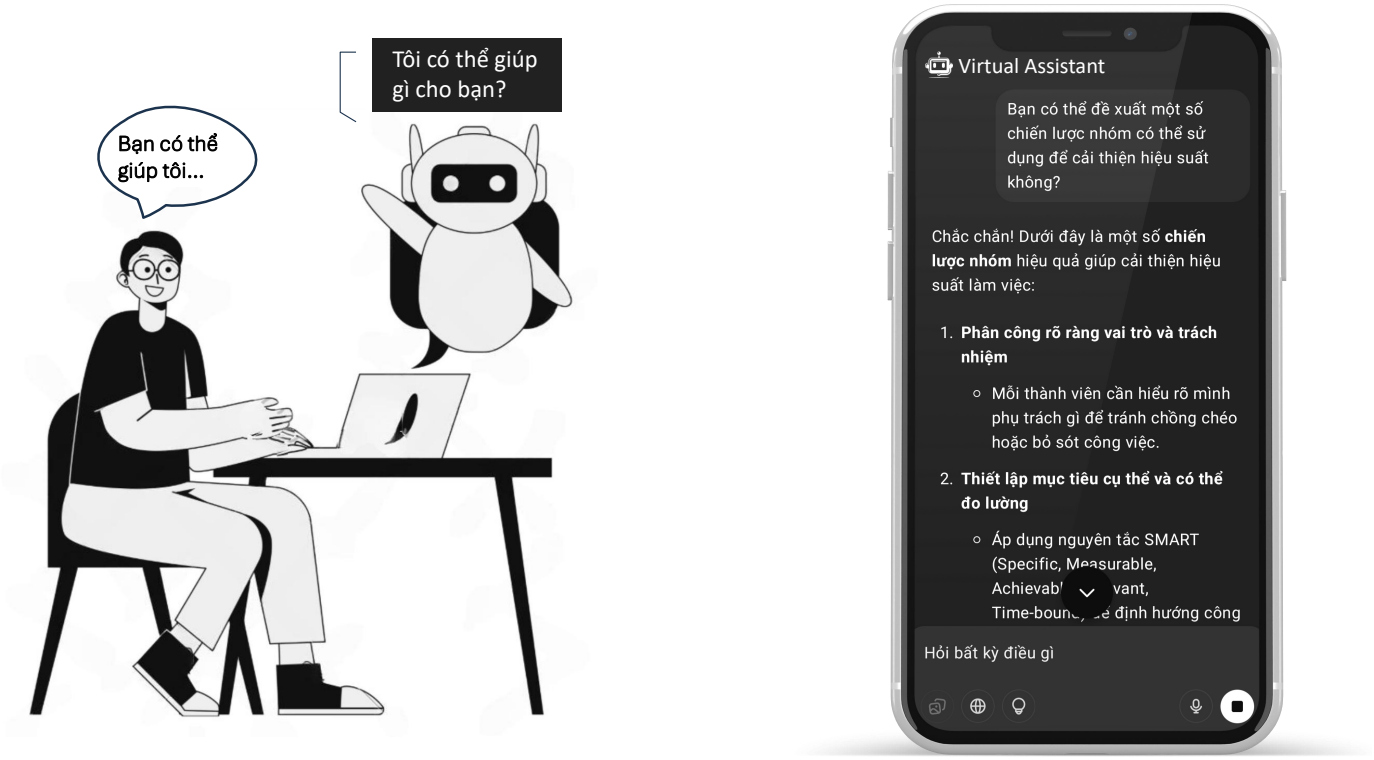
AI VIET NAM – AI COURSE 2024

Project: Vietnamese Chatbot using Large Language Models

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

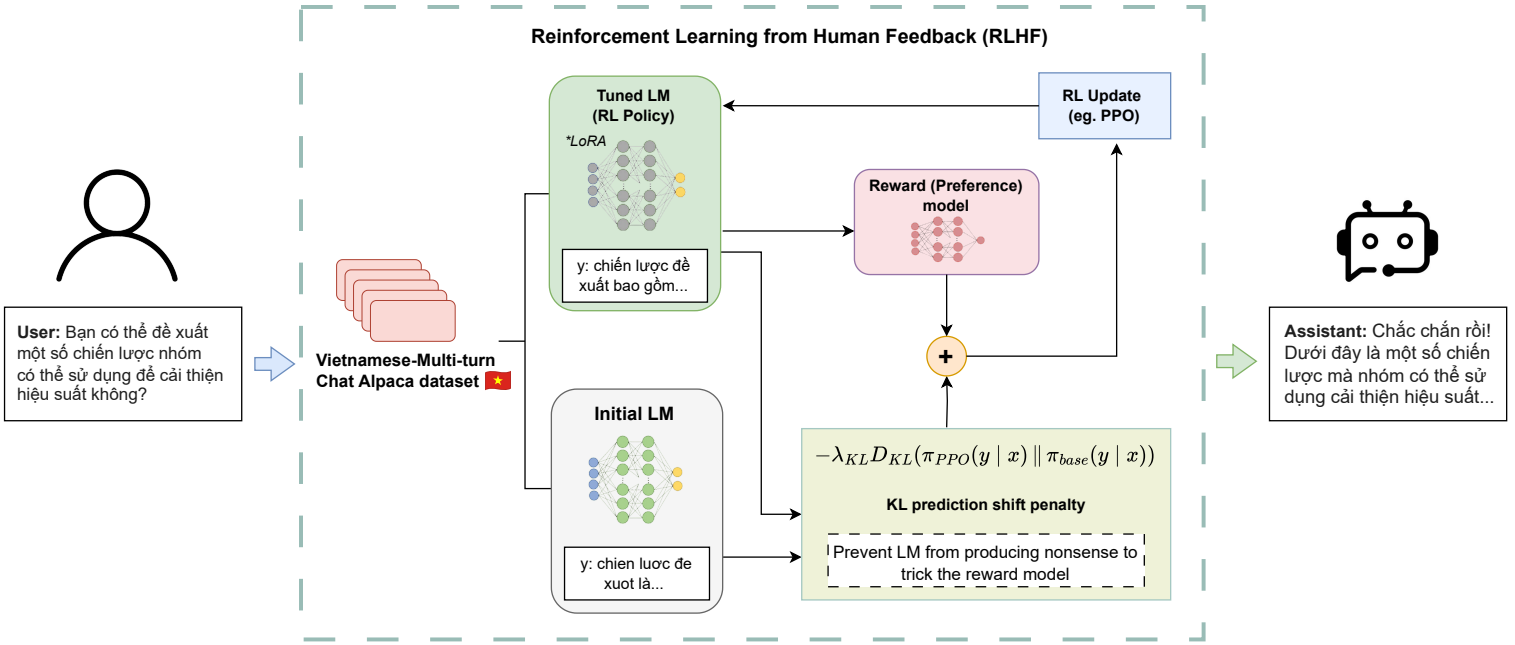
I. Giới thiệu

Chatbot là một hệ thống phần mềm cho phép con người tương tác với máy thông qua ngôn ngữ tự nhiên, thường được ứng dụng để hỗ trợ hỏi đáp, tư vấn hoặc trò chuyện thường nhật. Với sự phát triển mạnh mẽ của các mô hình ngôn ngữ lớn (Large Language Models - LLMs), việc xây dựng chatbot ngày càng trở nên khả thi và hiệu quả hơn. Các LLM hiện đại có khả năng tạo sinh văn bản linh hoạt và giàu ngữ nghĩa, phản hồi theo nhiều ngữ cảnh đa dạng chỉ từ các lời nhắc đơn giản của người dùng.



Hình 1: Minh họa về một ứng dụng Chatbot tiếng Việt.

Tuy nhiên, việc đảm bảo chất lượng phản hồi vẫn là một thách thức, nhất là trong hội thoại tiếng Việt thường nhật – nơi câu trả lời cần chính xác, tự nhiên và lịch sự. Trong bài viết này, chúng ta