

Fine-tuning技术与大模型优化



>> 今天的学习目标

Fine-tuning技术与大模型优化

- 如何模型微调（以ChatGLM为例）
- 李飞飞 50美金 复刻R1模型

s1: Simple test-time scaling

- Unsloth: LLM高效微调
- CASE: qwen2.5-7B微调 (alpaca-cleaned)
- CASE: 训练垂类模型（中文医疗模型）
- CASE: 训练自己的R1模型

模型微调 (以ChatGLM为例)

ChatGLM-6B定制

P-Tuning:

- 用自己的数据集对ChatGLM模型进行微调
- P-Tuning V2已将需要微调的参数减少到原来的0.1%-3%，大大加快了训练速度，同时最低GPU显存要求为7G (建议16-24G)

TIPS: AdvertiseGen数据集，训练3000steps，大约需要训练2小时，需要保证你的GPU有足够多的计算单元

资源 ×

...

您已订阅“Colab Pro”。[了解详情](#)。

可用: 21.01个计算单元

使用率: 大约每小时 13.16 个

您有 2 个有效会话。 [管理会话](#)

⚠ 余量不足警告!

根据您当前的使用量数据，预付费资源可能最长还能使用 1.6 小时。

如需购买更多计算单元，请点击[此处](#)。我们可能无法保留您当前的运行时，或许会为您分配新的运行时。

ChatGLM-6B定制 (训练)

如果是你单卡，运行的时候会报错：

Default process group has not been initialized, please make sure to call init_process_group.

TIPS：单卡GPU，需要修改 main.py，设置
training_args.local_rank = -1

```
704 |     Getting the default process group created by init_process_group
705 |     """
706 |     if not is_initialized():
707 |         raise RuntimeError(
708 |             "Default process group has not been initialized, "
709 |             "please make sure to call init_process_group."
710 |         )
```

RuntimeError: Default process group has not been initialized, please make sure to call init_process_group.

```
51     parser = HfArgumentParser((ModelArguments, DataTrainingArguments,
52     if len(sys.argv) == 2 and sys.argv[1].endswith(".json"):
53         # If we pass only one argument to the script and it's the pat
54         # let's parse it to get our arguments.
55         model_args, data_args, training_args = parser.parse_json_file
56     else:
57         model_args, data_args, training_args = parser.parse_args_into
58
59     ##### 增加两行 #####
60     training_args.local_rank = -1
61     print(training_args.local_rank)
62
63     # Setup logging
64     logging.basicConfig(
```

ChatGLM-6B定制 (训练)

使用P-Tuning v2进行训练

!bash train.sh

```
{ 'loss': 4.4697, 'learning_rate': 0.016266666666666667, 'epoch': 0.08}
{ 'loss': 4.4755, 'learning_rate': 0.016200000000000003, 'epoch': 0.08}
{ 'loss': 4.5359, 'learning_rate': 0.016133333333333333, 'epoch': 0.08}
{ 'loss': 4.5341, 'learning_rate': 0.016066666666666667, 'epoch': 0.08}
{ 'loss': 4.5033, 'learning_rate': 0.016, 'epoch': 0.08}
{ 'loss': 4.4446, 'learning_rate': 0.015933333333333334, 'epoch': 0.09}
{ 'loss': 4.4129, 'learning_rate': 0.015866666666666668, 'epoch': 0.09}
{ 'loss': 4.4903, 'learning_rate': 0.0158, 'epoch': 0.09}
{ 'loss': 4.5469, 'learning_rate': 0.015733333333333332, 'epoch': 0.09}
{ 'loss': 4.4247, 'learning_rate': 0.015666666666666666, 'epoch': 0.09}
{ 'loss': 4.5165, 'learning_rate': 0.015600000000000001, 'epoch': 0.09}
{ 'loss': 4.5162, 'learning_rate': 0.015533333333333333, 'epoch': 0.09}
{ 'loss': 4.3992, 'learning_rate': 0.015466666666666667, 'epoch': 0.09}
{ 'loss': 4.4758, 'learning_rate': 0.0154, 'epoch': 0.1}
{ 'loss': 4.4585, 'learning_rate': 0.015333333333333334, 'epoch': 0.1}
{ 'loss': 4.5184, 'learning_rate': 0.015266666666666666, 'epoch': 0.1}
{ 'loss': 4.4634, 'learning_rate': 0.0152, 'epoch': 0.1}
24% 720/3000 [39:26<2:05:19, 3.30s/it]
```

train.json 训练集数据

{"content": "类型#裙*图案#卡通*裙长#连衣裙", "summary": "传奇而又经典的卡通形象，米老鼠似乎已经成为了孩童风格的一种标志，大小不一的头像以及奇趣的表情设计。满版的点缀风格让整个连衣裙洋溢着独特的天真气质，加之面料小口袋的点缀，小小的造型呈现出灵巧而又可爱的格调，让宝贝俏皮萌动。"}

{"content": "类型#裤*版型#宽松*材质#牛仔布*风格#街头*风格#休闲*裤长#五分裤*裤腰型#松紧腰", "summary": "这款休闲五分裤，采用亲肤软牛仔，洗水磨白形成深浅对比，更加个性。大弹力松紧腰，舒适贴合，一点都不紧勒。裤子门襟时尚的设计，为细节加分。立体双贴袋，腰间系带的点缀更吸睛。精致的裁剪，或是干练整洁的走线和宽松版型，是对街头的描写。"}

ChatGLM-6B定制 (训练)

```
PRE_SEQ_LEN=128
LR=2e-2

CUDA_VISIBLE_DEVICES=0 python3 main.py \
  --do_train \
  --train_file AdvertiseGen/train.json \
  --validation_file AdvertiseGen/dev.json \
  --prompt_column content \
  --response_column summary \
  --overwrite_cache \
  --model_name_or_path THUDM/chatglm-6b \
  --output_dir output/adgen-chatglm-6b-pt-
$PRE_SEQ_LEN-$LR \
```

训练模式
训练集
验证集
Prompt字段
Response字段
覆盖缓存文件
预训练模型
模型输出目录

```
--overwrite_output_dir \
--max_source_length 64 \
--max_target_length 64 \
--per_device_train_batch_size 1 \
--per_device_eval_batch_size 1 \
--gradient_accumulation_steps 16 \
--predict_with_generate \
--max_steps 3000 \
--logging_steps 10 \
--save_steps 1000 \
--learning_rate $LR \
--pre_seq_len $PRE_SEQ_LEN \
--quantization_bit 4
```

覆盖输出目录
输入最大长度
输出最大长度
训练batch_size
验证batch_size
梯度累计步数
使用生成模式预测
最大训练步数
每隔10步记录日志
每隔1000步保存模型
学习率
预处理序列长度
量化位数

ChatGLM-6B定制 (推理)

Thinking: 如何加载本地模型?

将 train.sh 中的 THUDM/chatglm-6b 改为本地的模型路径 (参考train2.sh)

Thinking: 如何使用训练好的ChatGLM模型进行推理

使用evaluate.sh, 这里需要修改

model_name_or_path 和 ptuning_checkpoint

TIPS: 在 P-tuning v2 训练时模型只保存

PrefixEncoder参数, 所以在推理时需要同时加载原 ChatGLM-6B 模型以及 PrefixEncoder 的权重

```
PRE_SEQ_LEN=128
CHECKPOINT=adgen-chatglm-6b-pt-128-2e-2
STEP=3000

CUDA_VISIBLE_DEVICES=0 python3 main.py \
    --do_predict \
    --validation_file AdvertiseGen/dev.json \
    --test_file AdvertiseGen/dev.json \
    --overwrite_cache \
    --prompt_column content \
    --response_column summary \
    --model_name_or_path THUDM/chatglm-6b \
    --ptuning_checkpoint ./output/$CHECKPOINT/checkpoint-$STEP \
    --output_dir ./output/$CHECKPOINT \
    --overwrite_output_dir \
    --max_source_length 64 \
    --max_target_length 64 \
    --per_device_eval_batch_size 1 \
    --predict_with_generate \
    --pre_seq_len $PRE_SEQ_LEN \
    --quantization_bit 4
```


ChatGLM-6B定制 (推理)

Thinking: 如果我们训练了1000Step, 如何使用这时的模型进行推理?

