Homework 3

Deep Learning for Computer Vision
NTU, Fall 2025
TA: Bing-Yi Yang, Kuei-Chun Wang, Chang-Hsun Wu

Outline

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

Problems – Overview

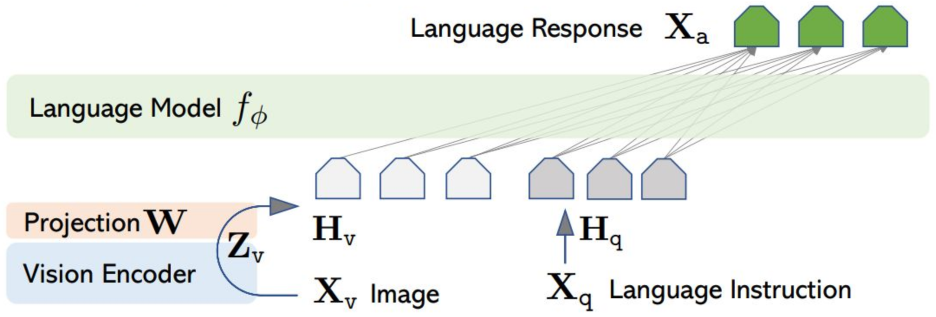
- Problem 1: Zero-shot inference with LLaVA (34%)
  - 1-1 Zero-shot with origin LLAVA
  - 1-2 Visual Contrastive Decoding
- Problem 2 :PEFT on Vision and Language Model for Image Captioning (66%)

Please refer to “Dataset” section for more details about all datasets.

Problem 1: Zero-shot inference with LLaVA

Problem 1-1: Zero-shot with origin LLAVA

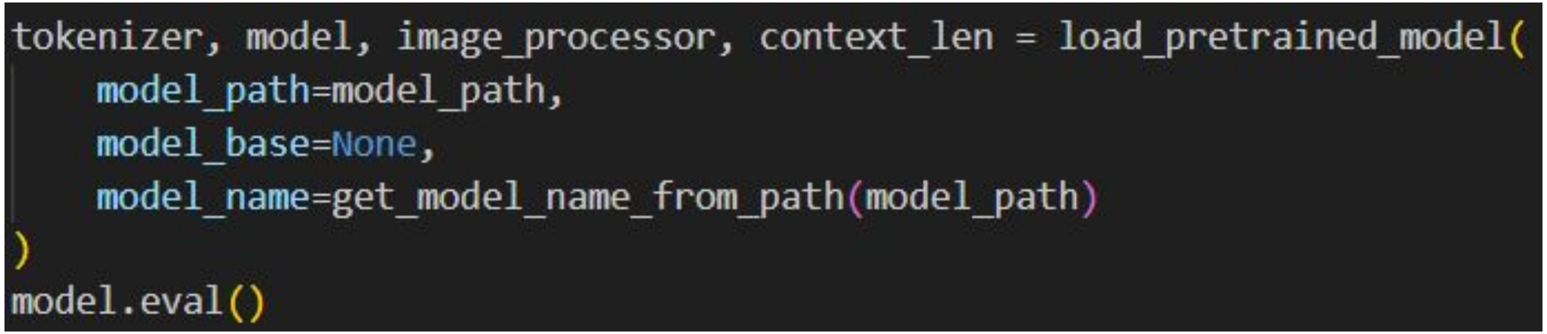
- Zero-shot: You **cannot** do any finetuning for the pretrained model
- In this problem, you only need to evaluate the **pretrained** LLaVA on Pope Dataset
  - Input: (1)image (2)language instruction (3)generation config
  - Output: The model's raw output must be **pruned to a single word**: only "Yes" or "No"
    - If the model outputs "Yes, there is ...", treat it as **Yes**
    - If the model outputs "No, I don't see ...", treat it as **No**



Reference: Visual Instruction Tuning

Problem 1-1: Zero-shot with origin LLAVA

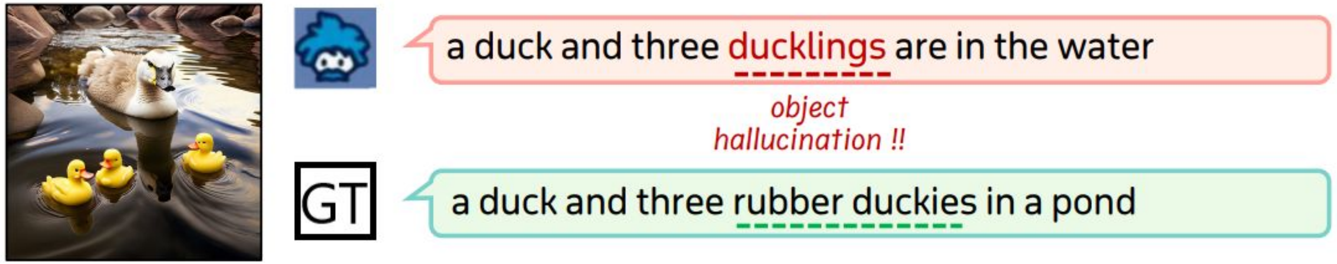
- Use the llava architecture provided
- Download the pretrained weights by cloning:
  git clone https://huggingface.co/liuhaotian/llava-v1.5-7b
- Load the model by **load_pretrained_model** function



```
tokenizer, model, image_processor, context_len = load_pretrained_model(
    model_path=model_path,
    model_base=None,
    model_name=get_model_name_from_path(model_path)
)
model.eval()
```

