Q1: LLM Tuning

- Describe:

```
$ python3 -c 'import json; print(len(json.load(open("train.json", "r", encoding="utf-8"))))'
10000
```

  - How much training data did you use? (2%) Training data = 10000 (code above reveals)
  - How did you tune your model? (2%)
  - The core of **tuning process is QLoRA**, a memory-efficient fine-tuning technique. This approach combines two main concepts:
    - **4-bit Quantization:** The large pre-trained base model (Qwen/Qwen3-4B) is loaded in a quantized 4-bit precision format using the bitsandbytes library. This is configured in the build_model_and_tokenizer function, which calls get_bnb_config() to set up the quantization parameters (specifically, 4-bit NormalFloat type). This drastically **reduces the memory footprint of the base model**.
    - **Low-Rank Adaptation (LoRA):** Instead of training all the model‘s parameters, **small “adapter” matrices are added** to specific layers of the model’s transformer architecture. **The original model weights remain frozen**. The attach_lora function uses the peft library to inject these adapters into the attention mechanism‘s projection layers (q_proj, k_proj, v_proj, o_proj) and the feed-forward network layers (gate_proj, up_proj, down_proj). **This combination allows for the fine-tuning of a large language model on a single GPU with significantly less VRAM than would be required for full fine-tuning.**
  - A custom JsonSupervisedDataset class to process the training data. It structures the data for the causal language modeling task:
    - **Prompt Formatting**: Each data entry, consisting of an instruction and an answer, is formatted into a single sequence: BOS_token + prompt_text + answer_text + EOS_token.
    - **Masked Loss Function**: To ensure the model is only trained to generate the desired answer, the **loss is calculated exclusively on the answer_text portion** of the sequence. This is achieved by setting the **labels corresponding to the prompt tokens to -100**, a value that is ignored by the cross-entropy loss function in PyTorch. The _build_ids_and_labels method explicitly implements this logic.

Q1: LLM Tuning(cont.)

- Describe:
  - What hyper-parameters did you use? (2%)
    - Model: Qwen/Qwen3-4B
    - Optimizer: paged_adamw_32bit
    - learning_rate=1e-4 / scheduler: cosine / warmup_ratio: 0.05
    - Lora r=64, alpha=128, dropout=0.05

    (no weight decay as it ruin performance)
    - Batch Size: The effective batch size is 16

    (4 per device with 4 gradient accumulation steps).
    - Epochs: The model is trained for 1 epochs.

    (2 epochs for graphing)

```
--per_device_train_batch_size 4 \
--gradient_accumulation_steps 4 \
--num_train_epochs 1 \
--learning_rate 1e-4 \
--warmup_ratio 0.05 \
--weight_decay 0.00 \
--lora_r 64 \
--lora_alpha 128 \
--lora_dropout 0.05 \
```

Q1: LLM Tuning (cont.)

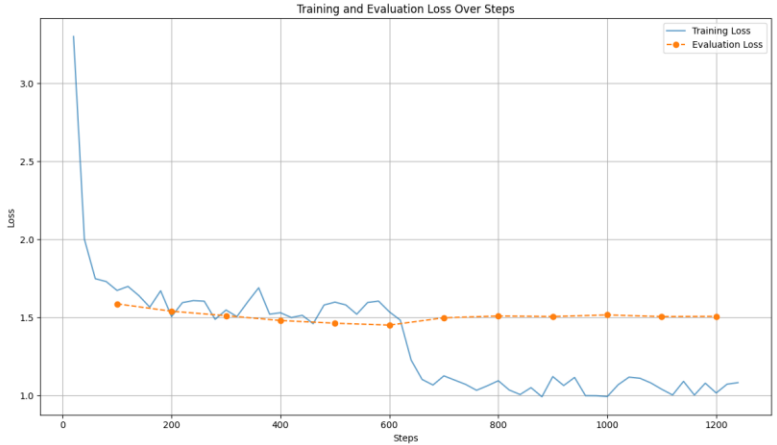
```
r14922159@cuda3:~/adlhw2$ ./run.sh
===== 步驟 1: 合併 QLoRA 模型 =====
`torch_dtype` is deprecated! Use `dtype` instead!
Loading checkpoint shards: 100%|
100%|
Mean perplexity: 6.15084375
```

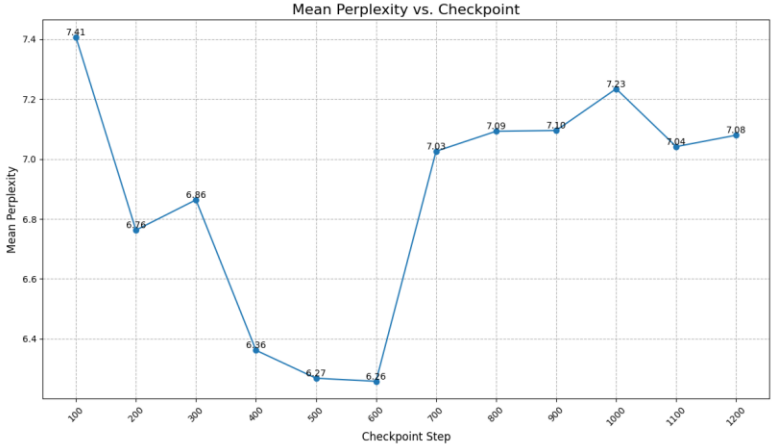
- Show your performance:
  - What is the final performance of your model on the public testing set? (2%)

    Mean perplexity: 6.15084375 (at checkpoint-625, end of epoch 1)
  - Plot the learning curve on the public testing set (2%) Two epoch were plotted.

