

# Updates (1/2)

- 11/19: Before implementing PEFT, you should first ensure the performance of the pretrained image encoder (choose a larger one) & cross attention block (include the projection layer). Without PEFT, the performance can be better than simple baseline at least.
- **11/27**: You should make sure your **predicted\_probs** & **gt\_tokens** for CrossEntropyLoss are correct. (correct position)
  - e.g (in training):
    - **model\_input\_tokens** : [**start\_token**, gt1, gt2, ..., gtN, **end\_token**, 50256, 50256] → length K  
(pad is 50256, for the same length K in a batch)
    - **predicted\_probs** : [probs1, probs2, ..., probsK] → length K (shape: (N, K, 50257))  
(each probs contains the prob for 50257 tokens)
    - **gt\_tokens** : [gt1, gt2, ..., gtN, **end\_token**, -100, -100, -100] → length K  
(pad is -100, for ignore\_index)
    - **loss** = cross\_entropy\_loss(**predicted\_probs** , **gt\_tokens**)
      - you need to reshape your **predicted\_probs** and **gt\_tokens**

# Updates (2/2)

- 11/22: We check the number of parameters by using your checkpoint, so you are only allowed to save the part of **trainable parameters**.

```
trainable_weights = [name for name, param in model.named_parameters() if param.requires_grad == True]
# ---Training process---
save_weights = {k: v for k, v in model.state_dict().items() if k in trainable_weights}
torch.save(save_weights, save_path)
```

- **(important)** 11/22: We **update the baseline of CLIPScore**. Please refer to **p.24** for the new baseline scores.
- 11/27: update the explanation in p.1 to avoid misunderstanding.

---

---

# Homework #3

Deep Learning for Computer Vision  
NTU, Fall 2023

---

---

# Outline

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

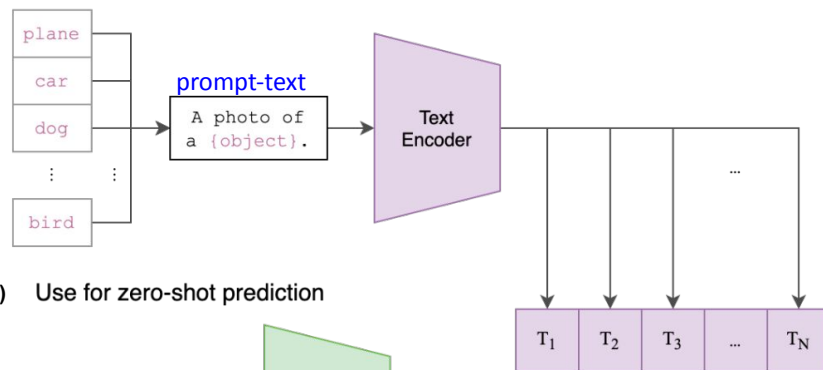
# Problems – Overview

- **Problem 1:** Zero-shot image classification with CLIP (30%)  
[hw3\_data/p1\_data]
- **Problem 2:** PEFT on Vision and Language Model for Image Captioning (70%)  
[hw3\_data/p2\_data], [hw3\_data/p3\_data]
- Please refer to “Dataset” section for more details about p1\_data/p2\_data/p3\_data.

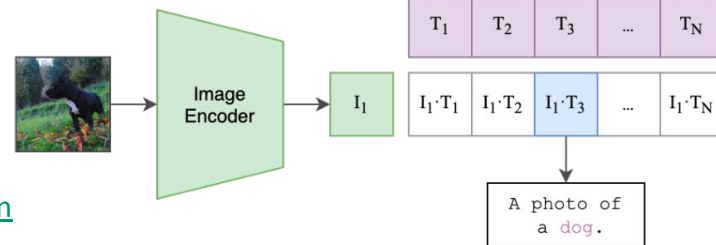
# Problem 1: Zero-shot Image Classification with CLIP

- Zero-shot: You **cannot** do any finetuning for the pretrained model.
- In this problem, you only need to evaluate the **pretrained** CLIP on image classification task.
  - Input:
    - image
    - text = [prompt-text; label text]
  - Output: class label  
(determined by similarity scores)
  - prompt-text can be designed by yourself
  - you can also modify CLIP image encoder

(1) Create dataset classifier from label text



(2) Use for zero-shot prediction



Reference: [Learning Transferable Visual Models From Natural Language Supervision](#)

GitHub: [OpenAI-CLIP](#)

# Problem 1: Evaluation

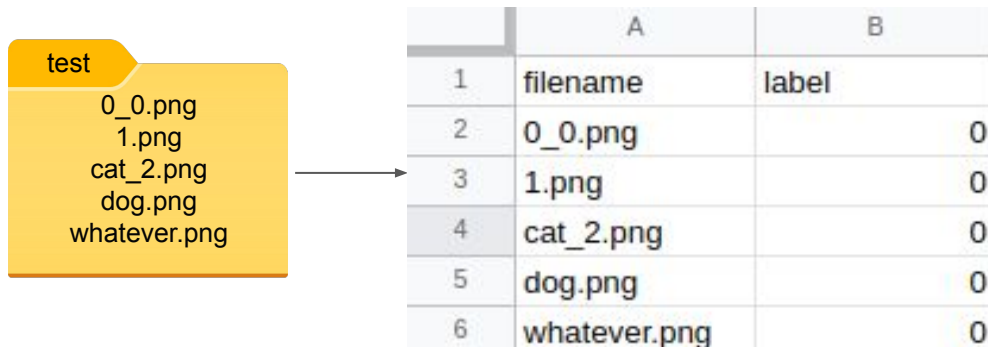
- Evaluation metric: Accuracy

- Accuracy is calculated over all test images.

