AI VIET NAM – AI COURSE 2023

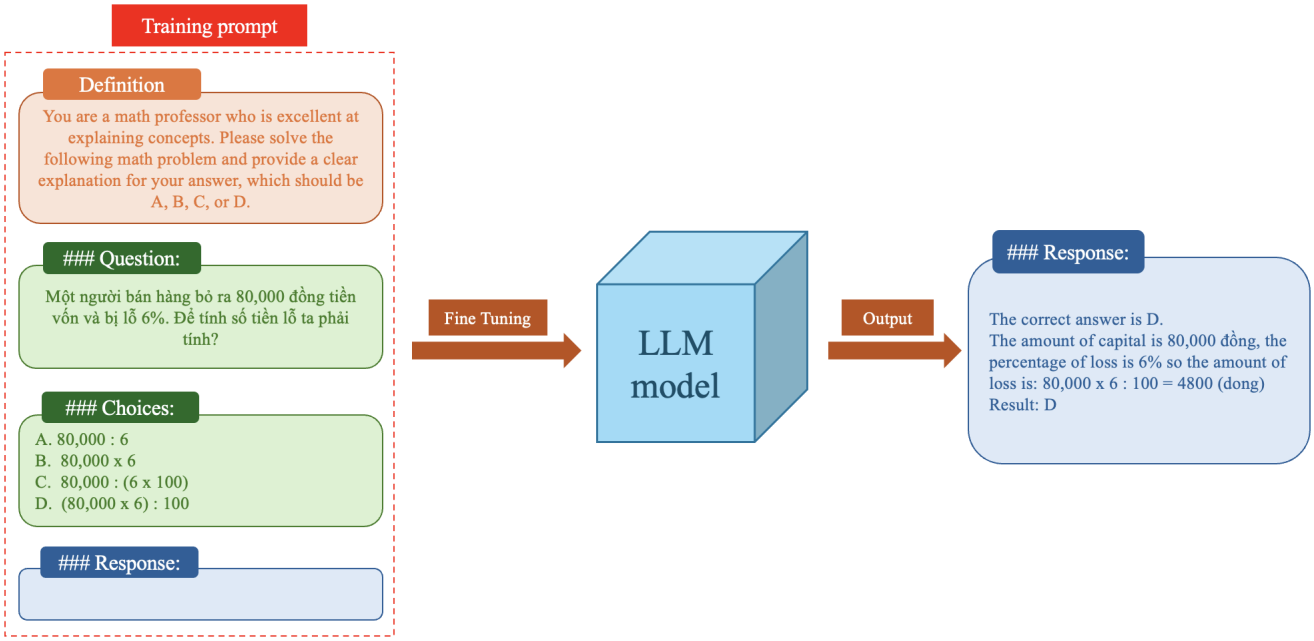
LLMs Series: Cải thiện khả năng giải toán trắc nghiệm của LLMs với Instruction Tuning

Dinh-Thang Duong, Nguyen-Thuan Duong và Quang-Vinh Dinh
PR-Team: Hoàng-Nguyên Vũ, Đăng-Nhã Nguyễn và Minh-Châu Phạm

Ngày 13 tháng 4 năm 2024

Phần I: Giới thiệu

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



Hình 1: Fine-tuning mô hình ngôn ngữ lớn với dữ liệu instruction giải toán trắc nghiệm

Trong bài viết này, chúng ta sẽ huấn luyện một mô hình ngôn ngữ lớn tiếng Việt với dữ liệu instruction giải toán trắc nghiệm. Từ đó, mô hình này sẽ có thể phần nào cải thiện khả năng giải toán.

Phần II: Cài đặt chương trình

Trong phần này, chúng ta sẽ xây dựng một Chatbot chuyên giải toán trắc nghiệm tiếng Việt sử dụng Mô hình ngôn ngữ lớn được huấn luyện chủ yếu trên bộ dữ liệu tiếng Việt là VinaLLaMA. Các bước thực hiện như sau:

1. **Cài đặt thư viện:** Chúng ta sẽ cần cài đặt một số thư viện sau để có thể chạy được một mô hình ngôn ngữ lớn từ thư viện HuggingFace:

```
!pip install -q -U bitsandbytes
!pip install -q -U datasets
!pip install -q -U git+https://github.com/huggingface/transformers.git
!pip install -q -U git+https://github.com/huggingface/peft.git
!pip install -q -U git+https://github.com/huggingface/accelerate.git
!pip install -q -U loralib
!pip install -q -U einops
!pip install -q -U googletrans==3.1.0a0
```

