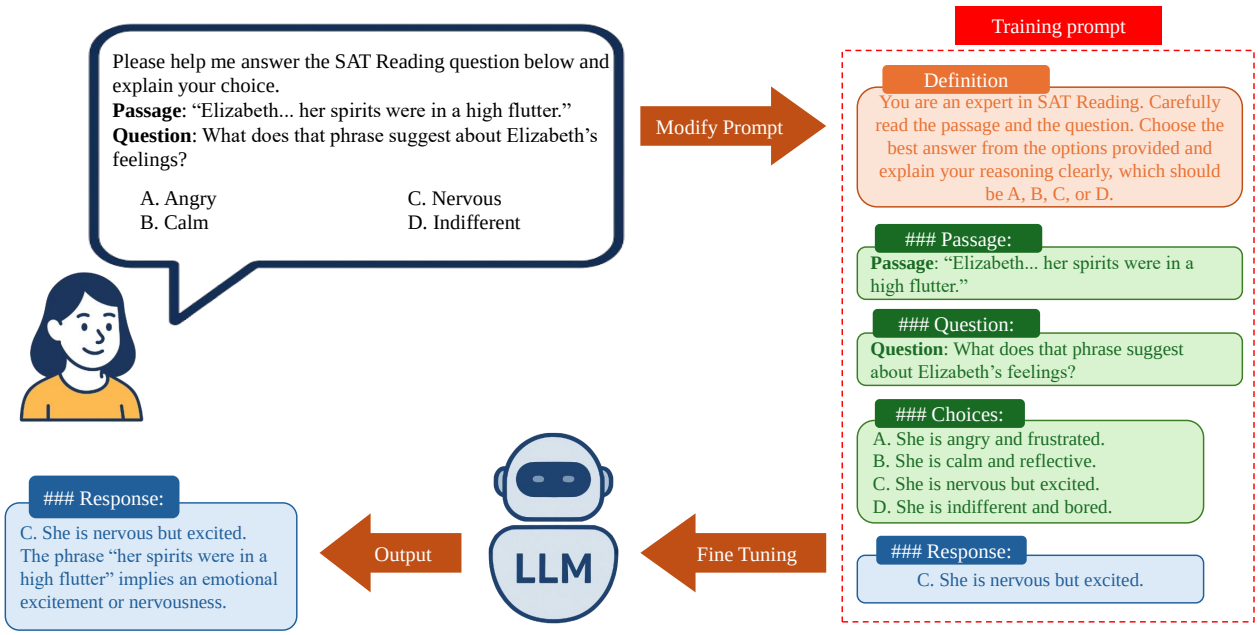
AI VIET NAM – AI COURSE 2024

# Tutorial: Boosting LLMs' SAT Reading Proficiency via Instruction Tuning

Dinh-Thang Duong, Yen-Linh Vu, Anh-Khoi Nguyen, Quang-Vinh Dinh

# I. Giới thiệu

**Instruction Tuning (IT)** là một trong những kỹ thuật training mô hình ngôn ngữ lớn (LLMs) rất quan trọng. Trong đó, IT giúp cải thiện khả năng của mô hình cũng như kiểm soát kết quả đầu ra. Là kiểu huấn luyện mô hình có giám sát từ bộ dữ liệu theo cặp (instruction-output), từ đó giúp mô hình thu hẹp khoảng cách giữa từ kế tiếp được sinh ra và sự chỉ dẫn của con người.