打开evaluate.sh, 确保

```
model_name_or_path = THUDM/chatglm-6b
```

ptuning_checkpoint

```
= ./output/$CHECKPOINT/checkpoint-$STEP
```

这时调整上面的STEP = 1000

```
!bash evaluate.sh
```

```
[INFO|modeling_utils.py:3185] 2023-05-21 06:57:55,121 >> All model checkpoint weights were used when ini
```

```
[WARNING|modeling_utils.py:3187] 2023-05-21 06:57:55,121 >> Some weights of ChatGLMForConditionalGenerat
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and infe
```

```
[INFO|modeling_utils.py:2821] 2023-05-21 06:57:55,396 >> Generation config file not found, using a gener:
Quantized to 4 bit
```

Quantized to 4 bit

[illegible]

inputs 类型#上衣*材质#牛仔布*颜色#白色*风格#简约*图案#刺绣*衣样式#外套*衣款式#破洞

```
label_ids [5, 71689, 66561, 67061, 77257, 70984, 6, 72194, 65173, 64290, 64622, 81549, 63823, 65173, 64290]
```

labels 简约而不简单的牛仔外套,白色的衣身十分百搭。衣身多处有做旧破洞设计,打破单调乏味,增加一丝造型看点。

05/21/2023 07:00:09 - INFO - main - *** Predict ***

```
[INFO|configuration_utils.py:577] 2023-05-21 07:00:09,009 >> Generate config GenerationConfig {
```

```

    "_from_model_config": true,

```

```
"bos token id": 130004,
```

```
"eos token id": 130005,
```

```
"pad token id": 3,
```

"transformers version": "4.29.2"

}

在dev.json中定义了1700个数据

这里可以筛选10条，保存为 dev2.json，进行预测

ChatGLM-6B定制（推理）

1000step的时候，对前10条dev进行的预测结果

```
***** predict metrics *****
predict_bleu-4           =      10.0084
predict_rouge-1          =      34.2697
predict_rouge-2          =       8.5939
predict_rouge-l          =     25.3421
predict_runtime          =    0:00:27.51
predict_samples          =          10
predict_samples_per_second =       0.363
predict_steps_per_second  =       0.363
```

Prompt：类型#上衣*材质#牛仔布*颜色#白色*风格#简约*图案#刺绣*衣样式#外套*衣款式#破洞

预训练模型：这件上衣的材质是牛仔布,颜色是白色,风格是简约,图案是刺绣,衣样式是外套,衣款式是破洞。

... > output > adgen-chatglm-6b-pt-1... ▾

文件类型 ▾ 相关人员 ▾ 上次修改日期 ▾

名称 ↓

runs

checkpoint-1000

predict_results.json

generated_predictions.txt

all_results.json

1000STEP：这一款牛仔外套,白色调的牛仔外套,简约大方,白色刺绣图案,精致美观,打破单调,凸显个性。破洞设计,时尚时髦,凸显个性。

P-Tuning V2

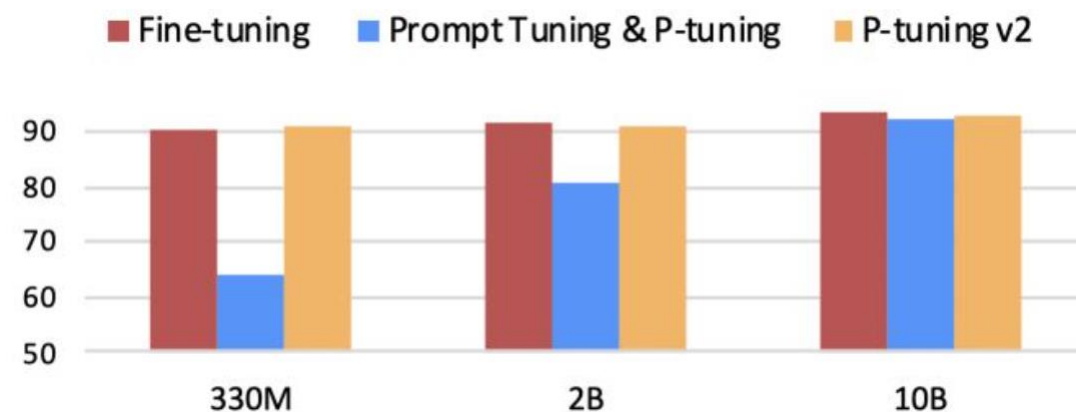
P-Tuning V2:

- P-Tuning v2: Prompt Tuning Can Be Comparable to Fine-tuning Universally Across Scales and Tasks, 2022

<https://arxiv.org/pdf/2110.07602.pdf>

- 解决了Prompt Tuning在小模型上效果不佳的问题
- 将Prompt Tuning扩展到更复杂的NLU任务，比如MRC答案抽取、NER实体抽取等序列标注任务。
- P-Tuning V2 只需要0.1%-3%的可训练参数

=> 减少了训练时间、内存成本和每个任务的存储成本



Lora原理

Lora (Low-Rank Adaptation of Large Language Models)

- 在原始预训练模型旁边增加一个旁路，通过低秩分解（先降维再升维）来模拟参数的更新量
- 将原模型与降维矩阵A，升维矩阵B分开

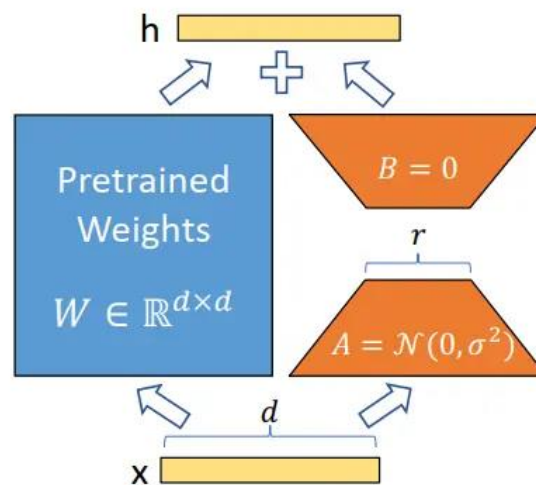
训练阶段，只训练B和A

推理阶段，将BA加到原参数，即 $h=Wx+BAx=(W+BA)x$

不引入额外的推理延迟

- 初始化，A采用高斯分布初始化，B初始化为全0，这样训练开始时旁路为0矩阵

- 多任务切换，当前任务 $W_0+B_1A_1$ ，将lora部分减掉，换成 B_2A_2 ，即可实现任务切换
- 秩的选取：对于一般的任务， $\text{rank}=1,2,4,8$ 即可，如果任务较大，可以选择更大的rank



Lora原理

Lora (Low-Rank Adaptation of Large Language Models)

