GLM4部署微调实战

石墨文档链接：https://shimo.im/docs/rp3OMWyp7aCdY5km

GLM家族史

论文地址：https://arxiv.org/pdf/2406.12793
GitHub： https://github.com/THUDM
HF： https://huggingface.co/THUDM

GLM-4-9B 模型具备更强大的推理性能、更长的上下文处理能力、多语言、多模态和 All Tools 等能力。包括基础版本 GLM-4-9B（8K）、对话版本 GLM-4-9B-Chat(128K)、超长上下文版本 GLM-4-9B-Chat-1M(1M)和多模态版本 GLM-4V-9B-Chat(8K)。

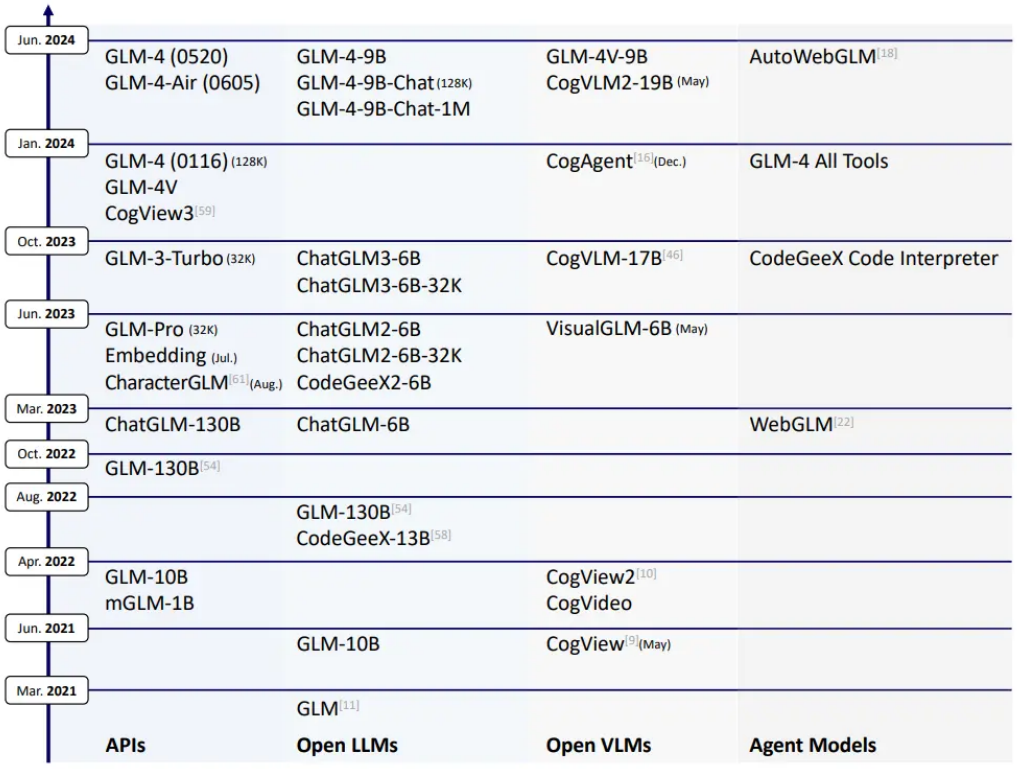
GLM-4-9B（8K）：基座模型，不具备对话能力，支持8K上下文；
GLM-4-9B-Chat（128K）：对话模型，具备工具调用能力，支持长度128K；
GLM-4-9B-Chat-1M(1M)：具备长文本能力的对话模型，具备工具调用能力，支持1M上下文；
GLM-4V-9B-Chat（8K）：视觉模型，支持8K上下文；

GLM-4-9B 的能力包括：
1. 基础能力:模型中英文综合性能比 ChatGLM3-6B 提升了40%;
2. 长文本能力:上下文从128K 扩展到1M tokens;
3. 多语言能力:支持26种语言，词表大小扩充到150k，编码效率提高30%;
4. Function Call 能力:在 Berkeley Function-Calling Leaderboard 上表现优秀;
5. All Tools 能力:模型可以使用外部工具完成任务;
6. 多模态能力:首次推出多模态模型，性能显著。

