AI VIET NAM – AIO COURSE 2024

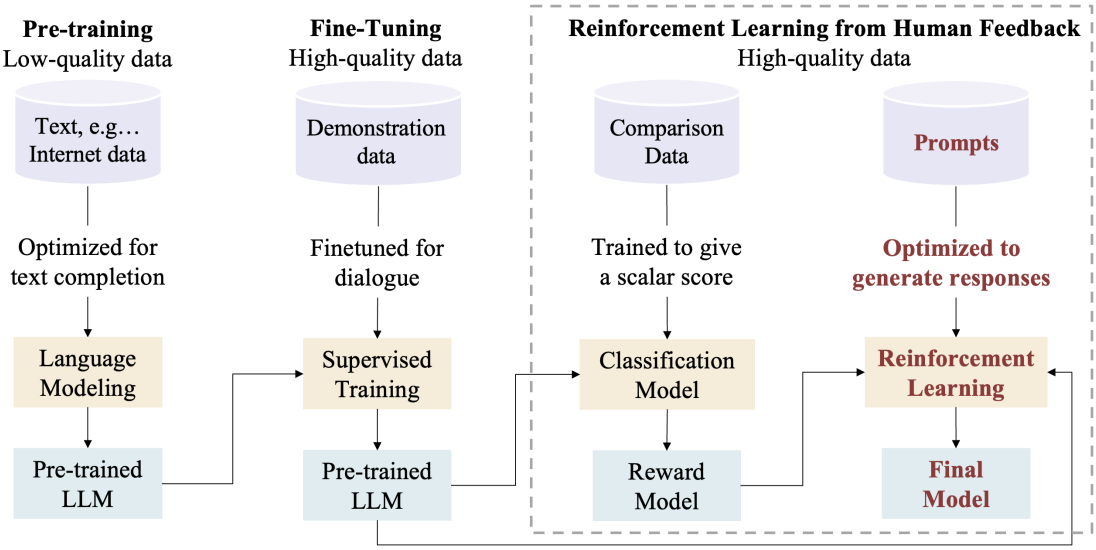
Preference Tuning with LLMs

Reinforcement Learning from Human Feedback
Direct Preference Optimization

Quoc-Thai Nguyen, Minh-Nam Tran và Quang-Vinh Dinh

Ngày 27 tháng 4 năm 2025

Phần 1. Giới thiệu



Hình 1: Reinforcement Learning from Human Feedback.

Trong những năm gần đây, mô hình ngôn ngữ lớn (LLMs) đã cho thấy khả năng ấn tượng, nhưng việc đảm bảo rằng hành vi của mô hình phù hợp với kỳ vọng của con người vẫn là thách thức. RLHF (Reinforcement Learning from Human Feedback) là một phương pháp tiếp cận giúp mô hình học cách tối ưu hóa phản hồi từ con người để cải thiện hành vi.

Khác với fine-tuning truyền thống (chỉ cần dữ liệu đầu ra đúng), RLHF xây dựng một hàm phần thưởng phức tạp phản ánh mức độ hài lòng của người dùng.

Các bước cơ bản:

- Supervised Fine-tuning (SFT): Huấn luyện ban đầu trên tập dữ liệu prompt → expected response bằng cross-entropy.

- Huấn luyện Reward Model: Từ dữ liệu cặp so sánh (prompt, chosen, rejected). Reward Model học đánh giá: chosen phải có điểm cao hơn rejected.

  Reward model r_ϕ được tối ưu bằng Binary Cross Entropy giữa sự khác biệt reward của chosen và rejected.

$$\mathcal{L}_{\text{reward}}(\phi) = -\mathbb{E}_{(x,y_c,y_r)}\left[\log \sigma\left(r_\phi(x,y_c) - r_\phi(x,y_r)\right)\right]$$

  Trong đó:

  (x, y_c, y_r): Prompt, câu trả lời được chọn, câu trả lời bị từ chối.

  σ là hàm sigmoid.

