一、llama3 简介

1. 开源模型

- Meta开发并发布了Meta Llama 3模型系列，包含预训练和指令调整的生成文本模型，大小为8B和70B参数。
- 发布日期：April 18, 2024

Llama 3	Training Data	Params	Context length	GQA	Token count	Knowledge cutoff
8B	公开可用的在线数据的新组合	8B	8k	Yes	15T+	March, 2023
70B	公开可用的在线数据的新组合	70B	8k	Yes	15T+	December, 2023

- 模型输入：仅限文本输入
- 模型输出：仅生成文本和代码

2. 性能表现

（1）Instruct Model 指令模型

① llama3 VS 其它模型

Llama 3的8B和70B参数模型在预训练和后续训练中取得显著进步：

Meta Llama 3 Instruct model performance

	Meta Llama 3 8B	Gemma 7B - It Measured	Mistral 7B Instruct Measured
MMLU 5-shot	68.4	53.3	58.4
GPQA 0-shot	34.2	21.4	26.3
HumanEval 0-shot	62.2	30.5	36.6
GSM-8K 8-shot, CoT	79.6	30.6	39.9
MATH 4-shot, CoT	30.0	12.2	11.0

	Meta Llama 3 70B	Gemini Pro 1.5 Published	Claude 3 Sonnet Published
MMLU 5-shot	82.0	81.9	79.0
GPQA 0-shot	39.5	41.5 CoT	38.5 CoT
HumanEval 0-shot	81.7	71.9	73.0
GSM-8K 8-shot, CoT	93.0	91.7 11-shot	92.3 0-shot
MATH 4-shot, CoT	50.4	58.5 Minerva prompt	40.5

开发1,800个提示的高质量人工评估集，覆盖12个关键用例，人工评估结果显示Llama 3的优势明显。

② llama3 VS llama2

Instruction tuned models

Benchmark	Llama 3 8B	Llama 2 7B	Llama 2 13B	Llama 3 70B	Llama 2 70B
MMLU (5-shot)	68.4	34.1	47.8	82.0	52.9
GPQA (0-shot)	34.2	21.7	22.3	39.5	21.0
HumanEval (0-shot)	62.2	7.9	14.0	81.7	25.6
GSM-8K (8-shot, CoT)	79.6	25.7	77.4	93.0	57.5
MATH (4-shot, CoT)	30.0	3.8	6.7	50.4	11.6

（2）Pretrained Model 预训练模型

① llama3 VS 其它模型

Meta Llama 3 Pre-trained model performance

	Meta Llama 3 8B	Mistral 7B		Gemma 7B	
		Published	Measured	Published	Measured
MMLU 5-shot	66.6	62.5	63.9	64.3	64.4
AGIEval English 3-5-shot	45.9	--	44.0	41.7	44.9
BIG-Bench Hard 3-shot, CoT	61.1	--	56.0	55.1	59.0
ARC-Challenge 25-shot	78.6	78.1	78.7	53.2 0-shot	79.1
DROP 3-shot, F1	58.4	--	54.4	--	56.3

	Meta Llama 3 70B	Gemini Pro 1.0	Mixtral 8x22B
		Published	Measured
MMLU 5-shot	79.5	71.8	77.7
AGIEval English 3-5-shot	63.0	--	61.2
BIG-Bench Hard 3-shot, CoT	81.3	75.0	79.2
ARC-Challenge 25-shot	93.0	--	90.7
DROP 3-shot, F1	79.7	74.1 variable-shot	77.6

② llama3 VS llama2

Base pretrained models

Category	Benchmark	Llama 3 8B	Llama2 7B	Llama2 13B	Llama 3 70B	Llama2 70B
General	MMLU (5-shot)	66.6	45.7	53.8	79.5	69.7
	AGIEval English (3-5 shot)	45.9	28.8	38.7	63.0	54.8
	CommonSenseQA (7-shot)	72.6	57.6	67.6	83.8	78.7
	Winogrande (5-shot)	76.1	73.3	75.4	83.1	81.8
	BIG-Bench Hard (3-shot, CoT)	61.1	38.1	47.0	81.3	65.7
	ARC-Challenge (25-shot)	78.6	53.7	67.6	93.0	85.3
Knowledge reasoning	TriviaQA-Wiki (5-shot)	78.5	72.1	79.6	89.7	87.5
Reading comprehension	SQuAD (1-shot)	76.4	72.2	72.1	85.6	82.6
	QuAC (1-shot, F1)	44.4	39.6	44.9	51.1	49.4
	BoolQ (0-shot)	75.7	65.5	66.9	79.0	73.1
	DROP (3-shot, F1)	58.4	37.9	49.8	79.7	70.2

