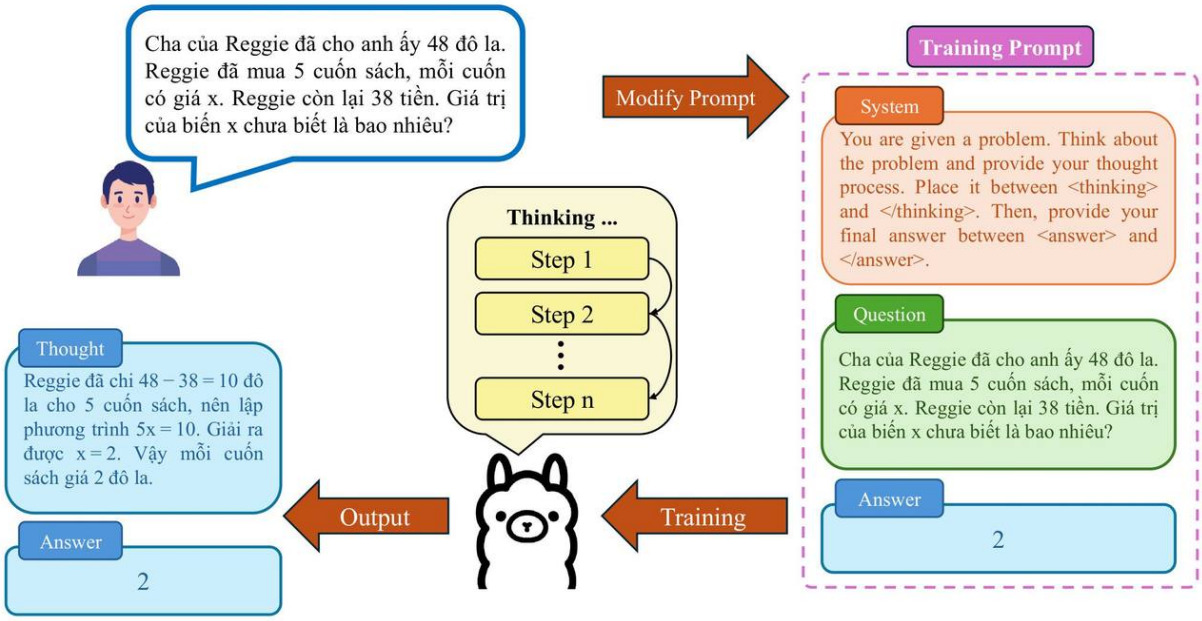
AI VIET NAM – AI COURSE 2024

Tutorial: Improving Reasoning Capability of LLMs with Reinforcement Learning

Dinh-Thang Duong, Truong-Binh Duong, Quang-Vinh Dinh

I. Giới thiệu

LLM Reasoning (Tạm dịch: LLM Suy Luận) là khả năng của các mô hình ngôn ngữ lớn trong việc mô phỏng quy trình tư duy logic và giải quyết vấn đề theo từng bước rõ ràng, từ việc trích xuất, liên kết các luận điểm đến diễn giải chi tiết trước khi đưa ra kết quả cuối cùng. Các mô hình như OpenAI o1 hay DeepSeek-R1 đã chứng tỏ thành công ấn tượng khi triển khai khả năng suy luận của mô hình lớn vào nhiều bài toán phức tạp, nâng cao cả độ chính xác lẫn khả năng giải thích của kết quả.



Hình 1: Huấn luyện mô hình ngôn ngữ lớn có khả năng suy luận với dữ liệu giải toán Tiếng Việt.

