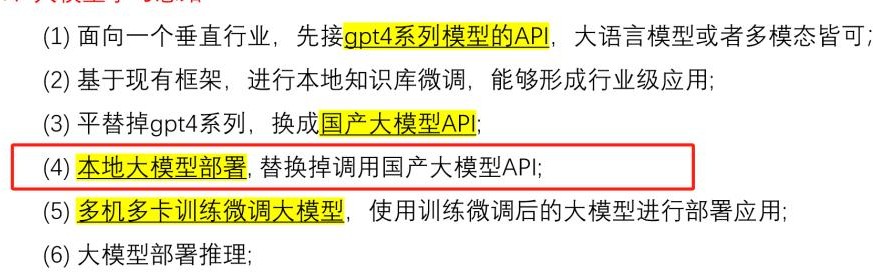
# 中文 llama3 仿 openai api 实战课程大纲



课程介绍

本次课属于【大模型本地部署应用】，基于 Chinese-LLaMA-Alpaca-3

（<https://github.com/ymcui/Chinese-LLaMA-Alpaca-3>）项目，介绍如何封装一个私有的兼容openai api 的大模型接口，并使用ChatGPTNextWeb 开源工具调用此接口。

1. 在前述课程，我们了解了多机多卡微调、本地调用国外/国内大模型API 以及对训练好的模型进行部署推理；
2. 本次课程，主要侧重大模型本地部署及应用，对已经开源的发布的大模型进行本地私有化部署，不需要接入公网，保障数据传输安全，在局域网内部署即可上线应用。以 llama3-8b模型为例，仿openai 发布接口进行对话问答系统发布，完成实战，部署完之后的应用完全不依赖互联网，直接在内网环境下应用。
3. 本地部署的客户端多操作系统适用，windows/macos，都可以用来做本次的实验。

计算端部署在同一局域网内的带显卡的机器上或者cpu 计算也可，看大家的硬件选择，无显卡也可以做实验，但需要内存（保底 24G）较大。

云平台用来训练、微调等，本地部署的实验需要用大家自己的机器。

llama3-8b 仿openai 发布的应用效果演示，本次实验，我们搭建了客户端进行演示：



# 实验环境

本文的基础环境如下 操作系统：ubuntu20.04 CUDA：12.2

英伟达显卡驱动版本：535.183.01

显卡型号：本实验用英伟达 3090（全模型（未开量化）需要单张显存大于 20GB 的英伟达显卡；

\*\*开量化之后的模型可以使用 11GB 显存的英伟达显卡，基本市面上大部分消费级显卡可以满足实验要求,

大家根据自己的显卡进行选择实验）

本文默认您在操作系统的用户名为：llm\_course，如果您使用的是其他用户名，请更改涉及到用户名的地址。

# 下载源码

cd ~/ && wget https://file.huishiwei.top/Chinese-LLaMA-Alpaca-3-3.0.tar.gz tar -xvf Chinese-LLaMA-Alpaca-3-3.0.tar.gz

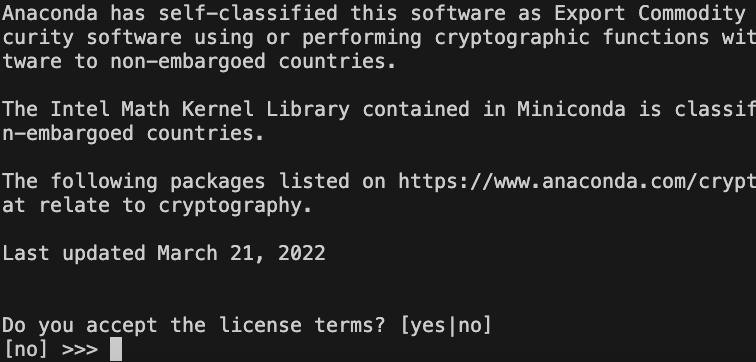
本教程使用的是 3.0 版本

# 安装 miniconda

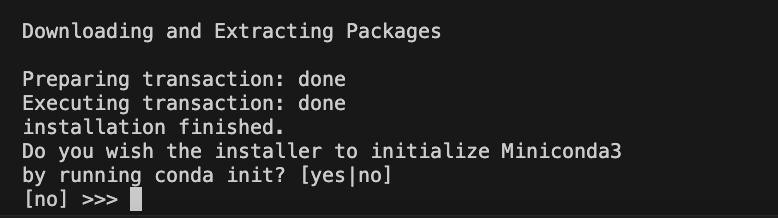
我们将使用miniconda 发行版python 进行后续开发，如果您已经安装 miniconda 或者 conda，可以跳过该章节。

