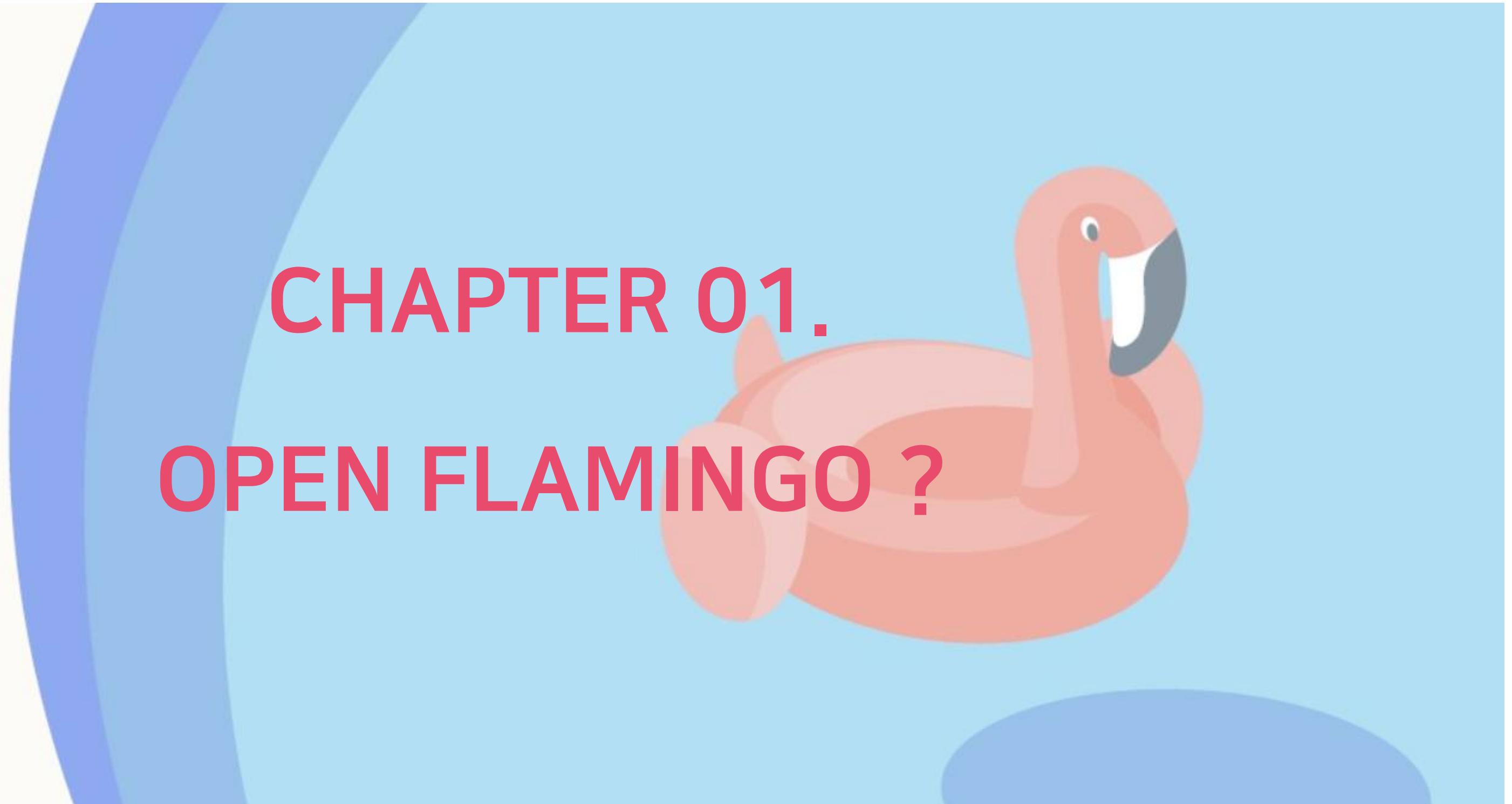




Flamingo | 권유진, 이선민, 임승섭

MULTIMODAL-GPT: A VISION AND LANGUAGE MODEL FOR DIALOGUE WITH HUMANS

Paper Review



CHAPTER 01. OPEN FLAMINGO ?





TOBIGS

Flamingo란?

Flamingo: a Visual Language Model for Few-Shot Learning, 2022, Jean-Baptiste Alayrac et al.

- 구글 딥마인드에서 발표한 **Visual Language Model**로, 이미지와 텍스트로 구성된 input을 받아 텍스트 output을 생성
- 다양한 Vision-Language Task에서 적은 수의 example로 학습해 fine-tuned model의 SOTA에 가까운 성능을 보임

BASE MODEL

BASE 논문에 대한 이해

MAIN PAPER

CODE SCRATCH

❖ **Flamingo: a Visual Language Model for Few-Shot Learning**



TOBIGS

0. Abstract

Flamingo: a Visual Language Model for Few-Shot Learning, 2022, Jean-Baptiste Alayrac et al.

- 몇 개의 annotated examples만으로도 novel tasks에 빠르게 적용할 수 있는 **Visual Language Models** (VLM), Flamingo를 제안
- Flamingo의 3가지 특징
 - 강력한 **pretrained vision-only, language-only** 모델을 연결
 - 정해진 순서가 아닌 **임의로 교차배열된 visual/textual data**를 처리할 수 있음
 - 이미지뿐만 아니라 비디오도 입력으로 사용할 수 있음 → **Perceiver-based 모델**이기에 가능

BASE MODEL

BASE 논문에 대한 이해

MAIN PAPER

CODE SCRATCH



TOBIGS

0. Abstract

Flamingo: a Visual Language Model for Few-Shot Learning, 2022, Jean-Baptiste Alayrac et al.

- 이러한 유연성 덕분에 Flamingo는 임의로 교차 배치된 텍스트와 이미지로 이루어진 **대규모 멀티모달 웹 데이터**로 학습이 가능함
- 또한, 대용량 데이터로부터 이미지와 텍스트의 일반적인 특징을 학습하여 다양한 멀티모달(텍스트, 이미지, 비디오) 벤치마크에서 뛰어난 **few-shot learning** 능력을 보여줌

BASE MODEL

BASE 논문에 대한 이해

MAIN PAPER

CODE SCRATCH



TOBIGS

0. Abstract

Flamingo: a Visual Language Model for Few-Shot Learning, 2022, Jean-Baptiste Alayrac et al.

- Flamingo Example

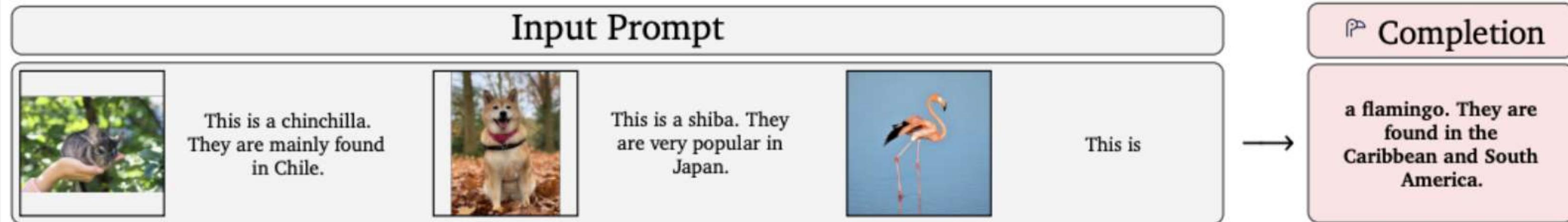
1. 동물 이미지 + 'This is' → 해당 동물에 대한 설명

—
BASE MODEL

BASE 논문에 대한 이해

MAIN PAPER

CODE SCRATCH



동물의 이미지를 보고 단순히 어떤 종인지 분류하는 것을 넘어 **이미지로 알 수 없는 동물의 특징까지 이해**



TOBIGS

0. Abstract

Flamingo: a Visual Language Model for Few-Shot Learning, 2022, Jean-Baptiste Alayrac et al.

- Flamingo Example

2. 미술작품 이미지 + 작품 관련 질문 + 'Answer :' → 답변

—
BASE MODEL
BASE 논문에 대한 이해

MAIN PAPER

CODE SCRATCH



What is the title of this painting? Answer: The Hallucinogenic Toreador.



Where is this painting displayed? Answer: Louvres Museum, Paris.



What is the name of the city where this was painted? Answer:



Arles.

명화를 보고 해당 그림이 그려진 도시를 맞춤



TOBIGS

0. Abstract

Flamingo: a Visual Language Model for Few-Shot Learning, 2022, Jean-Baptiste Alayrac et al.

- Flamingo Example

3. 표지판 이미지 + 'Output : ' → 표지판 속 글자

—
BASE MODEL
BASE 논문에 대한 이해

MAIN PAPER

CODE SCRATCH



이미지 속 글자를 읽을 수 있음



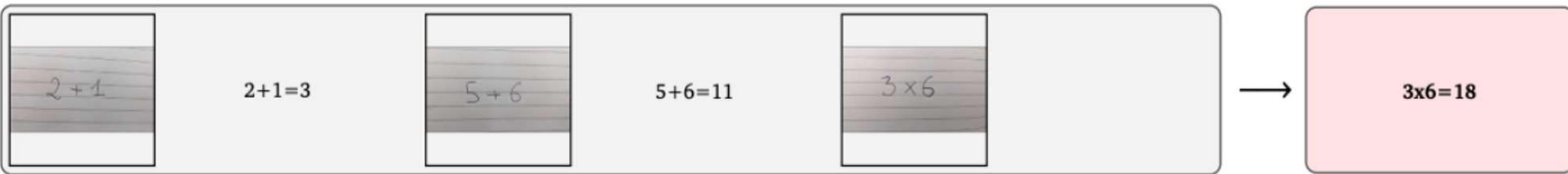
TOBIGS

0. Abstract

Flamingo: a Visual Language Model for Few-Shot Learning, 2022, Jean-Baptiste Alayrac et al.

- Flamingo Example

4. 수식 이미지 + 수식 글자 → 수식의 답



이미지 속 글자를 읽는 것을 넘어 '**수학 계산**'을 수행



TOBIGS

0. Abstract

Flamingo: a Visual Language Model for Few-Shot Learning, 2022, Jean-Baptiste Alayrac et al.

- Flamingo Example

5. DALL-E 2로 만든 이미지 + 'Output : ' → 이미지 설명

—
BASE MODEL

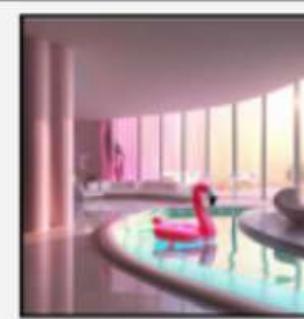
BASE 논문에 대한 이해

MAIN PAPER

CODE SCRATCH



Output: A propaganda poster depicting a cat dressed as French emperor Napoleon holding a piece of cheese.



Output: A pink room with a flamingo pool float.



Output:

A portrait of Salvador Dalí with a robot head.

학습된 web corpora dataset에 없는 생성된 이미지를 보고 납득할만한 caption을 생성



TOBIGS

0. Abstract

Flamingo: a Visual Language Model for Few-Shot Learning, 2022, Jean-Baptiste Alayrac et al.

- Flamingo Example

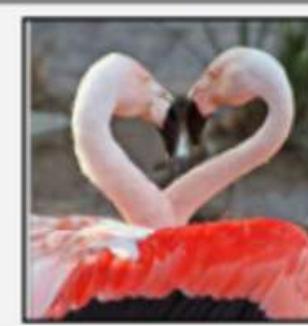
6. 이미지 + 프랑스어 설명 → 프랑스어 설명



Les sanglots longs des
violons de l'automne
blessent mon cœur
d'une langueur
monotone.



Pour qui sont ces
serpents qui sifflent sur
vos têtes?



→
Je suis un cœur qui
bat pour vous.

—
BASE MODEL

BASE 논문에 대한 이해

MAIN PAPER

CODE SCRATCH

영어뿐만 아니라 프랑스어로 문장 이해 및 생성 가능



TOBIGS

0. Abstract

Flamingo: a Visual Language Model for Few-Shot Learning, 2022, Jean-Baptiste Alayrac et al.

- Flamingo Example

7. 동물 이미지 + '동물 이름 : 수' → 동물 이름과 수



이미지 속 객체들을 분리하여 파악하고 그 수를 셀 수 있음



TOBIGS

0. Abstract

Flamingo: a Visual Language Model for Few-Shot Learning, 2022, Jean-Baptiste Alayrac et al.

- Flamingo Example

8. 문장 + 이미지 + 문장 + 이미지 + 문장 → 문장

—
BASE MODEL

BASE 논문에 대한 이해

MAIN PAPER

CODE SCRATCH



'이미지 + 문장'과 같이 정해진 입력 형식이 아닌 다른 순서의 입력도 처리 가능

('Dreams from my Father' - 오바마 전 대통령의 자서전)



TOBIGS

0. Abstract

Flamingo: a Visual Language Model for Few-Shot Learning, 2022, Jean-Baptiste Alayrac et al.

- Flamingo Example

9. 비디오 + 질문 → 답변



비디오와 질문을 input으로 받아 이에 대한 답변 생성



TOBIGS

0. Abstract

Flamingo: a Visual Language Model for Few-Shot Learning, 2022, Jean-Baptiste Alayrac et al.

- Flamingo Example

첫번째 대화 : 마지막 부분을 보면, 이 사진이 왜 놀랍냐는 질문에 대해 'teddy bears는 보통 달에서 발견할 수 없기 때문'이라고 답함.

teddy bear와 달의 일반적인 특징과 surprising이라는 감정의 의미도 이해하는 것을 알 수 있음

두번째 대화 : 플라밍고를 다른 스타일 (cartoon, real image, 3D)로 표현한 것을 구분함

—
BASE MODEL

BASE 논문에 대한 이해

MAIN PAPER

CODE SCRATCH

The image shows a visual language model interface with two main sections. The left section displays a photo of two teddy bears sitting on the moon, with a series of text bubbles representing a conversation. The right section displays three different images of flamingos (a cartoon, a real photo, and a 3D model) in separate boxes, each with a series of text bubbles representing a conversation. The text in the bubbles is as follows:

Left Section (Teddy Bears on the Moon):

- This is a picture of two teddy bears on the moon.
- What are they doing?
- They are having a conversation.
- What object are they using?
- It looks like a computer.
- Is this surprising?
- Yes, it is surprising.
- Why is this picture surprising to you?
- I think it is surprising because teddy bears are not usually found on the moon.

Right Section (Flamingos):

- What is the common thing about these three images?
- They are all flamingos.
- What is the difference between these three images?
- The first one is a cartoon, the second one is a real flamingo, and the third one is a 3D model of a flamingo.



TOBIGS

0. Abstract

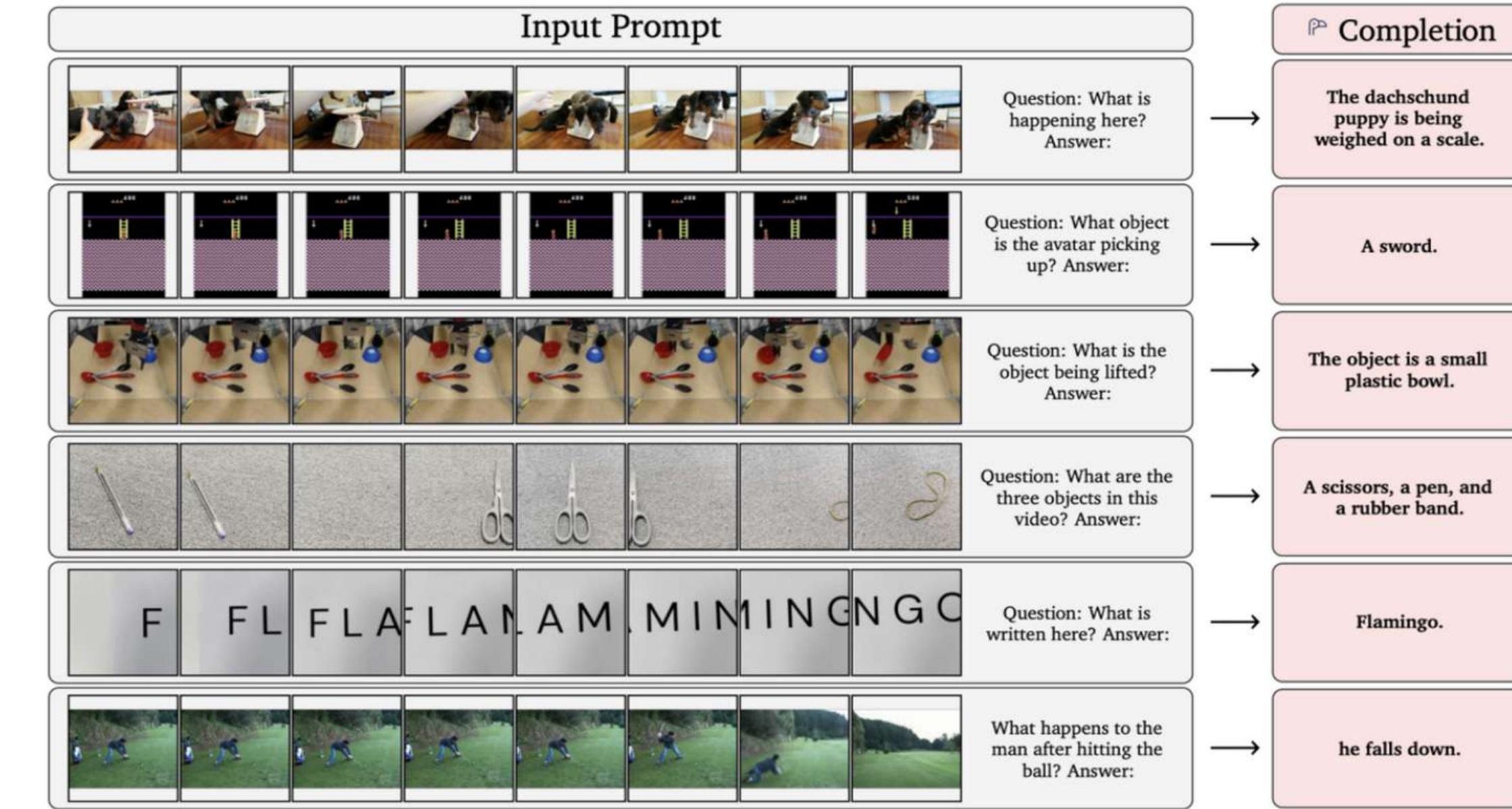
Flamingo: a Visual Language Model for Few-Shot Learning, 2022, Jean-Baptiste Alayrac et al.

BASE MODEL

BASE 논문에 대한 이해

MAIN PAPER

CODE SCRATCH



비디오에서 1FPS로 샘플링된 frame들과 질문을 input prompt로 넣어줬을 때, Flamingo는 이에 대한 답을 생성함

영상에서 어떤 객체가 있고, 어떤 일이 일어나고 있는지 파악하고, 한 frame에 나오지 않더라도 이를 인지함.



TOBIGS

0. Abstract

Flamingo: a Visual Language Model for Few-Shot Learning, 2022, Jean-Baptiste Alayrac et al.

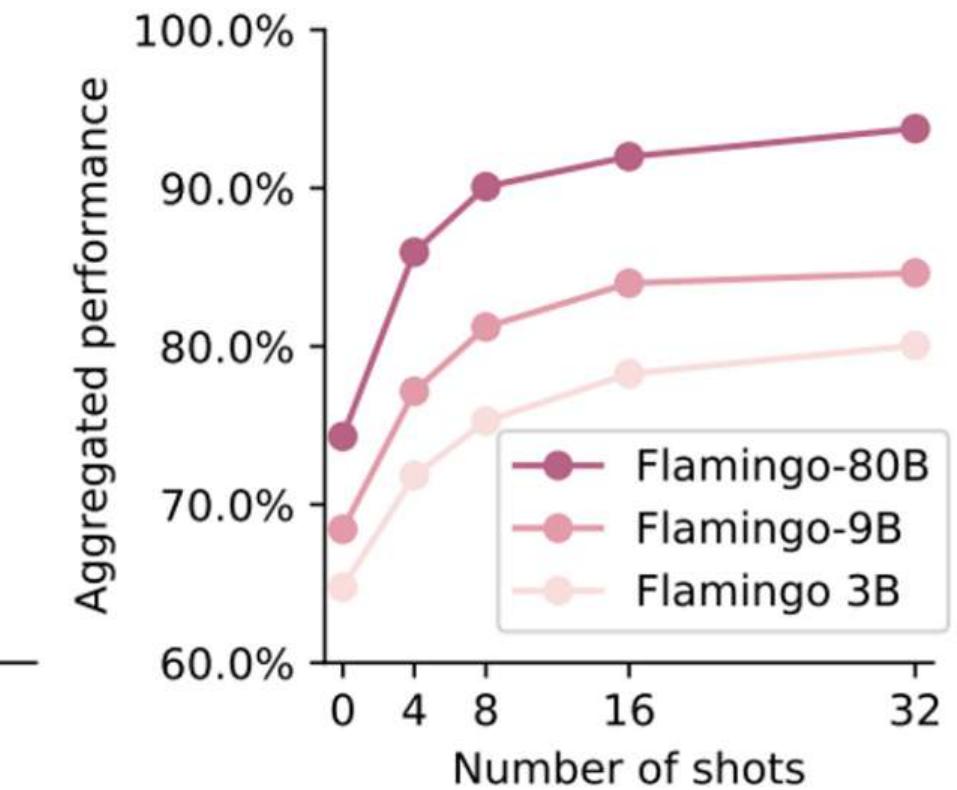
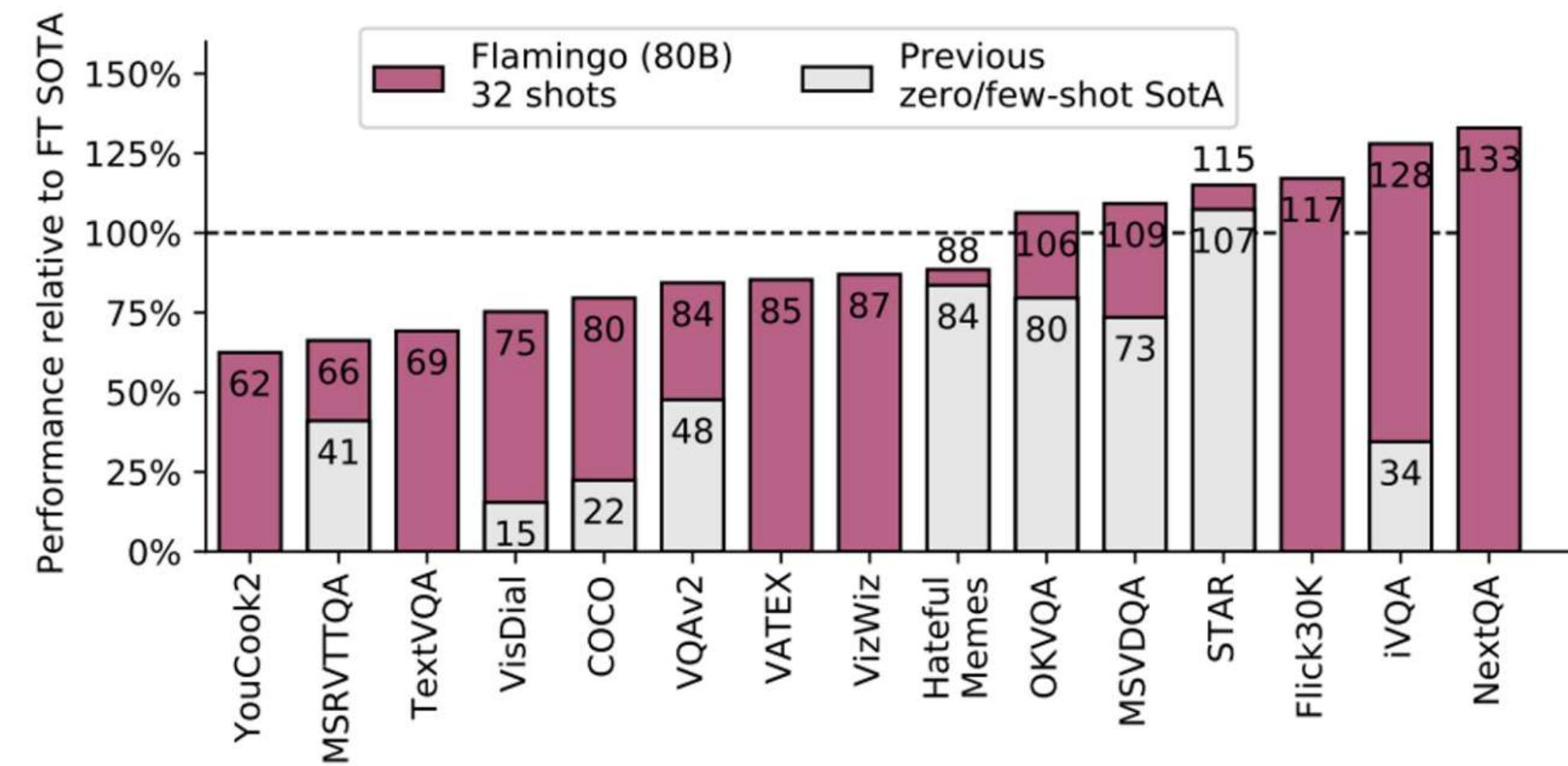
- 기존 Fine-tuned SOTA와의 비교

BASE MODEL

BASE 논문에 대한 이해

MAIN PAPER

CODE SCRATCH



간단하게 축약한 Flamingo의 성능비교차트.

