AI VIET NAM
@aivietnam.edu.vn

Module 10 – Project

Preference Tuning with LLMs
RLHF - DPO

Code - Data

Github

Nguyen Quoc Thai
MSc in Computer Science
STA Tran Minh Nam

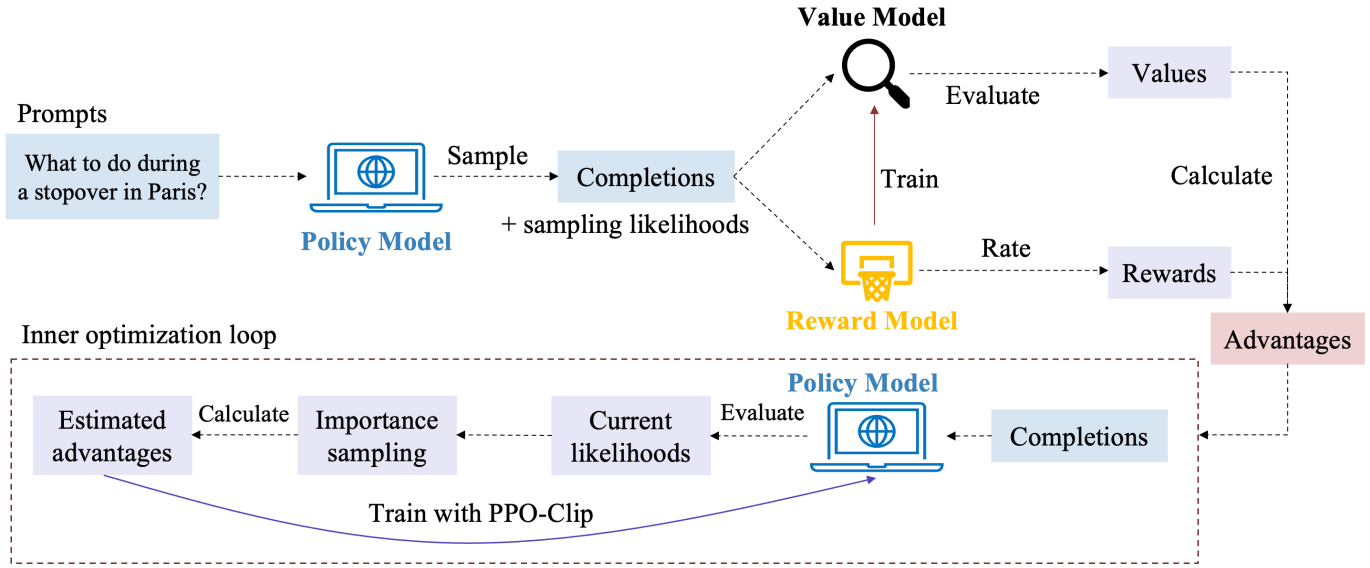
Objectives

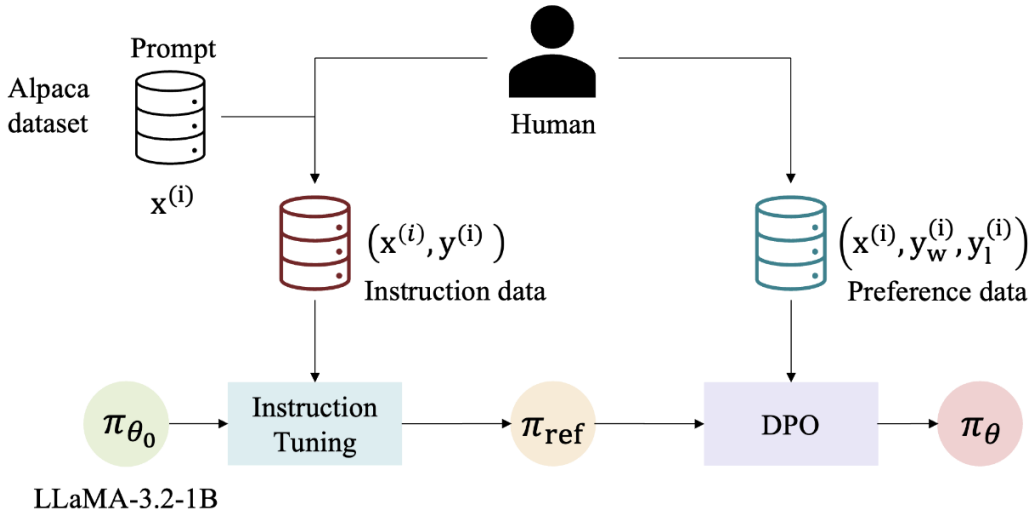
Preference Tuning

- Training LLMs
- Reward Modeling
- Reinforcement Learning in LLMs

Direct Preference Optimization

- Limit of RLHF
- Objective Function of DPO
- DPO for Vietnamese Custom Dataset





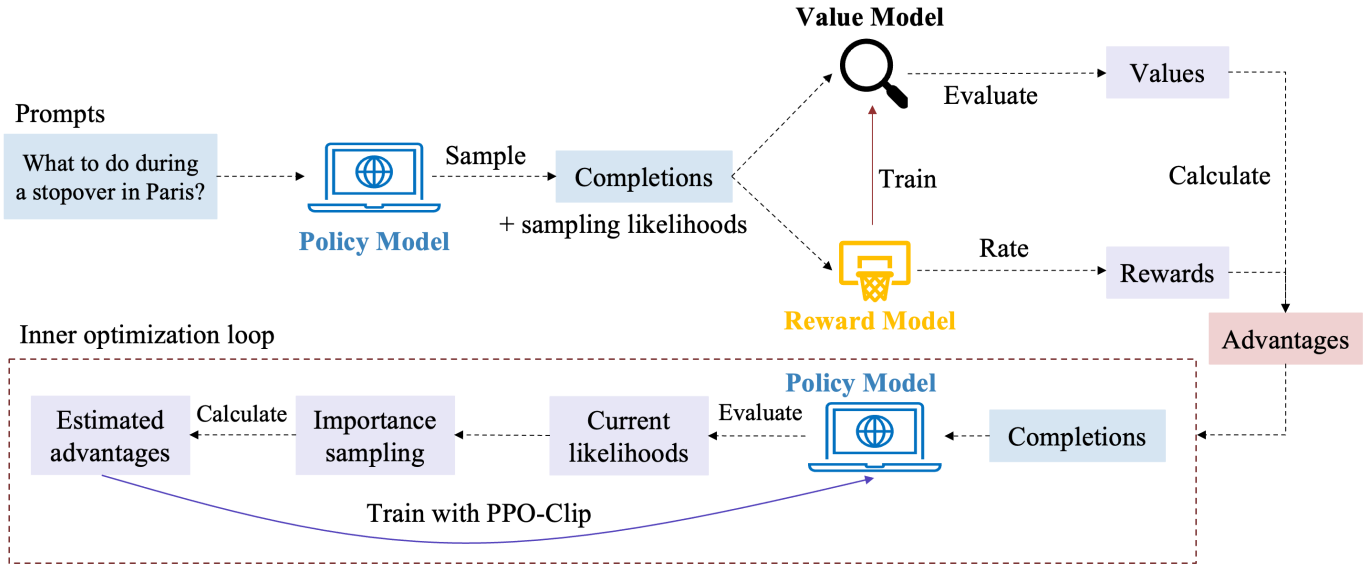
Outline

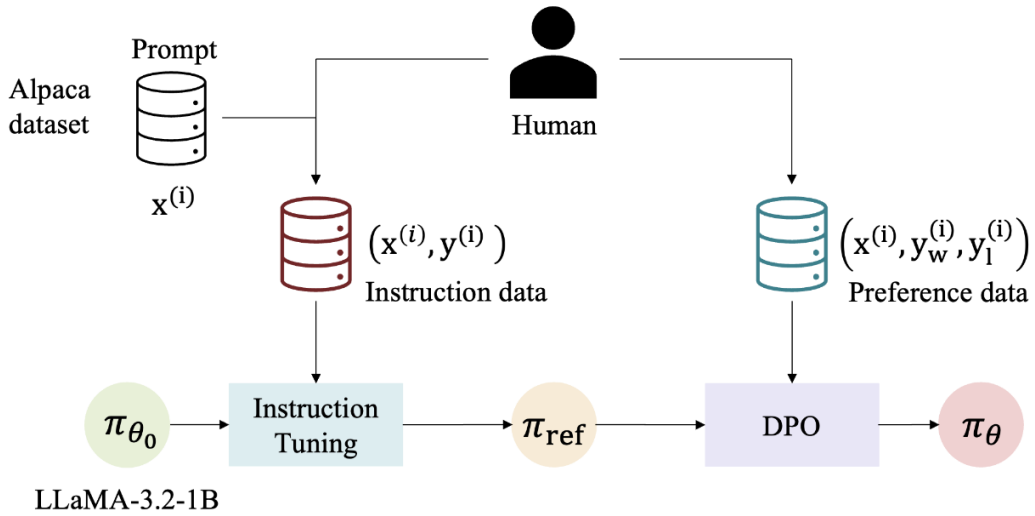
SECTION 1

Preference Tuning

SECTION 2

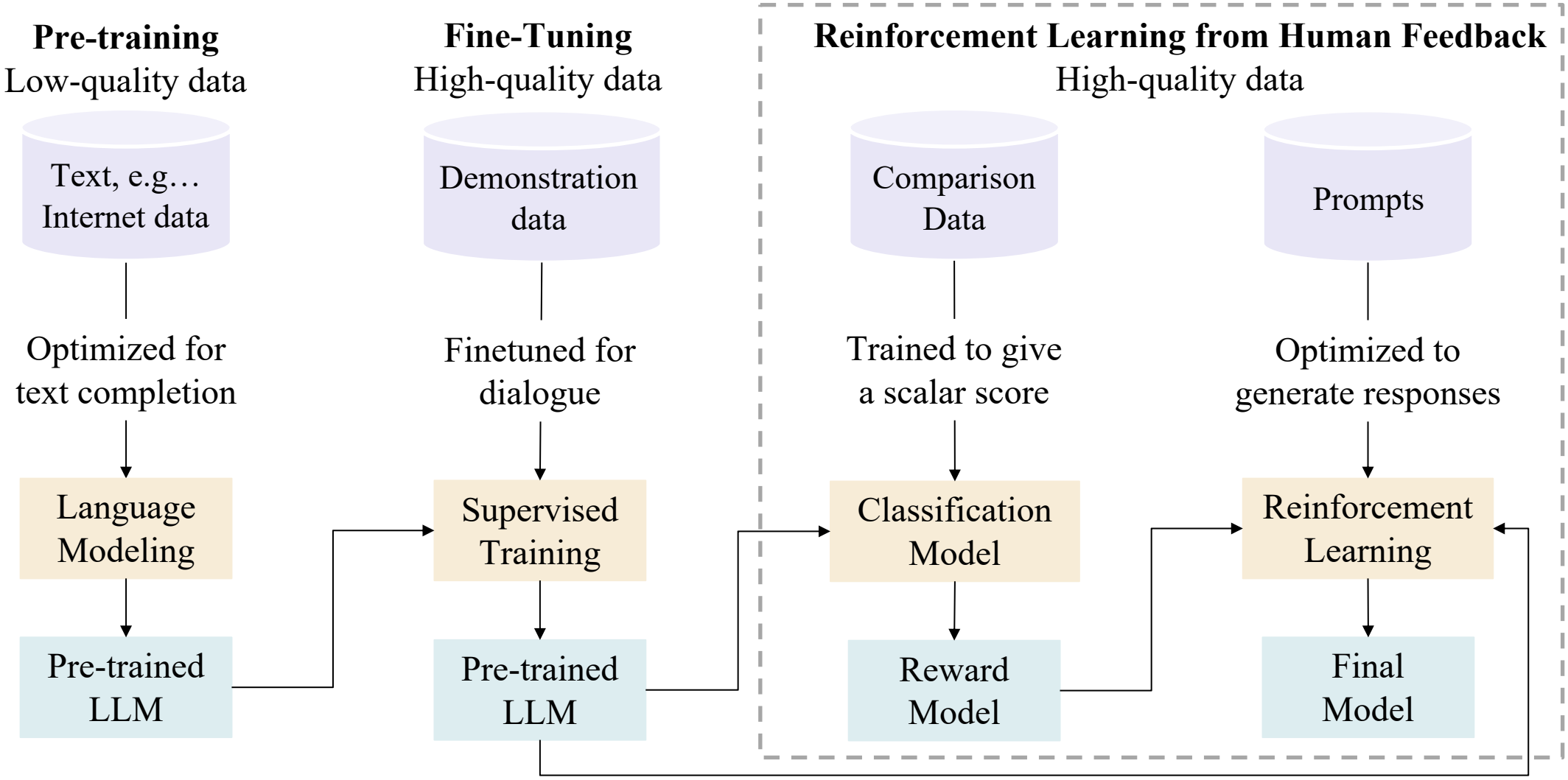
DPO





Preference Tuning

Training LLMs with Reinforcement Learning



Preference Tuning

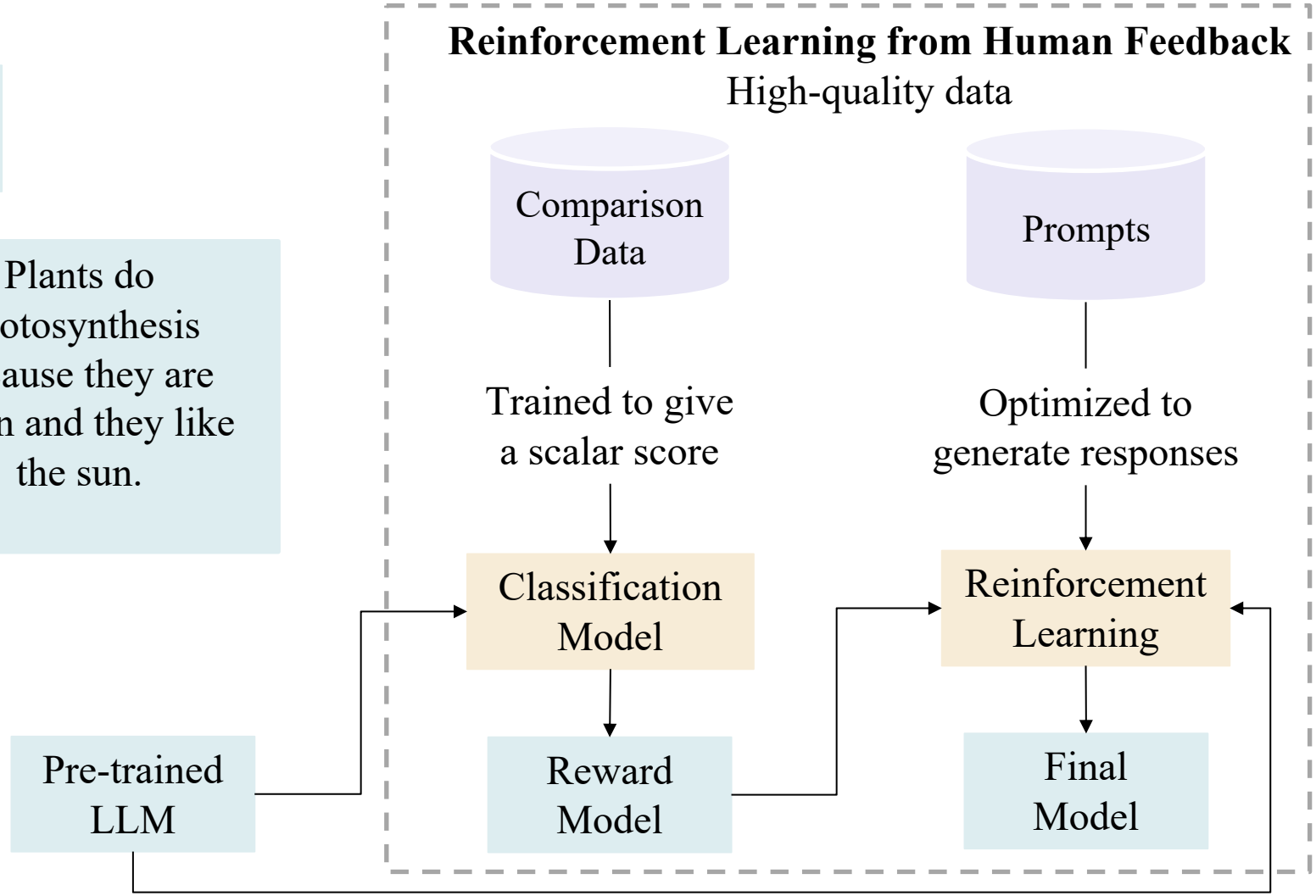
Training LLMs with Reinforcement Learning

Explain what photosynthesis is

Photosynthesis is the process by which plants make their own food using sunlight, carbon dioxide, and water.

Plants do photosynthesis because they are green and they like the sun.