2. **Import các thư viện cần thiết:** Sau khi đã tải xong, chúng ta sẽ thực hiện import các thư viện đã tải cũng như một số thư viện khác để phục vụ cho chương trình:

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

from googletrans import Translator
from pprint import pprint
from datasets import load_dataset
from huggingface_hub import notebook_login
from peft import (
    LoraConfig,
    PeftConfig,
    PeftModel,
    get_peft_model,
    prepare_model_for_kbit_training
)
from transformers import (
    AutoConfig,
    AutoModelForCausalLM,
    AutoTokenizer,
    BitsAndBytesConfig
)

os.environ["CUDA_VISIBLE_DEVICES"] = "0"
```

3. **Khởi tạo mô hình:** Mô hình ngôn ngữ lớn mà chúng ta sẽ sử dụng trong bài này có tên gọi là VinaLLaMA, một mô hình được nhóm tác giả người Việt thực hiện huấn luyện trên bộ dữ liệu chủ yếu về tiếng Việt. Để khởi tạo mô hình lên trên file notebook, chúng ta sẽ chạy đoạn code sau:

```
MODEL_NAME = "vilm/vinallama-7b-chat"

bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_use_double_quant=True,
```

```
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.bfloat16
)

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

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

model.gradient_checkpointing_enable()
model = prepare_model_for_kbit_training(model)

config = LoraConfig(
    r=16,
    lora_alpha=32,
    target_modules=[
        "q_proj",
        "up_proj",
        "o_proj",
        "k_proj",
        "down_proj",
        "gate_proj",
        "v_proj"
    ],
    lora_dropout=0.05,
    bias="none",
    task_type="CAUSAL_LM"
)

model = get_peft_model(model, config)
```

Trong đó:

- **Dòng 1:** Khai báo biến chứa tên của mô hình ngôn ngữ lớn chúng ta mong muốn sử dụng, ở đây sẽ là VinaLLaMA phiên bản 7b-chat.
- **Dòng 3 - 40:** Khởi tạo mô hình và các cài đặt cần thiết.

4. **Sử dụng mô hình (trước khi huấn luyện):** Mô hình vừa khởi tạo đã được nhóm tác giả huấn luyện trên một bộ dữ liệu rất lớn, có thể thực hiện được nhiều task khác nhau. Chúng ta có thể tương tác với mô hình ngay lúc này, bằng cách viết một đoạn chat mô tả mệnh lệnh nào đó (prompt) và gửi vào mô hình. Sau một khoảng thời gian tính toán, mô hình sẽ trả về câu trả lời phù hợp. Cách thực hiện như sau:

   (a) Cài đặt một vài tham số cần thiết cho mô hình, các tham số này sẽ ảnh hưởng đến kết quả trả lời của mô hình ngôn ngữ lớn:

```
generation_config = model.generation_config
generation_config.max_new_tokens = 200
generation_config.temperature = 0.7
generation_config.top_p = 0.7
generation_config.num_return_sequences = 1
generation_config.pad_token_id = tokenizer.eos_token_id
generation_config.eos_token_id = tokenizer.eos_token_id
```

(b) **Khai báo prompt:** Chúng ta sẽ khởi tạo một biến chứa đoạn prompt, câu mệnh lệnh hoặc một đoạn tin nhắn mà chúng ta muốn gửi vào mô hình. Cụ thể trong VinaLLaMA, chúng ta sẽ có format cố định cho đoạn prompt như sau:

```
<|im_start|>system
Bạn là một trợ lí AI hữu ích. Hãy trả lời người dùng một cách chính xác.
<|im_end|>
<|im_start|>user
{your_task}
<|im_end|>
<|im_start|>assistant
```

Hình 2: Format prompt của VinaLLaMA. Trong đó, {your_task} là một đoạn văn bản mô tả một nhiệm vụ, câu hỏi hay một câu nói bất kì mà bạn mong muốn gửi đến mô hình.

Dựa vào format trên, ta có thể thử đặt một yêu cầu xây dựng một hàm Python cho mô hình như trong môi trường code sau:

```
prompt = """
<|im_start|>system
Bạn là một trợ lí AI hữu ích. Hãy trả lời người dùng một cách chính xác.
<|im_end|>
<|im_start|>user
Viết một hàm tính tổng hai số trong python
<|im_end|>
<|im_start|>assistant
""".strip()
```

Hình 3: Minh họa về cách xây dựng prompt theo format của mô hình VinaLLaMA

(c) **Chạy mô hình:** Sử dụng đoạn code dưới đây, ta đưa đoạn prompt đã khởi tạo để lấy câu trả lời từ mô hình như sau:

```
%%time
device = 'cuda' if torch.cuda.is_available() else 'cpu'

encoding = tokenizer(prompt, return_tensors="pt").to(device)
with torch.inference_mode():
    outputs = model.generate(
        input_ids=encoding.input_ids,
        attention_mask=encoding.attention_mask,
        generation_config=generation_config
    )

print(tokenizer.decode(outputs[0], skip_special_tokens=True))
```

Khi quá trình tính toán hoàn tất, ta nhận được kết quả in ra màn hình là câu trả lời của mô hình ứng với đoạn prompt:

```
<|im_start|> system
Bạn là một trợ lí AI hữu ích. Hãy trả lời người dùng một cách chính xác.

<|im_start|> user
Viết một hàm tính tổng hai số trong python

<|im_start|> assistant
Dưới đây là một hàm Python nhận hai số làm đầu vào và trả về tổng của chúng:

```python
def add_two_numbers(a, b):
    return a + b

# Ví dụ sử dụng
result = add_two_numbers(5, 3)
print(result)  # Kết quả: 8
```

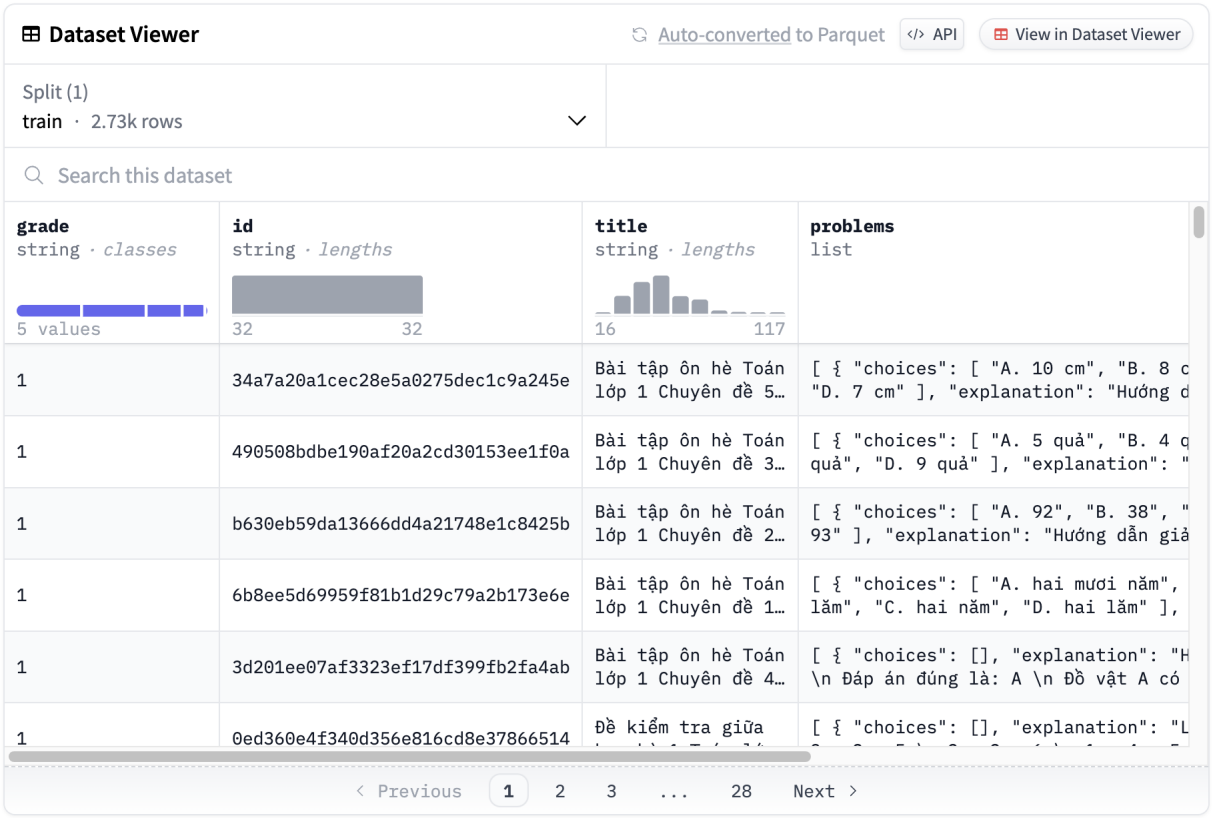
Hàm này, `add_two_numbers`, nhận hai tham số, `a` và `b`, đại diện cho hai số bạn muốn cộng. Nó sử dụng toán tử `+`
CPU times: user 20.3 s, sys: 231 ms, total: 20.5 s
Wall time: 20.6 s
```

Hình 4: Ví dụ về câu trả lời của mô hình về việc viết một hàm Python ứng với mô tả trong prompt

Như vậy, có thể thấy chỉ với mô hình gốc (gọi là pre-trained model), chúng ta đã có thể tương tác với mô hình ngôn ngữ lớn và yêu cầu thực hiện một tác vụ nào đó với độ chính xác tương đối. Trong lĩnh vực Machine Learning, chúng ta còn có thể cải thiện kết quả của pre-trained model với một task cụ thể nào đó bằng cách áp dụng một kỹ thuật được gọi là fine-tuning. Cụ thể, chúng ta sẽ tiếp tục thực hiện huấn luyện mô hình, trên một bộ dữ liệu với các task cụ thể hơn (ứng với nhu cầu và mục đích sử dụng của chúng ta).

5. **Tải bộ dữ liệu fine-tuning:** Trong bài này, vì mục tiêu của chúng ta là xây dựng chatbot chuyên dùng để giải toán trắc nghiệm tiếng Việt, nên ở phần sau chúng ta sẽ thực hiện fine-tuning VinaLLaMA trên bộ dữ liệu toán trắc nghiệm để cải thiện chất lượng câu trả lời. Ta thực hiện tải bộ dữ liệu có tên là **vi_grade_school_math_mcq** như sau:

```
data = load_dataset('hllj/vi_grade_school_math_mcq')
```



Hình 5: Minh họa một số mẫu dữ liệu trong bộ dữ liệu vừa tải về

Trong đoạn code, ta sử dụng hàm `load_data()` từ thư viện **datasets**, hàm này cho phép tải các bộ dữ liệu trong database của thư viện. Bộ dữ liệu được ta lưu vào biến `data`, khi in biến này, ta có thông tin như sau:

```
1 data
```

```
DatasetDict({
    train: Dataset({
        features: ['problems', 'grade', 'id', 'title', 'url'],
        num_rows: 2733
    })
})
```

Hình 6: Dữ liệu của biến `data`. Biến có kiểu dữ liệu là **DatasetDict**, một kiểu dữ liệu riêng biệt của thư viện **datasets**

6. **Xây dựng bộ dữ liệu fine-tuning:** Với bộ dữ liệu đã tải, chúng ta sẽ sử dụng để thực hiện fine-tuning mô hình, tức sẽ huấn luyện cho mô hình học thêm các dữ liệu từ bộ dữ liệu mới này. Các bước làm như sau:

   (a) **Xây dựng hàm tạo prompt:** Trong trường hợp huấn luyện VinaLLaMA, chúng ta cần thay đổi dữ liệu vào đúng format prompt như ở phần trước. Nhận thấy trong format prompt, ô user sẽ nhận input của người dùng, ứng với trường "prompt"của bộ dữ liệu. Ô assistant

là câu trả lời của mô hình, ứng với trường "response"của bộ dữ liệu. Vì vậy, ta sẽ xây dựng hàm để đưa vào đúng khuôn format như sau:

```
def generate_prompt(question, choices, explanation):
    return f"""
<|im_start|>system
Bạn là một chuyên gia về toán. Bạn sẽ nhận câu hỏi trắc nghiệm kèm theo các l
    ựa chọn, hãy giải step by step nếu có và chọn phương án đúng.