GLM-4 模型在 10 万亿个中文和英文的token以及来自 24 种语言的一小部分语料库上进行了预训练，并且主要针对中文和英文使用进行了对齐。高质量的对齐是通过多阶段的后训练过程实现的，该过程涉及监督微调和从人类反馈中学习。评估表明：GLM-4
1）在 MMLU、GSM8K、MATH、BBH、GPQA 和 HumanEval 等一般指标方面与 GPT-4 密切相关或优于 GPT-4；

2）在指令遵循方面接近 GPT-4-Turbo（根据IFEval指标评估）；
3）与GPT-4 Turbo （128K） 和 Claude 3 在处理长上下文任务的能力接近；
4）在中文对齐方面，GLM-4优于GPT-4（根据AlignBench指标评估）。
经过进一步优化的GLM-4 All Tools模型能够理解用户的意图，并自主决定何时以及如何使用包括网络浏览器、Python解释器、文本到图像模型以及用户自定义函数在内的工具来高效完成复杂任务。

2021年3月，GLM架构的模型首次被提出；
2021年6月，GLM-10B被提出；
2022年8月，GLM-130B 和 CodeGeeX-13B被提出；
2023年3月，ChatGLM-130B、ChatGLM-6B同日发布；
2023年6月，ChatGLM2-6B发布；
2023年10月，ChatGLM3-6B发布；
2024年1月，GLM-4发布；
2024年6月，GLM-4-9B发布；



ChatGLM-6B 在大约 1 万亿个中英文语料库标记上进行了预训练，上下文长度为 2048。

ChatGLM2-6B 于 6 月发布，通过采用FlashAttention 技术，其上下文长度扩展到 32K。此外，Multi-Query Attention 的集成有助于将推理速度提高 42%。

ChatGLM3-6B 通过进一步适配更多样化的训练数据集、更充足的训练步骤和更优化的训练策略，在语义、数学、推理、代码和知识方面突破了 42 项基准测试。从这一代开始，ChatGLM 还支持**函数调用和代码解释器，以及复杂的代理任务**。

GLM-4：团队凭借所有的经验教训和积累，开始了 GLM-4 的训练。然后，第一个截止检查点经历了一个多阶段的训练后过程（例如 SFT、RLHF、安全对齐），侧重于中文和英文。随后，它被开发成两个不同的版本：GLM-4 和 GLM-4 所有工具，都支持 128K 上下文长度。GLM-4-Air 的性能与 GLM-4（0116） 相当，但延迟和推理成本更低。

2024 年6月公开发布了 GLM-4-9B（128K 和 1M 上下文长度）模型。GLM-4-9B 在大约 10 万亿个多语言语料库标记上进行了预训练，上下文长度为 8192（8K），并使用与 GLM-4（0520） 相同的管道和数据进行后训练。GLM-4-9B 及其人类偏好对齐的版本 GLM-4-9B-Chat 均表现出超越 Llama-3-8B 的卓越性能。另外还推出了基于 GLM-4-9B 的多模态模型 GLM-4V-9B。