16개 중 6개는 fine-tuned 모델보다 뛰어나고, 기존 zero/few-shot에선 모두 SOTA를 달성



TOBIGS

1. Introduction

Flamingo: a Visual Language Model for Few-Shot Learning, 2022, Jean-Baptiste Alayrac et al.

최근 CLIP과 같은 모델은 novel task에 대해 뛰어난 zero-shot adaptation 능력을 보임

—
BASE MODEL

BASE 논문에 대한 이해

하지만 text와 image 사이의 **similarity score**만 제공하기에 **classification** 같은 문제에서는 효과

적이었지만 텍스트를 생성해야하는 captioning or visual question-answering과 같은

open-ended task에선 취약함

CODE SCRATCH

따라서 본 논문은 **open-ended vision and language task**에 대해 **few-shot learning**을 통해

광범위하게 적용가능한 **Visual Language Model(VLM)**인 Flamingo를 제안



TOBIGS

1. Introduction

Flamingo: a Visual Language Model for Few-Shot Learning, 2022, Jean-Baptiste Alayrac et al.

• Related work

최근 NLP에선 **large-language model (LLM)**이 **few-shot learner**로써 좋은 성능을 보인다는 연구들이 있었음.

본 논문에선 이게 **image/video understanding**에서도 **동일하게 적용가능함**을 보임.

이제 앞선 예시를 **visually-conditioned text prediction** 문제로 바꾼다는 의미.

LLM과의 차이점은 **multimodal prompt input**을 받음

—
BASE MODEL
BASE 논문에 대한 이해

MAIN PAPER

CODE SCRATCH



TOBIGS

1. Introduction

Flamingo: a Visual Language Model for Few-Shot Learning, 2022, Jean-Baptiste Alayrac et al.

• Features

- Flamingo는 **pre-trained and frozen vision/language model**을 사용하고 이 둘을 이어주기 위한 새로운 모듈을 이용. 이 때, 이 모듈이 **Perceiver** 기반이기 때문에 고해상도의 이미지나 비디오도 모델 구조의 변화없이 처리가능함
 - 어떻게? → **Attention** 기반의 **perceiver 모델**로 다양한 타입의 데이터셋을 입력으로 받을 수 있음 (자세한 내용은 논문 참조 (Perceiver: General Perception with Iterative Attention))
- LLM의 성능에서 중요한 요소는 결국 **대용량 텍스트 데이터에 대한 학습**임. 이는 Flamingo의 성능에 있어서도 중요
- 따라서 Flamingo는 웹 상에서 잘 골라진 **large-scale multimodal data**를 통해 학습

BASE MODEL

BASE 논문에 대한 이해

MAIN PAPER

CODE SCRATCH



TOBIGS

2. Approach

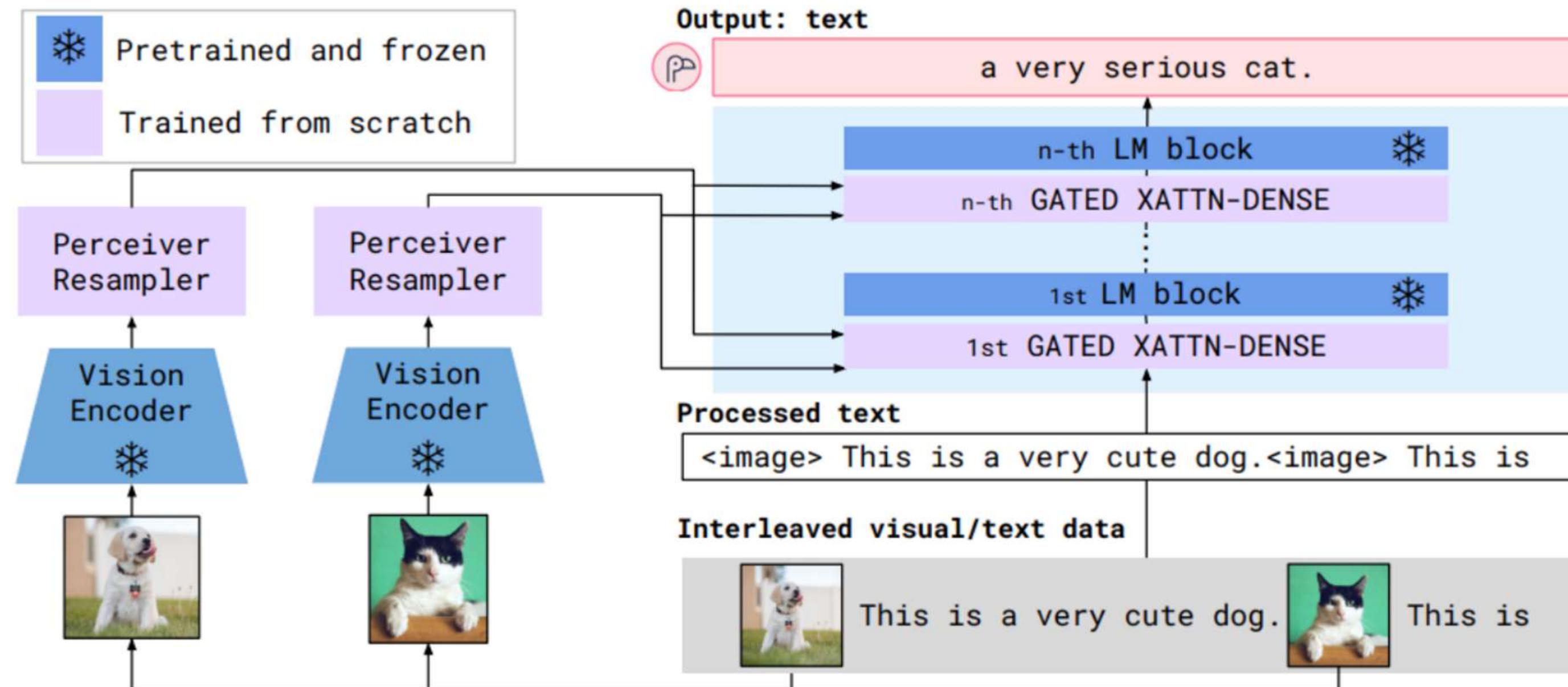
Flamingo: a Visual Language Model for Few-Shot Learning, 2022, Jean-Baptiste Alayrac et al.

BASE MODEL

BASE 논문에 대한 이해

MAIN PAPER

CODE SCRATCH



Flamingo architecture overview



TOBIGS

2. Approach

Flamingo: a Visual Language Model for Few-Shot Learning, 2022, Jean-Baptiste Alayrac et al.

- **Overview**

Flamingo는 간단히 요약하자면 **text interleaved with images/videos**를 입력으로 받아 **free-form text**를 출력하는 visual language 모델

이를 위해 Flamingo는 아키텍처에서와 같이 각각의 **pretrained and frozen Vision Encoder/LM**을 적절하게 연결해줌

- **text interleaved with images/videos**란? : 텍스트와 이미지 또는 비디오가 교차되어 함께 사용되는 형태를 의미

—
BASE MODEL
BASE 논문에 대한 이해

MAIN PAPER

CODE SCRATCH



TOBIGS

2. Approach

Flamingo: a Visual Language Model for Few-Shot Learning, 2022, Jean-Baptiste Alayrac et al.

- Overview

Flamingo 방법론 요약

1. Visual processing and the Perceiver Resampler

- 먼저, Vision Encoder의 출력인 다양한 형태의 spatio-temporal features를 Perceiver Resampler로 보내 고정된 형태의 visual tokens을 얻음
 - spatio-temporal features란? : 공간적 및 시간적인 특징을 나타내는 데이터의 속성

—
BASE MODEL

BASE 논문에 대한 이해

MAIN PAPER

CODE SCRATCH



TOBIGS

MultiModal-GPT란?

MultiModal-GPT: A Vision and Language Model for Dialogue with Humans, 2023, Tao Gong et al.

Risks

- Pre-train 과정에서 엄청난 cost가 들어가고, 기존 모델들에 비해 큰 모델을 사용함
- 실제 서비스에 적용하기 위해 경량화, 효율적인 사전학습 방안 등 많은 연구가 필요해 보임

BASE MODEL

MAIN PAPER

MAIN 논문에 대한 이해

CODE SCRATCH

Summary

- 최소한의 task-specific training data만으로도 image/video tasks에 빠르게 적용가능한 범용적인 모델, Flamingo를 제안.
- 기존 vision 모델에 비해 엄청난 flexibility를 가짐.



TOBIGS

2. Approach

Flamingo: a Visual Language Model for Few-Shot Learning, 2022, Jean-Baptiste Alayrac et al.

- Overview

Flamingo 방법론 요약

—
BASE MODEL

BASE 논문에 대한 이해

MAIN PAPER

CODE SCRATCH

2. Conditioning frozen language models on visual representations

- 얼어진 visual tokens는 frozen LM 앞에서 text와 함께 GATED XATTN-DENSE 블럭에서 cross-attention이 수행됨
- 즉, GATED XATTN-DENSE 블럭은 LM이 visually-conditioned next-token prediction을 가능하게 만들어줌
 - GATED XATTN-DENSE 블럭이란? → "GATED"은 게이트 기법을 의미하며, XATTN-DENSE는 "Cross Attention Dense"를 의미



TOBIGS

2. Approach

Flamingo: a Visual Language Model for Few-Shot Learning, 2022, Jean-Baptiste Alayrac et al.

- Overview

Flamingo 방법론 요약

—
BASE MODEL

BASE 논문에 대한 이해

MAIN PAPER

CODE SCRATCH

3. Multi-visual input support: per-image/video attention masking

- 교차배열된 visual/text 데이터로 학습되는 구조 덕분에 Flamingo는 자연스럽게 GPT-3의 few-shot text prompting처럼 in-context few-shot learning을 수행할 수 있음

$$p(y|x) = \prod_{l=1}^L p(y_l|y_{<l}, x_{\leq l})$$

- 위의 식이 Flamingo가 interleaved images/videos x 가 주어졌을 때 text y 를 생성하는 likelihood
- 여기서 l 은 입력 텍스트의 language token index이다.



TOBIGS

2. Approach

Flamingo: a Visual Language Model for Few-Shot Learning, 2022, Jean-Baptiste Alayrac et al.

1. 사전 학습한 pre-trained 모델 활용(그림에서 frozen으로 표시된 부분)

Vision side

- CLIP과 같이 text-image contrastive learning으로 vision encoder(NFNet)를 사전학습 한 후, Flamingo framework 안에서는 추가적인 학습을 하지 않음.
- 이러한 contrastive learning을 통해 vision encoder는 객체의 color, shape, nature, position 등 시각적 특징을 표현하는 semantic spatial feature를 추출할 수 있게 됨

—
BASE MODEL

BASE 논문에 대한 이해

MAIN PAPER

CODE SCRATCH



TOBIGS

2. Approach

Flamingo: a Visual Language Model for Few-Shot Learning, 2022, Jean-Baptiste Alayrac et al.

1. 사전 학습한 pre-trained 모델 활용(그림에서 frozen으로 표시된 부분)

—
BASE MODEL

BASE 논문에 대한 이해

Language side

- large and diverse text corpus로 학습된 autoregressive language model(LM, Chinchilla)을 사용
- 이처럼 사전학습한 모델을 가져와 기존 모델이 갖고 있던 강력한 언어 생성 능력을 얻고, LM weight에 저장된 수많은 양의 knowledge를 사용할 수 있게 됨
 - 각 uni-modal side의 장점을 유지하기 위해 freeze 했다는 것을 잊지 말아야 함 !!

MAIN PAPER

CODE SCRATCH



TOBIGS

2. Approach

Flamingo: a Visual Language Model for Few-Shot Learning, 2022, Jean-Baptiste Alayrac et al.

2. 두 개의 학습 가능한 components를 통해 vision-language side를 연결

—
BASE MODEL

BASE 논문에 대한 이해

Perceiver Resampler

MAIN PAPER

CODE SCRATCH

- Vision encoder로부터 얻은 다양한 사이즈를 갖는 spatio-temporal features를 고정된 작은 사이즈(64 sequence)의 visual tokens로 mapping하는 역할(그래서 resampler라는 이름을 가짐)
- 이처럼 출력 사이즈를 줄여 vision-text cross attention의 계산량을 줄일 수 있고, 특히 이러한 과정은 여러 개의 긴 비디오를 처리할 때 큰 이점으로 작용



TOBIGS

2. Approach

Flamingo: a Visual Language Model for Few-Shot Learning, 2022, Jean-Baptiste Alayrac et al.

2. 두 개의 학습 가능한 components를 통해 vision-language side를 연결

—
BASE MODEL

BASE 논문에 대한 이해

Gated Cross Attention layers

MAIN PAPER

- Attention의 key, value를 vision feature로부터 얻고, query로 language feature를 사용해 scratch부터 학습
- 이 layer에서는 시각적 정보와 언어적 정보를 통합해 LM이 다음 토큰을 예측하는 task를 수행하는데 도움을 줌

CODE SCRATCH



TOBIGS

3. Conclusion

Flamingo: a Visual Language Model for Few-Shot Learning, 2022, Jean-Baptiste Alayrac et al.

Limitations

- 사전학습한 언어 모델(pretrained language model)을 freeze해서 사용하기 때문에 이 모델의 단점 또한 그대로 유지됨
- 또한 transformer-based language model의 경우 매우 긴 test sequence가 들어오면 성능이 떨어짐. 뿐만 아니라 아래 그림과 같이 사진을 제대로 반영 못하거나 근거 없는 추측을 하는 경우도 발생함

BASE MODEL
BASE 논문에 대한 이해

MAIN PAPER

CODE SCRATCH



Question: What is on the phone screen? Answer:



Question: What can you see out the window? Answer:



Question: Whom is the person texting? Answer:

A text message from a friend.

A parking lot.

The driver.



TOBIGS

3. Conclusion

Flamingo: a Visual Language Model for Few-Shot Learning, 2022, Jean-Baptiste Alayrac et al.

Risks

- Pre-train 과정에서 엄청난 cost가 들어가고, 기존 모델들에 비해 큰 모델을 사용함
- 실제 서비스에 적용하기 위해 경량화, 효율적인 사전학습 방안 등 많은 연구가 필요해 보임

MAIN PAPER

Summary

- 최소한의 task-specific training data만으로도 image/video tasks에 빠르게 적용가능한 범용적인 모델, Flamingo를 제안.
- 기존 vision 모델에 비해 엄청난 flexibility를 가짐.

CODE SCRATCH



CHAPTER 02.

MULTIMODAL-GPT





TOBIGS

MultiModal-GPT란?

MultiModal-GPT: A Vision and Language Model for Dialogue with Humans, 2023, Tao Gong et al.

BASE MODEL

MAIN PAPER

MAIN 논문에 대한 이해

CODE SCRATCH

Image + Text



Answer !

이미지와 텍스트를 모두 넣었을 때,

이미지와 텍스트의 정보를 모두 이용하여

답변을 반환하는 모델

How to make this dish?



To make this dish, you will need to prepare the ingredients and follow the recipe. First, you will need to prepare the pasta sauce by combining tomato sauce, garlic, basil, and parmesan cheese. Next, you will need to assemble the lasagna by layering the noodles, sauce, and mozzarella cheese. Finally, you will need to bake the lasagna in the oven until it is fully cooked and the cheese is melted.

If I can not cook, where can I eat it?

You can order this dish from a restaurant or have it catered. Alternatively, you can order the ingredients online and follow the recipe to prepare it at home.

Figure 2: The MultiModal-GPT can give a recipe to bake lasagna, and tell users where to eat it.



TOBIGS

0. Abstract

MultiModal-GPT: A Vision and Language Model for Dialogue with Humans, 2023, Tao Gong et al.

BASE MODEL

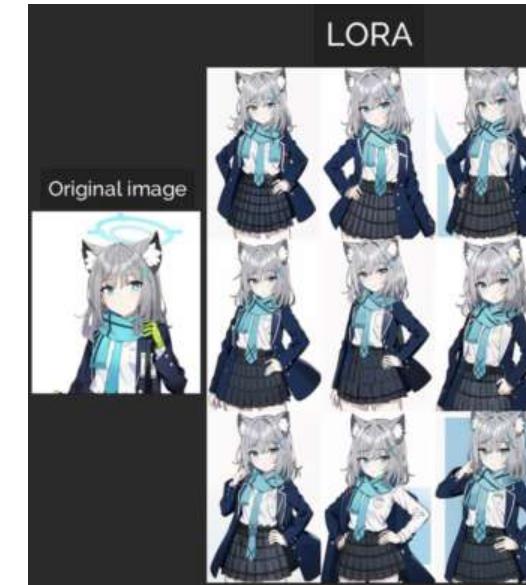
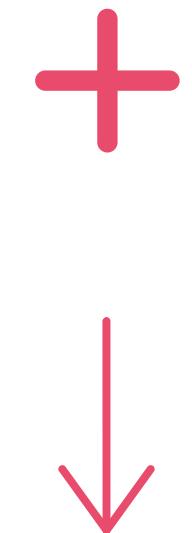
Flamingo: a Visual Language Model
for Few-Shot Learning

OpenFlamingo

MAIN PAPER

MAIN 논문에 대한 이해

CODE SCRATCH



LoRA

MultiModal-GPT

- 새로 제시된 이론은 없음
- Instruction 형식을 지정하여 Fine-Tuning함



TOBIGS

0. Abstract

MultiModal-GPT: A Vision and Language Model for Dialogue with Humans, 2023, Tao Gong et al.

- **MultiMoal-GPT의 성능을 높이기 위해 논문에서 사용한 방법**

: Language-only instruction following data과 Visual-language instructions following data를 이용해서 **joint training**을 진행함

BASE MODEL

MAIN PAPER

MAIN 논문에 대한 이해

CODE SCRATCH

- **Joint Training**이란?

: 여러 개의 loss들을 하나의 값으로 더해서 최종 loss로 사용하는 훈련 방식

Loss Function:

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \quad (1)$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right] \quad (2)$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2 \quad (3)$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{noobj}} (C_i - \hat{C}_i)^2 \quad (4)$$

$$+ \sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \quad (5)$$

total loss = loss1 + loss2 + loss3 + loss4 + loss5

-> 여러 개의 loss들을 더해서 모든 task들을 한 번에 학습하는 방식



TOBIGS

1. Introduction

MultiModal-GPT: A Vision and Language Model for Dialogue with Humans, 2023, Tao Gong et al.

- **MultiModal-GPT Model의 특징**

- pre-trained on an extensive dataset of image-text pairs

: Image와 Text pair로 되어있는 광범위한 데이터셋을 이용하여 pretraining을 진행함

- showcasing robust **few-shot** visual comprehension capabilities

: 모델이 적은 양의 데이터만으로도 시각 정보를 잘 이해하고 해석할 수 있는 능력을 갖추고 있음

BASE MODEL

MAIN PAPER

MAIN 논문에 대한 이해

CODE SCRATCH

- **Few-Shot Learning Ability**가 중요한 이유?

: 기계 학습 모델이 새로운 문제나 도메인에 대해 빠르게 적응하고 학습할 수 있는 잠재력을 가지고 있음을 의미하기 때문

- 적은 양의 데이터로 문제를 일반화하기 때문에 다른 모델에 적용하기 쉬움!



TOBIGS

1. Introduction

MultiModal-GPT: A Vision and Language Model for Dialogue with Humans, 2023, Tao Gong et al.

- **MultiModal-GPT Model의 한계점**

BASE MODEL

MAIN PAPER

MAIN 논문에 대한 이해

"It lacks the capacity to engage in zero-shot multturn image-text dialogues"

이전 대화의 문맥과 이미지-텍스트 상호작용을 활용하여 자연스러운 대화를 이어가는 능력이 부족하다

CODE SCRATCH

- **Zero-shot Learning**이란?

: 모델이 학습 과정에서 배우지 않은 작업을 수행하는 것



TOBIGS

1. Introduction

MultiModal-GPT: A Vision and Language Model for Dialogue with Humans, 2023, Tao Gong et al.

- **MultiModal-GPT 논문 요약**

- **논문에서 사용하는 방법**

- : Fine-tune OpenFlamingo using comprehensive datasets of image and text

- instructions

- > Instruction이 추가된 이미지와 텍스트 데이터셋을 이용하여 OpenFlamingo 모델을 Fine-Tuning한다.

- **논문의 목표**

- : To conduct conversations that more closely align with **human preferences**

- > 사람의 대화와 유사하게 대화 가능한 모델을 만든다.