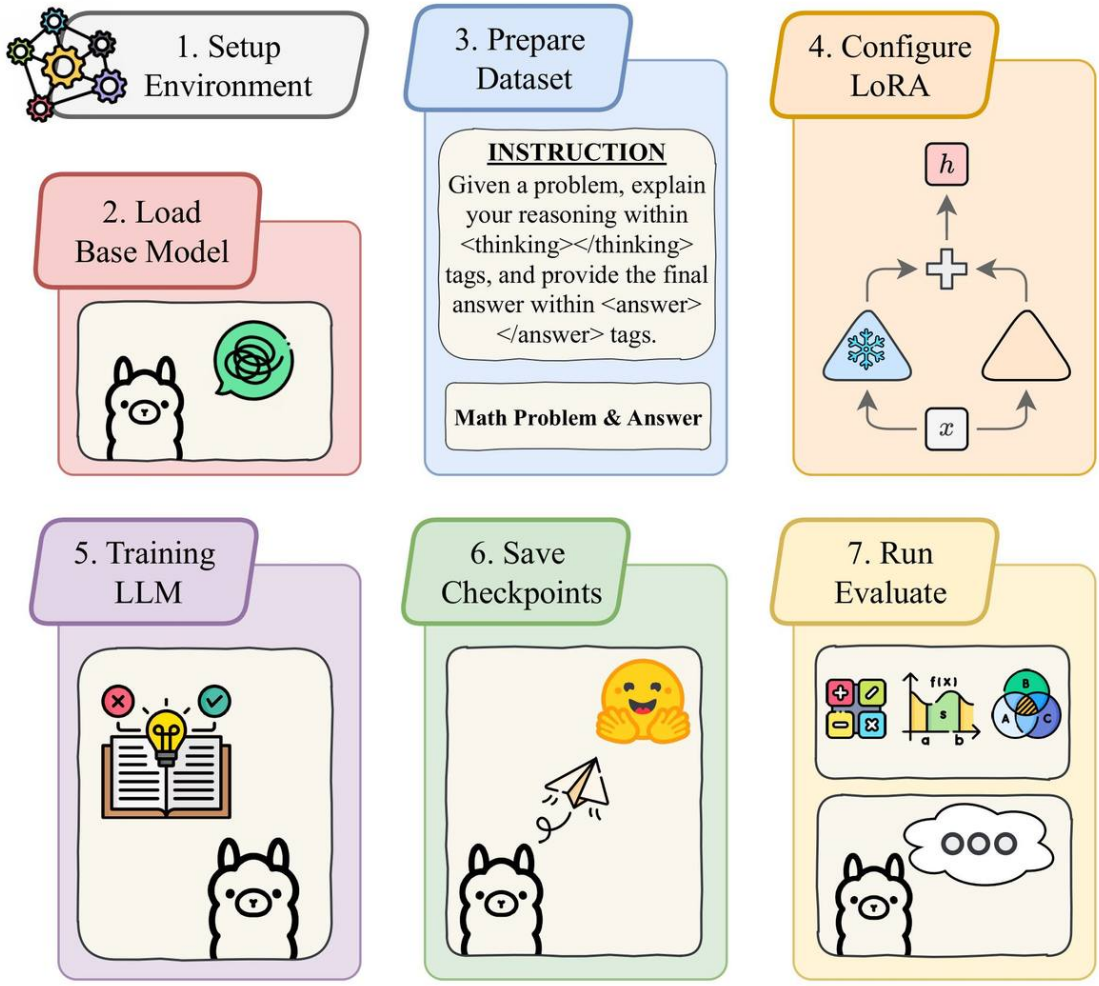
Thay vì chỉ sinh văn bản mạch lạc dựa trên dữ liệu đã học, LLM Reasoning đòi hỏi mô hình "suy nghĩ từng bước" để đưa ra lời giải chính xác và giải thích chi tiết quá trình tư duy, đồng thời cải

thiện tính minh bạch và độ tin cậy trên các nhiệm vụ như giải toán trắc nghiệm, lập luận khoa học hay phân tích ngữ nghĩa.

Trong bài viết này, chúng ta sẽ tìm hiểu cách cài đặt quá trình huấn luyện nhằm tăng cường khả năng suy luận cho một mô hình ngôn ngữ lớn thông qua dữ liệu các câu hỏi toán học. Mục tiêu là xây dựng một mô hình ngôn ngữ lớn chuyên cho việc giải toán trắc nghiệm tiếng Việt, không những có thể đưa ra đáp án đúng mà còn trình bày được chuỗi suy nghĩ logic dẫn đến đáp án đó (dẫn đến việc cải thiện hiệu suất mô hình). Bài toán được định nghĩa với đầu vào và đầu ra như sau:

- **Input:** Nội dung cụ thể của bài toán cần giải quyết.
- **Output:** Lời phản hồi từ mô hình, bao gồm quá trình suy nghĩ và đáp án cho câu hỏi.

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



Hình 2: Huấn luyện khả năng suy luận cho mô hình ngôn ngữ lớn bằng GRPO.