sẽ kết hợp mô hình ngôn ngữ lớn (LLMs) đã được huấn luyện trên dữ liệu hội thoại với kỹ thuật Reinforcement Learning from Human Feedback (RLHF) để xây dựng một chatbot tiếng Việt thân thiện, hiệu quả và phù hợp với nhu cầu sử dụng hàng ngày.



Hình 2: Minh họa xây dựng Chatbot tiếng Việt bằng RLHF.

Dù các mô hình LLMs sau giai đoạn tiền huấn luyện có thể sinh phản hồi linh hoạt và tự nhiên, nhưng để phản hồi thực sự phù hợp với ngữ cảnh và kỳ vọng người dùng, cần một bước tinh chỉnh bổ sung. RLHF hỗ trợ mô hình học cách ưu tiên các phản hồi rõ ràng, lịch sự và hữu ích hơn.

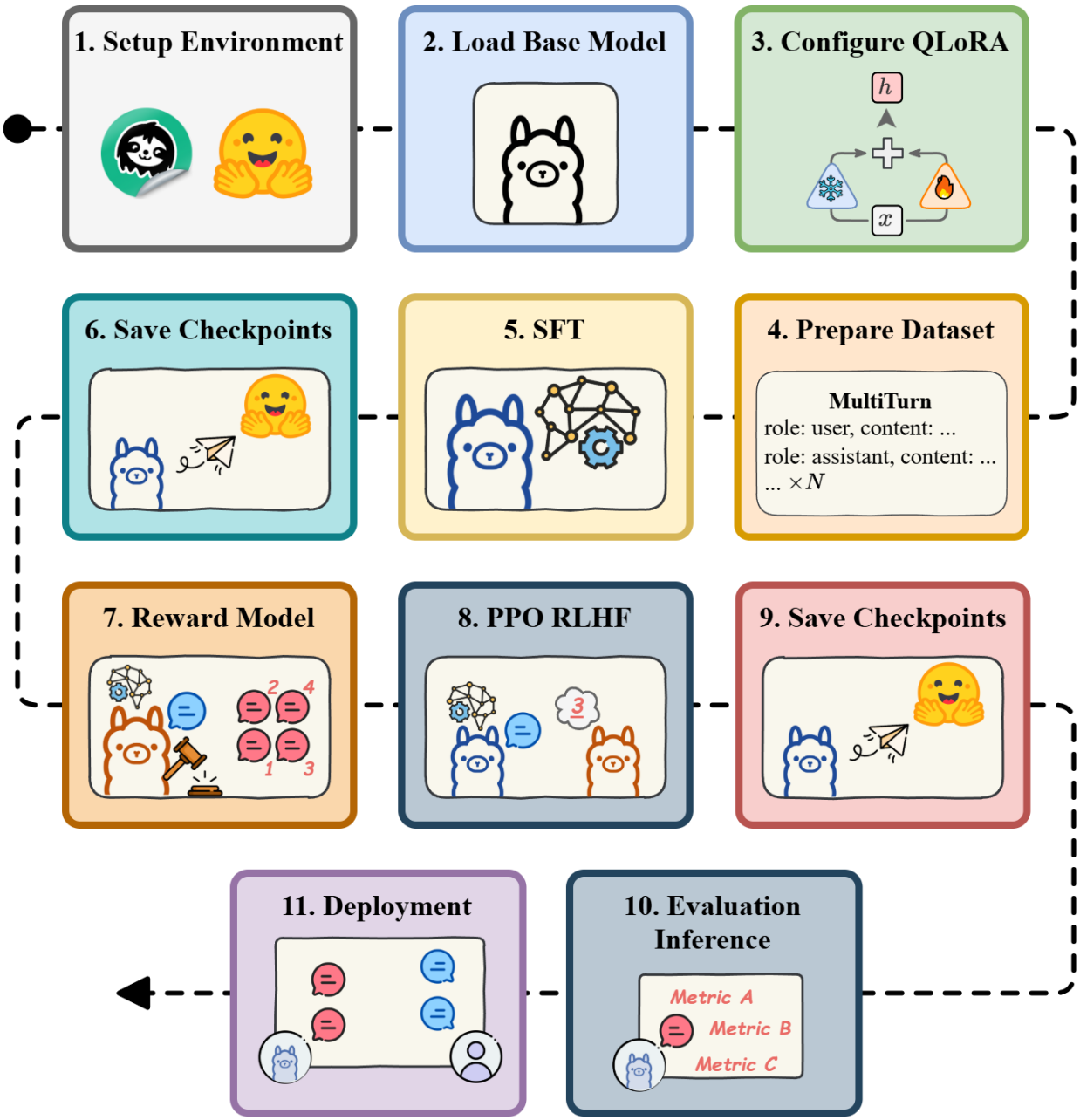
Như minh họa trong Hình 2, quy trình RLHF gồm ba bước chính: huấn luyện mô hình ngôn ngữ gốc, xây dựng mô hình phần thưởng từ phản hồi người dùng, và tinh chỉnh mô hình bằng thuật toán tăng cường như PPO. Trong hệ thống chatbot, phản hồi đầu ra sẽ được đánh giá và cải thiện qua các bước này để đảm bảo chất lượng tương tác cuối cùng với người dùng.