BASE MODEL

MAIN PAPER

MAIN 논문에 대한 이해

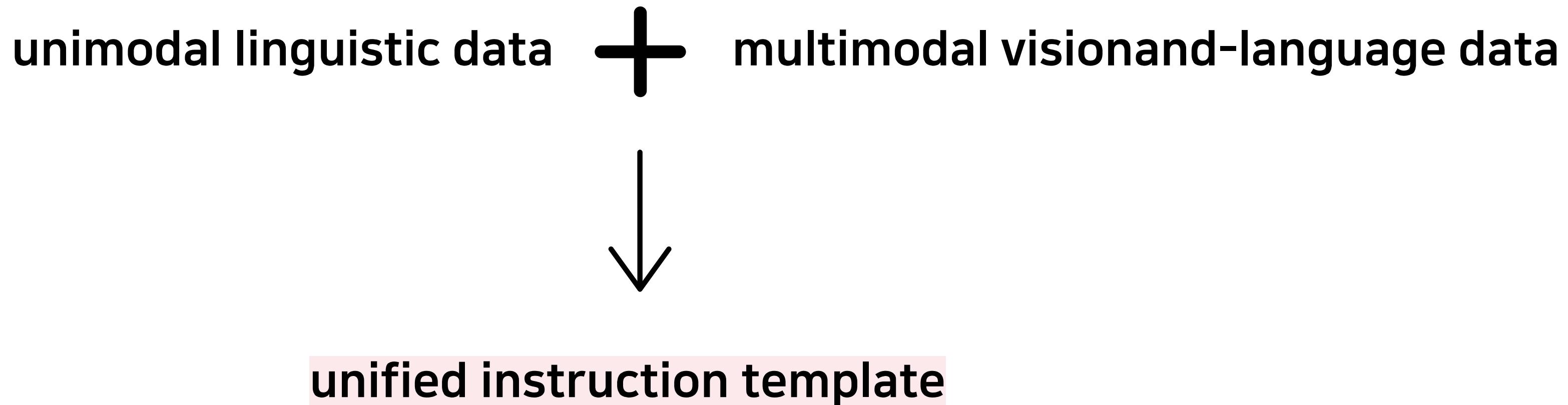
CODE SCRATCH



TOBIGS

2. Unified Instruction Template

MultiModal-GPT: A Vision and Language Model for Dialogue with Humans, 2023, Tao Gong et al.



- 데이터 양식 이용함
- 데이터의 심오한 뜻을 이해할 수 있도록 함



2. Unified Instruction Template

MultiModal-GPT: A Vision and Language Model for Dialogue with Humans, 2023, Tao Gong et al.

2.1 Language-only Instruction Template

: 자연어만 들어왔을 경우 프롬프트 처리 양식

- Vision과 language를 모두 다루는 프롬프트 처리하기 위해 각각 프롬프트 데이터 형식을 지정함

BASE MODEL

MAIN PAPER

MAIN 논문에 대한 이해

CODE SCRATCH

Evaluation

: **the Dolly 15K, Alpaca GPT4 datasets** 사용함

- Instruction Tuning with GPT-4 : English Instruction-Following data
 - Data generated by GPT-4 for building an instruction-following LLMs
 - Instruction-based tasks를 할때, 성능을 높일 수 있게 도와주는 데이터셋
(<https://github.com/Instruction-Tuning-with-GPT-4/GPT-4-LLM>)



TOBIGS

2. Unified Instruction Template

MultiModal-GPT: A Vision and Language Model for Dialogue with Humans, 2023, Tao Gong et al.

2.1 Language-only Instruction Template

BASE MODEL

Language-only 프롬프트 예시

- instruction-following format으로 작성됨

<BOS> Below is an instruction that describes a task. Write a response that appropriately completes the request

Instruction: {instruction}
Input: {input}
Response: {response} <EOS>

Table 1: The input sequence of language data used to train the model. The {instruction}, {input} and {response} are texts from the source data. Only the {response} part and <EOS> token will be calculated loss.

MAIN PAPER

MAIN 논문에 대한 이해

CODE SCRATCH



TOBIGS

2. Unified Instruction Template

MultiModal-GPT: A Vision and Language Model for Dialogue with Humans, 2023, Tao Gong et al.

2.2 Vision and Language Instruction Template

: 이미지와 자연어가 모두 들어왔을 경우 프롬프트 처리 양식

- 정형화된 양식을 사용하면 모델이 학습하기 쉽고, 올바른 대답을 내놓기 쉬워지므로 모델의 성능이 올라감

BASE MODEL

MAIN PAPER

MAIN 논문에 대한 이해

Dataset

: LLaVA, Mini-GPT4, A-OKVQA, COCO Caption, OCR VQA

(vision and language instruction-following datasets)

- applications 과 domains이 다양하므로 모델의 성능을 높여줌
- COCO Caption은 원래 instruction이 포함되어 있지 않았는데, GPT4를 이용해서 Instruction을 추가함

CODE SCRATCH



2. Unified Instruction Template

MultiModal-GPT: A Vision and Language Model for Dialogue with Humans, 2023, Tao Gong et al.

2.2 Vision and Language Instruction Template

Vision and Language 프롬프트 예시

- instruction-following format으로 작성됨

MAIN PAPER

MAIN 논문에 대한 이해

CODE SCRATCH

<BOS> Below is an instruction that describes a task. Write a response that appropriately completes the request

Image: <image_token>
Instruction: {question}
Response: {response}<EOS>
Instruction: {question}
Response: {response} <EOS>

Table 2: The input sequence of vision and language data used to train the model. The **{question}** and **{response}** are texts from the source data. **<image_token>** is a token denoting the existence of image. Note that there are multi-round dialogues if the dataset has. Only the **{response}** part and **<EOS>** token will be calculated loss.



TOBIGS

3. Method

MultiModal-GPT: A Vision and Language Model for Dialogue with Humans, 2023, Tao Gong et al.

3.1 Architecture

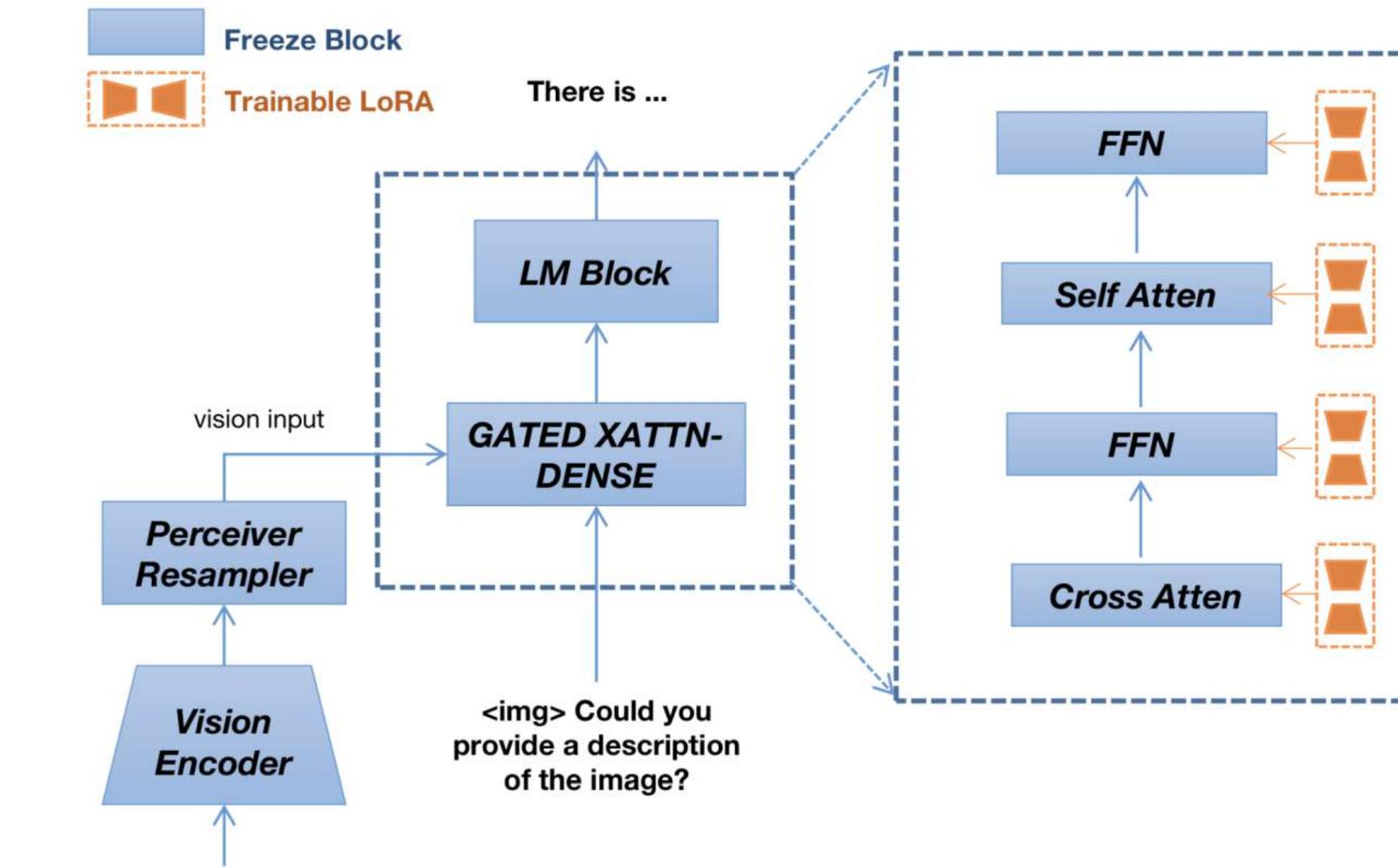
- based on the open-flamingo model

BASE MODEL

MAIN PAPER

MAIN 논문에 대한 이해

CODE SCRATCH





TOBIGS

3. Method

MultiModal-GPT: A Vision and Language Model for Dialogue with Humans, 2023, Tao Gong et al.

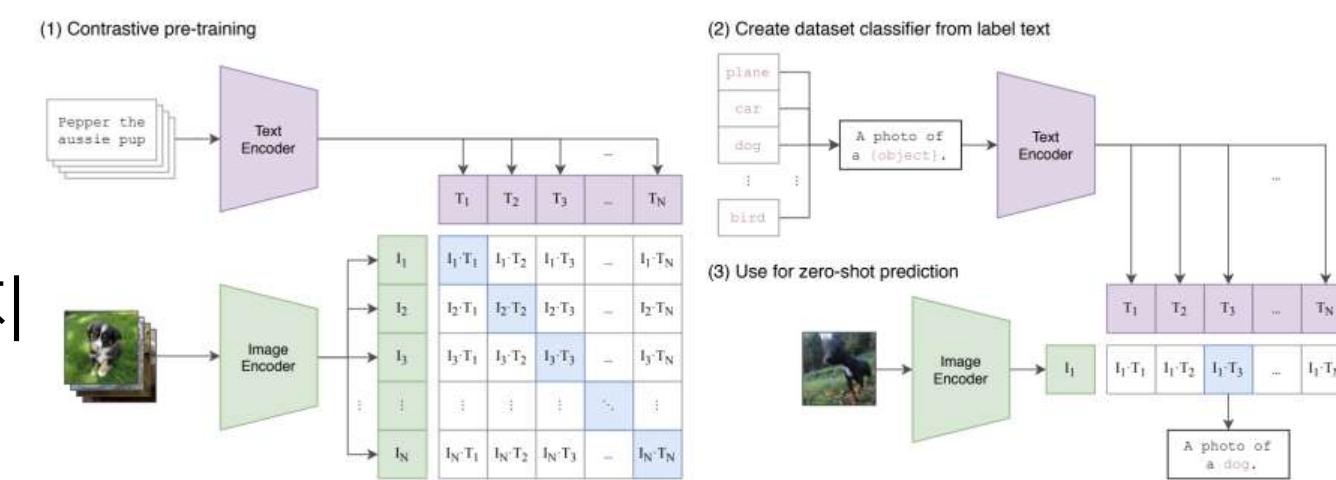
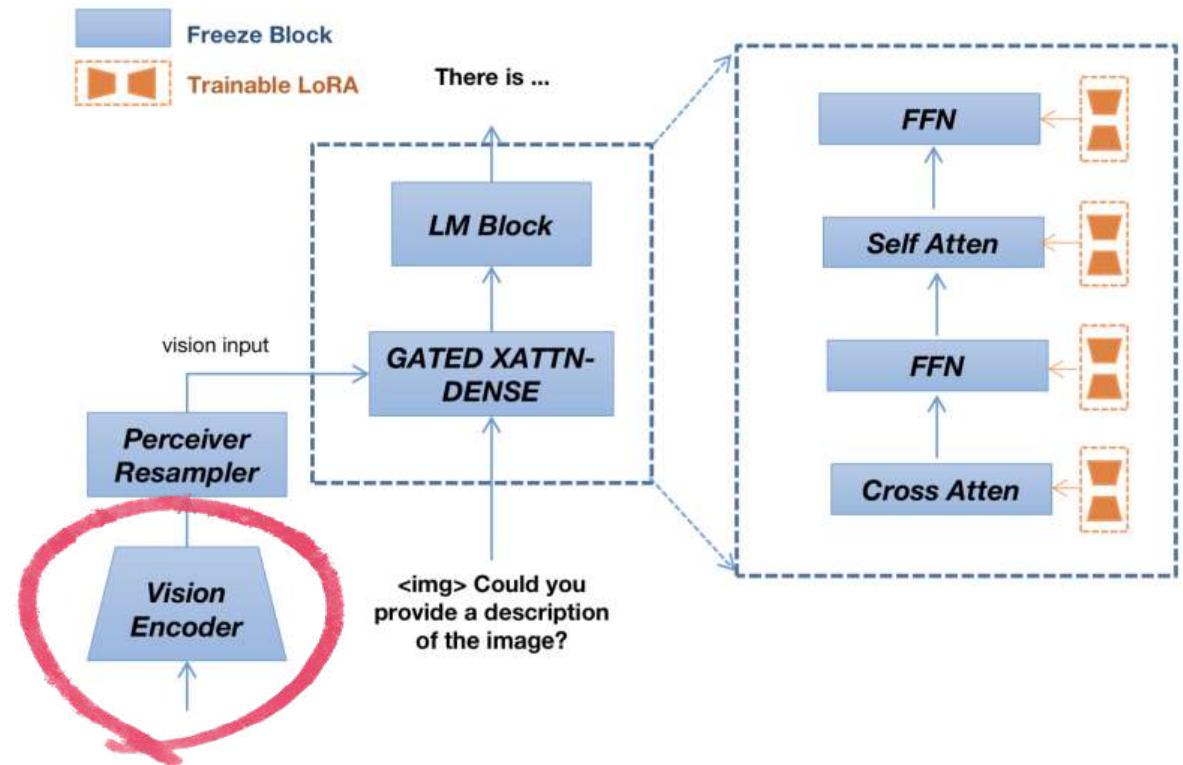
3.1 Architecture

Vision Encoder CLIP

: 이미지와 텍스트를 함께 사용하여 사전 훈련된 비지도 학습을 수행함

Image Decoder LLaMA

- : 이미지 디코딩을 위한 모델
- LLaMA는 사전 훈련된 CLIP Vision Encoder를 사용하여 이미지의 잠재 표현을 생성한 다음, 이를 사용하여 이미지를 생성하는 디코더를 구축함
- 논문에서는 Language Decoder로 사용함





TOBIGS

3. Method

MultiModal-GPT: A Vision and Language Model for Dialogue with Humans, 2023, Tao Gong et al.

3.1 Architecture

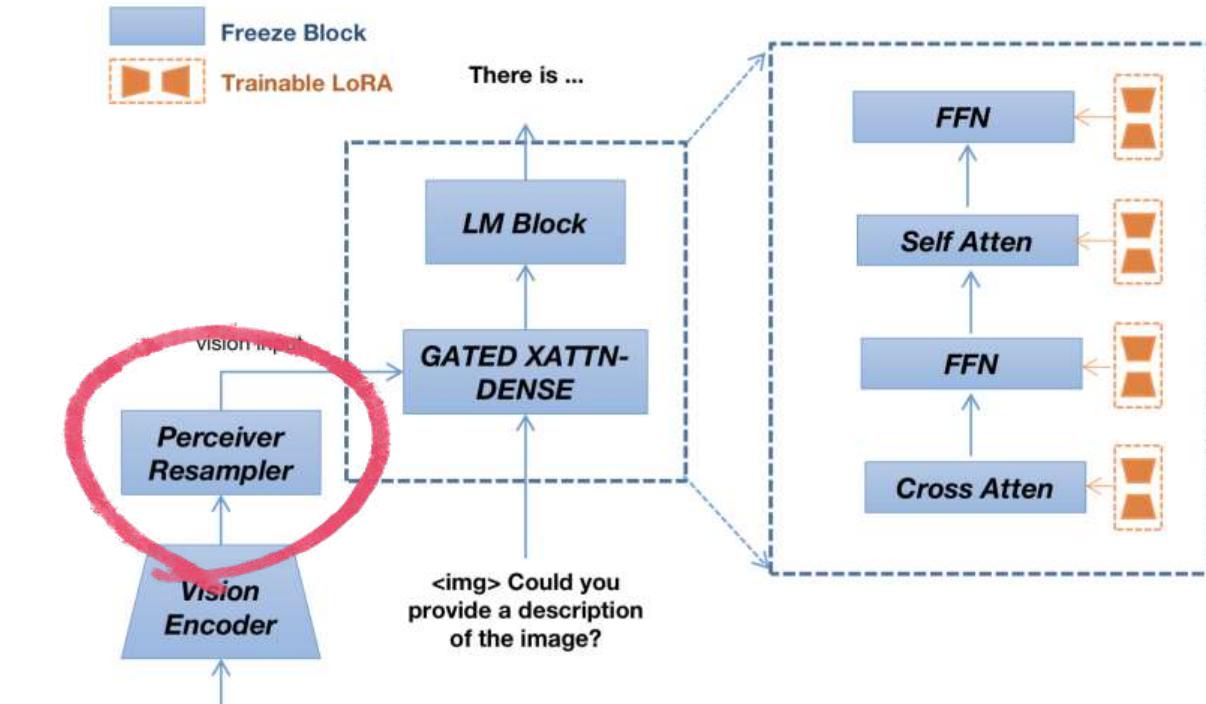
Perceiver Resampler

: 이미지 정보를 처리하는 Module

- Perceiver Resampler를 이용해 추출된 공간적인 특징들은 Language Decoder에 전달됨
- Language Decoder는 이런 공간적인 특징과 Attention Mechanism을 이용하여 이미지의 특징을 텍스트로 Encoding함

• Attention Mechanism이란?

: 이미지와 텍스트 간의 상호작용을 통해 적절한 표현을 생성하는 메커니즘





TOBIGS

3. Method

MultiModal-GPT: A Vision and Language Model for Dialogue with Humans, 2023, Tao Gong et al.

3.1 Architecture

Gated XATTN-DENSE

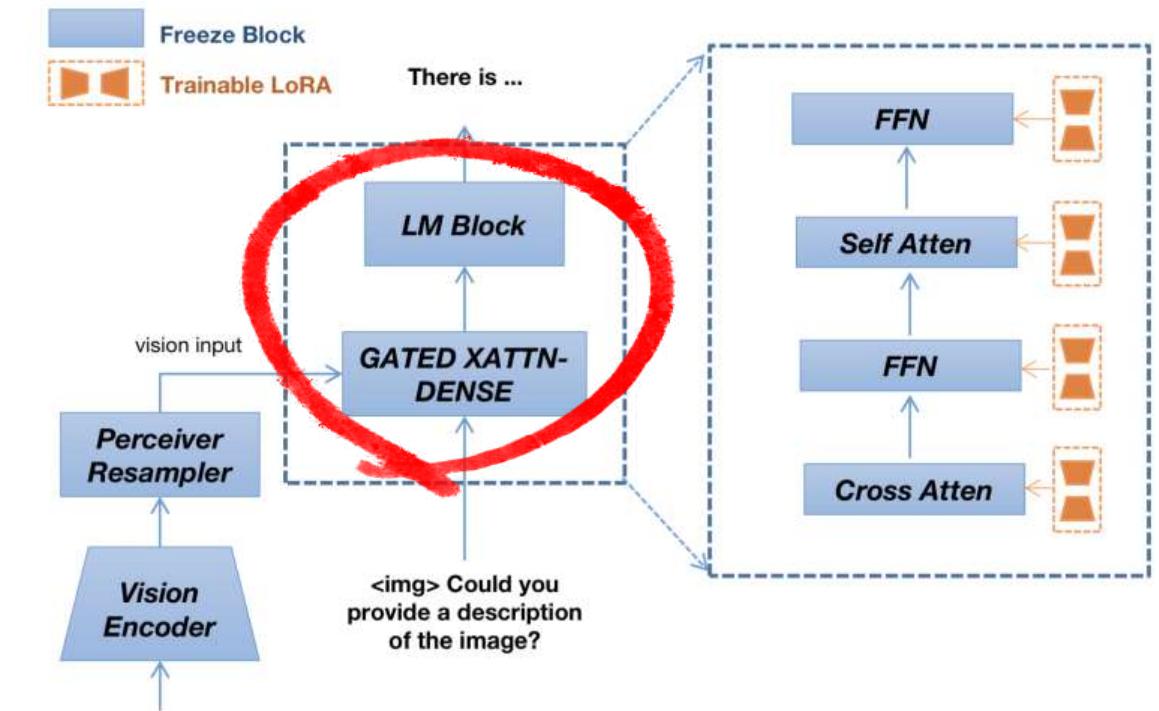
: visually-conditioning을 해줄 Transformer decoder 구조의 layer

- cross-attention과 FFW로 구성됨

LM Block

: text generation을 위한 layer

- pre-trained 되어있고, frozen되어 있음





TOBIGS

3. Method

MultiModal-GPT: A Vision and Language Model for Dialogue with Humans, 2023, Tao Gong et al.

3.1 Architecture

Cross Attention

: key, value로 같은 값을 사용하지만 query는 다른 값을 사용하는 어텐션 연산

- Cross-attention의 output을 입력받음

Self Attention

: query, key, value로 모두 같은 값을 사용하는 어텐션 FFN

- Self-attention의 output을 입력받음

- **FFN**이란?