3. 模型架构

- Llama 3采用标准的仅解码器 (transformer decoder-only) 架构
- 使用128K词汇的分词器(tokenizer)以更有效地编码语言
- 引入分组查询注意力(grouped query attention, GQA)机制以提高8B和70B大小模型的推理效率
- 训练时避免跨文档边界的自注意力，使用了8,192个标记的掩码(mask)

```
{
  "architectures": [
    "LlamaForCausalLM"
  ],
  "attention_bias": false,
  "attention_dropout": 0.0,
  "bos_token_id": 128000,
  "eos_token_id": 128001,
  "hidden_act": "silu",
  "hidden_size": 4096,
  "initializer_range": 0.02,
  "intermediate_size": 14336,
  "max_position_embeddings": 8192,
  "model_type": "llama",
  "num_attention_heads": 32,
  "num_hidden_layers": 32,
  "num_key_value_heads": 8,
  "pretraining_tp": 1,
  "rms_norm_eps": 1e-05,
  "rope_scaling": null,
  "rope_theta": 500000.0,
  "tie_word_embeddings": false,
  "torch_dtype": "bfloat16",
  "transformers_version": "4.40.0.dev0",
  "use_cache": true,
  "vocab_size": 128256
}
```


(1) `architectures`: 模型的架构，"LlamaForCausalLM"一个用于因果语言建模的模型，它用来预测文本序列中的下一个单词，模型的预测仅基于先前的单词（即因果关系）。

(2) `attention_bias`: 指示是否在注意力机制中使用偏置项，这里`false`，意味着不使用。在注意力机制中，偏置项通常用于调整输入单词之间的关系强度。

(3) `attention_dropout`: 注意力层中使用的dropout比率，这里是0.0，表示没有dropout。Dropout是一种正则化技术，用于防止神经网络过拟合。

(4) `bos_token_id` 和 `eos_token_id`: 分别表示文本序列开始和结束的特殊标记的ID，bose ID=128000，eos ID=128001，模型在处理文本时会用到这些索引来识别序列的开始和结束。

(5) `hidden_act`: 隐藏层激活函数的类型，这里是"silu"，即Swish激活函数。这个激活函数是ReLU的一种变体，其形式为 f(x) = x * sigmoid(x)，某些情况下Swish比传统的ReLU性能更好，因为它允许一些负值通过，从而可能提供更好的梯度流。

(6) `hidden_size`: 隐藏层的大小，这里是4096，说明每个隐藏层有4096个神经元，模型能够捕捉和学习更加复杂的特征，但同时这也意味着模型的参数量很大，计算成本也很高。在transformer模型中，`hidden_size`也是自注意力层输出的维度。

(7) `initializer_range`: 参数初始化时使用的范围，这里是0.02，一个相对小的数值，意味着模型的权重将从一个相对狭窄的范围内随机选择，这通常有助于确保模型训练的稳定性。

(8) `intermediate_size`: 在Transformer模型的架构中，模型内部每个注意力机制之后的前馈网络的大小，这里是14336，它表明模型的前馈网络相对宽，能够处理和产生更复杂的特征表示。

(9) `max_position_embeddings`: 可以处理的最大序列长度，这里是8192，长序列处理能力意味着模型可以在不丢失上下文的情况下处理更多的信息。

(10) `model_type`: 模型的类型，这里是llama。

(11) `num_attention_heads`: 注意力层的头数，这里是32，说明注意力机制被分割为32个不同的头部。在这种机制中，模型的注意力被分割成多个“头”，每个头独立地处理信息，然后其输出被组合起来。32个头意味着模型能够同时从32个不同的表示空间中学习信息，可以帮助模型更好地理解复杂的输入数据。