- Tối ưu Policy bằng Reinforcement Learning (PPO): Fine-tune mô hình để tối đa hóa reward model output thay vì likelihood ban đầu.

  Sau khi có reward model, ta dùng PPO (Proximal Policy Optimization) để update policy π_θ

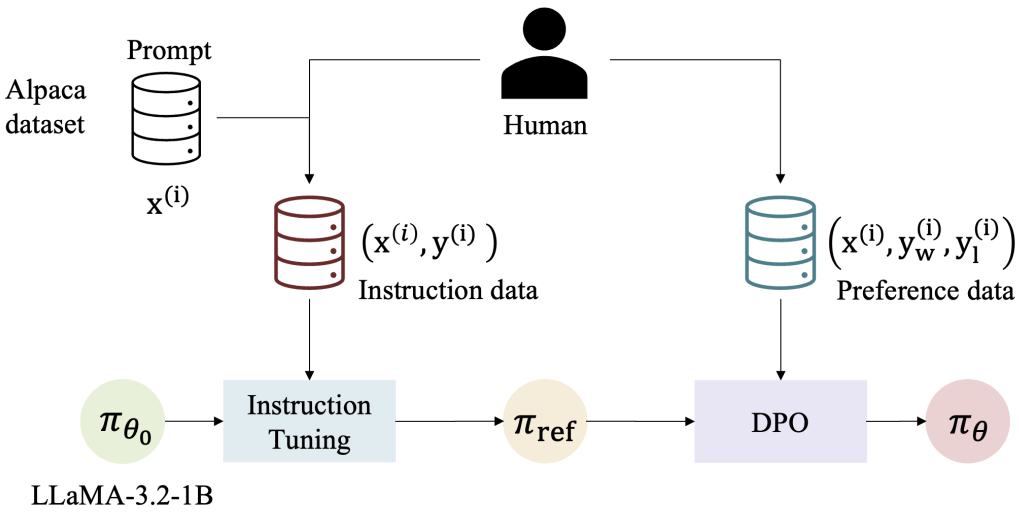
$$\mathcal{L}_{\text{PPO}}(\theta) = \mathbb{E}\left[\min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}\hat{A}(s,a), \text{clip}\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}, 1-\epsilon, 1+\epsilon\right)\hat{A}(s,a)\right)\right]$$

  Trong đó:

  $\hat{A}(s,a)$: Advantage function, đại diện cho độ tốt của hành động so với trung bình.

  ϵ (ví dụ 0.2) để clip gradient, tránh update quá mạnh.

Direct Preference Optimization (DPO)



Hình 2: Direct Preference Optimization.

DPO (Direct Preference Optimization) là phương pháp mới nhằm đơn giản hóa RLHF:

- Không huấn luyện reward model trung gian, không cần PPO.
- DPO tối ưu trực tiếp mô hình sinh ra đầu ra được người dùng ưa thích hơn.

Ý tưởng: nếu ta có (prompt, chosen, rejected), ta chỉ cần điều chỉnh policy sao cho:

$$P(chosen) > P(rejected)$$

Công thức tính hàm loss:

