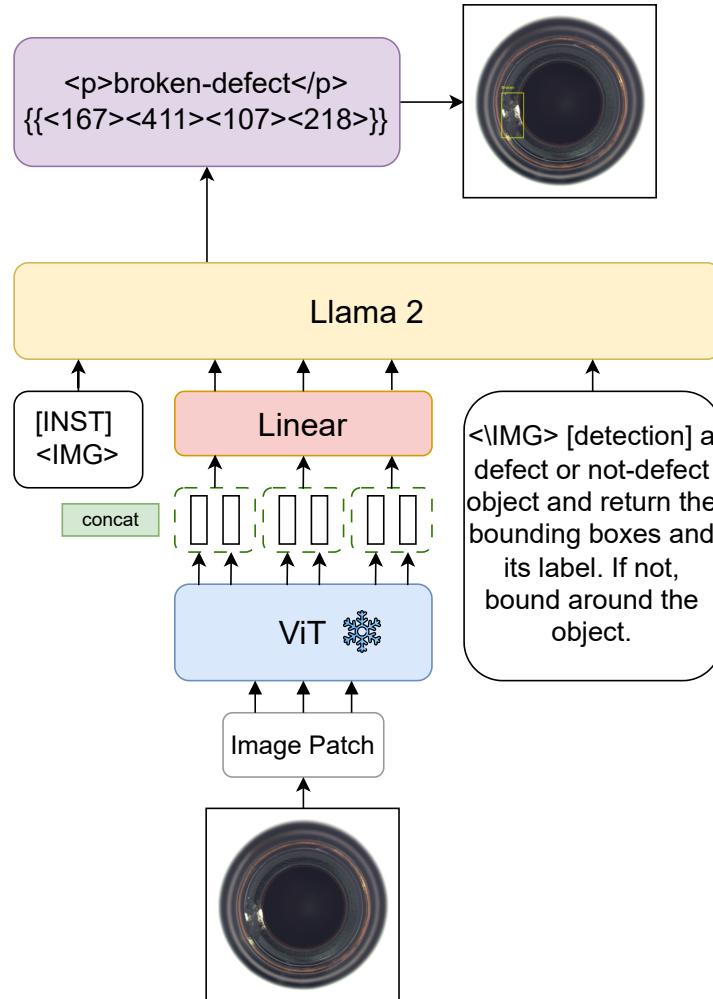
Tuy nhiên, đối với những hình ảnh có độ phân giải cao (ví dụ: 448x448), mô hình ViT có xu hướng tạo ra tới 1024 token, dẫn đến việc tăng đáng kể thời gian và tài nguyên huấn luyện. Để khắc phục vấn đề này, nhóm tác giả đã đề xuất ghép mỗi 4 token lân cận lại với nhau thành một token duy nhất trong vector đặc trưng thị giác trước khi đưa qua lớp projection. Giải pháp

này giúp giảm số lượng token cần xử lý, từ đó tiết kiệm tài nguyên mà vẫn đảm bảo giữ lại được những thông tin thị giác quan trọng.

$$\mathbf{v}'_i = \text{concat}(\mathbf{v}_{4i-3}, \mathbf{v}_{4i-2}, \mathbf{v}_{4i-1}, \mathbf{v}_{4i}), \text{ with } i = 1, 2, \dots, n,$$

III.2.3. Mô hình ngôn ngữ (LLM)

Dây là thành phần cốt lõi chịu trách nhiệm xử lý embedding và kết hợp thông tin từ ảnh. Đầu vào của LLM không chỉ bao gồm embedding các token văn bản mà còn chứa embedding của vector đặc trưng từ module projection. Nhờ đó, mô hình có thể “hiểu” ngữ cảnh của hình ảnh và đưa ra các câu trả lời hay chú thích một cách tự nhiên và linh hoạt. Mô hình LLM được lựa chọn ở đây không nhất thiết phải là mô hình Vicuna [6] mà có thể là các mô hình khác như LLaMA [7], RoBERTa [8], ...



Hình 4: Kiến trúc của mô hình MiniGPT-V.

Để tránh việc mô hình hiểu lầm về các tác vụ cần phải làm, ta cần giới thiệu các token đặc trưng cho từng tác vụ đa nhiệm dùng để huấn luyện. Các token này có chức năng như mô tả dưới đây:

Task	Identifier
VQA (Visual Question Answering)	[vqa]
Caption (Image Captioning)	[caption]
Grounded Image Captioning	[grounding]
REC (Referring Expression Comprehension)	[refer]
REG (Referring Expression Generation)	[identify]
Object Parsing and Grounding	[detection]

Bảng 1: Token nhận dạng tác vụ (task identifier token) cho 6 tác vụ khác nhau, bao gồm: VQA, tạo chú thích ảnh, tạo chú thích ảnh có liên kết đối tượng, nhận diện đối tượng theo mô tả (REC), sinh mô tả đối tượng (REG), và nhận diện vật thể (mô hình sẽ trích xuất các đối tượng từ đầu vào văn bản và xác định vị trí bounding box). Đối với các chỉ dẫn không liên quan đến thị giác, mô hình không sử dụng bất kỳ token nhận dạng tác vụ nào.