Trong đó:

1. **Setup Environment:** Thiết lập môi trường làm việc, bao gồm cài đặt và import các thư viện phù hợp cho huấn luyện mô hình.
2. **Load Base Model:** Tải mô hình ngôn ngữ lớn (LLM) gốc từ thư viện như Hugging Face làm nền tảng để tinh chỉnh.
3. **Configure LoRA:** Cấu hình LoRA để giảm tài nguyên huấn luyện bằng cách khóa các tầng gốc và chỉ tinh chỉnh các adapter nhẹ.
4. **Prepare Dataset:** Chuẩn bị và tiền xử lí dữ liệu training bao gồm câu hỏi và đáp án đúng theo định dạng đầu vào/đầu ra rõ ràng.
5. **Fine-tune LLM:** Huấn luyện mô hình trên tập dữ liệu toán học đã chuẩn bị bằng GRPO và kỹ thuật LoRA nhằm tối ưu hiệu suất.
6. **Save Checkpoints:** Lưu lại các checkpoint của mô hình trong quá trình huấn luyện để theo dõi tiến độ và phục hồi nếu cần.
7. **Run Evaluate:** Thực hiện suy luận trên các câu hỏi để đánh giá khả năng đọc hiểu của mô hình sau tinh chỉnh.

Thông qua pipeline trên, ta sẽ có một mô hình ngôn ngữ lớn đã được tinh chỉnh với chi phí thấp, có khả năng suy luận và trả lời hiệu quả các câu hỏi toán học.

II. Cài đặt chương trình

II.1. Cài đặt và import các thư viện cần thiết

Đầu tiên, chúng ta cần cài đặt và nạp các thư viện hỗ trợ quá trình huấn luyện với GRPO và LoRA trên nền Llama 3.2 1B. Trong đó:

- `unsloth` cung cấp lớp `FastLanguageModel` tối ưu cho inference và fine-tuning.
- `vllm` hỗ trợ việc tạo sinh output nhanh chóng với cấu hình linh hoạt.
- `datasets` dùng để tải và xử lý tập dữ liệu MetaMathQA.
- `trl` bao gồm `GRPOConfig` và `GRPOTrainer` cho module học tăng cường.

Chúng ta thực hiện cài đặt `unsloth` và `vllm` thông qua lệnh `pip install` dưới đây:

```
!pip install unsloth vllm==0.7.3
```

Sau khi cài đặt xong, chúng ta import các thư viện cần thiết để thiết lập mô hình, xử lý dữ liệu và cấu hình GRPO:

```
import os
import re

from vllm import SamplingParams
from unsloth import FastLanguageModel
from datasets import load_dataset, Dataset
from trl import GRPOConfig, GRPOTrainer
```

II.2. Load mô hình lớn và setup cài đặt LoRA

Tiếp theo, chúng ta khởi tạo mô hình Llama 3.2 1B từ Unsloth và kích hoạt LoRA để giảm số tham số cần huấn luyện. Lưu ý rằng, các bạn hoàn toàn có thể thay đổi sang các phiên bản mô hình Llama 3.2 với số lượng tham số lớn hơn để có thể đạt một mô hình với hiệu suất cao hơn nếu phần cứng cho phép.

```
max_seq_length = 2048
lora_rank = 64

model, tokenizer = FastLanguageModel.from_pretrained(
    model_name="meta-llama/Llama-3.2-1B-Instruct",
    max_seq_length=max_seq_length,
    load_in_4bit=False,
    fast_inference=True,
    max_lora_rank=lora_rank,
    gpu_memory_utilization=0.8,
)

model = FastLanguageModel.get_peft_model(
    model,
```

```
    r=lora_rank,
    target_modules=[
        "q_proj", "k_proj", "v_proj", "o_proj",
        "gate_proj", "up_proj", "down_proj",
    ],
    lora_alpha=lora_rank,
    use_gradient_checkpointing="unsloth",
    random_state=3407,
)
```

- `max_seq_length` định nghĩa độ dài tối đa của input sequence.
- `lora_rank` xác định kích thước của ma trận adapter LoRA.
- `load_in_4bit=False` tắt lượng tử hóa để giữ độ chính xác cho reasoning.
- `fast_inference=True` tối ưu hóa throughput khi sinh text.
- `gpu_memory_utilization=0.8` cho phép sử dụng tối đa 80% bộ nhớ GPU.