Problem 1-2: Visual Contastive Decoding

- MLLMs demonstrate remarkable success across various vision-language tasks.
- However, MLLMs suffer from **hallucinations**.
    - E.g. Object Hallucination is when a MLLM mentions nonexistent objects

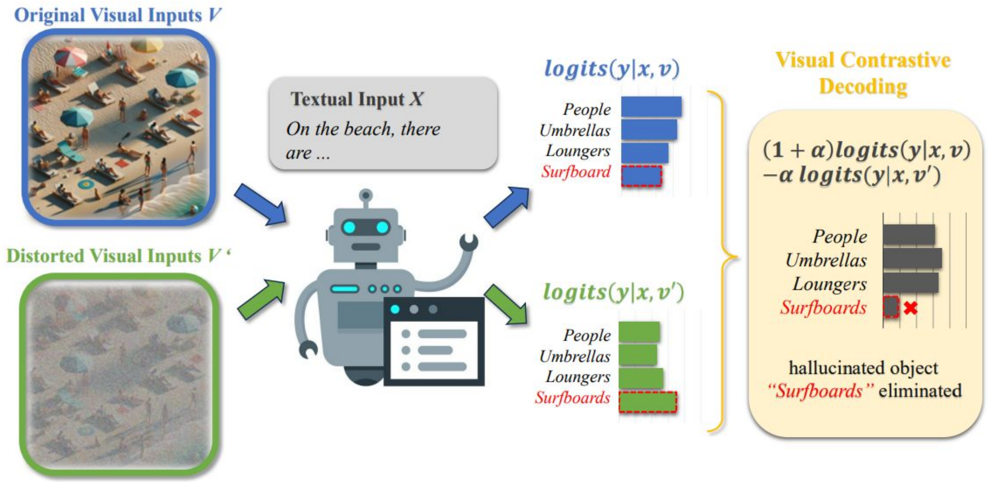

a duck and three ducklings are in the water
object hallucination !!
GT
a duck and three rubber duckies in a pond

- Why MLLMs suffer from Object Hallucination?
    - MLLMs rely on common data patterns instead of real visual cues. (**statistics bias**)
    - MLLMs follow common word associations, ignoring real visual cues. (**language priors**)

Reference: Pensieve: Retrospect-then-Compare Mitigates Visual Hallucination

Problem 1-2: Visual Contastive Decoding

- **Visual Contastive Decoding** is a zero-shot method aims to mitigate object hallucinations
  - **VCD** penalizing predictions that persist even when visual evidence is distorted.
  - Effectively filtering out tokens driven by **language priors** rather than visual reality.



Original Visual Inputs V
Distorted Visual Inputs V'
Textual Input X
On the beach, there are ...
$logits(y|x,v)$
People
Umbrellas
Loungers
Surfboard
$logits(y|x,v')$
People
Umbrellas
Loungers
Surfboards
Visual Contrastive Decoding
$(1+\alpha)logits(y|x,v) - \alpha\, logits(y|x,v')$
People
Umbrellas
Loungers
Surfboards
hallucinated object "Surfboards" eliminated

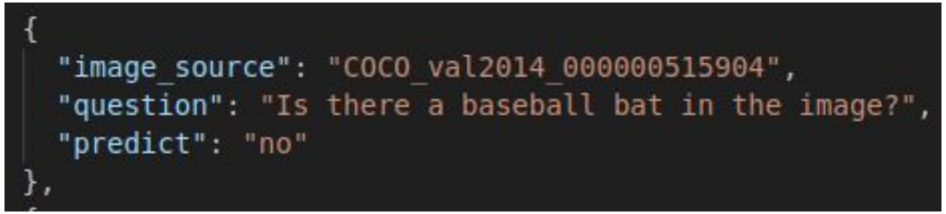
Reference: Mitigating Object Hallucinations in Large Vision-Language Models throughVisual Contrastive Decoding (CVPR 2024 Highlight)

Problem 1-2: Visual Contastive Decoding

- You need to modify functions in the LLaVA backbone provided to support **visual contrastive decoding (VCD)**
- In this problem, you only need to evaluate the **pretrained LLaVA with VCD** on the POPE dataset
  - Input: (1)image (2)language instruction (3)generation config
  - Output: The model's raw output must be **pruned to a single word**: only "Yes" or "No"
    - If the model outputs "Yes, there is ...", treat it as **Yes**
    - If the model outputs "No, I don't see ...", treat it as **No**
- Hint :
  - Add variables and implement a new function in llava_llama.py
  - Reference : https://github.com/DAMO-NLP-SG/VCD