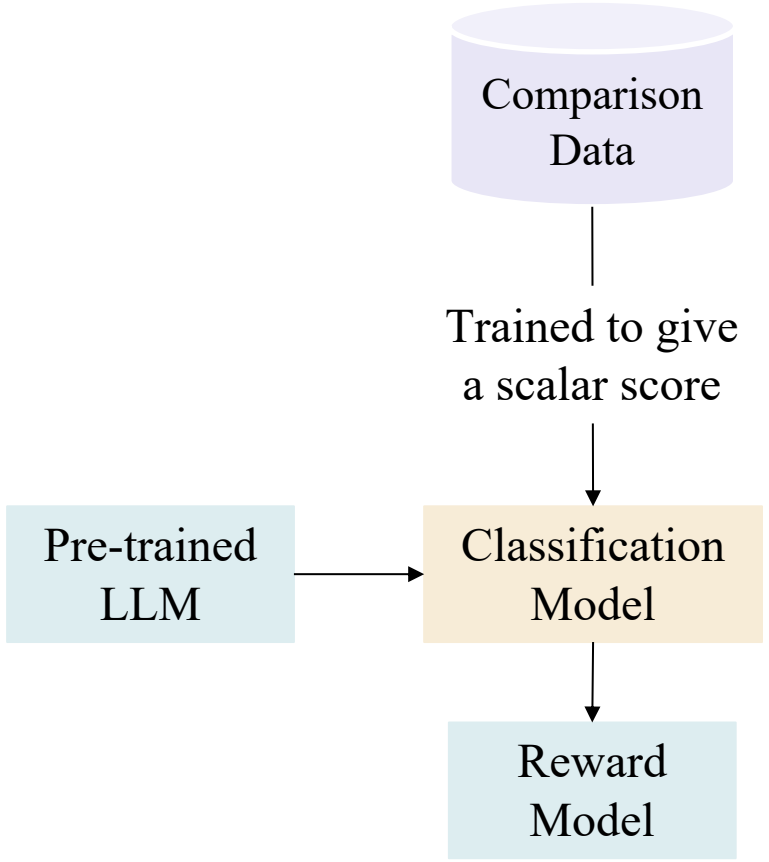
Best answer?

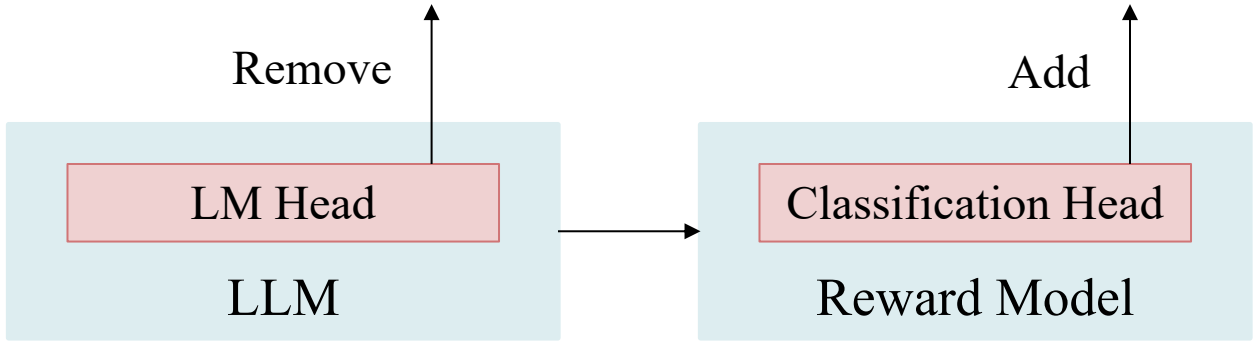


Preference Tuning

Reward Modeling

- Training a model to predict a score on a given input (a pair of prompt – response)
- A classification or regression task
- Tweak the LLM to become a reward model

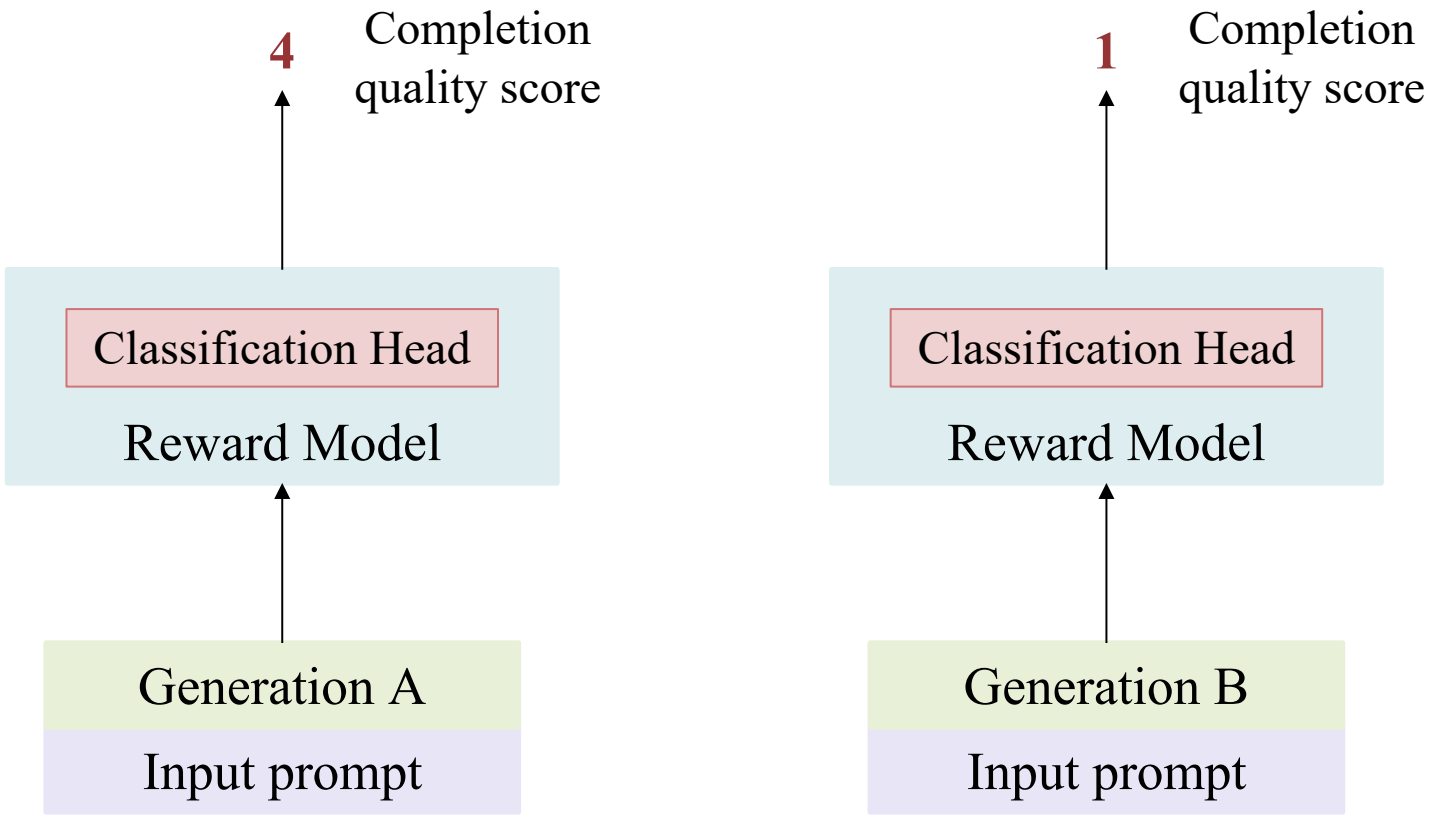




Preference Tuning

Reward Modeling

- The inputs and Outputs of a Reward Model (Regression Task)



Input prompt

Explain what photosynthesis is

Generation A

Photosynthesis is the process by which plants make their own food using sunlight, carbon dioxide, and water.

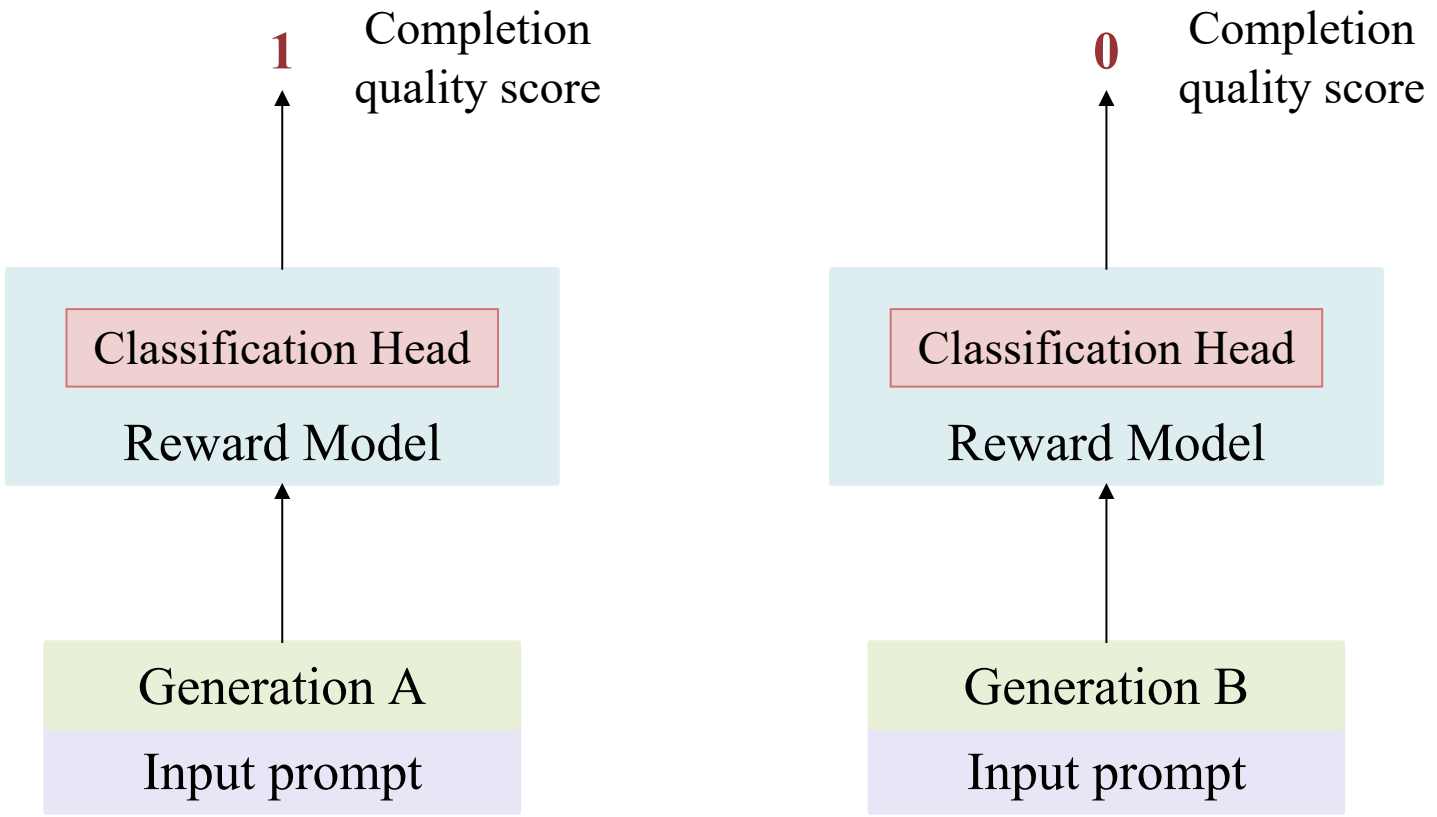
Generation B

Plants do photosynthesis because they are green and they like the sun.

Preference Tuning

Reward Modeling

- The inputs and Outputs of a Reward Model (Classification Task)



Input prompt

Explain what photosynthesis is

Generation A

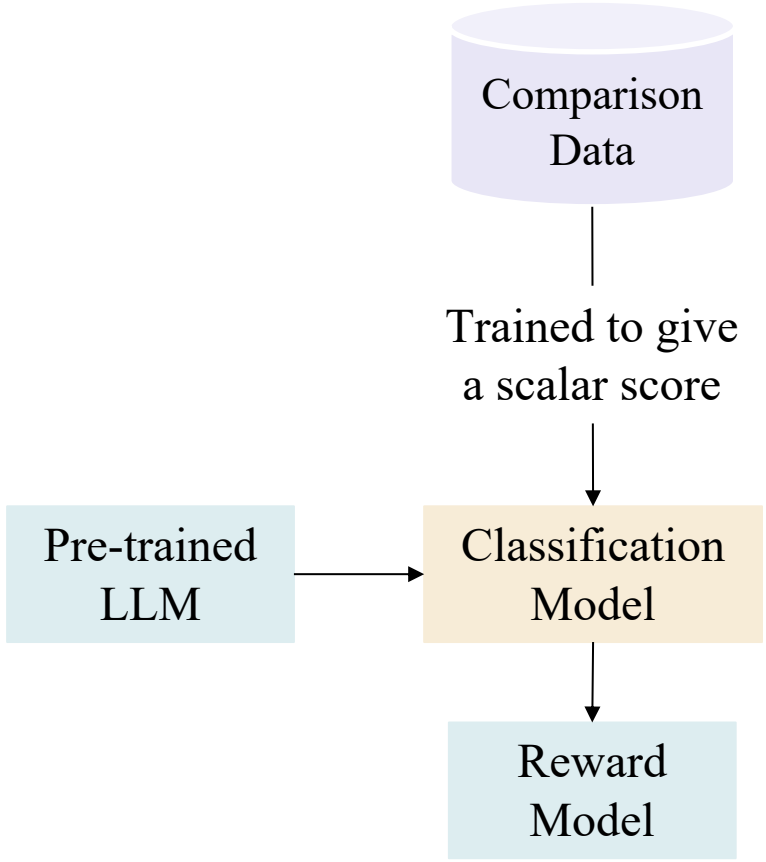
Photosynthesis is the process by which plants make their own food using sunlight, carbon dioxide, and water.

Generation B

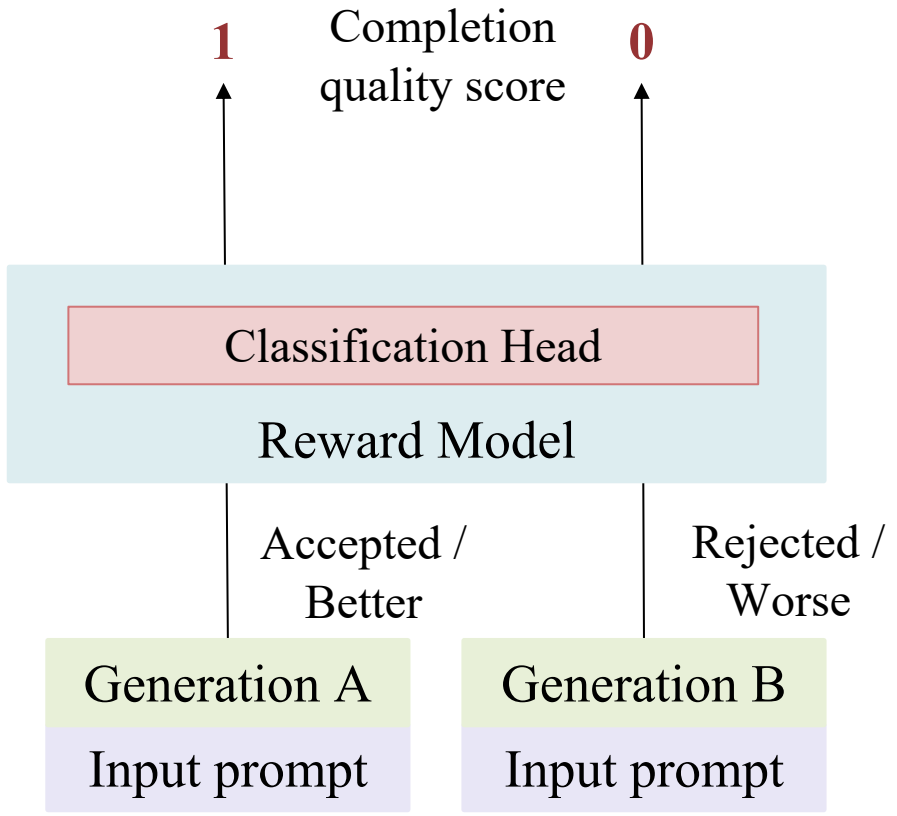
Plants do photosynthesis because they are green and they like the sun.