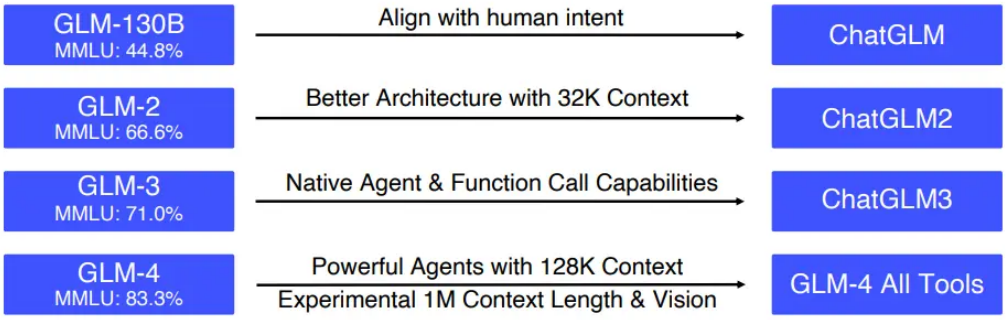


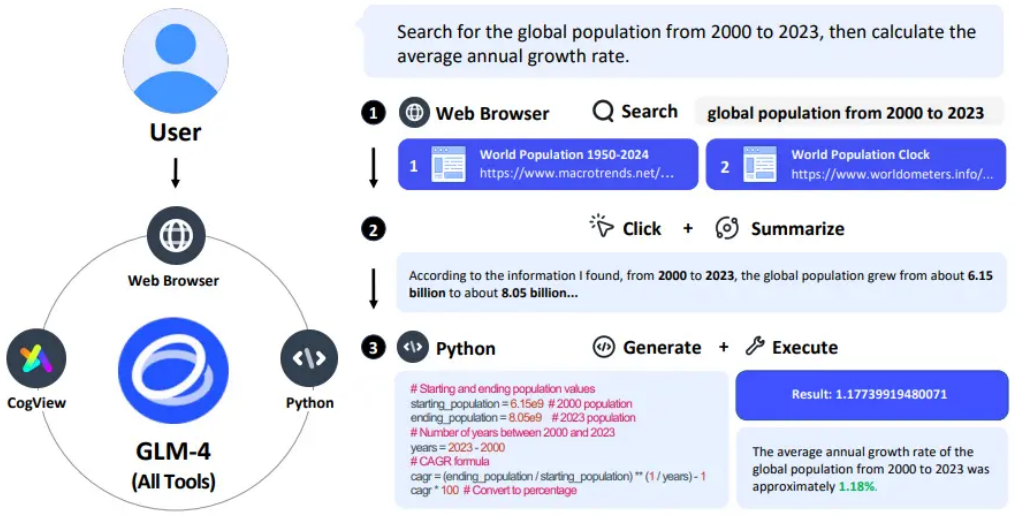
Table 1: Performance of Open ChatGLM-6B, ChatGLM2-6B, ChatGLM3-6B, and GLM-4-9B.

Language	Dataset	ChatGLM-6B (2023-03-14)	ChatGLM2-6B (2023-06-25)	ChatGLM3-6B-Base (2023-10-27)	GLM-4-9B (2024-06-05)
English	GSM8K	1.5	25.9	72.3	84.0
	MATH	3.1	6.9	25.7	30.4
	BBH	0.0	29.2	66.1	76.3
	MMLU	25.2	45.2	61.4	74.7
	GPQA	-	-	26.8	34.3
	HumanEval	0.0	9.8	58.5	70.1
	BoolQ	51.8	79.0	87.9	89.6
	CommonSenseQA	20.5	65.4	86.5	90.7
	HellaSwag	30.4	57.0	79.7	82.6
	PIQA	65.7	69.6	80.1	79.1
	DROP	3.9	25.6	70.9	77.2
Chinese	C-Eval	23.7	51.7	69.0	77.1
	CMMLU	25.3	50.0	67.5	75.1
	GAOKAO-Bench	26.8	46.4	67.3	74.5
	C3	35.1	58.6	73.9	77.2

GLM-4 All Tools

最新的 ChatGLM 模型系列包括 GLM-4 和 GLM-4 All Tools，这两个模型都是采用先进技术进行训练和对齐的。GLM-4 All Tools 是专门为支持智能体和相关任务而进一步优化的模型版本。它能够自主理解用户的意图，规划复杂的指令，并能够调用一个或多个工具（例如网络浏览器、Python解释器和文本到图像模型）以完成复杂的任务。

下图显示了GLM-4 All Tools系统的整体工作流程，当用户提出复杂请求时，模型会对任务进行分析，并逐步规划解决问题的过程。如果模型确定无法独立完成任务，它将依次调用一个或多个外部工具，利用它们的中间反馈和结果来协助解决任务。带有 Web 浏览器和 Python 解释器的 GLM-4 All Tools系统解决用户查询“搜索 2000 年至 2023 年的全球人口，然后计算平均年增长率”。



推理

所有测试均在单张GPU上进行测试,所有显存消耗都按照峰值左右进行测算

相关推理的压力测试数据如下：

GLM-4-9B-Chat

精度	显存占用	Prefilling	Decode Speed	Remarks
BF16	19 GB	0.2s	27.8 tokens/s	输入长度为 1000
BF16	21 GB	0.8s	31.8 tokens/s	输入长度为 8000
BF16	28 GB	4.3s	14.4 tokens/s	输入长度为 32000
BF16	58 GB	38.1s	3.4 tokens/s	输入长度为 128000

精度	显存占用	Prefilling	Decode Speed	Remarks
INT4	8 GB	0.2s	23.3 tokens/s	输入长度为 1000
INT4	10 GB	0.8s	23.4 tokens/s	输入长度为 8000
INT4	17 GB	4.3s	14.6 tokens/s	输入长度为 32000

prefilling指的是预填充阶段，是指模型在收到输入序列后，生成第一个输出词之前的计算过程。这个过程通过做一些预处理操作，以便在时间推理阶段能够更快速地相应请求。