Problem 1: Evaluation

- We will only test the result of VCD (Problem 1-2)
- Evaluation metrics: Accuracy
- Arguments:
  - pred_file: your output json file (filename: pred.json)
  - annotation_file: ground truth json file (e.g. “p1_data/val.json”)
  - llava weight path: path to the llava weight (e.g. llava-v1.5-7b)
  - images_root: path to the folder containing val images (e.g. “p1_data/images/val”)
- Output Format :



```
{
    "image_source": "COCO_val2014_000000515904",
    "question": "Is there a baseball bat in the image?",
    "predict": "no"
},
```

Problem 1: Grading – Baselines (25%)

- Public Baseline (15%) - 1000 validation data ([p1_data/images/val])
  - Simple baseline (10%) : 0.49
  - Strong baseline (5%) : 0.55
- Private Baseline (10%) - 1000 test data (not available for students)
  - Simple baseline (7%)
  - Strong baseline (3%)

Problem 1: Grading – Report (9%)

- Q1 (5%) : Report the accuracy of LLaVA w/o VCD (3-1) and w VCD (3-2)
- Q2 (4%) : Describe the underlying mechanism of Visual Contrastive Decoding (VCD) and how it helps mitigate object hallucinations in Large Vision-Language Models

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

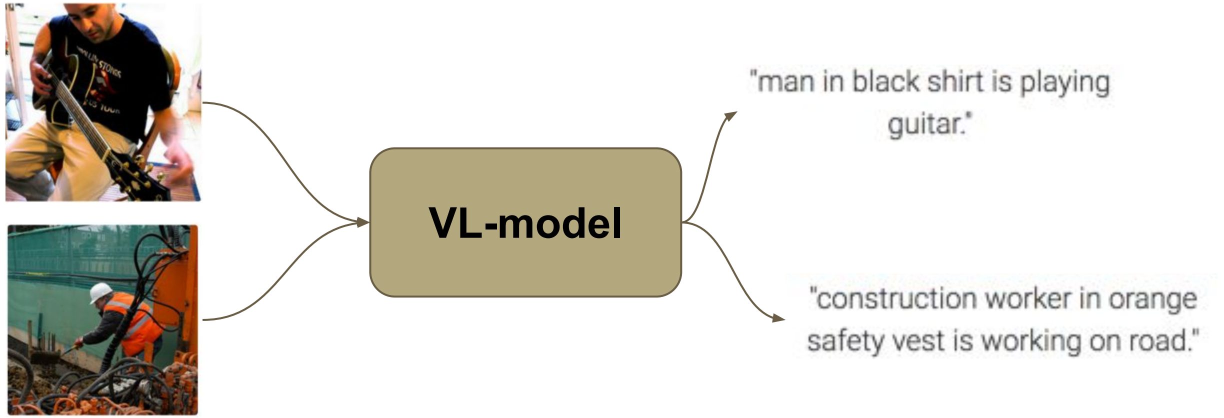
1. Task description
2. Model architecture
3. Fine-tune using PEFT method
4. Evaluation
5. Grading

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

Problem 2: Task Description

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 image captioning (from scratch)


VL-model
"man in black shirt is playing guitar."
"construction worker in orange safety vest is working on road."

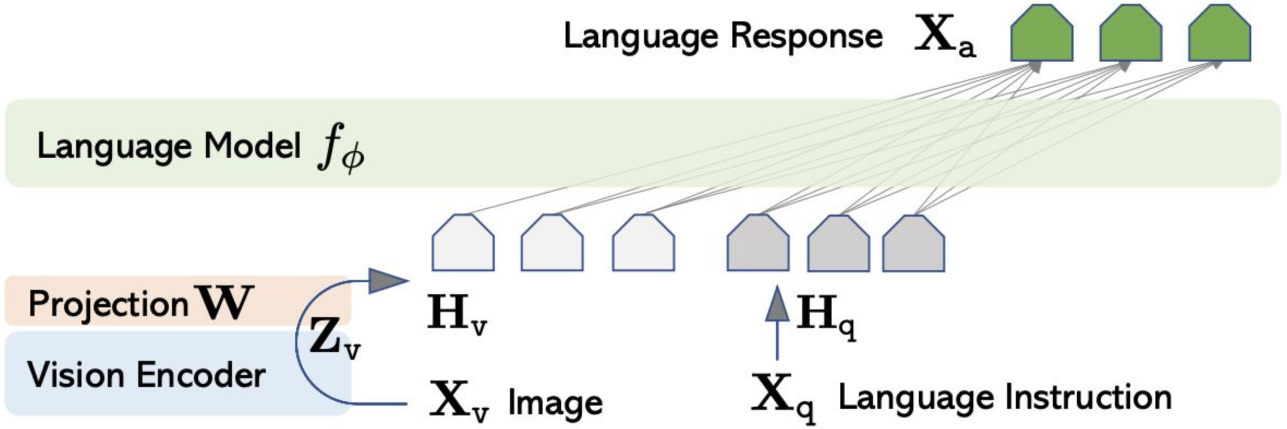
Problem 2:
PEFT on Vision and Language Model for Image Captioning

1. Task description
2. Model architecture
3. Fine-tune using PEFT method
4. Evaluation
5. Grading

Problem 2: Model Architecture – Overview

You are limited to use *LLaVA* architecture in this assignment

- Language Model: Pretrained transformer-base decoder
- Vision Encoder: ViT (Vision Transformer)
- Projection: An MLP maps visual token to language space