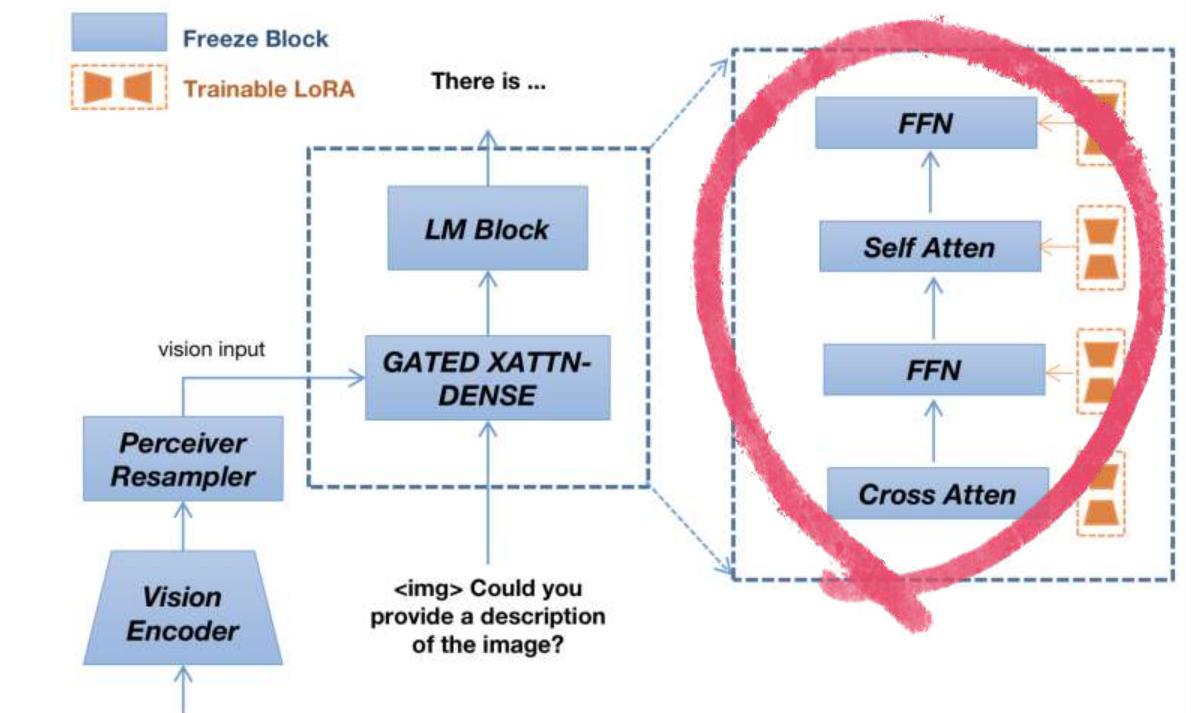
: Feed Forward Neural Network

BASE MODEL

MAIN PAPER

MAIN 논문에 대한 이해

CODE SCRATCH





TOBIGS

3. Method

MultiModal-GPT: A Vision and Language Model for Dialogue with Humans, 2023, Tao Gong et al.

3.2 Joint Training

: 새로운 데이터를 추가하여 처음부터 다시 시작

- **language-only instruction-following data + vision and language instruction-following data = Joint Training**

Joint Training

- freeze the whole open-flamingo
- add LoRA to the self-attention, cross-attention, and FFW part in the language

decoder to finetune MultiModal-GPT

- the {response}, <EOS> tokens만 loss 값 계산함

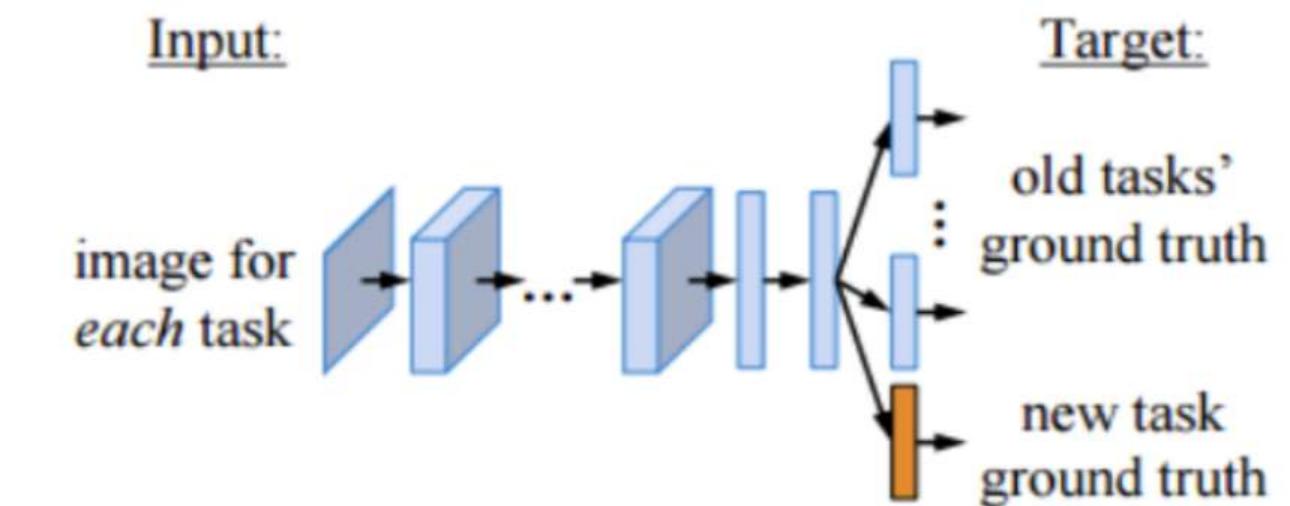
BASE MODEL

MAIN PAPER

MAIN 논문에 대한 이해

CODE SCRATCH

(d) Joint Training





TOBIGS

4. Experiments

MultiModal-GPT: A Vision and Language Model for Dialogue with Humans, 2023, Tao Gong et al.

4.1 Implementation Details

- 데이터셋 : mix of language data and vision and language data sources
 - diverse and rich training environment for the MultiModal-GPT model
 - 성능을 높이기위해 사용함
- 학습 과정
 - Language : incorporate the entire text corpus from language datasets
 - Vision : include all image-text pairs from image datasets
 - 문제점 : language dataset에 비해 image dataset의 quality가 떨어짐
 - 해결 방법 : include a random sample of 5000 image-text pairs from the A-OKVQA dataset and 512 image-text pairs each from the COCO Caption and OCR VQA datasets in the training process

BASE MODEL

MAIN PAPER

MAIN 논문에 대한 이해

CODE SCRATCH



TOBIGS

4. Experiments

MultiModal-GPT: A Vision and Language Model for Dialogue with Humans, 2023, Tao Gong et al.

4.2 The Quality of Data Matters

실험을 진행하면서 성능 향상을 위해 **데이터셋 몇개를 제외하였음**

BASE MODEL

- VQA v2.0, OKVQA, GQA, CLEVR, NLVR 데이터셋을 뺐음

MAIN PAPER

MAIN 논문에 대한 이해

- **데이터셋을 제외하게된 이유는?**

: 이 데이터셋들을 학습시키면 모델이 짧은 대답만 내놓았으므로, 좀 더 사용자에 친숙한 대화 형태

를 만들기 위해 최종 모델에 이 데이터셋들을 제외하기로 결정했음

CODE SCRATCH



TOBIGS

4. Experiments

MultiModal-GPT: A Vision and Language Model for Dialogue with Humans, 2023, Tao Gong et al.

4.3 Demos

BASE MODEL

MAIN PAPER

MAIN 논문에 대한 이해

CODE SCRATCH

How to make this dish?



To make this dish, you will need to prepare the ingredients and follow the recipe. First, you will need to prepare the pasta sauce by combining tomato sauce, garlic, basil, and parmesan cheese. Next, you will need to assemble the lasagna by layering the noodles, sauce, and mozzarella cheese. Finally, you will need to bake the lasagna in the oven until it is fully cooked and the cheese is melted.

If I can not cook, where can I eat it?

You can order this dish from a restaurant or have it catered. Alternatively, you can order the ingredients online and follow the recipe to prepare it at home.

Figure 2: The MultiModal-GPT can give a recipe to bake lasagna, and tell users where to eat it.

Who is Elon Musk?

Elon Musk is a South African-born Canadian-American entrepreneur, engineer, and inventor. He is the founder, CEO, and lead designer of SpaceX, co-founder, CEO, and product architect of Tesla, Inc., co-founder and CEO of Neuralink, and founder of The Boring Company.

Is this a photo of him?



Yes, this is a photo of Elon Musk.

How many companies does he have?

Elon Musk has three companies: SpaceX, Tesla, and The Boring Company.

Figure 3: The MultiModal-GPT can recognize the image of Elon Musk, and answer questions about Elon Musk.



TOBIGS

4. Experiments

MultiModal-GPT: A Vision and Language Model for Dialogue with Humans, 2023, Tao Gong et al.

4.3 Demos

BASE MODEL

MAIN PAPER

MAIN 논문에 대한 이해

CODE SCRATCH

How many people in this image?



There are four people in this image.

What is their gender

The people in the image are all women.

What are they doing

The women are wearing sunglasses and posing for a photo.

Do you like this?



Yes, I like this image because it features the movie poster for the animated film WALL-E. The movie is about a robot named WALL-E, who spends most of his life cleaning up the planet Earth after humans abandoned it due to pollution. WALL-E falls in love with another robot named EVE, and they embark on an adventure to find a new home for humanity. The image captures the essence of the story, with WALL-E and EVE featured prominently on the poster. It is a visually appealing and engaging poster for the movie.

Which studio made this film?

Pixar.

Figure 4: The MultiModal-GPT knows there are 4 women in the image, and knows what they are doing.

Figure 5: The MultiModal-GPT knows the film and knows which studio made the film.



TOBIGS

4. Experiments

MultiModal-GPT: A Vision and Language Model for Dialogue with Humans, 2023, Tao Gong et al.

4.3 Demos

BASE MODEL

MAIN PAPER

MAIN 논문에 대한 이해

CODE SCRATCH

The screenshot displays two separate demo sessions for the MultiModal-GPT model. On the left, under 'BASE MODEL', a user asks 'what is the title of this film' and receives an image of the 'Harry Potter COMPLETE 8 FILM COLLECTION' movie poster. Below it, another user asks 'what is the text in this image' and receives the response 'harry potter'. On the right, under 'MAIN PAPER', a user asks 'What can I do if I travel there?' and receives a scenic beach image of Kauai. Below it, the user asks 'How much will it cost?' and receives a detailed response about the costs of accommodation, transportation, and activities. At the bottom, the user asks 'What else should I prepare for this trip?' and receives advice on preparing for the weather and activities.

what is the title of this film

harry potter

what is the text in this image

harry potter

How many people in this image

Three

Who is the author of the original book of this movie?

j.k. rowling

What can I do if I travel there?

If you travel to the Hawaiian island of Kauai, you can enjoy a variety of activities such as swimming, snorkeling, surfing, hiking, and sightseeing. You can explore the island's natural beauty by visiting the Waimea Canyon, also known as the "Grand Canyon of the Pacific", and the Na Pali Coast, which boasts breathtaking cliffs and waterfalls. You can also explore the island's cultural heritage by visiting the National Tropical Botanical Garden and the Kauai Museum. Additionally, the island is home to several resorts and hotels, where you can stay and enjoy amenities such as swimming pools, spas, and restaurants.

How much will it cost?

The cost of traveling to Kauai can vary depending on your choice of accommodation, transportation, and activities. You can find affordable accommodation options, such as hostels and Airbnb rentals, as well as more luxurious options, such as hotels and resorts. Transportation costs can vary depending on your mode of transportation. For example, if you choose to rent a car, you will have to pay for gas and parking fees. If you prefer to use public transportation, you will have to pay for bus tickets. The cost of activities, such as swimming, snorkeling, surfing, hiking, and sightseeing, can also vary depending on your preferences and the availability of equipment.

What else should I prepare for this trip?

To prepare for your trip to Kauai, you should research the island's attractions, activities, and transportation options. You should also check the weather forecast to ensure that you are prepared for any potential weather conditions. Additionally, you may want to bring sunscreen, a swimsuit, and waterproof clothing, depending on your planned activities and the weather.

Figure 6: The MultiModal-GPT can recognize the characters in the image and knows the author of the book.

Figure 7: The MultiModal-GPT can answer general questions about the traveling of users.



TOBIGS

4. Experiments

MultiModal-GPT: A Vision and Language Model for Dialogue with Humans, 2023, Tao Gong et al.

4.3 Demos

BASE MODEL

MAIN PAPER

MAIN 논문에 대한 이해

CODE SCRATCH

Please give a detailed description of this image



The image shows a beautiful lake surrounded by lush green grass and trees. The lake is filled with water lilies, creating a tranquil and serene atmosphere.

The lake is situated in a mountainous landscape, with a mountain peak visible in the background.

There are several birds flying around the lake, adding to the tranquility and natural beauty of the scene.

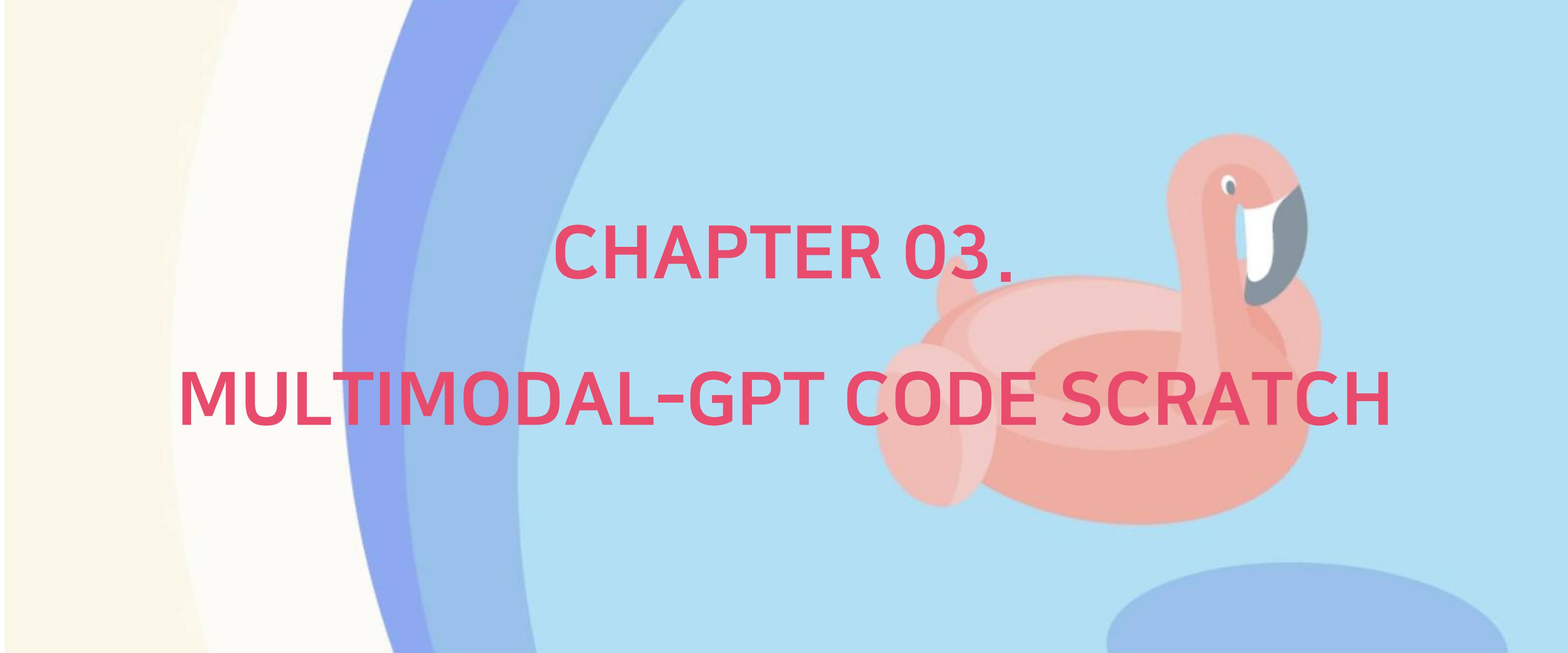
What is the season in the picture

The season in the image is summer, as evidenced by the lush green grass and the presence of water lilies in the lake.

If I want to travel to some places like in this image, what is your suggestion

If you want to experience a tranquil and serene environment similar to the one in the image, you could visit a natural lake surrounded by lush greenery and mountains.

Figure 8: The MultiModal-GPT can generate a detailed description for the image and has the ability to reason the season in the image.



CHAPTER 03.

MULTIMODAL-GPT CODE SCRATCH



TOBIGS

1. Directory Layout

<https://github.com/open-mmlab/Multimodal-GPT/tree/main>

BASE MODEL
MAIN PAPER
CODE SCRATCH
논문 속 CODE SCRATCH

```
├── checkpoints
├── configs
│   ├── dataset_config.py
│   └── lora_config.py
└── mmgpt
    ├── datasets
    ├── samplers
    │   ├── infinite_sampler.py
    │   └── *_dataset.py
    ├── ...
    ├── models
    ├── blip2
    ├── open_flamingo
    │   ├── builder.py
    │   ├── flamingo.py
    │   ├── flamingo_lm.py
    │   ├── helpers.py
    │   ├── utils.py
    │   └── builder.py
    └── train
        ├── distributed.py
        ├── instruction_finetune.py
        └── train_utils.py
```

모델 checkpoint 파일 보관
configuration 파일 보관
Dataset class
model 코드
학습 코드

The screenshot shows a GitHub repository interface for the 'main' branch. The repository has 2 branches and 0 tags. The commit history is as follows:

Commit	Date
GT9505 Fix training bug ...	7624425 2 weeks ago
checkpoints	init
configs	fix training bug
docs/images	init
mmgpt	fix
.gitignore	fix training bug
LICENSE	init
README.md	Update README.md (#19)
README_zh-CN.md	tech-report cn (#16)
app.py	beautiful memory show
environment.yml	init
requirements.txt	Update requirements.txt (#3)
setup.py	init



TOBIGS

2. 학습 코드

mmgpt/train/instruction_finetune.py

BASE MODEL

```
159     model, image_processor, tokenizer = create_model_and_transforms() 모델 불러오기
160     model_name="open_flamingo",
161     clip_vision_encoder_path=args.vision_encoder_path,
162     clip_vision_encoder_pretrained=args.vision_encoder_pretrained,
163     lang_encoder_path=args.lm_path,
164     tokenizer_path=args.tokenizer_path if args.tokenizer_path else args.lm_path,
165     use_media_placement_augmentation=args.use_media_placement_augmentation,
166     pretrained_model_path=args.pretrained_path,
167     tuning_config=tuning_config.tuning_config,
168 )
169
170     if args.dataset_config is not None:
171         dataset_config = Config.fromfile(args.dataset_config)
172     else:
173         raise ValueError("dataset_config must be specified")
174
```

CODE SCRATCH

논문 속 CODE SCRATCH

```
175     dataset = build_dataset(
176         dataset_config=dataset_config.visual_datasets,
177         vis_processor=image_processor,
178         tokenizer=tokenizer,
179     )
180     train_dataloader = DataLoader(
181         dataset,
182         batch_size=args.batch_size,
183         num_workers=args.workers,
184         sampler=DistributedSampler(dataset, shuffle=True, drop_last=True),
185         collate_fn=dataset.collater,
186     )
187
188     # build language dataset and dataloader for multi-modality training
189     if dataset_config.get('language_datasets') is not None and len(dataset_config.language_datasets) > 0:
190         lang_dataset = build_dataset(
191             dataset_config=dataset_config.language_datasets,
192             tokenizer=tokenizer,
193         )
194         lang_dataloader = DataLoader(
195             lang_dataset,
196             batch_size=args.batch_size,
197             num_workers=args.workers,
198             sampler=InfiniteSampler(lang_dataset, shuffle=True),
199             collate_fn=lang_dataset.collater,
200         )
201         lang_dataloader = iter(lang_dataloader)
202     else:
203         lang_dataloader = None
```

vision dataset
불러오기

language dataset
불러오기



TOBIGS

2. 학습 코드

mmgpt/train/instruction_finetune.py

BASE MODEL

```

286     ddp_model.train()
287
288     for epoch in range(resume_from_epoch, args.num_epochs):
289         train_dataloader.sampler.set_epoch(epoch)
290
291         train_one_epoch( 모델 학
292                         args=args,
293                         model=ddp_model,
294                         epoch=epoch,
295                         tokenizer=tokenizer,
296                         optimizer=optimizer,
297                         lr_scheduler=lr_scheduler,
298                         train_dataloader=train_dataloader,
299                         language_dataloader=lang_dataloader,
300                         device_id=device_id,
301                         wandb=wandb,
302                     )
303

```

CODE SCRATCH

논문 속 CODE SCRATCH

```

305     if args.rank == 0:
306         if not os.path.exists(args.run_name):
307             os.makedirs(args.run_name)
308
309         checkpoint_dict = {
310             "epoch": epoch,
311             "model_state_dict": get_checkpoint(ddp_model),
312             "optimizer_state_dict": optimizer.state_dict(),
313             "lr_scheduler_state_dict": lr_scheduler.state_dict(),
314             "tuning_config": tuning_config,
315         }
316
317         print(f"Saving checkpoint to {args.run_name}/checkpoint_{epoch}.pt")
318         torch.save(checkpoint_dict, f"{args.run_name}/checkpoint_{epoch}.pt")
319         if args.report_to_wandb and args.save_checkpoints_to_wandb:
320             wandb.save(f"{args.run_name}/checkpoint_{epoch}.pt")
321
322         if args.delete_previous_checkpoint:
323             if epoch > 0:
324                 os.remove(f"{args.run_name}/checkpoint_{epoch-1}.pt")
325
326         if args.rank == 0:
327             torch.save(
328                 {"model_state_dict": get_checkpoint(ddp_model.module), "tuning_config": tuning_config},
329                 f"{args.run_name}/final_weights.pt",
330             )
331         if args.report_to_wandb and args.save_checkpoints_to_wandb:
332             wandb.save(f"{args.run_name}/final_weights.pt")

```

학습 log 및
checkpoint 저장



TOBIGS

2. 학습 코드

mmgpt/models/open_flamingo/builder.py

BASE MODEL

MAIN PAPER

CODE SCRATCH

논문 속 CODE SCRATCH

```

14 def create_model_and_transforms(
15     clip_vision_encoder_path: str,
16     clip_vision_encoder_pretrained: str,
17     lang_encoder_path: str,
18     tokenizer_path: str,
19     decoder_layers_attr_name: str = None,
20     pretrained_model_path: str = None,
21     vision_encoder, _, image_processor = open_clip.create_model_and_transforms(      vision encoder, image 전처리기 가져오기
22         clip_vision_encoder_path, pretrained=clip_vision_encoder_pretrained          (from CLIP)
23     )
24     # set the vision encoder to output the visual features
25     vision_encoder.visual.output_tokens = True
26     print("init tokenizer")
27     text_tokenizer = LlamaTokenizer.from_pretrained(tokenizer_path)
28     # add Flamingo special tokens to the tokenizer
29     text_tokenizer.add_special_tokens({"additional_special_tokens": ["<|endofchunk|>", "<image>"]})
30     if text_tokenizer.pad_token is None:
31         # Issue: GPT models don't have a pad token, which we use to
32         # modify labels for the loss.
33         text_tokenizer.add_special_tokens({"pad_token": "<PAD>"})
34     text_tokenizer.bos_token_id = 1
35     text_tokenizer.eos_token_id = 2
36     print("init llama")
37     lang_encoder = LlamaForCausalLM.from_pretrained(lang_encoder_path)
38     extend_instance(lang_encoder, FlamingoLMMixin)
39     if decoder_layers_attr_name is None:
40         decoder_layers_attr_name = _infer_decoder_layers_attr_name(lang_encoder)
41     lang_encoder.set_decoder_layers_attr_name(decoder_layers_attr_name)
42     lang_encoder.resize_token_embeddings(len(text_tokenizer))
43

