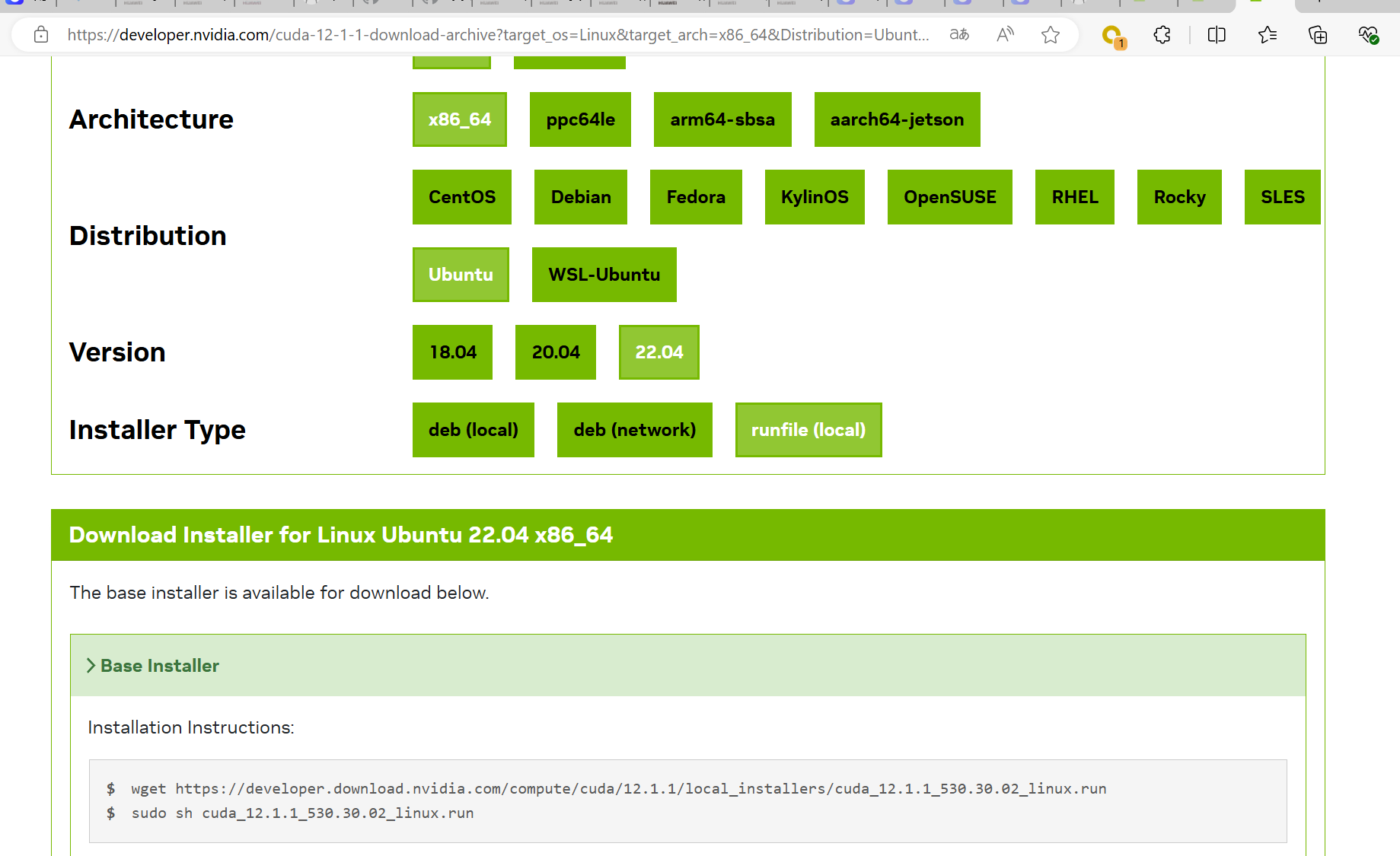
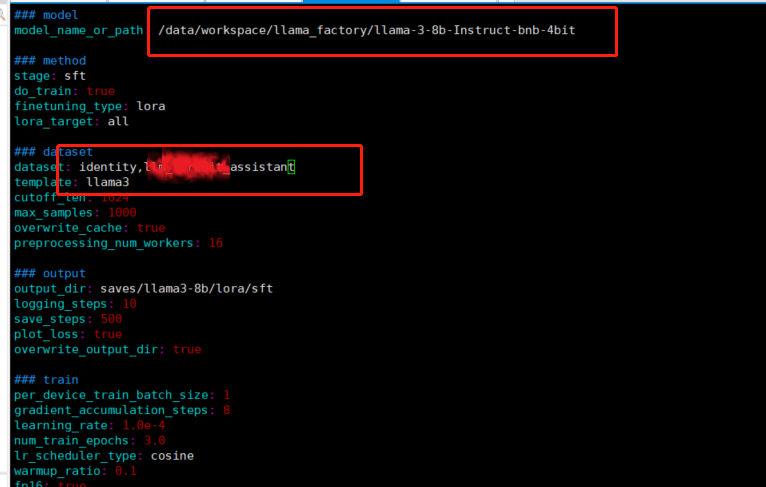
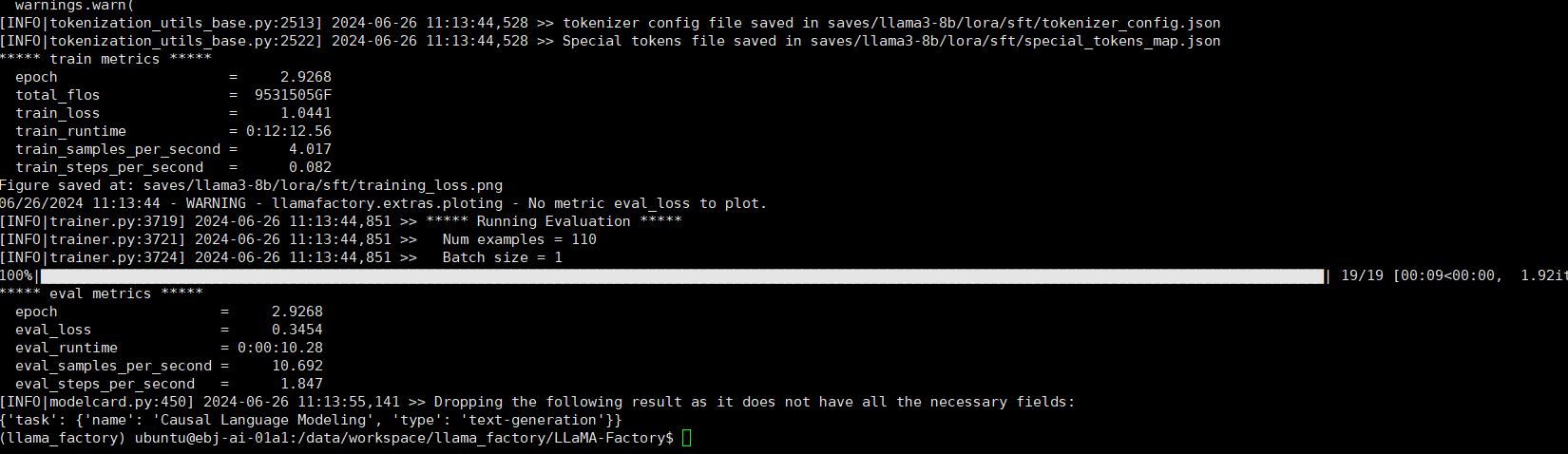
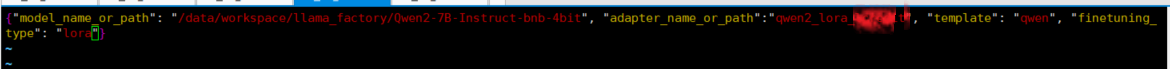
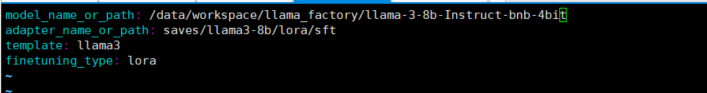
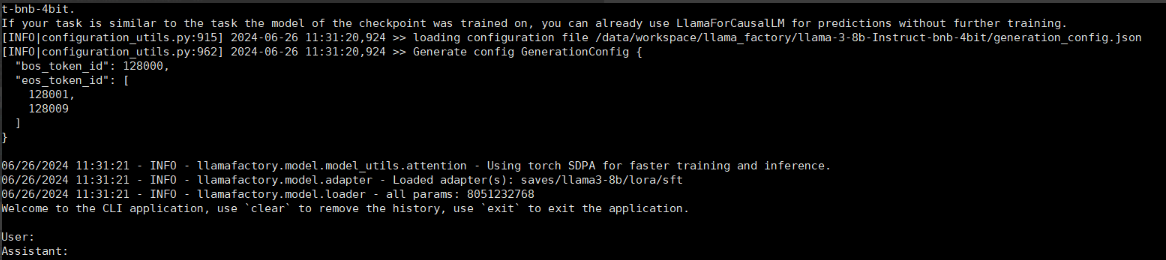
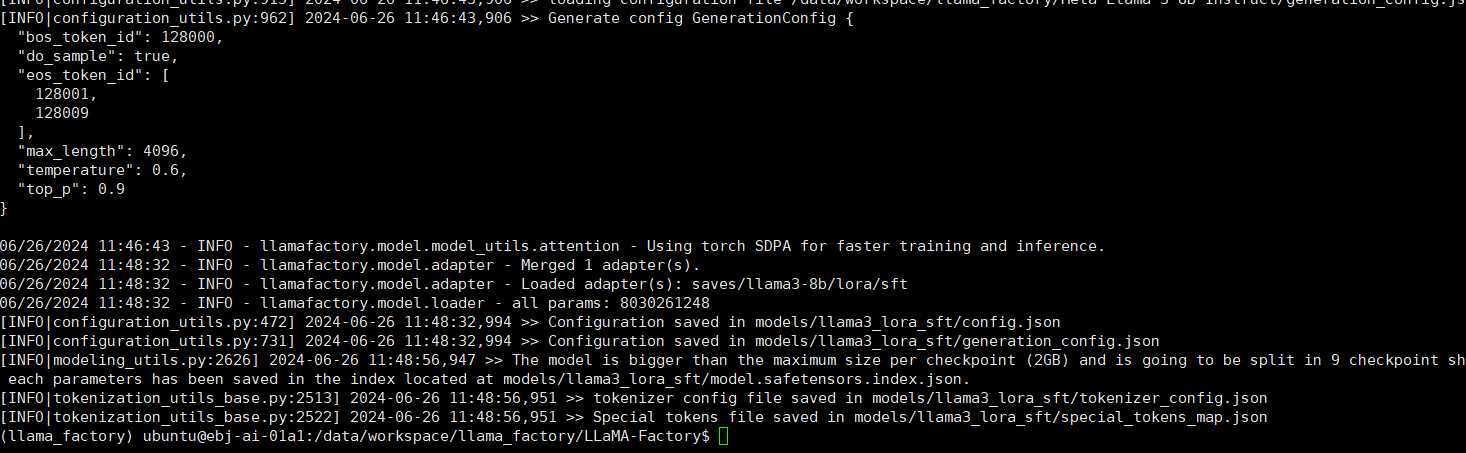
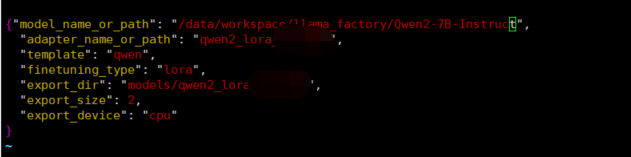
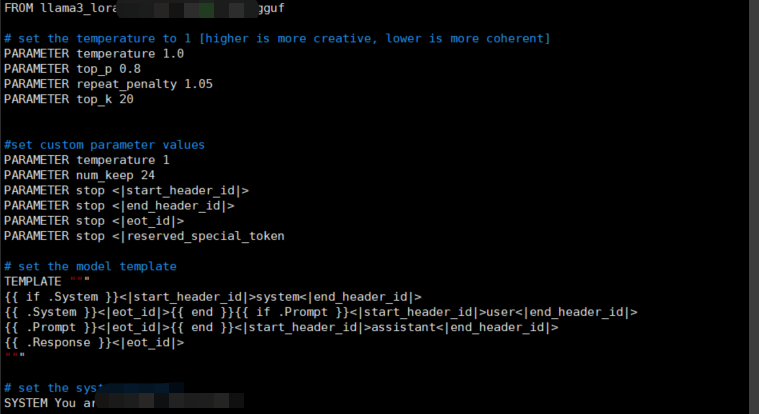
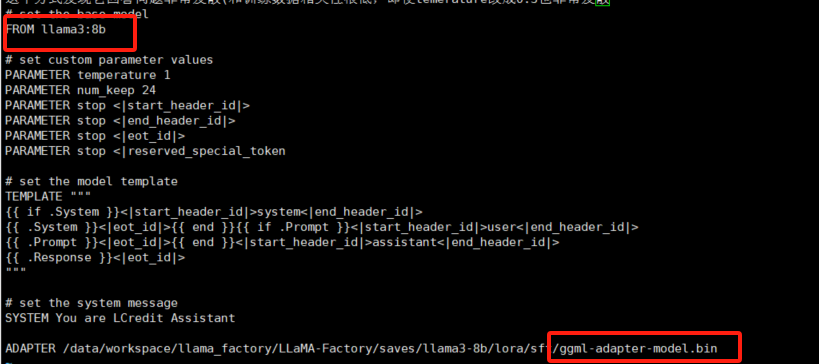
# [大模型微调环境准备及步骤](https://doc.zz.com/pages/viewpage.action?pageId=83952168)

本文是使用LLaMA-Factory对模型进行微调相关的jiaocheng（踩坑记录），示例中分别基于qwen2 7B和llama3 8B，结合LCredit客户数据对模型进行了微调，并且将微调后的模型导入ollama，供接口或web ui调用。（环境和工具可能一直在变化，这份文档到2024/06/27是ok的）。

模型微调和RAAG都可以将“私有”知识应用于大模型，它们的各自的特点如下：



1. 环境准备
2. 系统环境：使用cuda 12.1版本（最新的12.4版本还有很多库没有适配）。安装方式在NVIDIA网站有说明。
3. python 使用conda新建一个干净的环境。注意需要干净的，避免后续装很多其他包的依赖冲突
4. llama factory相关的配置。可以直接看 <https://github.com/hiyouga/LLaMA-Factory> 步骤介绍比较详细。