Preference Tuning

Reward Modeling

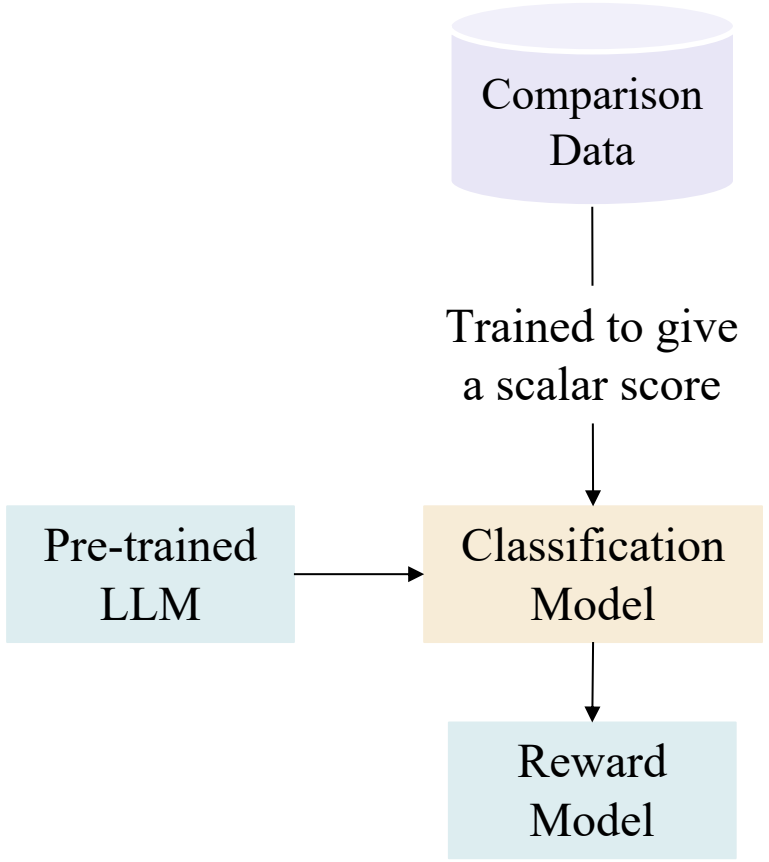


- Objective function:
- Accepted score should be larger than rejected score



Preference Tuning

Reward Modeling



❖ Objective function:

the reward model: $\boldsymbol{r_\theta}$

$\boldsymbol{x}$: the prompt, $\boldsymbol{y_w}$: the better completion, $\boldsymbol{y_l}$: the worse completion

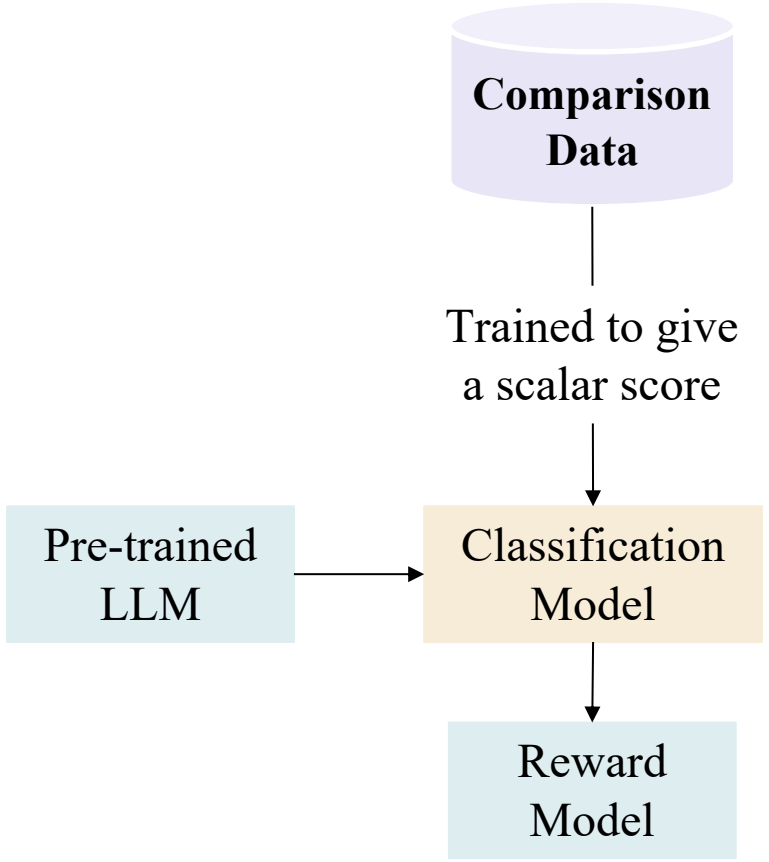
Reward on better completion

$$\text{loss}(\theta) = \text{E}_{(x,\text{y}_\text{w},\text{y}_\text{l})\sim D}\left[\log\left(\sigma\left(\text{r}_\theta(\text{x},\text{y}_\text{w}) - \text{r}_\theta(\text{x},\text{y}_\text{l})\right)\right)\right]$$

Reward on worse completion

Preference Tuning

Reward Modeling

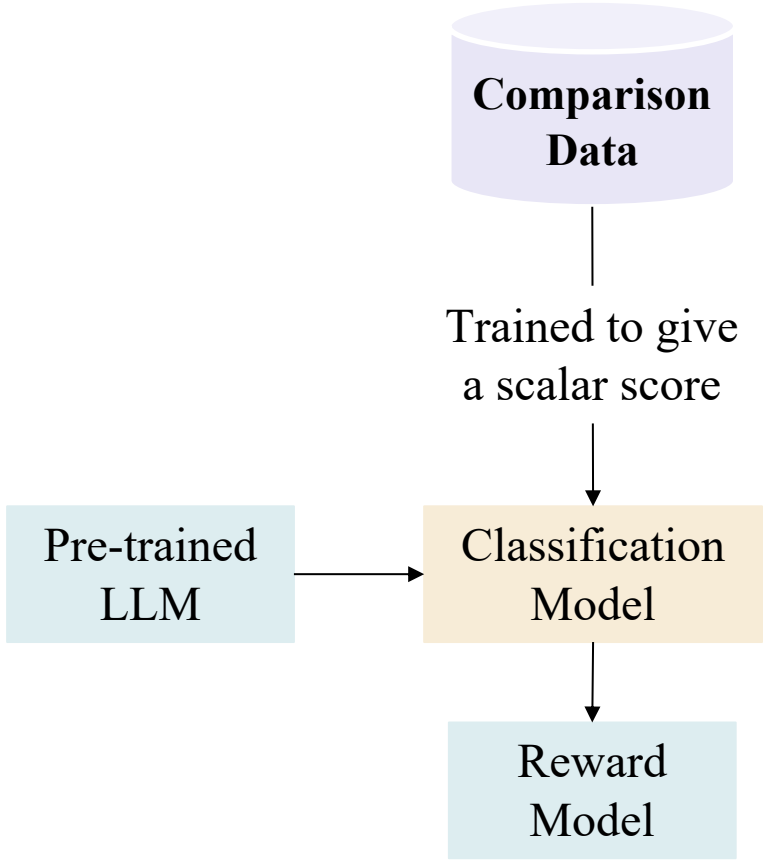


- Dataset

```python
from transformers import GPT2Tokenizer
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
tokenizer.pad_token = tokenizer.eos_token

samples = [
    {
        "prompt": "What is the capital of France?",
        "chosen": "Paris is the capital of France.",
        "rejected": "France is a city in Paris."
    },
    {
        "prompt": "What is 2 + 2?",
        "chosen": "The answer is 4.",
        "rejected": "2 plus 2 equals 22."
    }
]
```

Preference Tuning

Reward Modeling

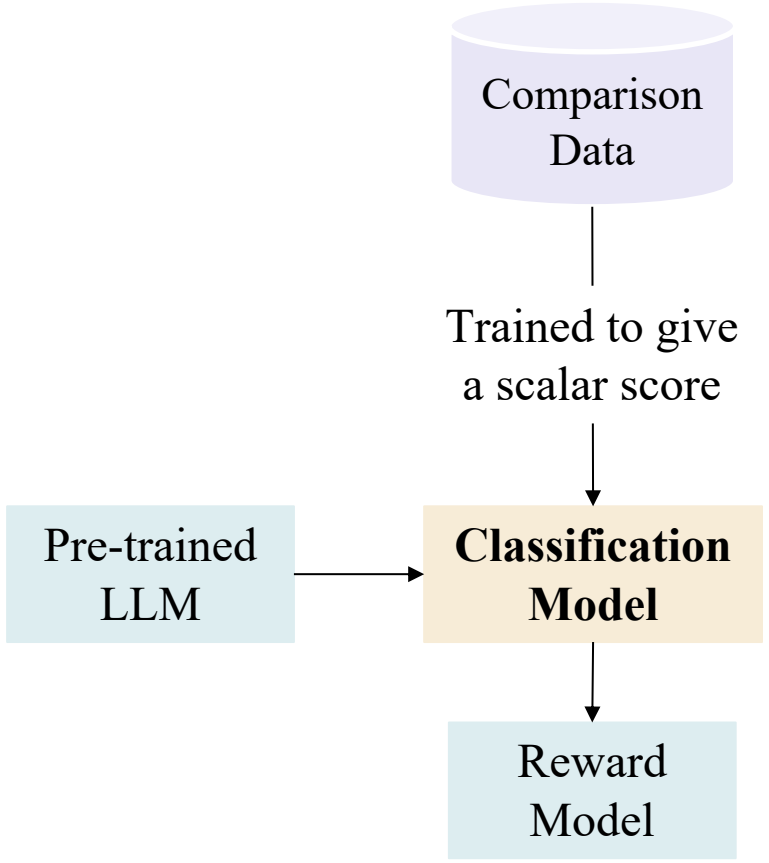


- Encoding

```python
def encode_batch(samples):
    chosen_texts = [s["prompt"] + " " + s["chosen"] for s in samples]
    rejected_texts = [s["prompt"] + " " + s["rejected"] for s in samples]

    chosen = tokenizer(
        chosen_texts, padding=True, truncation=True, return_tensors="pt"
    )
    rejected = tokenizer(
        rejected_texts, padding=True, truncation=True, return_tensors="pt"
    )
    return chosen, rejected
```

Preference Tuning

Reward Modeling

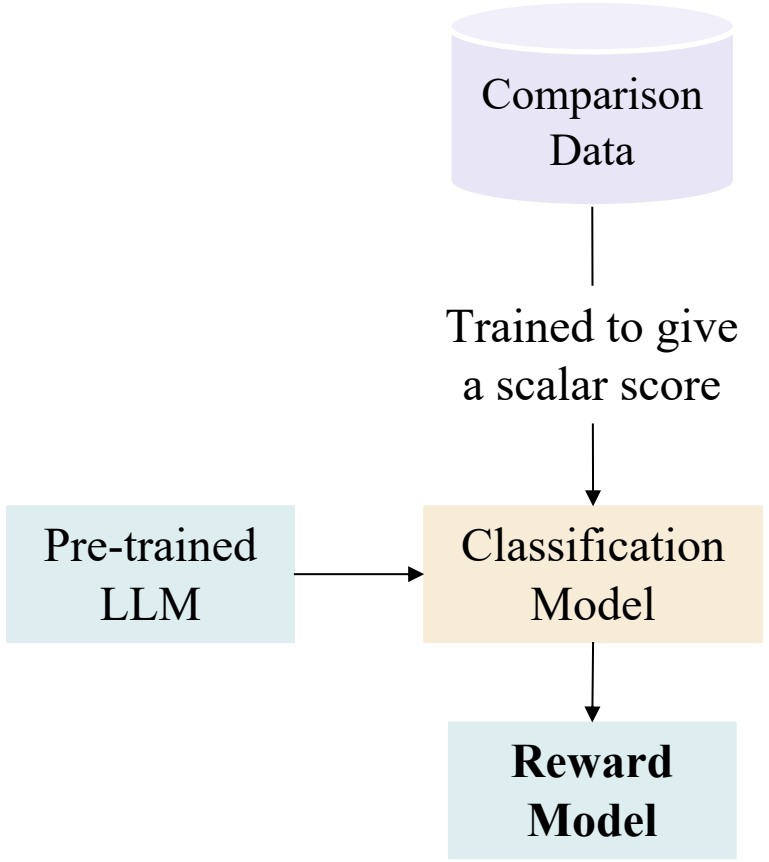


❖ Model

```python
import torch
import torch.nn as nn
from transformers import GPT2Model

class GPT2RewardModel(nn.Module):
    def __init__(self, model_name="gpt2"):
        super().__init__()
        self.transformer = GPT2Model.from_pretrained(model_name)
        self.value_head = nn.Linear(self.transformer.config.hidden_size, 1)

    def forward(self, input_ids, attention_mask=None):
        outputs = self.transformer(
            input_ids=input_ids, attention_mask=attention_mask
        )
        last_hidden = outputs.last_hidden_state
        value = self.value_head(last_hidden[:, -1, :])
        return value.squeeze(-1)
```