```

Tokenizer, Language model 가져오기
(from Llama)

vision encoder, image 전처리기 가져오기
(from CLIP)

```

65 model = Flamingo(
66     vision_encoder,
67     lang_encoder,
68     text_tokenizer.encode("<|endofchunk|>)[-1],
69     text_tokenizer.encode("<image>)[-1],
70     vis_dim=open_clip.get_model_config(clip_vision_encoder_path)["vision_cfg"]["width"],
71     cross_attn_every_n_layers=4,
72     **flamingo_kwargs,
73 )
74
75 if pretrained_model_path is not None:
76     print(f"loading pretrained model from {pretrained_model_path}")
77     model.load_state_dict(torch.load(pretrained_model_path), strict=False)
78
79 # Freeze all parameters
80 model.requires_grad_(False)
81 assert sum(p.numel() for p in model.parameters() if p.requires_grad) == 0
82
83 if tuning_config is not None:
84     model = prepare_model_for_tuning(model, tuning_config)    LoRA 적용
85 else:
86     raise ValueError("tuning_config must be provided")
87
88 return model, image_processor, text_tokenizer

```



TOBIGS

2. 학습 코드

mmgpt/train/instruction_finetune.py

BASE MODEL

MAIN PAPER

CODE SCRATCH

논문 속 CODE SCRATCH

```

334 def train_one_epoch(
335     # setup logging
336     step_time_m = AverageMeter() # time for one optimizer step (> 1 batch if using gradient accum)
337     data_time_m = (
338         AverageMeter()
339     ) # avg time to load one batch of both C4 AND laion (= 1 batch regardless of gradient accum)
340     end = time.time()
341
342     # loop through dataloader
343     for num_steps, batch in tqdm(
344         enumerate(train_dataloader),
345         disable=args.rank != 0,
346         total=total_training_steps,
347         initial=(epoch * num_batches_per_epoch),
348     ):
349         data_time_m.update(time.time() - end)
350
351         global_step = num_steps + epoch * num_batches_per_epoch
352
353         ##### VISION FORWARD PASS ##### vision&language instruction 학습
354         images = batch["image"].to(device_id, dtype=cast_dtype, non_blocking=True).unsqueeze(1).unsqueeze(1)
355         input_ids = batch["input_ids"].to(device_id, dtype=cast_dtype, non_blocking=True)
356         attention_mask = batch["attention_mask"].to(device_id, dtype=cast_dtype, non_blocking=True)
357         labels = batch["labels"].to(device_id, dtype=cast_dtype, non_blocking=True)
358
359         with autocast():
360             loss_batch = model(
361                 vision_x=images,
362                 lang_x=input_ids,
363                 attention_mask=attention_mask,
364                 labels=labels,
365             )[0]
366             loss = loss_batch / args.gradient_accumulation_steps
367             loss_vision = loss # for logging
368
369             ##### BACKWARD PASS #####
370             loss.backward()
371
372             data_time_m.update(time.time() - end)
373
374             global_step = num_steps + epoch * num_batches_per_epoch
375
376             ##### LANGUAGE FORWARD PASS ##### language only instruction 학습
377             if language_dataloader is not None: language_only_instruction 학습
378             batch_lang = next(language_dataloader)
379             lang_input_ids = batch_lang["input_ids"].to(device_id, dtype=cast_dtype, non_blocking=True)
380             lang_attention_mask = batch_lang["attention_mask"].to(device_id, dtype=cast_dtype, non_blocking=True)
381             lang_labels = batch_lang["labels"].to(device_id, dtype=cast_dtype, non_blocking=True)
382
383             with autocast():
384                 lang_loss_batch = model(
385                     vision_x=None,
386                     lang_x=lang_input_ids,
387                     attention_mask=lang_attention_mask,
388                     labels=lang_labels,
389                 )[0]
390                 lang_loss = lang_loss_batch / args.gradient_accumulation_steps
391
392                 ##### BACKWARD PASS #####
393                 lang_loss.backward()
394
395                 torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
396
397                 # step optimizer and log
398                 if (((num_steps + 1) % args.gradient_accumulation_steps) == 0) or (num_steps == num_batches_per_epoch - 1):
399                     optimizer.step()
400                     lr_scheduler.step()
401                     optimizer.zero_grad()
402
403             data_time_m.update(time.time() - end)
404
405             global_step = num_steps + epoch * num_batches_per_epoch
406
407             ##### BACKWARD PASS #####
408             loss.backward()
409
410             data_time_m.update(time.time() - end)
411
412             global_step = num_steps + epoch * num_batches_per_epoch
413
414             ##### BACKWARD PASS #####
415             loss.backward()
416
417             data_time_m.update(time.time() - end)
418
419             global_step = num_steps + epoch * num_batches_per_epoch
420
421             ##### BACKWARD PASS #####
422             loss.backward()
423
424             data_time_m.update(time.time() - end)
425
426             global_step = num_steps + epoch * num_batches_per_epoch
427
428             ##### BACKWARD PASS #####
429             loss.backward()
430
431             data_time_m.update(time.time() - end)
432
433             global_step = num_steps + epoch * num_batches_per_epoch
434
435             ##### BACKWARD PASS #####
436             loss.backward()
437
438             data_time_m.update(time.time() - end)
439
440             global_step = num_steps + epoch * num_batches_per_epoch
441
442             ##### BACKWARD PASS #####
443             loss.backward()
444
445             data_time_m.update(time.time() - end)
446
447             global_step = num_steps + epoch * num_batches_per_epoch
448
449             ##### BACKWARD PASS #####
450             loss.backward()
451
452             data_time_m.update(time.time() - end)
453
454             global_step = num_steps + epoch * num_batches_per_epoch
455
456             ##### BACKWARD PASS #####
457             loss.backward()
458
459             data_time_m.update(time.time() - end)
460
461             global_step = num_steps + epoch * num_batches_per_epoch
462
463             ##### BACKWARD PASS #####
464             loss.backward()
465
466             data_time_m.update(time.time() - end)
467
468             global_step = num_steps + epoch * num_batches_per_epoch
469
470             ##### BACKWARD PASS #####
471             loss.backward()
472
473             data_time_m.update(time.time() - end)
474
475             global_step = num_steps + epoch * num_batches_per_epoch
476
477             ##### BACKWARD PASS #####
478             loss.backward()
479
480             data_time_m.update(time.time() - end)
481
482             global_step = num_steps + epoch * num_batches_per_epoch
483
484             ##### BACKWARD PASS #####
485             loss.backward()
486
487             data_time_m.update(time.time() - end)
488
489             global_step = num_steps + epoch * num_batches_per_epoch
490
491             ##### BACKWARD PASS #####
492             loss.backward()
493
494             data_time_m.update(time.time() - end)
495
496             global_step = num_steps + epoch * num_batches_per_epoch
497
498             ##### BACKWARD PASS #####
499             loss.backward()
500
501             data_time_m.update(time.time() - end)
502
503             global_step = num_steps + epoch * num_batches_per_epoch
504
505             ##### BACKWARD PASS #####
506             loss.backward()
507
508             data_time_m.update(time.time() - end)
509
510             global_step = num_steps + epoch * num_batches_per_epoch
511
512             ##### BACKWARD PASS #####
513             loss.backward()
514
515             data_time_m.update(time.time() - end)
516
517             global_step = num_steps + epoch * num_batches_per_epoch
518
519             ##### BACKWARD PASS #####
520             loss.backward()
521
522             data_time_m.update(time.time() - end)
523
524             global_step = num_steps + epoch * num_batches_per_epoch
525
526             ##### BACKWARD PASS #####
527             loss.backward()
528
529             data_time_m.update(time.time() - end)
530
531             global_step = num_steps + epoch * num_batches_per_epoch
532
533             ##### BACKWARD PASS #####
534             loss.backward()
535
536             data_time_m.update(time.time() - end)
537
538             global_step = num_steps + epoch * num_batches_per_epoch
539
540             ##### BACKWARD PASS #####
541             loss.backward()
542
543             data_time_m.update(time.time() - end)
544
545             global_step = num_steps + epoch * num_batches_per_epoch
546
547             ##### BACKWARD PASS #####
548             loss.backward()
549
550             data_time_m.update(time.time() - end)
551
552             global_step = num_steps + epoch * num_batches_per_epoch
553
554             ##### BACKWARD PASS #####
555             loss.backward()
556
557             data_time_m.update(time.time() - end)
558
559             global_step = num_steps + epoch * num_batches_per_epoch
560
561             ##### BACKWARD PASS #####
562             loss.backward()
563
564             data_time_m.update(time.time() - end)
565
566             global_step = num_steps + epoch * num_batches_per_epoch
567
568             ##### BACKWARD PASS #####
569             loss.backward()
570
571             data_time_m.update(time.time() - end)
572
573             global_step = num_steps + epoch * num_batches_per_epoch
574
575             ##### BACKWARD PASS #####
576             loss.backward()
577
578             data_time_m.update(time.time() - end)
579
580             global_step = num_steps + epoch * num_batches_per_epoch
581
582             ##### BACKWARD PASS #####
583             loss.backward()
584
585             data_time_m.update(time.time() - end)
586
587             global_step = num_steps + epoch * num_batches_per_epoch
588
589             ##### BACKWARD PASS #####
590             loss.backward()
591
592             data_time_m.update(time.time() - end)
593
594             global_step = num_steps + epoch * num_batches_per_epoch
595
596             ##### BACKWARD PASS #####
597             loss.backward()
598
599             data_time_m.update(time.time() - end)
600
601             global_step = num_steps + epoch * num_batches_per_epoch
602
603             ##### BACKWARD PASS #####
604             loss.backward()
605
606             data_time_m.update(time.time() - end)
607
608             global_step = num_steps + epoch * num_batches_per_epoch
609
610             ##### BACKWARD PASS #####
611             loss.backward()
612
613             data_time_m.update(time.time() - end)
614
615             global_step = num_steps + epoch * num_batches_per_epoch
616
617             ##### BACKWARD PASS #####
618             loss.backward()
619
620             data_time_m.update(time.time() - end)
621
622             global_step = num_steps + epoch * num_batches_per_epoch
623
624             ##### BACKWARD PASS #####
625             loss.backward()
626
627             data_time_m.update(time.time() - end)
628
629             global_step = num_steps + epoch * num_batches_per_epoch
630
631             ##### BACKWARD PASS #####
632             loss.backward()
633
634             data_time_m.update(time.time() - end)
635
636             global_step = num_steps + epoch * num_batches_per_epoch
637
638             ##### BACKWARD PASS #####
639             loss.backward()
640
641             data_time_m.update(time.time() - end)
642
643             global_step = num_steps + epoch * num_batches_per_epoch
644
645             ##### BACKWARD PASS #####
646             loss.backward()
647
648             data_time_m.update(time.time() - end)
649
650             global_step = num_steps + epoch * num_batches_per_epoch
651
652             ##### BACKWARD PASS #####
653             loss.backward()
654
655             data_time_m.update(time.time() - end)
656
657             global_step = num_steps + epoch * num_batches_per_epoch
658
659             ##### BACKWARD PASS #####
660             loss.backward()
661
662             data_time_m.update(time.time() - end)
663
664             global_step = num_steps + epoch * num_batches_per_epoch
665
666             ##### BACKWARD PASS #####
667             loss.backward()
668
669             data_time_m.update(time.time() - end)
670
671             global_step = num_steps + epoch * num_batches_per_epoch
672
673             ##### BACKWARD PASS #####
674             loss.backward()
675
676             data_time_m.update(time.time() - end)
677
678             global_step = num_steps + epoch * num_batches_per_epoch
679
680             ##### BACKWARD PASS #####
681             loss.backward()
682
683             data_time_m.update(time.time() - end)
684
685             global_step = num_steps + epoch * num_batches_per_epoch
686
687             ##### BACKWARD PASS #####
688             loss.backward()
689
690             data_time_m.update(time.time() - end)
691
692             global_step = num_steps + epoch * num_batches_per_epoch
693
694             ##### BACKWARD PASS #####
695             loss.backward()
696
697             data_time_m.update(time.time() - end)
698
699             global_step = num_steps + epoch * num_batches_per_epoch
700
701             ##### BACKWARD PASS #####
702             loss.backward()
703
704             data_time_m.update(time.time() - end)
705
706             global_step = num_steps + epoch * num_batches_per_epoch
707
708             ##### BACKWARD PASS #####
709             loss.backward()
710
711             data_time_m.update(time.time() - end)
712
713             global_step = num_steps + epoch * num_batches_per_epoch
714
715             ##### BACKWARD PASS #####
716             loss.backward()
717
718             data_time_m.update(time.time() - end)
719
720             global_step = num_steps + epoch * num_batches_per_epoch
721
722             ##### BACKWARD PASS #####
723             loss.backward()
724
725             data_time_m.update(time.time() - end)
726
727             global_step = num_steps + epoch * num_batches_per_epoch
728
729             ##### BACKWARD PASS #####
730             loss.backward()
731
732             data_time_m.update(time.time() - end)
733
734             global_step = num_steps + epoch * num_batches_per_epoch
735
736             ##### BACKWARD PASS #####
737             loss.backward()
738
739             data_time_m.update(time.time() - end)
740
741             global_step = num_steps + epoch * num_batches_per_epoch
742
743             ##### BACKWARD PASS #####
744             loss.backward()
745
746             data_time_m.update(time.time() - end)
747
748             global_step = num_steps + epoch * num_batches_per_epoch
749
750             ##### BACKWARD PASS #####
751             loss.backward()
752
753             data_time_m.update(time.time() - end)
754
755             global_step = num_steps + epoch * num_batches_per_epoch
756
757             ##### BACKWARD PASS #####
758             loss.backward()
759
760             data_time_m.update(time.time() - end)
761
762             global_step = num_steps + epoch * num_batches_per_epoch
763
764             ##### BACKWARD PASS #####
765             loss.backward()
766
767             data_time_m.update(time.time() - end)
768
769             global_step = num_steps + epoch * num_batches_per_epoch
770
771             ##### BACKWARD PASS #####
772             loss.backward()
773
774             data_time_m.update(time.time() - end)
775
776             global_step = num_steps + epoch * num_batches_per_epoch
777
778             ##### BACKWARD PASS #####
779             loss.backward()
780
781             data_time_m.update(time.time() - end)
782
783             global_step = num_steps + epoch * num_batches_per_epoch
784
785             ##### BACKWARD PASS #####
786             loss.backward()
787
788             data_time_m.update(time.time() - end)
789
790             global_step = num_steps + epoch * num_batches_per_epoch
791
792             ##### BACKWARD PASS #####
793             loss.backward()
794
795             data_time_m.update(time.time() - end)
796
797             global_step = num_steps + epoch * num_batches_per_epoch
798
799             ##### BACKWARD PASS #####
800             loss.backward()
801
802             data_time_m.update(time.time() - end)
803
804             global_step = num_steps + epoch * num_batches_per_epoch
805
806             ##### BACKWARD PASS #####
807             loss.backward()
808
809             data_time_m.update(time.time() - end)
810
811             global_step = num_steps + epoch * num_batches_per_epoch
812
813             ##### BACKWARD PASS #####
814             loss.backward()
815
816             data_time_m.update(time.time() - end)
817
818             global_step = num_steps + epoch * num_batches_per_epoch
819
820             ##### BACKWARD PASS #####
821             loss.backward()
822
823             data_time_m.update(time.time() - end)
824
825             global_step = num_steps + epoch * num_batches_per_epoch
826
827             ##### BACKWARD PASS #####
828             loss.backward()
829
830             data_time_m.update(time.time() - end)
831
832             global_step = num_steps + epoch * num_batches_per_epoch
833
834             ##### BACKWARD PASS #####
835             loss.backward()
836
837             data_time_m.update(time.time() - end)
838
839             global_step = num_steps + epoch * num_batches_per_epoch
840
841             ##### BACKWARD PASS #####
842             loss.backward()
843
844             data_time_m.update(time.time() - end)
845
846             global_step = num_steps + epoch * num_batches_per_epoch
847
848             ##### BACKWARD PASS #####
849             loss.backward()
850
851             data_time_m.update(time.time() - end)
852
853             global_step = num_steps + epoch * num_batches_per_epoch
854
855             ##### BACKWARD PASS #####
856             loss.backward()
857
858             data_time_m.update(time.time() - end)
859
860             global_step = num_steps + epoch * num_batches_per_epoch
861
862             ##### BACKWARD PASS #####
863             loss.backward()
864
865             data_time_m.update(time.time() - end)
866
867             global_step = num_steps + epoch * num_batches_per_epoch
868
869             ##### BACKWARD PASS #####
870             loss.backward()
871
872             data_time_m.update(time.time() - end)
873
874             global_step = num_steps + epoch * num_batches_per_epoch
875
876             ##### BACKWARD PASS #####
877             loss.backward()
878
879             data_time_m.update(time.time() - end)
880
881             global_step = num_steps + epoch * num_batches_per_epoch
882
883             ##### BACKWARD PASS #####
884             loss.backward()
885
886             data_time_m.update(time.time() - end)
887
888             global_step = num_steps + epoch * num_batches_per_epoch
889
890             ##### BACKWARD PASS #####
891             loss.backward()
892
893             data_time_m.update(time.time() - end)
894
895             global_step = num_steps + epoch * num_batches_per_epoch
896
897             ##### BACKWARD PASS #####
898             loss.backward()
899
900             data_time_m.update(time.time() - end)
901
902             global_step = num_steps + epoch * num_batches_per_epoch
903
904             ##### BACKWARD PASS #####
905             loss.backward()
906
907             data_time_m.update(time.time() - end)
908
909             global_step = num_steps + epoch * num_batches_per_epoch
910
911             ##### BACKWARD PASS #####
912             loss.backward()
913
914             data_time_m.update(time.time() - end)
915
916             global_step = num_steps + epoch * num_batches_per_epoch
917
918             ##### BACKWARD PASS #####
919             loss.backward()
920
921             data_time_m.update(time.time() - end)
922
923             global_step = num_steps + epoch * num_batches_per_epoch
924
925             ##### BACKWARD PASS #####
926             loss.backward()
927
928             data_time_m.update(time.time() - end)
929
930             global_step = num_steps + epoch * num_batches_per_epoch
931
932             ##### BACKWARD PASS #####
933             loss.backward()
934
935             data_time_m.update(time.time() - end)
936
937             global_step = num_steps + epoch * num_batches_per_epoch
938
939             ##### BACKWARD PASS #####
940             loss.backward()
941
942             data_time_m.update(time.time() - end)
943
944             global_step = num_steps + epoch * num_batches_per_epoch
945
946             ##### BACKWARD PASS #####
947             loss.backward()
948
949             data_time_m.update(time.time() - end)
950
951             global_step = num_steps + epoch * num_batches_per_epoch
952
953             ##### BACKWARD PASS #####
954             loss.backward()
955
956             data_time_m.update(time.time() - end)
957
958             global_step = num_steps + epoch * num_batches_per_epoch
959
960             ##### BACKWARD PASS #####
961             loss.backward()
962
963             data_time_m.update(time.time() - end)
964
965             global_step = num_steps + epoch * num_batches_per_epoch
966
967             ##### BACKWARD PASS #####
968             loss.backward()
969
970             data_time_m.update(time.time() - end)
971
972             global_step = num_steps + epoch * num_batches_per_epoch
973
974             ##### BACKWARD PASS #####
975             loss.backward()
976
977             data_time_m.update(time.time() - end)
978
979             global_step = num_steps + epoch * num_batches_per_epoch
980
981             ##### BACKWARD PASS #####
982             loss.backward()
983
984             data_time_m.update(time.time() - end)
985
986             global_step = num_steps + epoch * num_batches_per_epoch
987
988             ##### BACKWARD PASS #####
989             loss.backward()
990
991             data_time_m.update(time.time() - end)
992
993             global_step = num_steps + epoch * num_batches_per_epoch
994
995             ##### BACKWARD PASS #####
996             loss.backward()
997
998             data_time_m.update(time.time() - end)
999
1000             global_step = num_steps + epoch * num_batches_per_epoch
1001
1002             ##### BACKWARD PASS #####
1003             loss.backward()
1004
1005             data_time_m.update(time.time() - end)
1006
1007             global_step = num_steps + epoch * num_batches_per_epoch
1008
1009             ##### BACKWARD PASS #####
1010             loss.backward()
1011
1012             data_time_m.update(time.time() - end)
1013
1014             global_step = num_steps + epoch * num_batches_per_epoch
1015
1016             ##### BACKWARD PASS #####
1017             loss.backward()
1018
1019             data_time_m.update(time.time() - end)
1020
1021             global_step = num_steps + epoch * num_batches_per_epoch
1022
1023             ##### BACKWARD PASS #####
1024             loss.backward()
1025
1026             data_time_m.update(time.time() - end)
1027
1028             global_step = num_steps + epoch * num_batches_per_epoch
1029
1030             ##### BACKWARD PASS #####
1031             loss.backward()
1032
1033             data_time_m.update(time.time() - end)
1034
1035             global_step = num_steps + epoch * num_batches_per_epoch
1036
1037             ##### BACKWARD PASS #####
1038             loss.backward()
1039
1040             data_time_m.update(time.time() - end)
1041
1042             global_step = num_steps + epoch * num_batches_per_epoch
1043
1044             ##### BACKWARD PASS #####
1045             loss.backward()
1046
1047             data_time_m.update(time.time() - end)
1048
1049             global_step = num_steps + epoch * num_batches_per_epoch
1050
1051             ##### BACKWARD PASS #####
1052             loss.backward()
1053
1054             data_time_m.update(time.time() - end)
1055
1056             global_step = num_steps + epoch * num_batches_per_epoch
1057
1058             ##### BACKWARD PASS #####
1059             loss.backward()
1060
1061             data_time_m.update(time.time() - end)
1062
1063             global_step = num_steps + epoch * num_batches_per_epoch
1064
1065             ##### BACKWARD PASS #####
1066             loss.backward()
1067
1068             data_time_m.update(time.time() - end)
1069
1070             global_step = num_steps + epoch * num_batches_per_epoch
1071
1072             ##### BACKWARD PASS #####
1073             loss.backward()
1074
1075             data_time_m.update(time.time() - end)
1076
1077             global_step = num_steps + epoch * num_batches_per_epoch
1078
1079             ##### BACKWARD PASS #####
1080             loss.backward()
1081
1082             data_time_m.update(time.time() - end)
1083
1084             global_step = num_steps + epoch * num_batches_per_epoch
1085
1086             ##### BACKWARD PASS #####
1087             loss.backward()
1088
1089             data_time_m.update(time.time() - end)
1090
1091             global_step = num_steps + epoch * num_batches_per_epoch
1092
1093             ##### BACKWARD PASS #####
1094             loss.backward()
1095
1096             data_time_m.update(time.time() - end)
1097
1098             global_step = num_steps + epoch * num_batches_per_epoch
1099
1100             ##### BACKWARD PASS #####
1101             loss.backward()
1102
1103             data_time_m.update(time.time() - end)
1104
1105             global_step = num_steps + epoch * num_batches_per_epoch
1106
1107             ##### BACKWARD PASS #####
1108             loss.backward()
1109
1110             data_time_m.update(time.time() - end)
1111
1112             global_step = num_steps + epoch * num_batches_per_epoch
1113
1114             ##### BACKWARD PASS #####
1115             loss.backward()
1116
1117             data_time_m.update(time.time() - end)
1118
1119             global_step = num_steps + epoch * num_batches_per_epoch
1120
1121             ##### BACKWARD PASS #####
1122             loss.backward()
1123
1124             data_time_m.update(time.time() - end)
1125
1126             global_step = num_steps + epoch * num_batches_per_epoch
1127
1128             ##### BACKWARD PASS #####
1129             loss.backward()
1130
1131             data_time_m.update(time.time() - end)
1132
1133             global_step = num_steps + epoch * num_batches_per_epoch
1134
1135             ##### BACKWARD PASS #####
1136             loss.backward()
1137
1138             data_time_m.update(time.time() - end)
1139
1140             global_step = num_steps + epoch * num_batches_per_epoch
1141
1142             ##### BACKWARD PASS #####
1143             loss.backward()
1144
1145             data_time_m.update(time.time() - end)
1146
1147             global_step = num_steps + epoch * num_batches_per_epoch
1148
1149             ##### BACKWARD PASS #####
1150             loss.backward()
1151
1152             data_time_m.update(time.time() - end)
1153
1154             global_step = num_steps + epoch
```