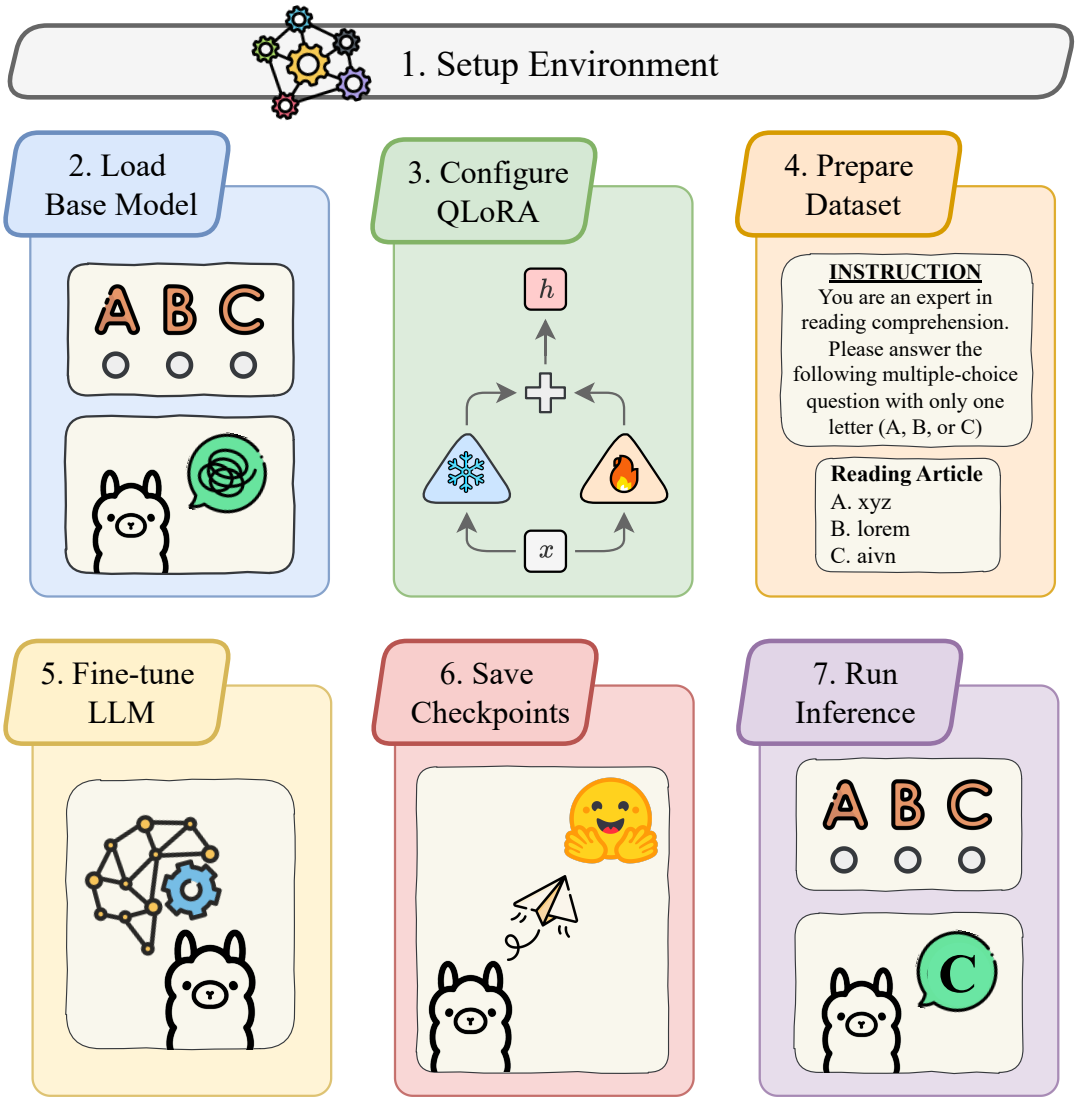



Hình 1: Instruction Fine-tuning mô hình ngôn ngữ lớn với dữ liệu trắc nghiệm.

Bài viết trình bày quá trình huấn luyện một mô hình ngôn ngữ lớn (LLM) với dữ liệu instruction để thực hiện các câu hỏi trắc nghiệm trong bài đọc hiểu thuộc kỳ thi SAT. Mục tiêu là xây dựng một mô hình có khả năng cải thiện độ chính xác trong các bài kiểm tra đọc hiểu SAT, đồng thời hỗ trợ nâng cao kỹ năng đọc hiểu tiếng Anh một cách tổng quát. Bài toán được định nghĩa với đầu vào và đầu ra như sau:

- **Input:** Một câu prompt với lời hướng dẫn (instruction) để LLM thực hiện một bài bài đọc hiểu SAT và lựa chọn phương án đúng.
- **Output:** Lời phản hồi từ mô hình, trong trường hợp này là một trong các phương án trắc nghiệm đã được mô tả trong prompt.

Thông qua nội dung mô tả trên, pipeline các bước xử lý trong bài này được minh họa theo hình dưới đây:




Hình 2: Fine-tuning mô hình ngôn ngữ lớn với dữ liệu instruction giải toán trắc nghiệm.

**Trong đó:**

1. **Setup Environment:**
   Đầu tiên là cài đặt một số thư viện sau để có thể chạy được các mô hình ngôn ngữ lớn từ thư viện HuggingFace:

```
!pip install -q -U bitsandbytes
!pip install -q -U datasets
!pip install -q -U git+https://github.com/huggingface/transformers.git
!pip install -q -U git+https://github.com/huggingface/peft.git
!pip install -q -U git+https://github.com/huggingface/accelerate.git
!pip install -q -U loralib
!pip install -q -U einops
```

   Sau khi cài đặt hoàn tất, bước tiếp theo là import các thư viện đã tải cũng như một số thư viện khác để phục vụ cho chương trình:

```
import json
import os
import bitsandbytes as bnb
import torch
import torch.nn as nn
import transformers

from pprint import pprint
from tqdm import tqdm
from datasets import load_dataset, Dataset

from peft import (
    LoraConfig,
    PeftConfig,
    PeftModel,
    get_peft_model,
    prepare_model_for_kbit_training
)
from transformers import (
    AutoConfig,
    AutoModelForCausalLM,
    AutoTokenizer,
    BitsAndBytesConfig
)
```