cd ~/ && wget https://repo.anaconda.com/miniconda/Miniconda3-py38\_23.5.2-0-Linux-x86\_64.sh bash Miniconda3-py38\_23.5.2-0-Linux-x86\_64.sh

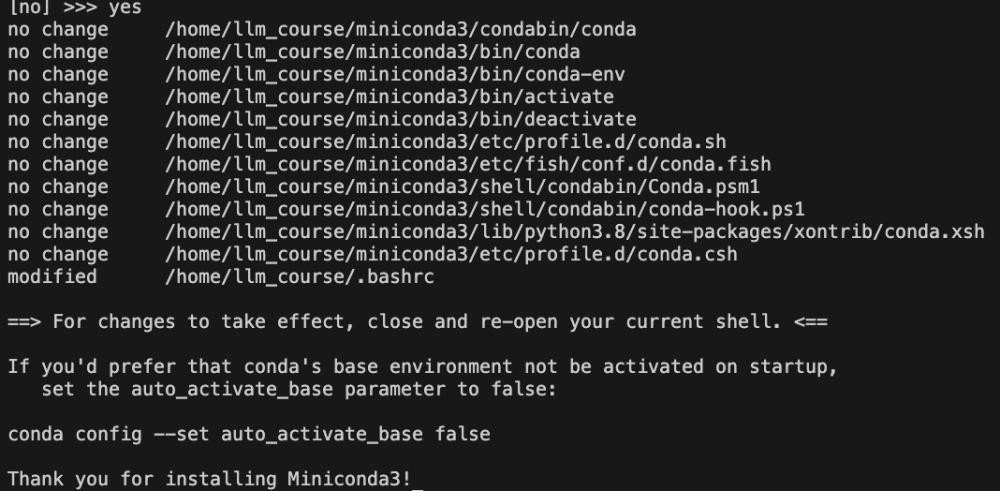
输入上述命令后，一直按回车，直到出现下面的内容：



输入yes，然后，按回车，当出现下面的内容：



输入yes，当出现下面的内容时，代表安装完毕：

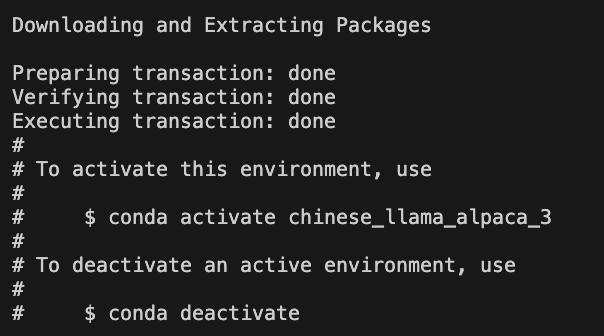


## 创建虚拟环境

关闭刚才的终端，然后重新打开终端，连入服务器，创建一个虚拟环境，用于后续开发（[什么是conda 虚拟环境？）](https://www.cnblogs.com/sx66/p/17823608.html)：

conda create -n chinese\_llama\_alpaca\_3 python=3.8.17 pip -y

出现下面的信息，代表虚拟环境安装完毕：



# 模型下载

我们通过modelscope 下载基座模型。首先激活刚才创建的虚拟环境：conda activate chinese\_llama\_alpaca\_3，然后通过下面命令安装 modelscope 核心组件：pip install modelscope -i https://mirrors.aliyun.com/pypi/simple

