



ChatGLM2-6B SFT 微调实战

Background

ChatGLM2-6B

ChatGLM2-6B是开源中英双语对话模型 [ChatGLM-6B](#) 的第二代版本。

ChatGLM 模型是由清华大学开源的、支持中英双语问答的对话语言模型，并针对中文进行了优化。该模型基于General Language Model (GLM) 架构，具有62 亿参数。结合模型量化技术，用户可以在消费级的 显卡 上进行本地部署（INT4 量化级别下最低只需 6GB 显存）。

为了方便下游开发者针对自己的应用场景定制模型，同时实现了基于 [P-Tuning v2](#) 的高效参数微调方法，INT4 量化级别下最低只需 7GB 显存即可启动微调。

不过，由于 ChatGLM-6B 的规模较小，目前已知其具有相当多的局限性，如事实性/数学逻辑错误，可能生成有害/有偏见内容，较弱的上下文能力，自我认知混乱，以及对英文指示生成与中文指示完全矛盾的内容。

硬件需求

量化等级	最低 GPU 显存 (推理)	最低 GPU 显存 (高效参数微调)
FP16 (无量化)	13 GB	14 GB
INT8	8 GB	9 GB
INT4	6 GB	7 GB

我的硬件参数：Amazon SageMaker平台，主要使用两块 NVIDIA A10-24GB GDDR6训练和部署。

P-Tuning 微调

P-Tuning 是一种对预训练语言模型进行少量参数微调的技术。所谓预训练语言模型，就是指在大规模的语言数据集上训练好的、能够理解自然语言表达并从中学习语言知识的模型。P-Tuning 所做的就是根据具体的任务，对预训练的模型进行微调，让它更好地适应于具体任务。相比于重新训练一个新的模型，微调可以大大节省计算资源，同时也可以获得更好的性能表现。

1. 部署ChatGLM2-6B

1.1 安装环境

1.1.1 配置基础环境

安装pytorch: <https://pytorch.org/>

cuda: <https://developer.nvidia.com/cuda-downloads>

使用如下代码验证是否成功安装

```
1 import torch
2 torch.cuda.is_available()
3 print(torch.version.cuda)
```

我使用的python版本为3.10, cuda 版本为12.0, pytorch 版本为 11.7。

注意:python版本不能高于3.10, cuda版本需要和pytorch版本对应, 具体信息查看<https://pytorch.org/get-started/previous-versions/>

1.1.2 从Github上下载项目文件

首先需要下载Github仓库:

```
1 git clone https://github.com/THUDM/ChatGLM2-6B
2 cd ChatGLM2-6B
```

接着使用pip安装依赖:

```
1 pip install -r requirements.txt
```

1.1.3 下载模型

官方的代码会直接从HuggingFace上下载模型:

```
1 from transformers import AutoTokenizer, AutoModel>>> tokenizer =
  AutoTokenizer.from_pretrained("THUDM/chatglm2-6b", trust_remote_code=True)
2 model = AutoModel.from_pretrained("THUDM/chatglm2-6b", trust_remote_code=True,
  device='cuda')
```

但是这样做不易于我们管理模型的地址, 而且网速较慢的情况下可能会出现下载失败的情况。

我们可以直接从Hugging Face上下载地址:

先安装Git LFS并验证成功:

```
1 $ git lfs install> Git LFS initialized.
```

接下来在刚刚从Github上clone的文件夹中打开命令行, 输入:

```
1 git clone https://huggingface.co/THUDM/chatglm2-6b
```

就可以下载模型了!

请确认一共有7个model文件, 每次加载时都会读取这7个文件。

pytorch_model-00001-of-00007.bin	9 days ago
pytorch_model-00002-of-00007.bin	9 days ago
pytorch_model-00003-of-00007.bin	9 days ago
pytorch_model-00004-of-00007.bin	9 days ago
pytorch_model-00005-of-00007.bin	9 days ago
pytorch_model-00006-of-00007.bin	9 days ago
pytorch_model-00007-of-00007.bin	9 days ago