The training and evaluation(public) loss is in the left image. The mean perplexity(public) is in the right image.

Both image indicates that overfitting start to occur after step 600 (about the end of first epoch.)





Q2: LLM Inference Strategies

```
# 執行 Python 腳本來合併模型
python3 pplNT.py \
    --test_data_path public_test.json

echo -e "\n✅ 評估完成！分數已輸出，且臨時檔案已清理。"
```

- Zero-Shot
  - What is your setting? How did you design your prompt? (1%)

a. The evaluation was performed using the original, pre-trained Qwen/Qwen3-4B model **without any fine-tuning.** The pplNT.py is the same as ppl.py to evaluate perplexity, instead that PEFT file requirement is removed. **Hence, PEFT file is no longer required by the bash/python.**

b. The objective was to assess **zero-shot translation performance** between Classical Chinese (文言文) and Modern Chinese (白話文) through in-context learning. The design is in the accompanying image. the logic for generating the prompts (get_prompt) in the utils.py file was kept constant(with LoRA and few-shot) and simple to ensure a fair comparison across all runs.

```
return f"翻譯任務。USER: {instruction} ASSISTANT:"
```

Q2: LLM Inference Strategies

Few-Shot (In-context Learning)

- What is your setting? How did you design your prompt? (1%)

The setting is the **same as zero-shot** which makes the PEFT file no longer required by the bash/python. **The design is in the accompanying image**. the logic for generating the prompts (get_prompt) in the utils.py file was kept constant and simple to ensure a fair comparison across all runs.

- How many in-context examples are utilized? How you select them? (1%)

I tested three different configurations for the number of in-context examples (k): **1-shot** (k=1), **3-shot** (k=3), **5-shot** (k=5). This approach allows for an analysis of how the model's performance scales with an increasing number of demonstrations provided in the context window.

Selection of Examples: The key characteristic of my example selection strategy was **bidirectional diversity**. The in-context examples were not limited to a single translation direction. Both direction: Classical Chinese to Modern Chinese translation examples and Modern Chinese to Classical Chinese translation examples were included.

```
examples = [
    (
        "翻譯成文言文：\n"
        "雅裏惱怒地說： 從前在福山田獵時，你誣陷獵官，現在又說這種話。\n"
        "答案：",
        "雅裏怒曰： 昔畋於福山，卿誣獵官，今復有此言。",
    ),
    (
        "沒過十天，鮑泉果然被拘捕。\n"
        "幫我把這句話翻譯成文言文",
        "後未旬，果見囚執。",
    ),
    (
        "辛未，命吳堅為左丞相兼樞密使，常楙參知政事。\n"
        "把這句話翻譯成現代文。",
        "初五，命令吳堅為左承相兼樞密使，常增為參知政事。",
    ),
]
example_prompt = "\n".join(
    f"USER: {inst}\nASSISTANT: {out}" for inst, out in examples
)
return (
    "翻譯任務。\n"
    f"{example_prompt}\n"
    f"USER: {instruction}\n"
    "ASSISTANT:"
)
```

Q2: LLM Inference Strategies(cont.)

- Comparison:
  - What's the difference between the results of zero-shot, few-shot, and LoRA? (2%)

Method	Type	Mean Perplexity
Zero-Shot	In-Context Learning	1386.82
Few-Shot (1-shot)	In-Context Learning	1106.69
Few-Shot (3-shot)	In-Context Learning	1064.02
Few-Shot (5-shot)	In-Context Learning	1557.4
LoRA	Fine-Tuning	6.26

I use mean perplexity to evaluate the result by different approaches.

1. **Zero-Shot Learning** acts as a performance baseline, but it is generally unreliable for tasks that require specific knowledge or formatting. **While the base model is a capable generalist, it has no specialized knowledge of this specific translation task.**

2. Few-Shot Learning is a useful technique that is better than nothing, but its effectiveness depends heavily on the number and quality of the examples. 5-shot result is fascinating because performance gets significantly worse than even the zero-shot attempt. This highlights the key limitation of in-context learning:
a. Context Clutter: Providing too many examples can confuse the model or "drown out" the actual instruction.
b. Example Quality: The specific examples chosen for the 5-shot prompt might have been less relevant or created conflicting patterns

3. Fine-Tuning (LoRA) is considered to most effective method by far. It produces a specialized model that is highly confident and accurate. **By updating its weights, it has become an expert on this specific task.**

Q3: Bonus: Try Llama3-Taiwan (8B) (2%)

- Llama-3-8b trained by traditional Chinese data, tune this model on the classical chinese data
- Describe your experimental settings:

(1) Key Parameters: The core configuration parameters, such as the lora_r, lora_alpha, and learning_rate, are as detailed in the accompanying image. The chosen optimizer for these experiments is paged_adamw_32bit.