2. **Load Base Model:**
   Mô hình ngôn ngữ lớn được sử dụng trong bài này là Llama đến từ Meta (Facebook), mô hình là phiên bản 3.1-Instruct với tổng cộng 8 tỷ tham số. Tuy nhiên, mô hình này cần xin quyền truy cập từ Meta. Để làm điều đó, trước tiên hãy vào đường link này, hãy đăng nhập và xin cấp quyền sau khi điền đủ các thông tin được yêu cầu. Thông thường, Meta sẽ duyệt trong vài phút kể từ lúc gửi. Sau khi được cấp quyền, hãy lấy **User Access Tokens** bằng cách nhấn vào link này (đã đăng nhập trước), chọn "Create New Token", chọn "Read", đặt tên và nhấn "Create token", và thay Access Tokens này vào đoạn code sau để đăng nhập HuggingFace trên notebook:

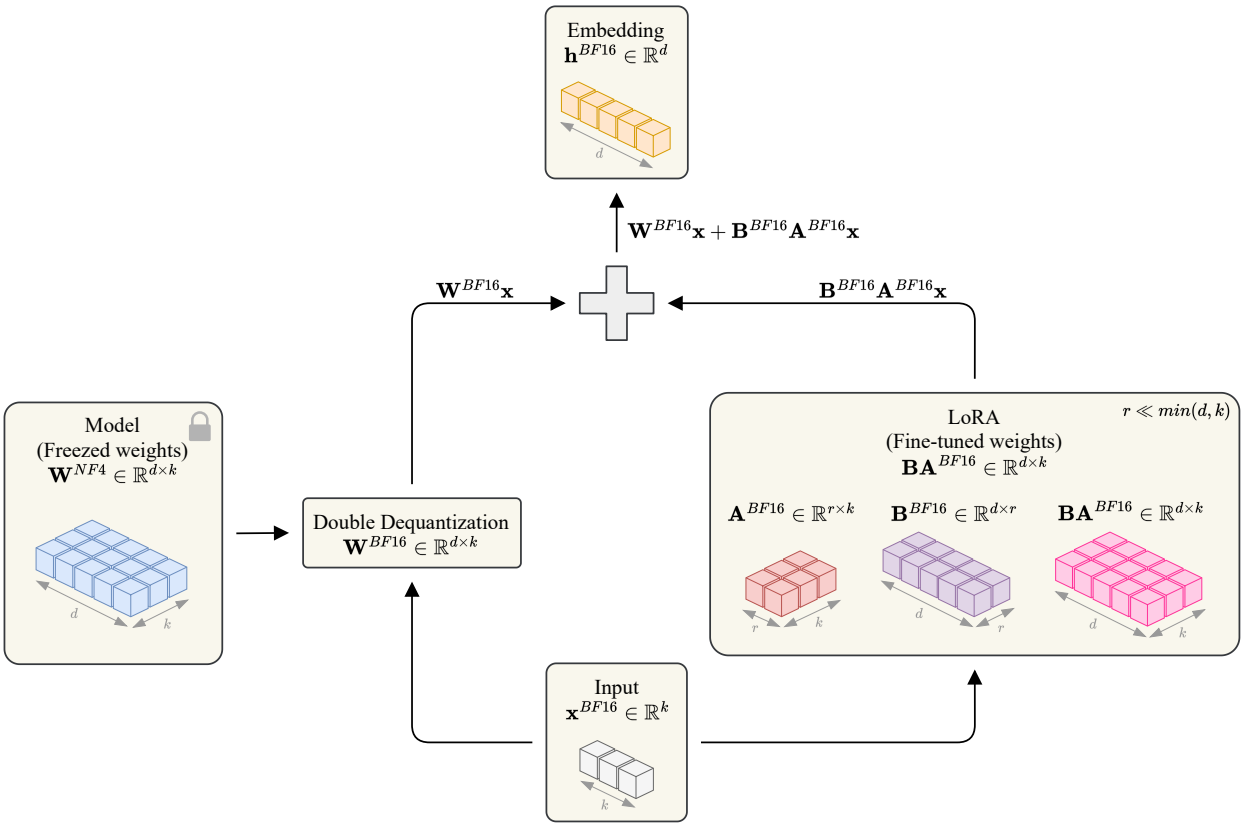
```python
from huggingface_hub import login

login(token="hf_xxxxxxxxxxxxxxxx")  # your access token
```