将模型下载到本地之后，将以上代码中的 THUDM/chatglm2-6b 替换为你本地的 chatglm2-6b 文件夹的路径，即可从本地加载模型。

1.2 部署模型

下载好模型后，我们就可以尝试部署了。可以通过两种方法部署模型。

1.2.1 直接部署

第一种是直接用命令行或者终端部署。将path更改为GLM2文件夹的绝对地址后运行下面的代码：

```
1 from transformers import AutoTokenizer, AutoModel
2 import torch
3
4 #设置运行的CPU，如果没有多块CPU就填0
5 DEVICE = "cuda"
6 DEVICE_ID = "3"
7 CUDA_DEVICE = f"{DEVICE}:{DEVICE_ID}" if DEVICE_ID else DEVICE
8
9 if torch.cuda.is_available():
10     torch.cuda.set_device(CUDA_DEVICE)
11
12 if __name__ == '__main__':
13     #path 更改为GLM2文件夹的绝对地址
14     path = "你的文件夹地址"
15     from transformers import AutoTokenizer, AutoModel
16     tokenizer = AutoTokenizer.from_pretrained(path, trust_remote_code=True)
17     model = AutoModel.from_pretrained(path, trust_remote_code=True, device=CUDA_DEVICE)
18     torch.cuda.set_device(CUDA_DEVICE)
19     model.eval()
20
21
22     response1, history = model.chat(tokenizer, "你好", history=[])
23     print(response1)
24     response1, history = model.chat(tokenizer, "晚上睡不着应该怎么办", history=hi
25     print(response1)
```

Loading checkpoint shards: 100%

7/7 [00:07<00:00, 1.05it/s]

你好👋！我是人工智能助手 ChatGLM2-6B，很高兴见到你，欢迎问我任何问题。
如果你晚上睡不着，可以尝试以下一些方法来帮助你入睡：

1. 放松身体：你可以通过轻松的活动，如伸展、深呼吸或冥想放松你的身体。
2. 放松大脑：你可以尝试进行放松练习，如渐进性肌肉松弛或冥想，以放松你的大脑。
3. 创造良好的睡眠环境：确保你的卧室安静、黑暗、凉爽和舒适。
4. 规律作息：尽量在同一时间入睡和起床，帮助身体建立规律的生物钟。
5. 避免刺激：避免在睡觉前看电子屏幕、吃油腻的食物或喝咖啡因饮料。
6. 远离压力：避免在睡觉前进行紧张的活动，如激烈的运动或激烈的争吵。
7. 睡前放松：在睡觉前进行一些轻松的活动，如阅读或听轻柔的音乐，有助于放松身心。

如果你尝试以上方法仍然睡不着，可以尝试寻求其他帮助，如咨询医生或心理学家。

1.2.2 Web Demo部署

第二种部署的方法是运行web demo。打开web_demo.py 文件，用上面同样的代码将path改好：

```
1 from transformers import AutoModel, AutoTokenizer
2 import gradio as gr
3 import mdtex2html
4 from utils import load_model_on_gpus
5
6 path = "你的文件夹地址"
7 tokenizer = AutoTokenizer.from_pretrained(path, trust_remote_code=True)
8 model = AutoModel.from_pretrained(path, trust_remote_code=True).cuda()
9 model = model.eval()
10
11 """Override Chatbot.postprocess"""
12
13
14 def postprocess(self, y):
15     if y is None:
16         return []
17     for i, (message, response) in enumerate(y):
18         y[i] = (
19             None if message is None else mdtex2html.convert((message)),
20             None if response is None else mdtex2html.convert(response),
21         )
22     return y
```

注意：服务器端口默认在本地进行，如果在服务器上运行或者需要供其他电脑访问的，需要将最后一行代码

```
1 demo.queue().launch(share=False, inbrowser=True)
```

改成

```
1 demo.queue().launch(share=True, inbrowser=True)
```

然后运行web_demo.py 文件，

```
1 (pytorch_p310) sh-4.2$ python web_demo.py
2 Loading checkpoint shards: 100%|
██████████████████████████████████████████████████████████████████████████
██████████████████████████████████████████████████████████████████████████ | 7/7 [00:05<00:00,
1.19it/s]
3 /home/ec2-user/SageMaker/chatglm2-6b/web_demo.py:94: GradDeprecationWarning:
The `style` method is deprecated. Please set these arguments in the
constructor instead.
4 user_input = gr.Textbox(show_label=False, placeholder="Input...",
lines=10).style(
5 #本地访问这个链接
6 Running on local URL: http://127.0.0.1:7860
7 #外网访问这个链接 (72小时过期)
8 Running on public URL: https://88f3b25c8c3f515f77.gradio.live
9
10 This share link expires in 72 hours. For free permanent hosting and GPU
upgrades, run `gradio deploy` from Terminal to deploy to Spaces
(https://huggingface.co/spaces)
```