5. 下载llama工具，地址<https://github.com/ggerganov/llama.cpp.git>
6. 基础模型准备，因为本地的硬件配置不是很高。训练的base采用4bit量化后的版本，但是训练结果合并时仍需要原始的模型。所以qwen2和llama3都需要量化和原始版都clone一份。相关的地址可以在hugingface或modelscope上找到：分别是<https://huggingface.co/unsloth/llama-3-8b-Instruct-bnb-4bit> ，<https://www.modelscope.cn/LLM-Research/Meta-Llama-3-8B-Instruct.git>  
   <https://huggingface.co/unsloth/Qwen2-7B-Instruct-bnb-4bit> ，<https://www.modelscope.cn/qwen/Qwen2-7B-Instruct.git>  
   这些clone下来后就可以准备模型微调了。
7. 训练步骤
   1. 准备数据，格式可以在LLama Factory的example示例找到
   2. 修改LLaMA-Factory/data目录下，并且修改dataset\_info.json（改json里的数据名称将会在模型训练时的参数中使用）
   3. 对于llama3模型训练，修改examples/train\_lora/llama3\_lora\_sft.yaml (model\_name\_or\_path是模型微调的base，也就是上面的量化过后的基础模型）  
      对qwen2模型，新建一个qwen2\_7b\_sft\_lcredit.json文件放到examples/train\_lora/目录