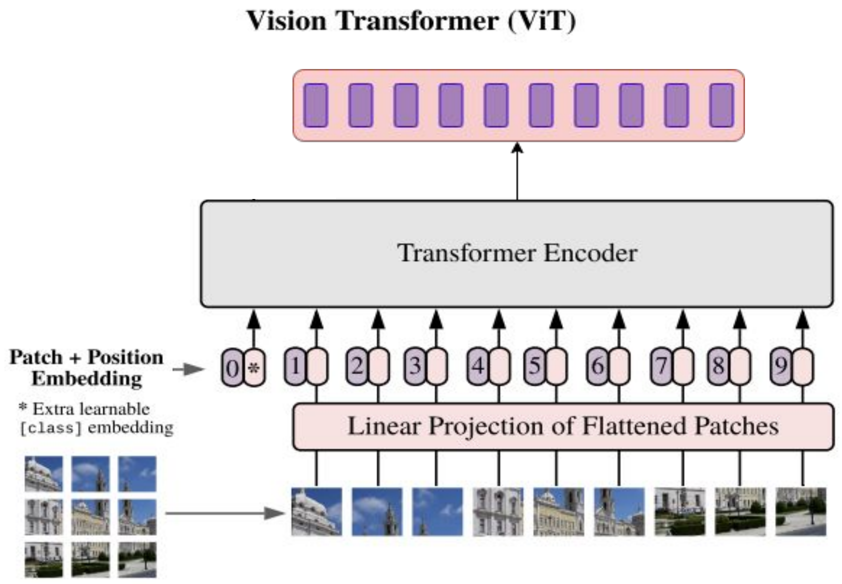


Reference: Improved Baselines with Visual Instruction Tuning

Problem 2: Model Architecture – Vision Encoder

You are limited to use *Vision Transformer (ViT)* as vision encoder

- Any **pretrained** ViTs (e.g., ViT-Base, CLIP-ViT-Base, ViT-Large, etc) are allowed
- You can use ViT from timm, openai/CLIP

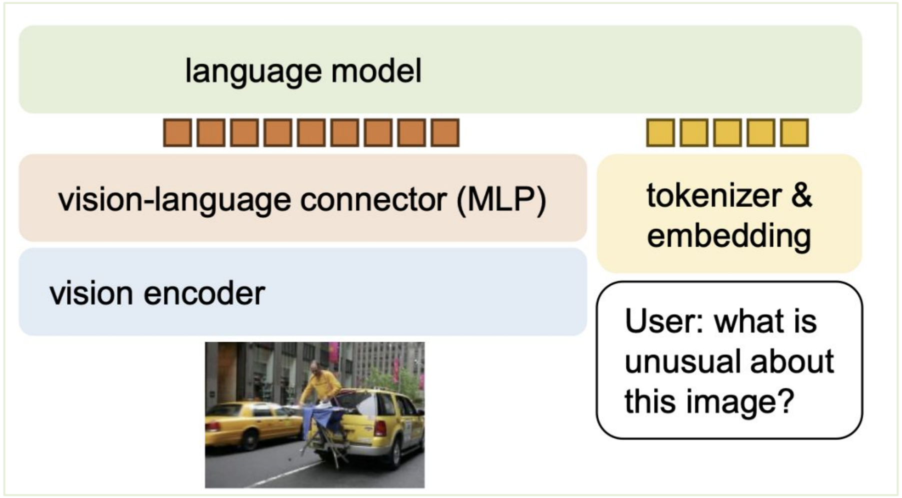

Vision Transformer (ViT)
Transformer Encoder
Patch + Position Embedding
* Extra learnable [class] embedding
0
*
1
2
3
4
5
6
7
8
9
Linear Projection of Flattened Patches