接下来访问对应链接就可以使用Web Demo啦！



Use via API · Built with Gradio

1.2.3 量化部署

由于我使用的A10,显存充足,所以上述代码都是运行的无量化模型,大约需要占用13G内存。如果你的显存不足,可以根据需要进行如下更改:

```
1 path = "/home/ec2-user/SageMaker/chatglm2-6b"
2 tokenizer = AutoTokenizer.from_pretrained(path, trust_remote_code=True)
3 model = AutoModel.from_pretrained(path, trust_remote_code=True).cuda()
4 #想要INT4量化就填4, INT8量化就填8
5 model = model.quantize(4)
6 model = model.cuda()
7 model = model.eval()
8
9 response, history = model.chat(tokenizer, "你好", history=[])
```

2. 微调ChatGLM2-6B

2.1 数据集选择

智源研究院COIG-PC数据集 <https://huggingface.co/datasets/BAAI/COIG-PC/tree/main/data>

COIG-PC 数据集是精心策划、全面的中文任务和数据集合,旨在促进中文自然语言处理 (NLP) 语言模型的微调和优化。该数据集旨在为研究人员和开发人员提供丰富的资源,以提高语言模型处理中文文本的能力,可用于文本生成、信息提取、情感分析、机器翻译等各个领域。

尽管数据集比较全面,但仍然有许多数据集质量不高,包含大量错误和垃圾内容,经过挑选后我选择了一个质量相对较高的数据集。00169-000-000-tang_poem_continuation。主要内容是古诗的续

写。包含5万6千多条古诗的续写回答。

原数据集格式：

- ```
1 {"instruction": "请根据唐诗的前半部分，补写后半部分。", "input": "度门能不访。冒雪屡西
东。已想人如玉。遥怜马似骢。", "output": "乍迷金谷路。稍变上阳宫。还比相思意。纷纷正满
空。", "split": null, "task_type": {"major": ["文本生成"], "minor": ["古诗词"]},
"domain": ["文学"], "other": null, "task_name_in_eng": "tang_poem_continuation"}
2 {"instruction": "请根据唐诗的前半部分，补写后半部分。", "input": "逍遥东城隅。双树寒葱
蒨。广庭流华月。高阁凝余霰。杜门非养素。抱疾阻良燕。孰谓无他人。", "output": "思君岁云
变。官曹亮先忝。陈躅慙俊彦。岂知晨与夜。相代不相见。缄书问所如。酬藻当芬绚。", "split":
null, "task_type": {"major": ["文本生成"], "minor": ["古诗词"]}, "domain": ["文
学"], "other": null, "task_name_in_eng": "tang_poem_continuation"}
3 ...
```

直接在HF仓库中搜索下载即可。

## 2.2 数据集清理

根据GLM内置的代码和示例，训练数据集需要整理为prompt\_column 和 response\_column 两部分，我将他们命名为：“content” 和 “output”。你也可以自己命名，注意名称一致。

打开ptuning 文件夹里的 tran.sh 文件，将这两行更改：

```
1 --prompt_column content \
2 --response_column output \
3
```

将原数据集清理成这样的格式。将“instruction” 和 “input” 段合并，中间加入“：”，改名为“content”。然后删除“output”之后的内容。由于我的数据集的原格式将所有字典并列，还需要把所有字典放入一个list中。

清洗之后的数据变成了这样：

```
1 [
2 {
3 "content": "请根据唐诗的前半部分，补写后半部分：君归呼。君归兴不孤。",
4 "output": "谢朓澄江今夜月。也应忆著此山夫。"
5 },
6 {
7 "content": "请根据唐诗的前半部分，补写后半部分：为郎复典郡。锦帐映朱轮。露冕随龙
8 "output": "早霜芦叶变。寒雨石榴新。莫怪谪风土。三年作逐臣。"
9 },
10]
```



```
11 ...
12
13]
```

如果你想训练多轮对话，官方也提供了方法：

打开ptuning 文件夹里的 trachat.sh 文件，将这三行更改：

```
1 --prompt_column prompt \
2 --response_column response \
3 --history_column history \
```

这是官方的多轮对话数据集格式

```
1 {"prompt": "长城h3风扇不转。继电器好的。保险丝好的传感器新的风扇也新的这是为什么。就是继电
2 {"prompt": "95", "response": "上下水管温差怎么样啊？空气是不是都排干净了呢？", "histor
3 {"prompt": "是的。上下水管都好的", "response": "那就要检查线路了，一般风扇继电器是由电
```