   4. 执行命令CUDA\_VISIBLE\_DEVICES=0,1,2,3,4,5 llamafactory-cli train examples/train\_lora/llama3\_lora\_sft.yaml，命令执行完成后如图（CUDA\_VISIBLE\_DEVICES表示nvidia-smi列出的各显卡编号）.如果训练qwen模型则将对应的yaml改成对应的json文件
   5. 修改examples/inference/llama3\_lora\_sft.yaml (model\_name\_or\_path是模型微调的base，adapter\_name\_or\_path是训练步骤的输出adapter）。对于qwen模型，则是新增qwen2\_7b\_sft\_lcredit.json文件。
   6. 执行命令CUDA\_VISIBLE\_DEVICES=0,1,2,3,4,5 llamafactory-cli chat examples/inference/llama3\_lora\_sft.yaml 可以与模型进行会话（因为base历史可能设置了某些变量可能会出现Some keys are not used by the HfArgumentParser异常，这个情况下关掉shell再次进入）。如图(clear清除会话历史纪录、exit退出会话）。如果是qwen模型，命令中改成对应的json文件。
   7. 修改merge\_lora/llama3\_lora\_sft.yaml，注意model\_name\_or\_path需要用非量化版的模型（为了降低训练的硬件要求，是基于量化版训练的。但是合并需要使用量化前的版本），否则会报“Cannot merge adapters to a quantized model”异常
   8. 执行命令 llamafactory-cli export examples/merge\_lora/llama3\_lora\_sft.yaml  （这个命令不是cuda执行，故不需要加cuda参数）。对于qwen模型，创建qwen2\_7b\_sft.json文件