## 直接使用命令下载

使用下面命令下载模型：

conda activate chinese\_llama\_alpaca\_3

pip install modelscope -i https://mirrors.aliyun.com/pypi/simple

modelscope download --model ChineseAlpacaGroup/llama-3-chinese-8b-instruct-v3

模型较大，可能需要下载 30 分钟-60 分钟，在此期间不要断开与服务器的连接。

## 基于 screen 下载

由于模型较大，可能需要下载 30 分钟-60 分钟，如果在此期间，你与服务器之间的 ssh 链接断开了，整个下载就中断了，所以我们推荐你使用 screen 的方式（[什么是screen？](https://www.runoob.com/linux/linux-comm-screen.html)）下载，命令如下：

screen -R model\_download

conda activate chinese\_llama\_alpaca\_3

pip install modelscope -i https://mirrors.aliyun.com/pypi/simple

modelscope download --model ChineseAlpacaGroup/llama-3-chinese-8b-instruct-v3

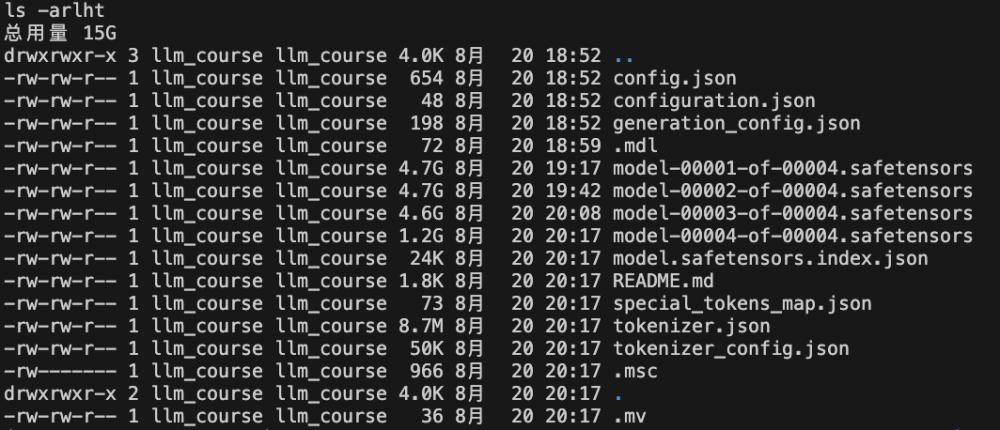
如果在操作过程中断开了服务器连接，可以使用 screen -r model\_download 恢复操作窗口。

## 模型存储位置

使用modelscope 下载完毕后，模型会存储在如下位置：

~/.cache/modelscope/hub/ChineseAlpacaGroup/llama-3-chinese-8b-instruct-v3，正常情况下，可以看到如下内容：

ls -alhrt ~/.cache/modelscope/hub/ChineseAlpacaGroup/llama-3-chinese-8b-instruct-v3



# 开源版 openai 接口启动

中文llama3 的开源版本实现在以下目录：/home/llm\_course/Chinese-LLaMA-Alpaca-3-3.0/scripts/oai\_api\_demo，下面分别是GPU 和CPU 版本的启动流程

## 推理脚本 BUG 修复

在启动GPU 或者CPU 版本的仿openai 接口（stream 流式方式，类似 openai 打字机回复效果）脚本之前，我们需要修复一个 bug，打开/home/llm\_course/Chinese-LLaMA-Alpaca-3-3.0/scripts/oai\_api\_demo/openai\_api\_server.py 文件，找到如下内容：

def stream\_predict( input, max\_new\_tokens=1024, top\_p=0.9, temperature=0.2, top\_k=40, num\_beams=4,

repetition\_penalty=1.1, do\_sample=True, model\_id="llama-3-chinese",

\*\*kwargs,

):

choice\_data = ChatCompletionResponseStreamChoice(

index=0, delta=DeltaMessage(role="assistant"), finish\_reason=None

)

chunk = ChatCompletionResponse(

model=model\_id, choices=[choice\_data], object="chat.completion.chunk",

)

yield "{}".format(chunk.json(exclude\_unset=True, ensure\_ascii=False))

if isinstance(input, str):

prompt = generate\_completion\_prompt(input) else:

prompt = generate\_chat\_prompt(input)

inputs = tokenizer(prompt, return\_tensors="pt") input\_ids = inputs["input\_ids"].to(device) generation\_config = GenerationConfig(

temperature=temperature, top\_p=top\_p, top\_k=top\_k, num\_beams=num\_beams, do\_sample=do\_sample,