## 2.3 数据集和训练集的选择

我选择将训练集打乱，从中取出一半，一共2万4千条当做训练集，再取出剩下的一半中的2000条做验证集。分成 train.json 和 test.json 两个文件。

由于生成唐诗的任务相对比较特殊，验证集仅用来判断在遇到训练数据集中没有出现过的唐诗时，模型能否能够生成合理的回答。模型在遇到唐诗时，会自动编写下文，而不会从数据库中寻找原文，所以对验证集进行准确度的分析是没有意义的。

## 2.4 配置训练参数

需要更改GLM2-6B文件中的tran.sh脚本。这是我训练使用的参数：

```
1 PRE_SEQ_LEN=128
2 LR=2e-2
3
4 #调整使用的GPU数量和编号
5 NUM_GPUS=2
6 export CUDA_VISIBLE_DEVICES=2,3
7
8 torchrun --standalone --nnodes=1 --nproc-per-node=$NUM_GPUS main.py \
9 --do_train \
10 --train_file train.json \
11 --validation_file train.json \
```

```

12 --preprocessing_num_workers 10 \
13 --prompt_column content \
14 --response_column output \
15 --overwrite_cache \
16 --model_name_or_path /home/ec2-user/SageMaker/chatglm2-6b \
17 --output_dir output/adgen-chatglm2-6b-pt5-PRE_SEQ_LEN-LR \
18 --overwrite_output_dir \
19 --max_source_length 64 \
20 --max_target_length 128 \
21 --per_device_train_batch_size 16 \
22 --per_device_eval_batch_size 1 \
23 --gradient_accumulation_steps 1 \
24 --predict_with_generate \
25 --max_steps 200 \
26 --logging_steps 10 \
27 --save_steps 25 \
28 --learning_rate $LR \
29 --pre_seq_len $PRE_SEQ_LEN \
30
31 #进行量化处理
32 --quantization_bit 4 \

```

修改 `train.sh` 中的 `train_file`、`validation_file` 和 `test_file` 为你自己的 JSON 格式数据集路径，并将 `prompt_column` 和 `response_column` 改为 JSON 文件中输入文本和输出文本对应的 KEY。可能还需要增大 `max_source_length` 和 `max_target_length` 来匹配你自己的数据集中的最大输入输出长度。并将模型路径 `THUDM/chatglm-6b` 改为你本地的模型路径。

如果你需要对验证集进行准确度评估，还需要对 `evaluate.sh` 进行相同的修改

`train.sh` 中的 `PRE_SEQ_LEN` 和 `LR` 分别是 soft prompt 长度和训练的学习率，可以进行调节以取得最佳的效果。我这里的 learning rate 采用的是  $2e-2$ ，相对来说是比较大的，可以避免找到局部最优解，但是会损失精度。

P-Tuning-v2 方法会冻结全部的模型参数，可通过调整 `quantization_bit` 来被原始模型的量化等级，不加此选项则为 FP16 精度加载。

在默认配置 `quantization_bit=4`、`per_device_train_batch_size=1`、`gradient_accumulation_steps=16` 下，INT4 的模型参数被冻结，一次训练迭代会以 1 的批处理大小进行 16 次累加的前后向传播，等效为 16 的总批处理大小，此时最低只需 6.7G 显存。若想在同等批处理大小下提升训练效率，可在二者乘积不变的情况下，加大 `per_device_train_batch_size` 的值，但也会带来更多的显存消耗，请根据实际情况酌情调整。

由于我的显存充足，所以我选择加大 `per_device_train_batch_size` 的值至16，同时降低 `gradient_accumulation_steps` 为1保持乘积，提高计算速度。此时我拿出的两块A10每块占用约14G显存。

`max_steps` 为训练的总步数，`save_steps` 为训练多少步保存一次检查点。经过多次实验，我发现模型在步数增大后会发生严重的遗忘现象，所以把Steps调低至200次，每25次保存一次。