II.3. Tải bộ dữ liệu

Trong bài này, ta sẽ tăng cường khả năng suy luận của mô hình LLM trong việc giải toán trắc nghiệm tiếng Việt. Theo đó, bộ dữ liệu MetaMathQA-40K phiên bản tiếng Việt sẽ được tải về thông qua đoạn code dưới đây:

```
dataset = load_dataset("5CD-AI/Vietnamese-meta-math-MetaMathQA-40K-gg-translated", split="
                                                    train")
```

Bài toán	Đáp án
Cha của Reggie đã cho anh ấy \$48. Reggie mua 5 cuốn sách, mỗi cuốn giá \$x và còn lại \$38. Hỏi giá trị của \$x là bao nhiêu?	2
Jean và ba người bạn chơi domino với bộ 28 quân, chia đều cho 4 người. Hỏi mỗi người nhận bao nhiêu quân?	7
Cally có 10 áo trắng, 5 áo màu, 7 quần đùi và 6 quần dài (tổng 28 bộ). Danny có 6 áo trắng, 8 áo màu, 10 quần đùi và 6 quần dài (tổng 30 bộ). Hỏi họ đã giặt bao nhiêu bộ quần áo?	58

Bảng 1: Một vài mẫu câu hỏi và đáp án trực tiếp (phần tiếng Việt) trong bộ MetaMathQA-40K.

II.4. Cài đặt format prompt cho reasoning

Trước khi huấn luyện, ta cần chuẩn hóa dữ liệu để mô hình học được chuỗi suy nghĩ và đáp án tách biệt. Các bước bao gồm:

1. Dùng regex `answer_pattern` để trích xuất đúng phần "đáp án là...".

2. Đánh dấu bắt đầu/kết thúc chuỗi suy nghĩ bằng `<thinking>...</thinking>` và phần đáp án bằng `<answer>...</answer>`.
3. Xây dựng `system_prompt` hướng dẫn model sinh reasoning rồi answer.
4. Chuyển danh sách thành `train_dataset` với hai trường `prompt` và `answer`.

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

Cutting Knowledge Date: December 2023
Today Date: 28 Apr 2025

You are given a problem.
Think about the problem and provide your thought process.
Place it between **<thinking>** and **</thinking>**.
Then, provide your final answer between **<SOLUTION></SOLUTION>**<|**eot_id**|>
<|**start_header_id**|>user<|**end_header_id**|>

Cha của Reggie đã cho anh ấy 48 đô la. Reggie đã mua 5 cuốn sách, mỗi cuốn có giá x. Reggie còn lại 38 tiền. Giá trị của biến x chưa biết là bao nhiêu?<|**eot_id**|>
<|**start_header_id**|>assistant<|**end_header_id**|>

Hình 3: Nội dung đầy đủ chi tiết của một mẫu prompt khi đưa vào mô hình Llama với một số thông tin về việc reasoning cũng như thông tin về bài toán.

```python
answer_pattern = re.compile(
    r"(đáp án là:|đáp án là :|câu trả lời là:|câu trả lời là :)\s*(.*)",
    re.IGNORECASE
)

formatted_dataset = []
for item in dataset:
    response = item["response_vi"].strip().lower()
    match = answer_pattern.search(response)
    if match:
        answer = match.group(2).strip()
        formatted_dataset.append({
            "question": item["query_vi"],
            "answer": answer
        })

reasoning_start = "<thinking>"
reasoning_end = "</thinking>"
solution_start = "<SOLUTION>"
```

```
solution_end = "</SOLUTION>"

system_prompt = \
    f"""You are given a problem.
Think about the problem and provide your thought process.
Place it between {reasoning_start} and {reasoning_end}.
Then, provide your final answer between {solution_start}{solution_end}"""

train_dataset = Dataset.from_list(formatted_dataset[:8000])
train_dataset = train_dataset.map(lambda x: {
    "prompt": [
        {"role": "system", "content": system_prompt},
        {"role": "user",   "content": x["question"]},
    ],
    "answer": x["answer"],
})
```