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

- Sample Output

- Output format: CSV file with the first row being 'filename,label'
- Output filename must be identical to input filename
- No need to consider the order of filename



The diagram illustrates the output format. On the left, a yellow folder icon labeled 'test' contains a list of image filenames: 0\_0.png, 1.png, cat\_2.png, dog.png, and whatever.png. An arrow points from this folder to a table on the right. The table has three columns: an index column, a 'filename' column, and a 'label' column. The first row is a header with 'filename' and 'label'. Subsequent rows show the filenames from the 'test' folder in the same order, each followed by a label value of 0.

	A	B
1	filename	label
2	0_0.png	0
3	1.png	0
4	cat_2.png	0
5	dog.png	0
6	whatever.png	0

## Problem 1: Grading - Baselines (15%)

- Public Baseline (10%) - Accuracy of 65% (validation data is in `p1_data/val/`)
- Private Baseline (5%) - TBD (testing data is not available for students)



# Problem 1: Grading - Report (15%)

## 1. Methods analysis (3%)

- Previous methods (e.g. VGG and ResNet) are good at one task and one task only, and requires significant efforts to adapt to a new task. Please explain why CLIP could achieve competitive zero-shot performance on a great variety of image classification datasets.

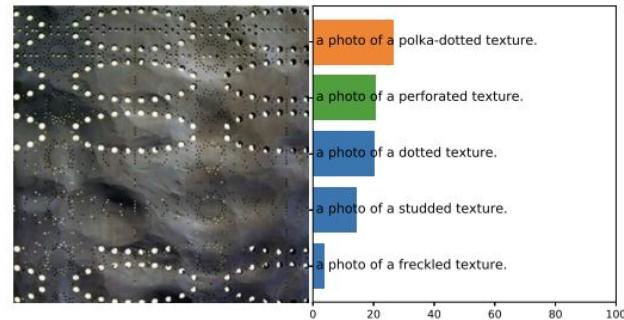
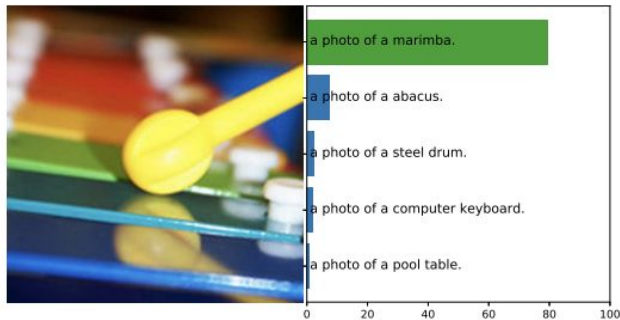
## 2. Prompt-text analysis (6%)

- Please compare and discuss the performances of your model with the following **three** prompt templates:
  - i. *"This is a photo of {object}"*
  - ii. *"This is not a photo of {object}"*
  - iii. *"No {object}, no score."*

# Problem 1: Grading - Report (cont'd) (15%)

## 3. Quantitative analysis (6%)

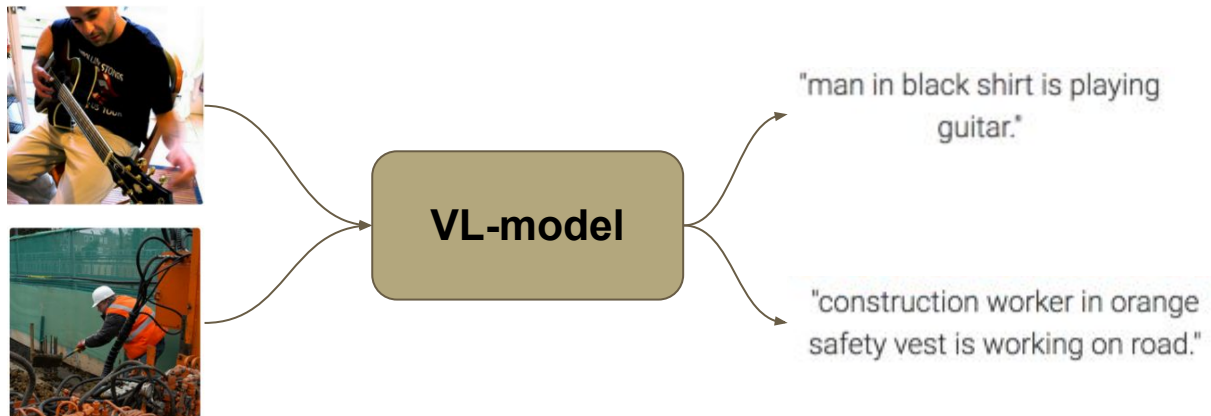
- Please sample **three** images from the validation dataset and then visualize the probability of the top-5 similarity scores as following example:



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

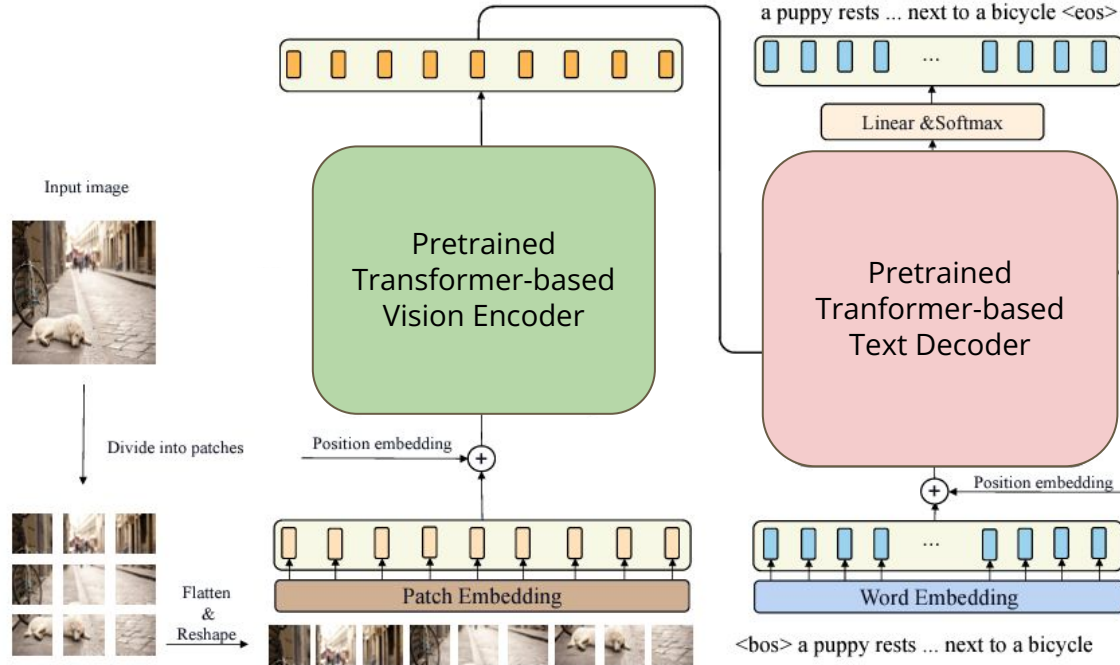
In this problem, you need to:

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)



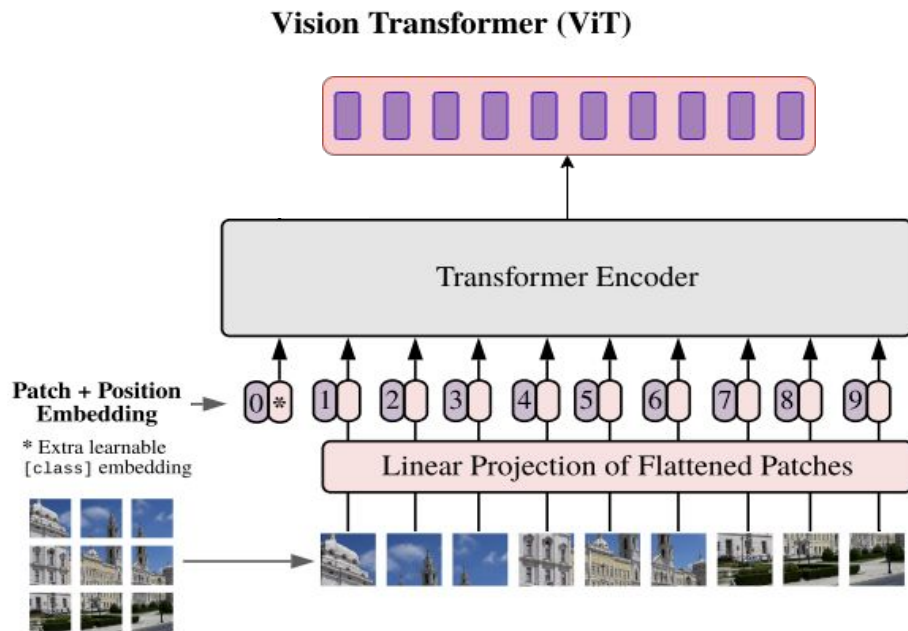
## Problem 2: Model Overview

- You are limited to use **transformer encoder + decoder** architecture in this assignment.



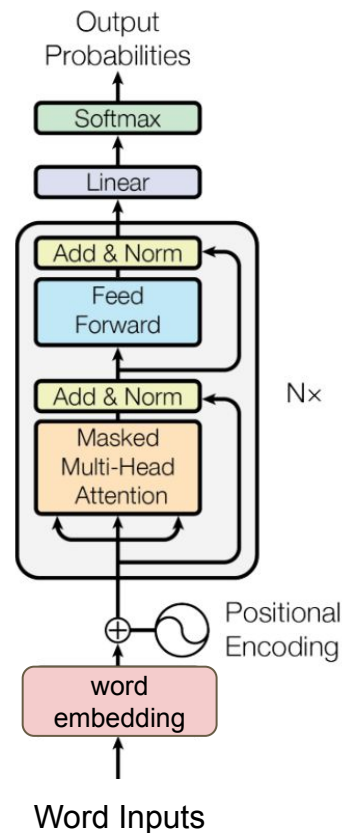
## Problem 2: Model Implementation

- 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](#)