<|im_start|>user
### Câu hỏi:
{question}
### Các lựa chọn:
{choices}
### Câu trả lời:

<|im_start|>assistant
{explanation}
""".strip()

def generate_and_tokenize_prompt(question, choices, explanation):
    full_prompt = generate_prompt(question, choices, explanation)
    tokenized_full_prompt = tokenizer(
        full_prompt,
        padding=True,
        truncation=True
    )

    return tokenized_full_prompt
```

(b) **Xây dựng hàm tokenization:** Đối với bất kì mô hình ngôn ngữ lớn nào, để xử lý một văn bản nào, trước hết chúng ta cần thực hiện tokenization lên văn bản đó. Hiểu một cách đơn giản, chúng ta sẽ đưa văn bản từ dạng string thành một list (vector) các con số:

Ở đây, ta sẽ thiết kế hàm tạo câu prompt với điểm dữ liệu gồm cặp (response, prompt) đầu vào, sau đó thực hiện tokenize câu prompt, code cài đặt như sau:

```
def generate_and_tokenize_prompt(question, choices, explanation):
    full_prompt = generate_prompt(question, choices, explanation)
    tokenized_full_prompt = tokenizer(
        full_prompt,
        padding=True,
        truncation=True
    )

    return tokenized_full_prompt
```

(c) **Áp dụng tokenization vào bộ dữ liệu:** Với hàm tokenization vừa xây dựng, ta sử dụng đoạn code sau đây để tách các thông tin về các lựa chọn trắc nghiệm (choices), lời giải thích kèm đáp án (explanation) và câu hỏi (question). Sau đó, đưa các thông tin này vào hàm tokenization để hình thành câu prompt cho mô hình. Sau đó, sử dụng hàm `Dataset.from_list()`

```
training_samples = []
for sample in tqdm(data['train']):
    for quest in sample['problems']:
        choices = quest['choices']
        explanation = quest['explanation'].strip()
        question = quest['question']

```

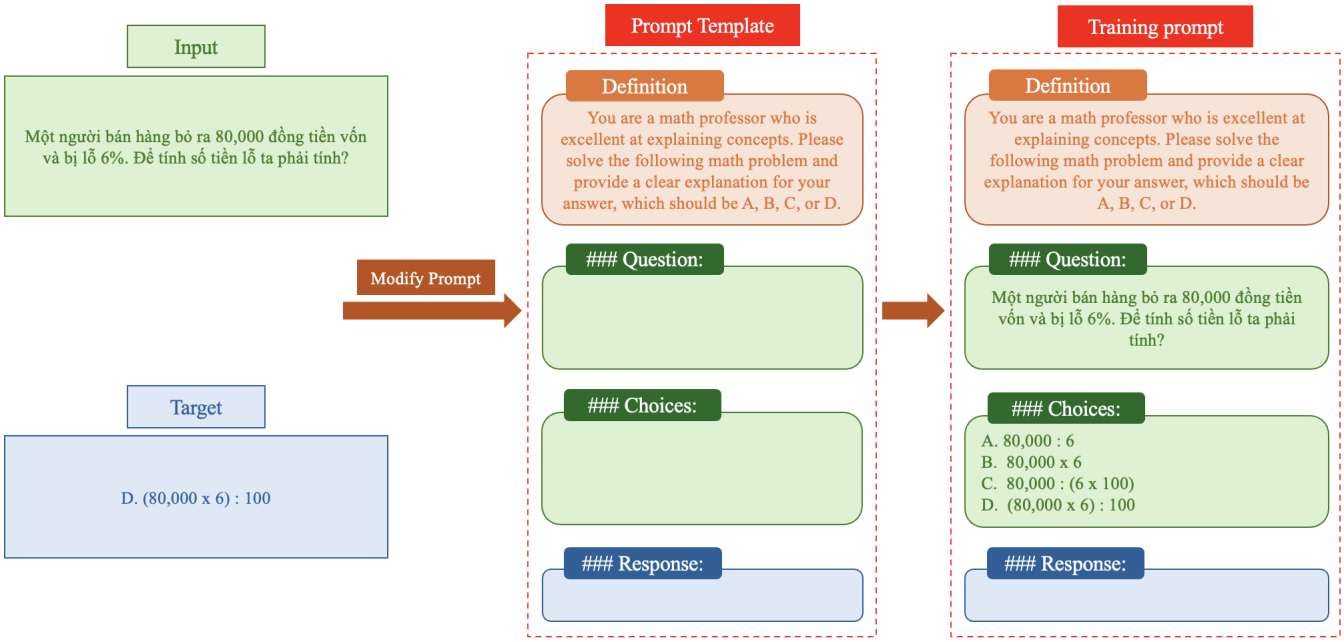
```
        if explanation == '' or question == '' or choices == []:
            continue