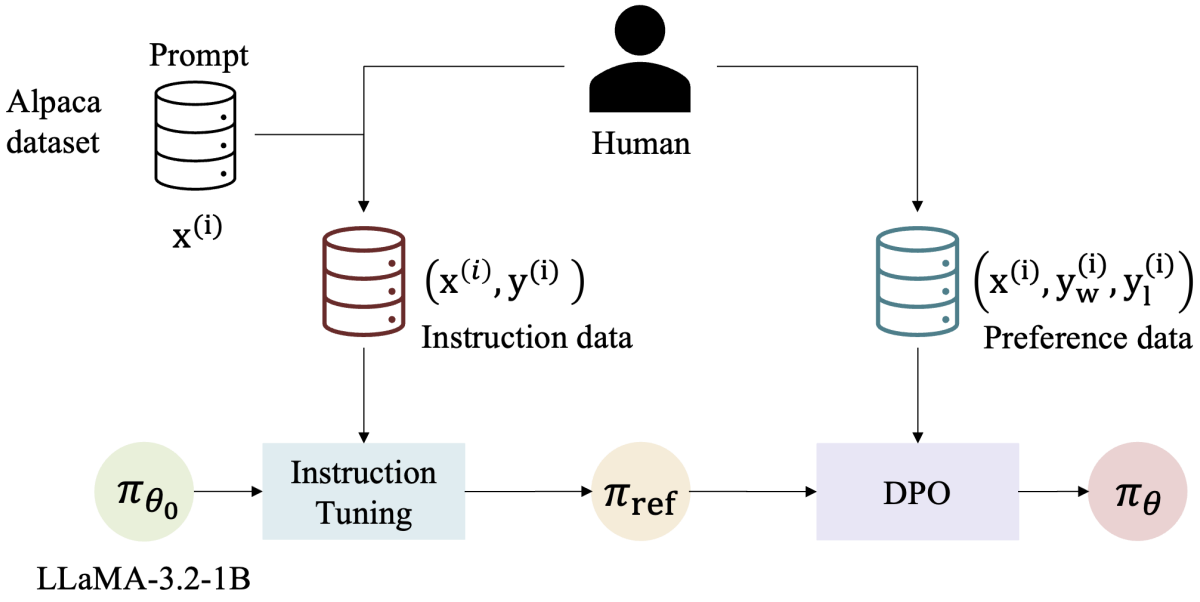
$$\mathcal{L}_{\text{DPO}}(\theta) = -\mathbb{E}_{(x,y_c,y_r)}\left[\log\left(\frac{\exp(\beta \log \pi_\theta(y_c|x))}{\exp(\beta \log \pi_\theta(y_c|x)) + \exp(\beta \log \pi_\theta(y_r|x))}\right)\right]$$

Hoặc được viết lại:

$$\mathcal{L}_{\text{DPO}}(\theta) = -\mathbb{E}\left[\log \sigma\left(\beta\left(\log \pi_\theta(y_c|x) - \log \pi_\theta(y_r|x)\right)\right)\right]$$

Phần 2. DPO for Vietnamese Custom Preference Dataset

Quá trình huấn luyện DPO cho một mô hình ngôn ngữ lớn gồm các bước, được minh hoạ trong hình



Hình 3: DPO pipeline (source).

Quá trình huấn luyện mô hình như sau:

1. **Cài đặt thư viện và tải về bộ dữ liệu**

   Đầu tiên là cài đặt một số thư viện để có thể chạy được các mô hình ngôn ngữ lớn từ thư viện `transformers`. Bộ dữ liệu được sử dụng là bộ dữ liệu Alpaca từ Stanford đã được dịch sang tiếng Việt và chuyển về format { `question`, `chosen`, `rejected` }. Bộ dữ liệu được lưu trữ ở đây: thainq107/Vi-Alpaca-Preference.

```
# Install libs
!pip install -q datasets trl peft bitsandbytes sentencepiece

from datasets import load_dataset

dataset = load_dataset("thainq107/Vi-Alpaca-Preference", cache_dir=CACHE_DIR)
dataset, dataset["train"][0]
```

   Thông tin về bộ dữ liệu và một data point của bộ dữ liệu được in ra như sau:

```
(DatasetDict({
    train: Dataset({
        features: ["id", "question", "chosen", "rejected"],
        num_rows: 65017
    })
    test: Dataset({
        features: ["id", "question", "chosen", "rejected"],
        num_rows: 2000
    })
```

```
 }),
 {"id": "dolly-14239",
  "question": ...,
  "chosen": ...,
  "rejected": ...
  })
```

