Homework #3

Deep Learning for Computer Vision
NTU, Fall 2024

Outline

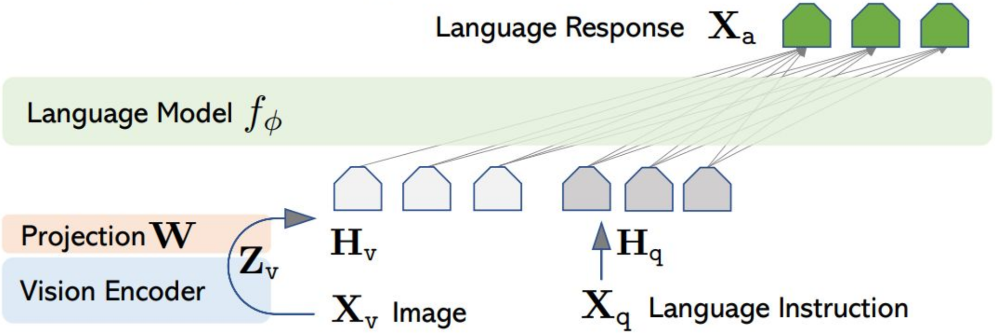
- Problems & Grading
- Dataset
- Submission & Rules
- Supplementary

Problems – Overview

- **Problem 1:** Zero-shot image captioning with LLaVA (24%)
  [hw3_data/p1_data]
- **Problem 2:** PEFT on Vision and Language Model for Image Captioning (50%)
  [hw3_data/p2_data]
- **Problem 3:** Visualization of Attention in Image Captioning (26%)
  [hw3_data/p3_data]
- Please refer to “Dataset” section for more details about p1_data/p2_data/p3_data.

Problem 1: Zero-shot Image Captioning with LLaVA

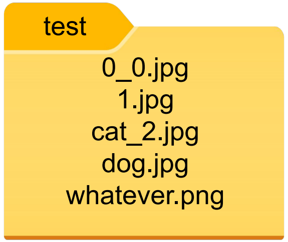
- Zero-shot: You **cannot** do any finetuning for the pretrained model.
- In this problem, you only need to evaluate the **pretrained** LLaVA on image captioning task.
  - Input: (1)image (2)language instruction (3)generation config
  - Output: caption
- Please use the model "llava-hf/llava-1.5-7b-hf" in the transformers package
- Goal: Choose a proper instruction and generation config to obtain better caption



Reference: Visual Instruction Tuning

Problem 1: Evaluation

- Evaluation metrics: CIDEr and CLIPScore (explained in next two page)
- Sample Output
    - Output format: json file with key and value being ‘filename’ and ‘predicted_caption’, respectively
    - Example:




```
{
    "0_0": "man in black shirt is playing gitar.",
    "1": "construction worker in orange safety vest is working.",
    "cat_2": "two young girls are playing with lego toy.",
    "dog": "a little girl in a pink hat is blowing bubbles.",
    "whatever": "two dogs play in the grass."
}
```

⚠️ **Please remove the filename extension**

Problem 1: Evaluation – CIDEr

- CIDEr is an quantitative metric to evaluate whether the candidate sentences (your prediction) have high consensus with reference sentences (ground truth).
    - Candidate sentence will have high consensus if it captures the more high tf-idf words (distinct and frequent) in reference sentences.

Reference Sentences

R1: A bald eagle sits on a perch.

R2: An american bald eagle sitting on a branch in the zoo.

R3: Bald eagle perched on piece of lumber.

...

R50: A large bird standing on a tree branch.

Candidate Sentences

C1: An eagle is perched among trees.

C2: A picture of a bald eagle on a rope stem.

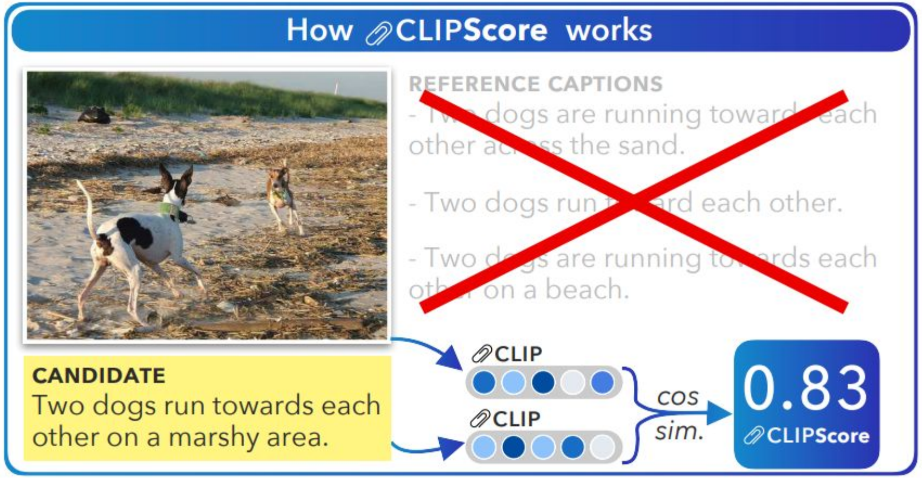
CIDEr scores: C1 < C2

Reference: [CIDEr: Consensus-based Image Description Evaluation]

Tookit: [coco evaluation]

Problem 1: Evaluation – CLIPScore

- CLIPScore uses CLIP to assess image-caption compatibility without any ground truth, just like humans.