Preference Tuning

Reward Modeling

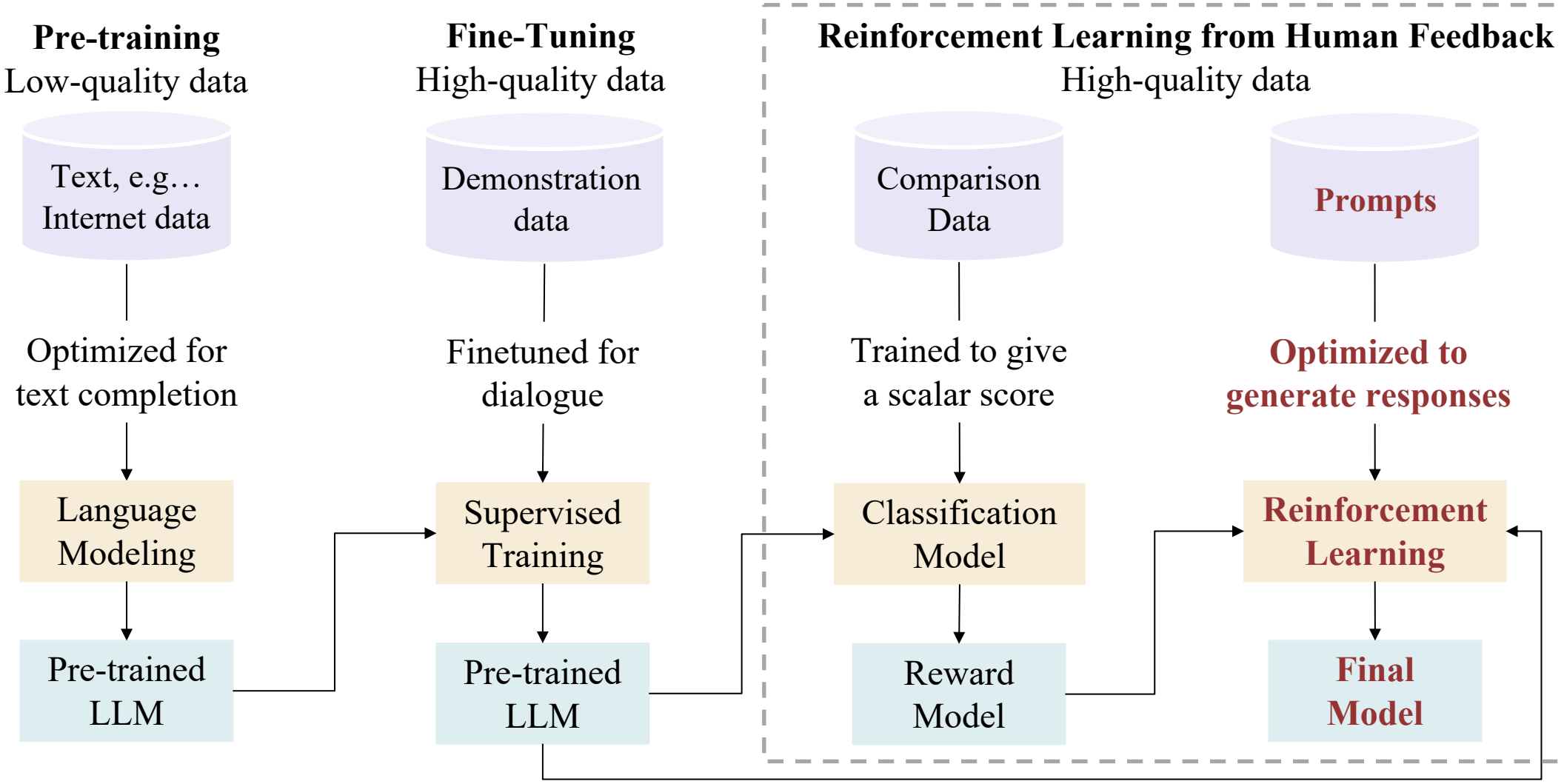


- Training

```
model = GPT2RewardModel()
optimizer = torch.optim.Adam(model.parameters(), lr=2e-5)
loss_fn = nn.MarginRankingLoss(margin=1.0)

chosen, rejected = encode_batch(samples)
chosen = {k: v for k, v in chosen.items()}
rejected = {k: v for k, v in rejected.items()}

for epoch in range(200):
    model.train()
    r_chosen = model(**chosen)
    r_rejected = model(**rejected)
    target = torch.ones_like(r_chosen)

    loss = loss_fn(r_chosen, r_rejected, target)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

Preference Tuning

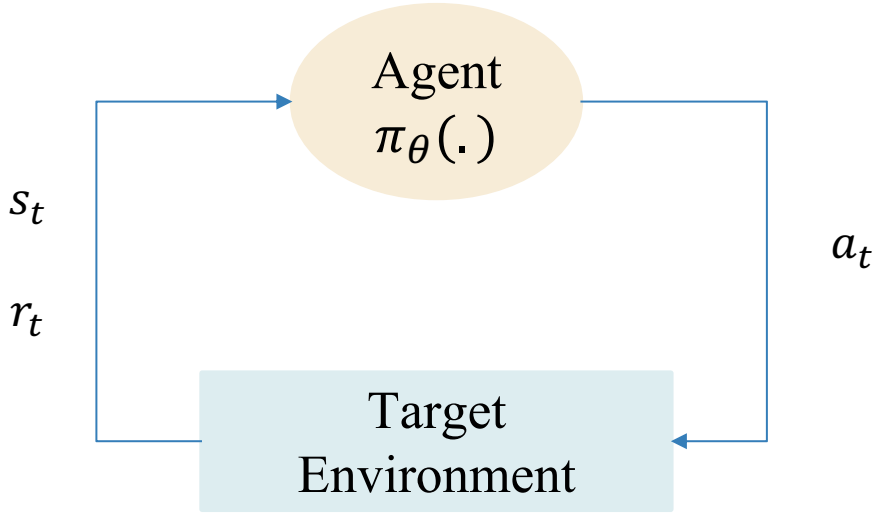
Training LLMs with Reinforcement Learning

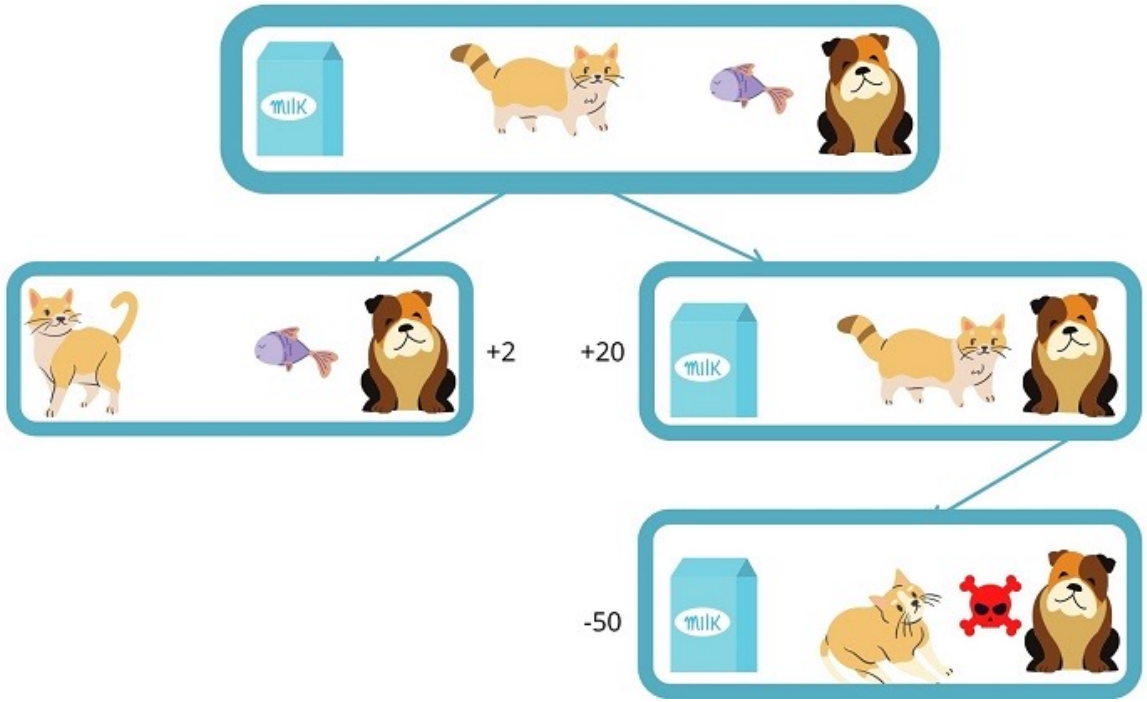


Preference Tuning

Reinforcement Learning Basics

- Reinforcement learning is a type of machine learning where an agent learns to make decisions by interacting with an environment and receiving rewards or penalties.





s_t : state

r_t : reward

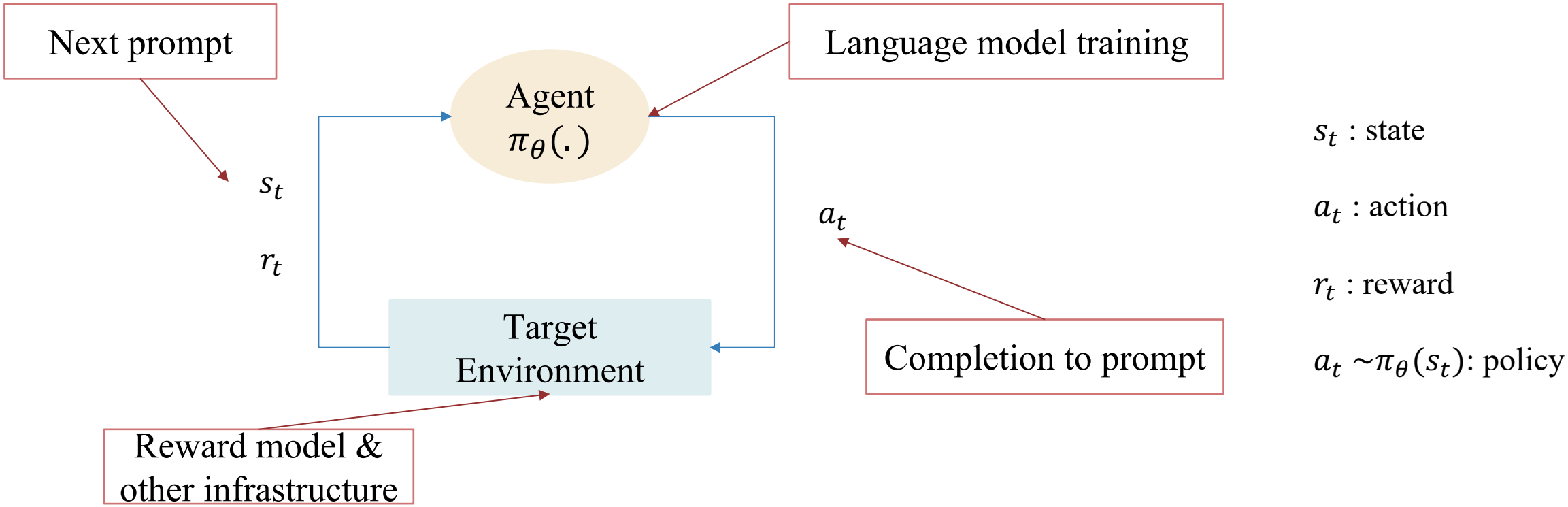
a_t : action

$a_t \sim \pi_\theta(s_t)$: policy

Preference Tuning

Reinforcement Learning Basics in Language

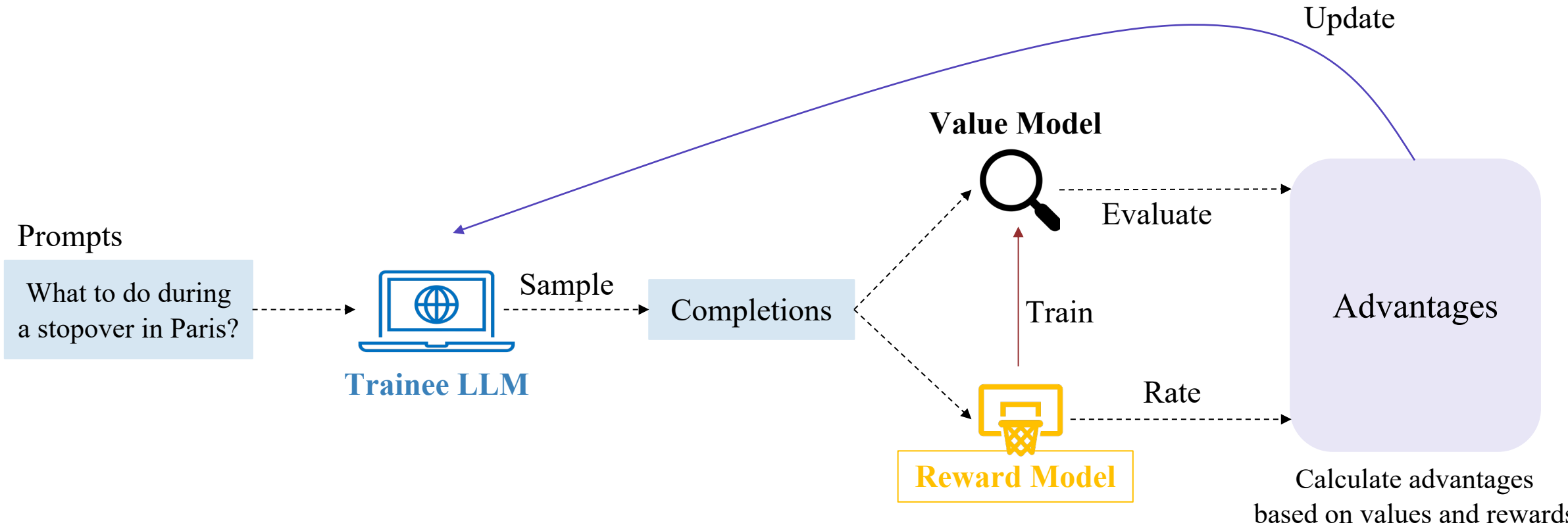
- Reinforcement learning is a type of machine learning where an agent learns to make decisions by interacting with an environment and receiving rewards or penalties.



Preference Tuning

Advantage Actor-Critic

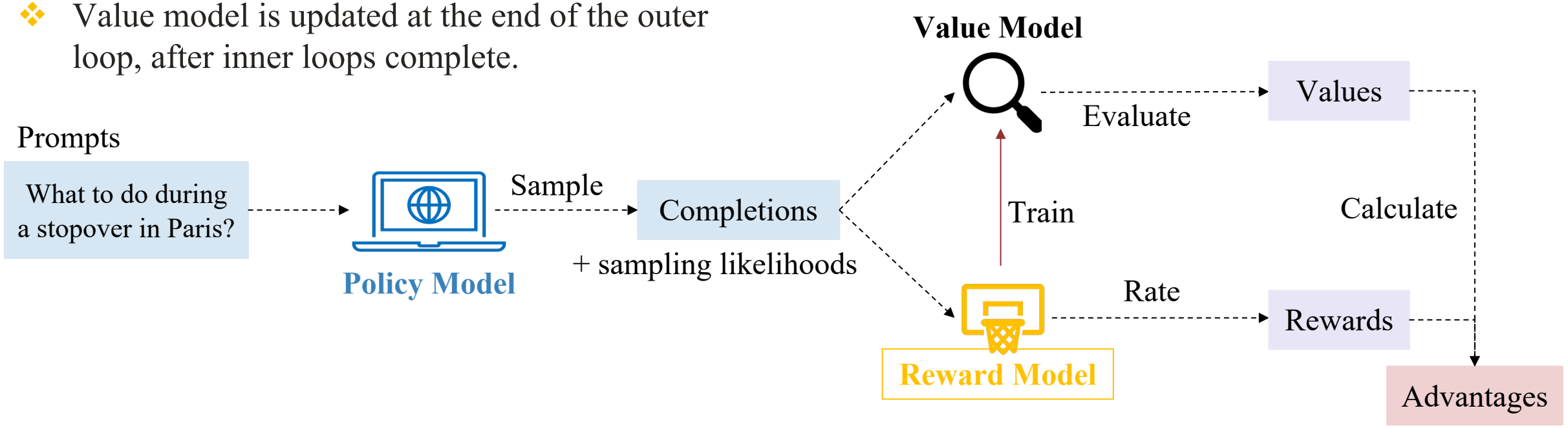
- Value and trainee LLM are trained along one another: A2C update is followed by a value update before sampling again.



Preference Tuning

PPO – Proximal Policy Optimization

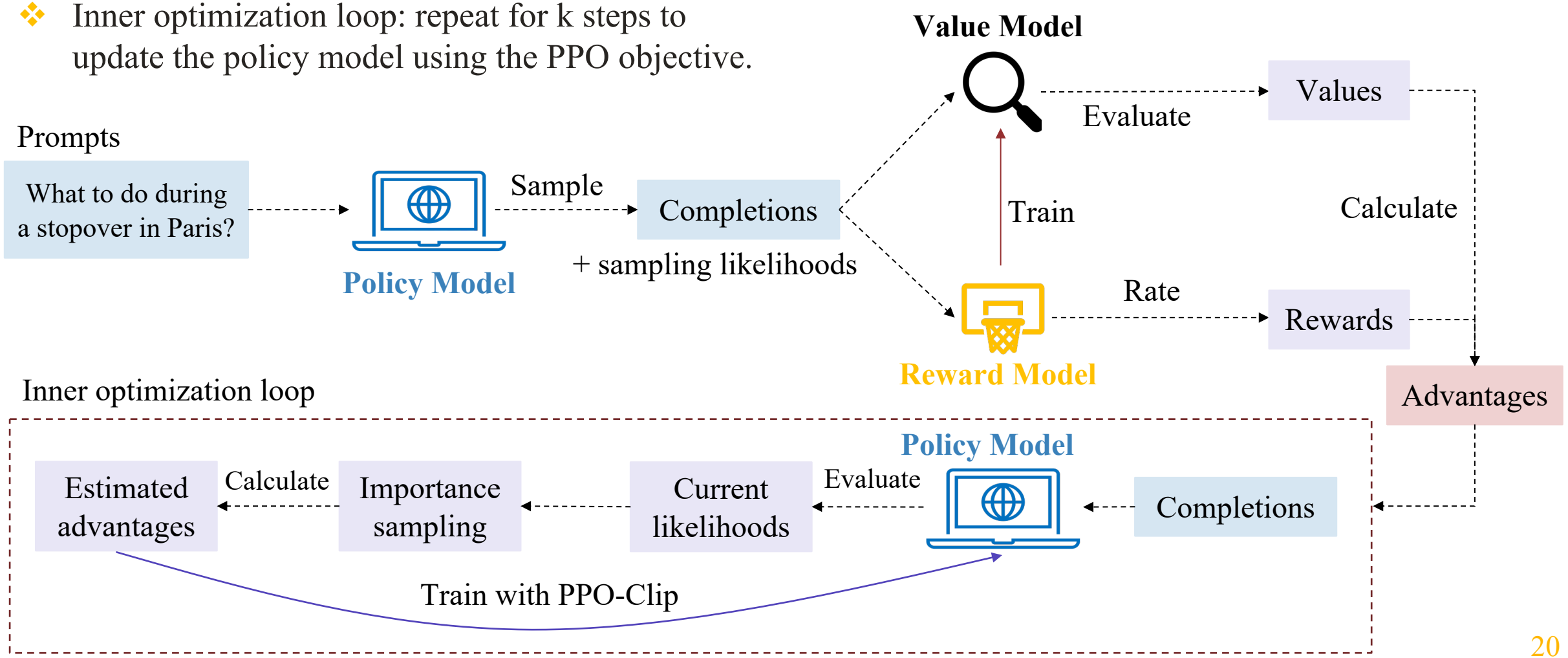
- Outer generation loop
- Value model is updated at the end of the outer loop, after inner loops complete.



Preference Tuning

PPO – Proximal Policy Optimization

- Inner optimization loop: repeat for k steps to update the policy model using the PPO objective.



Preference Tuning

PPO – Proximal Policy Optimization

- Maximizes or minimizes the likelihood of actions given their estimated advantage, within the constraints of clipping.
- Importance sampling ratio to estimate current advantage.
- Minimum and clipping to constrain importance sampling ratio

$$L_{PPO_\theta} = \underset{\theta}{\text{argmax}} \sum_{x \in D} \sum_{i=1}^{n} \min\left(A_\varphi r_\theta, A_\varphi clip(r_\theta, 1-\varepsilon, 1+\varepsilon)\right)$$

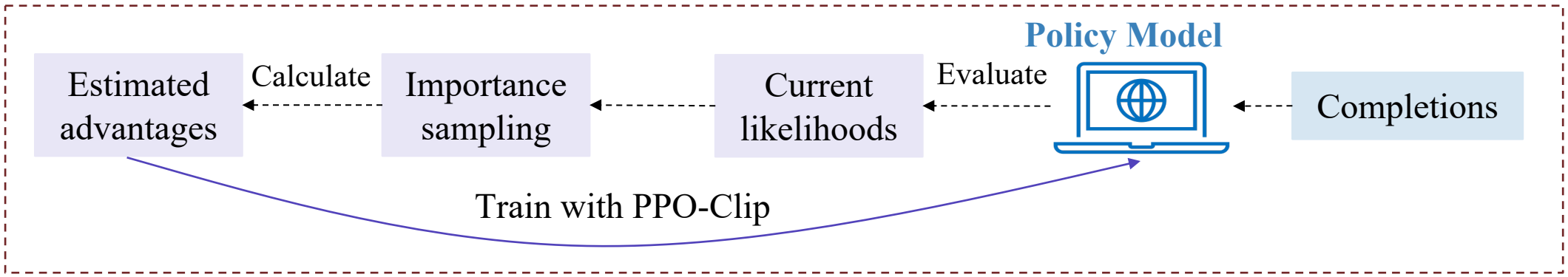
sum over samples generated in outer loop

Advantages sampling ratio to estimate current advantage

Calculate likelihood

$$r_\theta = \frac{\pi_\theta(x_i|x_{<i})}{\pi_{\theta_{old}}(x_i|x_{<i})}$$

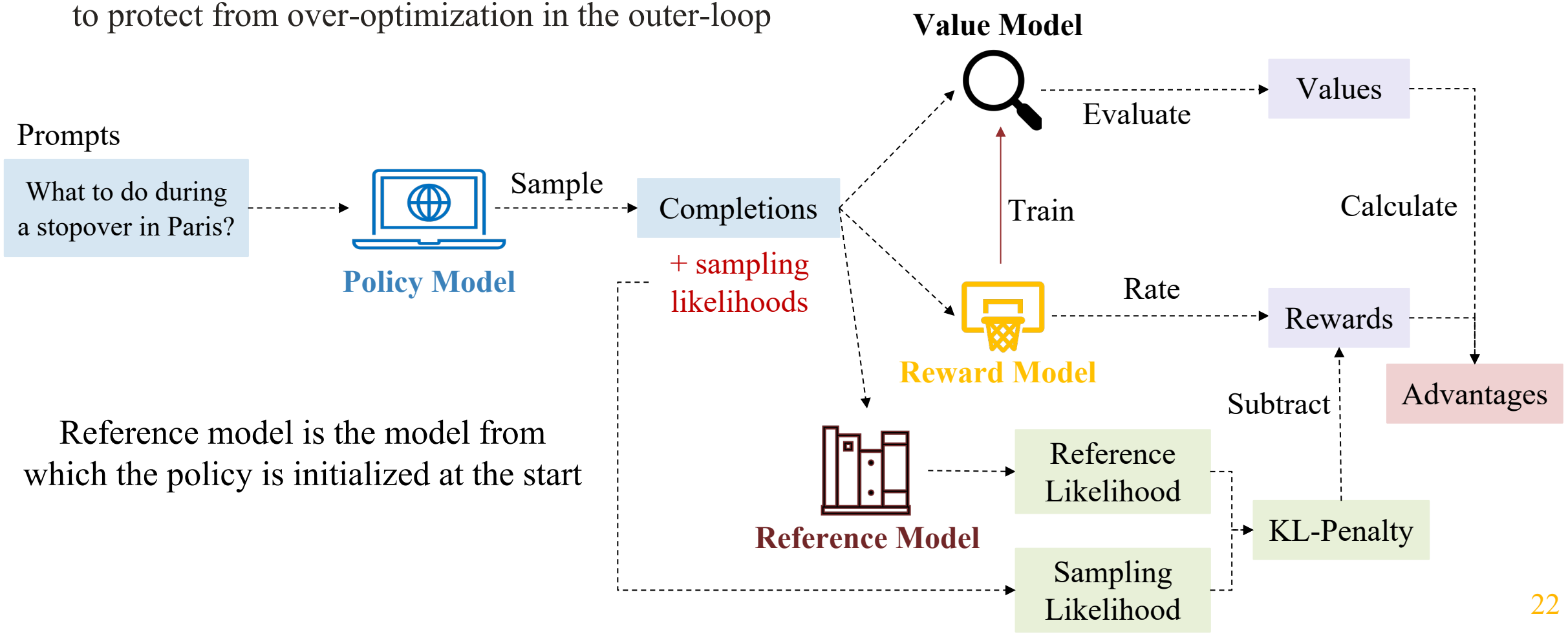
Sampling likelihood



Preference Tuning

PPO – Proximal Policy Optimization with KL Penalty

- A penalty for veering too far off from a reference model to protect from over-optimization in the outer-loop



Reference model is the model from which the policy is initialized at the start

Preference Tuning

PPO – Proximal Policy Optimization with KL Penalty

- KL-divergence between sampling policy and reference policy

$$L_{KL_\theta} = \beta D_{KL}\left(\pi_{\theta_{old}}(x_i|x_{<i}) \middle| \pi_{\theta_{ref}}(x_i|x_{<i})\right)$$

Scaling factor
for the KL-penalty

$$\boldsymbol{L_{RL} = L_{PPO_\theta} - L_{KL_\theta}}$$

- With clipping updates and KL-penalties relative to the reference policy, PPO effectively prevents drastic deviations during training, ensuring that generated outputs remain consistent with the pre-trained style.

Preference Tuning

PPO – Proximal Policy Optimization with KL Penalty

Policy Model

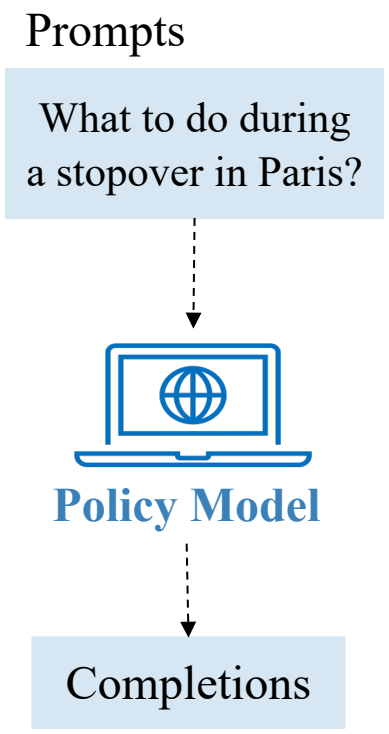
Reference Model

Reward Model