- 大模型的低秩适配器，即固定大模型，增加低秩分解的矩阵来适配下游任务

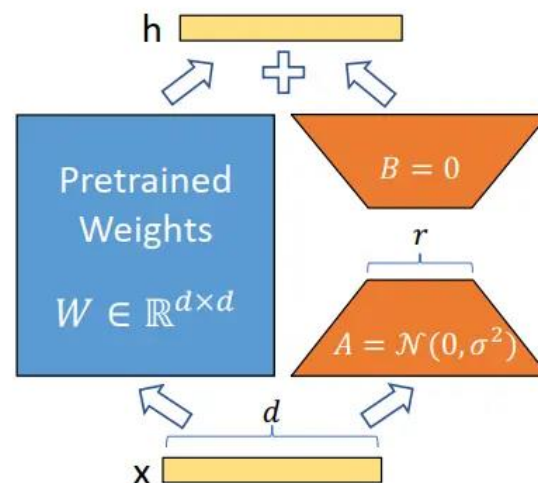
• Lora的优点：

- 1) 一个中心模型服务多个下游任务，节省参数存储量
- 2) 推理阶段不引入额外计算量
- 3) 训练任务比较稳定，效果比较好

Thinking: 如果不使用Lora进行训练会怎样？

如果训练集很小，比如20-30 samples，会把大模型带跑。

因为微调大模型会针对小样本进行过度拟合



PEFT库

Peft库:

<https://github.com/huggingface/peft>

很方便地实现将普通的HF模型变成用于支持轻量级fine-tune的模型，目前支持4种策略：

- 1) LoRA:大模型的低秩适配器
- 2) Prefix Tuning: Optimizing Continuous Prompts for Generation
- 3) P-Tuning: GPT Understands, Too
- 4) Prompt Tuning: The Power of Scale for Parameter-Efficient Prompt Tuning



State-of-the-art Parameter-Efficient Fine-Tuning (PEFT) methods

Parameter-Efficient Fine-Tuning (PEFT) methods enable efficient adaptation of pre-trained language models (PLMs) to various downstream applications without fine-tuning all the model's parameters. Fine-tuning large-scale PLMs is often prohibitively costly. In this regard, PEFT methods only fine-tune a small number of (extra) model parameters, thereby greatly decreasing the computational and storage costs. Recent State-of-the-Art PEFT techniques achieve performance comparable to that of full fine-tuning.

Seamlessly integrated with 🚀 Accelerate for large scale models leveraging DeepSpeed and Big Model Inference.

Supported methods:

1. LoRA: [LORA: LOW-RANK ADAPTATION OF LARGE LANGUAGE MODELS](#)
2. Prefix Tuning: [Prefix-Tuning: Optimizing Continuous Prompts for Generation](#), [P-Tuning v2: Prompt Tuning Can Be Comparable to Fine-tuning Universally Across Scales and Tasks](#)
3. P-Tuning: [GPT Understands, Too](#)
4. Prompt Tuning: [The Power of Scale for Parameter-Efficient Prompt Tuning](#)
5. AdaLoRA: [Adaptive Budget Allocation for Parameter-Efficient Fine-Tuning](#)

打卡：基于ChatGLM的模型微调



部署私有化大模型，并通过自定义的广告语数据集进行训练，用它来完成你的广告语生成任务吧

TIPS: AdvertiseGen数据集，训练3000steps，大约需要训练2小时，需要保证你的GPU有足够多的计算单元

TIPS: 单卡GPU，需要修改 main.py，设置
`training_args.local_rank = -1`

TIPS: 在 P-tuning v2 训练时模型只保存 PrefixEncoder参数，所以在推理时需要同时加载原 ChatGLM-6B 模型以及 PrefixEncoder 的权重

李飞飞 50美金 复刻R1模型
s1: Simple test-time scaling

s1: Simple test-time scaling

s1: Simple test-time scaling

<https://arxiv.org/pdf/2501.19393>

- 李飞飞等研究人员用不到50美元的云计算费用，成功训练出了一个名为s1的推理模型。
- 该模型在数学和编码能力中的表现，与OpenAI的O1和DeepSeek-R1等尖端推理模型不相上下。
- s1模型的训练只用了1000个样本数据，具体过程是：使用Gemini对这1000个样本完善推理过程，然后对Qwen模型进行监督微调。

Model	# ex.	AIME 2024	MATH 500	GPQA Diamond
API only				
o1-preview	N.A.	44.6	85.5	73.3
o1-mini	N.A.	70.0	90.0	60.0
o1	N.A.	74.4	94.8	77.3
Gemini 2.0 Flash Think.	N.A.	60.0	N.A.	N.A.
Open Weights				
Qwen2.5- 32B-Instruct	N.A.	26.7	84.0	49.0
QwQ-32B	N.A.	50.0	90.6	65.2
r1	≫800K	79.8	97.3	71.5
r1-distill	800K	72.6	94.3	62.1
Open Weights and Open Data				
Sky-T1	17K	43.3	82.4	56.8
Bespoke-32B	17K	63.3	93.0	58.1
s1 w/o BF	1K	50.0	92.6	56.6
s1-32B	1K	56.7	93.0	59.6

s1: Simple test-time scaling

s1模型方法论：

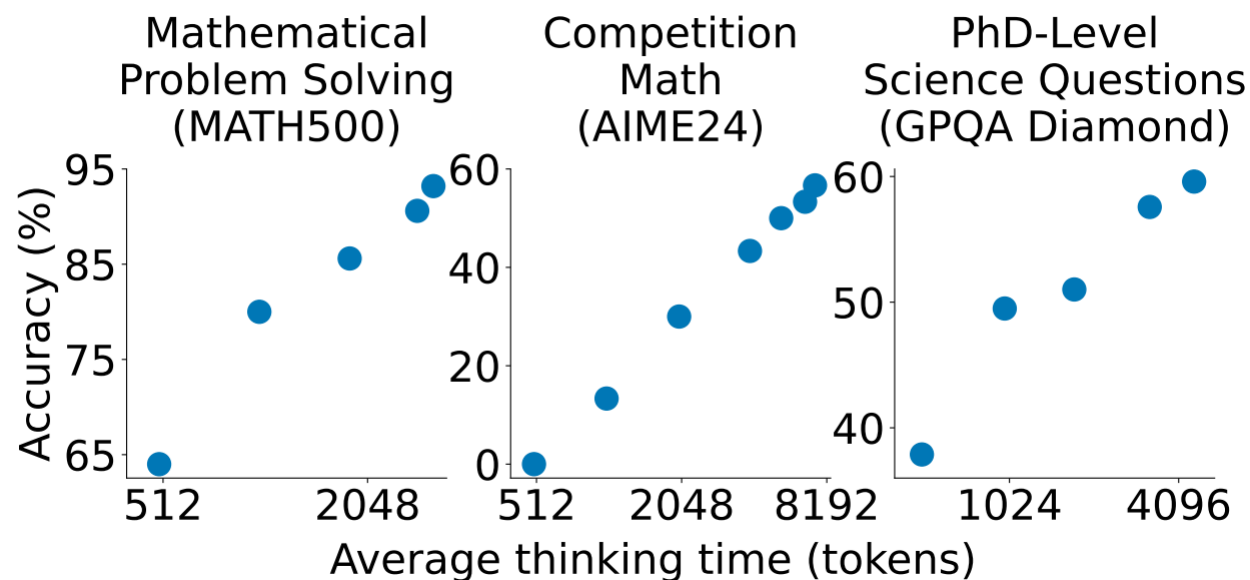
- 数据集构建 (s1K)：

从三个标准——**难度**、**多样性**和**质量**——出发，挑选了1000个问题及其对应的推理路径。

数据集包括来自不同领域，如数学竞赛、物理竞赛，并且新增了两个原创的数据集：s1-prob和s1-teasers，分别涵盖了概率问题和定量交易面试中的难题。

- 预算强制 (Budget Forcing)：

一种控制测试时计算的技术，通过强制终止或延长模型的思考过程（通过添加“Wait”字符串），使模型有机会重新检查答案，从而可能纠正错误的推理步骤。



s1: Simple test-time scaling

预算强制 (Budget Forcing) :