## 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. add cross attention block, adjust input)



## 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'

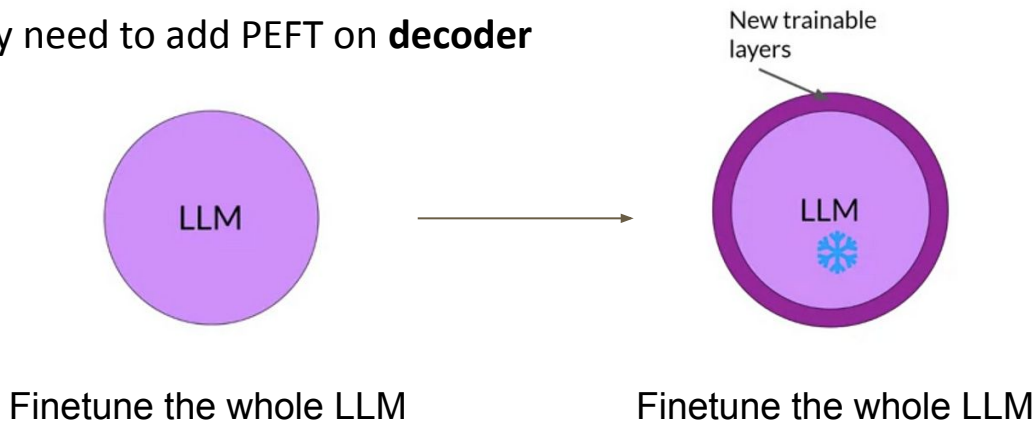
context = encoding.encode(prompt)
print(context)
print(encoding.decode(context))
```

✓ 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 **three** types of PEFT
- You only need to add PEFT on **decoder**

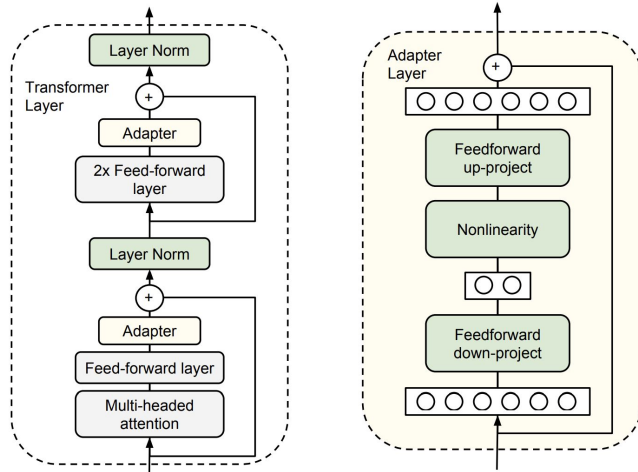




## Problem 2: PEFT (cont'd)

### Type 1: Adapter

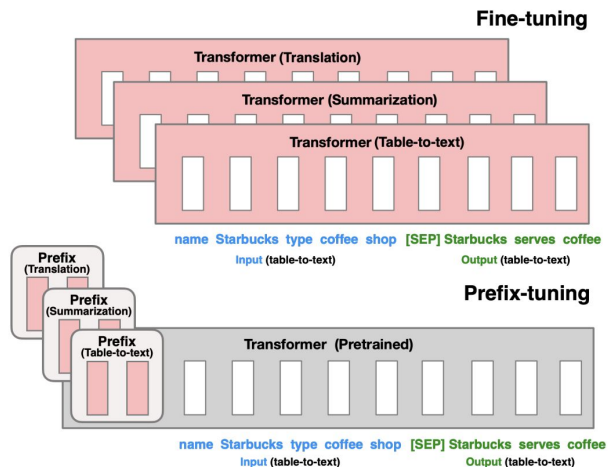
- add extra module in your transformer decoder
- design where to place your adapters (e.g. specific decoder layers), the intermediate dimension of your adapters
- you need to implement **from scratch**



## Problem 2: PEFT (cont'd)

### Type 2: prefix tuning

- add trainable “prefix” in your decoder input
- you can also try other prompting-based methods, e.g. prompt tuning, p-tuning, ...
- you need to implement **from scratch**



## Problem 2: PEFT (cont'd)

### Type 3: 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>

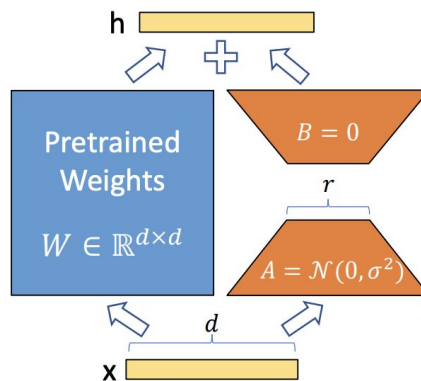
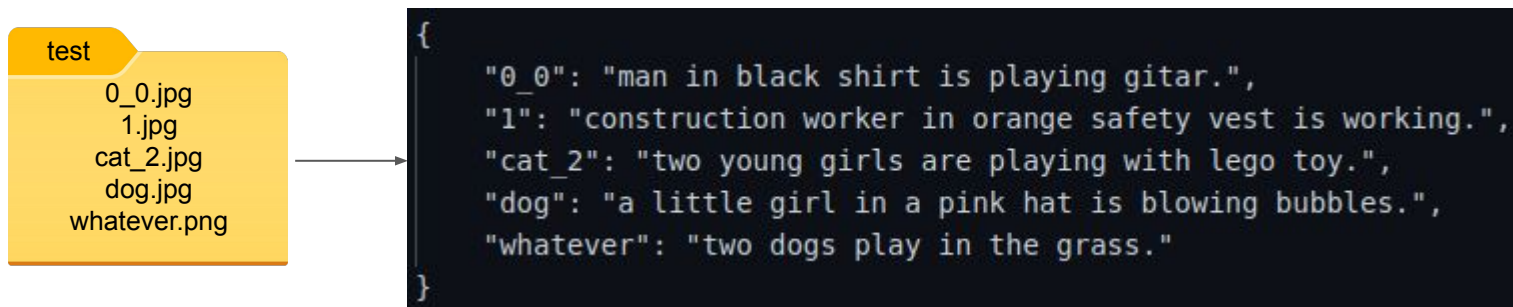


Figure 1: Our reparametrization. We only train  $A$  and  $B$ .

## Problem 2: 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:



**Please remove the filename extension**

## Problem 2: Evaluation – CIDEr

- CIDEr is a 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