`output_dir` 之后填写微调后模型的保存位置。

## 2.5 开始训练

一切配置完成后，执行`bash train.sh` 命令：

我的配置进行150步的训练只需要大概5-10分钟。

```
nsformers_modules.chatglm2-6b.modeling_chatglm - `use_cache=True` is incompatible with gradient checkpointing. Setting `use_cache=False`...
08/16/2023 08:52:03 - WARNING - transformers_modules.chatglm2-6b.modeling_chatglm - `use_cache=True` is incompatible with gradient checkpointing. Settin
g `use_cache=False`...
{'loss': 5.7582, 'learning_rate': 0.019, 'epoch': 0.01}
{'loss': 5.0518, 'learning_rate': 0.018000000000000002, 'epoch': 0.02}
{'loss': 4.9363, 'learning_rate': 0.017, 'epoch': 0.03}
{'loss': 5.0016, 'learning_rate': 0.016, 'epoch': 0.05}
{'loss': 4.9934, 'learning_rate': 0.015, 'epoch': 0.06}
25%|██████████ | 50/200 [02:00<05:59, 2.40s/it]Saving PrefixEncoder
```

训练完成后，打开文件夹就可以看到每个checkpoint和训练的具体参数。

/ ... / output / adgen-chatglm2-6b-pt3-128-2e-3 /

| Name               | Last Modified |
|--------------------|---------------|
| checkpoint-100     | a day ago     |
| checkpoint-125     | a day ago     |
| checkpoint-150     | a day ago     |
| checkpoint-25      | a day ago     |
| checkpoint-50      | a day ago     |
| checkpoint-75      | a day ago     |
| all_results.json   | a day ago     |
| train_results.json | a day ago     |
| trainer_state.json | a day ago     |

```
▼ root:
 epoch: 2.68
 train_loss: 3.0548795572916667
 train_runtime: 450.6916
 train_samples: 1764
 train_samples_per_second: 10.65
 train_steps_per_second: 0.333
```

Epoch代表训练次数，可以看出我一共对训练集训练了2.68遍。误差为3.05。虽然这个误差没有降的很小，但是为了避免致命的过拟合，这样一个误差也能使我们相对安全地完成目标了。

## 2.6 模型推理

按照如下代码加载微调后的模型：

```
1 #原模型地址
2 path = "/home/ec2-user/SageMaker/chatglm2-6b"
3 tokenizer = AutoTokenizer.from_pretrained(path, trust_remote_code=True)
4 config = AutoConfig.from_pretrained(path, trust_remote_code=True, pre_seq_len=12)
5 model = AutoModel.from_pretrained(path, trust_remote_code=True, config=config, de
6 #微调后的checkpoint地址
7 prefix_state_dict = torch.load(os.path.join("/home/ec2-user/SageMaker/chatglm2-6
8 new_prefix_state_dict = {}
9 for k, v in prefix_state_dict.items():
10 if k.startswith("transformer.prefix_encoder."):
11 new_prefix_state_dict[k[len("transformer.prefix_encoder."):]] = v
12 model.transformer.prefix_encoder.load_state_dict(new_prefix_state_dict)
13 model = model.cuda()
14 model = model.eval()
```

微调前后效果对比

原模型回答：

```
[4]: #训练集测试
response1, history1 = model1.chat(tokenizer, "请根据唐诗的前半部分，补写后半部分：眠云喜道存，读易过朝昏。乔木青连郭，长河白泻门。", history=[])
print(response1)

这是一首唐代诗人王之涣的《登鹳雀楼》诗。前半部分已经给出，下半部分如下：

白日依山尽。黄河入海流。欲穷千里目。更上一层楼。

这首诗是王之涣登上鹳雀楼后所写，表达了他对逝去岁月和人生的感慨和对未来的期许。

[12]: #测试集测试
response1, history1 = model1.chat(tokenizer, "请根据唐诗的前半部分，补写后半部分：闻道并州镇，尚书训士齐。几时通蓟北。当日报关西。", history=[])
print(response1)

闻道并州镇，尚书训士齐。
几时通蓟北，当日报关西。

这首诗是唐代诗人王之涣所作的《登鹳雀楼》的前半部分。后半部分的内容是：

闻道并州镇，尚书训士齐。
几时通蓟北，当日报关西。

并州是唐代时的一个地名，现在可能已经没有这个地名了。在这首诗中，诗人表达了自己对于并州的向往和期望。尚书训士齐是指朝廷中的官员，他们正在推行一些有益于国家的政策。几时通蓟北，当日报关西则表达了诗人对于蓟北和关西等地方的向往和期望。蓟北和关西都是唐代时的地名，现在可能已经没有这些地方了。
```

```
[13]: #测试集测试
response, history1 = model1.chat(tokenizer, "请根据唐诗的前半部分, 补写后半部分: 一曲狂歌酒百分。蛾眉画出月争新。", history=[])
print(response)
```