GLM-4-9B-Chat-1M

精度	显存占用	Prefilling	Decode Speed	Remarks
BF16	75 GB	98.4s	2.3 tokens/s	输入长度为 200000

如果您的输入超过200K，建议您使用vLLM后端进行多卡推理，以获得更好的性能。

vllm部署

vLLM 框架是一个高效的大型语言模型（LLM）推理和部署服务系统，具备以下特性：

- **高效的内存管理**：通过 PagedAttention 算法，vLLM 实现了对 KV 缓存的高效管理，减少了内存浪费，优化了模型的运行效率。
- **高吞吐量**：vLLM 支持异步处理和连续批处理请求，显著提高了模型推理的吞吐量，加速了文本生成和处理速度。
- **易用性**：vLLM 与 HuggingFace 模型无缝集成，支持多种流行的大型语言模型，简化了模型部署和推理的过程。兼容 OpenAI 的 API 服务器。
- **分布式推理**：框架支持在多 GPU 环境中进行分布式推理，通过模型并行策略和高效的数据通信，提升了处理大型模型的能力。
- **开源**：vLLM 是开源的，拥有活跃的社区支持，便于开发者贡献和改进，共同推动技术发展。

使用 transformers 后端进行推理

```
import torch
from transformers import AutoModelForCausalLM, AutoTokenizer
import os

os.environ['CUDA_VISIBLE_DEVICES'] = '0' # 设置 GPU 编号，如果单机单卡指定一个
MODEL_PATH = "THUDM/glm-4-9b-chat"

device = "cuda" if torch.cuda.is_available() else "cpu"

tokenizer = AutoTokenizer.from_pretrained(MODEL_PATH, trust_remote_code=

query = "你好"

inputs = tokenizer.apply_chat_template([{"role": "user", "content": quer
                                        add_generation_prompt=True,
                                        tokenize=True,
                                        return_tensors="pt",
                                        return_dict=True
                                        )

inputs = inputs.to(device)
model = AutoModelForCausalLM.from_pretrained(
    MODEL_PATH,
    torch_dtype=torch.bfloat16,
    low_cpu_mem_usage=True,
    trust_remote_code=True,
    device_map="auto"
).eval()

gen_kwargs = {"max_length": 2500, "do_sample": True, "top_k": 1}
with torch.no_grad():
    outputs = model.generate(**inputs, **gen_kwargs)
    outputs = outputs[:, inputs['input_ids'].shape[1]:]
    print(tokenizer.decode(outputs[0], skip_special_tokens=True))

```

使用 vLLM 后端进行推理

从 vLLM 库中导入 LLM 和 SamplingParams 类。 `LLM` 类是使用 vLLM 引擎运行离线推理的主要类。 `SamplingParams` 类指定采样过程的参数，用于控制和调整生成文本的随机性和多样性。

```
from transformers import AutoTokenizer
from vllm import LLM, SamplingParams

# GLM-4-9B-Chat-1M
# max_model_len, tp_size = 1048576, 4
# 如果遇见 OOM 现象，建议减少max_model_len，或者增加tp_size
max_model_len, tp_size = 131072, 1
model_name = "THUDM/glm-4-9b-chat"
prompt = [{"role": "user", "content": "你好"}]

tokenizer = AutoTokenizer.from_pretrained(model_name, trust_remote_code=
llm = LLM(
    model=model_name,
    tensor_parallel_size=tp_size,
    max_model_len=max_model_len,
    trust_remote_code=True,
    enforce_eager=True,
    # GLM-4-9B-Chat-1M 如果遇见 OOM 现象，建议开启下述参数
    # enable_chunked_prefill=True,
    # max_num_batched_tokens=8192
)
stop_token_ids = [151329, 151336, 151338]
sampling_params = SamplingParams(temperature=0.95, max_tokens=1024, stop

inputs = tokenizer.apply_chat_template(prompt, tokenize=False, add_gener
outputs = llm.generate(prompts=inputs, sampling_params=sampling_params)

print(outputs[0].outputs[0].text)
```