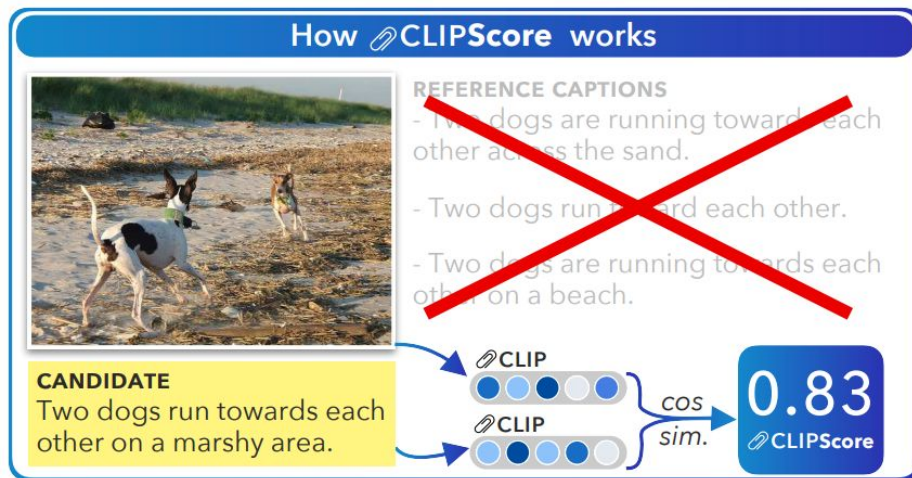
**C1:** An eagle is perched among trees.

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

CIDEr scores:  $C1 < C2$

## Problem 2: Evaluation – CLIPScore

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



## Problem 2: Evaluation

- We provide evaluation code which calculates CIDEr and CLIPScore to check performance [[p2\\_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%) - 1789 validation data ([p2\_data/images/val])
  - Simple baseline (13%) - CIDEr of **0.8**, CLIPScore of **0.68**
  - Strong baseline (7%) - CIDEr of **0.89**, CLIPScore of **0.71**
- Private Baseline (20%) - 1404 test data (not available for students)
  - Simple baseline (13%) - TBD
  - Strong baseline (7%) - TBD
- **The total number of trained parameters: < 35M**
  - You are only allowed to save the trained parameters, model over 35M **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: Grading – Report (30%)

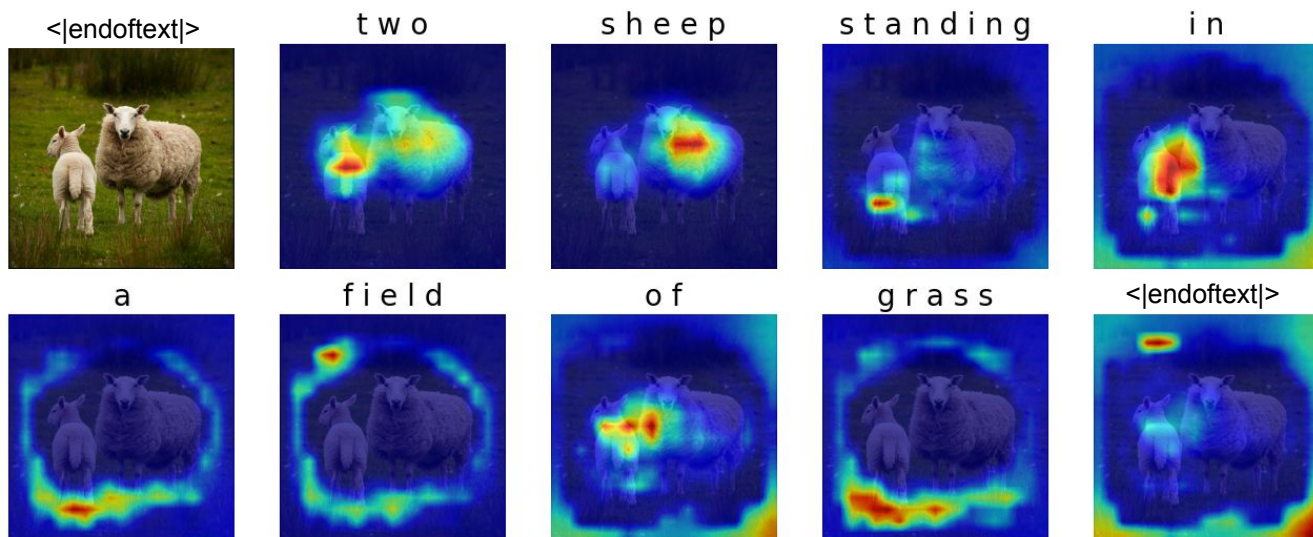
- Evaluation metrics report (10%)
- Visualization of Attention in Image Captioning (20%)

## Evaluation metrics report (10%)

1. Report your best setting and its corresponding CIDEr & CLIPScore on the validation data. (TA will reproduce this result) (2.5%)
2. Report 3 different attempts of PEFT and their corresponding CIDEr & CLIPScore. (7.5%, each setting for 2.5%)

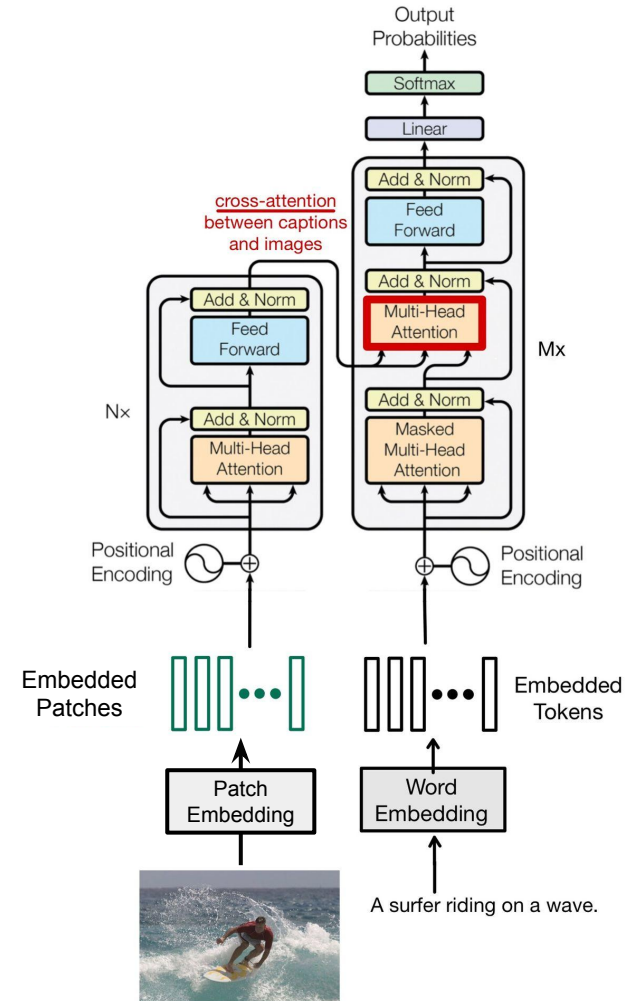
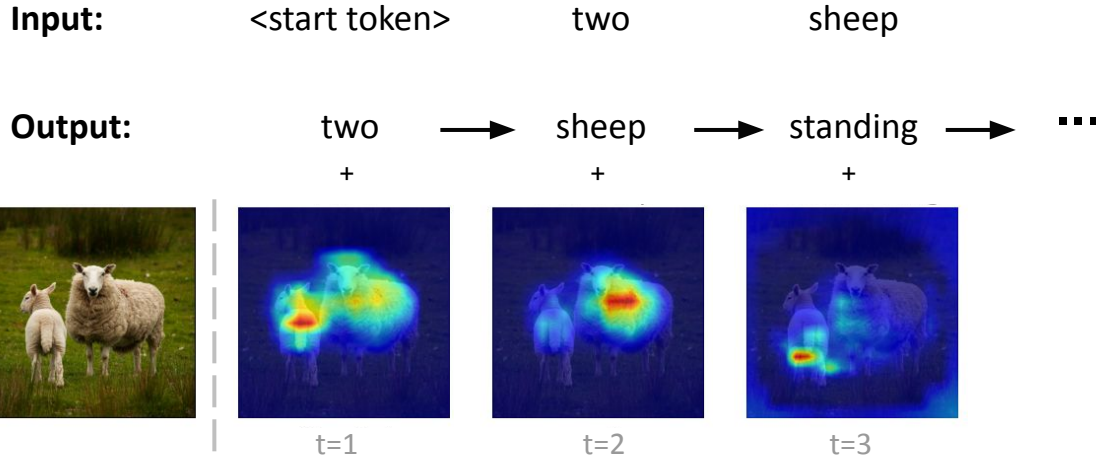