II.5. Định nghĩa các hàm reward cho học tăng cường

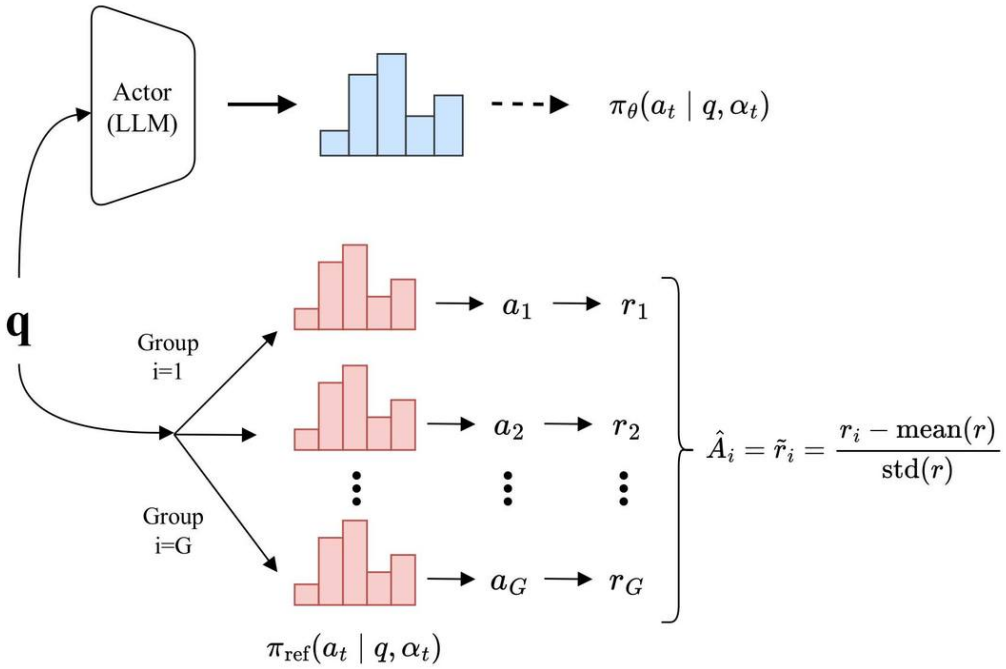
Đối với các thuật toán học tăng cường, việc đánh giá hiệu suất mô hình thường được thông qua các hàm điểm thưởng (reward function). Trong trường hợp này, các hàm reward được cân nhắc triển khai đánh giá chất lượng output theo hai tiêu chí: đúng format reasoning và đúng đáp án. Các bạn hoàn toàn có thể tìm hiểu để đưa ra thêm các tiêu chí điểm thưởng phù hợp.

II.5.1. Reward cho output đúng format toán

Đầu tiên, ta tạo ra hai hàm có chức năng tính điểm cho mô hình nếu nó cho ra một format output phù hợp. Ở đây, ta có hàm `match_format_exactly()`, nếu hoàn toàn khớp pattern reasoning + answer, cộng 3.0 điểm. Bên cạnh đó, hàm `match_format_approximately()` sẽ kiểm tra số lượng tag xuất hiện, cộng 0.5 mỗi tag đúng, trừ 1.0 nếu không đúng.

```
match_format = re.compile(
    rf"^[\s]{{0,}}"
    rf"{reasoning_start}.+?{reasoning_end}.*?"
    rf"{solution_start}(.+?){solution_end}"
    rf"[\s]{{0,}}\$",
    flags=re.MULTILINE | re.DOTALL
)

def match_format_exactly(completions, **kwargs):
    scores = []
    for completion in completions:
        score = 0
        response = completion[0]["content"]
        if match_format.search(response) is not None:
            score += 3.0
        scores.append(score)
    return scores

def match_format_approximately(completions, **kwargs):
    scores = []
    for completion in completions:
```

```
        score = 0
        response = completion[0]["content"]
        score += 0.5 if response.count(reasoning_start) == 1 else -1.0
        score += 0.5 if response.count(reasoning_end) == 1 else -1.0
        score += 0.5 if response.count(solution_start) == 1 else -1.0
        score += 0.5 if response.count(solution_end) == 1 else -1.0
        scores.append(score)
    return scores
```



Hình 4: Minh họa quy trình GRPO, trong đó LLM đóng vai trò actor sinh phân phối xác suất cho hành động, sau đó lấy mẫu thành nhiều nhóm, đánh giá reward cho từng nhóm và chuẩn hóa kết quả để cập nhật chính sách.

II.5.2. Reward cho output đúng đáp án

Tiếp theo, ta định nghĩa hàm `check_answer()` để so khớp chuỗi giữa tag `<answer>`, cộng 3.0 nếu đúng, 1.5 nếu chỉ khác khoảng trắng, trừ 1.5 nếu sai hoàn toàn. Cuối cùng, ta có hàm `check_numbers()` với chức năng trích xuất số, so sánh giá trị float, cộng 1.5 nếu đúng, trừ 0.5 nếu sai.

```
match_numbers = re.compile(
    solution_start + r".*?([\d\.\,]{1,})",
    flags=re.MULTILINE | re.DOTALL
)

def check_answer(prompts, completions, answer, **kwargs):
    responses = [completion[0]["content"] for completion in completions]