Khi đã đăng nhập thành công, tiến hành khởi tạo mô hình Llama như sau:

```python
MODEL_NAME = "meta-llama/Llama-3.1-8B-Instruct"
# MODEL_NAME = "meta-llama/Llama-3.2-3B-Instruct"

bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.bfloat16,
    bnb_4bit_use_double_quant=True
)

model = AutoModelForCausalLM.from_pretrained(
    MODEL_NAME,
    quantization_config=bnb_config,
    device_map="auto",
    trust_remote_code=True
)

tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)
tokenizer.pad_token = tokenizer.eos_token

model.gradient_checkpointing_enable()
```

Trong đoạn code trên, mô hình được tải với kỹ thuật lượng tử hóa 4-bit (NF4) thông qua thư viện BitsAndBytes, giúp giảm dung lượng và tối ưu hóa hiệu suất tính toán trên GPU. Tokenizer phù hợp với mô hình cũng được tải về và thiết lập token padding bằng ký tự kết thúc chuỗi (EOS). Tiếp theo, gradient checkpointing được kích hoạt để tiết kiệm bộ nhớ khi huấn luyện. Sau đó, hàm `prepare_model_for_kbit_training()` được áp dụng nhằm điều chỉnh mô hình sẵn sàng cho quá trình fine-tuning.

3. **Configure QLoRA:**
   Kế đến, kỹ thuật LoRA được áp dụng lên các module attention (`q_proj` và `v_proj`) để fine-tune mô hình nhanh chóng, nhẹ nhàng, nhưng vẫn hiệu quả.




Hình 3: Trực quan về kỹ thuật QLoRA.

```python
model = prepare_model_for_kbit_training(model)

peft_config = LoraConfig(
    r=8,
    lora_alpha=16,
    target_modules=[
        "q_proj",
        "v_proj",
    ],
    lora_dropout=0.05,
    bias="none",
    task_type="CAUSAL_LM"
)

model = get_peft_model(model, peft_config)
```

4. **Prepare Dataset:**
Để thực hiện nhiệm vụ "đọc bài đọc và trả lời câu hỏi trắc nghiệm", mô hình LLaMA được fine-tune trên tập dữ liệu `emozilla/sat-reading` từ Hugging Face. Bộ dữ liệu này bao gồm các đoạn văn và câu hỏi trích từ phần đọc hiểu của mười bài thi thử SAT được công bố công khai. Mỗi mục dữ liệu chứa đoạn văn, câu hỏi, bốn lựa chọn trả lời, và đáp án đúng. Một số câu hỏi yêu cầu tham chiếu đến dòng cụ thể trong văn bản, được đánh dấu bằng trường boolean `requires_line`. Các mục liên quan đến biểu đồ và bảng biểu đã bị

loại bỏ nhằm đảm bảo tính nhất quán của tập dữ liệu. Dữ liệu được chia thành ba phần: tập huấn luyện (298 mục), tập kiểm định (39 mục), và tập kiểm tra (38 mục), tổng cộng 375 mục. Việc tải và đọc dữ liệu được thực hiện như sau:

```
data = load_dataset("emozilla/sat-reading")
```

Dưới đây là một vài ví dụ điển hình trong tập dữ liệu:

| **id** | **text (shortened)** | **answer** | **requires line** |
|---|---|---|---|
| sat-practice_7-question_3 | SAT READING COMPREHENSION TEST<br>This passage is adapted from ...<br>Question 3: Which statement best describes a technique the narrator uses to represent Silas's character before he adopted Eppie?<br>A) ...<br>B) ...<br>C) ...<br>D) ...<br>Answer: | A | false |
| sat-practice_7-question_10 | SAT READING COMPREHENSION TEST<br>This passage is adapted from ...<br>Question 10: As used in line 65, "fine" most nearly means<br>A) acceptable<br>B) delicate<br>C) ornate<br>D) keen<br>Answer: | D | true |

Tuy nhiên, để đảm bảo cấu trúc rõ ràng cho các mẫu huấn luyện, hai hàm được định nghĩa nhằm trích xuất thông tin từ văn bản gốc, gồm:

- `extract_sections()` thực hiện phân tách nội dung đầu vào thành các phần rõ ràng như đoạn văn, câu hỏi, các lựa chọn, và đáp án dưới dạng ký tự (A, B, C, D).
- `map_answer()` được dùng để ánh xạ ký tự đáp án sang nội dung đầy đủ tương ứng trong danh sách lựa chọn.

```
def extract_sections(text):
    sections = {
        "passage": "",
        "question": "",
        "choices": [],
        "answer_letter": ""
    }

    answer_part = text.split("Answer:")[-1].strip()
    sections["answer_letter"] = answer_part[0] if answer_part else ""

    content = text.split("SAT READING COMPREHENSION TEST")[-1].split("Answer:")[0]
```

```
    blocks = [b.strip() for b in content.split("\n\n") if b.strip()]

    passage_lines = []
    for line in blocks:
        if line.startswith("Question"):
            break
        passage_lines.append(line)
    sections["passage"] = "\n".join(passage_lines).strip()

    for block in blocks:
        if block.startswith("Question"):
            q_part = block.split(")", 1) if ")" in block else (block, "")
            sections["question"] = q_part[-1].split("\n")[0].strip()
            sections["choices"] = [line.strip() for line in block.split("\n")[1:]
                                   if line.startswith(("A)", "B)", "C)", "D)"))]
    return sections


def map_answer(text, letter):
    sections = extract_sections(text)
    for choice in sections["choices"]:
        if choice.startswith(f"{letter})"):
            return choice
    return letter
```

Sau đó, dùng hai hàm vừa định nghĩa để xây dựng một hàm tạo prompt với cấu trúc như mong muốn bằng cách sau:

```
LLAMA3_SYSTEM_PROMPT = """You are a helpful AI assistant developed by Meta. Respond
                                          safely and accurately."""

def generate_prompt(text, answer_letter):
    sections = extract_sections(text)
    choices_text = "\n".join(sections['choices'])

    return [
        {
            "role": "system",
            "content": LLAMA3_SYSTEM_PROMPT
        },
        {
            "role": "user",
            "content": f"""Read the passage and answer the question.

### Passage:
{sections["passage"]}

### Question:
{sections["question"]}

### Choices:
{choices_text}

Respond with ONLY the letter and full text of the correct answer."""
        },
```

```
        {
            "role": "assistant",
            "content": map_answer(text, answer_letter)
        }
    ]
```

Tiếp tục thực hiện hàm tokenize để chuẩn bị đầu vào phù hợp cho mô hình. Cụ thể là:

- Hàm `generate_and_tokenize_prompt()` nhận đầu vào là một cặp (`user_input`, `answer`) và sinh ra prompt hoàn chỉnh thông qua hàm `generate_prompt()`.
- Prompt sau đó được xử lý định dạng bằng `tokenizer.apply_chat_template()` để đảm bảo phù hợp với định dạng hội thoại mà tokenizer yêu cầu.
- Hàm `tokenizer()` được dùng để chuyển prompt thành tensor `input_ids`, cùng với `attention_mask` phục vụ cho quá trình huấn luyện.
- Cuối cùng, nhãn huấn luyện (labels) được gán bằng chính `input_ids` để mô hình học theo dạng tự hồi tiếp (causal language modeling).

```
def generate_and_tokenize_prompt(user_input, answer):
    try:
        full_prompt = generate_prompt(user_input, answer)

        prompt_str = tokenizer.apply_chat_template(
            full_prompt,
            tokenize=False,
            add_generation_prompt=False
        )

        tokenized = tokenizer(
            prompt_str,
            padding="max_length",
            truncation=True,
            max_length=1024,
            return_tensors="pt"
        )

        input_ids = tokenized["input_ids"][0]
        labels = input_ids.clone()

        return {
            "input_ids": input_ids,
            "attention_mask": tokenized["attention_mask"][0],
            "labels": labels
        }

    except Exception as e:
        print(f"Error processing sample: {e}")
        return None
```

Với các hàm đã được chuẩn bị, bước cuối cùng trong phần này là xây dựng bộ dữ liệu huấn luyện theo các bước sau:

- Duyệt qua từng mẫu dữ liệu trong tập huấn luyện gốc.

- Thực hiện tiền xử lý văn bản bằng cách loại bỏ tiêu đề thừa và chuẩn hóa đáp án bằng hàm `map_answer()`.
- Áp dụng hàm `generate_and_tokenize_prompt()` để sinh ra đầu vào tokenized cho từng mẫu.
- Cuối cùng, tập dữ liệu được chia thành hai phần: tập huấn luyện (`train_dataset`) và tập đánh giá (`eval_dataset`) theo tỷ lệ 90/10.

```
training_samples = []
for sample in tqdm(data["train"]):
    try:
        processed_text = sample["text"].replace("SAT READING COMPREHENSION TEST", "").
                                                strip()
        processed_answer = map_answer(sample["text"], sample["answer"].strip())

        tokenized_sample = generate_and_tokenize_prompt(processed_text, processed_answer
                                                )
        if tokenized_sample is not None:
            training_samples.append(tokenized_sample)
    except Exception as e:
        print(f"Skipping invalid sample: {e}")

training_samples = [s for s in training_samples if s is not None]

train_samples, val_samples = train_test_split(training_samples, test_size=0.1,
                                                random_state=42)
train_dataset = Dataset.from_list(train_samples)
eval_dataset = Dataset.from_list(val_samples)
```

<|begin_of_text|><|start_header_id|>system<|end_header_id|>

Cutting Knowledge Date: December 2023
Today Date: 26 Jul 2024

You are a helpful AI assistant developed by Meta. Respond safely and accurately.
<|eot_id|><|start_header_id|>user<|end_header_id|>

Read the passage and answer the question.