Như vậy, trong project này, chúng ta sẽ xây dựng một Chatbot ngôn ngữ tiếng Việt hỗ trợ các giao tiếp thường nhật dựa trên mô hình ngôn ngữ lớn. Khi hoàn thành, hệ thống có Input/Output như sau:

- **Input:** Người dùng đặt ra một câu hỏi bằng tiếng Việt.
- **Output:** Chatbot sinh ra phản hồi tự nhiên, phù hợp với mong đợi của người dùng.

II. Cài đặt chương trình

Theo pipeline như ở Hình 3, chúng ta cần đi qua bước triển khai chính để có thể hoàn thiện được toàn bộ hệ thống Chatbot. Với các phần huấn luyện mô hình đều được thực hiện trên Google Colab. Riêng phần triển khai giao diện Chat sẽ được cài đặt và thực thi ở máy cá nhân.



Hình 3: Quy trình xây dựng chatbot tiếng Việt.

II.1. Huấn luyện Chatbot tiếng Việt

Trong mục này, tiến hành xây dựng một chatbot tiếng Việt bằng cách tinh chỉnh (fine-tuning) mô hình ngôn ngữ lớn.

II.1.1. Cài đặt và import các thư viện cần thiết

Đầu tiên, tiến hành cài đặt các thư viện cần thiết, bao gồm `unsloth` để tăng tốc độ huấn luyện và `datasets` để tải bộ dữ liệu.

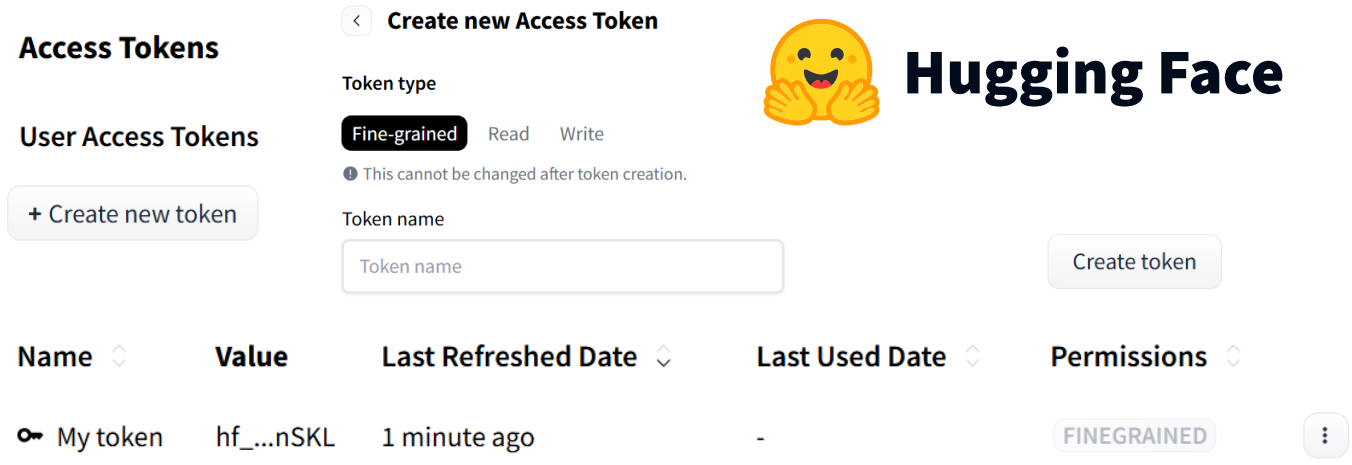
```
!pip install -q unsloth==2025.4.7 datasets==3.5.1
```

Sau đó, import các thư viện và hàm cần sử dụng:

```
import torch
from trl import SFTTrainer
from datasets import load_dataset
from unsloth import FastLanguageModel
from unsloth import FastLanguageModel
from transformers import GenerationConfig
from transformers import TrainingArguments
```

Cuối cùng, đăng nhập Hugging Face để có thể tải và lưu mô hình:

```
login(token="your huggingface token")
```



Hình 4: Minh họa tạo access token cá nhân trên Hugging Face.

II.1.2. Khai báo mô hình LLM

Kế tiếp, khai báo và khởi tạo mô hình LLM `Llama-3.2-1B-Instruct` với cấu hình 4-bit quantization để tiết kiệm tài nguyên bộ nhớ và tăng tốc tính toán:

```
MAX_SEQ_LENGTH = 2048

```

```
model, tokenizer = FastLanguageModel.from_pretrained(
    model_name="unsloth/Llama-3.2-1B-Instruct-bnb-4bit",
    max_seq_length=MAX_SEQ_LENGTH,
    load_in_4bit=True,
    dtype=torch.bfloat16,
)
```