2. Modeling

Mô hình LLaMA-3.2-1B-Instruct được tải từ thư viện `transformers` với kỹ thuật lượng tử hóa 4-bit từ thư viện `bitsandbytes`, giúp giảm dung lượng và tối ưu hóa hiệu suất tính toán trên GPU.

```
# Model to fine-tune
model_name = "thainq107/Llama-3.2-1B-Instruct-sft"
cache_dir = "./cache"

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

base_model = AutoModelForCausalLM.from_pretrained(
    BASE_MODEL_ID,
    trust_remote_code=True,
    device_map={'' : torch.cuda.current_device()},
    token=hf_token,
    cache_dir=CACHE_DIR,
    torch_dtype=torch.bfloat16,
    quantization_config=bnb_config,
)
base_model.config.use_cache = False

tokenizer = AutoTokenizer.from_pretrained(BASE_MODEL_ID, cache_dir=CACHE_DIR,
    trust_remote_code=True)

if tokenizer.pad_token is None:
    tokenizer.pad_token = tokenizer.eos_token
    tokenizer.pad_token_id = tokenizer.eos_token_id
```

Tiếp theo, kỹ thuật LoRA được áp dụng lên nhiều thành phần của mô hình, bao gồm các modules attention (`q_proj`, `k_proj`, `v_proj`, `o_proj`), các modules feed-forward network (`up_proj`, `down_proj`), và module gating (`gate_proj`).

```
# QLoRA configuration
peft_config = LoraConfig(
    r=16,
    lora_alpha=16,
    lora_dropout=0.05,
    bias="none",
    task_type="CAUSAL_LM",
    target_modules=[
        "q_proj",
        "k_proj",
        "v_proj",
        "o_proj",
        "gate_proj",
        "up_proj",
```

```
        "down_proj"
    ]
)
```

Bên cạnh đó, gradient checkpointing được kích hoạt giúp tiết kiệm bộ nhớ trong quá trình huấn luyện mà không làm giảm hiệu suất.

3. Preprocessing

Để huấn luyện mô hình, ta sử dụng chat template có sẵn của Llama3.2 như sau:

```
conversation = [
    {"role": "system", "content": "You are a helpful assistant."},
    {"role": "user", "content": "This is the prompt"},
    {"role": "assistant", "content": "This is the chosen"},
]
print(tokenizer.apply_chat_template(
    conversation, tokenize=False, add_generation_prompt=False
))
```

và đây là kết quả khi in ra màn hình cấu trúc của prompt:

```
"""<|begin_of_text|><|start_header_id|>system<|end_header_id|>

Cutting Knowledge Date: December 2023
Today Date: 20 Apr 2025

You are a helpful assistant.<|eot_id|><|start_header_id|>user<|end_header_id|>

This is the prompt<|eot_id|><|start_header_id|>assistant<|end_header_id|>

This is the chosen<|eot_id|>"""
```

4. Training

Quá trình train DPO gồm 3 bước: pre-training, supervised fine-tuning, và policy optimizing. Khi chúng ta sử dụng mô hình Llama3.2-1B-Instruct, mô hình đã trải qua bước pre-training (và supervised fine-tuning). Do đó, ta cần huấn luyện lại mô hình trên preference dataset của mình ở hai bước supervised fine-tuning và policy optimizing.

Đầu tiên, ta thiếu lập các hyperparameters như sau:

```
hyperparameters = {
    "per_device_train_batch_size": 2,
    "gradient_accumulation_steps": 8,
    "gradient_checkpointing": True,
    "learning_rate": 3e-5,
    "logging_steps": 500,
    "max_steps": 5_000,
    "save_strategy": "no",
    "overwrite_output_dir": True,
    "optim": "paged_adamw_8bit",
    "warmup_steps": 500,
    "bf16": True,
}
MAX_LENGTH = 512
```

Trong đó

- `per_device_train_batch_size = 2`: Số lượng mẫu dữ liệu được xử lý trên mỗi thiết bị (ví dụ mỗi GPU) trong một bước huấn luyện.