分析和解答:

这句话是唐代诗人李白的《将进酒》中的前半部分, 描述的是酒醉的情景和月光下的美景。而后半部分则是描述了醉酒之后的状态和醉酒者的内心感受。

具体来说, 这句话中的“蛾眉画出月争新”是在形容月光下酒醉者的脸部表情, 用蛾眉来比喻, 表现出酒醉者的神态和气质。而“一百分”则是形容酒醉的程度, 用百分比来表示, 表现出醉酒者的醉酒程度。

综上所述, 这句话的后半部分是描述了醉酒者的内心感受和醉酒程度, 用“蛾眉画出月争新”来形容酒醉者的神态和气质, 用百分比来表示醉酒的程度。

可以看到, 原模型似乎会参考自己的训练材料, 然后直接回答原文, 但因为提问的古诗并不在它的训练材料里, 所以它的回答明显是答非所问。

微调后模型回答:

```
[11]: response, history = model.chat(tokenizer, "回答问题: 某班有男生15人, 女生27人, 男生占女生总人数的几分之几?", history=[])
print(response)
```

首先, 我们需要计算出男女生人数之和, 即 $15+27=42$ 。然后, 我们用男生人数除以总人数, 即 $15/42=1/3$ 。因此, 男生占女生总人数的 $1/3$ 。

```
[12]: response, history = model.chat(tokenizer, "1+1等于几?", history=[])
print(response)
```

1+1=2

```
[13]: model.chat(tokenizer, "请根据唐诗的前半部分, 补写后半部分: 梁苑城西薜萝头。玉鞭公子醉风流。几多红粉低鬟恨。一部清商驻拍留。", history=[])
print(response)
```

风起云飞千万里。月明江月万重秋。愿君千万相留恋。相留恋处更无愁。

```
[14]: response, history = model.chat(tokenizer, "请根据唐诗的前半部分, 补写后半部分: 眠云喜真存。读易过朝昏。乔木青连郭。长河白泻门。", history=[])
print(response)
```

春生千树绿。月落万山纹。更信仙游仙。空闻石溜声。

```
[27]: response, history = model.chat(tokenizer, "请根据唐诗的前半部分, 补写后半部分: 闻滇并州镇。尚书训士齐。几时通蜀北。当日报关西。", history=[])
print(response)
```

塞上闻鸡早。关中送日迟。平明闻塞声。应道有佳期。

```
[16]: #测试集测试
response, history = model.chat(tokenizer, "请根据唐诗的前半部分, 补写后半部分: 一曲狂歌酒百分。蛾眉画出月争新。", history=[])
print(response)
```

今宵月半思君意。尽是花中肠断人。

微调后的模型在生成古诗的能力上有了很大的提升, 它不再是从数据库中寻找, 而是直接根据上文编写下文, 尽管有些诗歌的生成效果可能不好, 但是总体来看生成的诗句比较符合上文的意境, 而且诗句能做到一些押韵。

```
[13]: model.chat(tokenizer, "请根据唐诗的前半部分，补写后半部分：梁苑城西薜萝头。玉鞭公子醉风流。几多红粉低鬟恨。一部清商驻拍留。", _history=[])

燮子长将燮子短。黄冠不著黄冠人。清时斗酒斗清人。笑看汉官争白头。

[14]: response, history = model.chat(tokenizer, "请根据唐诗的前半部分，补写后半部分：眠云喜道存。读易过朝昏。乔木青连郭。长河白泻门。", _history=[])
response)

愿君知此乐。不似采莲人。更作南村客。归家更少村。

[15]: response, history = model.chat(tokenizer, "请根据唐诗的前半部分，补写后半部分：闻道并州镇。尚书训士齐。几时通蓟北。当日报关西。", _history=[])
response)

独抱真经去。无将俗物齐。功成方就归。笑傲边城时。

[16]: # 测试集测试
response, history = model.chat(tokenizer, "请根据唐诗的前半部分，补写后半部分：一曲狂歌酒百分。蛾眉画出月争新。", _history=[])
print(response)

美人一笑花千面。不笑君来见罗频。
```

美中不足的是，尽管已经做了很多措施来防止模型出现知识遗忘的现象，在面对训练集中没有出现的知识时，相对于原模型，微调后的模型的性能仍然会下降。