```
model_name = "gpt2"
tokenizer = AutoTokenizer.from_pretrained(model_name)
tokenizer.pad_token = tokenizer.eos_token

# Policy model
policy = AutoModelForCausalLM.from_pretrained(model_name)
policy_ref = AutoModelForCausalLM.from_pretrained(model_name)
policy.train()
policy_ref.eval()

# Reward model
reward_model = AutoModelForSequenceClassification.from_pretrained(
    model_name, num_labels=1
)
reward_model.eval()
```

Preference Tuning

PPO – Proximal Policy Optimization with KL Penalty

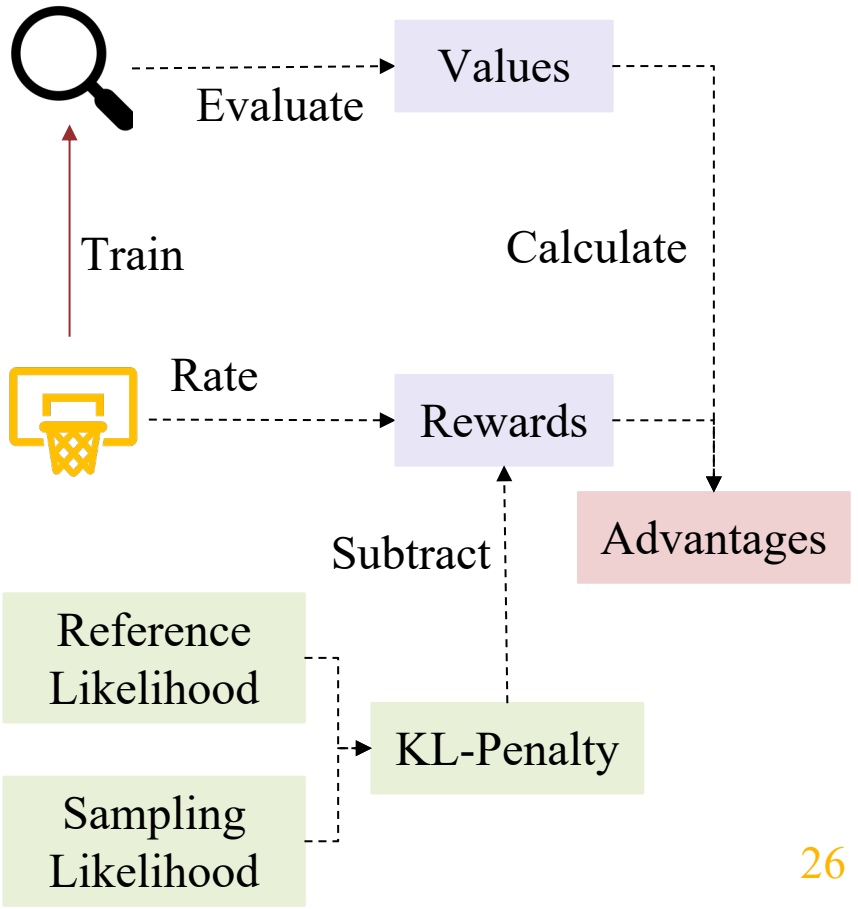



```
prompt = "I am studying"
inputs = tokenizer(
    prompt, return_tensors="pt", padding=True
)
input_ids = inputs["input_ids"]

with torch.no_grad():
    gen_ids = policy.generate(
        input_ids=input_ids,
        max_new_tokens=20,
        do_sample=True,
        top_k=50,
        temperature=1.0,
        pad_token_id=tokenizer.pad_token_id
    )
response_ids = gen_ids[:, input_ids.shape[-1]:]
query_response = torch.cat([input_ids, response_ids], dim=1)
```

Preference Tuning

PPO – Proximal Policy Optimization with KL Penalty

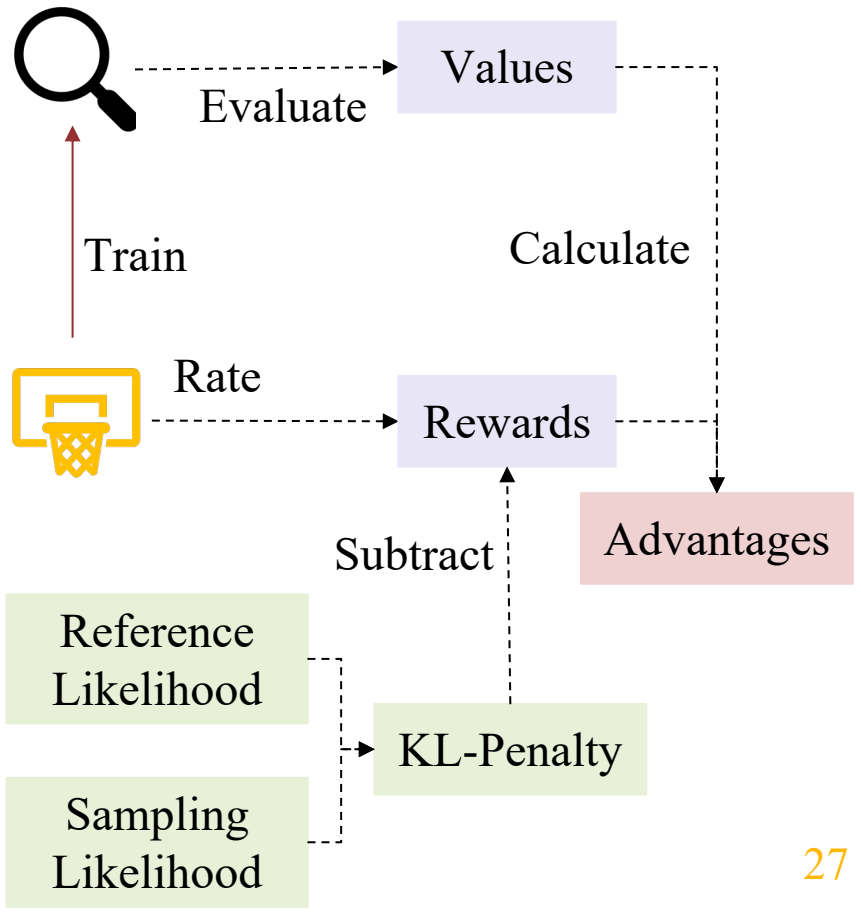
```python
# log-probs
def get_log_prob_sum(model, input_ids):
    labels = input_ids.clone()
    with torch.no_grad():
        outputs = model(input_ids=input_ids, labels=labels)
        loss = outputs.loss  # average negative log-likelihood
    return -loss  # return log-likelihood

logprob_policy = get_log_prob_sum(policy, query_response)
logprob_ref = get_log_prob_sum(policy_ref, query_response)

# reward score
with torch.no_grad():
    reward_inputs = tokenizer(
        tokenizer.decode(query_response[0], skip_special_tokens=True),
        return_tensors="pt", truncation=True, padding=True
    )
    reward = reward_model(**reward_inputs).logits.squeeze().detach()
```



Preference Tuning

PPO – Proximal Policy Optimization with KL Penalty

```
# advantage, PPO loss, KL loss
baseline = reward.detach()
advantage = reward - baseline
log_ratio = logprob_policy - logprob_ref
ratio = torch.exp(log_ratio)

# PPO-clip loss
clip_eps = 0.2
loss1 = ratio * advantage
loss2 = torch.clamp(ratio, 1 - clip_eps, 1 + clip_eps) * advantage
ppo_clip_loss = -torch.min(loss1, loss2)

# KL loss (optional penalty)
kl_loss = torch.mean(log_ratio**2)

# Loss
kl_coef = 0.01  # KL-Pen
ppo_loss = ppo_clip_loss + kl_coef * kl_loss
```




QUIZ TIME

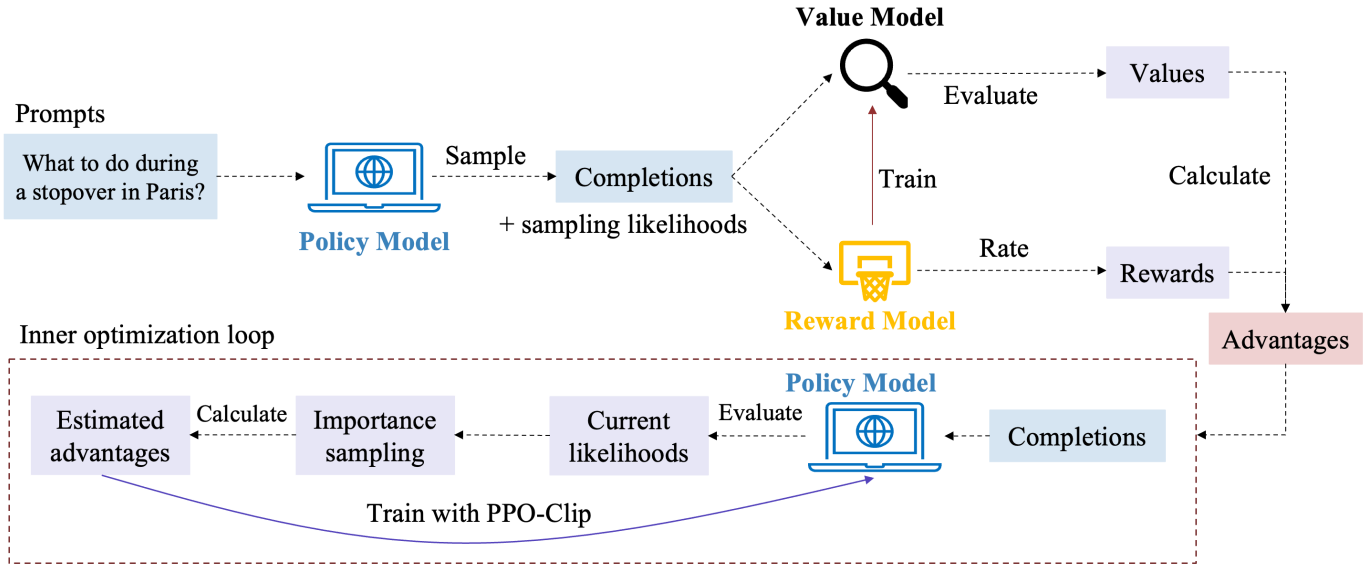
Outline

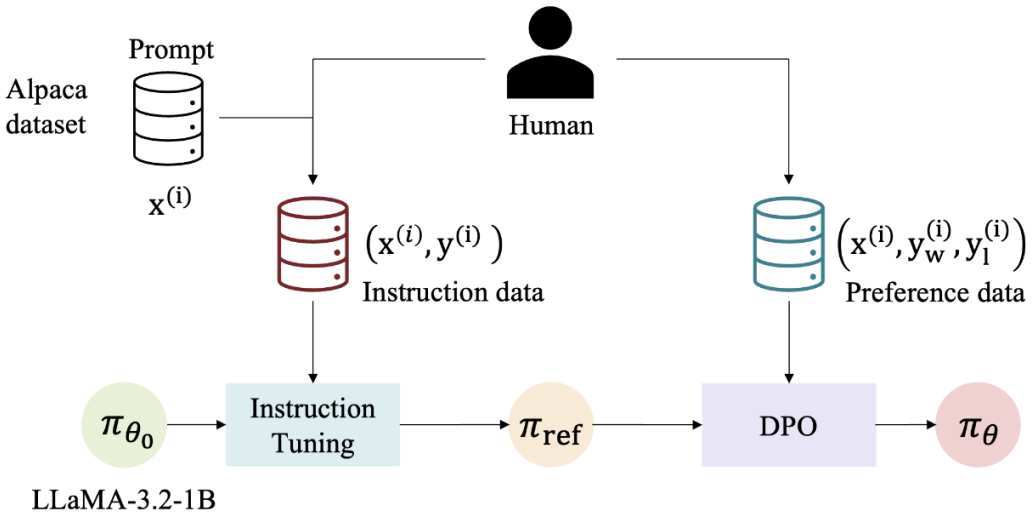
SECTION 1

Preference Tuning

SECTION 2

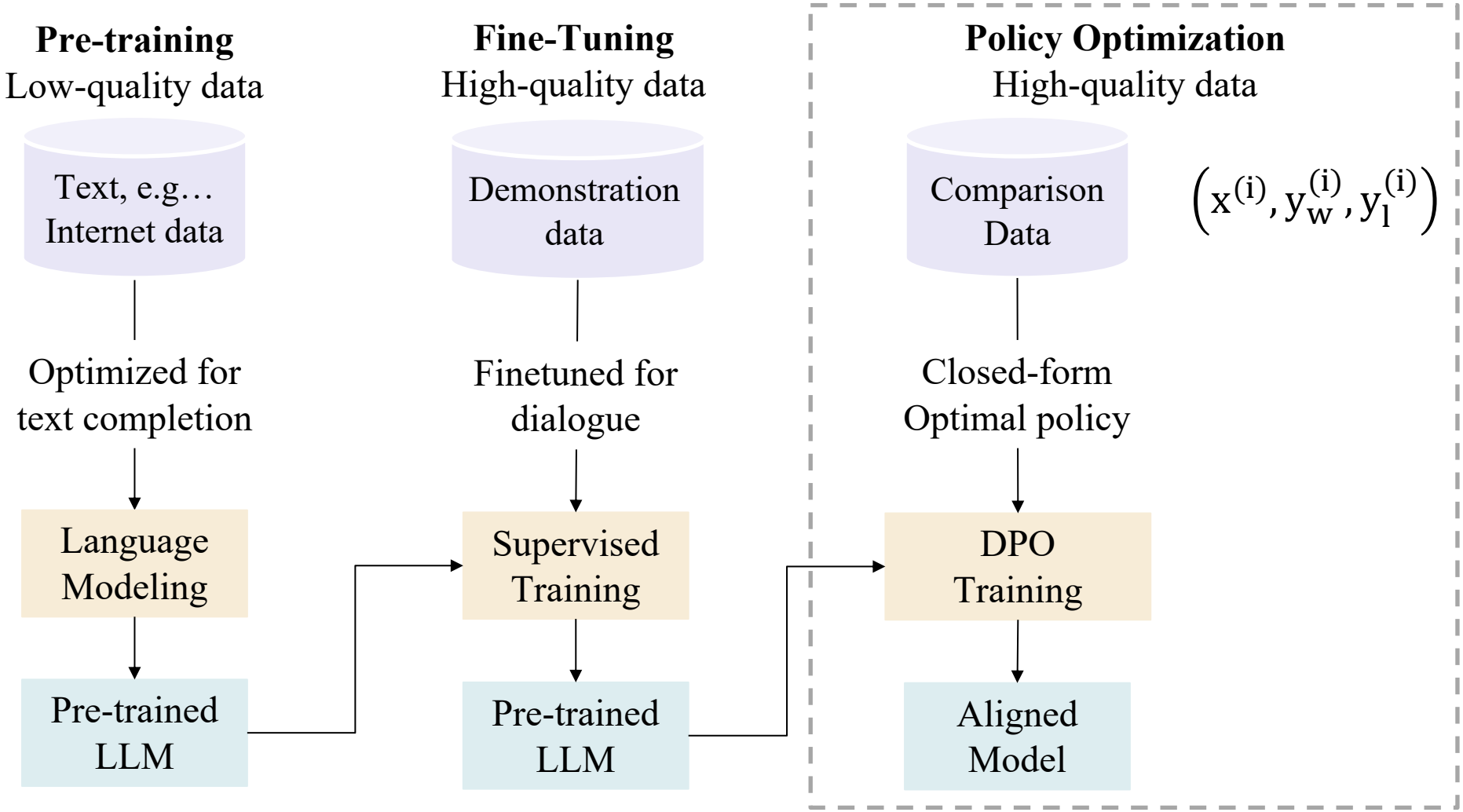
DPO





DPO

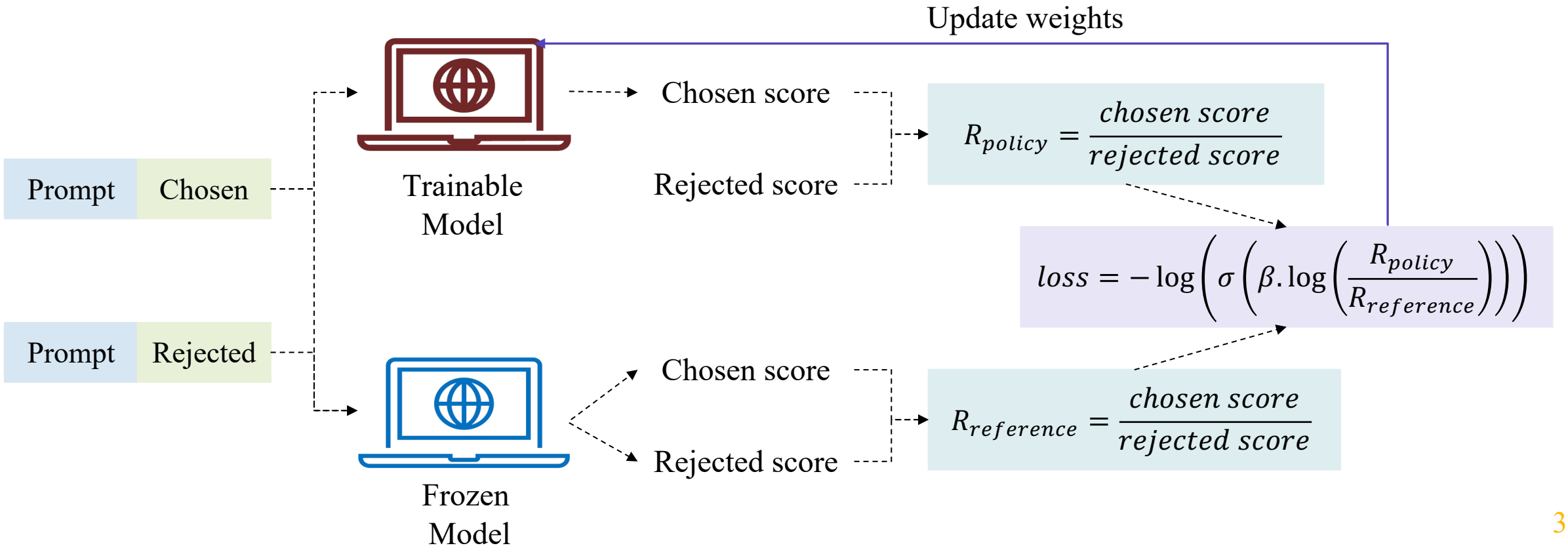
Direct Preference Optimization



DPO

Direct Preference Optimization

- The objective of DPO: improve the alignment of language models to human preferences
- In DPO, there is no reinforcement learning, and the model is directly optimized in this preference data



DPO

Direct Preference Optimization

Trainable Model

Frozen Model