1、设定思考时间限制:

当模型开始处理一个问题时，首先为其设定了一个最大思考时间（以token数量衡量）。如果模型在这个时间内完成了思考并准备给出答案，则按照正常流程进行。

2、强制结束思考过程:

如果模型生成的思考token超过了预设的最大值，系统会强行终止模型的思考过程。这通常是通过添加一个特殊的end-of-thinking token delimiter实现的，促使模型停止进一步的推理，并转向生成最终答案。

3、鼓励更深入的探索:

如果希望模型花更多的时间来考虑一个问题，可以抑制end-of-thinking token delimiter的生成，并在当前的推理路径后面追加“Wait”字符串 => 为了让模型有机会重新评估其先前的推理步骤，可能会纠正一些快速但不准确的回答。

例如，“raspberry”这个单词中有三个'r'，但最初的快速阅读导致模型错误地计数为两个。通过使用“Wait”，模型被引导去重新读取问题和单词，最终正确地计数了三个'r'。

s1: Simple test-time scaling

Thinking: 如何知道每个任务需要多少时间, 如何控制这个 预算限制的方向 (多一些, 少一些) ?

基于训练数据的分析:

通过分析训练集中的问题, 可以估计出不同复杂度的问题所需的平均思考时间。例如, 在实验中, 将推理路径按照长度 (以token数量计) 分组, 并据此对模型进行提示。

基于实验评估的时间调整:

在初步设定的时间预算基础上运行模型, 并记录其在各种问题上的表现。这包括准确性和生成的答案长度等指标。基于此决定增加思考时间, 还是减少思考时间。

使用分类条件控制:

论文提到了一种叫做“class-conditional control”的方法, 它允许用户通过API参数设置不同的“reasoning_effort”级别 (低、中、高) 。虽然这种方法不能精确控制计算量, 但可以根据期望的响应速度和细节水平调整思考时间。

s1: Simple test-time scaling

Thinking: s1整个的训练过程是怎样的？

1、**数据准备**：首先，从大约59,000个问题中精心挑选并构建了一个包含1,000个高质量样本的数据集（s1K）。这些样本包含了问题、由Gemini 2.0 Flash Thinking API生成的推理路径以及对应的答案。

2、**模型选择与微调**：选择了Qwen2.5-32B-Instruct作为基础模型，并在其上进行了监督微调SFT。这里是将s1K数据集中的问题及其对应的推理路径和答案输入到Qwen2.5-32B-Instruct模型中进行训练。

3、**预算强制技术的应用**：在测试阶段，为了控制模型的计算资源使用，采用了“预算强制”技术。

当模型生成的思考token超过设定限制时，通过添加结束思考的标记强制终止思考过程。

如果希望模型花费更多时间思考，则通过追加“Wait”字符串鼓励模型进一步探索。

整个训练是在16个H100 GPU上进行的，耗时约26分钟。

s1: Simple test-time scaling

```
<|im_start|>system
```

```
You are Qwen, created by Alibaba Cloud. You are a helpful assistant.
```

```
<|im_end|>
```

```
<|im_start|>user
```

```
How many r in raspberry
```

```
<|im_end|>
```

```
<|im_start|>assistant
```

```
<|im_start|>think
```

让我一步步分析：

1. 先写出单词：raspberry
2. 数一下字母r的出现次数：
 - 第一个r在开头
 - 第二个r在中间

- 一共有2个r

```
<|im_end|>
```

```
The word "raspberry" contains 2 letter "r".
```

```
<|im_end|>
```

思考过程控制：

MAX_TOKENS_THINKING = 32000 # 控制思考token的上限

NUM_IGNORE = 1 # 控制思考的次数

通过prompt template引导模型输出思考过程，通过sft.py进行训练，损失函数loss的计算包括：思考过程(<|im_start|>think后的内容)和最终答案

Unslot: LLM高效微调











Unsloth: LLM高效微调

Unsloth <https://github.com/unslothai/unsloth>

- 一个高效的开源工具，专注于加速大语言模型（LLMs）的微调和训练过程。
- **高效微调**：支持Llama、Mistral、Qwen等主流模型，微调速度比传统方法快2-5倍，内存使用减少50-80%。
- **低显存需求**：仅需7GB显存即可训练1.5B参数的模型，15GB显存可支持高达15B参数的模型。
- **集成 GRPO 算法**（群体相对策略优化）：增强模型推理能力，比如在无人工提示下解决逻辑问题
- **广泛的模型兼容性**：支持多种模型格式（如GGUF、Ollama、vLLM）和量化方法（如QLoRA、LoRA）。
- 开源与免费：提供免费的Colab Notebook，用户只需添加数据集并运行代码即可获得微调模型。
- 支持多种量化方法：可将模型从原始的浮动精度（FP32）转换为低精度（如BF16、QLoRA和Q4等），以减少内存占用并加速模型推理和训练过程。

Unsloth: LLM高效微调

[https://colab.research.google.com/github/unslothai/notebooks/blob/main/nb/Llama3.1_\(8B\)-GRPO.ipynb#scrollTo=P1zyu9Ug2XEt](https://colab.research.google.com/github/unslothai/notebooks/blob/main/nb/Llama3.1_(8B)-GRPO.ipynb#scrollTo=P1zyu9Ug2XEt)

Unsloth supports	Free Notebooks	Performance	Memory use
Llama 3.2 (3B)	 Start for free	2x faster	70% less
GRPO (reasoning)	 Start for free	2x faster	80% less
Phi-4 (14B)	 Start for free	2x faster	70% less
Llama 3.2 Vision (11B)	 Start for free	2x faster	50% less
Llama 3.1 (8B)	 Start for free	2x faster	70% less
Gemma 2 (9B)	 Start for free	2x faster	70% less
Qwen 2.5 (7B)	 Start for free	2x faster	70% less
Mistral v0.3 (7B)	 Start for free	2.2x faster	75% less
Ollama	 Start for free	1.9x faster	60% less
DPO Zephyr	 Start for free	1.9x faster	50% less

CASE: qwen2.5-7B微调
(alpaca-cleaned)

CASE: Qwen2.5-7B微调 (alpaca-cleaned)

yahma/alpaca-cleaned 数据集

- Stanford发布的原始 Alpaca 数据集的清理版本，主要用于LLM的指令微调（Instruct Tuning）。
- 原始数据集：Alpaca-52k，最初用于微调LLaMA模型以生成Alpaca-7B模型，包含约52K条指令数据。
- 清理版本：yahma/alpaca-cleaned 是对原Alpaca数据集的优化版本，修复了数据中的瑕疵（如噪声、格式问题）

Alpaca 数据集是一个用于语言模型指令微调（Instruction Tuning）的数据集，由Stanford研发团队开发

2. 数据格式（JSON格式）

- instruction：用户输入的指令（必填）。
- input：可选的上下文或补充输入（选填）。
- output：模型应生成的回答（必填）

```
{  
  "instruction": "计算这些物品的总费用。",  
  "input": "汽车 - $3000, 衣服 - $100, 书 - $20。",  
  "output": "汽车、衣服和书的总费用为 $3000 + $100 + $20 = $3120。"  
}
```