## Visualization of Attention in Image Captioning (20%)

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



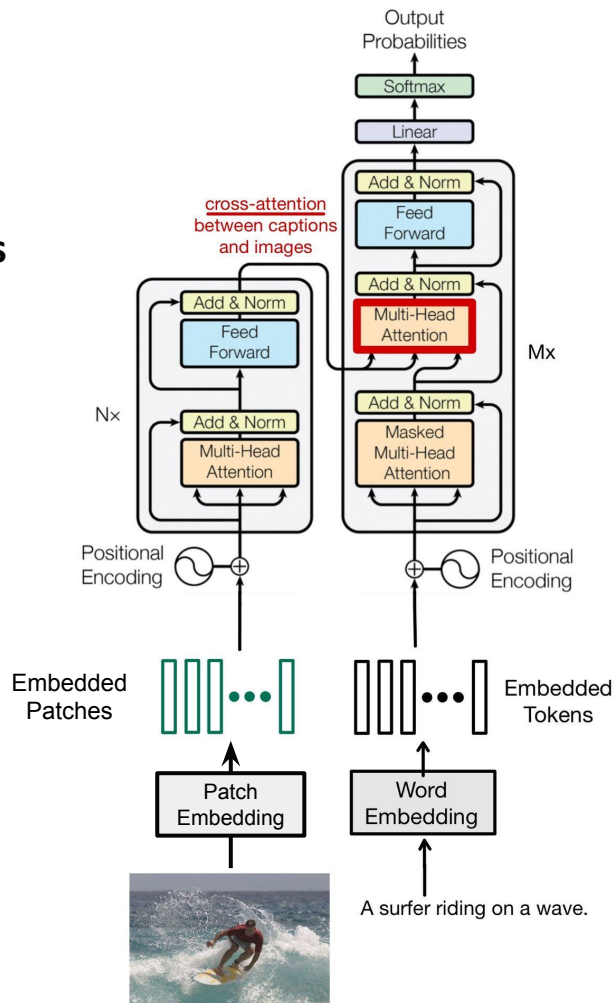
# Models and Settings

- 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 cross-attention between the image patches and each predicted word in your own caption.



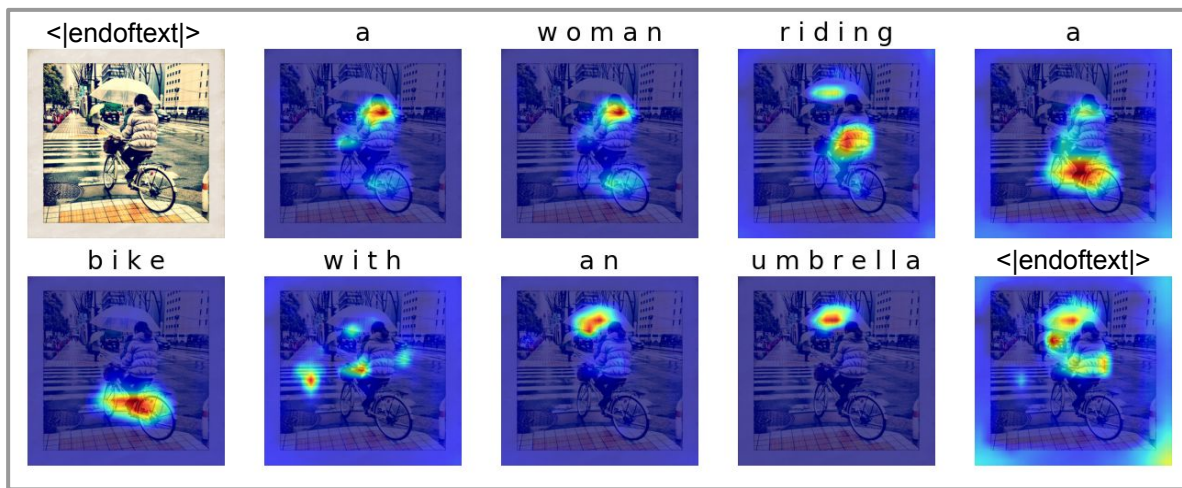
# Models and Settings

- The attention weights you have to visualize are in the **cross attention block** you add into decoder.
  - The cross-attention weights in the last decoder layer (or the second to last) have better visualization results.
  - If you not use cross attention block, visualize the attention weights between all image tokens and each predicted word in masked multi-head attention.
- Practically, you should **modify** the code in problem 2 to get the attention weights for visualization.



# Visualization of Attention in Image Captioning (20%) (cont'd)

1. TA will give you 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: (10%, each image for 2%)



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

## Visualization of Attention in Image Captioning (20%) (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. (5%)

3. 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? (5%)

# 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/11rP6KmR5Qwjhx0rfag0b5TZGBTRuPtQR/view?usp=sharing>

- (Option 2) Run the bash script provided in the hw3 repository