```

```
    extracted_responses = [
        guess.group(1)
        if (guess := match_format.search(r)) is not None else None
        for r in responses
    ]

    scores = []
    for guess, true_answer in zip(extracted_responses, answer):
        score = 0
        if guess is None:
            scores.append(0)
            continue
        if guess == true_answer:
            score += 3.0
        elif guess.strip() == true_answer.strip():
            score += 1.5
        else:
            score -= 1.5
        scores.append(score)
    return scores

def check_numbers(prompts, completions, answer, **kwargs):
    question = prompts[0][-1]["content"]
    responses = [completion[0]["content"] for completion in completions]

    extracted_responses = [
        guess.group(1)
        if (guess := match_numbers.search(r)) is not None else None
        for r in responses
    ]

    count = getattr(check_numbers, 'counter', 0) + 1
    check_numbers.counter = count
    if count % 5 == 0:
        print('*'*20, f"Question:{question}", f"\nResponse:\n{responses[0]}",
              f"\nExtracted: {extracted_responses[0]}", f"\nGT Answer: {answer[0]}")

    scores = []
    for guess, true_answer in zip(extracted_responses, answer):
        if guess is None:
            scores.append(0)
            continue
        try:
            true_answer = float(true_answer.strip())
            # Remove commas like in 123,456
            guess = float(guess.strip().replace(",", ""))
            scores.append(1.5 if guess == true_answer else -0.5)
        except:
            scores.append(0)
    return scores
```