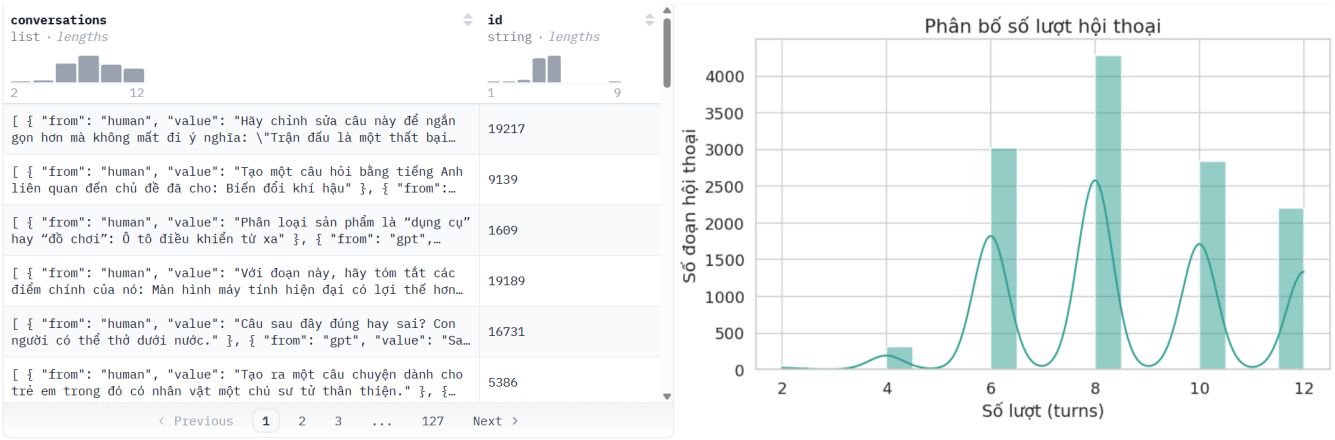
Tiếp theo, cấu hình kỹ thuật LoRA (Low-Rank Adaptation) để tinh chỉnh chỉ một phần nhỏ các tham số, nhờ đó giảm đáng kể chi phí tính toán nhưng vẫn đảm bảo hiệu quả huấn luyện:

```
model = FastLanguageModel.get_peft_model(
    model,
    r=16,
    lora_alpha=16,
    lora_dropout=0,
    target_modules=[
        "q_proj", "k_proj", "v_proj", "up_proj",
        "down_proj", "o_proj", "gate_proj"],
    use_rslora=True,
    use_gradient_checkpointing="unsloth",
    random_state = 42,
)
```

II.1.3. Tải và tổ chức bộ dữ liệu multiturn conversation

Sau khi đã thiết lập mô hình, tiến hành tải bộ dữ liệu hội thoại đa lượt tiếng Việt để huấn luyện:

```
dataset_name = "5CD-AI/Vietnamese-Multi-turn-Chat-Alpaca"
raw_dataset = load_dataset(dataset_name, split="train")
```



Hình 5: Tổng quan tập dữ liệu Vietnamese-Multi-turn-Chat-Alpaca.

Biểu đồ bên phải minh họa phân bố số lượt trao đổi (turns) giữa người dùng và mô hình trong mỗi đoạn hội thoại. Dễ nhận thấy các đoạn hội thoại chủ yếu tập trung vào các mốc cụ thể như 6, 8, 10 và 12 lượt. Đây không phải là sự ngẫu nhiên, mà là kết quả của quá trình thiết kế tập dữ liệu có chủ đích, nhằm giúp mô hình học cách xử lý hiệu quả các chuỗi tương tác phức tạp — yếu tố then chốt trong các ứng dụng như trợ lý ảo và chatbot.

Tiếp theo, chuyển đổi dữ liệu sang định dạng phù hợp để huấn luyện mô hình hội thoại:

```
SYS_INSTRUCT = "Bạn là một trợ lý AI thân thiện, hãy trả lời bằng tiếng Việt."

def convert_to_chat_format(conversations):
    messages = [{"role": "system", "content": SYS_INSTRUCT}]
    for msg in conversations:
        role = "user" if msg["from"] == "human" else "assistant"
        messages.append({"role": role, "content": msg["value"]})
    return messages

def format_prompt(example):
    messages = convert_to_chat_format(example["conversations"])
    return {
        "text": tokenizer.apply_chat_template(
            messages,
            tokenize=False,
            add_generation_prompt=False
        )
    }

def tokenize_function(examples):
    return tokenizer(
        examples["text"],
        truncation=True,
        max_length=MAX_SEQ_LENGTH,
        padding="max_length",
    )

dataset = raw_dataset.map(format_prompt, remove_columns=raw_dataset.column_names)
dataset = dataset.map(tokenize_function, batched=True)
```

II.1.4. Huấn luyện mô hình

Sau khi chuẩn bị xong dữ liệu, bắt đầu quá trình huấn luyện với các thông số đã được thiết lập như sau:

```
training_args = TrainingArguments(
    per_device_train_batch_size=32,
    gradient_accumulation_steps=2,
    learning_rate=1e-4,
    save_total_limit=4,
    logging_steps=20,
    output_dir="./checkpoint/llama3-1b-multi-conversation",
    optim="paged_adamw_8bit",
    lr_scheduler_type="cosine",
    warmup_ratio=0.05,
    save_strategy="steps",
    save_steps=50,
    report_to="none",
    remove_unused_columns=True,
    max_steps=400,
    bf16=True,
)

```

```
trainer=SFTTrainer(
    model=model,
    args=training_args,
    train_dataset=dataset,
    tokenizer=tokenizer,
)

trainer.train()
```

Khi hoàn thành huấn luyện, tải lại checkpoint tốt nhất của mô hình để sử dụng cho các bước tiếp theo:

```
model, tokenizer = FastLanguageModel.from_pretrained(
    "./checkpoint/llama3-1b-multi-conversation/checkpoint-400",
    max_seq_length=2048,
    load_in_4bit=True,
    dtype=torch.bfloat16,
)
```

II.1.5. Lưu mô hình và inference

Cuối cùng, lưu mô hình lên Hugging Face Hub để dễ dàng sử dụng cho những lần sau:

```
model.push_to_hub_merged(
    "your_huggingface_username/Llama-3.2-1B-Instruct-Chat-sft",
    commit_message="Merge weights to push to hub",
)

tokenizer.push_to_hub(
    "your_huggingface_username/Llama-3.2-1B-Instruct-Chat-sft",
    commit_message="Push tokenizer to hub",
)
```