Cuối cùng, đầu vào cho mô hình (prompt) được biểu diễn như sau:

"[INST] <ImageFeature> [Task Identifier] Instruction [/INST]"

Trong prompt này, [INST] được xem là vai trò người dùng, còn [/INST] được xem là vai trò trợ lý. Ba thành phần còn lại được giải thích như sau:

- Đặc trưng hình ảnh:** Là các embedding của ảnh, được đặt trong một block để mô hình hiểu được thông tin thị giác.
- Token nhận dạng tác vụ:** Chỉ định tác vụ mà mô hình cần phải làm, với câu hỏi liền sau để làm rõ hơn.
- Hướng dẫn đầu vào:** Hướng dẫn hoặc câu hỏi.

III.3. Quá trình huấn luyện

Toàn bộ quá trình huấn luyện của MiniGPT-v2 được chia thành ba giai đoạn chính như sau:

III.3.1. Stage 1: Tiền huấn luyện (Pre-training)

Để mô hình có được nền tảng kiến thức rộng về nhiệm vụ thị giác-ngôn ngữ, ta huấn luyện nó trên tập hợp các bộ dữ liệu có **nhãn yếu** (weakly-labeled) và các bộ dữ liệu **chi tiết** (fine-grained). Trong giai đoạn này, việc huấn luyện ưu tiên tỉ lệ lấy mẫu (sampling ratio) cao cho dữ liệu có nhãn yếu nhằm học các kiến thức đa dạng.

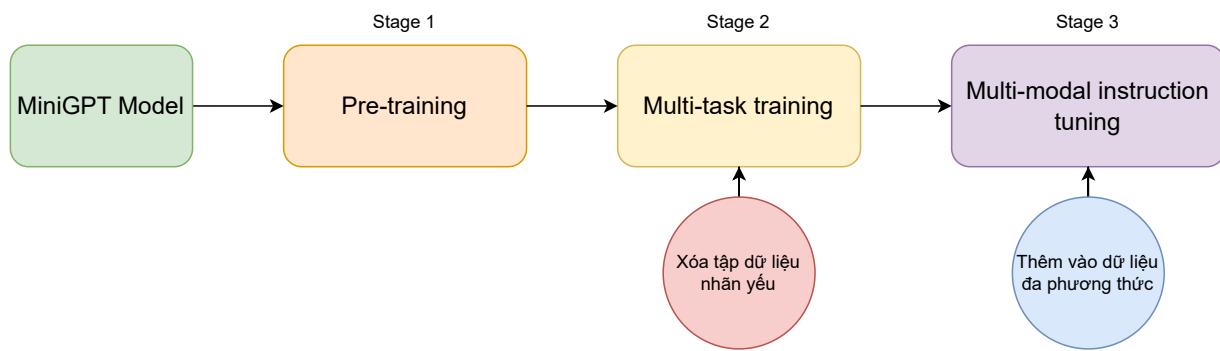
III.3.2. Stage 2: Huấn luyện đa nhiệm (Multi-task training)

Để cải thiện hiệu suất của MiniGPT-v2 trên từng tác vụ được liệt kê ở bảng 1, ở giai đoạn này ta chỉ sử dụng các bộ dữ liệu chi tiết. Đồng thời, **loại bỏ những bộ dữ liệu gán nhãn yếu**

(chẳng hạn như GRIT-20M[9] và LAION[10]) từ giai đoạn 1, cũng như điều chỉnh lại tỉ lệ lấy mẫu dựa trên tần suất của mỗi tác vụ. Nhờ vậy, mô hình có thể ưu tiên các dữ liệu ảnh-văn bản được liên kết chất lượng cao, nâng cao hiệu suất trên nhiều tác vụ khác nhau.

III.3.3. Stage 3: Tinh chỉnh mô hình với hướng dẫn đa phương thức (Multi-modal instruction tuning)

Cuối cùng, ta tiếp tục tinh chỉnh mô hình bằng cách **thêm các bộ dữ liệu hướng dẫn đa phương thức**, đồng thời cải thiện khả năng hội thoại như một chatbot. Ở giai đoạn này, các bộ dữ liệu từ giai đoạn 2 vẫn được giữ lại, đồng thời bổ sung thêm các bộ dữ liệu hướng dẫn (instruction data) khác, bao gồm LLaVA [11], Flickr30k [12],... Tương tự như giai đoạn 2, tỉ lệ lấy mẫu ở đây thấp hơn cho các bộ dữ liệu chi tiết từ giai đoạn 2 và tỉ lệ lấy mẫu cao hơn cho các bộ dữ liệu hướng dẫn mới.



Hình 5: Quá trình huấn luyện của mô hình MiniGPT-4V.