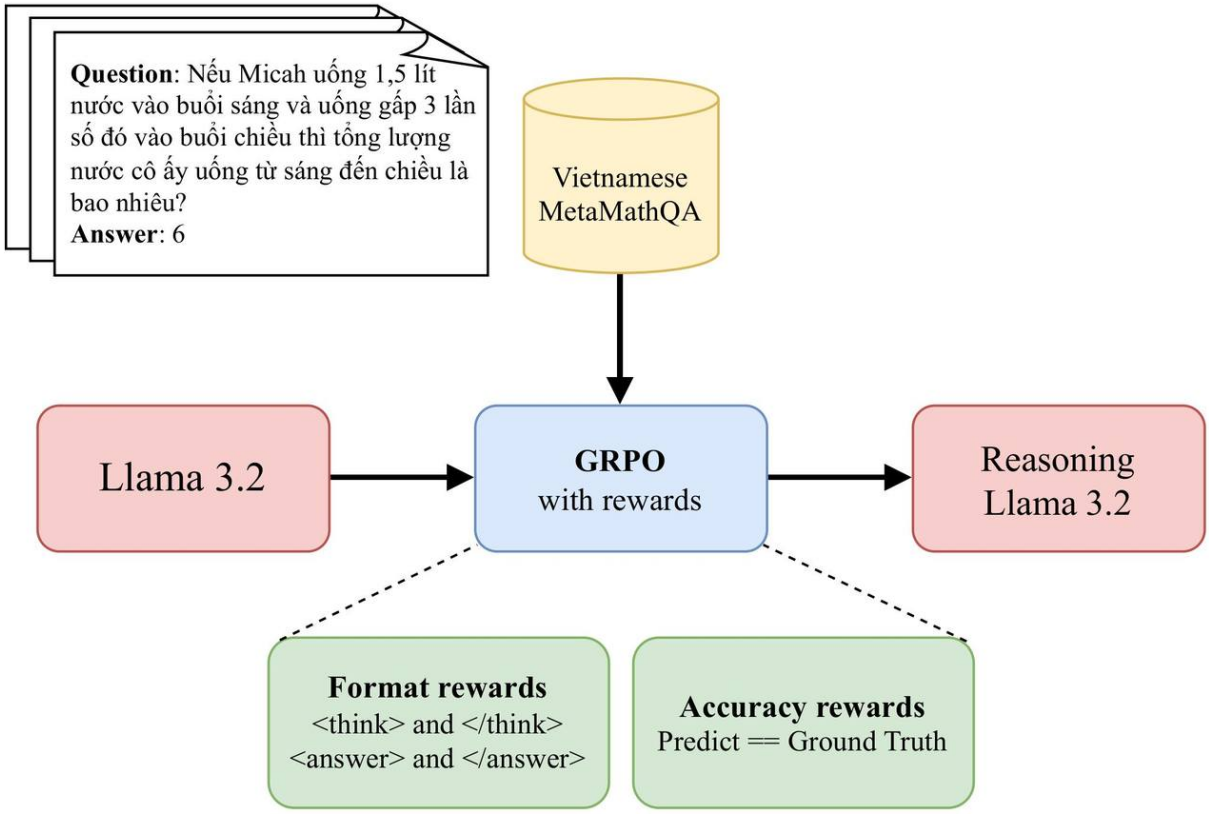
II.6. Huấn luyện mô hình

Với tất cả các thông tin trên mà chúng ta đã khai báo, để tiến hành huấn luyện, ta chỉ cần cấu hình và chạy GRPOTrainer để thực hiện training học tăng cường với các hàm reward đã xây dựng:

```
max_len = max(dataset.map(
    lambda x: {"tokens": tokenizer.apply_chat_template(
        x["prompt"], add_generation_prompt=True, tokenize=True)},
    batched=True,
).map(lambda x: {"length": len(x["tokens"])})["length"])

max_prompt_length = max_len + 1

training_args = GRPOConfig(
    learning_rate=5e-6,
    weight_decay=5e-4,
    warmup_ratio=0.1,
    lr_scheduler_type="cosine",
    optim="adamw_torch_fused",
    logging_steps=1,
    per_device_train_batch_size=2,
    gradient_accumulation_steps=64,
    num_generations=8,
    max_prompt_length=max_prompt_length,
    max_completion_length=max_seq_length - max_prompt_length,
    num_train_epochs=1,
    max_steps=-1,
    save_steps=250,
    max_grad_norm=0.1,
    report_to="wandb",
    output_dir="outputs_bz2",
)

trainer = GRPOTrainer(
    model=model,
    processing_class=tokenizer,
    reward_funcs=[
        match_format_exactly,
        match_format_approximately,
        check_answer,
        check_numbers,
    ],
    args=training_args,
    train_dataset=dataset,
)
trainer.train()
```

- Tính `max_prompt_length()` dựa trên độ dài token tối đa của prompt đã format.
- Cấu hình `GRPOConfig` với learning rate, weight decay, warmup ratio, scheduler cosine, batch size, số lượng generations, gradient accumulation, và các bước lưu checkpoint.
- Khởi tạo `GRPOTrainer` với danh sách các hàm reward và dataset phù hợp.

- Gọi `trainer.train()` để bắt đầu huấn luyện.



Hình 5: Sử dụng thuật toán học tăng cường GRPO nhằm khuyến khích việc đưa ra các suy luận trước khi tạo sinh phương án cuối cùng của mô hình Llama.

II.7. Lưu trữ và inference

Sau khi huấn luyện xong, ta lưu adapter LoRA đã học và tiến hành thử suy luận trên một ví dụ:

```
model.save_lora("grpo_saved_lora")
```

```
idx = 0
messages = [
    {"role": "system", "content": system_prompt},
    {"role": "user",   "content": train_dataset[idx]["question"]},
]
sampling_params = SamplingParams(
    temperature = 0.8,
    top_p = 0.95,
    max_tokens = 1024,
)

text = tokenizer.apply_chat_template(
    messages,
```

```
    add_generation_prompt = True,
    tokenize = False,
)

path_lora = "grpo_saved_lora"
output = model.fast_generate(
    [text],
    sampling_params = sampling_params,
    lora_request = model.load_lora(path_lora),
)[0].outputs[0].text

print(f"Problem:\n{train_dataset[idx]['question']}")
print(f"Response:\n{output}")
print("GT Answer:", train_dataset[idx]["answer"])
```

<thinking>
Để tìm giá trị của x, ta cần hiểu rằng Reggie đã chi một khoản tiền 48 - 38 = 10 đô la để mua các sách. Vì Reggie đã mua 5 cuốn sách với giá x mỗi cuốn nên ta có thể tạo ra một công thức để tính tổng chi phí mua sách: 5x = 10.

Để giải quyết cho x, ta chia cả hai bên của công thức bằng 5: x = 10 / 5.

Solvers lại ta có: x = 2.
Vậy giá của mỗi cuốn sách là 2 đô la.
</thinking>
<SOLUTION>2**</SOLUTION>**

Hình 6: Minh họa một kết quả tạo sinh của mô hình với phần reasoning và đáp án cuối cùng.

Ví dụ minh họa ở Hình 6 cho thấy Llama 3.2 1B sau khi được tinh chỉnh với GRPO và LoRA đã có khả năng sinh chuỗi suy nghĩ chi tiết bằng thẻ `<thinking>` trước khi đưa ra đáp án cuối cùng trong thẻ `<SOLUTION>`. Điều này khẳng định tính hiệu quả của việc kết hợp học tăng cường và kỹ thuật LoRA, không chỉ nâng cao độ chính xác mà còn cải thiện tính giải thích được (explainability) cho các nhiệm vụ yêu cầu tư duy logic chặt chẽ. Theo đó, các bạn hoàn toàn có thể mở rộng phương pháp này cho các bài toán phức tạp hơn hoặc kết hợp thêm các tín hiệu đánh giá đa dạng để tiếp tục tối ưu khả năng suy luận của LLMs.