Reference: [CLIPScore: A Reference-free Evaluation Metric for Image Captioning]()

Problem 1: Evaluation

- We provide evaluation code which calculates CIDEr and CLIPScore to check performance [evaluate.py]
- Arguments:
  - pred_file: your output json file ({filename: predicted_caption})
  - annotation_file: ground truth json file (e.g. “p1_data/val.json”)
  - images_root: path to the folder containing val images (e.g. “p1_data/images/val”)
- Output: CIDEr & CLIPScore

Problem 1: Grading – Baselines (15%)

- You only need to submit **your best setting**
- Public Baseline (10%) - 500 validation data ([p1_data/images/val])
  - Simple baseline (7%) - CIDEr of **1.00**, CLIPScore of **0.65**
  - Strong baseline (3%) - CIDEr of **1.14**, CLIPScore of **0.77**
- Private Baseline (5%) - 500 test data (not available for students)
  - Simple baseline (3%) - TBD
  - Strong baseline (2%) - TBD

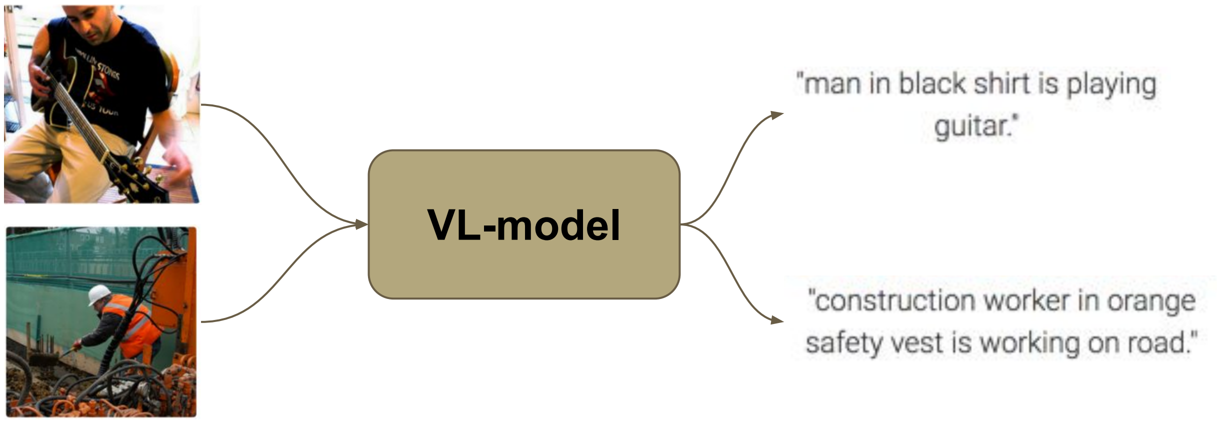
Problem 1: Grading - Report (9%)

1. Paper reading(3%)
   - Please read the paper “Visual Instruction Tuning” and briefly describe the important components (modules or techniques) of LLaVA.
2. Prompt-text analysis (6%)
   - Please come up with two settings (different instructions or generation config). Compare and discuss their performances.

Problem 2: PEFT on Vision and Language Model for Image Captioning

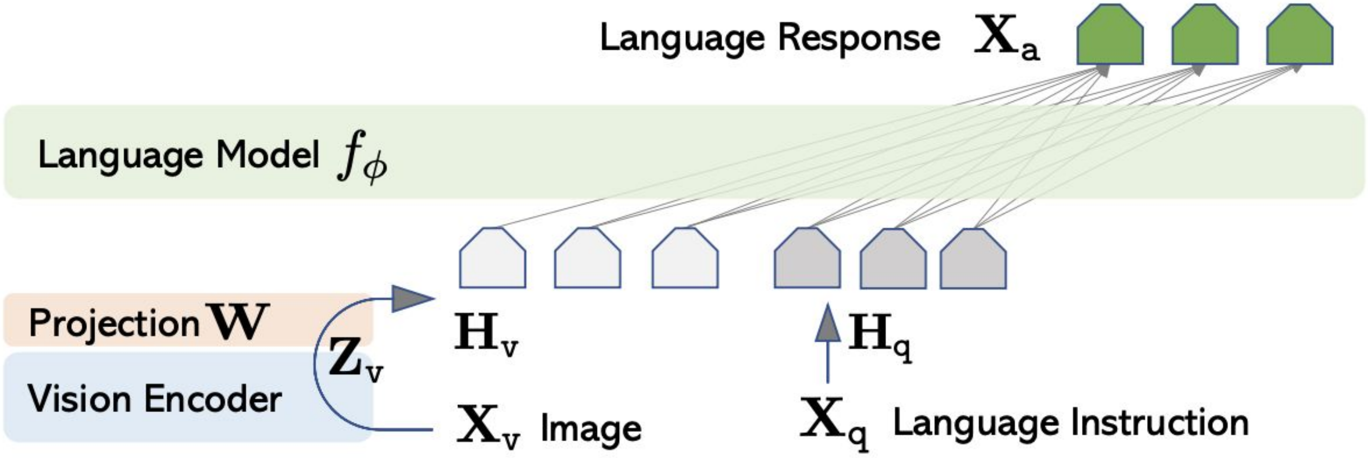
In this problem, you need to:

1. Use **pretrained visual encoder and text decoder** for image captioning.
    - Input: RGB image
    - Output: image caption (text)
2. Implement PEFT (Parameter-Efficient Fine-Tuning) for the task of image captioning (from scratch)