TOBIGS

3. 모델 코드

mmgpt/models/open_flamingo/flamingo.py

BASE MODEL

MAIN PAPER

CODE SCRATCH

논문 속 CODE SCRATCH

```

 9    class Flamingo(nn.Module):
10        def __init__(
11            self,
12            vision_encoder: nn.Module,
13            lang_encoder: nn.Module,
14            eoc_token_id: int,
15            media_token_id: int,
16            vis_dim: int,
17            cross_attn_every_n_layers: int = 1,
18            use_media_placement_augmentation: bool = False,
19        ):
20            super().__init__()
21            self.eoc_token_id = eoc_token_id
22            self.media_token_id = media_token_id
23            self.use_media_placement_augmentation = use_media_placement_augmentation
24            self.vis_dim = vis_dim
25            self.vision_encoder = vision_encoder
26            self.perceiver = PerceiverResampler(dim=self.vis_dim)
27            self.lang_encoder = lang_encoder
28            self.lang_encoder.init_flamingo(
29                media_token_id=media_token_id,
30                vis_hidden_size=self.vis_dim,
31                cross_attn_every_n_layers=cross_attn_every_n_layers,
32                use_media_placement_augmentation=use_media_placement_augmentation,
33            )
34
35
36
37
38
39
40
41
42
43
44

```

```

184    def _encode_vision_x(self, vision_x: torch.Tensor):
185        """
186            Compute media tokens from vision input by passing it through vision encoder and conditioning language model
187            Args:
188                vision_x (torch.Tensor): Vision input
189                shape (B, T_img, F, C, H, W)
190                Images in the same chunk are collated along T_img, and frames are collated along F
191                Currently only F=1 is supported (single-frame videos)
192
193            rearrange code based on https://github.com/dhansmair/flamingo-mini
194        """
195
196        assert vision_x.ndim == 6, "vision_x should be of shape (b, T_img, F, C, H, W)"
197        b, T, F = vision_x.shape[:3]
198        assert F == 1, "Only single frame supported"
199
200        vision_x = rearrange(vision_x, "b T F c h w -> (b T F) c h w")
201        with torch.no_grad():
202            vision_x = self.vision_encoder.visual(vision_x)[1]
203            vision_x = rearrange(vision_x, "(b T F) v d -> b T F v d", b=b, T=T, F=F)
204            vision_x = self.perceiver(vision_x) # reshapes to (b, T, n, d)
205
206        for layer in self.lang_encoder._get_decoder_layers():
207            layer.condition_vis_x(vision_x)
208

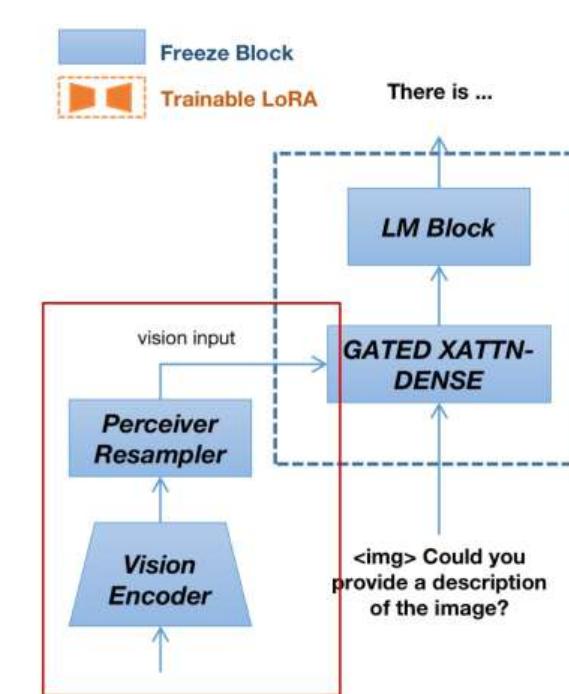
```

vision feature 추출

vision encoder 통과

perceiver resampler 통과

Language model에 vision information 전달
(for cross attention)





TOBIGS

3. 모델 코드

mmgpt/models/open_flamingo/flamingo.py

BASE MODEL

MAIN PAPER

CODE SCRATCH

논문 속 CODE SCRATCH

```
46 ~     def forward(
47         self,
48         vision_x: torch.Tensor,
49         lang_x: torch.Tensor,
50         attention_mask: torch.Tensor = None,
51         labels: torch.Tensor = None,
52         use_cached_vision_x: bool = False,
53         clear_conditioned_layers: bool = True,
54         past_key_values=None,
55         use_cache: bool = False,
56     ):
57         if vision_x is None and use_cached_vision_x is False:
58             for layer in self.lang_encoder._get_decoder_layers():
59                 layer.condition_only_lang_x(True)
60             output = self.lang_encoder(
61                 input_ids=lang_x,
62                 attention_mask=attention_mask,
63                 labels=labels,
64                 past_key_values=past_key_values,
65                 use_cache=use_cache,
66             )
67             for layer in self.lang_encoder._get_decoder_layers():
68                 layer.condition_only_lang_x(False)
69             return output
70         assert (
71             vision_x is not None
72         ) or use_cached_vision_x, "Must provide either vision_x or use_cached_vision_x to True."
```

language only instruction

```
94         if use_cached_vision_x:
95             # Case: use cached; vision_x should be cached and other
96             # vision-related inputs should not be provided.
97             assert vision_x is None, "Expect vision_x to be None when use_cached_vision_x is True."
98             assert self.lang_encoder.is_conditioned()
99
100        else: vision information encoding
101            # Case: do not use caching (i.e. this is a standard forward pass);
102            self._encode_vision_x(vision_x=vision_x)
103
104        output = self.lang_encoder(
105            input_ids=lang_x,
106            attention_mask=attention_mask,
107            labels=labels,
108            past_key_values=past_key_values,
109            use_cache=use_cache,
110        )
111
112        if clear_conditioned_layers:
113            self.lang_encoder.clear_conditioned_layers()
114
115        return output
```

vision&language instruction



TOBIGS

3. 모델 코드

mmgpt/models/open_flamingo/helpers.py

BASE MODEL

MAIN PAPER

CODE SCRATCH

논문 속 CODE SCRATCH

```

68     class PerceiverResampler(nn.Module):
69         def __init__(self,
70             *,
71                 dim,
72                 depth=6,
73                 dim_head=64,
74                 heads=8,
75                 num_latents=64,
76                 max_num_media=None,
77                 max_num_frames=None,
78                 ff_mult=4,
79             ):
80             super().__init__()
81             self.latents = nn.Parameter(torch.randn(num_latents, dim))
82             self.frame_embs = nn.Parameter(torch.randn(max_num_frames, dim)) if exists(max_num_frames) else None
83             self.media_time_embs = nn.Parameter(torch.randn(max_num_media, 1, dim)) if exists(max_num_media) else None
84
85             self.layers = nn.ModuleList([])
86             for _ in range(depth):
87                 self.layers.append(
88                     nn.ModuleList([
89                         PerceiverAttention(dim=dim, dim_head=dim_head, heads=heads),
90                         FeedForward(dim=dim, mult=ff_mult),
91                     ])
92                 )
93
94             self.norm = nn.LayerNorm(dim)
95
96
97
98
99         def forward(self, x):
100             """
101             Args:
102                 x (torch.Tensor): image features
103                     shape (b, T, F, v, D)
104
105             Returns:
106                 shape (b, T, n, D) where n is self.num_latents
107             """
108             b, T, F, v = x.shape[:4]
109
110             # frame and media time embeddings
111             if exists(self.frame_embs):
112                 frame_embs = repeat(self.frame_embs[:F], "F d -> b T F v d", b=b, T=T, v=v)
113                 x = x + frame_embs
114
115             x = rearrange(x, "b T F v d -> b T (F v) d") # flatten the frame and spatial dimensions
116             if exists(self.media_time_embs):
117                 x = x + self.media_time_embs[:T]
118
119             # blocks
120             latents = repeat(self.latents, "n d -> b T n d", b=b, T=T)
121             for attn, ff in self.layers:
122                 latents = attn(x, latents) + latents
123                 latents = ff(latents) + latents
124
125             return self.norm(latents)

```

predefined latent input query 학습(파라미터)

비디오일 경우,
frame embedding 추가



TO BIGS

3. 모델 코드

BASE MODEL

MAIN PAPER

CODE SCRATCH

논문 속 CODE SCRATCH

mmgpt/models/open_flamingo/builder.py 中

```

14     def create_model_and_transforms():
57         lang_encoder = LlamaForCausalLM.from_pretrained(lang_encoder_path)
58         extend_instance(lang_encoder, FlamingoLMMixin)
59
60         if decoder_layers_attr_name is None:
61             decoder_layers_attr_name = _infer_decoder_layers_attr_name(lang_encoder)
62         lang_encoder.set_decoder_layers_attr_name(decoder_layers_attr_name)
63         lang_encoder.resize_token_embeddings(len(text_tokenizer)) token embedding size 지정

```

mmgpt/models/open_flamingo/utils.py

```

1 ✓  def extend_instance(obj, mixin):
2      """Apply mixins to a class instance after creation"""
3      base_cls = obj.__class__
4      base_cls_name = obj.__class__.__name__
5      obj.__class__ = type(
6          base_cls_name, (mixin, base_cls), {}
7      ) # mixin needs to go first for our forward() logic to work

```

forward 수행 시,mixin class를 먼저 통과 후 기존 class 통과
 (해당 모델에서는 FlamingoLMMixin -> LlamaForCasualLM 순서로 수행)

mmgpt/models/open_flamingo/flamingo_lm.py

```

10 ✓  class FlamingoLayer(nn.Module):
11    def __init__(self, gated_cross_attn_layer, decoder_layer):
12        super().__init__()
13        self.gated_cross_attn_layer = gated_cross_attn_layer
14        self.decoder_layer = decoder_layer
15        self.vis_x = None
16        self.media_locations = None
17        self.only_lang_x = False
18
19    def is_conditioned(self) -> bool:
20        """Check whether the layer is conditioned."""
21        return self.vis_x is not None
22
23    # Used this great idea from this implementation of Flamingo (https://github.com/dhansmair/flamingo-mini/)
24    def condition_vis_x(self, vis_x):
25        self.vis_x = vis_x vision information 가져오기
26
27    def condition_only_lang_x(self, only_lang_x=False):
28        self.only_lang_x = only_lang_x
29
30    def condition_media_locations(self, media_locations):
31        self.media_locations = media_locations
32
33    def condition_attend_previous(self, attend_previous):
34        self.attend_previous = attend_previous
35
36    def forward(
37        self,
38        lang_x,
39        attention_mask=None,
40        **decoder_layer_kwargs,
41    ):
42        if self.gated_cross_attn_layer is None or self.only_lang_x:
43            return self.decoder_layer(lang_x, attention_mask=attention_mask, **decoder_layer_kwargs)
44
45        if self.vis_x is None:
46            raise ValueError("vis_x must be conditioned before forward pass")
47
48        if self.media_locations is None:
49            raise ValueError("media_locations must be conditioned before forward pass")
50
51        lang_x = self.gated_cross_attn_layer(gated cross attention을 통해
52                                             lang_x,
53                                             self.vis_x,
54                                             media_locations=self.media_locations,
55                                             attend_previous=self.attend_previous,
56                                         )
57        lang_x = self.decoder_layer(lang_x, attention_mask=attention_mask, **decoder_layer_kwargs)
58

```



TOBIGS

3. 모델 코드

[transformers 라이브러리] transformers/src/transformers/llama/modeling_llma.py

BASE MODEL

MAIN PAPER

CODE SCRATCH

논문 속 CODE SCRATCH

```
611     class LlamaForCausalLM(LlamaPreTrainedModel):
612         def __init__(self, config):
613             super().__init__(config)
614             self.model = LlamaModel(config)
615
616             self.lm_head = nn.Linear(config.hidden_size, config.vocab_size, bias=False)
617
618             def forward(
619
620                 output_attentions = output_attentions if output_attentions is not None else self.config.output_atten
621                 output_hidden_states =
622                     output_hidden_states if output_hidden_states is not None else self.config.output_hidden_states
623                 )
624                 return_dict = return_dict if return_dict is not None else self.config.use_return_dict
625
626                 # decoder outputs consists of (dec_features, layer_state, dec_hidden, dec_attn)
627                 outputs = self.model
628                     Language model 통과
629                     input_ids=input_ids,
630                     attention_mask=attention_mask,
631                     position_ids=position_ids,
632                     past_key_values=past_key_values,
633                     inputs_embeds=inputs_embeds,
634                     use_cache=use_cache,
635                     output_attentions=output_attentions,
636                     output_hidden_states=output_hidden_states,
637                     return_dict=return_dict,
638                 )
639
640
641             hidden_states = outputs[0]
642             logits = self.lm_head(hidden_states) lm_head 통과
643
644             loss = None
645             if labels is not None:
646                 # Shift so that tokens < n predict n
647                 shift_logits = logits[..., :-1, :].contiguous()
648                 shift_labels = labels[..., 1:].contiguous()
649
650                 # Flatten the tokens
651                 loss_fct = CrossEntropyLoss() CrossEntropy Loss 사용
652                 shift_logits = shift_logits.view(-1, self.config.vocab_size)
653                 shift_labels = shift_labels.view(-1)
654
655                 # Enable model parallelism
656                 shift_labels = shift_labels.to(shift_logits.device)
657                 loss = loss_fct(shift_logits, shift_labels)
658
659
660             if not return_dict:
661                 output = (logits,) + outputs[1:]
662                 return (loss,) + output if loss is not None else output
663
664             return CausalLMOutputWithPast(
665                 loss=loss,
666                 logits=logits,
667                 past_key_values=outputs.past_key_values,
668                 hidden_states=outputs.hidden_states,
669                 attentions=outputs.attentions,
670             )
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725 )
```