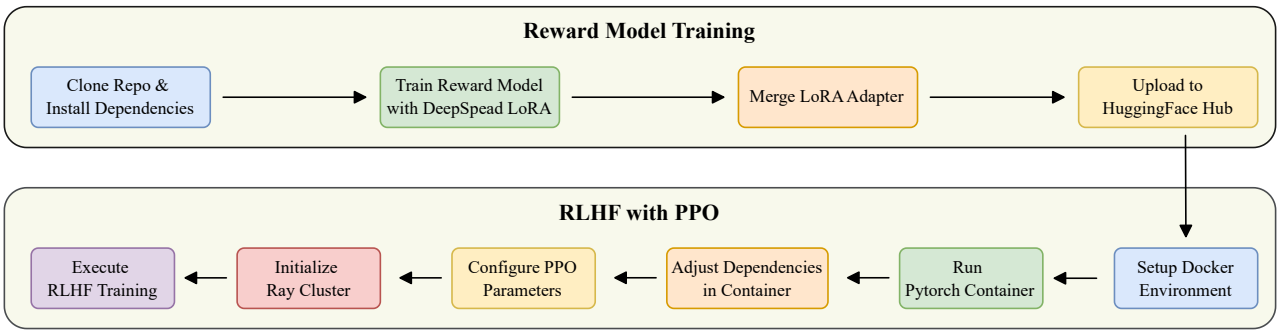
Sau khi lưu mô hình, thực hiện bước kiểm thử mô hình bằng một ví dụ hội thoại như sau:

```
generation_config = GenerationConfig(
    max_new_tokens=128,
    temperature=1.0,
    do_sample=False,
    num_return_sequences=1,
    pad_token_id=tokenizer.eos_token_id,
    eos_token_id=tokenizer.eos_token_id,
    repetition_penalty=1.3
)

prompt = [
    {"role": "system", "content": (
        "Bạn là một trợ lý AI thân thiện, "
        "hãy trả lời bằng tiếng Việt."
    )},

    {"role": "user", "content": (
        "Hãy chỉnh sửa câu này để ngắn gọn hơn mà không mất đi ý nghĩa: "
        "\"Trận đấu là một thất bại nặng nề "
        "mặc dù thực tế là cả đội đã tập luyện trong nhiều tuần.\""
```

```
    )},

    {"role": "assistant", "content": (
        "Nhiều tuần huấn luyện của đội đã dẫn đến một thất bại nặng nề."
    )},

    {"role": "user", "content": (
        "Bạn có thể đề xuất một số chiến lược mà "
        "nhóm có thể sử dụng để cải thiện hiệu suất "
        "của họ trong trận đấu tiếp theo không?"
    )}
]

chat_text = tokenizer.apply_chat_template(
    prompt,
    add_generation_prompt=True,
    tokenize=False
)

inputs = tokenizer(
    chat_text,
    return_tensors="pt"
).to("cuda:0")

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)

if "assistant\n" in output_text:
    answer = output_text.split("assistant\n")[-1].strip()
else:
    answer = output_text.strip()

print("Assistant reply:", answer)
```

II.2. Huấn luyện mô hình điểm thưởng

Để có thể áp dụng phương pháp RLHF (Reinforcement Learning from Human Feedback), trước tiên một mô hình điểm thưởng (reward model) cần được huấn luyện nhằm mô phỏng sở thích của con người. Mô hình này đóng vai trò đánh giá và so sánh các phản hồi đầu ra của mô hình ngôn ngữ, từ đó cung cấp tín hiệu thưởng để dẫn dắt quá trình học tăng cường. Trong phần này, quy trình huấn luyện mô hình reward sẽ được trình bày chi tiết, bao gồm các bước cài đặt thư viện, huấn luyện từ dữ liệu phản hồi có ưu tiên, tích hợp các trọng số huấn luyện vào mô hình chính, và tải mô hình lên Hugging Face để sử dụng cho các bước RL tiếp theo như DPO hoặc PPO.



Hình 6: Pipeline tổng quát huấn luyện mô hình tính điểm thưởng.

II.2.1. Cài đặt thư viện

Đầu tiên, mã nguồn của dự án `OpenRLHF` cần được nhân bản và các thư viện phụ thuộc phải được cài đặt. Thư viện này cung cấp các công cụ huấn luyện cho pipeline RLHF một cách đơn giản và hiệu quả.

```
!git clone https://github.com/OpenRLHF/OpenRLHF.git
!cd OpenRLHF
!pip install openrlhf[vllm_latest]
```

Hình 7: Logo của thư viện OpenRLHF. Nguồn: link.

II.2.2. Huấn luyện mô hình reward

Sau khi đã cài đặt thư viện cần thiết, mô hình reward sẽ được huấn luyện từ một checkpoint đã huấn luyện sơ bộ bằng phương pháp supervised fine-tuning (SFT). Dữ liệu huấn luyện là một tập gồm các cặp phản hồi được gán nhãn "được chọn" (chosen) và "bị từ chối" (rejected). Mục tiêu của mô hình là học cách gán điểm số cao hơn cho phản hồi được ưu tiên hơn.

Quá trình huấn luyện sẽ được thực hiện bằng cách sử dụng DeepSpeed để tận dụng tối đa tài nguyên phần cứng. Các tùy chọn như LoRA, gradient checkpointing và flash attention đã được kích hoạt để giảm thiểu chi phí tính toán và bộ nhớ.

```
!deepspeed --module openrlhf.cli.train_rm \
   --save_path ./checkpoint/Llama-3.2-1B-rm-dpo \
   --save_steps -1 \
   --logging_steps 1 \
   --eval_steps -1 \
   --train_batch_size 96 \
   --micro_train_batch_size 8 \
```

```
    --pretrain your_huggingface_username/Llama-3.2-1B-Instruct-Chat-sft \
    --value_head_prefix score \
    --bf16 \
    --max_epochs 1 \
    --max_len 2048 \
    --zero_stage 2 \
    --learning_rate 5e-6 \
    --dataset thuanan/Vi-Alpaca-Preference \
    --apply_chat_template \
    --chosen_key chosen \
    --rejected_key rejected \
    --flash_attn \
    --load_checkpoint \
    --packing_samples \
    --gradient_checkpointing \
    --adam_offload \
    --lora_rank 16 \
    --lora_alpha 32
```