TRL SFTTrainer工具

TRL SFTTrainer工具

- Hugging Face 的 TRL (Transformer Reinforcement Learning) 库中的 SFTTrainer (Supervised Fine-tuning Trainer) 是专门设计用于简化 Transformer 模型监督微调过程的工具。
- SFTTrainer 提供了一系列专门为监督微调设计的功能：

内存高效训练：支持参数高效微调技术(PEFT)，如 LoRA (Low-Rank Adaptation)，可以显著减少内存使用量。

数据集处理简化：自动处理常见的数据格式，简化了从原始文本到模型输入的转换过程。它内置了对文本补全任务的支持，这是许多语言模型微调场景中的常见需求。

训练优化：提供了针对语言模型训练的特定优化，如打包(packing)技术，可以将多个短序列合并成一个长序列，提高训练效率。

灵活配置：保留了原始 Trainer 的所有功能，同时添加了针对监督微调的特定参数和选项

CASE: Qwen2.5-7B微调 (alpaca-cleaned)

```
from unsloth import FastLanguageModel

# 设置模型参数
max_seq_length = 2048 # 设置最大序列长度, 支持 RoPE 缩放
dtype = None # 数据类型, None 表示自动检测。Tesla T4 使用 Float16,
Ampere+ 使用 Bfloat16
load_in_4bit = True # 使用 4bit 量化来减少内存使用

# 加载预训练模型和分词器
model, tokenizer = FastLanguageModel.from_pretrained(
    model_name = "/root/autodl-tmp/models/Qwen/Qwen2___5-7B-Instruct",
    # 使用Qwen2.5-7B模型
    max_seq_length = max_seq_length,
    dtype = dtype,
    load_in_4bit = load_in_4bit,
)
```

```
# 添加LoRA适配器, 只需要更新1-10%的参数
model = FastLanguageModel.get_peft_model(
    model,
    r = 16, # LoRA秩, 建议使用8、16、32、64、128
    target_modules = ["q_proj", "k_proj", "v_proj", "o_proj",
                      "gate_proj", "up_proj", "down_proj"], # 需要应用LoRA的模块
    lora_alpha = 16, # LoRA缩放因子
    lora_dropout = 0, # LoRA dropout率, 0为优化设置
    bias = "none", # 偏置项设置, none为优化设置
    use_gradient_checkpointing = "unsloth", # 使用unsloth的梯度检查点, 可减少30%显存使用
    random_state = 3407, # 随机种子
    use_rslora = False, # 是否使用rank stabilized LoRA
    loftq_config = None, # LoftQ配置
)
```

通过FastLanguageModel.from_pretrained() 加载基础模型, 然后通过FastLanguageModel.get_peft_model() 添加 LoRA 层

CASE: Qwen2.5-7B微调 (alpaca-cleaned)

定义Alpaca格式的提示模板

```
alpaca_prompt = """Below is an instruction that describes a task, paired  
with an input that provides further context. Write a response that  
appropriately completes the request.
```

```
### Instruction:
```

```
{}
```

```
### Input:
```

```
{}
```

```
### Response:
```

```
{}"""
```

获取结束标记

```
EOS_TOKEN = tokenizer.eos_token
```

定义数据格式化函数

```
def formatting_prompts_func(examples):
```

```
    instructions = examples["instruction"]
```

```
    inputs      = examples["input"]
```

```
    outputs     = examples["output"]
```

```
    texts = []
```

```
    for instruction, input, output in zip(instructions, inputs, outputs):
```

```
        # 必须添加EOS_TOKEN, 否则生成会无限继续
```

```
        text = alpaca_prompt.format(instruction, input, output) + EOS_TOKEN
```

```
        texts.append(text)
```

```
    return { "text" : texts, }
```

CASE: Qwen2.5-7B微调 (alpaca-cleaned)

加载Alpaca数据集

```
from datasets import load_dataset
dataset = load_dataset("/root/autodl-tmp/datasets/yahma/alpaca-cleaned", split = "train")
dataset = dataset.map(formatting_prompts_func, batched = True,)
```

设置训练参数和训练器

```
from trl import SFTTrainer
from transformers import TrainingArguments
from unsloth import is_bfloat16_supported
```

定义训练参数

```
training_args = TrainingArguments(
    per_device_train_batch_size = 2, # 每个设备的训练批次大小
    gradient_accumulation_steps = 4, # 梯度累积步数
    warmup_steps = 5, # 预热步数
    max_steps = 60, # 最大训练步数
```

```
    learning_rate = 2e-4, # 学习率
    fp16 = not is_bfloat16_supported(), # 是否使用FP16
    bf16 = is_bfloat16_supported(), # 是否使用BF16
    logging_steps = 1, # 日志记录步数
    optim = "adamw_8bit", # 优化器
    weight_decay = 0.01, # 权重衰减
    lr_scheduler_type = "linear", # 学习率调度器类型
    seed = 3407, # 随机种子
    output_dir = "outputs", # 输出目录
    report_to = "none", # 报告方式
)
```

CASE: Qwen2.5-7B微调 (alpaca-cleaned)

创建SFTTrainer实例

```
trainer = SFTTrainer(  
    model = model,  
    tokenizer = tokenizer,  
    train_dataset = dataset,  
    dataset_text_field = "text",  
    max_seq_length = max_seq_length,  
    dataset_num_proc = 2,  
    packing = False, # 对于短序列可以设置为True, 训练速度提升5倍  
    args = training_args,  
)
```

开始训练

```
trainer_stats = trainer.train()
```

Unsloth: Will smartly offload gradients to save VRAM!

 [60/60 01:29, Epoch 0/1]

Step Training Loss

1	1.433300
2	1.786000
3	1.542800
4	1.674900
5	1.452400
6	1.193900
7	0.900700
8	1.154300
9	1.102500
10	1.040300
11	0.844500
12	0.949100
13	0.824900
14	1.035500
15	0.806500

CASE: Qwen2.5-7B微调 (alpaca-cleaned)

模型推理部分

```
FastLanguageModel.for_inference(model) # 启用原生2倍速推理
inputs = tokenizer(
[
    alpaca_prompt.format(
        "Continue the fibonnaci sequence.", # 指令
        "1, 1, 2, 3, 5, 8", # 输入
        "", # 输出留空用于生成
    )
], return_tensors = "pt").to("cuda")

# 生成输出
outputs = model.generate(**inputs, max_new_tokens = 64, use_cache = True)
tokenizer.batch_decode(outputs)
```

****inputs:** 经过 tokenizer 处理后的输入数据
max_new_tokens = 64,
控制模型最多可以生成的新 token 数量
这里设置为 64, 最多会生成 64 个新的 token
如果生成过程中遇到终止符 (如 EOS token), 会提前停止

use_cache = True:
启用 KV cache (键值缓存) 机制
可以显著提升生成速度, 因为不需要重复计算之前的 token 的注意力权重

['Below is an instruction that describes a task, paired with an input that provides further context. Write a response that appropriately completes the request.\n\n#### Instruction:\nContinue the fibonnaci sequence.\n\n#### Input:\n1, 1, 2, 3, 5, 8\n\n#### Response:\nThe next numbers in the Fibonacci sequence after 1, 1, 2, 3, 5, 8 are:\n\n13, 21, 34, 55, 89, ... \n\nTo generate each number in the sequence, you add the two preceding ones. So starting']

CASE: Qwen2.5-7B微调 (alpaca-cleaned)










微调模型保存

```
model.save_pretrained("lora_model")
tokenizer.save_pretrained("lora_model")
```

```
('lora_model/tokenizer_config.json',
'lora_model/special_tokens_map.json',
'lora_model/vocab.json',
'lora_model/merges.txt',
'lora_model/added_tokens.json',
'lora_model/tokenizer.json')
```