IV. Cài đặt chương trình

IV.1. Tổng quan về MiniGPT-4 Repository

Thông thường, các repository trong Deep Learning sẽ bao gồm các thành phần cơ bản sau: dataset, model, trainer và các file cấu hình cho việc huấn luyện, đánh giá. Trong phạm vi của bài viết này, chúng ta sẽ tập trung vào việc **supervised fine-tuning** model `miniGPT4_v2` xuống các nhiệm vụ cụ thể (downstream tasks) như Object Detection, Visual Question Answering - VQA. Trong đó, chúng ta sẽ sử dụng MiniGPT-4 Repository như là **codebase** kèm theo một vài sự thay đổi bổ sung ở phần dataset cũng như các file cấu hình, các phần còn lại chúng ta sẽ tận dụng và giữ nguyên.

IV.1.1. VQA Dataset

Trong MiniGPT-4, các dataset được định nghĩa trong thư mục "`minigpt4/datasets/datasets-`". Mỗi tập dữ liệu sẽ được khai báo dưới dạng một class kế thừa từ `torch.utils.data.Dataset` của PyTorch, giúp dễ dàng quản lý và truy xuất dữ liệu trong quá trình huấn luyện hoặc suy luận. Ví dụ, để thêm một dataset mới mang tên **ViVQA**, chúng ta có thể định nghĩa trong file "`vqa_dataset.py`", nằm trong thư mục "`datasets`". Cụ thể, đoạn mã dưới đây minh họa cách thiết lập một dataset chuyên dùng cho bài toán VQA, trong đó mỗi mẫu dữ liệu bao gồm hình ảnh, câu hỏi liên quan đến hình ảnh đó và câu trả lời tương ứng.

```

1 class ViVQADataset(torch.utils.data.Dataset):
2     def __init__(self, data):
3         self.data = data
4
5     def __len__(self):
6         return len(self.data)
7
8     def __getitem__(self, index):
9         image = self.data[index]["image"]
10        question = self.data[index]["question"]
11        answer = self.data[index]["answer"]
12        return image, question, answer

```

Tuy nhiên, chúng ta đang kế thừa codebase từ MiniGPT-4. Vì vậy, cần phải định nghĩa **class-ViVQADataset** kèm theo các **tham số (parameters)** giống như MiniGPT-4 quy định sau đây:

```

1 # minigpt4/datasets/datasets/vqa_dataset.py
2
3 class VQADataset(BaseDataset):
4     def __init__(self, vis_processor, text_processor, vis_root, ann_paths):
5         super().__init__(vis_processor, text_processor, vis_root, ann_paths)

```

Trong đó, `vis_processor` và `text_processor` là các bộ tiền xử lý cho ảnh và văn bản sẽ truyền vào trong lúc **build** dataset (sẽ được trình bày trong phần tiếp theo). Và hai tham số còn lại đó là `vis_root` và `ann_paths` tương ứng là thư mục chứa ảnh và file dữ liệu (cụ thể trong bộ dữ liệu ViVQA thì file này là một file csv có 4 cột: "`question`", "`answer`", "`img_id`" và "`type`"). Khi đó `class ViVQADataset` của chúng ta sẽ được viết thành:

```

1 class ViVQADataset(torch.utils.data.Dataset):
2     def __init__(self, vis_processor, text_processor, vis_root, ann_path):
3         self.vis_root = vis_root
4         self.vis_processor = vis_processor
5         self.text_processor = text_processor
6
7         self.instruction_pool = [
8             "[vqa] {}",
9             "[vqa] Based on the image, respond to this question with a short
10            answer: {}"
11         ]
12         self.data = pd.read_csv(ann_path)
13         self.img_name_prefix = "COCO_train2014_"
14
15     def __len__(self):
16         return len(self.data)
17
18     def __getitem__(self, index):
19         sample = self.data.iloc[index]
20         image_id = str(sample["img_id"]).zfill(12)
21         image_path = os.path.join(self.vis_root, f"{image_id}.jpg")
22         image = Image.open(image_path).convert("RGB")
23         image = self.vis_processor(image)
24         question = self.text_processor(sample["question"])
25         answer = self.text_processor(sample["answer"])
26         instruction = random.choice(self.instruction_pool).format(question)
27         instruction = "<Img><ImageHere></Img> {}".format(instruction)
28         return {
29             "image": image,
30             "instruction_input": instruction,
31             "answer": answer,
32             "image_id": image_id
33         }

```

Như đã đề cập ở trên, dataset trong MiniGPT4 cần phải được build, chính vì vậy tiếp theo chúng ta cần khai báo các cấu hình cần thiết để xây dựng dataset. Chúng ta sẽ khai báo `class ViVQABuilder` tại file theo đường dẫn sau:

"`minigpt4/datasets/builders/image_text_pair_builder.py`" và thêm đoạn code dưới đây:

```

1 @registry.register_builder("vivqa")
2 class ViVQABuilder(BaseDatasetBuilder):
3     train_dataset_cls = ViVQADataset
4     DATASET_CONFIG_DICT = {
5         "default": "configs/datasets/vivqa/default.yaml",
6     }
7     def build_datasets(self):
8         logging.info("Building datasets...")
9         self.build_processors()
10        build_info = self.config.build_info
11        datasets = dict()
12
13        # create datasets
14        dataset_cls = self.train_dataset_cls
15        datasets["train"] = dataset_cls(
16            vis_processor=self.vis_processors["train"],

```

```

17         text_processor=self.text_processors["train"] ,
18         vis_root=build_info.image_path ,
19         ann_path=build_info.ann_path ,
20     )
21     return datasets

```

Trong khai báo `class builder` ở trên, chúng ta đã khai báo một file cấu hình cho ViVQA dataset tại đường dẫn "`configs/datasets/vivqa/default.yaml`". File này đơn giản là khai báo các đường dẫn đến các thư mục chứa ảnh và file csv, các bạn tạo file với đường dẫn tương ứng ở trên và chén nội dung như sau:

```

1 datasets:
2   vivqa:
3     data_type: images
4     build_info:
5       image_path: ./data/ViVQA/images
6       ann_path: ./data/ViVQA/train.csv

```

Như vậy, chúng ta đã cấu hình xong phần dataset. Tiếp đến, cấu hình cho việc huấn luyện thông qua một file chứa các tham số huấn luyện cũng như các checkpoint cần thiết cần được điều chỉnh. Theo đó, cần tạo một file cấu hình tại đường dẫn sau:

"`train_configs/minigptv2_finetune_vivqa.yaml`" và điền nội dung sau đây:

```

1 model:
2   arch: minigpt_v2
3   model_type: pretrain
4   max_txt_len: 1024
5   image_size: 448
6   end_sym: "</s>"
7   llama_model: "meta-llama/Llama-2-7b-chat-hf"
8   ckpt: "./ckpt/checkpoint_stage3.pth"
9   use_grad_checkpoint: True
10  low_resource: True
11  chat_template: True
12  lora_r: 64
13  lora_alpha: 16
14
15 datasets:
16   vivqa:
17     batch_size: 2
18     vis_processor:
19       train:
20         name: "blip2_image_train"
21         image_size: 448
22     text_processor:
23       train:
24         name: "blip_caption"
25     sample_ratio: 100
26
27 run:
28   task: image_text_pretrain
29   # optimizer
30   lr_sched: "linear_warmup_cosine_lr"
31   init_lr: 1e-5
32   min_lr: 1e-6

```

```

33    warmup_lr: 1e-6
34
35    weight_decay: 0.05
36    max_epoch: 5
37    num_workers: 6
38    warmup_steps: 500
39    iters_per_epoch: 2000
40
41    seed: 42
42    output_dir: "vivqa_outputs"
43
44    amp: True
45    resume_ckpt_path: null
46
47    evaluate: False
48    train_splits: ["train"]
49
50    device: "cuda"
51    world_size: 1
52    dist_url: "env://"
53    distributed: True
54
55    wandb_log: False
56    job_name: minigptv2_finetune

```

Trong cấu hình huấn luyện trên, các bạn cần lưu ý 3 phần sau đây:

- **Các checkpoints:** Để có được checkpoint tại dòng 7, các bạn cần đăng nhập bằng **access token** của tài khoản HuggingFace. Còn checkpoint tại dòng 8 các bạn có thể tải bằng Google Drive với link được cung cấp bởi tác giả.
- **Khai báo dataset training:** Các bạn khai báo dataset mà chúng ta đã định nghĩa ở trên với các giá trị mặc định và không chỉnh sửa gì thêm.
- **Hyper-parameters:** Đây là các siêu tham số cho quá trình huấn luyện, các bạn có thể thay đổi tùy thuộc vào tài nguyên huấn luyện mà các bạn có.

Tới đây, chúng ta đã hoàn toàn có thể tiến hành huấn luyện mô hình, một thay đổi cuối cùng đó chính là việc đánh giá mô hình sau huấn luyện. Các bạn chỉ cần thay đổi nội dung trong file cấu hình đánh giá tại "["eval_configs/minigptv2_benchmark_evaluation.yaml"](#)" thành như sau:

```

1 model:
2   arch: minigpt_v2
3   model_type: pretrain
4   max_txt_len: 500
5   end_sym: "</s>"
6   low_resource: True
7   prompt_template: "[INST] {} [/INST]"
8   llama_model: "meta-llama/Llama-2-7b-chat-hf"
9   ckpt: "./minigpt4/vivqa_outputs/20250314075/checkpoint_1.pth"
10  lora_r: 64
11  lora_alpha: 16
12
13
14 datasets:

```

```

15 cc_sbu_align:
16   vis_processor:
17     train:
18       name: "blip2_image_eval"
19       image_size: 448
20   text_processor:
21     train:
22       name: "blip_caption"
23
24
25 evaluation_datasets:
26   vivqa:
27     batch_size: 4
28     eval_file_path: ./data/ViVQA/test.csv
29     img_path: ./data/ViVQA/images
30     max_new_tokens: 20
31
32 run:
33   task: image_text_pretrain
34   name: minigptv2_evaluation
35   save_path: eval_outputs

```

Trong file này, các phần có thể được tuỳ chỉnh như: tại dòng 9 có thể truyền đường dẫn checkpoint của mô hình đã được huấn luyện hoặc checkpoint của tác giả cung cấp tuỳ vào mục đích đánh giá mô hình. Tại các dòng từ 27 tới 30, các bạn thay đổi cho phù hợp với thư mục lưu "**val/test data**" cũng như **batch_size** phù hợp với tài nguyên sẵn có.

Như vậy, chúng ta đã hoàn thành việc cấu hình và các file cần thiết để huấn luyện là đánh giá dataset ViVQA. Trong phần tiếp theo dưới đây, chúng ta sẽ cấu hình cho Object Detection dataset. Việc này cũng tương tự như việc cài đặt cho VQA dataset.

IV.1.2. Object Detection Dataset

Dầu tiên là **class MVTecDataset** tại "[minigpt4/datasets/datasets/mvtec_dataset.py](#)":

```

1 class MVTecDataset(Dataset):
2     def __init__(self, vis_processor, text_processor, vis_root, ann_path):
3         self.vis_root = vis_root
4         self.vis_processor = vis_processor
5         self.text_processor = text_processor
6
7         self.instruction_pool = [
8             "[detection] {}",
9         ]
10
11     with open(ann_path, "r") as f:
12         self.ann = json.load(f)
13
14     def __len__(self):
15         return len(self.ann)
16
17     def __getitem__(self, index):
18         info = self.ann[index]
19         gt_bbox = info["bbox"]

```

```

20     ans_cls = info["class"]
21
22     image_path = os.path.join(self.vis_root, info["image_path"])
23     image = Image.open(image_path).convert("RGB")
24     image = self.vis_processor(image)
25
26     input = "a defect or not-defect object and return the bounding boxes
27             and its label. If not, bound around the
28             object."
29
30     ans_defect = "defect" if info["is_broken"] == True else "not-defect"
31     ans_para = f"<p>{ans_cls}-{ans_defect}</p>"
32     answer = f"{ans_para}{{<{gt_bbox[0]}><{gt_bbox[1]}><{gt_bbox[2]}><{gt_bbox[3]}>}}"
33
34     instruction = random.choice(self.instruction_pool).format(input)
35     instruction = "<Img><ImageHere></Img> {}".format(instruction)
36
37     return {
38         "image": image,
39         "instruction_input": instruction,
40         "answer": answer,
41         "image_id": info["image_path"],
42     }

```

Tiếp theo là builder tại file "minigpt4/datasets/image_text_pair_builder.py":

```

1 @registry.register_builder("mvtec_ad")
2 class MVTECADBuilder(BaseDatasetBuilder):
3     train_dataset_cls = MVTecDataset
4     DATASET_CONFIG_DICT = {
5         "default": "configs/datasets/mvtec/default.yaml",
6     }
7     def build_datasets(self):
8         logging.info("Building datasets...")
9         self.build_processors()
10        build_info = self.config.build_info
11        datasets = dict()
12
13        # create datasets
14        dataset_cls = self.train_dataset_cls
15        datasets["train"] = dataset_cls(
16            vis_processor=self.vis_processors["train"],
17            text_processor=self.text_processors["train"],
18            ann_path=build_info.ann_path,
19            vis_root=build_info.image_path,
20        )
21        return datasets

```

Cấu hình đường dẫn cho dataset tại "minigpt4/configs/datasets/mvtec/default.yaml":

```

1 datasets:
2     mvtec_ad:
3         data_type: images
4         build_info:
5             image_path: ./data/MVTEC_det/images
6             ann_path: ./data/MVTEC_det/train_data.json

```

Cuối cùng là file cấu hình huấn luyện, các bạn copy lại nội dung của file "["train_configs--minigptv2_finetune_vivqa.yaml"](#)" và tạo file với tên mới là "["minigptv2_finetune_mvtec.yaml"](#)". Phần dataset các bạn thay đổi thành như sau:

```

1 datasets:
2   mvtec_ad:
3     batch_size: 2
4     vis_processor:
5       train:
6         name: "blip2_image_train"
7         image_size: 448
8     text_processor:
9       train:
10      name: "blip_caption"
11      sample_ratio: 100

```

Các phần còn lại, các bạn tự chỉnh theo tài nguyên máy tính thích hợp.