\*\*kwargs,

)

streamer = TextIteratorStreamer(tokenizer, skip\_prompt=True, skip\_special\_tokens=True) generation\_kwargs = dict(

streamer=streamer, input\_ids=input\_ids, generation\_config=generation\_config, return\_dict\_in\_generate=True, output\_scores=False, max\_new\_tokens=max\_new\_tokens,

repetition\_penalty=float(repetition\_penalty)

)

将generation\_kwargs 改为如下值：

generation\_kwargs = dict( streamer=streamer, input\_ids=input\_ids, generation\_config=generation\_config, return\_dict\_in\_generate=True, output\_scores=False, max\_new\_tokens=max\_new\_tokens,

repetition\_penalty=float(repetition\_penalty), pad\_token\_id=tokenizer.eos\_token\_id, # 新添加的参数

eos\_token\_id=[tokenizer.eos\_token\_id, tokenizer.convert\_tokens\_to\_ids("<|eot\_id|>")] #新添加的参数

)

加入上面的参数主要是为了兼容llama3 特有的停止token，不然流式接口返回的内容会不断的自动重复，不停止。

## GPU 版本

备份脚本

使用如下命令备份/home/llm\_course/Chinese-LLaMA-Alpaca-3-3.0/requirements.txt 文件：

mv /home/llm\_course/Chinese-LLaMA-Alpaca-3-3.0/requirements.txt /home/llm\_course/Chinese-LLaMA-Alpaca-3-3.0/requirements.bk.txt

安装依赖

通过如下命令创建新的 requirements.txt

cat <<EOF > /home/llm\_course/Chinese-LLaMA-Alpaca-3-3.0/requirements.txt accelerate==0.30.0

aiohttp==3.9.5

aiosignal==1.3.1 annotated-types==0.6.0 anyio==4.3.0

async-timeout==4.0.3 attrs==23.2.0 bitsandbytes==0.43.1 certifi==2024.2.2 charset-normalizer==3.3.2 click==8.1.7 datasets==2.20.0 deepspeed==0.13.1 dill==0.3.7 dnspython==2.6.1 einops==0.8.0 email\_validator==2.1.1 exceptiongroup==1.2.1 fastapi==0.109.2

fastapi-cli==0.0.3 filelock==3.14.0 frozenlist==1.4.1 fsspec==2023.10.0 h11==0.14.0

hjson==3.1.0 httpcore==1.0.5 httptools==0.6.1 httpx==0.27.0 huggingface-hub==0.23.3 idna==3.7

Jinja2==3.1.4 joblib==1.4.2 markdown-it-py==3.0.0 MarkupSafe==2.1.5 mdurl==0.1.2 modelscope==1.17.1 mpmath==1.3.0 multidict==6.0.5

multiprocess==0.70.15 networkx==3.1 ninja==1.11.1.1 numpy==1.24.4

nvidia-cublas-cu12==12.1.3.1 nvidia-cuda-cupti-cu12==12.1.105 nvidia-cuda-nvrtc-cu12==12.1.105 nvidia-cuda-runtime-cu12==12.1.105 nvidia-cudnn-cu12==8.9.2.26 nvidia-cufft-cu12==11.0.2.54 nvidia-curand-cu12==10.3.2.106 nvidia-cusolver-cu12==11.4.5.107 nvidia-cusparse-cu12==12.1.0.106 nvidia-nccl-cu12==2.18.1

nvidia-nvjitlink-cu12==12.4.127 nvidia-nvtx-cu12==12.1.105 orjson==3.10.3

packaging==24.0 pandas==2.0.3 peft==0.7.1 psutil==5.9.8

py-cpuinfo==9.0.0 pyarrow==16.0.0 pyarrow-hotfix==0.6 pydantic==1.10.11 pydantic\_core==2.18.2 Pygments==2.18.0 pynvml==11.5.0

python-dateutil==2.9.0.post0 python-decouple==3.8

python-dotenv==1.0.1 python-multipart==0.0.9 pytz==2024.1 PyYAML==6.0.1

regex==2024.4.28 requests==2.32.3

rich==13.7.1 safetensors==0.4.3 scikit-learn==1.3.2 scipy==1.10.1 shellingham==1.5.4 shortuuid==1.0.13 six==1.16.0