```
$ bash get_dataset.sh
```

# Problem 1: Dataset

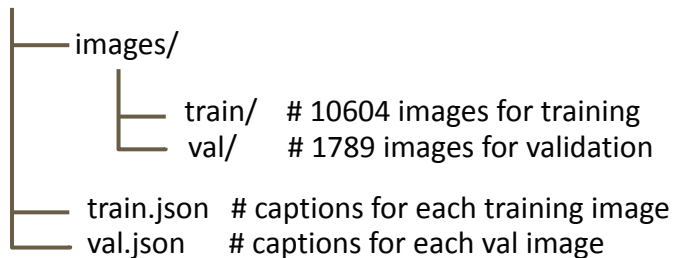
- The dataset is the same as the dataset in problem 1 of homework1, which consists of images in **50** classes
- Since problem 1 is a zero-shot task, we don't provide training data.
- We only provide validation data for evaluation:
  - p1\_data/val/
    - **2500** images
    - Images are named '{class\_label}\_{image\_id}.png'
  - p1\_data/id2label.json
    - This file contains mapping from *class\_label* to *class\_name*.

```
{  
  "0": "bicycle",  
  "1": "chair",  
  "2": "camel",  
  "3": "rabbit",  
  "4": "lamp",  
  "5": "clock",  
}
```

# Problem 2: Dataset

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

p2\_data/