Reference code: timm

Problem 2: Model Architecture – Language Model

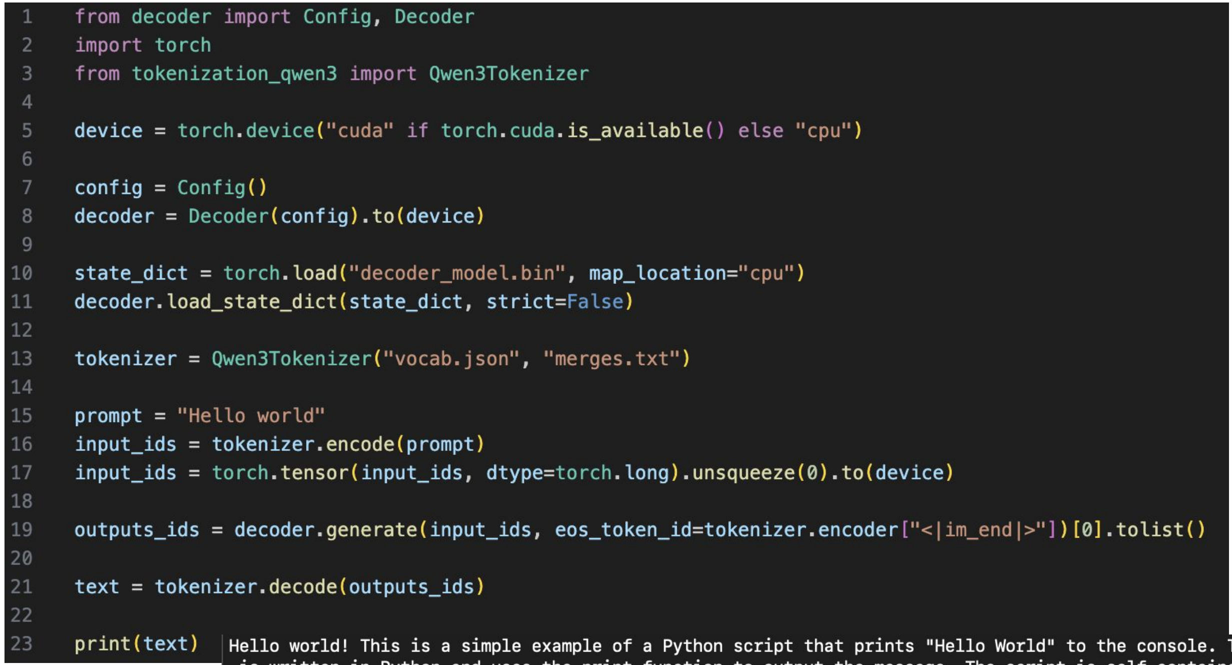
You are limited to use the pretrained decoder we provide

- We provide [decoder.py][p2_data/decoder_model.bin]
- **You need to modify the decoder** to generate captions based on visual features (e.g., adjust input, concat visual token with text token)

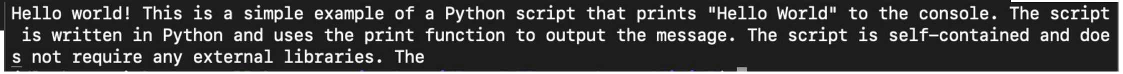

language model
vision-language connector (MLP)
tokenizer & embedding
vision encoder
User: what is unusual about this image?

Problem 2: Model Architecture – Language Model

Here’s an example of how to load the decoder and input a prompt



```python
from decoder import Config, Decoder
import torch
from tokenization_qwen3 import Qwen3Tokenizer

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

config = Config()
decoder = Decoder(config).to(device)

state_dict = torch.load("decoder_model.bin", map_location="cpu")
decoder.load_state_dict(state_dict, strict=False)

tokenizer = Qwen3Tokenizer("vocab.json", "merges.txt")

prompt = "Hello world"
input_ids = tokenizer.encode(prompt)
input_ids = torch.tensor(input_ids, dtype=torch.long).unsqueeze(0).to(device)

outputs_ids = decoder.generate(input_ids, eos_token_id=tokenizer.encoder["<|im_end|>"])[0].tolist()

text = tokenizer.decode(outputs_ids)

print(text)
```

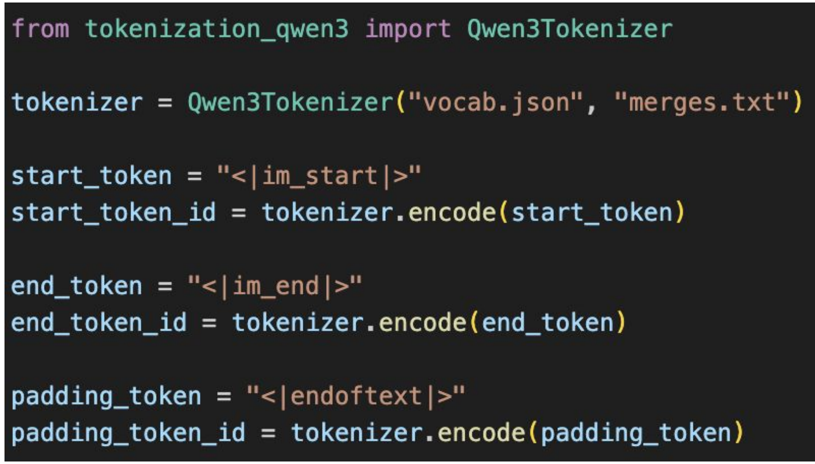


```
Hello world! This is a simple example of a Python script that prints "Hello World" to the console. The script
 is written in Python and uses the print function to output the message. The script is self-contained and doe
s not require any external libraries. The
```

Problem 2: Model Architecture – Tokenizer

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:
  - vocab.json, merge.txt
  - class **Qwen3Tokenizer** in tokenization_qwen3.py
- The **start token** should be ”<|im_start|>”
- The **end token** should be “<|im_end|>”
- The **padding token** should be “<|endoftext|>”



```python
from tokenization_qwen3 import Qwen3Tokenizer

tokenizer = Qwen3Tokenizer("vocab.json", "merges.txt")

start_token = "<|im_start|>"
start_token_id = tokenizer.encode(start_token)

end_token = "<|im_end|>"
end_token_id = tokenizer.encode(end_token)

padding_token = "<|endoftext|>"
padding_token_id = tokenizer.encode(padding_token)
```