Problem 2: Model Overview

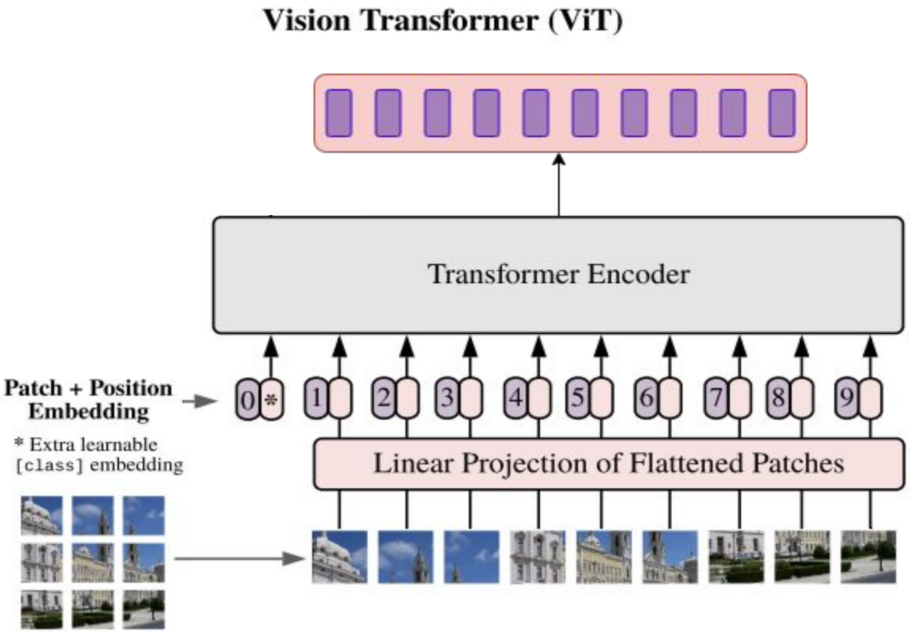
- You are limited to use **transformer encoder + decoder** architecture in this assignment.
  - Language model: pretrained transformer-base decoder



Reference: [Improved Baselines with Visual Instruction Tuning]()

Problem 2: Model Implementation

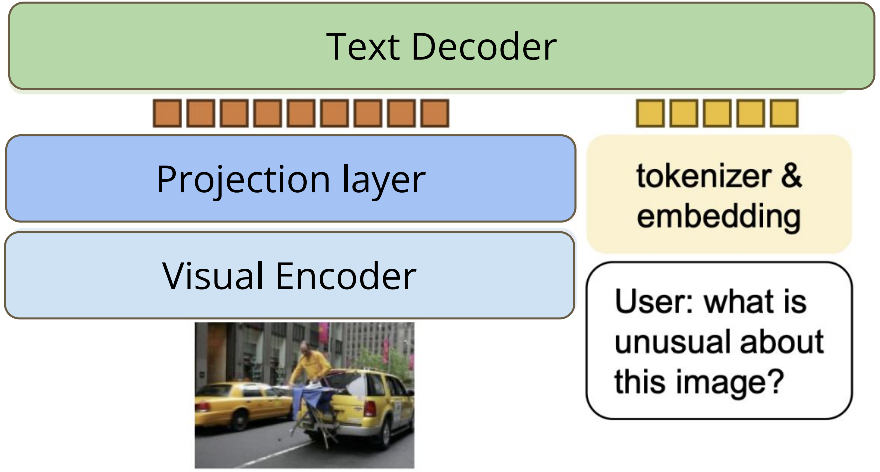
- You are limited to use vision transformer (ViT) as vision encoder. And, any **pretrained** ViTs (e.g. ViT-Base, CLIP-ViT-Base, ViT-Large, etc.) are allowed.
- You can use ViT from timm, openai/CLIP



Reference code: timm

Problem 2: Model Implementation

- As for text decoder, you are limited to use the pretrained decoder we provide. [decoder.py][p2_data/decoder_model.bin]
- You need to **modify decoder** to generate captions based on visual features. (e.g. adjust input, concat visual token with text token)



Problem 2: Tokenizer

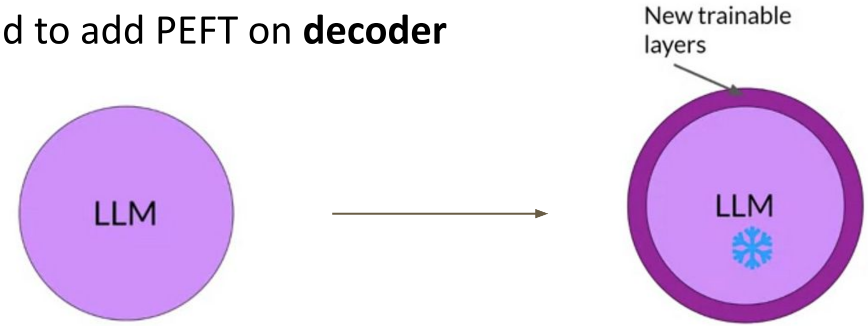
- Since the text-decoder generate the caption autoregresively, you should first use the tokenizer to tokenize the caption in data pre-processing.
- Due to pretrained language decoder, you need to use the tokenizer we provide:
  - encoder.json, vocab.bpe
  - class **BPETokenizer** in tokenizer.py
- The **start token** and **end token** should be “<|endoftext|>”.

```
encoding = BPETokenizer()
prompt = 'a kitchen with a sink and many cooking machines and a pot of food'

context = encoding.encode(prompt)
print(context)
print(encoding.decode(context))
✓ 0.0s
[64, 9592, 351, 257, 14595, 290, 867, 10801, 8217, 290, 257, 1787, 286, 2057]
a kitchen with a sink and many cooking machines and a pot of food
```