IV.2. Supervised fine-tuning MiniGPT-4 v2

IV.2.1. Cài đặt mã nguồn

Dầu tiên, các bạn tải source code thông qua lệnh `git clone` và di chuyển vào bên trong repository như sau:

```

1 git clone https://github.com/ThuanNaN/MiniGPT-4.git
2 cd MiniGPT-4

```

Sau đó, các bạn tiến hành cài đặt các thư viện cần thiết được định nghĩa trong file `requirements.txt` như sau:

```

1 pip3 install -r requirements.txt

```

Tiếp đến, các bạn hãy đăng nhập vào trang [huggingface.co](#), truy cập vào phần tài khoản để tạo **Access token**, sau đó dán vào sau khi chạy đoạn mã dưới đây:

```

1 from huggingface_hub import notebook_login
2 notebook_login()

```

Màn hình in ra có dòng chữ **Login successful** báo hiệu đã đăng nhập thành công.

IV.2.2. Tải bộ dữ liệu

Để tổ chức các thư mục dữ liệu phù hợp với các cấu hình ở trên, ta di chuyển vào thư mục **data** bằng lệnh `cd`:

```

1 cd ./data

```

Để tải bộ dữ liệu ViVQA, các bạn chạy lần lượt các lệnh dưới đây:

```

1 git clone https://github.com/kh4nh12/ViVQA
2 cd ViVQA
3 gdown 1jvFik5rNEk--in1Ismk-aiA6NqDfJyXI

```

```

4 unzip -q ./vivqa_images.zip
5 cd ..

```

Tiếp theo, tải bộ dữ liệu MVTEC, các bạn cũng chạy lần lượt các câu lệnh sau đây:

```

1 gdown 1e01JVGKLIyIPoGA5kQLS3LJtWmC8obK8
2 unzip -q ./MVTEC_det.zip
3 cd ..

```

IV.2.3. Tải checkpoint

Trước hết, các bạn tạo một thư mục để lưu checkpoint và di chuyển vào thư mục này:

```

1 mkdir ./ckpt
2 cd ckpt

```

Sau đó, tải file checkpoint thông qua gdown và id được cung cấp từ tác giả:

```

1 gdown 1HkoUUrjzFGn33cSiUkI-KcT-zysCynAz
2 cd ..

```

IV.2.4. Huấn luyện

Để huấn luyện mô hình, các bạn chạy lệnh `python train.py` kèm theo file cấu hình tương ứng. Ví dụ, để huấn luyện nhiệm vụ VQA các bạn chạy lệnh sau:

```
1 python train.py --cfg-path train_configs/minigptv2_finetune_vivqa.yaml
```

Tương tự, để huấn luyện nhiệm vụ phát hiện vùng hư hỏng của tập dữ liệu MVTEC, các bạn chạy lệnh sau:

```
1 python train.py --cfg-path train_configs/minigptv2_finetune_mvtec.yaml
```

IV.2.5. Đánh giá

Trước khi tiến hành đánh giá mô hình đã được huấn luyện, các bạn cần kiểm tra lại vị trí lưu trữ trọng số. Vị trí mặc định theo cấu hình các file ở trên là: "`minigpt4/vivqa_outputs`" cho nhiệm vụ ViVQA và "`minigpt4/mvtec_outputs`" cho nhiệm vụ Detection. Tiếp theo, các bạn sẽ phải dùng đường dẫn file trọng số này thay thế trọng số pre-trained ở dòng thứ 9 tại file "`minigptv2_benchmark_evaluation`" ở thư mục "`eval_configs`". Sau đó, các bạn tiến hành quá trình đánh giá mô hình thông qua các câu lệnh sau đây:

Chạy đánh giá cho dữ liệu ViVQA:

```
1 python eval_vivqa.py --cfg-path ./eval_configs/minigptv2_benchmark_evaluation.yaml
```

Chúng ta xem xét độ đo accuracy để đánh giá cho bài toán VQA tiếng Việt này. Sử dụng file trọng số "`ckpt_checkpoint_stage3`" để đánh pre-trained model của nhóm tác giả và file trọng số "`minigpt4/vivqa_outputs/20250314075/checkpoint_1.pth`" - đổi lại cho phù hợp với vị trí lưu của các bạn để đánh giá mô hình sau khi chúng ta sau khi thực hiện supervised fine-tuning.

Chạy đánh giá cho dữ liệu MVTEC:

```
1 python eval_mvtec.py --cfg-path ./eval_configs/minigptv2_benchmark_evaluation.yaml
```

Đối với bài toán Detecion, chúng ta sẽ xem xét độ đo mAP để đo lường hiệu năng của hai mô hình. Tương tự như tác vụ VQA, các bạn lần lượt thay thế file trọng số vào file "["minigptv2_benchmark_evaluation.yaml"](#)" để tiến hành chạy đánh giá. Khi tổng hợp toàn bộ kết quả đánh giá, ta được một bảng thực nghiệm sau đây:

Dataset	Metric	Pre-trained	Fine-tuned
ViVQA	Accuracy	0.86%	59%
MVTec	mAP	10.16%	50.73%