II.2.3. Kết nối Lora Adapter vào mô hình

Sau khi huấn luyện xong, adapter LoRA chứa trọng số huấn luyện sẽ được hợp nhất vào checkpoint ban đầu để tạo ra một mô hình hoàn chỉnh. Việc kết hợp này sẽ được thực hiện bằng tiện ích `lora_combiner` có sẵn trong thư viện.

```
!python -m openrlhf.cli.lora_combiner \
    --model_path your_huggingface_username/Llama-3.2-1B-Instruct-Chat-sft \
    --lora_path ./checkpoint/Llama-3.2-1B-rm-dpo \
    --output_path ./checkpoint/Llama-3.2-1B-rm-dpo-combined \
    --is_rm \
    --bf16
```

II.2.4. Đưa mô hình đã huấn luyên lên HuggingFace

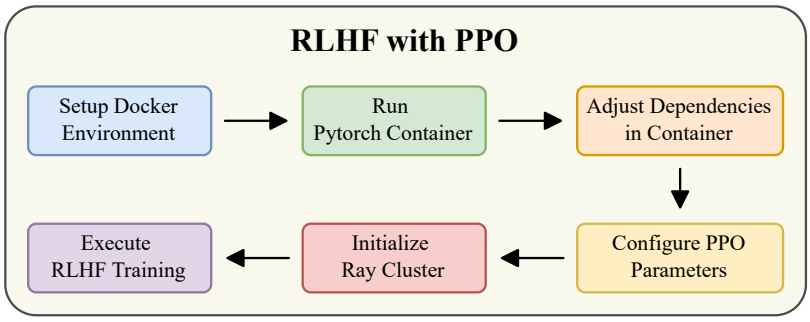
Sau cùng, mô hình reward sau khi hợp nhất được tải lên Hugging Face Hub để phục vụ các bước tiếp theo.

```
from transformers import AutoTokenizer, AutoModelForSequenceClassification

ckpt_path = "./checkpoint/Llama-3.2-1B-rm-dpo-combined"
tokenizer = AutoTokenizer.from_pretrained(ckpt_path)
model = AutoModelForSequenceClassification.from_pretrained(ckpt_path)

model.push_to_hub(
    "your_huggingface_username/Llama-3.2-1B-RM-DPO",
    commit_message="Add model ckpt",
)

tokenizer.push_to_hub(
    "your_huggingface_username/Llama-3.2-1B-RM-DPO",
    commit_message="Add tokenizer",
)
```

II.3. Cải thiện phản hồi Chatbot với RLHF

Tiếp tục sử dụng thư viện OpenRLHF nhằm thực hiện căn chỉnh phản hồi của mô hình. Để đơn giản hoá quy trình, có thể sử dụng các cấu hình được nhóm tác giả OpenRLHF xây dựng sẵn như sau:

1. Sử dụng Pytorch trong môi trường container (Docker).
2. Cài đặt các thư viện cần thiết như openrlhf, flash-attn,... bên trong container.
3. Tiến hành căn chỉnh mô hình (SFT) bằng reward model (RM) thông qua PPO.



Hình 8: Chi tiết quy trình thực hiện PPO.

Đầu tiên, clone github OpenRLHF về và tiến hành cài đặt nvidia-container-toolkit như sau:

```
git clone https://github.com/OpenRLHF/OpenRLHF.git
cd OpenRLHF

bash examples/scripts/nvidia_docker_install.sh
```

Tiếp theo, cũng tại vị trí thư mục OpenRLHF, tiến hành khởi chạy Pytorch container và cài đặt các thư viện cần thiết bằng các câu lệnh sau:

```
bash examples/scripts/docker_run.sh

# Sau khi image được pull về hoàn tất và container được khởi chạy.
# Hãy chạy các lệnh này bên trong container.

pip uninstall xgboost transformer_engine flash_attn pynvml -y
pip install openrlhf[vllm]
```

Tại file ở đường dẫn `./examples/scripts/train\_ppo\_llama\_ray.sh` hãy sử dụng cấu hình sau đây:

```
ray job submit --address="http://127.0.0.1:8265" \
   --runtime-env-json='{"setup_commands": ["pip install openrlhf[vllm]"]}' \
   -- python3 -m openrlhf.cli.train_ppo_ray \
   --ref_num_nodes 1 \
   --ref_num_gpus_per_node 1 \
   --reward_num_nodes 1 \
   --reward_num_gpus_per_node 1 \
```

```
    --critic_num_nodes 1 \
    --critic_num_gpus_per_node 1 \
    --actor_num_nodes 1 \
    --actor_num_gpus_per_node 1 \
    --vllm_num_engines 1 \
    --vllm_tensor_parallel_size 1 \
    --colocate_critic_reward \
    --colocate_actor_ref \
    --pretrain your_huggingface_username/Llama-3.2-1B-Instruct-Chat-sft \
    --reward_pretrain thuanan/Llama-3.2-1B-RM-DPO \
    --save_path /openrlhf/examples/checkpoint/Llama-3.2-1B-RLHF-2k-vi-alpaca \
    --micro_train_batch_size 2 \
    --train_batch_size 32 \
    --micro_rollout_batch_size 4 \
    --rollout_batch_size 1024 \
    --max_epochs 1 \
    --prompt_max_len 1024 \
    --generate_max_len 1024 \
    --zero_stage 3 \
    --bf16 \
    --actor_learning_rate 5e-7 \
    --critic_learning_rate 9e-6 \
    --init_kl_coef 0.01 \
    --prompt_data thuanan/Prompt-Vi-Alpaca-Preference-2k \
    --input_key context_messages \
    --apply_chat_template \
    --normalize_reward \
    --packing_samples \
    --adam_offload \
    --flash_attn \
    --gradient_checkpointing \
    --load_checkpoint \
    --use_wandb "<Your-token>"
```

Có thể điều chỉnh các tham số liên quan đến batch size sao cho phù hợp với dung lượng GPU hiện có. Thêm WandB API Key để theo dõi tài nguyên và các metrics trong quá trình huấn luyện.