III. Câu hỏi trắc nghiệm

1. Theo bài viết, "LLM Reasoning" là gì?
   (*a*) Khả năng mô hình ngôn ngữ lớn sinh văn bản một cách mạch lạc.
   (*b*) Khả năng mô hình ngôn ngữ lớn mô phỏng quy trình tư duy logic và giải quyết vấn đề theo từng bước.
   (*c*) Khả năng mô hình ngôn ngữ lớn dịch thuật giữa các ngôn ngữ.
   (*d*) Khả năng mô hình ngôn ngữ lớn tóm tắt văn bản dài.

2. Mô hình ngôn ngữ lớn (LLM) gốc nào được sử dụng làm nền tảng trong bài viết?
   (*a*) OpenAI o1.
   (*b*) DeepSeek-R1.
   (*c*) Llama-3.2-1B-Instruct.
   (*d*) GPT-4.

3. Kỹ thuật nào được sử dụng để giảm số lượng tham số cần huấn luyện, bằng cách khóa các tầng gốc và chỉ tinh chỉnh adapter?
   (*a*) Quantization (Lượng tử hóa).
   (*b*) Gradient Checkpointing.
   (*c*) LoRA (Low-Rank Adaptation).
   (*d*) GRPO (Group Relative Policy Optimization).

4. Tập dữ liệu nào được sử dụng để huấn luyện mô hình giải toán tiếng Việt?
   (*a*) GSM8K.
   (*b*) MATH.
   (*c*) Vietnamese-meta-math-MetaMathQA-40K-gg-translated.
   (*d*) Alpaca.

5. Cặp thẻ nào được sử dụng để đánh dấu phần chuỗi suy nghĩ của mô hình trong format prompt?
   (*a*) `<logic>...</logic>`.
   (*b*) `<reasoning>...</reasoning>`.
   (*c*) `<stepbystep>...</stepbystep>`.
   (*d*) `<thinking>...</thinking>`.

6. Cặp thẻ nào được sử dụng để đánh dấu phần đáp án cuối cùng của mô hình trong format prompt?
   (*a*) `<solution>...</solution>`.
   (*b*) `<final_answer>...</final_answer>`.

(*c*) `<result>...</result>`.

(*d*) `<answer>...</answer>`.

7. Trong hàm reward `match_format_exactly()`, mô hình nhận được bao nhiêu điểm nếu output hoàn toàn khớp với pattern reasoning + answer?

   (*a*) 1.0 điểm.

   (*b*) 1.5 điểm.

   (*c*) 3.0 điểm.

   (*d*) 0.5 điểm.

8. Trong hàm reward `check_answer()`, mô hình nhận được bao nhiêu điểm nếu đáp án đoán ra (guess) hoàn toàn trùng khớp (exact match) với đáp án đúng (true_answer)?

   (*a*) 1.5 điểm.

   (*b*) 3.0 điểm.

   (*c*) -1.5 điểm.

   (*d*) 1.0 điểm.

9. Thuật toán nào được sử dụng để huấn luyện mô hình với các hàm reward đã định nghĩa?

   (*a*) PPO (Proximal Policy Optimization).

   (*b*) DPO (Direct Preference Optimization).

   (*c*) GRPO (Group Relative Policy Optimization).

   (*d*) AdamW.

10. Hàm reward `check_numbers()` được thiết kế để đánh giá tiêu chí nào của output?

    (*a*) Kiểm tra output có tuân thủ đúng định dạng thẻ `<thinking>` và `<answer>` hay không.

    (*b*) So sánh toàn bộ chuỗi văn bản trong thẻ `<answer>` với đáp án gốc.

    (*c*) Trích xuất và so sánh giá trị số học (dạng float) trong phần đáp án của output với đáp án gốc.

    (*d*) Đánh giá độ dài và chi tiết của phần giải thích trong thẻ `<thinking>`.

IV. 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). - Kiến thức về học tăng cường cơ bản và thuật toán học tăng cường GRPO. - Khái niệm về Reasoning trong mô hình lớn và các cách thức kích hoạt khả năng này cho 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 khả năng suy luận của mô hình lớn (LLM Reasoning). - Hiểu được cơ bản về lý thuyết học tăng cường cũng như thuật toán GRPO được ứng dụng. - Nắm được lý thuyết cơ bản về kỹ thuật LoRA. - Có khả năng cài đặt và thực hiện huấn luyện học tăng cường mô hình lớn trên một bộ dữ liệu bất kì nhằm cải thiện khả năng suy luận của một mô hình ngôn ngữ lớn.

- Hết -