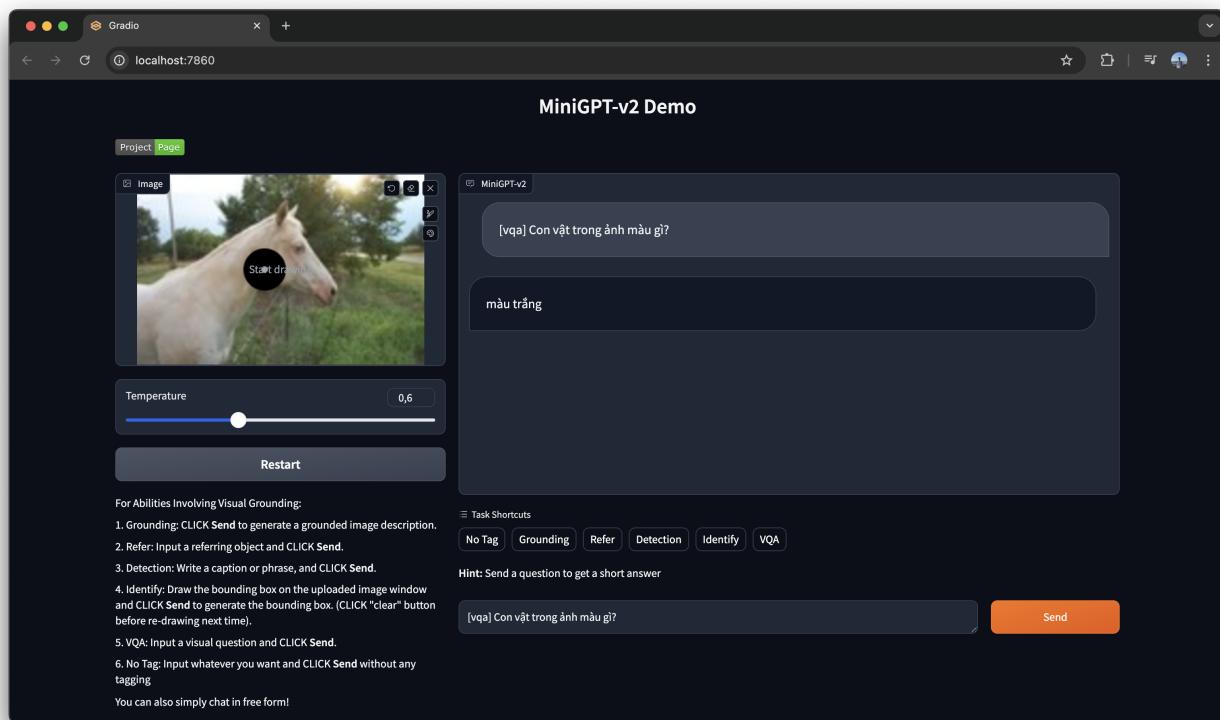
Bảng 2: Kết quả đánh giá mô hình MiniGPT-4 trước và sau khi thực hiện supervised fine-tuning (SFT) trên hai bộ dữ liệu ViVQA và MVTec. Mô hình được SFT trên bộ train và đánh giá trên bộ test tương ứng của các bộ dữ liệu.

IV.2.6. Demo

Cuối cùng, các bạn có thể tự chạy một ứng dụng web đơn giản được nhóm tác giả cung cấp, công việc của các bạn đó chính là thay thế đường dẫn file trọng số của file "["eval_configs/minigptv2_eval_vivqa.yaml"](#)" và khởi chạy demo nhiệm vụ VQA bằng lệnh:

```
1 python demo_v2.py --cfg-path eval_configs/minigptv2_eval_vivqa.yaml
```

Sau đó, các bạn tương tác với mô hình trên giao diện bằng cách chọn thẻ "["\[vqa\]"](#)" là không cần thêm câu prompt nào khác. Kết quả xem hình [6](#).