- `gradient_accumulation_steps = 8`: Tích lũy gradient trong 8 bước trước khi thực hiện cập nhật trọng số, giúp mô phỏng một batch lớn mà không cần dùng quá nhiều bộ nhớ.
- `gradient_checkpointing = True`: Kích hoạt cơ chế lưu trữ trung gian giúp giảm sử dụng bộ nhớ GPU bằng cách tính lại một số phần trong quá trình backpropagation.
- `learning_rate = 3e-5`: Tốc độ học (learning rate) của mô hình – là hệ số quyết định mức độ cập nhật trọng số sau mỗi bước huấn luyện.
- `logging_steps = 500`: Ghi lại log (ví dụ loss, learning rate,...) sau mỗi 500 bước huấn luyện để theo dõi tiến trình.
- `max_steps = 5000`: Tổng số bước huấn luyện sẽ được thực hiện.
- `save_strategy = "no"`: Không lưu mô hình tự động trong quá trình huấn luyện. Việc lưu mô hình cần được thực hiện thủ công hoặc bằng đoạn mã riêng.
- `overwrite_output_dir = True`: Cho phép ghi đè lên thư mục đầu ra nếu đã tồn tại nội dung từ các lần huấn luyện trước.
- `optim = "paged_adamw_8bit"`: Sử dụng trình tối ưu hoá AdamW phiên bản 8-bit kết hợp với phân trang (paged) để tiết kiệm bộ nhớ và phù hợp với mô hình lớn.
- `warmup_steps = 500`: Số bước warm-up, tức là giai đoạn khởi động với learning rate tăng dần từ nhỏ đến đầy đủ trong 500 bước đầu tiên.
- `bf16 = True`: Sử dụng định dạng số học **bfloat16** để tăng hiệu suất tính toán và giảm bộ nhớ mà vẫn giữ được độ chính xác tương đối tốt.
- `MAX_LENGTH = 512`: Chiều dài tối đa của chuỗi đầu vào mà mô hình có thể xử lý, thường áp dụng cho tokenized input.

Ta chạy supervised fine-tuning như sau:

```
def formatting_prompt_with_chat_template(example):
    conversation = [
        {"role": "system", "content": "You are a helpful assistant."},
        {"role": "user", "content": example["question"]},
        {"role": "assistant", "content": example["chosen"]},
    ]
    prompt = tokenizer.apply_chat_template(
        conversation, tokenize=False, add_generation_prompt=False
    )
    return prompt

SFT_OUTPUT_DIR = f"{ROOT_OUTPUT_DIR}-SFT"

sft_config = SFTConfig(
    **{ **hyperparameters, "output_dir": SFT_OUTPUT_DIR , "max_seq_length":
    MAX_LENGTH }
)
sft_trainer = SFTTrainer(
    model=base_model,
    peft_config=peft_config,
    processing_class=tokenizer,
    args=sft_config,
    train_dataset=dataset["train"],
    formatting_func=formatting_prompt_with_chat_template
)

sft_trainer.train()
```

Khi các tham số huấn luyện đã được cấu hình, mô hình huấn luyện được bắt đầu với `SFTTrainer` và Các tham số huấn luyện được truyền vào qua đối tượng `args`.

Cuối cùng, sau khi kết thúc huấn luyện bước supervised fine-tuning này, ta lưu mô hình như sau:

```
sft_trainer.save_model(SFT_OUTPUT_DIR)
sft_trainer.push_to_hub(f"{HF_USER}/{MODEL_NAME}-SFT", token=hf_token) # hf_token
    is your huggingface token
```

Sau khi huấn luyện mô hình với supervised fine-tuning, ta tiếp tục bước policy optimization.

Ta convert dữ liệu về conversational format:

```
def convert_to_conversational_preference_format(example):
    return {
        "id": example["id"],
        "prompt": [{"role": "system", "content": "You are a helpful assistant."},
    {"role": "user", "content": example["question"]}],
        "chosen": [{"role": "assistant", "content": example["chosen"]}],
        "rejected": [{"role": "assistant", "content": example["rejected"]}],
    }

dpo_dataset = dataset.map(convert_to_conversational_preference_format)
```

Tương tự như supervised fine-tuning, ta convert dataset về conversational format.