### Passage:
This passage is adapted from Elizabeth Cady Stanton's address to the 1869 Woman Suffrage Convention in Washington, DC. I urge a sixteenth amendment, because

. . .

and not with her consent would one drop of blood ever be shed, one life sacrificed in vain.

### Question:
Stanton claims that which of the following was a relatively recent historical development?

### Choices:
A) The control of society by men
B) The spread of war and injustice
C) The domination of domestic life by men
D) The acknowledgment of women's true character

Respond with ONLY the letter and full text of the correct answer.<|eot_id|>
<|start_header_id|>assistant<|end_header_id|>

D) The acknowledgment of women's true character<|eot_id|>

Hình 4: Trực quan một prompt đã được tổ chức lại.

5. **Fine-tune LLM:**
Sau khi dữ liệu huấn luyện được chuẩn bị, quá trình huấn luyện mô hình được thực hiện theo các bước sau:

- Lớp **LogLossCallback** được định nghĩa nhằm ghi lại giá trị hàm mất mát (loss) sau mỗi bước logging, hỗ trợ theo dõi tiến trình huấn luyện theo thời gian thực.
- Các tham số huấn luyện được thiết lập thông qua **TrainingArguments**, bao gồm:
  - Batch size nhỏ (1), kết hợp với **gradient accumulation** để phù hợp với mô hình lớn.
  - Sử dụng optimizer 8-bit (`paged_adamw_8bit`) nhằm tiết kiệm bộ nhớ và tăng hiệu suất.
  - Scheduler dạng cosine với tỉ lệ warmup ban đầu.
  - Đánh giá và lưu mô hình mỗi 50 bước thông qua `eval_steps` và `save_steps`.
  - Cấu hình tự động tải lại checkpoint có hiệu suất tốt nhất theo metric loss.

- Hàm `DataCollatorForLanguageModeling()` được sử dụng để xử lý đầu vào theo chuẩn causal language modeling (không áp dụng masking).
- Trước khi huấn luyện, gradient cho đầu vào được kích hoạt và mô hình được biên dịch bằng `torch.compile()` để tối ưu hiệu suất trên phần cứng hiện đại.
- Quá trình huấn luyện được thực hiện thông qua lệnh gọi `Trainer.train()`.

```
class LogLossCallback(TrainerCallback):
    def on_log(self, args, state, control, logs=None, **kwargs):
        if logs is not None and "loss" in logs:
            print(f"Step {state.global_step} - Loss: {logs['loss']:.4f}")

training_args = TrainingArguments(
    per_device_train_batch_size=1,
    gradient_accumulation_steps=2,
    num_train_epochs=2,
    learning_rate=2e-4,
    fp16=True,
    save_total_limit=3,
    logging_steps=10,
    output_dir="llama3-8b-sat-reading",
    optim="paged_adamw_8bit",
    lr_scheduler_type="cosine",
    warmup_ratio=0.05,
    eval_strategy="steps",
    eval_steps=50,
    save_strategy="steps",
    save_steps=50,
    load_best_model_at_end=True,
    metric_for_best_model="loss",
    greater_is_better=False,
    report_to="none",
    remove_unused_columns=False
)

data_collator = DataCollatorForLanguageModeling(
    tokenizer=tokenizer,
    mlm=False,
    pad_to_multiple_of=8
)

trainer = Trainer(
    model=model,
    train_dataset=train_dataset,
    eval_dataset=eval_dataset,
    args=training_args,
    data_collator=data_collator,
    callbacks=[LogLossCallback()]
)

model.config.use_cache = False
model.enable_input_require_grads()
model = torch.compile(model)

trainer.train()
```

6. **Save Checkpoints:**
   Khi quá trình huấn luyện hoàn tất, mô hình được lưu để sử dụng hoặc chia sẻ sau này. Cụ thể:

   - Mô hình sau huấn luyện được lưu cục bộ vào thư mục `"trained-model"` bằng hàm `save_pretrained()`.
   - Sau đó, mô hình được đẩy lên Hugging Face Hub thông qua hàm `push_to_hub()`, giúc dễ dàng chia sẻ và tái sử dụng trong các dự án khác.
   - Để thực hiện bước này, người dùng cần có tài khoản trên Hugging Face và phải cung cấp `use_auth_token=True` để xác thực quyền truy cập.

```
model.save_pretrained("trained-model")
PEFT_MODEL = "your_huggingface_user_name/instructionTuning-llama-3-1-8B-SAT-reading-
                                                    solver"
model.push_to_hub(PEFT_MODEL, use_auth_token=True)
```