Problem 2: PEFT (Parameter-Efficient Fine-Tuning)

- Avoid extremely high computational cost for finetuning LLM (Large Language Model)
- The number of parameters to be trained is limited.
- **DO NOT** directly use Hugging Face (transformers), but you can reference the code.
- In this problem, you have to implement **LoRA**
- You only need to add PEFT on **decoder**



Finetune the whole LLM

Finetune the whole LLM

Reference: Introduction of PEFT
Reference code: PEFT

Problem 2: PEFT

Lora

- train LLM with low-rank decomposition
- you can implement by **loralib:** https://pypi.org/project/loralib/
- see more implement details from its GitHub: https://github.com/microsoft/LoRA

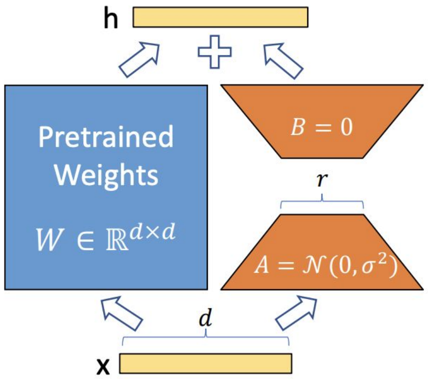
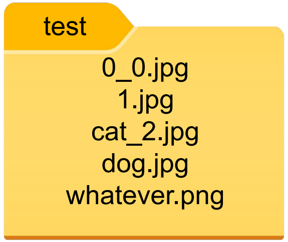


Figure 1: Our reparametrization. We only train A and B.

Reference: https://arxiv.org/pdf/2106.09685.pdf

Problem 2: Evaluation

- Evaluation metrics: CIDEr and CLIPScore (same as problem 1)
- Sample Output
  - Output format: json file with key and value being ‘filename’ and ‘predicted_caption’, respectively
  - Example:




```
{
    "0_0": "man in black shirt is playing gitar.",
    "1": "construction worker in orange safety vest is working.",
    "cat_2": "two young girls are playing with lego toy.",
    "dog": "a little girl in a pink hat is blowing bubbles.",
    "whatever": "two dogs play in the grass."
}
```

Please remove the filename extension

Problem 2: Evaluation

- We provide evaluation code which calculates CIDEr and CLIPScore to check performance [evaluate.py]
- Arguments:
  - pred_file: your output json file ({filename: predicted_caption})
  - annotation_file: ground truth json file (e.g. “p2_data/val.json”)
  - images_root: path to the folder containing val images (e.g. “p2_data/images/val”)
- Output: CIDEr & CLIPScore

Problem 2: Grading – Baselines (40%)

- You only need to submit **your best setting**
- Public Baseline (20%) - 2000 validation data ([p2_data/images/val])
  - Simple baseline (13%) - CIDEr of **0.8**, CLIPScore of **0.69**
  - Strong baseline (7%) - CIDEr of **0.94**, CLIPScore of **0.73**
- Private Baseline (20%) - 2000 test data (not available for students)
  - Simple baseline (13%) - TBD
  - Strong baseline (7%) - TBD
- **The total number of trained parameters: < 10M**
  - You are only allowed to save the trained parameters, model over 10M **will not be graded**
  - In inference stage, load your checkpoints with “strict=False” since only trained parameters are saved (you can refer to decoder.py)

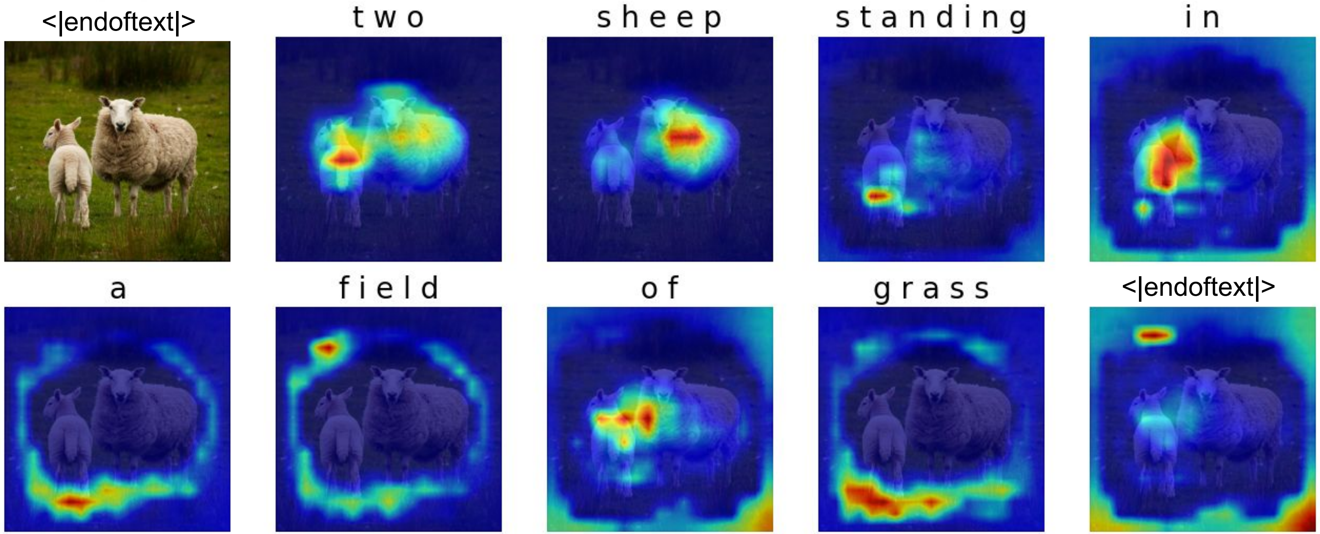
```
print("Total params:", sum(p.numel() for p in model.parameters() if p.requires_grad))
```