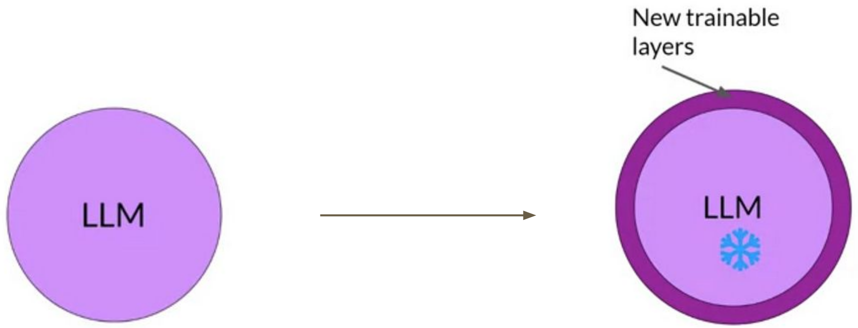
Problem 2:
PEFT on Vision and Language Model for Image Captioning

1. Task description
2. Model architecture
3. Fine-tune using PEFT method
4. Evaluation
5. Grading

Problem 2: Fine-tune using PEFT method

PEFT (Parameter Efficient Fine-Tuning)

- Avoid extremely high computational cost for finetuning LLM
- The number of parameters to be trained is limited



Finetune the whole LLM

Finetune the whole LLM

Reference: Introduction of PEFT

Problem 2: Fine-tune using PEFT method – LoRA

In this problem, you have to implement LoRA, a widely used PEFT method

- **DO NOT** directly use Hugging Face (transformers), but you can reference the code
- You only need to add PEFT on **decoder**

LoRA (Low-Rank Adaptation)

- 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

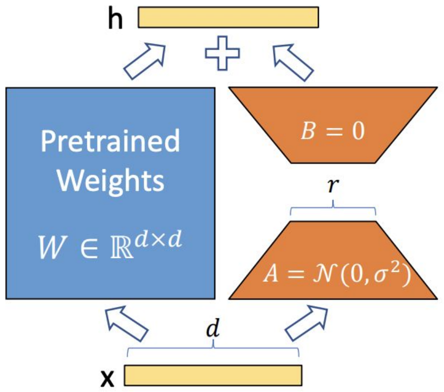


Figure 1: Our reparametrization. We only train A and B.

Reference: LoRA: Low-Rank Adaptation of Large Language Models
Reference code: PEFT

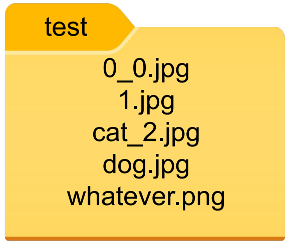
Problem 2:
PEFT on Vision and Language Model for Image Captioning

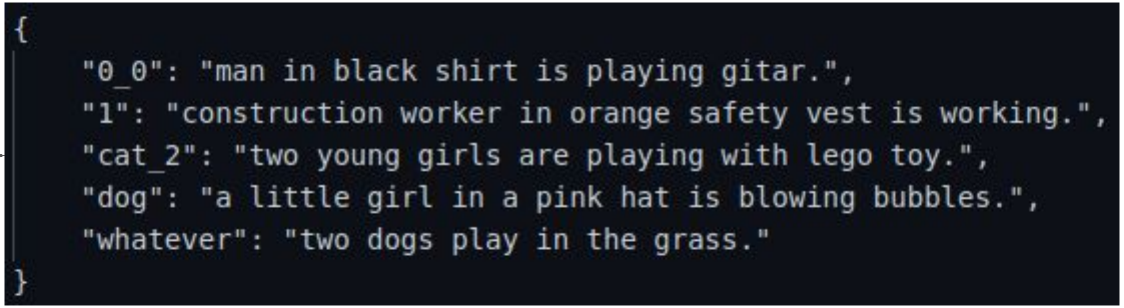
1. Task description
2. Model architecture
3. Fine-tune using PEFT method
4. Evaluation
5. Grading

Problem 2: Evaluation – Metrics

Evaluation metrics: CIDEr and CLIPScore

- 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 – Evaluation Code

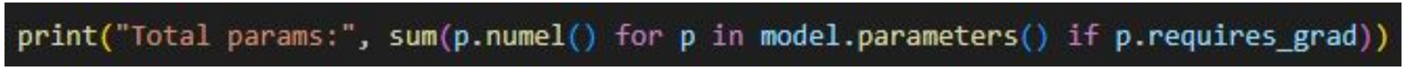
Evaluation Code

- 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:
PEFT on Vision and Language Model for Image Captioning

Problem 2: Grading – Baselines (56%)

- You only need to submit **your best setting**
- Public Baseline (28%) - 2000 validation data ([p2_data/images/val])
  - Simple baseline (11%) - CIDEr of **0.76**, CLIPScore of **0.67**
  - Medium baseline (9%) - CIDEr of **0.85**, CLIPScore of **0.69**
  - Strong baseline (8%) - CIDEr of **0.90**, CLIPScore of **0.71**
- Private Baseline (28%) - 2000 test data (not available for students)
  - Simple baseline (11%) - TBD
  - Medium baseline (9%) - TBD
  - Strong baseline (8%) - TBD
- **The total number of trained parameters should < 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: Grading – 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 **two different attempts of LoRA setting** (e.g. initialization, alpha, rank…) and their corresponding **CIDEr & CLIPScore** (5%, each setting for 2.5%)