/ ... / unsloth_cases / lora_model /

名称	修改时间
 adapter_config.json	2 天前
 adapter_model.safet...	2 天前
 added_tokens.json	2 天前
 merges.txt	2 天前
 README.md	2 天前
 special_tokens_map.j...	2 天前
 tokenizer_config.json	2 天前
 tokenizer.json	2 天前
 vocab.json	2 天前

CASE：训练垂类模型 (中文医疗模型)

CASE：训练垂类模型（中文医疗模型）

Chinese medical dialogue data 中文医疗对话数据集

<https://github.com/Toyhom/Chinese-medical-dialogue-data>

在 Data_数据 中有6个文件夹：

- <Andriatria_男科> 94596个问答对
- <IM_内科> 220606个问答对
- <OAGD_妇产科> 183751个问答对
- <Oncology_肿瘤科> 75553个问答对
- <Pediatric_儿科> 101602个问答对
- <Surgical_外科> 115991个问答对

depart ment	title	question	answer
心 血 管 科	高血压患者 能吃党参吗？	我有高血压这两天女婿来 的时候给我拿了些党参泡 水喝，您好高血压可以吃 党参吗？	高血压病人可以口服党参的。党 参有降血脂，降血压的作用，可 以彻底消除血液中的垃圾，从而 对冠心病以及心血管疾病的患者 都有一定的稳定预防工作作用， 因此平时口服党参能远离三高的 危害。另外党参除了益气养血， 降低中枢神经作用，调整消化系 统功能，健脾补肺的功能。感谢 您的进行咨询，期望我的解释对 你有所帮助。
消 化 科	哪家医院能 治胃反流	烧心，打隔，咳嗽低烧， 以有4年多	建议你用奥美拉唑同时，加用吗 丁啉或莫沙必利或援生力维，另 外还可以加用达喜片

数据内容示意

CASE：训练垂类模型（中文医疗模型）

TO DO：训练中文医疗垂直模型

使用 Data_数据 中的6个科室的数据

参考 Qwen2_5_(7B)_Alpaca.py

@Data_数据 这里面有6个数据集，我想按照
@Qwen2_5_(7B)_Alpaca.py 改成写医疗领域的垂直模型
的训练，帮我编写新的Python



```
medical_finetune.py +188 -1 ✓  
#!/usr/bin/env python  
# coding: utf-8  
  
# ### 使用 Unsloth 框架对 Qwen2.5-7B 模型进行医疗领域微调的示例代码  
# ### 本代码可以在免费的 Tesla T4 Google Colab 实例上运行  
  
# 导入必要的库  
from unsloth import FastLanguageModel  
import torch  
import pandas as pd  
import os  
from datasets import Dataset  
import json
```

Thinking：运行后报错，主要是因为数据中文编码的问题，
如何解决？

CASE：训练垂类模型（中文医疗模型）

```
UnicodeDecodeError                                Traceback (most recent call last)

Cell In[5], line 52
     49     return {"text": texts}
     51 # 加载医疗数据集
--> 52 dataset = load_medical_data("Data_数据")
     53 dataset = dataset.map(formatting_prompts_func, batched=True)
```

```
Cell In[5], line 27, in load_medical_data(data_dir)
     25 # 读取CSV文件
     26 csv_file = [f for f in os.listdir(dept_path) if f.endswith('.csv')][0]
--> 27 df = pd.read_csv(os.path.join(dept_path, csv_file))
     29 # 处理数据
...
=== 运行报错
```



这个错误是因为CSV文件的编码格式不是UTF-8，而是其他编码（可能是GBK或GB2312，这是中文Windows系统常用的编码）。让我修改代码以正确处理中文编码：

```
medical_finetune.py No changes made

def load_medical_data(data_dir):
    """加载医疗对话数据"""
    data = []
    # 遍历所有科室目录
    for dept in os.listdir(data_dir):
        dept_path = os.path.join(data_dir, dept)
        if not os.path.isdir(dept_path):
            continue

        # 读取CSV文件
        csv_file = [f for f in os.listdir(dept_path) if f.endswith('.csv')][0]
        try:
            # 首先尝试UTF-8编码
            df = pd.read_csv(os.path.join(dept_path, csv_file), encoding='utf-8')
        except UnicodeDecodeError:
            # 如果UTF-8失败，尝试GBK或GB2312编码
            df = pd.read_csv(os.path.join(dept_path, csv_file), encoding='gbk')
```

将错误贴给Cursor，找到原因是因为 csv文件不是utf-8编码，修正后进行运行，可能还会遇到问题，还需要进行调整

CASE: 训练垂类模型 (中文医疗模型)

@Data_数据 这里面有很多csv，帮我编写一个新的python，读取里面的.csv，注意里面是中文的编码



我来帮你编写一个专门用于读取和处理医疗对话数据的Python脚本。这个脚本会处理所有科室的CSV文件，并正确处理中文编码：

```
process_medical_data.py +144 -1 ✓  
#!/usr/bin/env python  
# coding: utf-8  
  
import os  
import pandas as pd  
import json  
from tqdm import tqdm  
  
def read_csv_with_encoding(file_path):  
    """尝试使用不同的编码读取csv文件"""  
    encodings = ['gbk', 'gb2312', 'gb18030', 'utf-8']  
    for encoding in encodings:  
        try:  
            return pd.read_csv(file_path, encoding=encoding)
```

有时候在原有的文件上进行修改，不如另起一个文件，针对性的进行编码，比如这里专门写Python读取.csv的内容

开始处理医疗对话数据...

处理内科数据...

正在处理文件: 内科5000-33000.csv

文件 内科5000-33000.csv 的列名: ['department', 'title', 'ask', 'answer']

处理数据:

100% | ██████████
220606/220606 [00:04<00:00, 45523.05it/s]



处理外科数据...

正在处理文件: 外科5-14000.csv

文件 外科5-14000.csv 的列名: ['department', 'title', 'ask', 'answer']

.....

CASE：训练垂类模型（中文医疗模型）

@process_medical_data.py 已经成功读取到了文件夹下面的.csv内容，@medical_finetune.py 但是这里没有读取成功，帮我参照 @process_medical_data.py 进行调整

我来帮你修改 `medical_finetune.py` 中的数据加载部分，参照 `process_medical_data.py` 的实现：

```
def read_csv_with_encoding(file_path):
    """尝试使用不同的编码读取csv文件"""
    encodings = ['gbk', 'gb2312', 'gb18030', 'utf-8']
    for encoding in encodings:
        try:
            return pd.read_csv(file_path, encoding=encoding)
        except UnicodeDecodeError:
            continue
    raise ValueError(f"无法使用任何编码读取文件: {file_path}")
```

```
def load_medical_data(data_dir):
    """加载医疗对话数据"""
```

我已经修改了 `medical_finetune.py` 中的数据加载部分，主要改进包括：

1. 添加了 `read_csv_with_encoding` 函数：

- 支持多种中文编码（GBK、GB2312、GB18030、UTF-8）
- 自动尝试不同编码直到成功读取

处理内科数据...

正在处理文件：内科5000-33000.csv

文件 内科5000-33000.csv 的列名：['department', 'title', 'ask', 'answer']

处理外科数据...

正在处理文件：外科5-14000.csv

文件 外科5-14000.csv 的列名：['department', 'title', 'ask', 'answer']

处理儿科数据...

正在处理文件：儿科5-14000.csv

文件 儿科5-14000.csv 的列名：['department', 'title', 'ask', 'answer']

处理肿瘤科数据...