TOBIGS

3. 모델 코드

[transformers 라이브러리] transformers/src/transformers/llama/modeling_llma.py

```

431     class LlamaModel(LlamaPreTrainedModel):
432         def forward(
433             self,
434             input_ids=None,
435             past_key_values=None,
436             position_ids=None,
437             inputs_embeds=None,
438             attention_mask=None,
439             past_key_value=None,
440             use_cache=True,
441             output_attentions=False,
442             output_hidden_states=False,
443             return_dict=True,
444             **kwargs
445         ):
446             r"""
447                 position_ids (intTensor): position_ids는 각 토큰의 위치를 나타내는 정수 텐서입니다. 이는 시�퀀스의 길이와 일치해야 합니다. 예를 들어, [CLS] 토큰은 0으로 표시되고, 다음 토큰은 1로 표시됩니다. 이 텐서는 모델의 각 레이어에 전달됩니다.
448                 inputs_embeds (floatTensor): input_ids 대신 토큰의 임베디팅을 제공하는 텐서입니다. 이는 모델의 각 레이어에 전달됩니다.
449                 attention_mask (boolTensor): 시�퀀스에서 주목해야 하는 토큰을 표시하는 마스크 텐서입니다. 이는 모델의 각 레이어에 전달됩니다.
450                 past_key_value (floatTensor): 이전 키 및 값 텐서입니다. 이는 모델의 각 레이어에 전달됩니다.
451                 use_cache (bool): 캐시 사용 여부입니다. True면 캐시를 사용하고, False면 캐시를 사용하지 않습니다.
452                 output_attentions (bool): 어텐션 출력 여부입니다. True면 어텐션 출력을 포함하는 딕셔너리를 반환합니다.
453                 output_hidden_states (bool): 히든 스테이트 출력 여부입니다. True면 히든 스테이트 출력을 포함하는 딕셔너리를 반환합니다.
454                 return_dict (bool): 딕셔너리 형태로 결과를 반환 여부입니다. True면 딕셔너리를 반환하고, False면 튜플을 반환합니다.
455             """
456             if input_ids is not None:
457                 input_ids = input_ids.to(self.device)
458                 inputs_embeds = self.embed_tokens(input_ids)
459
460             if position_ids is None:
461                 position_ids = torch.arange(
462                     seq_length_with_past + past_key_values_length,
463                     seq_length_with_past + seq_length,
464                     dtype=torch.long,
465                     device=device
466                 )
467
468             if attention_mask is None:
469                 attention_mask = torch.ones(
470                     (batch_size, seq_length_with_past),
471                     dtype=torch.bool,
472                     device=inputs_embeds.device
473                 )
474
475             attention_mask = self._prepare_decoder_attention_mask(
476                 attention_mask,
477                 (batch_size, seq_length),
478                 inputs_embeds,
479                 past_key_values_length
480             )
481
482             hidden_states = inputs_embeds
483
484             for idx, decoder_layer in enumerate(self.layers):
485                 if output_hidden_states:
486                     all_hidden_states += (hidden_states,)
487
488                 past_key_value = past_key_values[idx] if past_key_values is not None else None
489
490                 if self.gradient_checkpointing and self.training:
491                     def create_custom_forward(module):
492                         def custom_forward(*inputs):
493                             # None for past_key_value
494                             return module(*inputs, output_attentions, None)
495
496                     return custom_forward
497
498                 layer_outputs = torch.utils.checkpoint.checkpoint(
499                     create_custom_forward(decoder_layer),
500                     hidden_states,
501                     attention_mask,
502                     position_ids,
503                     None,
504                 )
505
506                 else:
507                     layer_outputs = decoder_layer(
508                         hidden_states,
509                         attention_mask=attention_mask,
510                         position_ids=position_ids,
511                         past_key_value=past_key_value,
512                         output_attentions=output_attentions,
513                         use_cache=use_cache,
514                     )
515
516                 hidden_states = layer_outputs[0]
517
518                 if use_cache:
519                     next_decoder_cache += (layer_outputs[2] if output_attentions else 1, )
520
521                 if output_attentions:
522                     all_self_attns += (layer_outputs[1], )
523
524             hidden_states = self.norm(hidden_states)
525
526             return BaseModelOutputWithPast(
527                 last_hidden_state=hidden_states,
528                 past_key_value=past_key_value,
529                 hidden_states=all_hidden_states,
530                 attentions=all_self_attns,
531                 cache=next_decoder_cache
532             )
533
534         def _prepare_decoder_attention_mask(
535             self,
536             attention_mask,
537             input_shape,
538             batch_size,
539             seq_length,
540             past_key_values_length
541         ):
542             r"""
543                 attention_mask (boolTensor): 시�퀀스에서 주목해야 하는 토큰을 표시하는 마스크 텐서입니다. 이는 모델의 각 레이어에 전달됩니다.
544                 input_shape (tuple): 시�퀀스의 차원입니다.
545                 batch_size (int): 배치 크기입니다.
546                 seq_length (int): 시�퀀스의 길이입니다.
547                 past_key_values_length (int): 이전 키 및 값 텐서의 길이입니다.
548             """
549             if attention_mask is None:
550                 attention_mask = torch.ones(
551                     (batch_size, seq_length),
552                     dtype=torch.bool,
553                     device=input_shape[-1].device
554                 )
555
556             attention_mask = attention_mask.unsqueeze(-1).unsqueeze(-1)
557
558             return attention_mask
559
560         def _get_norm_fn(self):
561             if self.config.normalize_x:
562                 return nn.LayerNorm(self.config.hidden_size)
563             else:
564                 return nn.Identity()
565
566         def _get_clones(self, module, N):
567             return nn.ModuleList([copy.deepcopy(module) for _ in range(N)])
568
569         def _get_bias(self, direction='right'):
570             if direction == 'left':
571                 return nn.Parameter(torch.zeros(1))
572             else:
573                 return nn.Parameter(torch.zeros(1))
574
575         def _get_weight(self, direction='right'):
576             if direction == 'left':
577                 return nn.Parameter(torch.ones(1))
578             else:
579                 return nn.Parameter(torch.ones(1))
580
581         def _get_norm(self, direction='right'):
582             if direction == 'left':
583                 return nn.LayerNorm(self.config.hidden_size)
584             else:
585                 return nn.Identity()
586
587         def _get_bias_and_weight(self, direction='right'):
588             if direction == 'left':
589                 return nn.Parameter(torch.zeros(1)), nn.Parameter(torch.ones(1))
590             else:
591                 return nn.Parameter(torch.zeros(1)), nn.Parameter(torch.ones(1))
592
593         def _get_norm_fn(self):
594             if self.config.normalize_x:
595                 return nn.LayerNorm(self.config.hidden_size)
596             else:
597                 return nn.Identity()
598
599         def _get_clones(self, module, N):
600             return nn.ModuleList([copy.deepcopy(module) for _ in range(N)])
601
602         def _get_bias(self, direction='right'):
603             if direction == 'left':
604                 return nn.Parameter(torch.zeros(1))
605             else:
606                 return nn.Parameter(torch.zeros(1))
607
608         def _get_weight(self, direction='right'):
609             if direction == 'left':
610                 return nn.Parameter(torch.ones(1))
611             else:
612                 return nn.Parameter(torch.ones(1))
613
614         def _get_norm(self, direction='right'):
615             if direction == 'left':
616                 return nn.LayerNorm(self.config.hidden_size)
617             else:
618                 return nn.Identity()
619
620         def _get_bias_and_weight(self, direction='right'):
621             if direction == 'left':
622                 return nn.Parameter(torch.zeros(1)), nn.Parameter(torch.ones(1))
623             else:
624                 return nn.Parameter(torch.zeros(1)), nn.Parameter(torch.ones(1))
625
626         def _get_norm_fn(self):
627             if self.config.normalize_x:
628                 return nn.LayerNorm(self.config.hidden_size)
629             else:
630                 return nn.Identity()
631
632         def _get_clones(self, module, N):
633             return nn.ModuleList([copy.deepcopy(module) for _ in range(N)])
634
635         def _get_bias(self, direction='right'):
636             if direction == 'left':
637                 return nn.Parameter(torch.zeros(1))
638             else:
639                 return nn.Parameter(torch.zeros(1))
640
641         def _get_weight(self, direction='right'):
642             if direction == 'left':
643                 return nn.Parameter(torch.ones(1))
644             else:
645                 return nn.Parameter(torch.ones(1))
646
647         def _get_norm(self, direction='right'):
648             if direction == 'left':
649                 return nn.LayerNorm(self.config.hidden_size)
650             else:
651                 return nn.Identity()
652
653         def _get_bias_and_weight(self, direction='right'):
654             if direction == 'left':
655                 return nn.Parameter(torch.zeros(1)), nn.Parameter(torch.ones(1))
656             else:
657                 return nn.Parameter(torch.zeros(1)), nn.Parameter(torch.ones(1))
658
659         def _get_norm_fn(self):
660             if self.config.normalize_x:
661                 return nn.LayerNorm(self.config.hidden_size)
662             else:
663                 return nn.Identity()
664
665         def _get_clones(self, module, N):
666             return nn.ModuleList([copy.deepcopy(module) for _ in range(N)])
667
668         def _get_bias(self, direction='right'):
669             if direction == 'left':
670                 return nn.Parameter(torch.zeros(1))
671             else:
672                 return nn.Parameter(torch.zeros(1))
673
674         def _get_weight(self, direction='right'):
675             if direction == 'left':
676                 return nn.Parameter(torch.ones(1))
677             else:
678                 return nn.Parameter(torch.ones(1))
679
680         def _get_norm(self, direction='right'):
681             if direction == 'left':
682                 return nn.LayerNorm(self.config.hidden_size)
683             else:
684                 return nn.Identity()
685
686         def _get_bias_and_weight(self, direction='right'):
687             if direction == 'left':
688                 return nn.Parameter(torch.zeros(1)), nn.Parameter(torch.ones(1))
689             else:
690                 return nn.Parameter(torch.zeros(1)), nn.Parameter(torch.ones(1))
691
692         def _get_norm_fn(self):
693             if self.config.normalize_x:
694                 return nn.LayerNorm(self.config.hidden_size)
695             else:
696                 return nn.Identity()
697
698         def _get_clones(self, module, N):
699             return nn.ModuleList([copy.deepcopy(module) for _ in range(N)])
700
701         def _get_bias(self, direction='right'):
702             if direction == 'left':
703                 return nn.Parameter(torch.zeros(1))
704             else:
705                 return nn.Parameter(torch.zeros(1))
706
707         def _get_weight(self, direction='right'):
708             if direction == 'left':
709                 return nn.Parameter(torch.ones(1))
710             else:
711                 return nn.Parameter(torch.ones(1))
712
713         def _get_norm(self, direction='right'):
714             if direction == 'left':
715                 return nn.LayerNorm(self.config.hidden_size)
716             else:
717                 return nn.Identity()
718
719         def _get_bias_and_weight(self, direction='right'):
720             if direction == 'left':
721                 return nn.Parameter(torch.zeros(1)), nn.Parameter(torch.ones(1))
722             else:
723                 return nn.Parameter(torch.zeros(1)), nn.Parameter(torch.ones(1))
724
725         def _get_norm_fn(self):
726             if self.config.normalize_x:
727                 return nn.LayerNorm(self.config.hidden_size)
728             else:
729                 return nn.Identity()
730
731         def _get_clones(self, module, N):
732             return nn.ModuleList([copy.deepcopy(module) for _ in range(N)])
733
734         def _get_bias(self, direction='right'):
735             if direction == 'left':
736                 return nn.Parameter(torch.zeros(1))
737             else:
738                 return nn.Parameter(torch.zeros(1))
739
740         def _get_weight(self, direction='right'):
741             if direction == 'left':
742                 return nn.Parameter(torch.ones(1))
743             else:
744                 return nn.Parameter(torch.ones(1))
745
746         def _get_norm(self, direction='right'):
747             if direction == 'left':
748                 return nn.LayerNorm(self.config.hidden_size)
749             else:
750                 return nn.Identity()
751
752         def _get_bias_and_weight(self, direction='right'):
753             if direction == 'left':
754                 return nn.Parameter(torch.zeros(1)), nn.Parameter(torch.ones(1))
755             else:
756                 return nn.Parameter(torch.zeros(1)), nn.Parameter(torch.ones(1))
757
758         def _get_norm_fn(self):
759             if self.config.normalize_x:
760                 return nn.LayerNorm(self.config.hidden_size)
761             else:
762                 return nn.Identity()
763
764         def _get_clones(self, module, N):
765             return nn.ModuleList([copy.deepcopy(module) for _ in range(N)])
766
767         def _get_bias(self, direction='right'):
768             if direction == 'left':
769                 return nn.Parameter(torch.zeros(1))
770             else:
771                 return nn.Parameter(torch.zeros(1))
772
773         def _get_weight(self, direction='right'):
774             if direction == 'left':
775                 return nn.Parameter(torch.ones(1))
776             else:
777                 return nn.Parameter(torch.ones(1))
778
779         def _get_norm(self, direction='right'):
780             if direction == 'left':
781                 return nn.LayerNorm(self.config.hidden_size)
782             else:
783                 return nn.Identity()
784
785         def _get_bias_and_weight(self, direction='right'):
786             if direction == 'left':
787                 return nn.Parameter(torch.zeros(1)), nn.Parameter(torch.ones(1))
788             else:
789                 return nn.Parameter(torch.zeros(1)), nn.Parameter(torch.ones(1))
790
791         def _get_norm_fn(self):
792             if self.config.normalize_x:
793                 return nn.LayerNorm(self.config.hidden_size)
794             else:
795                 return nn.Identity()
796
797         def _get_clones(self, module, N):
798             return nn.ModuleList([copy.deepcopy(module) for _ in range(N)])
799
800         def _get_bias(self, direction='right'):
801             if direction == 'left':
802                 return nn.Parameter(torch.zeros(1))
803             else:
804                 return nn.Parameter(torch.zeros(1))
805
806         def _get_weight(self, direction='right'):
807             if direction == 'left':
808                 return nn.Parameter(torch.ones(1))
809             else:
810                 return nn.Parameter(torch.ones(1))
811
812         def _get_norm(self, direction='right'):
813             if direction == 'left':
814                 return nn.LayerNorm(self.config.hidden_size)
815             else:
816                 return nn.Identity()
817
818         def _get_bias_and_weight(self, direction='right'):
819             if direction == 'left':
820                 return nn.Parameter(torch.zeros(1)), nn.Parameter(torch.ones(1))
821             else:
822                 return nn.Parameter(torch.zeros(1)), nn.Parameter(torch.ones(1))
823
824         def _get_norm_fn(self):
825             if self.config.normalize_x:
826                 return nn.LayerNorm(self.config.hidden_size)
827             else:
828                 return nn.Identity()
829
830         def _get_clones(self, module, N):
831             return nn.ModuleList([copy.deepcopy(module) for _ in range(N)])
832
833         def _get_bias(self, direction='right'):
834             if direction == 'left':
835                 return nn.Parameter(torch.zeros(1))
836             else:
837                 return nn.Parameter(torch.zeros(1))
838
839         def _get_weight(self, direction='right'):
840             if direction == 'left':
841                 return nn.Parameter(torch.ones(1))
842             else:
843                 return nn.Parameter(torch.ones(1))
844
845         def _get_norm(self, direction='right'):
846             if direction == 'left':
847                 return nn.LayerNorm(self.config.hidden_size)
848             else:
849                 return nn.Identity()
850
851         def _get_bias_and_weight(self, direction='right'):
852             if direction == 'left':
853                 return nn.Parameter(torch.zeros(1)), nn.Parameter(torch.ones(1))
854             else:
855                 return nn.Parameter(torch.zeros(1)), nn.Parameter(torch.ones(1))
856
857         def _get_norm_fn(self):
858             if self.config.normalize_x:
859                 return nn.LayerNorm(self.config.hidden_size)
860             else:
861                 return nn.Identity()
862
863         def _get_clones(self, module, N):
864             return nn.ModuleList([copy.deepcopy(module) for _ in range(N)])
865
866         def _get_bias(self, direction='right'):
867             if direction == 'left':
868                 return nn.Parameter(torch.zeros(1))
869             else:
870                 return nn.Parameter(torch.zeros(1))
871
872         def _get_weight(self, direction='right'):
873             if direction == 'left':
874                 return nn.Parameter(torch.ones(1))
875             else:
876                 return nn.Parameter(torch.ones(1))
877
878         def _get_norm(self, direction='right'):
879             if direction == 'left':
880                 return nn.LayerNorm(self.config.hidden_size)
881             else:
882                 return nn.Identity()
883
884         def _get_bias_and_weight(self, direction='right'):
885             if direction == 'left':
886                 return nn.Parameter(torch.zeros(1)), nn.Parameter(torch.ones(1))
887             else:
888                 return nn.Parameter(torch.zeros(1)), nn.Parameter(torch.ones(1))
889
890         def _get_norm_fn(self):
891             if self.config.normalize_x:
892                 return nn.LayerNorm(self.config.hidden_size)
893             else:
894                 return nn.Identity()
895
896         def _get_clones(self, module, N):
897             return nn.ModuleList([copy.deepcopy(module) for _ in range(N)])
898
899         def _get_bias(self, direction='right'):
900             if direction == 'left':
901                 return nn.Parameter(torch.zeros(1))
902             else:
903                 return nn.Parameter(torch.zeros(1))
904
905         def _get_weight(self, direction='right'):
906             if direction == 'left':
907                 return nn.Parameter(torch.ones(1))
908             else:
909                 return nn.Parameter(torch.ones(1))
910
911         def _get_norm(self, direction='right'):
912             if direction == 'left':
913                 return nn.LayerNorm(self.config.hidden_size)
914             else:
915                 return nn.Identity()
916
917         def _get_bias_and_weight(self, direction='right'):
918             if direction == 'left':
919                 return nn.Parameter(torch.zeros(1)), nn.Parameter(torch.ones(1))
920             else:
921                 return nn.Parameter(torch.zeros(1)), nn.Parameter(torch.ones(1))
922
923         def _get_norm_fn(self):
924             if self.config.normalize_x:
925                 return nn.LayerNorm(self.config.hidden_size)
926             else:
927                 return nn.Identity()
928
929         def _get_clones(self, module, N):
930             return nn.ModuleList([copy.deepcopy(module) for _ in range(N)])
931
932         def _get_bias(self, direction='right'):
933             if direction == 'left':
934                 return nn.Parameter(torch.zeros(1))
935             else:
936                 return nn.Parameter(torch.zeros(1))
937
938         def _get_weight(self, direction='right'):
939             if direction == 'left':
940                 return nn.Parameter(torch.ones(1))
941             else:
942                 return nn.Parameter(torch.ones(1))
943
944         def _get_norm(self, direction='right'):
945             if direction == 'left':
946                 return nn.LayerNorm(self.config.hidden_size)
947             else:
948                 return nn.Identity()
949
950         def _get_bias_and_weight(self, direction='right'):
951             if direction == 'left':
952                 return nn.Parameter(torch.zeros(1)), nn.Parameter(torch.ones(1))
953             else:
954                 return nn.Parameter(torch.zeros(1)), nn.Parameter(torch.ones(1))
955
956         def _get_norm_fn(self):
957             if self.config.normalize_x:
958                 return nn.LayerNorm(self.config.hidden_size)
959             else:
960                 return nn.Identity()
961
962         def _get_clones(self, module, N):
963             return nn.ModuleList([copy.deepcopy(module) for _ in range(N)])
964
965         def _get_bias(self, direction='right'):
966             if direction == 'left':
967                 return nn.Parameter(torch.zeros(1))
968             else:
969                 return nn.Parameter(torch.zeros(1))
970
971         def _get_weight(self, direction='right'):
972             if direction == 'left':
973                 return nn.Parameter(torch.ones(1))
974             else:
975                 return nn.Parameter(torch.ones(1))
976
977         def _get_norm(self, direction='right'):
978             if direction == 'left':
979                 return nn.LayerNorm(self.config.hidden_size)
980             else:
981                 return nn.Identity()
982
983         def _get_bias_and_weight(self, direction='right'):
984             if direction == 'left':
985                 return nn.Parameter(torch.zeros(1)), nn.Parameter(torch.ones(1))
986             else:
987                 return nn.Parameter(torch.zeros(1)), nn.Parameter(torch.ones(1))
988
989         def _get_norm_fn(self):
990             if self.config.normalize_x:
991                 return nn.LayerNorm(self.config.hidden_size)
992             else:
993                 return nn.Identity()
994
995         def _get_clones(self, module, N):
996             return nn.ModuleList([copy.deepcopy(module) for _ in range(N)])
997
998         def _get_bias(self, direction='right'):
999             if direction == 'left':
1000                 return nn.Parameter(torch.zeros(1))
1001             else:
1002                 return nn.Parameter(torch.zeros(1))
1003
1004         def _get_weight(self, direction='right'):
1005             if direction == 'left':
1006                 return nn.Parameter(torch.ones(1))
1007             else:
1008                 return nn.Parameter(torch.ones(1))
1009
1010         def _get_norm(self, direction='right'):
1011             if direction == 'left':
1012                 return nn.LayerNorm(self.config.hidden_size)
1013             else:
1014                 return nn.Identity()
1015
1016         def _get_bias_and_weight(self, direction='right'):
1017             if direction == 'left':
1018                 return nn.Parameter(torch.zeros(1)), nn.Parameter(torch.ones(1))
1019             else:
1020                 return nn.Parameter(torch.zeros(1)), nn.Parameter(torch.ones(1))
1021
1022         def _get_norm_fn(self):
1023             if self.config.normalize_x:
1024                 return nn.LayerNorm(self.config.hidden_size)
1025             else:
1026                 return nn.Identity()
1027
1028         def _get_clones(self, module, N):
1029             return nn.ModuleList([copy.deepcopy(module) for _ in range(N)])
1030
1031         def _get_bias(self, direction='right'):
1032             if direction == 'left':
1033                 return nn.Parameter(torch.zeros(1))
1034             else:
1035                 return nn.Parameter(torch.zeros(1))
1036
1037         def _get_weight(self, direction='right'):
1038             if direction == 'left':
1039                 return nn.Parameter(torch.ones(1))
1040             else:
1041                 return nn.Parameter(torch.ones(1))
1042
1043         def _get_norm(self, direction='right'):
1044             if direction == 'left':
1045                 return nn.LayerNorm(self.config.hidden_size)
1046             else:
1047                 return nn.Identity()
1048
1049         def _get_bias_and_weight(self, direction='right'):
1050             if direction == 'left':
1051                 return nn.Parameter(torch.zeros(1)), nn.Parameter(torch.ones(1))
1052             else:
1053                 return nn.Parameter(torch.zeros(1)), nn.Parameter(torch.ones(1))
1054
1055         def _get_norm_fn(self):
1056             if self.config.normalize_x:
1057                 return nn.LayerNorm(self.config.hidden_size)
1058             else:
1059                 return nn.Identity()
1060
1061         def _get_clones(self, module, N):
1062             return nn.ModuleList([copy.deepcopy(module) for _ in range(N)])
1063
1064         def _get_bias(self, direction='right'):
1065             if direction == 'left':
1066                 return nn.Parameter(torch.zeros(1))
1067             else:
1068                 return nn.Parameter(torch.zeros(1))
1069
1070         def _get_weight(self, direction='right'):
1071             if direction == 'left':
1072                 return nn.Parameter(torch.ones(1))
1073             else:
1074                 return nn.Parameter(torch.ones(1))
1075
1076         def _get_norm(self, direction='right'):
1077             if direction == 'left':
1078                 return nn.LayerNorm(self.config.hidden_size)
1079             else:
1080                 return nn.Identity()
1081
1082         def _get_bias_and_weight(self, direction='right'):
1083             if direction == 'left':
1084                 return nn.Parameter(torch.zeros(1)), nn.Parameter(torch.ones(1))
1085             else:
1086                 return nn.Parameter(torch.zeros(1)), nn.Parameter(torch.ones(1))
1087
1088         def _get_norm_fn(self):
1089             if self.config.normalize_x:
1090                 return nn.LayerNorm(self.config.hidden_size)
1091             else:
1092                 return nn.Identity()
1093
1094         def _get_clones(self, module, N):
1095             return nn.ModuleList([copy.deepcopy(module) for _ in range(N)])
1096
1097         def _get_bias(self, direction='right'):
1098             if direction == 'left':
1099                 return nn.Parameter(torch.zeros(1))
1100             else:
1101                 return nn.Parameter(torch.zeros(1))
1102
1103         def _get_weight(self, direction='right'):
1104             if direction == 'left':
1105                 return nn.Parameter(torch.ones(1))
1106             else:
1107                 return nn.Parameter(torch.ones(1))
1108
1109         def _get_norm(self, direction='right'):
1110             if direction == 'left':
1111                 return nn.LayerNorm(self.config.hidden_size)
1112             else:
1113                 return nn.Identity()
1114
1115         def _get_bias_and_weight(self, direction='right'):
1116             if direction == 'left':
1117                 return nn.Parameter(torch.zeros(1)), nn.Parameter(torch.ones(1))
1118             else:
1119                 return nn.Parameter(torch.zeros(1)), nn.Parameter(torch.ones(1))
1120
1121         def _get_norm_fn(self):
1122             if self.config.normalize_x:
1123                 return nn.LayerNorm(self.config.hidden_size)
1124             else:
1125                 return nn.Identity()
1126
1127         def _get_clones(self, module, N):
1128             return nn.ModuleList([copy.deepcopy(module) for _ in range(N)])
1129
1130         def _get_bias(self, direction='right'):
1131             if direction == 'left':
1132                 return nn.Parameter(torch.zeros(1))
1133             else:
1134                 return nn.Parameter(torch.zeros(1))
1135
1136         def _get_weight(self, direction='right'):
1137             if direction == '
```



TOBIGS

3. 모델 코드

BASE MODEL

MAIN PAPER

CODE SCRATCH

논문 속 CODE SCRATCH

mmgpt/models/open_flamingo/builder.py

```

115     def prepare_model_for_tuning(model: nn.Module, config):
116         if config.lora:
117             lora_config = LoraConfig(
118                 r=config.lora_r,
119                 lora_alpha=config.lora_alpha,
120                 target_modules=config.lora_target_modules,
121                 lora_dropout=config.lora_dropout,
122                 bias="none", # won't use bias currently
123                 modules_to_save=[], # TODO: might be helpful if save partial model
124                 task_type="VL",   LORA 모델 구조 결합 (language model만)
125             )
126             model.lang_encoder = get_peft_model(model.lang_encoder, peft_config=lora_config)
127
128             # manually unfreeze modules, we use a 'substring' fashion matching
129             for name, param in model.named_parameters():
130                 if any(substr in name for substr in config.unfrozen):
131                     param.requires_grad = True
132
133                 if config.vis and is_rank0():
134                     Visualization(model).structure_graph()
135             return model
        ) 나머지 파라미터 freeze

```

[peft 라이브러리] src/peft/mapping.py

```

27     MODEL_TYPE_TO_PEFT_MODEL_MAPPING = {
28         "SEQ_CLS": PeftModelForSequenceClassification,
29         "SEQ_2_SEQ_LM": PeftModelForSeq2SeqLM,
30         "CAUSAL_LM": PeftModelForCausalLM,
31         "TOKEN_CLS": PeftModelForTokenClassification,
32     }
33     def get_peft_model(model, peft_config):
34         """
35             Returns a Peft model object from a model and a config.
36
37             Args:
38                 model (``transformers.PreTrainedModel``): Model to be wrapped.
39                 peft_config (``PeftConfig``): Configuration object containing the parameters of the Peft model.
40
41             """
42
43             model_config = model.config.to_dict()
44             peft_config.base_model_name_or_path = model._dict_.get("name_or_path", None)
45             if peft_config.task_type not in MODEL_TYPE_TO_PEFT_MODEL_MAPPING.keys():
46                 peft_config = _prepare_lora_config(peft_config, model_config)
47                 return PeftModel(model, peft_config) 모델 형태에 알맞게 LoRA 가져옴
48             if not isinstance(peft_config, PromptLearningConfig):
49                 peft_config = _prepare_lora_config(peft_config, model_config)
50             else:
51                 peft_config = _prepare_prompt_learning_config(peft_config, model_config)
52             return MODEL_TYPE_TO_PEFT_MODEL_MAPPING[peft_config.task_type](model, peft_config)
53