sniffio==1.3.1

sse-starlette==2.1.0 starlette==0.36.3 sympy==1.12 threadpoolctl==3.5.0 tokenizers==0.19.1 torch==2.1.2 tqdm==4.66.4 transformers==4.41.2 triton==2.1.0 typer==0.12.3 typing\_extensions==4.11.0 tzdata==2024.1 ujson==5.9.0 urllib3==2.2.1 uvicorn==0.29.0 uvloop==0.19.0 watchfiles==0.21.0 websockets==12.0 xxhash==3.4.1

yarl==1.9.4 EOF

安装依赖：

pip install -r requirements.txt -i https://mirrors.aliyun.com/pypi/simple

启动服务

通过如下命令启动服务：

python openai\_api\_server.py --gpus 0 --base\_model

/home/llm\_course/.cache/modelscope/hub/ChineseAlpacaGroup/llama-3-chinese-8b-instruct-v3

请注意--gpus 后面的参数 0，代表我们要使用第一块GPU 卡运行大模型服务，如果您的机器只有一块卡，您需要将该参数始终设置为 0，也就是--gpus 0

使用--load\_in\_8bit 或者--load\_in\_4bit 可以开启量化加载。

（\*\*\*这一步，大家根据自己的显卡显存大小进行选择\*\*\*）

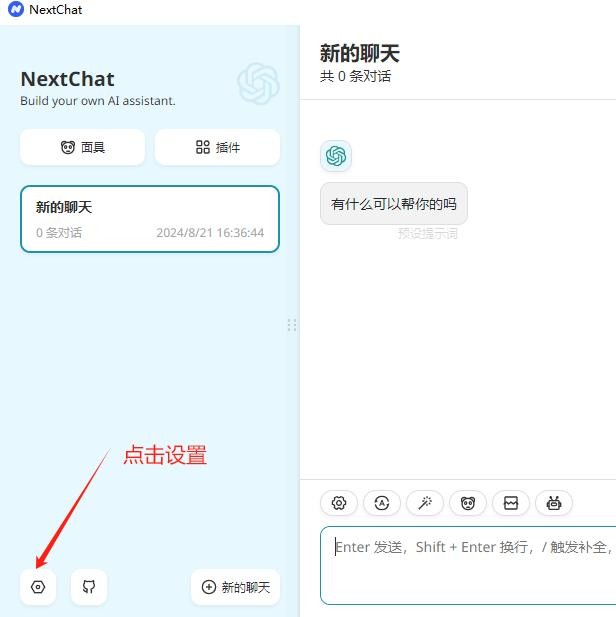
测试效果

我们使用ChatGPTNextWeb 工具测试我们的接口，如果您没有下载客户端，可以通过下面地址下载：

Mac：<https://github.com/ChatGPTNextWeb/ChatGPT-Next-Web/releases/download/v2.14.2/NextChat_2.14.2_universal.dmg>

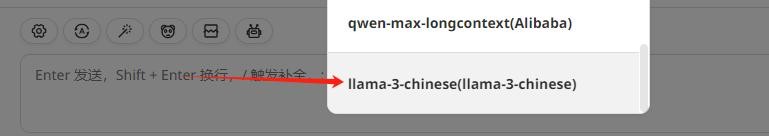
Windows：<https://github.com/ChatGPTNextWeb/ChatGPT-Next-Web/releases/download/v2.14.2/NextChat_2.14.2_x64-setup.exe>

下载后，安装即可。下面开始设置接入我们刚才部署的大模型服务：









## CPU 版本

创建 CPU 版本专用虚拟环境

conda create -n chinese\_llama\_alpaca\_3\_cpu python=3.8.17 pip -y

安装依赖

pip3 install torch==2.3.0 --index-url https://download.pytorch.org/whl/cpu

pip3 install fastapi==0.111.0 peft==0.7.1 pydantic==1.10.11 pydantic\_core==2.18.2 shortuuid==1.0.13 sse-starlette==2.1.0 starlette==0.37.2 transformers==4.41.2 -i