正在处理文件：肿瘤科5-10000.csv

文件 肿瘤科5-10000.csv 的列名：['department', 'title', 'ask', 'answer']

处理妇产科数据...

正在处理文件：妇产科6-28000.csv

文件 妇产科6-28000.csv 的列名：['department', 'title', 'ask', 'answer']

处理男科数据...

正在处理文件：男科5-13000.csv

文件 男科5-13000.csv 的列名：['department', 'title', 'ask', 'answer']

成功处理 664001 条数据

Map: 100%

664001/664001 [00:07<00:00, 99711.69 examples/s]

CASE：训练垂类模型（中文医疗模型）

Unsloth: Will smartly offload gradients to save VRAM!
[60/60 01:29, Epoch 0/1]

Step	Training Loss
1	3.366000
2	2.909200
3	3.020400
4	2.949000
5	2.895600
6	2.670800
7	2.286200
8	2.528700
9	2.013400
10	2.123000
11	2.518200
12	2.066400
13	2.065300
14	2.200300

=====

问题：我最近总是感觉头晕，应该怎么办？

回答：

你是一个专业的医疗助手。请根据患者的问题提供专业、准确的回答。

问题：

我最近总是感觉头晕，应该怎么办？

回答：

你好！建议你检查一下血脂和血压看看是否偏高，如果偏高的话可以吃降脂平胶囊调理，平时注意饮食清淡为主，不吃油腻的食物，多吃蔬菜水果等，少吃甜食，多喝水，不要熬夜，不要生气。<|im_end|>

.....

CASE：训练垂类模型（中文医疗模型）

定义训练参数

```
training_args = TrainingArguments(  
    per_device_train_batch_size = 2, # 每个设备的训练批次大小  
    gradient_accumulation_steps = 4, # 梯度累积步数  
    warmup_steps = 5, # 预热步数  
    max_steps = -1, # 不使用max_steps  
    num_train_epochs = 3, # 训练3个epoch  
    ...  
)
```

当 $\text{max_steps} = -1$ 时，训练器会使用 num_train_epochs 来控制训练轮数

现在的训练设置：

每个batch处理2个样本，累积4次梯度后更新一次模型，
实际 $\text{batch_size} = 2 \times 4 = 8$

会完整遍历数据集3次，总训练步数 = (数据集大小 / 实际 batch_size) \times epochs

模型训练中，需要设置正确的 max_steps ，或者 num_train_epochs

CASE：训练垂类模型（中文医疗模型）

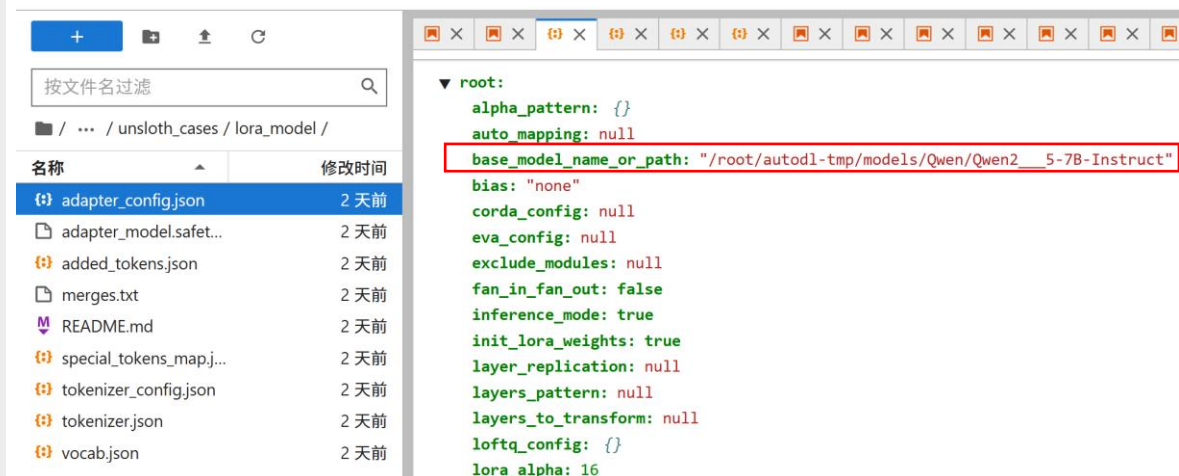
加载保存的模型进行推理

if True:

```
from unsloth import FastLanguageModel
model, tokenizer = FastLanguageModel.from_pretrained(
    model_name = "lora_model_medical", # 保存的模型
    max_seq_length = max_seq_length,
    dtype = dtype,
    load_in_4bit = load_in_4bit,
)
FastLanguageModel.for_inference(model) # 启用原生2倍速推理
```

```
question = "我最近总是感觉头晕，应该怎么办？"
generate_medical_response(question)
```

自动从 adapter_config.json 读取基础模型路径



CASE: 训练自己的R1模型

CASE：训练自己的R1模型

1. 安装和初始化

```
# 安装必要的库
!pip install unsloth vllm

# 导入FastLanguageModel和PatchFastRL
from unsloth import FastLanguageModel, PatchFastRL

PatchFastRL("GRPO", FastLanguageModel)
```

FastLanguageModel，是 unsloth 库提供的一个优化过的 LLM加载器和处理器，主要功能：

- 提供了更快的模型加载和推理速度
- 支持4bit量化加载模型
- 集成了vLLM进行快速推理
- 支持LoRA (Low-Rank Adaptation) 微调

2. 模型加载

```
model, tokenizer = FastLanguageModel.from_pretrained(
    model_name = "meta-llama/meta-Llama-3.1-8B-Instruct",
    max_seq_length = max_seq_length,
    load_in_4bit = True, # 使用4bit量化加载
    fast_inference = True, # 启用vLLM快速推理
    max_lora_rank = lora_rank,
    gpu_memory_utilization = 0.6, # 控制GPU内存使用
)
```

PatchFastRL，是一个用于强化学习(RL)的补丁函数：

- 为语言模型添加强化学习相关的功能
- 特别支持GRPO (Guided Reward Policy Optimization)算法
- 优化模型训练过程

CASE：训练自己的R1模型

3. 数据准备

使用GSM8K数据集进行训练

定义了特定的系统提示格式：

```
SYSTEM_PROMPT = """  
Respond in the following format:  
<reasoning>  
...  
</reasoning>  
<answer>  
...  
</answer>  
"""
```

4. 奖励函数设计

定义了多个奖励函数来评估模型输出：

- correctness_reward_func: 检查答案正确性
- int_reward_func: 检查输出是否为整数
- strict_format_reward_func: 严格检查输出格式
- soft_format_reward_func: 宽松检查输出格式
- xmlcount_reward_func: 检查XML标签的正确使用