Tiếp theo là khởi tạo Ray node để quản lý runtime như sau:

```
ray start --head --node-ip-address 0.0.0.0 --num-gpus 1
```

Dựa trên tài nguyên sẵn có, có thể điều chỉnh số lượng GPU nhằm tối ưu quá trình huấn luyện. Cuối cùng, chạy script sau để huấn luyện mô hình:

```
cd /openrlhf

bash ./examples/scripts/train_ppo_llama_ray.sh
```

Hình 9: Quan sát, theo dõi quá trình huấn luyện trên WandB.

II.4. Triển khai Chatbot lên giao diện chat

Đầu tiên, cài đặt môi trường và các thư viện cần thiết thông qua conda (hoặc venv):

```
conda create -n rlhf-serve python=3.12 --y
conda activate rlhf-serve
pip install gradio openai vllm
```

Tiếp theo là khởi chạy vLLM server để serving mô hình thành API.

```
vllm serve thuanan/Llama-3.2-1B-RLHF-2k-vi-alpaca \
  --api-key "<Your-key>" \
  --port 8000 \
  --quantization bitsandbytes \
  --enable-prefix-caching \
  --swap-space 16 \
  --gpu-memory-utilization 0.9 \
  --disable-log-requests \
  --max-model-len 2048
```

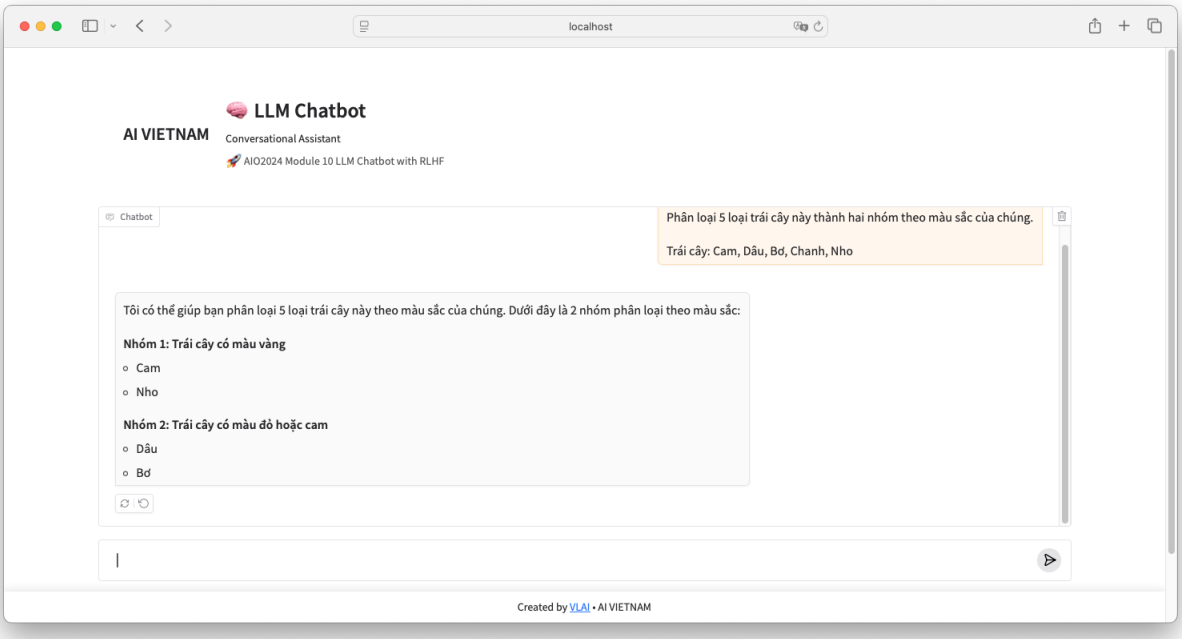
Cuối cùng, tạo file app.py và dán đoạn code sau để xây dựng giao diện với thư viện gradio.

```
import gradio as gr
import os
from openai import OpenAI

client = OpenAI(
    api_key=os.getenv("OPENAI_API_KEY"),
    base_url=os.getenv("OPENAI_API_BASE_URL"),
)
```

```

# Chat function
def respond(message, history, system_message, max_tokens, temperature, top_p):
    messages = [{"role": "system", "content": system_message}]
    for msg in history:
        messages.append(msg)
    messages.append({"role": "user", "content": message})

    stream = client.chat.completions.create(
        model=os.getenv("CHAT_MODEL"),
        messages=messages,
        max_tokens=max_tokens,
        temperature=temperature,
        top_p=top_p,
        stream=True,
    )

    response = ""
    for chunk in stream:
        if chunk.choices and chunk.choices[0].delta.content:
            token = chunk.choices[0].delta.content
            response += token
            yield {"role": "assistant", "content": response}

# UI
chat = gr.ChatInterface(
    fn=respond,
    additional_inputs=[
        gr.Textbox(value="You are a friendly chatbot.", label="System message"),
        gr.Slider(1, 2048, value=512, step=1, label="Max new tokens"),
        gr.Slider(0.1, 4.0, value=0.7, step=0.1, label="Temperature"),
        gr.Slider(0.1, 1.0, value=0.95, step=0.05, label="Top-p (nucleus sampling)")
    ],
    type="messages",
)

with gr.Blocks() as demo:
    chat.render()

if __name__ == "__main__":
    demo.launch(server_name="0.0.0.0", server_port=7860)
```

Khởi chạy gradio app bằng lệnh sau:

```
python app.py
```



Hình 10: Giao diện chatbot sử dụng thư viện gradio.

III. Câu hỏi trắc nghiệm

1. Chatbot là gì trong bối cảnh trí tuệ nhân tạo hiện đại?
   (*a*) Một hệ thống tương tác với con người bằng ngôn ngữ tự nhiên.
   (*b*) Một phần mềm chỉ dùng để điều khiển thiết bị IoT.
   (*c*) Một hệ thống có khả năng sinh nội dung video tự động.
   (*d*) Một chương trình tự động thu thập dữ liệu từ website.