Dataset

Tools for Dataset

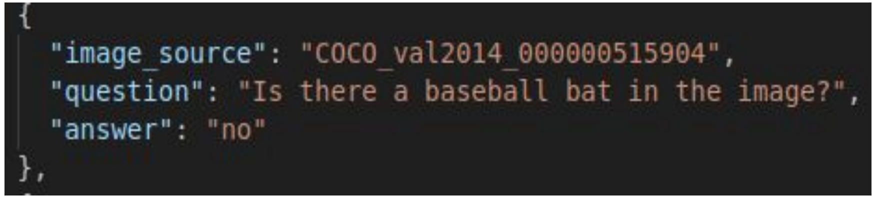
- Download the dataset
    - (Option 1) Manually download the dataset
      hw3_data.zip
    - (Option 2) Run the bash script provided in the GitHub classroom repository
      $ bash get_dataset.sh

Problem 1: Dataset

- The dataset consists of images and QA pairs
- Format:

```
└── p1_data/
    ├── images/
    |   └── val/ # 490 images for validation
    └── val.json   # 1000 QA pairs
```

"Note: **A single image may be associated with multiple QA pairs**."



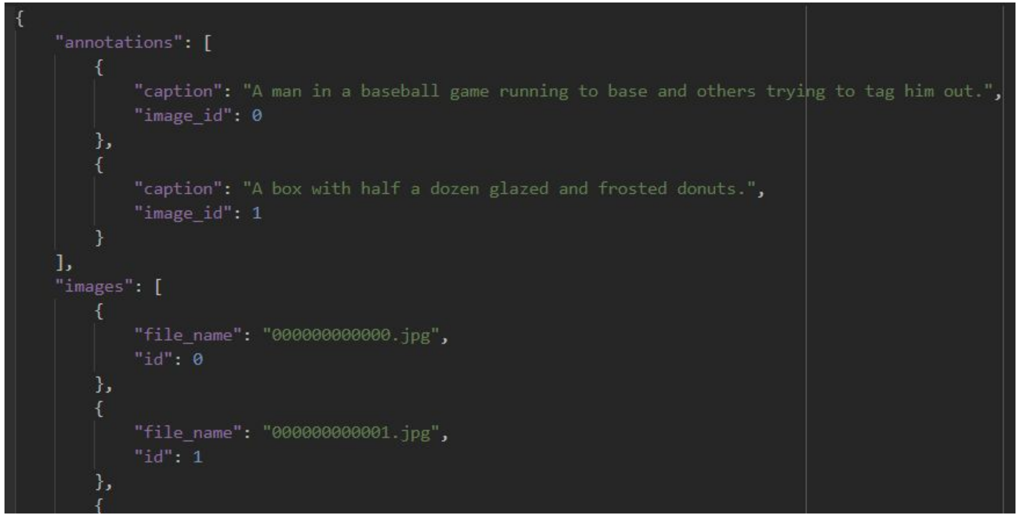
```
{
  "image_source": "COCO_val2014_000000515904",
  "question": "Is there a baseball bat in the image?",
  "answer": "no"
},
```

A brief example for pope_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

Submission & Rules

Submission

- Click the following link to get submission repo with your GitHub account: https://classroom.github.com/a/bemE4H4Y
    - 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 (ntu-dlcv-2025-fall-ta@googlegroups.com)
- By default, we will grade your **last submission (commit) before the deadline**. 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)
  - Model files (can be loaded by your python file)
- No need to upload evaluate.py
- **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 $4**
    - $1: path to the annotation **json** file (e.g. hw3/p1_data/test.json)
    - $2: path to the **folder** containing test images (e.g. hw3/p1_data/images/test/)
    - $3: path to the llava weight (e.g. llava-v1.5-7b)
    - $4: path to the output **json** file (e.g. hw3/output_p1/pred.json)
- Please follow the naming rules in p.10
- Note that you should **NOT** hard code any path in your file or script.
- Your testing code have to be finished in **30 minutes**.

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/val/)
        - $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.26
- Note that you should **NOT** hard code any path in your file or script.
- Your testing code have to be finished in **40 minutes**.

Rules – Bash Script

- You must **not** use commands such as **rm, sudo, CUDA_VISIBLE_DEVICES, cp, mv, mkdir, cd, pip** or other commands to change the 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 – Download Checkpoints

- If your checkpoints are larger than GitHub's maximum capacity (50 MB), you could download 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.
- 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 **10 minutes**.