```
# Load GPT-2
model_name = "gpt2"
tokenizer = AutoTokenizer.from_pretrained(model_name)
tokenizer.pad_token = tokenizer.eos_token

# Policy model (trainable)
policy_model = AutoModelForCausalLM.from_pretrained(model_name)
policy_model.train()

# Reference model (fixed, no grad)
reference_model = AutoModelForCausalLM.from_pretrained(model_name)
reference_model.eval()
```

DPO

Direct Preference Optimization

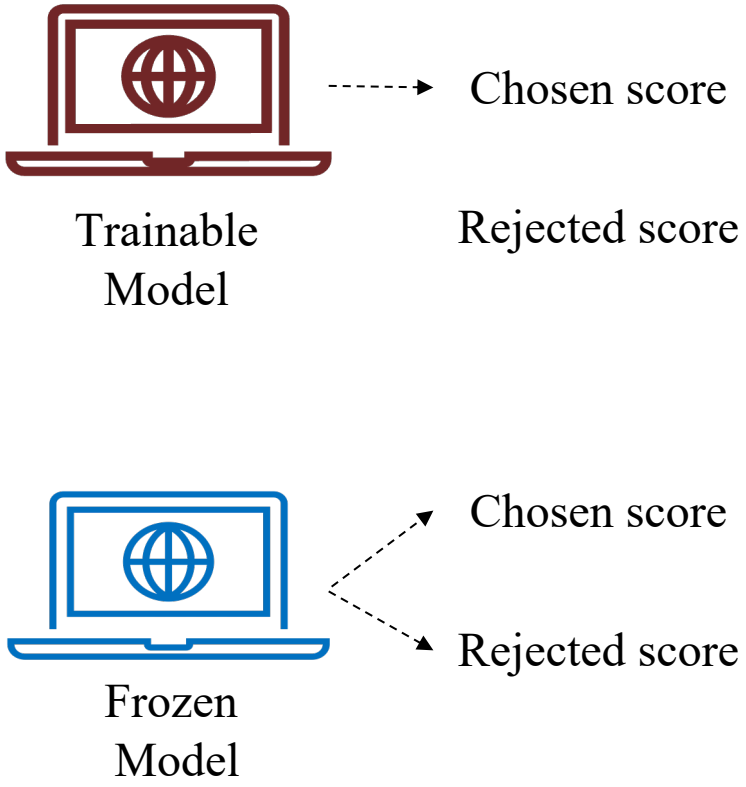
Prompt	Chosen

Prompt	Rejected

```
sample = {
    "prompt": "The weather today is",
    "chosen": " sunny and warm.",
    "rejected": " not a banana."
}

prompt = sample["prompt"]

# Tokenize
chosen = tokenizer(
    prompt + sample["chosen"], return_tensors="pt", padding=True
)
rejected = tokenizer(
    prompt + sample["rejected"], return_tensors="pt", padding=True
)
```

DPO

Direct Preference Optimization




```
# Forward pass - policy model
policy_chosen_outputs = policy_model(**chosen)
policy_rejected_outputs = policy_model(**rejected)

# Forward pass - reference model (no_grad)
with torch.no_grad():
    ref_chosen_outputs = reference_model(**chosen)
    ref_rejected_outputs = reference_model(**rejected)
```

DPO

Direct Preference Optimization

Trainable Model

Chosen score

Rejected score

Frozen Model

Chosen score

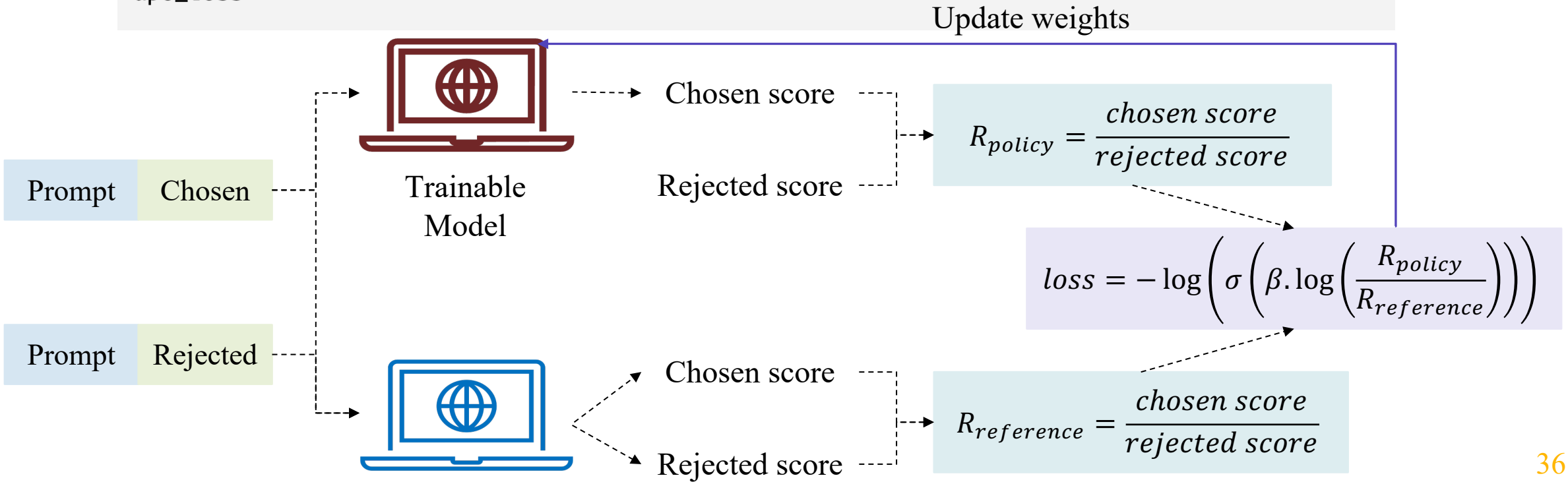
Rejected score

```python
# Get logits
def compute_log_prob(outputs, inputs):
    logits = outputs.logits[:, :-1, :]
    labels = inputs["input_ids"][:, 1:]
    log_probs = torch.gather(
        logits.log_softmax(dim=-1), dim=2, index=labels.unsqueeze(-1)
    ).squeeze(-1).sum(dim=1)
    return log_probs

policy_chosen_logp = compute_log_prob(policy_chosen_outputs, chosen)
policy_rejected_logp = compute_log_prob(policy_rejected_outputs, rejected)
ref_chosen_logp = compute_log_prob(ref_chosen_outputs, chosen)
ref_rejected_logp = compute_log_prob(ref_rejected_outputs, rejected)
```

DPO

Direct Preference Optimization

```
# DPO loss function
beta = 0.1

diff = (policy_chosen_logp - ref_chosen_logp) - (policy_rejected_logp - ref_rejected_logp)
dpo_loss = -torch.nn.functional.logsigmoid(beta * diff).mean()
dpo_loss
```



DPO for Vietnamese Dataset

Vietnamese Custom Preference Dataset

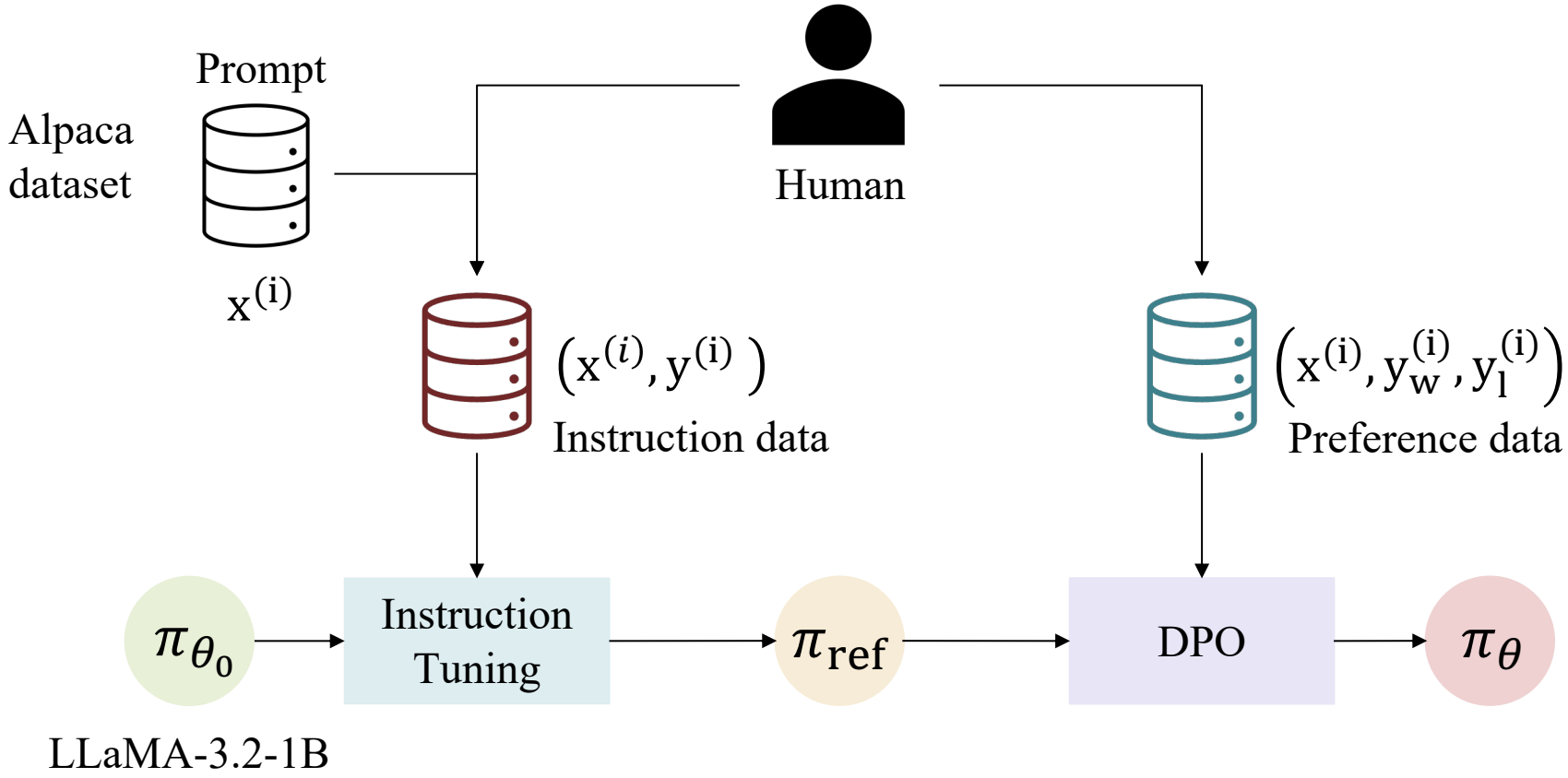
Stanford Alpaca

- Stanford Alpaca dataset:
  - 52k instruction-following samples.
  - Translated to Vietnamese.
- To convert to preference format:
  - Ground truth as “chosen” sentence.
  - Semantic search to find “rejected” answer.

- Use chat format of LLaMA-3.2-1B to perform supervised fine-tuned LLaMA-3.2-1B-Instruct on the preference dataset.

DPO for Vietnamese Dataset

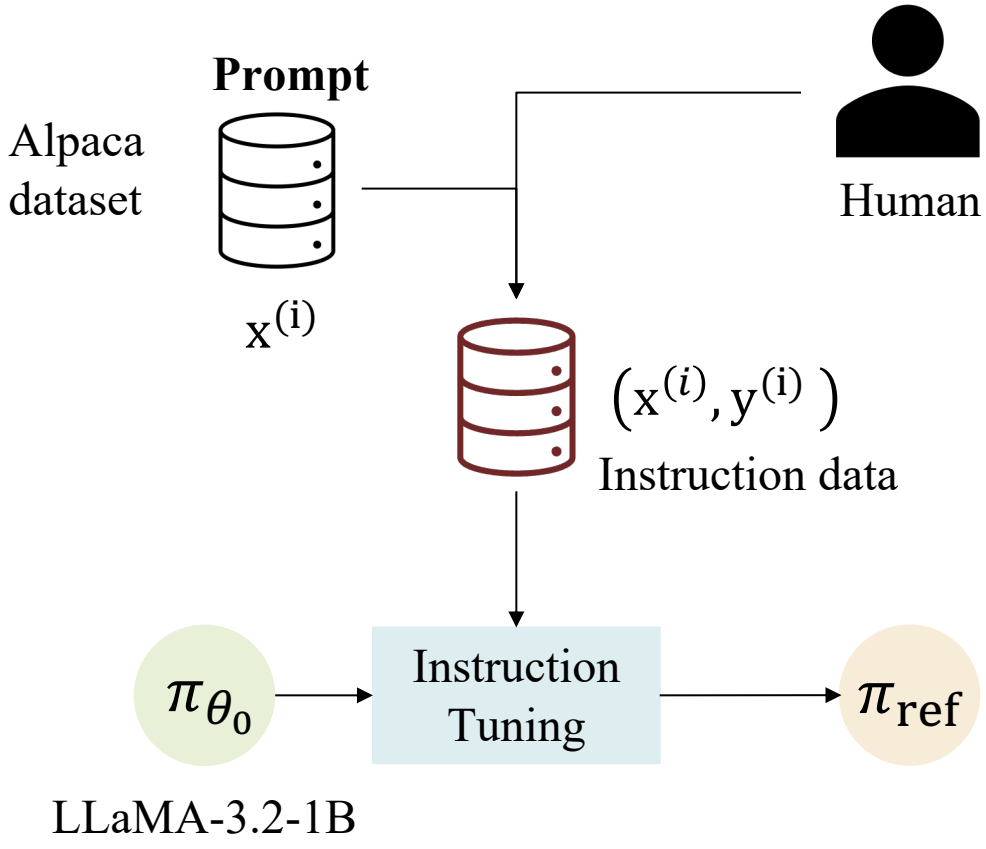
Pipeline



DPO for Vietnamese Dataset

Supervised Fine-Tuning

- Dataset



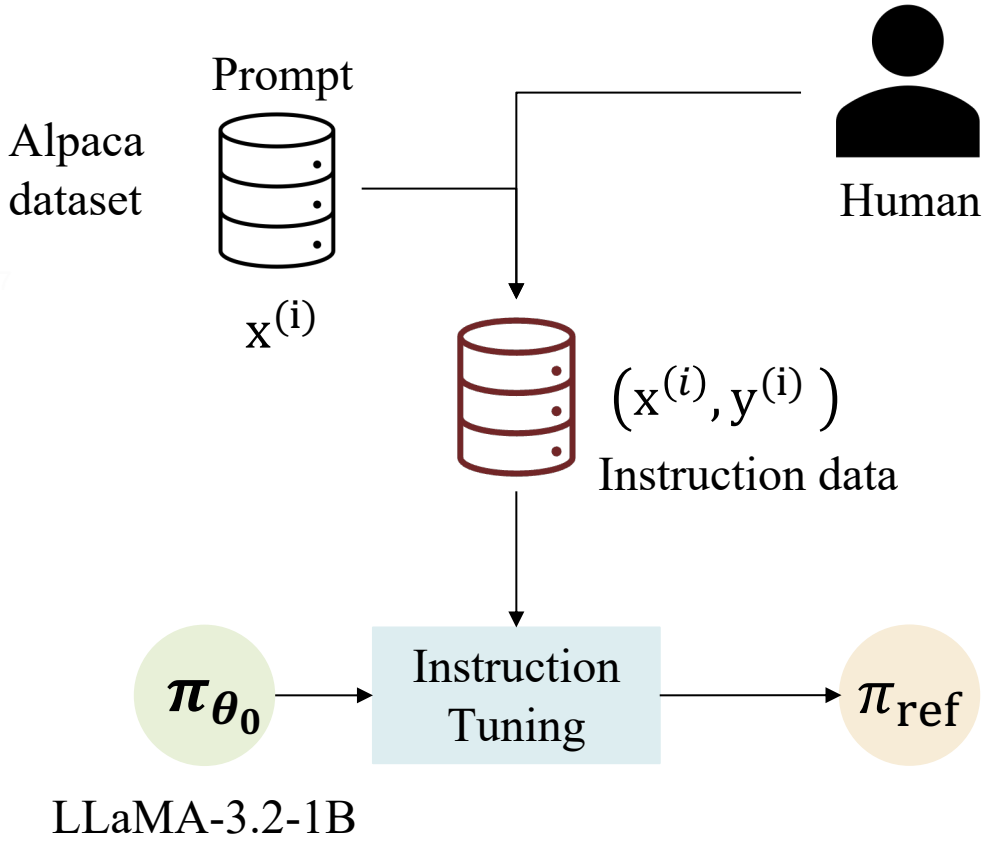