Problem 2: Evaluation metrics report (10%)

1. Report your best setting and its corresponding CIDEr & CLIPScore on the validation data. Briefly introduce your method. (TA will reproduce this result) (5%)
2. Report 2 different attempts of LoRA setting (e.g. initialization, alpha, rank…) and their corresponding CIDEr & CLIPScore. (5%, each setting for 2.5%)

Problem 3:
Visualization of Attention in Image Captioning (26%)

- In this problem, you have to analyze your image captioning model **in problem 1 and 2** by visualizing the **self-attention** between images tokens and generated captions' tokens.



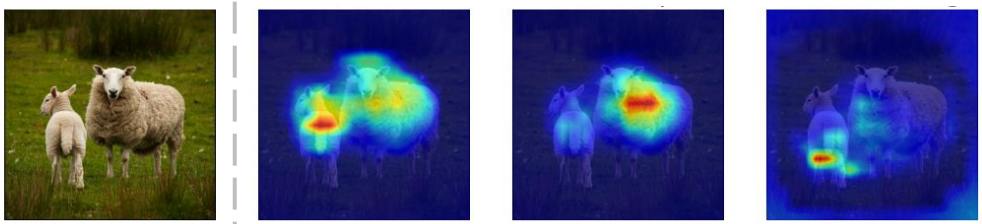
Problem 3: Models and Settings

- use your **best model** in problem 2
- Given an input image, your model would be able to generate a corresponding caption sequentially, and you have to visualize the self-attention between the image patches and each predicted word in your own caption.

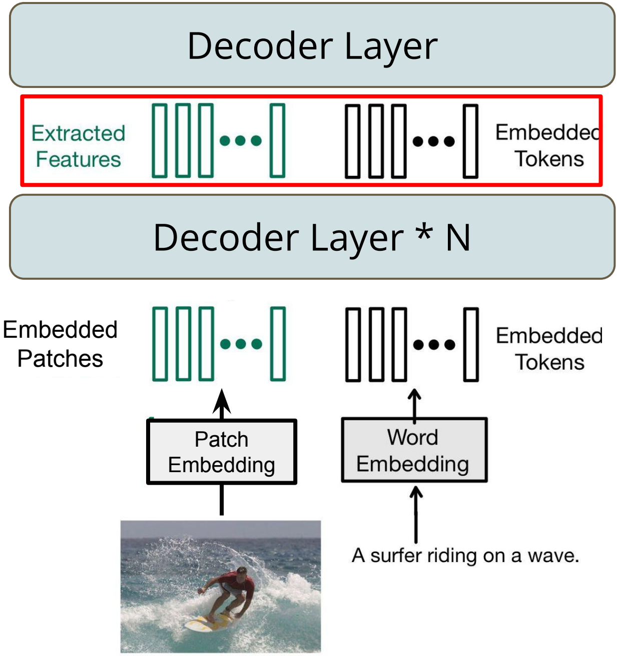
Input: <start token> two sheep

Output: two → sheep → standing → ...