Tiếp theo, ta load LoRA weights đã huấn luyện với supervised fine-tuning để tiếp tục quá trình policy optimization với `DPOTrainer`.

```
dpo_full_model = base_model.load_adapter(
    SFT_OUTPUT_DIR, is_trainable=True, adapter_name="dpo_full_adapter"
)
```

Và cuối cùng, chúng ta dùng `DPOTrainer` để huấn luyện mô hình:

```
DPO_FULL_OUTPUT_DIR = f"{ROOT_OUTPUT_DIR}-DPO-full"
dpo_full_args = DPOConfig(
    **{ **hyperparameters, "output_dir": DPO_FULL_OUTPUT_DIR, "max_length":
    MAX_LENGTH }
)

dpo_full_trainer = DPOTrainer(
    dpo_full_model,
    args=dpo_full_args,
    train_dataset=dpo_dataset["train"],
    processing_class=tokenizer,
    peft_config=peft_config,
)
dpo_full_trainer.train()
```

5. Save Models

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

```
dpo_full_trainer.save_model(DPO_FULL_OUTPUT_DIR)
dpo_full_trainer.push_to_hub(f"{HF_USER}/{MODEL_NAME}-DPO-full", token=hf_token)
```

- Mô hình sau huấn luyện được lưu cục bộ vào thư mục `DPO_FULL_OUTPUT_DIR` bằng hàm `save_model()`.
- Sau đó, mô hình được đẩy lên Hugging Face Hub thông qua hàm `push_to_hub()`, giúp 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 HuggingFace Token để xác thực quyền truy cập.

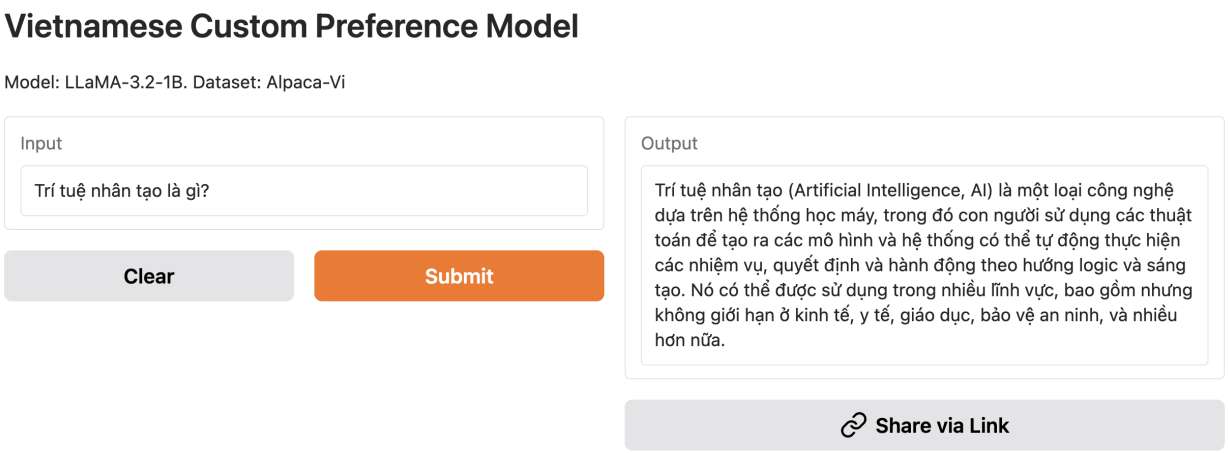
6. Inference

Sau khi huấn luyện, sử dụng mô hình để thực hiện suy luận (inference) trên một ví dụ như sau:

```python
def get_model_response(model, tokenizer, instruction):
    cur_conversation = [
        {"role": "system", "content": "You are a helpful assistant."},
        {"role": "user", "content": instruction}
    ]
    cur_input_prompt = tokenizer.apply_chat_template(cur_conversation, add_generation_prompt=True, tokenize=True)
    cur_output_ids = model.generate(input_ids=torch.LongTensor([cur_input_prompt]).to(model.device), max_new_tokens=1000)
    cur_generated_ids = cur_output_ids[0][len(cur_input_prompt):]
    return tokenizer.decode(cur_generated_ids, skip_special_tokens=True)
```

7. Deployment

Sau khi huấn luyện mô hình, triển khai ứng dụng sử dụng thư viện Gradio của Huggingface như sau: Space



Hình 4: Deployment.