Rules – Download Checkpoints (cont’d)

- 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

Rules – Environment (cont'd)

- 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 and other shell during inference.
- You must **NOT** use commands such as **sudo, CUDA_VISIBLE_DEVICES** 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

- python==3.10
- torchvision==0.15.2
- transformers==4.31.0
- torch==2.0.1
- tokenizers==0.13.3
- sentencepiece==0.1.99
- shortuuid
- accelerate==0.21.0
- peft==0.4.0
- bitsandbytes==0.41.0
- numpy==1.26.4
- scikit-learn==1.2.2
- gradio==3.35.2
- gradio_client==0.2.9
- requests
- httpx==0.24.0
- uvicorn
- fastapi
- einops==0.6.1
- einops-exts==0.0.4
- timm==0.6.13

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

Packages – Problem 2

- python==3.10
- imageio==2.36.1
- matplotlib==3.9.2
- numpy==1.26.4
- Pillow==11.0.0
- scipy==1.14.1
- opencv-python==4.10.0.84
- loralib==0.1.2
- pycocotools==2.0.8
- timm==1.0.9
- torch==2.5.1
- torchvision==0.20.1
- pandas==2.2.3
- tqdm==4.66.5
- gdown==5.2.0
- regex

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

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

Ohter Reminders & Common Problems

- When using **wget**, surround the url with quotation marks (“) to prevent running in background.
- Use **os.path.join** to deal with path as often as possible.
- Avoid making assumptions about the arguments (such as paths or filenames).
- If you train on GPU ids other than 0, remember to deal with the “map location” issue when you load model.

Deadline and Academic Honesty

- **Deadline: 2025/11/12 (Wed.) 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.

Reproducibility

- If we cannot execute your code, TAs will give you a chance to make minor modifications to your code. After you modify your code,
  - You can not train a new model as a modification.
  - If we can execute your code, you will still receive a 30% penalty in the score corresponding to the modified code.
  - If we still cannot execute your code, no point will be given.

DOs and DONTs for the TAs & Instructor

- Do NOT send messages to TAs via their individual emails / social media.
- 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.

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!
  - HW3 TAs are available on **Monday, Wednesday, and Thursday, 16:30–17:20, at MK-514**
- Post your question under HW3 discussion section on NTU COOL
- Contact TAs by e-mail: ntu-dlcv-2025-fall-ta@googlegroups.com
  - **Title should start with [DLCV 2025 Fall HW3]**
  - **Email with the wrong title may be filtered and not be replied**

Supplementary

Supplementary - Padding

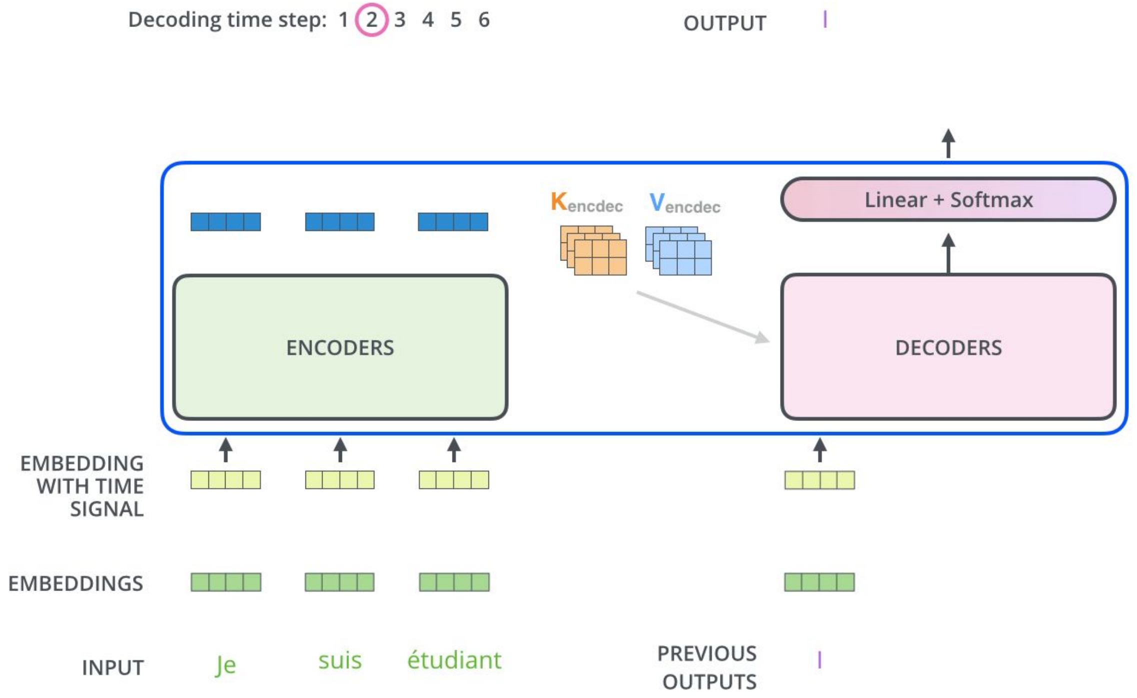
- Different ground truth texts vary in length after tokenization, yet within the same batch, they need to be of uniform length
- To achieve this, all ground truth ids are padded to the same length using the value 151643, which corresponds to “<|endoftext|>” (see p.21)
- When calculating cross entropy loss, **remember to ignore those padding tokens**

[64, 9592, 351]
[14595, 290]
[867, 10801, 8217, 290]

➡

[64, 9592, 351, 151643]
[14595, 290, 151643, 151643]
[867, 10801, 8217, 151643]

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