(12) `num_hidden_layers`: 隐藏层的数量，这里是32层，允许模型学习非常复杂的特征表示。

(13) `num_key_value_heads`: 键-值对在注意力机制中的头数，这里是8。

(14) `pretraining_tp`: 模型特定的预训练设置，这里是1。

(15) `rms_norm_eps`: RMS归一化的eps值，用于数值稳定性，这里是1e-05。RMS归一化是一种变体的层归一化技术，可能用于稳定深层网络的训练。

(16) `rope_scaling` 和 `rope_theta`: 一种特定的位置编码技术或优化策略

(17) `tie_word_embeddings`: 输入的词嵌入层和输出层是否共享权重，这里是`false`，意味着不共享。

(18) `torch_dtype`: 模型使用的数据类型，这里是"bfloat16"，这是一种精度较低但计算效率更高的数据类型。

(19) `transformers_version`: 指示用于训练或运行模型的transformers库的版本，这里是一个开发版"4.40.0.dev0"。

(20) `use_cache`: 是否在模型前向传播时缓存一些中间结果以加速计算，尤其是在生成序列时，这里是`true`。

(21) `vocab_size`: 词汇表的大小，这里是128256，128256个词表明模型可以处理一个非常大的词汇，能够覆盖广泛的语言使用。

4. 训练数据

- 从公共来源收集的超过15T（万亿）个tokens上预训练，训练集体量是Llama 2的七倍，包括的代码量是四倍；
- 预训练数据中包含了超过5%为未来多语言用例准备的高质量非英语数据，涵盖30多种语言；
- 7B 模型的预训练数据截止时间为 2023 年 3 月，70B 模型的预训练数据截止时间为 2023 年 12 月。
- 为确保高质量数据，开发了数据过滤流水线，包括启发式过滤器、不适宜内容过滤器（NSFW filters）、语义去重手段和文本分类器；

- 利用Llama 2生成文本质量分类器的训练数据；

5. 指令微调

- 采用多种方法结合微调，包括SFT、PPO、DPO，以便与人类对有用性和安全性的偏好保持一致。
- 注重通过人工注释进行数据质量保证，偏好排名学习显著提升推理和编码任务性能。

6. 未来规划

- 8B和70B模型只是开始，未来还有更多规模更大的模型将发布，例如400B参数模型。
- 在未来几个月内将发布具备多模态性、多语言对话能力、更长上下文窗口和更强大整体能力的模型。
- 一旦Llama 3训练完成，将发布详细的研究论文。

Meta Llama 3 400B+ (still training)
Checkpoint as of Apr 15, 2024

	PRE-TRAINED		INSTRUCT	
	Meta Llama 3 400B+		Meta Llama 3 400B+	
MMLU 5-shot	84.8	GPQA 0-shot	48.0	
AGIEval English 3-5-shot	69.9	HumanEval 0-shot	84.1	
BIG-Bench Hard 3-shot, CoT	85.3	GSM-8K 8-shot, CoT	94.1	
ARC-Challenge 25-shot	96.0	MATH 4-shot, CoT	57.8	
DROP 3-shot, F1	83.5			

二、llama3 案例实战

1. 下载模型

1.1 官网下载

```
https://llama.meta.com/llama-downloads/
```

1.2 HuggingFace下载

```
https://huggingface.co/collections/meta-llama/meta-llama-3-66214712577ca38149ebb2b6
```

2. 环境配置

```
# 1. 安装虚拟环境
conda create -n llama python=3.11
conda activate llama

# 2. 下载 llama3 项目
git clone https://github.com/meta-llama/llama3.git
```

```
(llama) root@ubuntu22:~# git clone https://github.com/meta-llama/llama3.git
正克隆到 'llama3'...
remote: Enumerating objects: 337, done.
remote: Counting objects: 100% (174/174), done.
remote: Compressing objects: 100% (76/76), done.
remote: Total 337 (delta 130), reused 128 (delta 98), pack-reused 163
接收对象中：100% (337/337), 588.33 KiB | 1.38 MiB/s, 完成.
处理 delta 中：100% (189/189), 完成.
```

```
# 3. 安装依赖包
cd llama3

pip install -e .
```

```
(llama) root@ubuntu22:~#
(llama) root@ubuntu22:~# cd llama3/
(llama) root@ubuntu22:~/llama3# pip install -e .
Looking in indexes: https://pypi.tuna.tsinghua.edu.cn/simple/
Obtaining file:///root/llama3
  Preparing metadata (setup.py) ... done
Collecting torch (from llama3==0.0.1)
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/c3/33/d7a6123231bd4d04c7005dde8507235772f3bc4622a2
5f3a88c016415d49/torch-2.2.2-cp311-cp311-manylinux1_x86_64.whl (755.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 755.5/755.6 MB 12.8 MB/s eta 0:00:01
```