\+ \+ \+



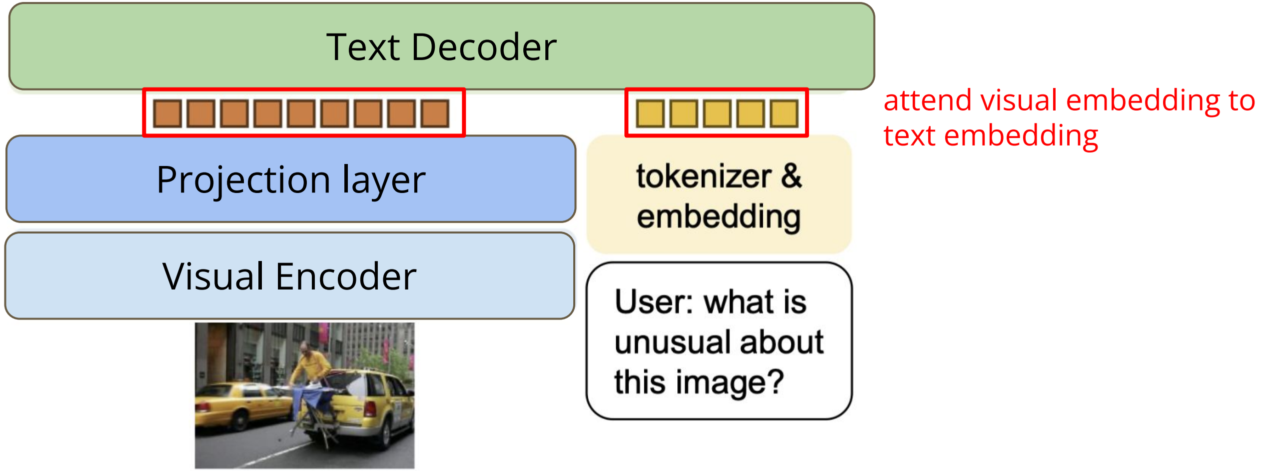
t=1 t=2 t=3



Problem 3: Models and Settings

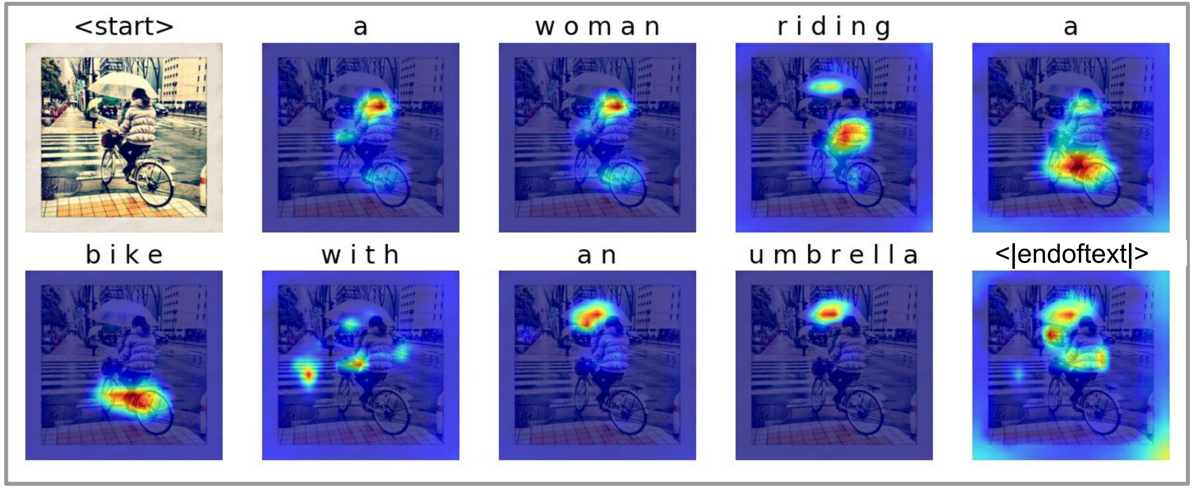
- The attention weights you have to visualize are in the **self attention block** in decoder
  - Self-attention weights in the last decoder layer (or the second to last) have better visulization results.

- **You should modify the code in problem 2 to get the attention weights for visualization.**



Problem 3:
Visualization of Attention in Image Captioning (26%) (cont'd)

1. Given five test images ([p3_data/images/]), and please visualize the **predicted caption** and the corresponding series of **attention maps** in your report with the following template: (20%, each image for 2%, **you need to visualize 5 images for both problem 1 & 2**)



In your report, a visualization example for bike.jpg should be like above.

Hint: You can use the "output_attentions" arguments for problem 1

Problem 3:
Visualization of Attention in Image Captioning (26%) (cont'd)

2. According to **CLIPScore**, you need to:
    i. visualize top-1 and last-1 image-caption pairs
    ii. report its corresponding CLIPScore

   in the validation dataset of problem 2. (3%)

2. Analyze the predicted captions and the attention maps for each word according to the previous question. Is the caption reasonable? Does the attended region reflect the corresponding word in the caption? (3%)

Outline

- Problems & Grading
- Dataset
- Submission & Rules
- Supplementary

Tools for Dataset

- Download the dataset
  - (Option 1) Manually download the dataset

    https://drive.google.com/file/d/11WqMRxzHcVqvjcbLt61g9Lt1i0fOcgp7/view?usp=sharing
  - (Option 2) Run the bash script provided in the hw3 repository

    $ bash get_dataset.sh

Problem 1: Dataset

- The dataset consists of images with each of them contains multiple captions.
- Format

```
p1_data/
├── images/
│   └── val/        # 500 images for validation
└── val.json        # captions for each val image
```

```json
{
    "annotations": [
        {
            "caption": "A man in a baseball game running to base and others trying to tag him out.",
            "image_id": 0
        },
        {
            "caption": "A box with half a dozen glazed and frosted donuts.",
            "image_id": 1
        }
    ],
    "images": [
        {
            "file_name": "000000000000.jpg",
            "id": 0
        },
        {
            "file_name": "000000000001.jpg",
            "id": 1
        },
        {
```

A brief example for val.json

Problem 2: Dataset

- The dataset consists of images with each of them contains multiple captions.
- Format

```
p2_data/
├── images/
│   ├── train/   # 10000 images for training
│   └── val/     # 2000 images for validation
├── train.json   # captions for each training image
└── val.json     # captions for each val image
```

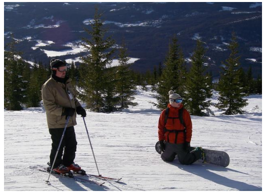

```
{
    "annotations": [
        {
            "caption": "A man in a baseball game running to base and others trying to tag him out.",
            "image_id": 0
        },
        {
            "caption": "A box with half a dozen glazed and frosted donuts.",
            "image_id": 1
        }
    ],
    "images": [
        {
            "file_name": "000000000000.jpg",
            "id": 0
        },
        {
            "file_name": "000000000001.jpg",
            "id": 1
        },
        {
```


A brief example for val.json

Visualization of Attention in Image Captioning - MS COCO 2017

- COCO (Common Objects in Context) is a large-scale object detection, segmentation, and captioning dataset. Each image has five captions.
- We provide **five** images (from MS COCO 2017 Explore) for captioning with these five images.
- These five test images and as below:
  - **p3_data/images/**
    - **bike.jpg**
    - **girl.jpg**
    - **sheep.jpg**
    - **ski.jpg**
    - **umbrella.jpg**




Outline

- Problems & Grading
- Dataset
- Submission & Rules
- Supplementary

Submission

- Click the following link to get your submission repository with your GitHub account:

  https://classroom.github.com/a/OlMw7_sq

  - You should connect your Github account to the classroom with your **student ID**
  - If you cannot find your student ID in the list, please contact us (ntudlcv@gmail.com)
- By default, we will grade your last submission (commit) before the deadline (**NOT** your last submission). Please e-mail the TAs if you'd like to submit another version of your repository and let us know which commit to grade.
- We will clone the **main** branch of your repository.