建议如果本地没有机器，可以再autodl上租一台24G显存的机器：https://www.autodl.com/market/list

1. 下载模型仓库及配置环境

开启学术加速

```
source /etc/network_turbo
```

下载代码仓库

```
git clone https://github.com/THUDM/GLM-4.git
```

安装vllm

```
pip install vllm==0.6.0
```

如果vllm安装出现版本问题，可以通过下面命令安装指定版本的vllm

```
export VLLM_VERSION=0.6.0
export PYTHON_VERSION=312
pip install https://github.com/vllm-project/vllm/releases/download/v${VL
```

2. 下载模型

安装modelscope

```
pip install modelscope
```

创建一个model_download.py的文件，用python命令运行

```
from modelscope import snapshot_download, AutoModel, AutoTokenizer
osmodel_dir = snapshot_download('ZhipuAI/glm-4-9b-chat', cache_dir='/roo
```

vLLM 部署实现 OpenAI API 协议的服务器非常方便。默认会在 http://localhost:8000 启动服务器。服务器当前一次托管一个模型，并实现列表模型、completions 和 chat completions 端口。

- completions：是基本的文本生成任务，模型会在给定的提示后生成一段文本。这种类型的任务通常用于生成文章、故事、邮件等。
- chat completions：是面向对话的任务，模型需要理解和生成对话。这种类型的任务通常用于构建聊天机器人或者对话系统。

在创建服务器时，我们可以指定模型名称、模型路径、聊天模板等参数。

- --host 和 --port 参数指定地址。
- --model 参数指定模型名称。
- --chat-template 参数指定聊天模板。
- --served-model-name 指定服务模型的名称。
- --max-model-len 指定模型的最大长度。

```
python -m vllm.entrypoints.openai.api_server --model /root/autodl-tmp/Zh
```

1. 通过 curl 命令查看当前的模型列表。

```
curl http://localhost:8000/v1/models
```

得到的返回值如下所示：

```
{"object":"list","data":[{"id":"glm-4-9b-chat","object":"model","created
```

1. 使用 curl 命令测试 OpenAI Completions API 。

```
curl http://localhost:8000/v1/completions \
    -H "Content-Type: application/json" \
    -d '{
            "model": "glm-4-9b-chat",
            "prompt": "你好",
            "max_tokens": 7,
            "temperature": 0
        }'
```

得到的返回值如下所示：

```
{"id":"cmpl-8bba2df7cfa1400da705c58946389cc1","object":"text_completion"
```

也可以用 python 脚本请求 OpenAI Completions API 。这里面设置了额外参数 `extra_body`，我们传入了 `stop_token_ids` 停止词 id。当 openai api 无法满足时可以采用 vllm 官方文档方式添加。https://docs.vllm.ai/en/latest/serving/openai_compatible_server.html

```
from openai import OpenAI
client = OpenAI(
    base_url="http://localhost:8000/v1",
    api_key="token-abc123", # 随便设，只是为了通过接口参数校验
)

completion = client.chat.completions.create(
  model="glm-4-9b-chat",
  messages=[
    {"role": "user", "content": "你好"}
  ],
  # 设置额外参数
  extra_body={
    "stop_token_ids": [151329, 151336, 151338]
  }
)

print(completion.choices[0].message)
```

得到的返回值如下所示：

```
ChatCompletionMessage(content='\n你好👋！很高兴见到你，有什么可以帮助你的吗？',
```

1. 用 curl 命令测试 OpenAI Chat Completions API 。

```
curl http://localhost:8000/v1/chat/completions \
    -H "Content-Type: application/json" \
    -d '{
            "model": "glm-4-9b-chat",
            "messages": [
                  {"role": "system", "content": "You are a helpful assis
                  {"role": "user", "content": "你好"}
            ],
            "max_tokens": 7,
            "temperature": 0

        }'
```

得到的返回值如下所示：

```
{"id":"cmpl-8b02ae787c7747ecaf1fb6f72144b798","object":"chat.completion"
```

也可以用 python 脚本请求 OpenAI Chat Completions API 。

```
from openai import OpenAI
openai_api_key = "EMPTY" # 随便设，只是为了通过接口参数校验
openai_api_base = "http://localhost:8000/v1"client = OpenAI(
    api_key=openai_api_key,
    base_url=openai_api_base,
)

chat_outputs = client.chat.completions.create(
    model="glm-4-9b-chat",
    messages=[
        {"role": "system", "content": "You are a helpful assistant."},
        {"role": "user", "content": "你好"},
    ],
      # 设置额外参数
      extra_body={
        "stop_token_ids": [151329, 151336, 151338]
  }
)
print(chat_outputs)
```