   9. 模型转换成ollama支持的格式
      1. 先创建gguf模型的存放路径，mkdir llama3\_lora\_lcredit（或mkdir qwen2\_lora\_lcredit）
      2. 如果是qwen2模型，执行python /llama.cpp/convert-hf-to-gguf.py --outfile /data/workspace/llama\_factory/qwen2\_lora\_lcredit/qwen2\_lora\_xxx.gguf /data/workspace/llama\_factory/LLaMA-Factory/models/qwen2\_lora\_lcredit  
         如果是llama3模型，执行python /llama.cpp/convert-hf-to-gguf.py --outfile /data/workspace/llama\_factory/llama3\_lora\_lcredit/llama3\_lora\_xxx.gguf /data/workspace/llama\_factory/LLaMA-Factory/models/llama3\_lora\_sft  
         (python llama.cpp/convert.py --vocab-type hfft  --outfile /data/workspace/llama\_factory/llama3\_lora\_lcredit/llama3\_lora\_xxx.gguf /data/workspace/llama\_factory/LLaMA-Factory/models/llama3\_lora\_sft 这个命令也能生成模型文件，但是ollma运行报prompt encoding 异常。可能是 --vocab-type不对，但是没找到相关文档。）
      3. 生成的guff 文件比较大（例如llama3 8B为15G），为了降低执行硬件的要求可以对其进行量化处理。量化操作命令：/data/workspace/project/llama.cpp/bin/bin/quantize llama3\_lora\_xxxx.gguf llama3\_lora\_xxx.guuf Q4\_K\_M命令执行完成后模型大小由16G降到了4.9G
      4. 创建modelfile文件 
      5. 执行命令ollama create llama3\_8b\_xx -f llama3\_lora\_xx
      6. 执行ollama list，已经可以看到刚创建并且注册到ollama的模型了。可以通过webui和api接口调用（和其他部署到ollama的模型一样）。
   10. 对于训练模型导入ollama还有一个方式，没有报错、模型也能run起来。但是训练效果几乎没有。因为网上有人介绍，为了避免踩坑在此也介绍下。
       1. ./convert-lora-to-ggml.py /data/workspace/llama\_factory/LLaMA-Factory/saves/llama3-8b/lora/sft llama 得到 ggml-adapter-model.bin
       2. 创建modelfile
       3. 然后执行ollama create命令，是能直接得到一个新的模型。但是基于相同的训练结果（例如llama3 就是/data/workspace/llama\_factory/LLaMA-Factory/saves/llama3-8b/lora/sft），它的微调效果很差（回答太发散了）
   11. 过程中可以使用top和watch -n 2 nvidia-smi 查看资源的使用情况