        try:
            question = question.split('\n \n')[1].strip()
        except:
            continue

        choices = '\n'.join(choices)
        training_sample = generate_and_tokenize_prompt(
            question, choices, explanation
        )

        training_samples.append(training_sample)

choices_data = Dataset.from_list(training_samples)
```



Hình 7: Minh họa về mẫu dữ liệu instruction giải toán trắc nghiệm.

7. **Thực hiện huấn luyện mô hình (fine-tuning):** Sau khi đã chuẩn bị xong bộ dữ liệu hoàn tất, chúng ta bắt đầu huấn luyện mô hình ngôn ngữ lớn, chạy các dòng lệnh sau:

```
training_args = transformers.TrainingArguments(
      per_device_train_batch_size=1,
      gradient_accumulation_steps=4,
      num_train_epochs=1,
      learning_rate=2e-4,
      fp16=True,
      save_total_limit=3,
      logging_steps=1,
      output_dir="experiments",
      optim="paged_adamw_8bit",
      lr_scheduler_type="cosine",
      warmup_ratio=0.05,
)
```

```
trainer = transformers.Trainer(
    model=model,
    train_dataset=data,
    args=training_args,
    data_collator=transformers.DataCollatorForLanguageModeling(tokenizer, mlm=False)
)
model.config.use_cache = False
trainer.train()
```

Khi các bạn thấy bảng dưới đây xuất hiện, điều đó chứng tỏ tiến trình huấn luyện đã bắt đầu thành công, việc còn lại của chúng ta sẽ chỉ cần chờ cho tới khi việc thực thi hoàn tất.

Step	Training Loss
1	4.489600
2	3.980300
3	2.588000
4	2.978800
5	3.521400
6	3.824300
7	4.167600
8	3.548400
9	4.254500
10	3.988800

Hình 8: Ảnh minh họa bảng hiển thị các thông tin trong quá trình thực hiện huấn luyện mô hình ngôn ngữ lớn

8. **Chạy mô hình đã fine-tuning:** Cuối cùng, ta sẽ thử tương tác với mô hình sau khi đã được fine-tuning như sau:

```
%%time
device = 'cuda' if torch.cuda.is_available() else 'cpu'

prompt = """
<|im_start|>system
Bạn là một chuyên gia về toán. Bạn sẽ nhận câu hỏi trắc nghiệm kèm theo các lựa chọn, hãy giải step by step nếu có và chọn phương án đúng.

<|im_start|>user
### Câu hỏi:
1 + 1 =
### Các lựa chọn:
A. 1
B. 2
C. 3
D. 4
### Câu trả lời:
```

```
<|im_start|>assistant
""".strip()

encoding = tokenizer(prompt, return_tensors="pt").to(device)
with torch.inference_mode():
    outputs = model.generate(
        input_ids=encoding.input_ids,
        attention_mask=encoding.attention_mask,
        generation_config=generation_config
    )

print(tokenizer.decode(outputs[0], skip_special_tokens=True))
```

Kết quả trả về của mô hình cho câu prompt trên được mô tả như hình dưới đây:

```
<|im_start|> system
Bạn là một chuyên gia về toán. Bạn sẽ nhận câu hỏi trắc nghiệm kèm theo các lựa chọn, hãy giải step by step nếu có và chọn phương án đúng.

<|im_start|> user
### Câu hỏi:
1 + 1 =
### Các lựa chọn:
A. 1
B. 2
C. 3
D. 4
### Câu trả lời:

<|im_start|> assistant
Đáp án B 1 + 1 = 2
 Chọn B.
```

Hình 9: Câu trả lời của mô hình

- Hết -