得到的返回值如下所示：

```
ChatCompletion(id='cmpl-16b1c36dc695426cacee23b79d179d52', choices=[Choi
```

在处理请求时 API 后端也会打印一些日志和统计信息。

基于llama-factory微调

1、新建一个conda环境并激活

```
source activate
conda create -n llama-factory python==3.12
conda activate llama-factory
```

2、下载代码仓库并配置环境

```
source /etc/network_turbo
git clone https://github.com/hiyouga/LLaMA-Factory.git
cd LLaMA-Factory
pip install -e .[torch,metrics]
```

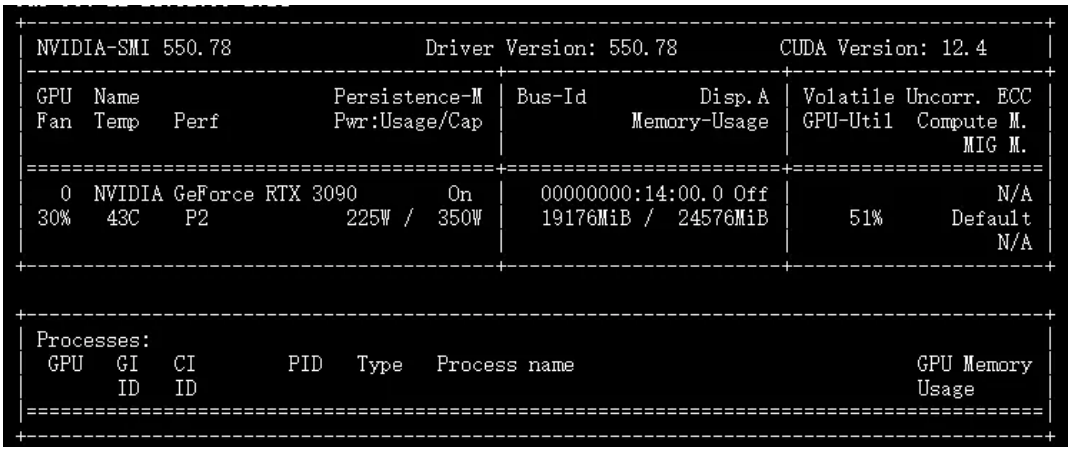
3、下载模型
模型在前面已下载好

4、修改数据格式

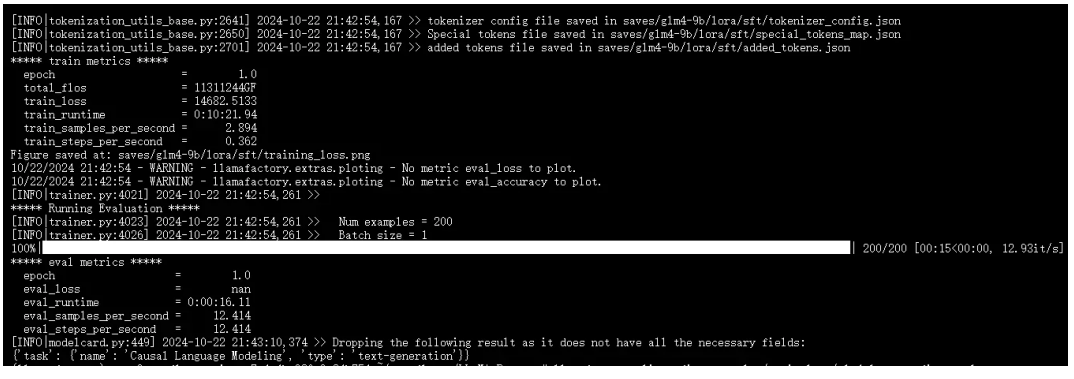
```
  "adv_train": {
    "file_name": "adv_train.json",
      "columns": {
      "prompt": "instruction",
      "query": "input",
      "response": "output",
    }
  },
  "adv_dev": {
    "file_name": "adv_dev.json",
      "columns": {
      "prompt": "instruction",
      "query": "input",
      "response": "output",
    }
```

5、修改配置文件并执行命令

```
llamafactory-cli train examples/train_lora/glm4_lora_sft.yaml
```



运行完成



6、推理

修改配置文件并执行命令

```
llamafactory-cli train examples/train_lora/glm4_lora_predict.yaml
```