Submission

- Your GitHub repository should include the following files
  - hw3_<studentID>.pdf (report)
  - hw3_1.sh (for Problem 1)
  - hw3_2.sh (for Problem 2)
  - Python files (e.g., training code & inference code & visualization code)
  - Model files (can be loaded by your python file)
- No need to upload evaluate.py
- Visualization code for problem 3 can be .ipynb file
- **<u>Don’t push the dataset to your repo.</u>**
- If any of the file format is wrong, you will get zero point.

Bash Script - Problem 1

- TA will run your code as shown below
  - bash hw3_1.sh $1 $2
    - $1: path to the **folder** containing test images (e.g. hw3/p1_data/images/test/)
    - $2: path to the output **json file** (e.g. hw3/output_p1/pred.json)
- Please follow the naming rules in **p.5**
- Note that you should **NOT** hard code any path in your file or script.
- Your testing code have to be finished in **30 mins.**

Bash Script - Problem 2

- TA will run your code as shown below
  - bash hw3_2.sh $1 $2 $3
    - $1: path to the **folder** containing test images (e.g. hw3/p2_data/images/test/)
    - $2: path to the output **json file** (e.g. hw3/output_p2/pred.json)
    - $3: path to the **decoder weights** (e.g. hw3/p2_data/decoder_model.bin)
      (This means that you don't need to upload decoder_model.bin)
- Please follow the naming rules in **p.18**
- Note that you should **NOT** hard code any path in your file or script.
- Your testing code have to be finished in **40 mins.**

Bash Script (cont'd)

- You must not use commands such as **rm**, **sudo**, **CUDA_VISIBLE_DEVICES**, **cp**, **mv**, **mkdir**, **cd**, **pip** or other commands to change the Linux environment.
- In your submitted script, please use the command **python3** to execute your testing python files.
    - For example: python3 test.py $1 $2
- We will execute you code on **Linux** system, so try to make sure you code can be executed on Linux system before submitting your homework.

Rules – Submission

- If your model checkpoints are larger than GitHub's maximum capacity (50 MB), you could download and preprocess (e.g. unzip, tar zxf, etc.) them in hw3_download.sh.
    - TAs will run `bash hw3_download.sh` prior to any inference if the download script exists, i.e. it is **NOT** necessary to create a blank `hw3_download.sh` file.
    - We recommend you first download all the pretrained models in the `hw3_download.sh` (e.g python -c "import clip; clip.load('ViT-B/32')") to avoid the time limited issue.
- Do **NOT** delete your model checkpoints before the TAs release your score and before you have ensured that your score is correct.
- Your download script have to be finished in **20 mins.**

Rules – Submission

- Please use **wget** to download the model checkpoints from cloud drive (e.g. Dropbox) or your working station.
  - You should use **-O argument** to specify the filename of the downloaded checkpoint.
- Please refer to this Dropbox Guide for a detailed tutorial.
- Google Drive is a widely used cloud drive, so it is allowed to use **gdown** to download your checkpoints from your drive.
  - It is also recommended to use -O argument to specify the filename.
  - Remember to set the permission visible to public, otherwise TAs are unable to grade your submission, resulting in zero point.
  - If you have set the permission correspondingly but failed to download with **gdown** because of Google's policy, TAs will manually download them, no worries!!

Rules – Environment

- Ubuntu 22.04.4 LTS
- NVIDIA RTX A4500(20 GB)
- GNU bash, version 5.1.16(1)-release
- Python 3.8.5

Rules – Environment

- Ensure your code can be executed successfully on **Linux** system before your submission.
- Use only **Python3** and **Bash** script conforming to our environment, do not use other languages (e.g. CUDA) and other shell (e.g. zsh, fish) during inference.
  - Use the command **"python3"** to execute your testing python files.
- You must **NOT** use commands such as **sudo, CUDA_VISIBLE_DEVICES** or other commands to interfere with the environment; **any malicious attempt against the environment will lead to zero point in this assignment**.
- You shall **NOT** hardcode any path in your python files or scripts, while the dataset given would be the absolute path to the directory.

Packages - Problem 1

- torch==1.13.1
- torchvision==0.14.1
- bitsandbytes==0.44.1
- accelerate==1.0.1
- transformers==4.45.2
- language-evaluation、clip (please follow these github repos to install the packages)

- **E-mail or ask TA first if you want to import other packages.**

Packages - Problem 2

- imageio==2.21.3
- matplotlib==3.6.1
- numpy==1.23.4
- Pillow==9.2.0
- scipy==1.9.1
- opencv-python==4.6.0.66
- loralib==0.1.2
- pycocotools==2.0.5
- timm==0.9.10
- torch==1.12.1
- torchvision==0.13.1
- pandas==1.5.1
- tqdm, gdown, glob, yaml, skimage
- language-evaluation、clip (please follow these github repos to install the packages)
- Any dependencies of above packages, and other standard python packages

- **TAs will run the P1 and P2 in two different environments**
- **E-mail or ask TA first if you want to import other packages**

Packages and Reminders

- Python==3.8
- Do not use **imshow()** or **show()** in your code or your code will crash.
- Use **os.path.join** to deal with path as often as possible.
- If you train on GPU ids other than 0, remember to deal with the “**map location**” issue when you load model. (More details about this issue, please refer to https://github.com/pytorch/pytorch/issues/15541)

Deadline and Academic Honesty

- Deadline: **113/11/19 (Tue.) 11:59 PM (GMT+8)**
- Late policy : Up to 3 free late days in a semester. After that, late homework will be deducted 30% each day.
- Taking any unfair advantages over other class members (or letting anyone do so) is strictly prohibited. Violating university policy would result in F for this course.
- Students are encouraged to discuss the homework assignments, but you must complete the assignment by yourself. TA will compare the similarity of everyone’s homework. Any form of cheating or plagiarism will not be tolerated, which will also result in F for students with such misconduct.