2. Mô hình ngôn ngữ lớn (LLM) đóng vai trò gì trong việc xây dựng chatbot hiện đại?
   (*a*) Là một công cụ để lưu trữ dữ liệu người dùng.
   (*b*) Là thuật toán tìm kiếm nhanh trong bộ nhớ máy chủ.
   (*c*) Là hệ thống xử lý hình ảnh để tạo biểu cảm khuôn mặt cho chatbot.
   (*d*) Là thành phần cốt lõi giúp sinh phản hồi ngôn ngữ tự nhiên.

3. Chatbot có thể được phân loại dựa trên yếu tố nào sau đây?
   (*a*) Loại hệ điều hành mà chatbot hoạt động.
   (*b*) Khả năng kết nối Internet của chatbot.
   (*c*) Mức độ kiểm soát phản hồi: rule-based, retrieval-based, generative-based.
   (*d*) Tốc độ mạng mà chatbot sử dụng để trả lời người dùng.

4. Điều nào sau đây là đúng về Retrieval-Augmented Generation (RAG) khi xây dựng chatbot?
   (*a*) Là kỹ thuật để huấn luyện mô hình với ảnh và video.
   (*b*) Là cách để giảm số lượng token đầu vào khi sinh văn bản.
   (*c*) Là phương pháp kết hợp tìm kiếm dữ liệu ngoài với khả năng sinh ngôn ngữ của mô hình.
   (*d*) Là phương pháp giúp chatbot tự động tạo câu hỏi khảo sát người dùng.

5. Để một chatbot tuân theo hướng dẫn người dùng tốt hơn, kỹ thuật nào sau đây thường được sử dụng?
   (*a*) Quantization.
   (*b*) Instruction Tuning.
   (*c*) Data Compression.
   (*d*) Graph Embedding.

6. RLHF được sử dụng trong huấn luyện chatbot nhằm mục tiêu gì?
   (*a*) Điều chỉnh mô hình để tạo phản hồi phù hợp hơn với sở thích con người.
   (*b*) Giúp chatbot thực hiện tác vụ xử lý ảnh hiệu quả hơn.
   (*c*) Giảm thời gian trả lời của chatbot xuống mức tối thiểu.

(*d*) Tăng độ chính xác trong việc phân loại văn bản.

7. Trong quy trình RLHF, mô hình reward đảm nhiệm vai trò gì?

   (*a*) Tạo dữ liệu đào tạo mới cho mô hình gốc.

   (*b*) Dự đoán token tiếp theo như mô hình ngôn ngữ thông thường.

   (*c*) Thực hiện fine-tuning toàn bộ mô hình bằng kỹ thuật supervised learning.

   (*d*) Đánh giá và xếp hạng các phản hồi đầu ra dựa trên mức độ phù hợp với ý định người dùng.

8. Điều nào sau đây mô tả đúng cách mô hình policy được cập nhật trong RLHF dựa trên phản hồi từ mô hình reward?

   (*a*) Mô hình policy được tối ưu để sinh ra phản hồi có điểm thưởng cao hơn theo đánh giá của mô hình reward.

   (*b*) Mô hình policy được huấn luyện để tạo ra phản hồi giống hệt phản hồi được con người chọn.

   (*c*) Mô hình reward tạo ra gradient và trực tiếp cập nhật trọng số của mô hình policy.

   (*d*) Mô hình policy và reward được huấn luyện đồng thời trên cùng một batch dữ liệu.

9. Đoạn code sau có chức năng gì trong quá trình fine-tuning mô hình hội thoại dựa trên LLM?

```
def format_prompt(example):
    messages = convert_to_chat_format(example["conversations"])
    return {
        "text": tokenizer.apply_chat_template(
            messages,
            tokenize=False,
            add_generation_prompt=False
        )
    }
```

   (*a*) Dùng để sinh phản hồi mẫu từ mô hình trước khi huấn luyện.

   (*b*) Tính toán loss giữa prompt và nhãn đúng trong RLHF.

   (*c*) Tạo embedding vector từ hội thoại để phục vụ truy xuất.

   (*d*) Chuyển đổi các hội thoại thành định dạng chuẩn để huấn luyện mô hình.

10. Trong đoạn mã dưới đây, đâu là mục đích chính của tham số `advantages` được sử dụng trong bước cập nhật policy của PPO?

```
from trl import PPOTrainer

for batch in dataset:
    query_tensors = tokenizer(batch["prompt"], return_tensors="pt").input_ids
    response_tensors = model.generate(query_tensors)
    rewards = reward_model(query_tensors, response_tensors)

    logprobs = compute_logprobs(model, query_tensors, response_tensors)
    advantages = rewards - rewards.mean()
```

```
10
11      trainer.step(query_tensors, response_tensors, rewards, logprobs, advantages)
```

(*a*) Giúp tăng tính ngẫu nhiên trong quá trình sinh phản hồi.

(*b*) Làm mượt gradient để mô hình hội tụ nhanh hơn.

(*c*) Điều chỉnh mức độ cập nhật theo độ tốt tương đối của phản hồi.

(*d*) Tránh việc sinh phản hồi trùng lặp với dữ liệu đã huấn luyện.

IV. Phụ lục

1. **Datasets:** Các file dataset được đề cập trong bài có thể được tải tại đây.
2. **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).
3. **Demo:** Web demo và mã nguồn của ứng dụng có thể được truy cập tại đây.
4. **Rubric:**

Mục	Kiến Thức	Đánh Giá
I.	- Kiến thức về mô hình lớn (LLMs). - Khái niệm Chatbot và Chatbot ứng dụng LLMs. - Kiến thức về kỹ thuật Instruction Tuning trong LLMs. - Kiến thức về kỹ thuật RLHF để cải thiện phản hồi của mô hình lớn. - Kiến thức về thư viện PyTorch và HuggingFace. - Triển khai PEFT (LoRA) và thiết lập các tham số liên quan đến LoRA.	- Nắm được lý thuyết về mô hình lớn (LLMs) và Chatbot. - Nắm kỹ thuật Instruction Tuning và RLHF trong LLMs. - Nắm được lý thuyết cơ bản về kỹ thuật LoRA. - Có khả năng cài đặt, thực hiện Instruction Tuning và RLHF cho mô hình lớn nhằm phục vụ cho ứng dụng Chatbot.

- Hết -