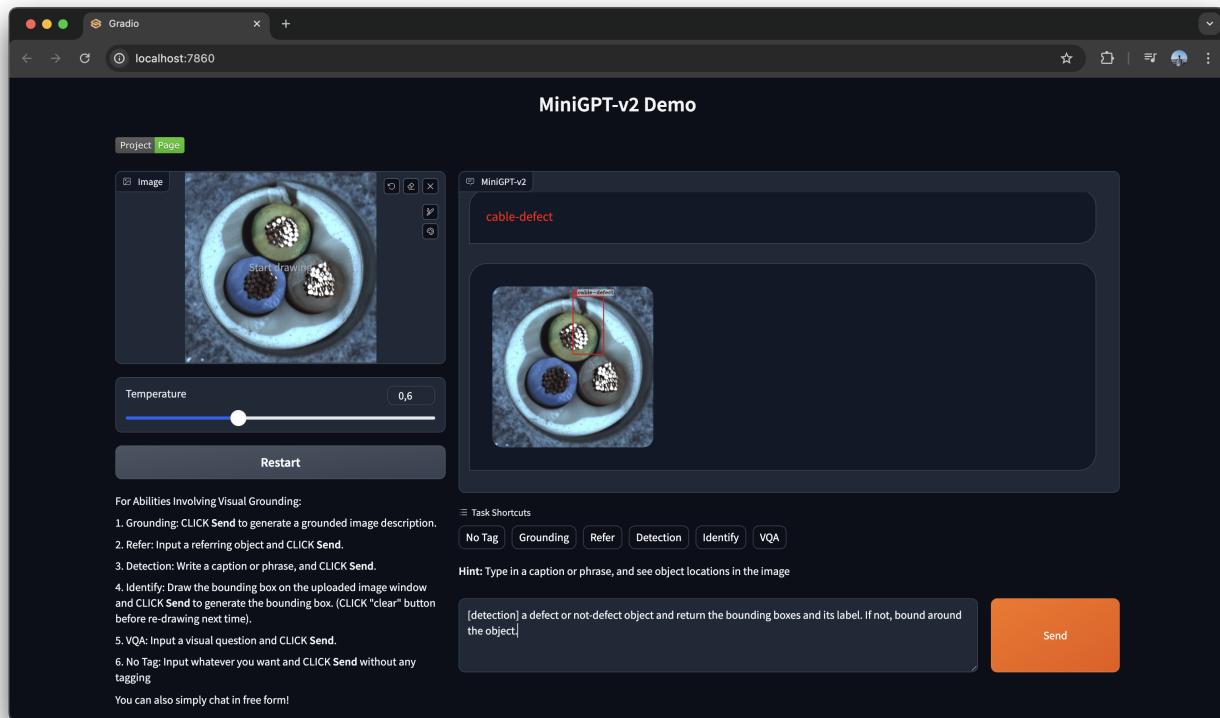


Hình 6: Kết quả VQA tiếng Việt sử dụng trọng số sau quá trình supervised fine-tuning của mô hình.

hoặc nhiệm vụ detection bằng lệnh:

```
1 python demo_v2.py --cfg-path eval_configs/minigptv2_eval_mvtec.yaml
```

Các bạn sử dụng thẻ [detection] và câu prompt sau: "**a defect or not-defect object and return the bounding boxes and its label. If not, bound around the object**". Kết quả xem hình 7.



Hình 7: Kết quả phát hiện lỗi sử dụng trọng số sau quá trình supervised fine-tuning của mô hình.

V. Tài liệu tham khảo

- [1] P. Bergmann, M. Fauser, D. Sattlegger **and** C. Steger, “MVTec AD — A Comprehensive Real-World Dataset for Unsupervised Anomaly Detection,” *in 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) 2019*, **pages** 9584–9592.
- [2] K. Q. Tran, A. T. Nguyen, A. T.-H. Le **and** K. V. Nguyen, “ViVQA: Vietnamese Visual Question Answering,” *in Proceedings of the 35th Pacific Asia Conference on Language, Information and Computation* Shanghai, China: Association for Computational Linguistics, **november** 2021, **pages** 546–554.
- [3] J. Chen, D. Zhu, X. Shen **and others**, “MiniGPT-v2: large language model as a unified interface for vision-language multi-task learning,” *arXiv preprint arXiv:2310.09478*, 2023.
- [4] A. Dosovitskiy, L. Beyer, A. Kolesnikov **and others**, “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale,” *ICLR*, 2021.
- [5] K. He, X. Zhang, S. Ren **and** J. Sun, “Deep Residual Learning for Image Recognition,” *in 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2016*, **pages** 770–778.
- [6] W.-L. Chiang, Z. Li, Z. Lin **and others**, *Vicuna: An Open-Source Chatbot Impressing GPT-4 with 90%* ChatGPT Quality*, 2023.
- [7] H. Touvron, T. Lavril, G. Izacard **and others**, “LLaMA: Open and Efficient Foundation Language Models,” *arXiv preprint arXiv:2302.13971*, 2023.
- [8] Y. Liu, M. Ott, N. Goyal **and others**, “RoBERTa: A Robustly Optimized BERT Pretraining Approach,” *ArXiv, jourvol* abs/1907.11692, 2019.
- [9] Z. Peng, W. Wang, L. Dong **and others**, “Kosmos-2: Grounding Multimodal Large Language Models to the World,” *ArXiv, jourvol* abs/2306.14824, 2023.
- [10] C. Schuhmann, R. Vencu, R. Beaumont **and others**, *LAION-400M: Open Dataset of CLIP-Filtered 400 Million Image-Text Pairs*, 2021.
- [11] H. Liu, C. Li, Q. Wu **and** Y. J. Lee, *Visual Instruction Tuning*, 2023.
- [12] P. Young, A. Lai, M. Hodosh **and** J. Hockenmaier, “From image descriptions to visual denotations: New similarity metrics for semantic inference over event descriptions,” *TACL, jourvol* 2, **pages** 67–78, 2014.

- *Hết* -