```
1  dataset = load_dataset("thainq107/Vi-Alpaca-Preference")
2  dataset, dataset["train"][0]
```

```
(DatasetDict({
    train: Dataset({
        features: ['id', 'question', 'chosen', 'rejected'],
        num_rows: 65017
    })
    test: Dataset({
        features: ['id', 'question', 'chosen', 'rejected'],
        num_rows: 2000
    })
}),
{'id': 'alpaca-7294',
 'question': 'Xác định và sửa lỗi ngữ pháp.\n\nTôi đã đi đến cửa hàng.',
 'chosen': 'Không có lỗi ngữ pháp. Câu này đã chính xác.',
 'rejected': 'Câu này không có lỗi ngữ pháp.'})
```

DPO for Vietnamese Dataset

Supervised Fine-Tuning

- Model

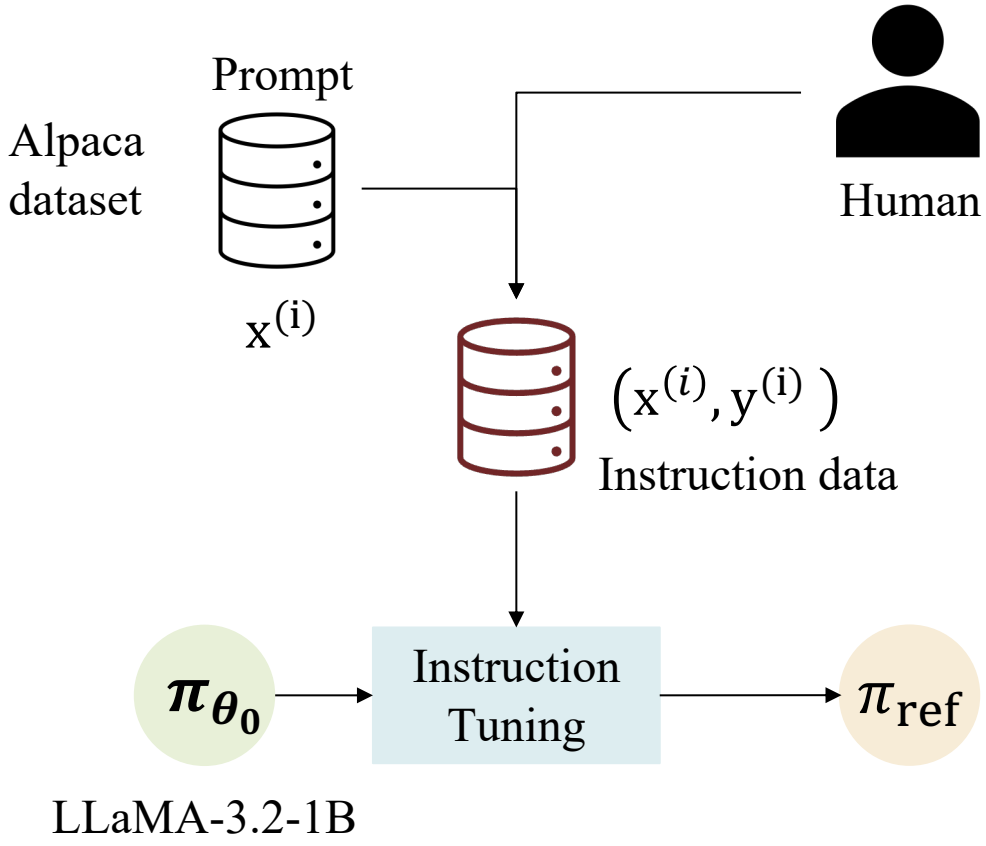



```python
# Model to fine-tune

model_name = f"meta-llama/Llama-3.2-1B-Instruct"
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(
    model_name,
    trust_remote_code=True,
    device_map={"" : torch.cuda.current_device()},
    token="###",
    cache_dir=cache_dir,
    torch_dtype=torch.bfloat16,
    quantization_config=bnb_config,
)
base_model.config.use_cache = False
```

DPO for Vietnamese Dataset

Supervised Fine-Tuning

- Model

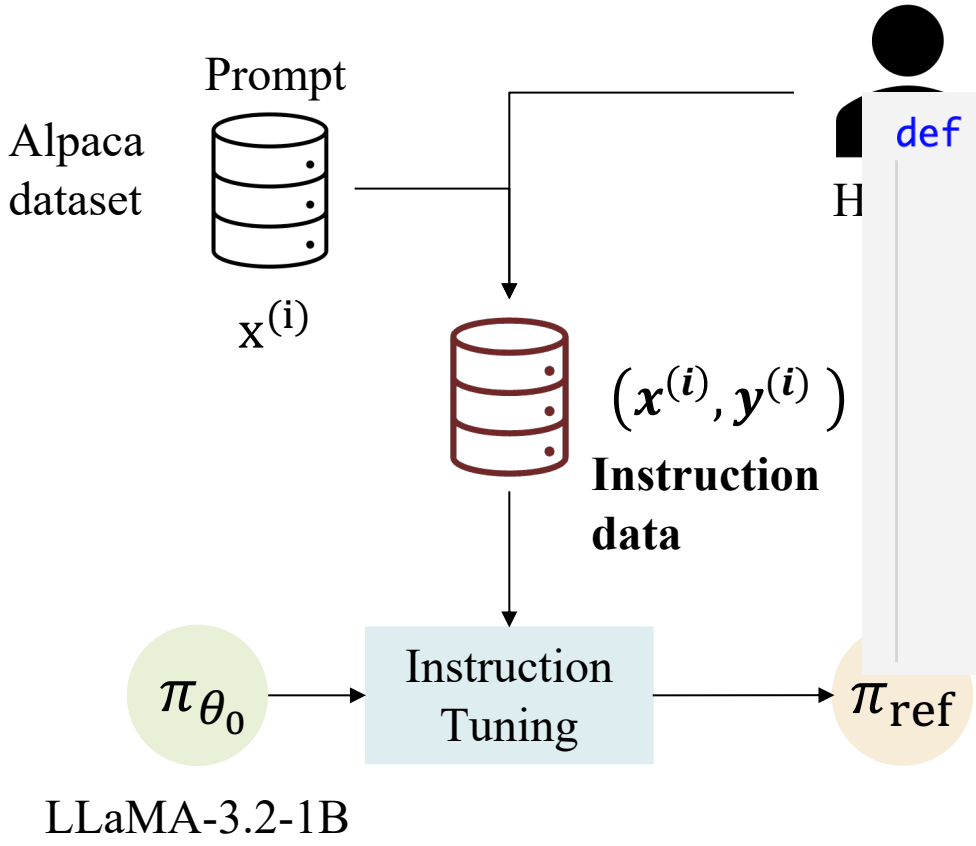



```python
# QLoRA configuration
peft_config = LoraConfig(
    r=16,
    lora_alpha=16,
    lora_dropout=0.05,
    bias="none",
    task_type="CAUSAL_LM",
    target_modules=[
        "q_proj",
        "k_proj",
        "v_proj",
        "o_proj",
        "gate_proj",
        "up_proj",
        "down_proj"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

DPO for Vietnamese Dataset

Supervised Fine-Tuning

- Model

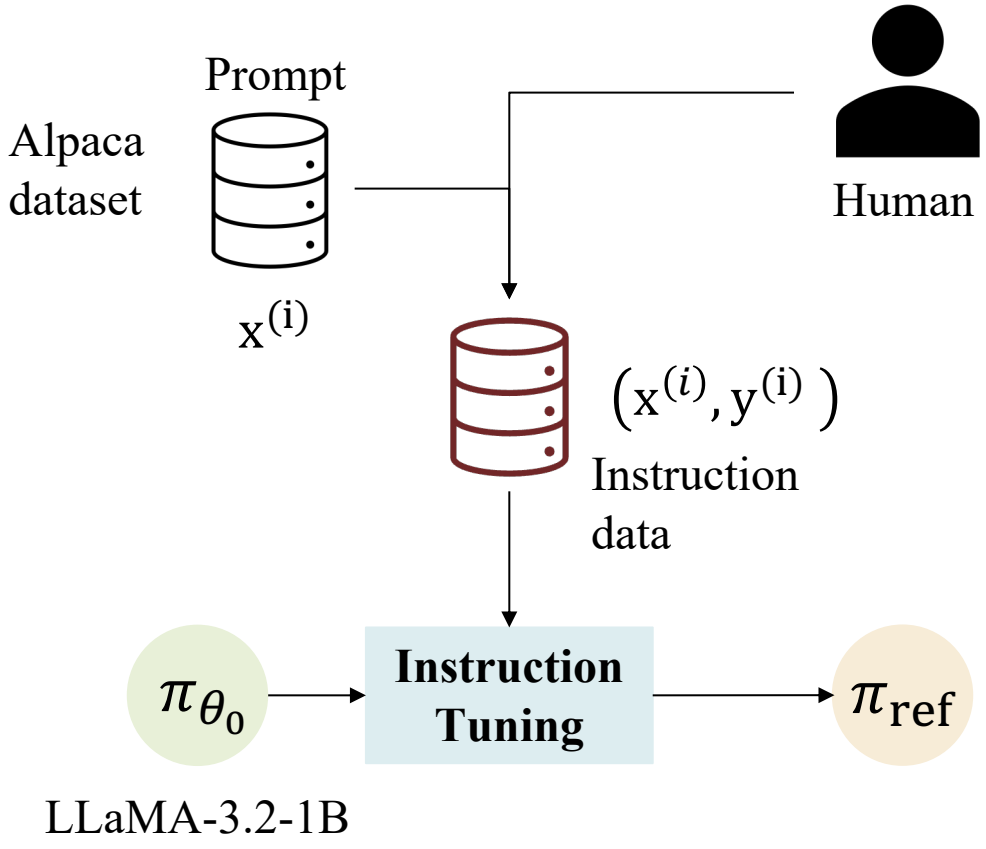



```python
def formatting_prompt_with_chat_template(example):
    conversation = [
        {"role": "system", "content": "You are a helpful assistant."},
        {"role": "user", "content": example["question"]},
        {"role": "assistant", "content": example["chosen"]},
    ]
    prompt = tokenizer.apply_chat_template(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

DPO for Vietnamese Dataset

Supervised Fine-Tuning

- Model

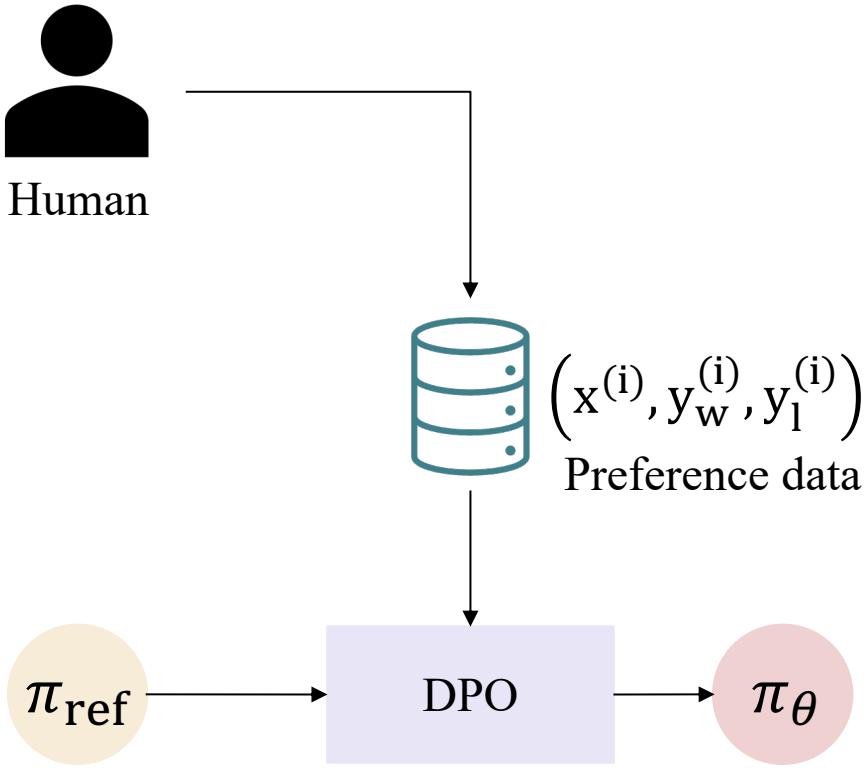



```
SFT_OUTPUT_DIR = f"Llama-3.2-1B-Instruct-sft"

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()
```

DPO for Vietnamese Dataset

Preference Model

- Dataset



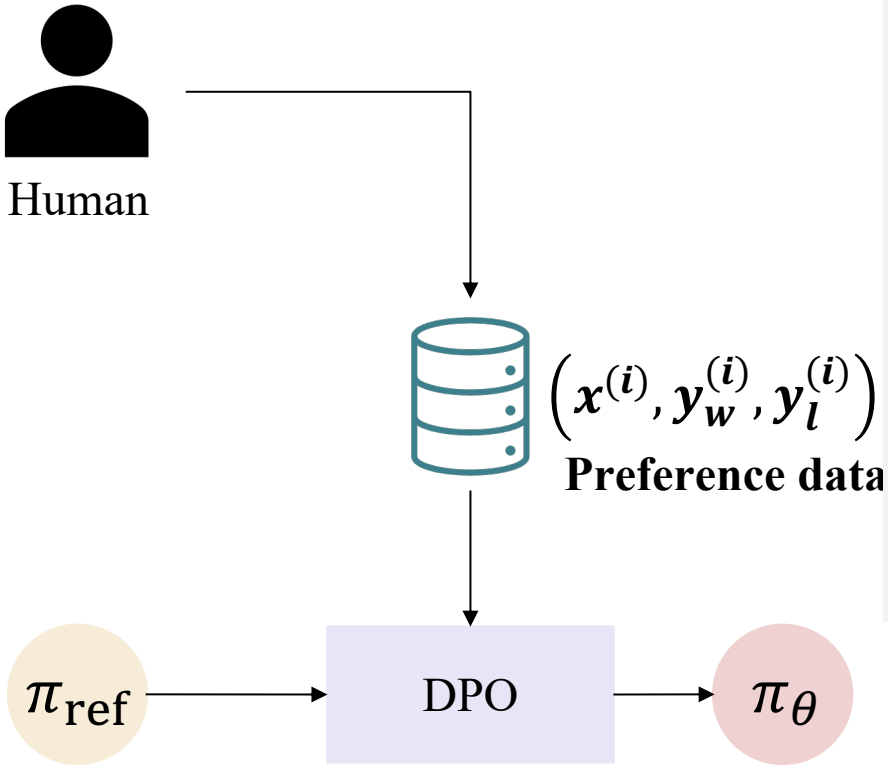

```
dataset = load_dataset("thainq107/Vi-Alpaca-Preference")
dataset, dataset["train"][0]
```