```



TO BIGS

3. 모델 코드

[peft 라이브러리] src/peft/peft_model.py

```

46     class PeftModel(PushToHubMixin, torch.nn.Module):
47         def __init__(self, model, peft_config: PeftConfig):
48             if isinstance(self.peft_config, PromptLearningConfig):
49                 self._setup_prompt_encoder()
50             else:
51                 self.base_model = LoraModel(peft_config, model)
52             if getattr(self.peft_config, "modules_to_save", None) is not None:
53                 self.modules_to_save = self.peft_config.modules_to_save
54             _set_trainable(self)
55             self.device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
56
57             # Actual trainable parameters
58             if r > 0:    행렬 분해
59                 self.lora_A = nn.Linear(in_features, r, bias=False)
60                 self.lora_B = nn.Linear(r, out_features, bias=False)
61                 self.scaling = self.lora_alpha / self.r
62                 # Freezing the pre-trained weight matrix
63                 self.weight.requires_grad = False
64                 self.reset_parameters()
65                 if fan_in_fan_out:
66                     self.weight.data = self.weight.data.T
67
68             self.lora_A = nn.Linear(in_features, r * sum(enable_lora), bias=False)
69             self.lora_B = nn.Conv1d(
70                 r * sum(enable_lora),
71                 out_features // len(enable_lora) * sum(enable_lora),
72                 kernel_size=1,
73                 groups=2,
74                 bias=False,
75             )
76             self.scaling = self.lora_alpha / self.r
77             # Freezing the pre-trained weight matrix
78             self.weight.requires_grad = False
79             # Compute the indices
80             self.lora_ind = self.weight.new_zeros((out_features,), dtype=torch.bool).view(len(enable_lora),
81             self.lora_ind[enable_lora, :] = True
82             self.lora_ind = self.lora_ind.view(-1)
83             self.reset_parameters()
84
85             # Find and replace
86             self._find_and_replace()
87             mark_only_lora_as_trainable(self.model, self.peft_config.bias)
88             self.forward = self.model.forward
89             def _find_and_replace(self):
90                 is_target_modules_in_base_model = False
91                 kwargs = {
92                     "r": self.peft_config.r,
93                     "lora_alpha": self.peft_config.lora_alpha,
94                     "lora_dropout": self.peft_config.lora_dropout,
95                     "fan_in_fan_out": self.peft_config.fan_in_fan_out,
96                     "merge_weights": self.peft_config.merge_weights,
97                 }
98                 key_list = [key for key, _ in self.model.named_modules()]
99                 for key in key_list:  tuning을 설정한 파라미터 모두 수행
100                     if any(key.endswith(target_key) for target_key in self.peft_config.target_modules):
101                         if not is_target_modules_in_base_model:
102                             is_target_modules_in_base_model = True
103                             parent, target, target_name = self._get_submodules(key)
104                             bias = target.bias is not None
105                             if isinstance(target, bnb.nn.Linear8bitLt) and self.peft_config.enable_lora is None:
106                                 kwargs.update(
107                                     {
108                                         "has_fp16_weights": target.state.has_fp16_weights,
109                                         "memory_efficient_backward": target.state.memory_efficient_backward,
110                                         "threshold": target.state.threshold,
111                                         "index": target.index,
112                                     }
113                                 )
114                             new_module = Linear8bitLt(target.in_features, target.out_features, bias=bias, **kwargs)
115                         elif isinstance(target, torch.nn.Linear) and self.peft_config.enable_lora is None:
116                             new_module = Linear(target.in_features, target.out_features, bias=bias, **kwargs)
117                         elif self.peft_config.enable_lora is not None:
118                             kwargs.update({"enable_lora": self.peft_config.enable_lora})
119                             if isinstance(target, Conv1D):
120                                 in_features, out_features = target.weight.shape
121                             else:
122                                 in_features, out_features = target.in_features, target.out_features
123                             if kwargs["fan_in_fan_out"]:
124                                 warnings.warn(
125                                     "fan_in_fan_out is set to True but the target module is not a Conv1D."
126                                     "Setting fan_in_fan_out to False."
127                                 )
128                             kwargs["fan_in_fan_out"] = False
129                             new_module = MergedLinear(in_features, out_features, bias=bias, **kwargs)
130                         self._replace_module(parent, target_name, new_module, target)
131
132             # Find and replace
133             self._find_and_replace()
134             mark_only_lora_as_trainable(self.model, self.peft_config.bias)
135             self.forward = self.model.forward
136             def _find_and_replace(self):
137                 is_target_modules_in_base_model = False
138                 kwargs = {
139                     "r": self.peft_config.r,
140                     "lora_alpha": self.peft_config.lora_alpha,
141                     "lora_dropout": self.peft_config.lora_dropout,
142                     "fan_in_fan_out": self.peft_config.fan_in_fan_out,
143                     "merge_weights": self.peft_config.merge_weights,
144                 }
145                 key_list = [key for key, _ in self.model.named_modules()]
146                 for key in key_list:  tuning을 설정한 파라미터 모두 수행
147                     if any(key.endswith(target_key) for target_key in self.peft_config.target_modules):
148                         if not is_target_modules_in_base_model:
149                             is_target_modules_in_base_model = True
150                             parent, target, target_name = self._get_submodules(key)
151                             bias = target.bias is not None
152                             if isinstance(target, bnb.nn.Linear8bitLt) and self.peft_config.enable_lora is None:
153                                 kwargs.update(
154                                     {
155                                         "has_fp16_weights": target.state.has_fp16_weights,
156                                         "memory_efficient_backward": target.state.memory_efficient_backward,
157                                         "threshold": target.state.threshold,
158                                         "index": target.index,
159                                     }
160                                 )
161                             new_module = Linear8bitLt(target.in_features, target.out_features, bias=bias, **kwargs)
162                         elif isinstance(target, torch.nn.Linear) and self.peft_config.enable_lora is None:
163                             new_module = Linear(target.in_features, target.out_features, bias=bias, **kwargs)
164                         elif self.peft_config.enable_lora is not None:
165                             kwargs.update({"enable_lora": self.peft_config.enable_lora})
166                             if isinstance(target, Conv1D):
167                                 in_features, out_features = target.weight.shape
168                             else:
169                                 in_features, out_features = target.in_features, target.out_features
170                             if kwargs["fan_in_fan_out"]:
171                                 warnings.warn(
172                                     "fan_in_fan_out is set to True but the target module is not a Conv1D."
173                                     "Setting fan_in_fan_out to False."
174                                 )
175                             kwargs["fan_in_fan_out"] = False
176                             new_module = MergedLinear(in_features, out_features, bias=bias, **kwargs)
177                         self._replace_module(parent, target_name, new_module, target)
178
179             # Find and replace
180             self._find_and_replace()
181             mark_only_lora_as_trainable(self.model, self.peft_config.bias)
182             self.forward = self.model.forward
183             def _find_and_replace(self):
184                 is_target_modules_in_base_model = False
185                 kwargs = {
186                     "r": self.peft_config.r,
187                     "lora_alpha": self.peft_config.lora_alpha,
188                     "lora_dropout": self.peft_config.lora_dropout,
189                     "fan_in_fan_out": self.peft_config.fan_in_fan_out,
190                     "merge_weights": self.peft_config.merge_weights,
191                 }
192                 key_list = [key for key, _ in self.model.named_modules()]
193                 for key in key_list:  tuning을 설정한 파라미터 모두 수행
194                     if any(key.endswith(target_key) for target_key in self.peft_config.target_modules):
195                         if not is_target_modules_in_base_model:
196                             is_target_modules_in_base_model = True
197                             parent, target, target_name = self._get_submodules(key)
198                             bias = target.bias is not None
199                             if isinstance(target, bnb.nn.Linear8bitLt) and self.peft_config.enable_lora is None:
200                                 kwargs.update(
201                                     {
202                                         "has_fp16_weights": target.state.has_fp16_weights,
203                                         "memory_efficient_backward": target.state.memory_efficient_backward,
204                                         "threshold": target.state.threshold,
205                                         "index": target.index,
206                                         }
207                                 )
208                             new_module = Linear8bitLt(target.in_features, target.out_features, bias=bias, **kwargs)
209                         elif isinstance(target, torch.nn.Linear) and self.peft_config.enable_lora is None:
210                             new_module = Linear(target.in_features, target.out_features, bias=bias, **kwargs)
211                         elif self.peft_config.enable_lora is not None:
212                             kwargs.update({"enable_lora": self.peft_config.enable_lora})
213                             if isinstance(target, Conv1D):
214                                 in_features, out_features = target.weight.shape
215                             else:
216                                 in_features, out_features = target.in_features, target.out_features
217                             if kwargs["fan_in_fan_out"]:
218                                 warnings.warn(
219                                     "fan_in_fan_out is set to True but the target module is not a Conv1D."
220                                     "Setting fan_in_fan_out to False."
221                                 )
222                             kwargs["fan_in_fan_out"] = False
223                             new_module = MergedLinear(in_features, out_features, bias=bias, **kwargs)
224                         self._replace_module(parent, target_name, new_module, target)
225
226             # Find and replace
227             self._find_and_replace()
228             mark_only_lora_as_trainable(self.model, self.peft_config.bias)
229             self.forward = self.model.forward
230             def _find_and_replace(self):
231                 is_target_modules_in_base_model = False
232                 kwargs = {
233                     "r": self.peft_config.r,
234                     "lora_alpha": self.peft_config.lora_alpha,
235                     "lora_dropout": self.peft_config.lora_dropout,
236                     "fan_in_fan_out": self.peft_config.fan_in_fan_out,
237                     "merge_weights": self.peft_config.merge_weights,
238                 }
239                 key_list = [key for key, _ in self.model.named_modules()]
240                 for key in key_list:  tuning을 설정한 파라미터 모두 수행
241                     if any(key.endswith(target_key) for target_key in self.peft_config.target_modules):
242                         if not is_target_modules_in_base_model:
243                             is_target_modules_in_base_model = True
244                             parent, target, target_name = self._get_submodules(key)
245                             bias = target.bias is not None
246                             if isinstance(target, bnb.nn.Linear8bitLt) and self.peft_config.enable_lora is None:
247                                 kwargs.update(
248                                     {
249                                         "has_fp16_weights": target.state.has_fp16_weights,
250                                         "memory_efficient_backward": target.state.memory_efficient_backward,
251                                         "threshold": target.state.threshold,
252                                         "index": target.index,
253                                         }
254                                 )
255                             new_module = Linear8bitLt(target.in_features, target.out_features, bias=bias, **kwargs)
256                         elif isinstance(target, torch.nn.Linear) and self.peft_config.enable_lora is None:
257                             new_module = Linear(target.in_features, target.out_features, bias=bias, **kwargs)
258                         elif self.peft_config.enable_lora is not None:
259                             kwargs.update({"enable_lora": self.peft_config.enable_lora})
260                             if isinstance(target, Conv1D):
261                                 in_features, out_features = target.weight.shape
262                             else:
263                                 in_features, out_features = target.in_features, target.out_features
264                             if kwargs["fan_in_fan_out"]:
265                                 warnings.warn(
266                                     "fan_in_fan_out is set to True but the target module is not a Conv1D."
267                                     "Setting fan_in_fan_out to False."
268                                 )
269                             kwargs["fan_in_fan_out"] = False
270                             new_module = MergedLinear(in_features, out_features, bias=bias, **kwargs)
271                         self._replace_module(parent, target_name, new_module, target)
272
273             # Find and replace
274             self._find_and_replace()
275             mark_only_lora_as_trainable(self.model, self.peft_config.bias)
276             self.forward = self.model.forward
277             def _find_and_replace(self):
278                 is_target_modules_in_base_model = False
279                 kwargs = {
280                     "r": self.peft_config.r,
281                     "lora_alpha": self.peft_config.lora_alpha,
282                     "lora_dropout": self.peft_config.lora_dropout,
283                     "fan_in_fan_out": self.peft_config.fan_in_fan_out,
284                     "merge_weights": self.peft_config.merge_weights,
285                 }
286                 key_list = [key for key, _ in self.model.named_modules()]
287                 for key in key_list:  tuning을 설정한 파라미터 모두 수행
288                     if any(key.endswith(target_key) for target_key in self.peft_config.target_modules):
289                         if not is_target_modules_in_base_model:
290                             is_target_modules_in_base_model = True
291                             parent, target, target_name = self._get_submodules(key)
292                             bias = target.bias is not None
293                             if isinstance(target, bnb.nn.Linear8bitLt) and self.peft_config.enable_lora is None:
294                                 kwargs.update(
295                                     {
296                                         "has_fp16_weights": target.state.has_fp16_weights,
297                                         "memory_efficient_backward": target.state.memory_efficient_backward,
298                                         "threshold": target.state.threshold,
299                                         "index": target.index,
300                                         }
301                                 )
302                             new_module = Linear8bitLt(target.in_features, target.out_features, bias=bias, **kwargs)
303                         elif isinstance(target, torch.nn.Linear) and self.peft_config.enable_lora is None:
304                             new_module = Linear(target.in_features, target.out_features, bias=bias, **kwargs)
305                         elif self.peft_config.enable_lora is not None:
306                             kwargs.update({"enable_lora": self.peft_config.enable_lora})
307                             if isinstance(target, Conv1D):
308                                 in_features, out_features = target.weight.shape
309                             else:
310                                 in_features, out_features = target.in_features, target.out_features
311                             if kwargs["fan_in_fan_out"]:
312                                 warnings.warn(
313                                     "fan_in_fan_out is set to True but the target module is not a Conv1D."
314                                     "Setting fan_in_fan_out to False."
315                                 )
316                             kwargs["fan_in_fan_out"] = False
317                             new_module = MergedLinear(in_features, out_features, bias=bias, **kwargs)
318                         self._replace_module(parent, target_name, new_module, target)
319
320             # Find and replace
321             self._find_and_replace()
322             mark_only_lora_as_trainable(self.model, self.peft_config.bias)
323             self.forward = self.model.forward
324             def _find_and_replace(self):
325                 is_target_modules_in_base_model = False
326                 kwargs = {
327                     "r": self.peft_config.r,
328                     "lora_alpha": self.peft_config.lora_alpha,
329                     "lora_dropout": self.peft_config.lora_dropout,
330                     "fan_in_fan_out": self.peft_config.fan_in_fan_out,
331                     "merge_weights": self.peft_config.merge_weights,
332                 }
333                 key_list = [key for key, _ in self.model.named_modules()]
334                 for key in key_list:  tuning을 설정한 파라미터 모두 수행
335                     if any(key.endswith(target_key) for target_key in self.peft_config.target_modules):
336                         if not is_target_modules_in_base_model:
337                             is_target_modules_in_base_model = True
338                             parent, target, target_name = self._get_submodules(key)
339                             bias = target.bias is not None
340                             if isinstance(target, bnb.nn.Linear8bitLt) and self.peft_config.enable_lora is None:
341                                 kwargs.update(
342                                     {
343                                         "has_fp16_weights": target.state.has_fp16_weights,
344                                         "memory_efficient_backward": target.state.memory_efficient_backward,
345                                         "threshold": target.state.threshold,
346                                         "index": target.index,
347                                         }
348                                 )
349                             new_module = Linear8bitLt(target.in_features, target.out_features, bias=bias, **kwargs)
350                         elif isinstance(target, torch.nn.Linear) and self.peft_config.enable_lora is None:
351                             new_module = Linear(target.in_features, target.out_features, bias=bias, **kwargs)
352                         elif self.peft_config.enable_lora is not None:
353                             kwargs.update({"enable_lora": self.peft_config.enable_lora})
354                             if isinstance(target, Conv1D):
355                                 in_features, out_features = target.weight.shape
356                             else:
357                                 in_features, out_features = target.in_features, target.out_features
358                             if kwargs["fan_in_fan_out"]:
359                                 warnings.warn(
360                                     "fan_in_fan_out is set to True but the target module is not a Conv1D."
361                                     "Setting fan_in_fan_out to False."
362                                 )
363                             kwargs["fan_in_fan_out"] = False
364                             new_module = MergedLinear(in_features, out_features, bias=bias, **kwargs)
365                         self._replace_module(parent, target_name, new_module, target)
366
367             # Find and replace
368             self._find_and_replace()
369             mark_only_lora_as_trainable(self.model, self.peft_config.bias)
370             self.forward = self.model.forward
371             def _find_and_replace(self):
372                 is_target_modules_in_base_model = False
373                 kwargs = {
374                     "r": self.peft_config.r,
375                     "lora_alpha": self.peft_config.lora_alpha,
376                     "lora_dropout": self.peft_config.lora_dropout,
377                     "fan_in_fan_out": self.peft_config.fan_in_fan_out,
378                     "merge_weights": self.peft_config.merge_weights,
379                 }
380                 key_list = [key for key, _ in self.model.named_modules()]
381                 for key in key_list:  tuning을 설정한 파라미터 모두 수행
382                     if any(key.endswith(target_key) for target_key in self.peft_config.target_modules):
383                         if not is_target_modules_in_base_model:
384                             is_target_modules_in_base_model = True
385                             parent, target, target_name = self._get_submodules(key)
386                             bias = target.bias is not None
387                             if isinstance(target, bnb.nn.Linear8bitLt) and self.peft_config.enable_lora is None:
388                                 kwargs.update(
389                                     {
390                                         "has_fp16_weights": target.state.has_fp16_weights,
391                                         "memory_efficient_backward": target.state.memory_efficient_backward,
392                                         "threshold": target.state.threshold,
393                                         "index": target.index,
394                                         }
395                                 )
396                             new_module = Linear8bitLt(target.in_features, target.out_features, bias=bias, **kwargs)
397                         elif isinstance(target, torch.nn.Linear) and self.peft_config.enable_lora is None:
398                             new_module = Linear(target.in_features, target.out_features, bias=bias, **kwargs)
399                         elif self.peft_config.enable_lora is not None:
400                             kwargs.update({"enable_lora": self.peft_config.enable_lora})
401                             if isinstance(target, Conv1D):
402                                 in_features, out_features = target.weight.shape
403                             else:
404                                 in_features, out_features = target.in_features, target.out_features
405                             if kwargs["fan_in_fan_out"]:
406                                 warnings.warn(
407                                     "fan_in_fan_out is set to True but the target module is not a Conv1D."
408                                     "Setting fan_in_fan_out to False."
409                                 )
410                             kwargs["fan_in_fan_out"] = False
411                             new_module = MergedLinear(in_features, out_features, bias=bias, **kwargs)
412                         self._replace_module(parent, target_name, new_module, target)
413
414             # Find and replace
415             self._find_and_replace()
416             mark_only_lora_as_trainable(self.model, self.peft_config.bias)
417             self.forward = self.model.forward
418             def _find_and_replace(self):
419                 is_target_modules_in_base_model = False
420                 kwargs = {
421                     "r": self.peft_config.r,
422                     "lora_alpha": self.peft_config.lora_alpha,
423                     "lora_dropout": self.peft_config.lora_dropout,
424                     "fan_in_fan_out": self.peft_config.fan_in_fan_out,
425                     "merge_weights": self.peft_config.merge_weights,
426                 }
427                 key_list = [key for key, _ in self.model.named_modules()]
428                 for key in key_list:  tuning을 설정한 파라미터 모두 수행
429                     if any(key.endswith(target_key) for target_key in self.peft_config.target_modules):
430                         if not is_target_modules_in_base_model:
431                             is_target_modules_in_base_model = True
432                             parent, target, target_name = self._get_submodules(key)
433                             bias = target.bias is not None
434                             if isinstance(target, bnb.nn.Linear8bitLt) and self.peft_config.enable_lora is None:
435                                 kwargs.update(
436                                     {
437                                         "has_fp16_weights": target.state.has_fp16_weights,
438                                         "memory_efficient_backward": target.state.memory_efficient_backward,
439                                         "threshold": target.state.threshold,
440                                         "index": target.index,
441                                         }
442                                 )
443                             new_module = Linear8bitLt(target.in_features, target.out_features, bias=bias, **kwargs)
444                         elif isinstance(target, torch.nn.Linear) and self.peft_config.enable_lora is None:
445                             new_module = Linear(target.in_features, target.out_features, bias=bias, **kwargs)
446                         elif self.peft_config.enable_lora is not None:
447                             kwargs.update({"enable_lora": self.peft_config.enable_lora})
448                             if isinstance(target, Conv1D):
449                                 in_features, out_features = target.weight.shape
450                             else:
451                                 in_features, out_features = target.in_features, target.out_features
452                             if kwargs["fan_in_fan_out"]:
453                                 warnings.warn(
454                                     "fan_in_fan_out is set to True but the target module is not a Conv1D."
455                                     "Setting fan_in_fan_out to False."
456                                 )
457                             kwargs["fan_in_fan_out"] = False
458                             new_module = MergedLinear(in_features, out_features, bias=bias, **kwargs)
459                         self._replace_module(parent, target_name, new_module, target)
460
461             # Find and replace
462             self._find_and_replace()
463             mark_only_lora_as_trainable(self.model, self.peft_config.bias)
464             self.forward = self.model.forward
465             def _find_and_replace(self):
466                 is_target_modules_in_base_model = False
467                 kwargs = {
468                     "r": self.peft_config.r,
469                     "lora_alpha": self.peft_config.lora_alpha,
470                     "lora_dropout": self.peft_config.lora_dropout,
471                     "fan_in_fan_out": self.peft_config.fan_in_fan_out,
472                     "merge_weights": self.peft_config.merge_weights,
473                 }
474                 key_list = [key for key, _ in self.model.named_modules()]
475                 for key in key_list:  tuning을 설정한 파라미터 모두 수행
476                     if any(key.endswith(target_key) for target_key in self.peft_config.target_modules):
477                         if not is_target_modules_in_base_model:
478                             is_target_modules_in_base_model = True
479                             parent, target, target_name = self._get_submodules(key)
480                             bias = target.bias is not None
481                             if isinstance(target, bnb.nn.Linear8bitLt) and self.peft_config.enable_lora is None:
482                                 kwargs.update(
483                                     {
484                                         "has_fp16_weights": target.state.has_fp16_weights,
485                                         "memory_efficient_backward": target.state.memory_efficient_backward,
486                                         "threshold": target.state.threshold,
487                                         "index": target.index,
488                                         }
489                                 )
490                             new_module = Linear8bitLt(target.in_features, target.out_features, bias=bias, **kwargs)
491                         elif isinstance(target, torch.nn.Linear) and self.peft_config.enable_lora is None:
492                             new_module = Linear(target.in_features, target.out_features, bias=bias, **kwargs)
493                         elif self.peft_config.enable_lora is not None:
494                             kwargs.update({"enable_lora": self.peft_config.enable_lora})
495                             if isinstance(target, Conv1D):
496                                 in_features, out_features = target.weight.shape
497                             else:
498                                 in_features, out_features = target.in_features, target.out_features
499                             if kwargs["fan_in_fan_out"]:
500                                 warnings.warn(
501                                     "fan_in_fan_out is set to True but the target module is not a Conv1D."
502                                     "Setting fan_in_fan_out to False."
503                                 )
504                             kwargs["fan_in_fan_out"] = False
505                             new_module = MergedLinear(in_features, out_features, bias=bias, **kwargs)
506                         self._replace_module(parent, target_name, new_module, target)
507
508             # Find and replace
509             self._find_and_replace()
510             mark_only_lora_as_trainable(self.model, self.peft_config.bias)
511             self.forward = self.model.forward
512             def _find_and_replace(self):
513                 is_target_modules_in_base_model = False
514                 kwargs = {
515                     "r": self.peft_config.r,
516                     "lora_alpha": self.peft_config.lora_alpha,
517                     "lora_dropout": self.peft_config.lora_dropout,
518                     "fan_in_fan_out": self.peft_config.fan_in_fan_out,
519                     "merge_weights": self.peft_config.merge_weights,
520                 }
521                 key_list = [key for key, _ in self.model.named_modules()]
522                 for key in key_list:  tuning을 설정한 파라미터 모두 수행
523                     if any(key.endswith(target_key) for target_key in self.peft_config.target_modules):
524                         if not is_target_modules_in_base_model:
525                             is_target_modules_in_base_model = True
526                             parent, target, target_name = self._get_submodules(key)
527                             bias = target.bias is not None
528                             if isinstance(target, bnb.nn.Linear8bitLt) and self.peft_config.enable_lora is None:
529                                 kwargs.update(
530                                     {
531                                         "has_fp16_weights": target.state.has_fp16_weights,
532                                         "memory_efficient_backward": target.state.memory_efficient_backward,
533                                         "threshold": target.state.threshold,
534                                         "index": target.index,
535                                         }
536                                 )
537                             new_module = Linear8bitLt(target.in_features, target.out_features, bias=bias, **kwargs)
538                         elif isinstance(target, torch.nn.Linear) and self.peft_config.enable_lora is None:
539                             new_module = Linear(target.in_features, target.out_features, bias=bias, **kwargs)
540                         elif self.peft_config.enable_lora is not None:
541                             kwargs.update({"enable_lora": self.peft_config.enable_lora})
542                             if isinstance(target, Conv1D):
543                                 in_features, out_features = target.weight.shape
544                             else:
545                                 in_features, out_features = target.in_features, target.out_features
546                             if kwargs["fan_in_fan_out"]:
547                                 warnings.warn(
548                                     "fan_in_fan_out is set to True but the target module is not a Conv1D."
549                                     "Setting fan_in_fan_out to False."
550                                 )
551                             kwargs["fan_in_fan_out"] = False
552                             new_module = MergedLinear(in_features, out_features, bias=bias, **kwargs)
553                         self._replace_module(parent, target_name, new_module, target)
554
555             # Find and replace
556             self._find_and_replace()
557             mark_only_lora_as_trainable(self.model, self.peft_config.bias)
558             self.forward = self.model.forward
559             def _find_and_replace(self):
560                 is_target_modules_in_base_model = False
561                 kwargs = {
562                     "r": self.peft_config.r,
563                     "lora_alpha": self.peft_config.lora_alpha,
564                     "lora_dropout": self.peft_config.lora_dropout,
565                     "fan_in_fan_out": self.peft_config.fan_in_fan_out,
566                     "merge_weights": self.peft_config.merge_weights,
567                 }
568                 key_list = [key for key, _ in self.model.named_modules()]
569                 for key in key_list:  tuning을 설정한 파라미터 모두 수행
570                     if any(key.endswith(target_key) for target_key in self.peft_config.target_modules):
571                         if not is_target_modules_in_base_model:
572                             is_target_modules_in_base_model = True
573                             parent, target, target_name = self._get_submodules(key)
574                             bias = target.bias is not None
575                             if isinstance(target, bnb.nn.Linear8bitLt) and self.peft_config.enable_lora is None:
576                                 kwargs.update(
5
```



감사합니다!

Flamingo | 권유진, 이선민, 임승섭