7. **Run Inference:**
   Cuối cùng, để đánh giá mô hình sau huấn luyện, quá trình tải mô hình và thực hiện suy luận (inference) trên tập kiểm tra được tiến hành như sau:

   - Mô hình được tải từ Hugging Face Hub thông qua ID đã định nghĩa trước trong biến `PEFT_MODEL`
   - Tiếp theo là định nghĩa lại các thành phần liên quan đến mô hình tương tự như các đoạn code đã thực hiện bên trên.
   - Sau khi tải mô hình và tokenizer, mô hình được gắn adapter PEFT thông qua hàm `PeftModel.from_pretrained()` để khôi phục các tham số đã fine-tuned.

```
PEFT_MODEL = "your_huggingface_user_name/instructionTuning-llama-3-1-8B-SAT-reading-
                                                    solver"

config = PeftConfig.from_pretrained(PEFT_MODEL)

bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_use_double_quant=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.bfloat16
)

model = AutoModelForCausalLM.from_pretrained(
    config.base_model_name_or_path,
    quantization_config=bnb_config,
    device_map="auto",
    trust_remote_code=True
)

tokenizer = AutoTokenizer.from_pretrained(config.base_model_name_or_path)
tokenizer.pad_token = tokenizer.eos_token

model = PeftModel.from_pretrained(model, PEFT_MODEL)
```

Khi đã khôi phục mô hình, quá trình định dạng prompt và sinh câu trả lời được thực hiện để đánh giá khả năng suy luận của mô hình trên các mẫu kiểm tra, chi tiết như sau:

- Trong hàm `predict()`, prompt đầu vào được định dạng lại theo chuẩn hội thoại bằng `tokenizer.apply_chat_template`, sau đó thực hiện sinh câu trả lời bằng hàm `model.generate()` với các tham số cấu hình sinh (ví dụ: không sampling, temperature = 0.0, giới hạn số token đầu ra).
- Hàm `extract_answer()` được dùng để trích xuất câu trả lời đầy đủ từ đầu ra của mô hình, và được so sánh với đáp án gốc sau khi ánh xạ bằng `map_answer()`.
- Vòng lặp đánh giá thực hiện suy luận trên 10 mẫu đầu tiên của tập kiểm tra, in ra nội dung câu hỏi, các lựa chọn, dự đoán từ mô hình, đáp án đúng và kết quả đúng/sai.

```
generation_config = GenerationConfig(
    max_new_tokens=64,
    temperature=0.0,
    top_p=1.0,
    do_sample=False,
    repetition_penalty=1.0,
    eos_token_id=tokenizer.eos_token_id,
    pad_token_id=tokenizer.eos_token_id
)

def predict(text):
    messages = format_test_prompt(text)

    prompt_text = tokenizer.apply_chat_template(
        messages,
        add_generation_prompt=True,
        tokenize=False
    )

    inputs = tokenizer(prompt_text, return_tensors="pt").to(model.device)

    with torch.no_grad():
        outputs = model.generate(
            input_ids=inputs["input_ids"],
            attention_mask=inputs["attention_mask"],
            generation_config=generation_config
        )

    output_text = tokenizer.decode(outputs[0], skip_special_tokens=True)
    return extract_answer(output_text)

for i in range(10):
    print("="*100)
    sample = data["test"][i]
    input_text = sample["text"]
    true_answer = sample["answer"].strip()

    predicted_answer = predict(input_text)

    true_answer_full = map_answer(input_text, true_answer)

```

```
    pred_choice = extract_choice_letter(predicted_answer)
    true_choice = extract_choice_letter(true_answer_full)

    print(f"### Sample {i+1}")
    print(f"[Question]\n{extract_sections(input_text)["question"]}")
    print(f"[Choices]\n{extract_sections(input_text)["choices"]}")
    print(f"\n[Model Prediction]\n{predicted_answer}")
    print(f"\n[Ground Truth]\n{true_answer_full}")
    print(f"""\nResult: {"CORRECT" if pred_choice == true_choice else "INCORRECT"}""")
    print("="*100 + "\n")
```

# II. Câu hỏi trắc nghiệm

1. Trong ngữ cảnh về mô hình ngôn ngữ lớn (LLMs), Instruction Tuning được hiểu như thế nào?
   (*a*) Huấn luyện mô hình trên task mới mà không cần mẫu dữ liệu nào.
   (*b*) Điều chỉnh kết quả của mô hình trong quá trình deploy.
   (*c*) Huấn luyện mô hình để tuân theo các yêu cầu cụ thể (instruction).
   (*d*) Giảm kích thước của mô hình để tăng độ hiệu quả.