```
(DatasetDict({
    train: Dataset({
        features: ['id', 'question', 'chosen', 'rejected'],
        num_rows: 65017
    })
    test: Dataset({
        features: ['id', 'question', 'chosen', 'rejected'],
        num_rows: 2000
    })
}),
{'id': 'alpaca-7294',
 'question': 'Xác định và sửa lỗi ngữ pháp.\n\nTôi đã đi đến cửa hàng.',
 'chosen': 'Không có lỗi ngữ pháp. Câu này đã chính xác.',
 'rejected': 'Câu này không có lỗi ngữ pháp.'})
```

DPO for Vietnamese Dataset

Preference Model

- Dataset

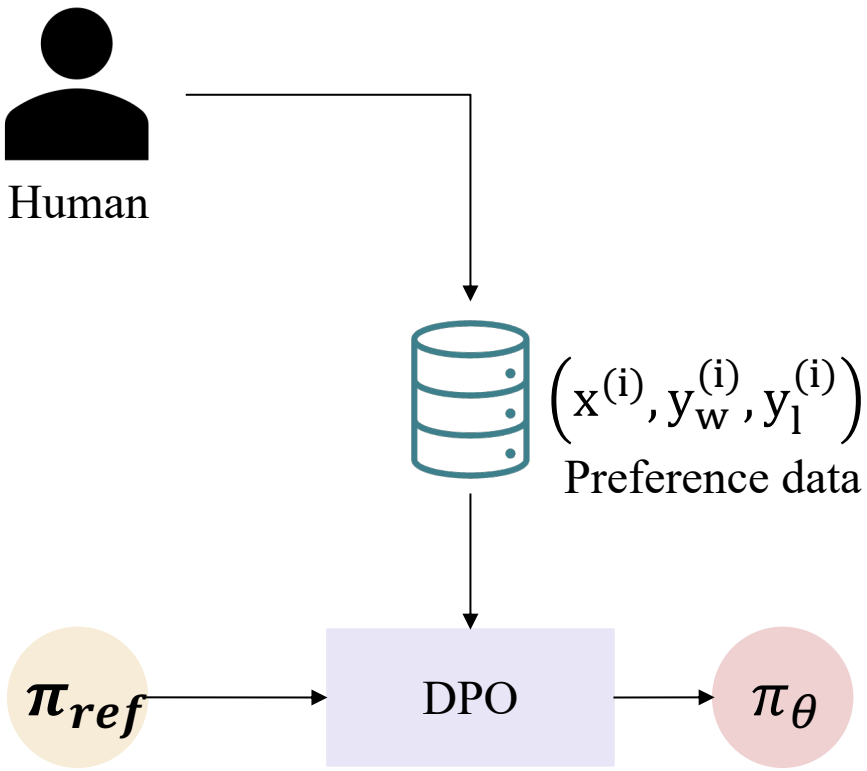



```python
def convert_to_conversational_preference_format(example):
    return {
        "id": example["id"],
        "prompt": [{"role": "system",
                    "content": "You are a helpful assistant."},
                   {"role": "user",
                    "content": example["question"]}],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

dpo_dataset = dataset.map(convert_to_conversational_preference_format)
```

DPO for Vietnamese Dataset

Preference Model

- Model

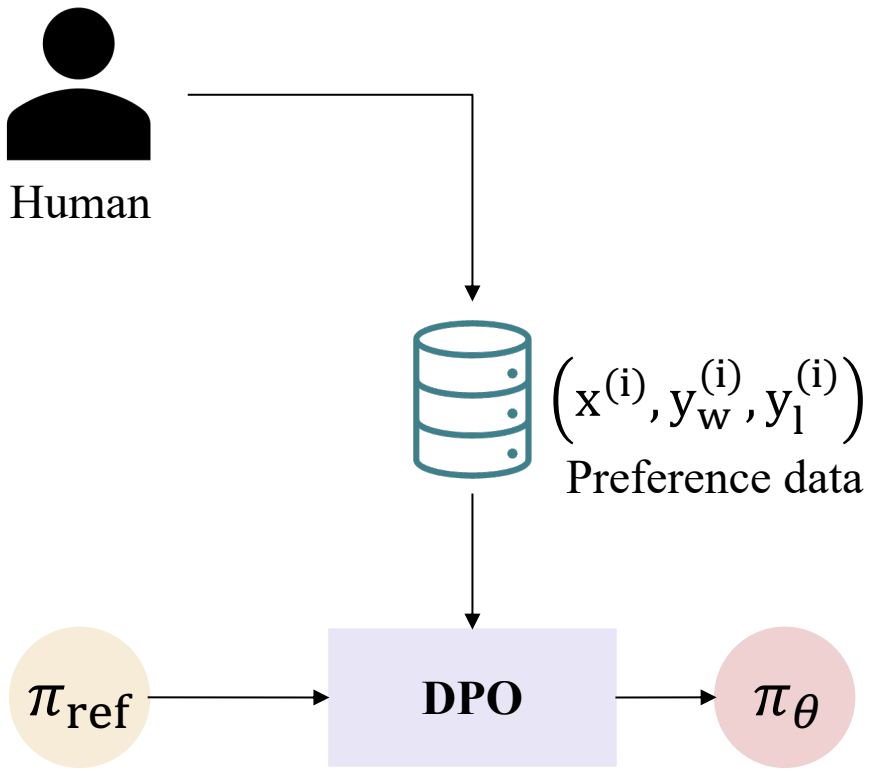



```python
model_name = "thainq107/Llama-3.2-1B-Instruct-sft"
cache_dir = "./cache"

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(
    model_name,
    trust_remote_code=True,
    device_map={"" : torch.cuda.current_device()},
    token="###",
    cache_dir=cache_dir,
    torch_dtype=torch.bfloat16,
    quantization_config=bnb_config,
)
base_model.config.use_cache = False
```

DPO for Vietnamese Dataset

Preference Model

- Model




```
DPO_OUTPUT_DIR = f"Llama-3.2-1B-Instruct-dpo"
dpo_args = DPOConfig(
    **{ **hyperparameters, "output_dir":
       DPO_OUTPUT_DIR, "max_length": MAX_LENGTH }
)

dpo_trainer = DPOTrainer(
    base_model,
    args=dpo_args,
    train_dataset=dpo_dataset['train'],
    processing_class=tokenizer,
    peft_config=peft_config,
)
dpo_trainer.train()
```

DPO for Vietnamese Dataset

Deployment

- Github - Demo

Vietnamese Custom Preference Model

Model: LLaMA-3.2-1B. Dataset: Alpaca-Vi

Input

Trí tuệ nhân tạo là gì?

Clear | Submit

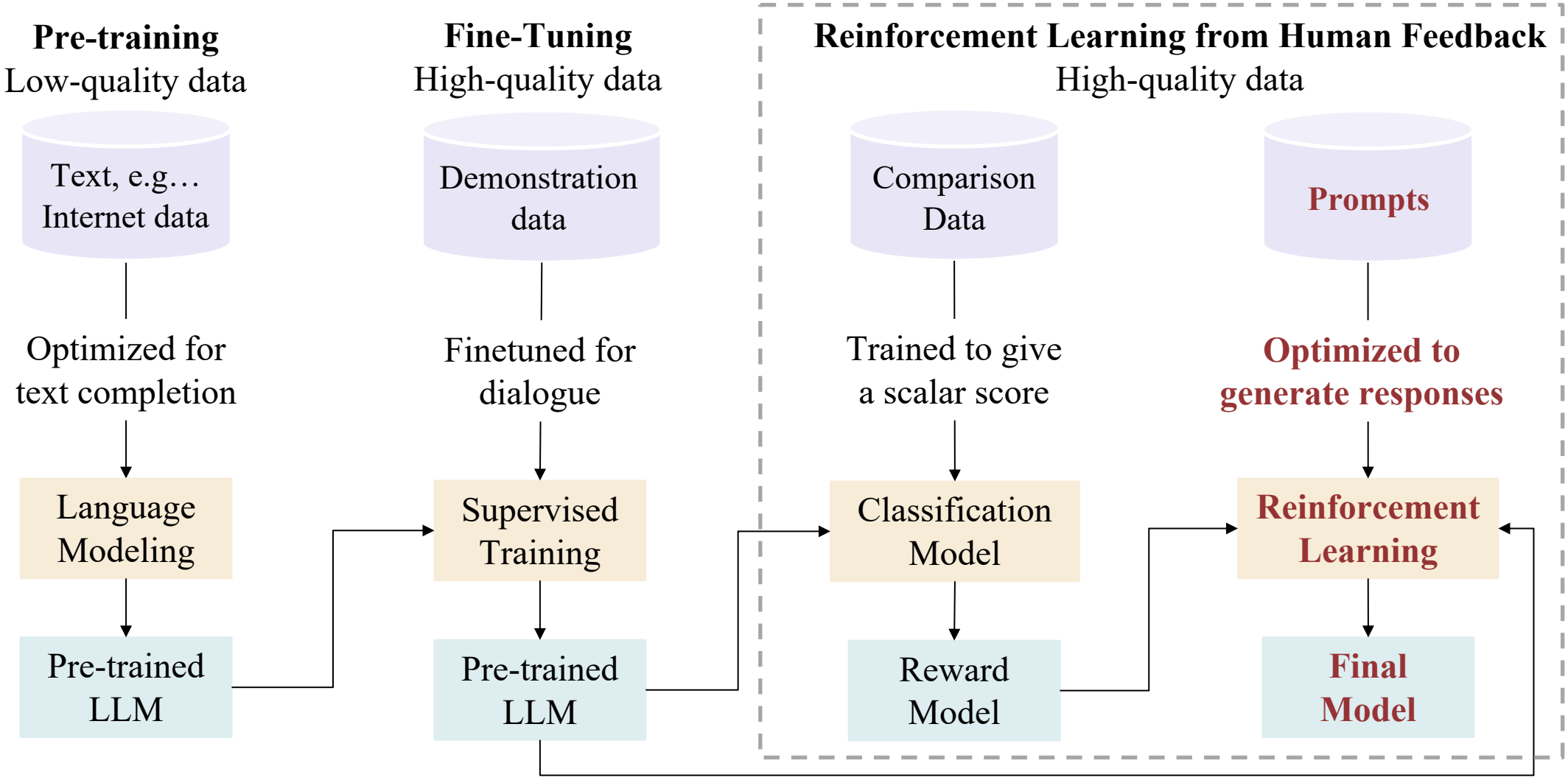
Output

Trí tuệ nhân tạo (Artificial Intelligence, AI) là một loại công nghệ dựa trên hệ thống học máy, trong đó con người sử dụng các thuật toán để tạo ra các mô hình và hệ thống có thể tự động thực hiện các nhiệm vụ, quyết định và hành động theo hướng logic và sáng tạo. Nó có thể được sử dụng trong nhiều lĩnh vực, bao gồm nhưng không giới hạn ở kinh tế, y tế, giáo dục, bảo vệ an ninh, và nhiều hơn nữa.

Share via Link

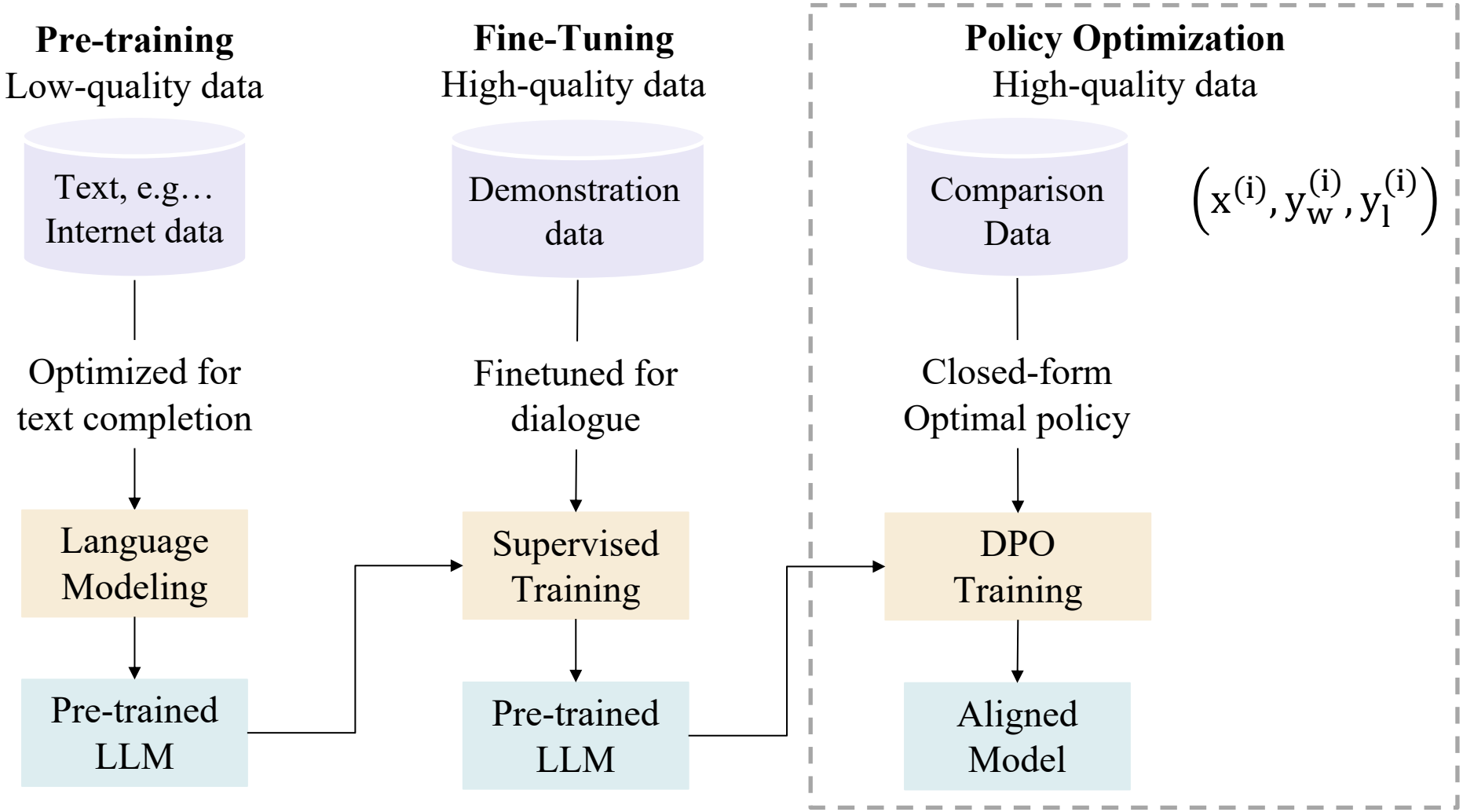
Preference Tuning

Summary: RLHF



Preference Tuning

Summary: DPO



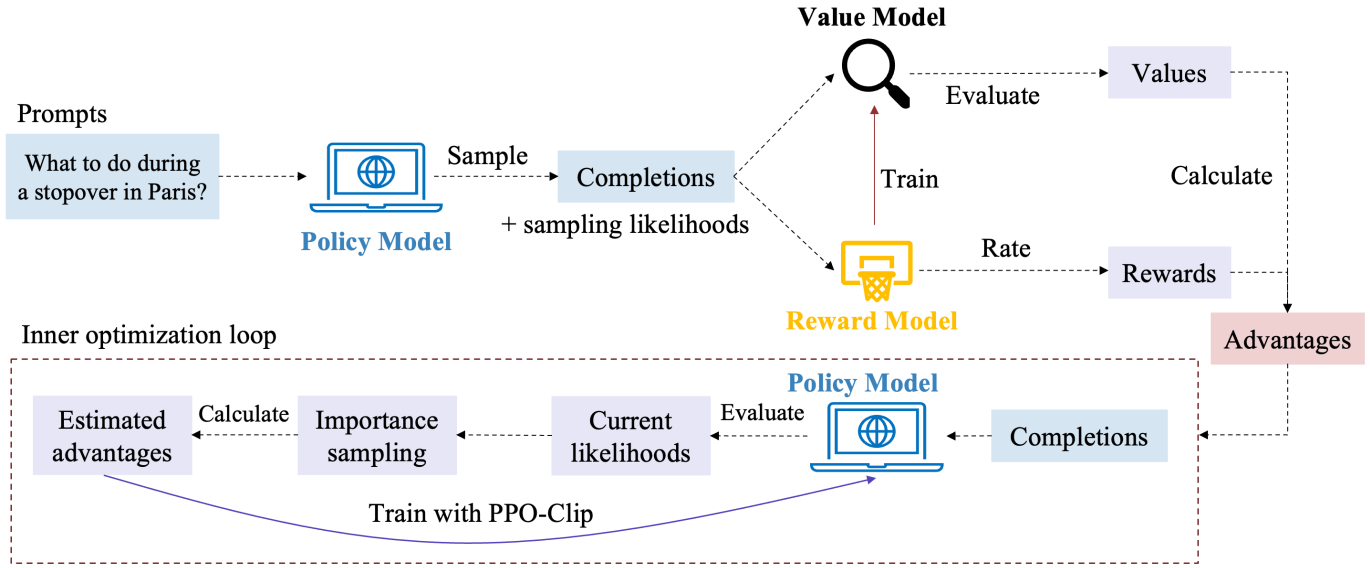
Objectives

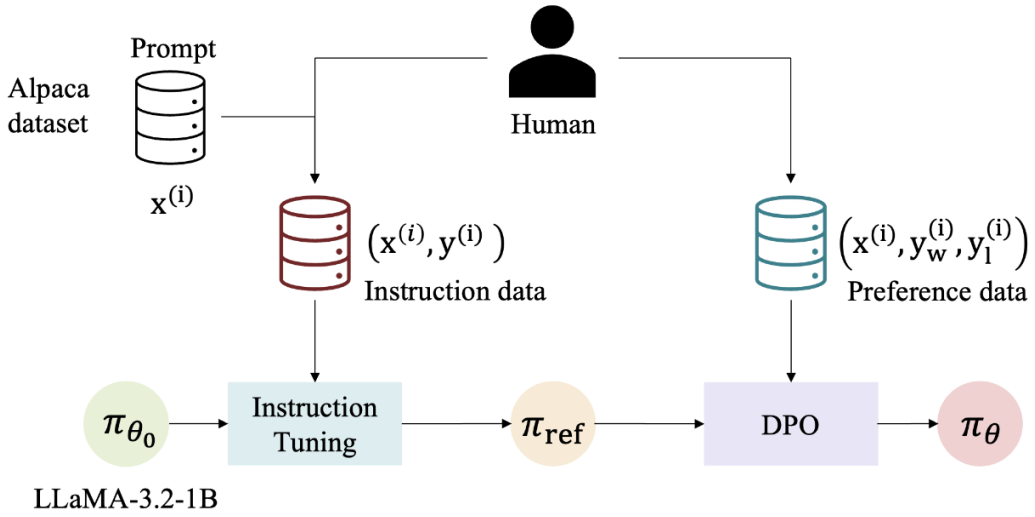
Preference Tuning

- Training LLMs
- Reward Modeling
- Reinforcement Learning in LLMs

Direct Preference Optimization

- Limit of RLHF
- Objective Function of DPO
- DPO for Vietnamese Custom Dataset





Thanks!

Any questions?