Phần 3. Câu hỏi trắc nghiệm

Câu hỏi 1 Trong RLHF, vai trò của Reward Model là gì?
a) Sinh dữ liệu mới cho mô hình
b) Đánh giá mức độ phù hợp của phản hồi
c) Thay thế hoàn toàn Loss Function
d) Tăng tốc quá trình training

Câu hỏi 2 Khi huấn luyện Reward Model, loss function phổ biến là gì?
a) Triplet Loss
b) Cross-Entropy Loss
c) Margin Loss giữa chosen và rejected
d) Reconstruction Loss

Câu hỏi 3 PPO trong RLHF có vai trò gì?
a) Tối ưu likelihood sinh phản hồi
b) Tối ưu trực tiếp Reward Model
c) Cập nhật policy model để tối đa hóa reward
d) Dự đoán xác suất từ prompt

Câu hỏi 4 Ưu điểm nổi bật của DPO so với RLHF là gì?
a) Huấn luyện nhanh hơn và ổn định hơn
b) Sinh ra phản hồi dài hơn
c) Tối ưu tốt hơn reward tuyệt đối
d) Cần ít dữ liệu hơn để huấn luyện reward model

Câu hỏi 5 DPO loss chủ yếu dựa trên loại hàm nào?
a) Mean Squared Error
b) Margin Ranking Loss
c) Sigmoid Binary Cross Entropy
d) KL Divergence

Câu hỏi 6 Công thức Advantage trong PPO được tính như thế nào?
a) $A(s, a) = r - V(s)$
b) $A(s, a) = V(s) - Q(s, a)$
c) $A(s, a) = \log \pi(a|s) - \log \pi_{\text{old}}(a|s)$
d) $A(s, a) = r(s, a)$

Câu hỏi 7 Trong DPO, tham số β dùng để làm gì?
a) Làm loss function smooth hơn
b) Điều chỉnh độ sắc nét giữa các xác suất
c) Kiểm soát learning rate
d) Giảm memory usage trong training

Câu hỏi 8 Đặc điểm chung của DPO và RLHF?
a) Đều cần human feedback data
b) Đều sử dụng reward model
c) Đều yêu cầu reward số tuyệt đối

d) Đều yêu cầu PPO cho policy update

Câu hỏi 9 Trong DPO, nếu β tiến về 0, loss function sẽ gần giống với:
a) Cross-entropy Loss giữa chosen và rejected
b) Margin Ranking Loss với margin = 1
c) Uniform random loss
d) KL Divergence loss giữa two outputs

Câu hỏi 10 Nếu reward model huấn luyện chưa tốt, điều gì xảy ra với RLHF?
a) Policy học hành vi sai lệch
b) Policy convergence nhanh hơn
c) Mô hình đạt loss thấp hơn
d) Không ảnh hưởng nhiều vì PPO cân bằng lại

Phần 4. Phụ lục

1. **Code**: Thư mục chứa CODE
2. **Code**: Tài liệu code trên GITHUB
3. **Rubric**:

Phần	Kiến Thức	Đánh Giá
1	- Hiểu rõ mô hình ngôn ngữ lớn - Các bước huấn luyện các LLMs	- Các bước thực hiện trong quá trình fine-tuning LLMs với PPO và DPO
2.	- Các sử tạo bộ dữ liệu cho bài toán - Kỹ thuật DPO áp dụng cho bộ dữ liệu tiếng việt đã tạo	- Sử dụng thư viện transformers, trl để huấn luyện DPO - So sánh với PPO.

- Hết -