https://mirrors.aliyun.com/pypi/simple

启动服务

python openai\_api\_server.py --only\_cpu --base\_model

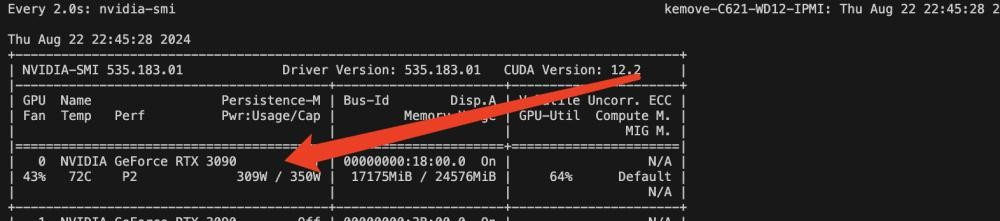
/home/llm\_course/.cache/modelscope/hub/ChineseAlpacaGroup/llama-3-chinese-8b-instruct-v3

测试效果

可以使用GPU 版本同样的方法测试即可。

# GPU 资源释放问题（进阶）

在本项目中，如果项目的使用场景是生成长文本，即使用户提前终止大模型回复，脚本依然会占用GPU 资源直到整个结果完全生成。下面举例说明该情况：



可以看到虽然我们让大模型停止输出，显卡的功耗并没有下降。在我们多次停止再提问时， API 接口大概率会卡死。下面是未修改前的接口代码(openai\_api\_server.py)：

@app.post("/v1/chat/completions")

async def create\_chat\_completion(request: ChatCompletionRequest): """Creates a completion for the chat message"""

msgs = request.messages if isinstance(msgs, str):

msgs = [ChatMessage(role="user", content=msgs)] else:

msgs = [ChatMessage(role=x["role"], content=x["content"]) for x in msgs] if request.stream:

generate = stream\_predict( input=msgs,

max\_new\_tokens=request.max\_tokens, top\_p=request.top\_p, top\_k=request.top\_k, temperature=request.temperature, num\_beams=request.num\_beams,

repetition\_penalty=request.repetition\_penalty, do\_sample=request.do\_sample,

)

return EventSourceResponse(generate, media\_type="text/event-stream") output = predict(

input=msgs, max\_new\_tokens=request.max\_tokens, top\_p=request.top\_p, top\_k=request.top\_k, temperature=request.temperature, num\_beams=request.num\_beams,

repetition\_penalty=request.repetition\_penalty, do\_sample=request.do\_sample,

)

choices = [

ChatCompletionResponseChoice(index=i, message=msg) for i, msg in enumerate(msgs)

]

choices += [

ChatCompletionResponseChoice(

index=len(choices), message=ChatMessage(role="assistant", content=output)

)

]

return ChatCompletionResponse(choices=choices)

我们需要对源码做一些修改，首先是引入一些包

# 在文件的开头处引入额外引入 asyncio 和 StoppingCriteria 以及 StoppingCriteriaList import asyncio # 额外引入

from transformers import StoppingCriteria, StoppingCriteriaList # 额外引入 from peft import PeftModel

然后定义StopOnEvent 类

from openai\_api\_protocol import ( ChatCompletionRequest, ChatCompletionResponse, ChatMessage, ChatCompletionResponseChoice, CompletionRequest, CompletionResponse, CompletionResponseChoice, EmbeddingsRequest, EmbeddingsResponse, ChatCompletionResponseStreamChoice, DeltaMessage,

)

# 在这里定义 StopOnEvent 类

class StopOnEvent(StoppingCriteria): def init (self) -> None: self.is\_stop = False

super(). init ()

def call (self, input\_ids: torch.LongTensor, scores: torch.FloatTensor, \*\*kwargs) -

> bool:

if self.is\_stop: return True

return False

在create\_chat\_completion 函数内添加一个名为 event\_publisher 的函数（注意是在 create\_chat\_completion 函数内添加），然后将原来调用stream\_predict 的地方改为调用 event\_publisher：