```
{
  "info": "dlcv2023-hw3-train",
  "licenses": "",
  "annotations": [
    {
      "caption": "a young girl inhales with the int",
      "id": 152106,
      "image_id": 60623
    }
    // ...
  ],
  "images": [
    {
      "file_name": "000000060623.jpg",
      "id": 60623
    }
    // ...
  ]
}
```

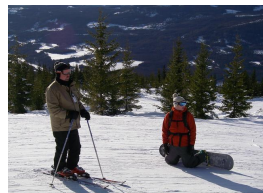
A brief example for train.json

## Problem 2: 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 are 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/gj9HrVrt>

- 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 `p2_evaluate.py`
- Visualization code for problem 3 can be `.ipynb` file
- **Don't push the dataset to your repo.**
- 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 $3`
    - \$1: path to the **folder** containing test images (images are named xxxx.png, where xxxx could be any string)
    - \$2: path to the **id2label.json**
    - \$3: path of the output **csv file** (e.g. output\_p1/pred.csv)
- Please follow the naming rules in **p.7**
- Note that you should **NOT** hard code any path in your file or script.
- Your testing code have to be finished in **10 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.17**
- 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 your code on **Linux** system, so try to make sure your 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 20.04.1 LTS
- NVIDIA GeForce RTX 2080 Ti (11 GB)
- GNU bash, version 5.0.17(1)-release
- Python 3.8

# 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

- 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
- [language-evaluation](#)、[clip](#) (please follow these github repos to install the packages)
- Any dependencies of above packages, and other standard python packages
- 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

• **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: **112/11/28 (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!
- 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.**
  - Wed. 13:20~14:10 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]

[14595, 290]

[867, 10801, 8217, 290]

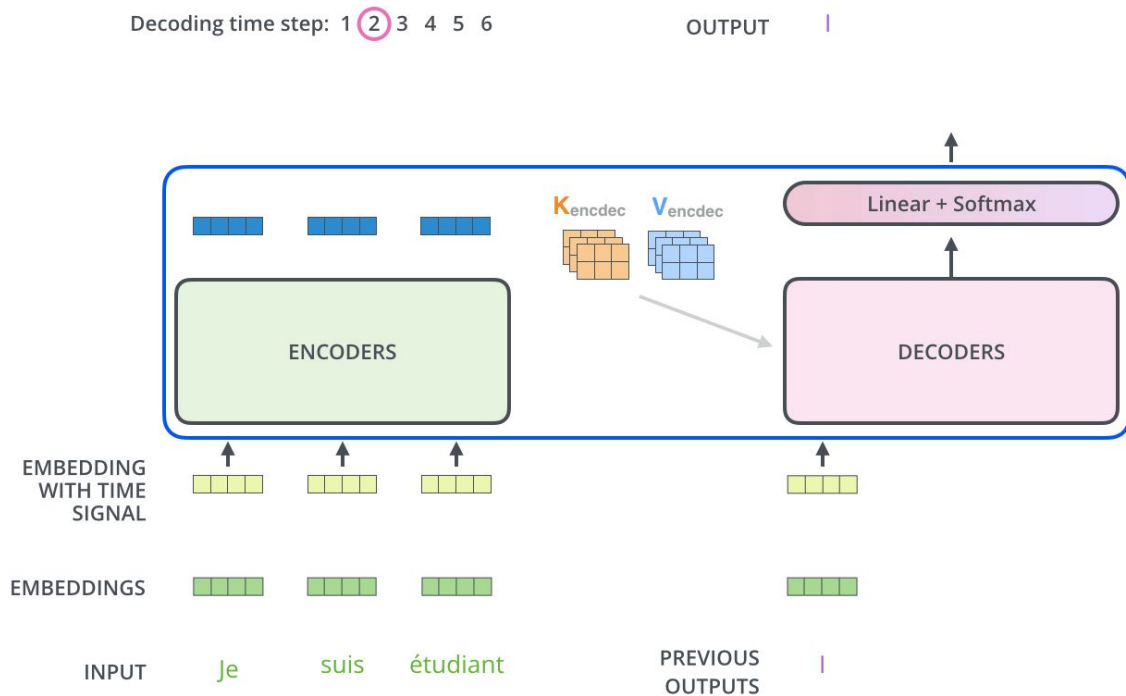


[64, 9592, 351, -100]

[14595, 290, -100, -100]

[867, 10801, 8217, 290]

# Supplementary - what is autoregressive?





# 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](#)