Penalty

- If we cannot execute your code, TAs will give you a chance to make minor modifications to your code. After you modify your code,
  - If we can execute your code, you will still receive a 30% penalty in your model performance score.
  - If we still cannot execute your code, no point will be given.

Reminder

- Please start working on this homework as early as possible.
- The training may take hours on a GPU or days on CPUs.
- Please read and follow the HW rules carefully.
- If not sure, please ask your TAs!

How to Find Help

- Google or ChatGPT!
- Use TA hours (please check course website for time/location).
  - Please seek help from **the TAs in charge of this assignment as possible as you can**.
  - Fri. or Tue. 14:20～15:10 in MK-514
- Post your question under HW3 discussion section on NTU COOL.
- Contact TAs by e-mail: ntudlcv@gmail.com.

DOs and DONTs for the TAs (& Instructor)

- Do NOT send private messages to TAs via Facebook.
- TAs are happy to help, but they are not your tutors 24/7.
- TAs will NOT debug for you, including addressing coding, environmental, library dependency problems.
- TAs do NOT answer questions not related to the course.
- If you cannot make the TA hours, please email the TAs to schedule an appointment instead of stopping by the lab directly.

Outline

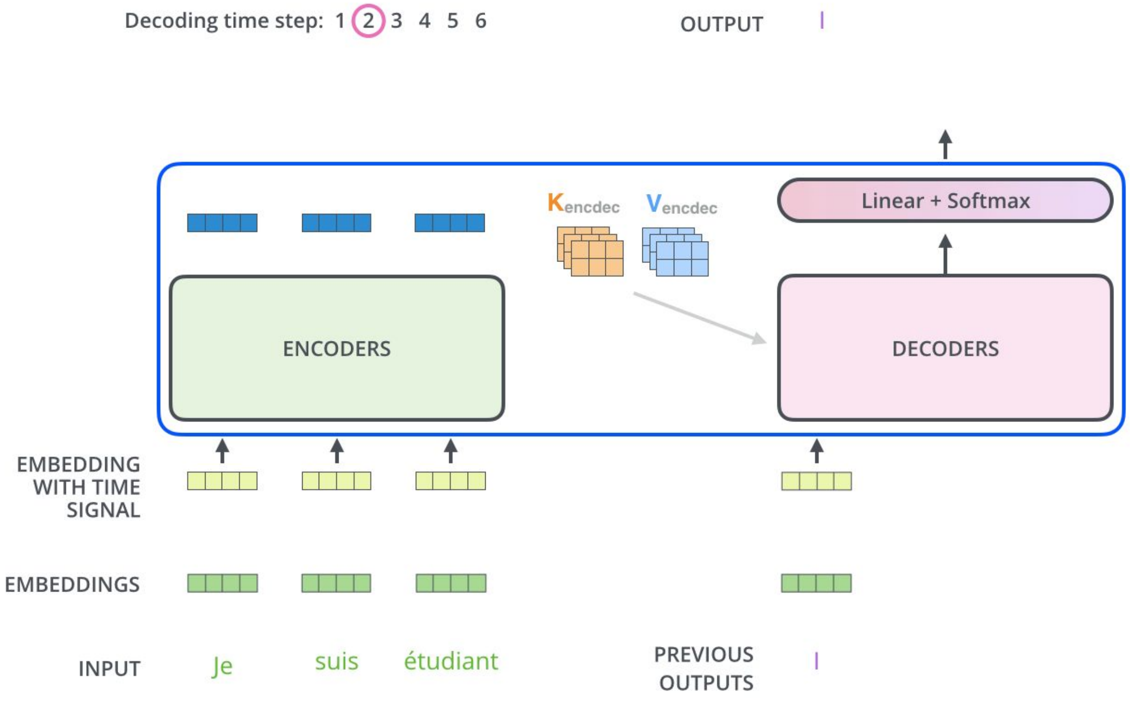
- Problems & Grading
- Dataset
- Submission & Rules
- Supplementary

Supplementary - Padding

- Different ground truth texts vary in length after tokenization, yet within the same batch, they need to be of uniform length. To achieve this, all **ground truth ids** are padded to the same length using the value -100. -100 will be ignore when computing cross entropy loss.

[64, 9592, 351]	→	[64, 9592, 351, -100]
[14595, 290]		[14595, 290, -100, -100]
[867, 10801, 8217, 290]		[867, 10801, 8217, 290]

Supplementary - what is autoregressive?



Reference: The Illustrated Transformer

Supplementary - Decoding strategy

- You could choose the advanced decoding strategy to improve your model when you autoregressively generate captions. For example:
    - Greedy search
    - Top-K sampling
    - Beam search
    - Nucleus sampling
    - …etc.

Please read the reference tutorial for details.

Supplementary - Toolkit summary

We provide some toolkits for you to more efficiently finish this homework.
Therefore, we summarize them in the end as follows:

- Pretrained OpenAI-CLIP
- pytorch-transformer
- Pytorch Image Models (timm)
- PEFT
- The annotated transformer