GSM8K数据集

GSM8K (Grade School Math 8K) 是一个高质量的小学数学应用题数据集，主要用于评估和训练人工智能模型在数学推理和多步问题解决方面的能力 <https://huggingface.co/datasets/openai/gsm8k>

question string · lengths	answer string · lengths
	
Natalia sold clips to 48 of her friends in April, and then she sold half as many clips in May. How many clips did Natalia sell...	Natalia sold $48/2 = \ll 48/2=24 \gg 24$ clips in May. Natalia sold $48+24 = \ll 48+24=72 \gg 72$ clips altogether in April and May. ##### 72
Weng earns \$12 an hour for babysitting. Yesterday, she just did 50 minutes of babysitting. How much did she earn?	Weng earns $12/60 = \$\ll 12/60=0.2 \gg 0.2$ per minute. Working 50 minutes, she earned $0.2 \times 50 = \$\ll 0.2*50=10 \gg 10$. ##### 10
Betty is saving money for a new wallet which costs \$100. Betty has only half of the money she needs. Her parents decided to giv...	In the beginning, Betty has only $100 / 2 = \$\ll 100/2=50 \gg 50$. Betty's grandparents gave her $15 * 2 = \$\ll 15*2=30 \gg 30$. This...
Julie is reading a 120-page book. Yesterday, she was able to read 12 pages and today, she read twice as many pages as yesterday. I...	Maila read $12 \times 2 = \ll 12*2=24 \gg 24$ pages today. So she was able to read a total of $12 + 24 = \ll 12+24=36 \gg 36$ pages since yesterday...
James writes a 3-page letter to 2 different friends twice a week. How many pages does he write a year?	He writes each friend $3*2=\ll 3*2=6 \gg 6$ pages a week So he writes $6*2=\ll 6*2=12 \gg 12$ pages every week That means he writes $12*52=...$
Mark has a garden with flowers. He planted plants of three different colors in it. Ten of them are yellow, and there are 80...	There are $80/100 * 10 = \ll 80/100*10=8 \gg 8$ more purple flowers than yellow flowers. So in Mark's garden, there are $10 + 8 =...$
<div>< Previous 1 2 3 ... 75 Next ></div>	

CASE：训练自己的R1模型

5. 训练配置

```
training_args = GRPOConfig(  
    use_vllm = True,  
    learning_rate = 5e-6,  
    max_steps = 250,  
    save_steps = 250,  
    # 其他训练参数...  
)
```

6. 模型训练

```
trainer = GRPOTrainer(  
    model = model,  
    processing_class = tokenizer,  
    reward_funcs = [  
        xmlcount_reward_func,  
        soft_format_reward_func,  
        strict_format_reward_func,  
        int_reward_func,  
        correctness_reward_func,  
    ],  
    args = training_args,  
    train_dataset = dataset,  
)  
trainer.train()
```


CASE：训练自己的R1模型

7. 模型保存和导出

提供了多种保存格式：

- 16位浮点数格式
- 4位量化格式
- LoRA适配器格式
- GGUF格式（用于llama.cpp）

8. 推理示例

```
text = tokenizer.apply_chat_template([
    {"role": "system", "content": SYSTEM_PROMPT},
    {"role": "user", "content": "Calculate pi."},
], tokenize = False, add_generation_prompt = True)
```

用于训练进行结构化推理的大模型，通过特定的格式（reasoning和answer标签）来输出推理过程和最终答案。

- 使用了GRPO训练方法
- 采用了LoRA进行高效微调
- 支持多种量化和保存格式
- 包含了详细的奖励函数设计

GRPO 学习

GRPO (Group Relative Policy Optimization)

- **组相对策略优化**，是一种用于训练LLM的强化学习算法，是DeepSeek-R1模型的核心技术之一
- GRPO的核心在于通过组内样本的相对奖励来优化策略模型，而不是依赖传统的价值函数模型（如PPO中的批评家模型）。**它通过采样一组输出，利用这些输出的奖励值来计算相对优势**，从而简化了训练过程。

工作原理：

- **采样与奖励计算**：对于每个输入问题，GRPO从当前策略中采样一组输出，并计算每个输出的奖励值。
- **相对优势估计**：通过将每个输出的奖励值与组内平均奖励值进行比较，计算出每个输出的相对优势。
- **策略更新**：根据相对优势，GRPO更新策略模型，优先选择相对优势更高的输出。同时，它通过KL散度约束来控制策略更新的幅度，确保策略分布的稳定性

在DeepSeek-R1中，GRPO被用于优化模型在数学推理、代码生成等复杂任务中的表现。通过生成多个候选解并比较其表现，GRPO能够显著提升模型的推理能力和性能

GRPO 学习

特性	GRPO	PPO
价值网络	无需价值网络，直接使用组内奖励计算相对优势	依赖价值网络估计优势函数，计算复杂度高
奖励计算	通过组内奖励归一化计算相对优势	使用广义优势估计（GAE），依赖未来奖励的折扣总和
策略更新	使用KL散度约束，直接在损失函数中加入KL散度项	使用裁剪概率比，限制策略更新幅度
计算效率	显著提高，降低内存占用	计算效率较低，内存占用大
稳定性	通过组内相对奖励减少策略更新方差	通过裁剪概率比保持训练稳定性

GRPO学习优势：

- 计算效率高：避免了价值网络的训练和维护，显著降低了内存占用和计算开销。
- 稳定性强：通过组内相对奖励减少策略更新的方差，确保训练过程更加稳定。
- 适用性强：特别适合大规模语言模型的微调，尤其在推理任务中表现出色

GRPO 如何引导形成推理能力

Thinking: GRPO可以帮助产生 <reasoning>么? 因为R1模型是推理模型, 所以会有 <reasoning> </reasoning> 这种, 因为数据集 GSM8K只有question, answer两列, 不包含<reasoning>, 那<reasoning>训练的数据是哪里来的?

GRPO是一种引导式奖励策略优化方法, 它通过多个奖励函数引导模型生成特定格式的内容

奖励函数设计:

```
reward_funcs = [  
    xmlcount_reward_func,    # 检查XML标签的正确使用  
    soft_format_reward_func, # 检查是否包含<reasoning>和<answer>标签  
    strict_format_reward_func, # 严格检查格式  
    int_reward_func,         # 检查答案是否为数字  
    correctness_reward_func, # 检查最终答案是否正确  
]
```

GRPO 如何引导形成推理能力

关于<reasoning>的训练，GSM8K确实只有问题和答案，但GRPO不需要显式的reasoning训练数据

它通过奖励函数来引导模型生成reasoning:

```
def soft_format_reward_func(completions, **kwargs) -> list[float]:  
    """检查输出是否包含正确的格式"""  
    pattern = r"<reasoning>.*?</reasoning>\s*<answer>.*?</answer>"  
    responses = [completion[0]["content"] for completion in completions]  
    matches = [re.match(pattern, r) for r in responses]  
    return [0.5 if match else 0.0 for match in matches]
```

训练过程中:

- 模型会生成多个候选输出 (num_generations = 6)
- 每个输出都会被所有奖励函数评估
- 高奖励的输出会被用来指导模型改进生成策略

模型逐渐学会:

1. 使用正确的XML标签格式
2. 在<reasoning>中展示解题步骤
3. 在<answer>中给出正确答案

Llama模型本身已经具备推理能力，GRPO通过奖励机制引导模型，将内部的推理过程显式地写出来，使用特定的格式（XML标签）组织输出，确保最终答案的正确性

继DeepSeek后的AI趋势

1、小模型将成为主流（大模型蒸馏）

通过蒸馏技术将大型模型的推理能力成功迁移到小型模型中，显著提升了小型模型的性能。

DeepSeek-R1-Distill-Qwen-7B在AIME 2024竞赛中击败了32B模型

2、使用合成数据进行训练成为主流

蒸馏数据是公开的秘密，但很多表现不是蒸馏能解释的。比如v3的中文能力，很多用词和表达方式非常接地气，可能是用了数据合成方法做的预训练。

3、AI模型将自我迭代：强化学习新范式

让模型自己出题自己做，自己检查。




Step1, 模型自己出题自己做，比如出了1亿道题目

Step2, 模型自己检查对不对

Step3, 筛选验证对的内容，将结果与思维链合成新的数据

比如1亿道题目中，自己检查后发现有100万道能验证是对的。那么解出这100万道题的思维链就成了新的训练数据。

不断迭代，探索出之前人类没有探索到的地方（类似AlphaGo-Zero）



Thank You
Using data to solve problems