2. Instruction Tuning là một dạng của kiểu học gì?
   (*a*) Supervised Learning.
   (*b*) Self-supervised Learning.
   (*c*) Unsupervised Learning.
   (*d*) Reinforcement Learning.

3. Khi thực hiện Instruction Tuning, hàm loss nào sau đây có thể được sử dụng?
   (*a*) Mean Squared Error (MSE).
   (*b*) Hinge Loss.
   (*c*) Kullback-Leibler Divergence.
   (*d*) Cross-Entropy Loss.

4. Mệnh đề sau đúng hay sai: "Ta luôn luôn nên áp dụng Instruction Tuning để giảm thiểu chi phí tính toán trong quá trình training"?
   (*a*) Đúng.
   (*b*) Sai.

5. Trong LLMs, khái niệm prompt được hiểu như thế nào?
   (*a*) Một phần mềm bổ trợ giúp tối ưu hóa hiệu suất của mô hình.
   (*b*) Một kỹ thuật mã hóa thông tin riêng tư trong quá trình đào tạo mô hình.
   (*c*) Một câu hỏi hoặc yêu cầu mà người dùng đưa ra cho mô hình.
   (*d*) Một thuật toán đặc biệt để phân loại dữ liệu đầu vào.

6. Trong LLMs, ta nên áp dụng kỹ thuật nào sau đây để cải thiện khả năng thực hiện một task cụ thể nào đó của mô hình mà không cần training?
   (*a*) Sử dụng kỹ thuật Transfer Learning.
   (*b*) Ứng dụng khả năng Zero-shot Learning của mô hình.
   (*c*) Lập trình thủ công tại bước hậu xử lý cho mỗi task.
   (*d*) Mở rộng bộ dữ liệu training.

7. Kỹ thuật prompting nào dưới đây cung cấp cho mô hình chỉ một ví dụ về task cần làm?
   (*a*) One-shot Learning.
   (*b*) Few-shot Learning.
   (*c*) Continuous Learning.
   (*d*) Transfer Learning.

8. Câu nào sau đây mô tả đúng về kỹ thuật Parameter Efficient Fine-tuning (PEFT)?
   (*a*) Một kỹ thuật dùng để huấn luyện mô hình trên bộ dữ liệu cực lớn.
   (*b*) Một kỹ thuật liên quan đến việc cập nhật một phần nhỏ tham số của mô hình khi huấn luyện.
   (*c*) Một kỹ thuật huấn luyện dành riêng cho các mô hình có kích thước nhỏ (dưới 1 tỷ tham số).
   (*d*) Một kỹ thuật để tăng chi phí tính toán của mô hình.

9. Mệnh đề sau đúng hay sai: "Low-Rank Adaptation (LoRA) là một kỹ thuật về PEFT."?
   (*a*) Đúng.
   (*b*) Sai.

10. So với LoRA, QLoRA có điểm gì khác biệt gì trong việc huấn luyện LLMs?
    (*a*) QLoRA lượng tử hóa (quantize) tham số mô hình; LoRA thì không.
    (*b*) QLoRA sử dụng nhiều tham số mô hình; LoRA ít hơn.
    (*c*) QLoRA tối ưu khả năng tổng quát của mô hình; LoRA tối ưu trên một task cụ thể.
    (*d*) QLoRA giảm kích thước mô hình; LoRA tăng lên.

# III. Phụ lục

1. **Datasets:** Các file dataset được đề cập trong bài có thể được tải tại đây.
2. **Hint:** Các file code gợi ý có thể được tải tại đây.
3. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải tại đây (Lưu ý: Sáng thứ 6 khi hết deadline phần nội dung này, ad mới copy các tài liệu bài giải nêu trên vào đường dẫn).
4. **Demo:** Web demo và mã nguồn của ứng dụng có thể được truy cập tại đây.
5. **Rubric:**

| Mục | Kiến Thức | Đánh Giá |
|---|---|---|
| I. | - Kiến thức về mô hình lớn (LLMs).<br>- Kiến thức về kỹ thuật Instruction Tuning trong LLMs.<br>- Kiến thức về thư viện PyTorch và HuggingFace.<br>- Triển khai PEFT (LoRA) và thiết lập các tham số liên quan đến LoRA. | - Nắm được lý thuyết về mô hình lớn (LLMs) và kỹ thuật Instruction Tuning trong LLMs.<br>- Nắm được lý thuyết cơ bản về kỹ thuật LoRA.<br>- Có khả năng cài đặt và thực hiện Instruction Tuning mô hình lớn trên bộ dữ liệu bất kì, trong trường hợp này về bộ dữ liệu IELTS Reading. |

***- Hết -***