(2) Although different models were utilized during the experiments, the logic for generating the prompts (the get_prompt function) in the utils.py file was kept constant to ensure a fair comparison across all runs.

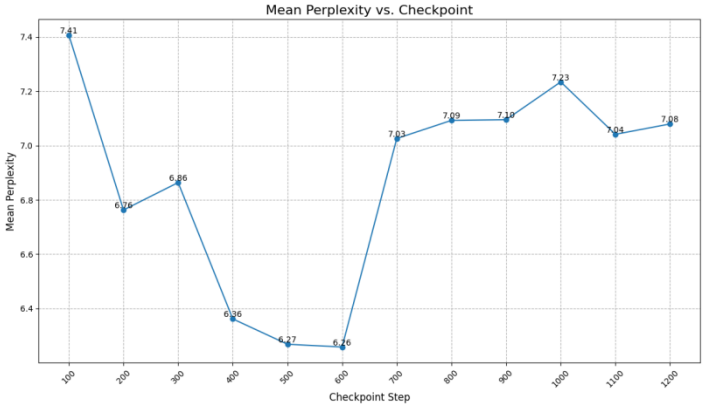
(3) Training Epochs: While a single training epoch (Epoch=1) is generally sufficient for fine-tuning the model, two epochs (Epoch=2) were used in this study to facilitate subsequent graphical analysis and data observation.

```
export NCCL_P2P_DISABLE="1"
export NCCL_IB_DISABLE="1"

SCRIPT_DIR="$(cd "$(dirname "${BASH_SOURCE[0]}")" && pwd)"

echo "Starting training ..."

python3 "$SCRIPT_DIR/train.py" \
    --base_model_path yentinglin/Llama-3.1-Taiwan-8B  \
    --train_file "$SCRIPT_DIR/train.json" \
    --val_file "$SCRIPT_DIR/public_test.json" \
    --output_dir "$SCRIPT_DIR/adapter" \
    --max_length 512 \
    --per_device_train_batch_size 4 \
    --gradient_accumulation_steps 4 \
    --num_train_epochs 2 \
    --learning_rate 1e-4 \
    --warmup_ratio 0.05 \
    --weight_decay 0.00 \
    --lora_r 64 \
    --lora_alpha 128 \
    --lora_dropout 0.05 \
    --logging_steps 20 \
    --eval_steps 100 \
    --save_steps 100 \
    --save_total_limit 100 \
    --seed 1234
```

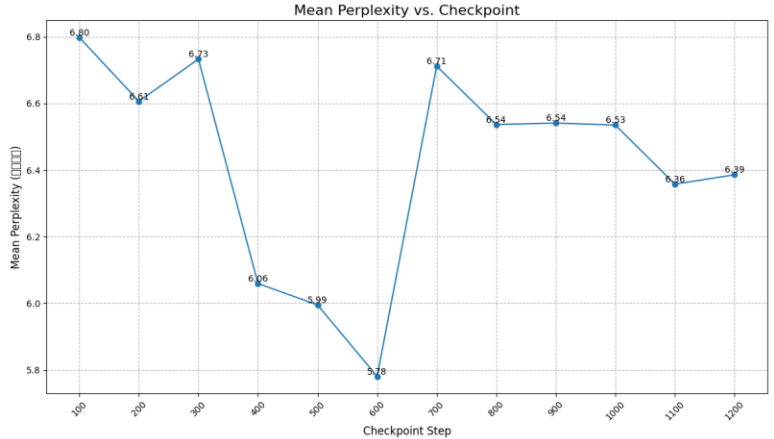
Q3: Bonus: Try Llama3-Taiwan (8B) (2%)(cont.)

- Compare the results to those obtained from your original methods
- **Superior Language Modeling:** The **yentinglin/Llama-3.1-Taiwan-8B** model, with a **lower Mean Perplexity of 5.78**, is demonstrably better at predicting the next word in the text compared to **Qwen/Qwen3-4B** (6.26). A lower perplexity generally indicates a more **fluent, confident, and accurate** language model.
- **Convergence Point:** Both models achieved their best performance (lowest perplexity) at checkpoint-600, which you noted is approximately the end of the first epoch. This suggests a **consistent training schedule** and that initial training significantly improved both models' general language understanding before diminishing returns might set in.

Qwen/Qwen3-4B



Llama-3.1-Taiwan-8B



Reference

1. https://huggingface.co/docs/peft/index
2. https://github.com/huggingface/peft
3. https://huggingface.co/blog/4bit-transformers-bitsandbytes
4. https://huggingface.co/docs/transformers/main_classes/quantization#advanced-use-cases
5. https://github.com/artidoro/qlora
6. https://github.com/huggingface/trl
7. LLM: Train.py/Plot.py/pplNT.py were bulid with the help of Gemini