@app.post("/v1/chat/completions")

async def create\_chat\_completion(request: ChatCompletionRequest): """Creates a completion for the chat message"""

async def event\_publisher( input, max\_new\_tokens=4096, top\_p=0.75, temperature=0.1, top\_k=40, num\_beams=4,

repetition\_penalty=1.0, do\_sample=True,

model\_id="chinese-llama-alpaca-2",

\*\*kwargs,

):

choice\_data = ChatCompletionResponseStreamChoice(

index=0, delta=DeltaMessage(role="assistant"), finish\_reason=None

)

chunk = ChatCompletionResponse( model=model\_id, choices=[choice\_data], object="chat.completion.chunk",

)

yield "{}".format(chunk.json(exclude\_unset=True, ensure\_ascii=False))

if isinstance(input, str):

prompt = generate\_completion\_prompt(input) else:

prompt = generate\_chat\_prompt(input) inputs = tokenizer(prompt, return\_tensors="pt") input\_ids = inputs["input\_ids"].to(device) generation\_config = GenerationConfig(

temperature=temperature, top\_p=top\_p, top\_k=top\_k, num\_beams=num\_beams, do\_sample=do\_sample,

\*\*kwargs,

)

streamer = TextIteratorStreamer(tokenizer, skip\_prompt=True, skip\_special\_tokens=True)

stop = StopOnEvent() generation\_kwargs = dict(

streamer=streamer, input\_ids=input\_ids, generation\_config=generation\_config, return\_dict\_in\_generate=True, output\_scores=False, max\_new\_tokens=max\_new\_tokens,

repetition\_penalty=float(repetition\_penalty), stopping\_criteria=StoppingCriteriaList([stop]), pad\_token\_id=tokenizer.eos\_token\_id, eos\_token\_id = [tokenizer.eos\_token\_id,

tokenizer.convert\_tokens\_to\_ids("<|eot\_id|>")]

)

i = 0

Thread(target=model.generate, kwargs=generation\_kwargs).start() try:

for new\_text in streamer:

choice\_data = ChatCompletionResponseStreamChoice(

index=0, delta=DeltaMessage(content=new\_text), finish\_reason=None

)

chunk = ChatCompletionResponse(

model=model\_id, choices=[choice\_data], object="chat.completion.chunk"

)

chunk\_str = "{}".format(chunk.json(exclude\_unset=True, ensure\_ascii=False)) i += 1

yield chunk\_str

await asyncio.sleep(0.03) except asyncio.CancelledError as e:

print(f"Disconnected from client (via refresh/close)") # Do any other cleanup, if any

stop.is\_stop = True raise e

msgs = request.messages if isinstance(msgs, str):

msgs = [ChatMessage(role="user", content=msgs)] else:

msgs = [ChatMessage(role=x["role"], content=x["content"]) for x in msgs] if request.stream:

# generate = stream\_predict( generate = event\_publisher(

input=msgs, max\_new\_tokens=request.max\_tokens, top\_p=request.top\_p, top\_k=request.top\_k, temperature=request.temperature, num\_beams=request.num\_beams,

repetition\_penalty=request.repetition\_penalty, do\_sample=request.do\_sample,

)

return EventSourceResponse(generate, media\_type="text/event-stream") output = predict(

input=msgs, max\_new\_tokens=request.max\_tokens, top\_p=request.top\_p,

top\_k=request.top\_k, temperature=request.temperature, num\_beams=request.num\_beams, repetition\_penalty=request.repetition\_penalty, do\_sample=request.do\_sample,

)

choices = [

ChatCompletionResponseChoice(index=i, message=msg) for i, msg in enumerate(msgs)

]

choices += [

ChatCompletionResponseChoice(

index=len(choices), message=ChatMessage(role="assistant", content=output)

)

]

return ChatCompletionResponse(choices=choices)

# 常见问题

19327 端口冲突

启动脚本时，19327 端口冲突，需要修改脚本。找到第 394 行，如下内容：

uvicorn.run(app, host="0.0.0.0", port=19327, workers=1, log\_config=log\_config)

其中port=19327 就是web 服务启动的端口号，将 19327 改成其他数字即可。

# 作业

1. 完成本次实验的本地化部署及客户端演示，能够调用 llama3 本地模型进行智能问答（应用时无网环境）；
2. 切换llama3 模型，可以尝试接入其他模型，如千问等，对比效果；

（大家完成每次作业后，发给班主任，即可获得云平台的机时奖励）