3. 模型推理 - GPU版本

3.1 快速实践

（1）模型并行值：

- 8B模型的模型并行（MP）值为1。
- 70B模型的模型并行（MP）值为8。

> 在Llama 3模型中，MP值指的是并行化模型时使用的硬件单元数。一个MP值为1的8B模型可能意味着整个模型可以在单个计算设备上运行，而一个MP值为8的70B模型可能意味着模型被分成8部分，每部分在不同的计算设备上运行。

（2）序列长度和批处理大小：

- 所有模型支持最多8192个tokens的序列长度。
- 缓存是根据`max_seq_len`和`max_batch_size`值预分配的，应根据你的硬件配置这些值。

（3）基于transformers进行推理：

```
import transformers
import torch

pipeline = transformers.pipeline(
task="text-generation",
model=""/root/models/Meta-Llama-3-8B-instruct"", # "/root/models/Meta-Llama-3-8B",
model_kwargs={"torch_dtype": torch.bfloat16},
device="cuda",
)
print(pipeline("Hey how are you doing today ?"))
```

3.2 基于 vLLM 进行模型推理

（1）安装 vLLM

```
pip install vllm
```

（2）模型推理

（2.1）completion模式

```
# 1. 服务部署
python -m vllm.entrypoints.openai.api_server --model /root/models/Meta-Llama-3-8B --dtype
auto --api-key 123456

# 2. 服务测试 （vllm_completion_test.py）

from openai import OpenAI
client = OpenAI(
    base_url="http://localhost:8000/v1",
    api_key="123456",
)
print("服务连接成功")
completion = client.completions.create(
  model="/root/models/Meta-Llama-3-8B",
  prompt="San Francisco is a",
  max_tokens=128,
)
print("### San Francisco is : ")
print("Completion result: ", completion)
```

（2.2）chat模式

```
# 1. 服务部署
python -m vllm.entrypoints.openai.api_server --model /root/models/Meta-Llama-3-8B-Instruct
--dtype auto --api-key 123456

# 2. 服务测试

from openai import OpenAI
client = OpenAI(
    base_url="http://localhost:8000/v1",
    api_key="123456",
)
print("服务连接成功")
completion = client.chat.completions.create(
  model="/root/models/Meta-Llama-3-8B-Instruct",
  messages=[
    {"role": "system", "content": "You are a helpful assistant."},
    {"role": "user", "content": "what is the capital of America ?"}
  ],
  max_tokens = 128,
)
print(completion.choices[0].message)
```

3.3 预训练模型推理

- 这些模型没有针对聊天或问答（Q&A）进行微调。
- 它们应该通过提示进行操作，以使预期答案成为提示的自然延续。
- 示例代码见`example_text_completion.py`文件。
- 使用llama-3-8b模型时，命令行参数`nproc_per_node`需要设置为模型并行（MP）值。

```
torchrun --nproc_per_node 1 example_text_completion.py \
    --ckpt_dir /root/models/Meta-Llama-3-8B/original \
    --tokenizer_path /root/models/Meta-Llama-3-8B/original/tokenizer.model \
    --max_seq_len 128 --max_batch_size 4
```

3.4 指令微调模型推理

- 微调模型是针对对话应用训练的。
- 要获得预期的功能和性能，需要遵循`ChatFormat`中定义的特定格式：
  - 提示以`<begin_of_text>`特殊标记开始。
  - 然后是一个或多个消息，每个消息以`<start_header_id>`标记开始，标识角色（系统、用户或助手）。
  - 每条消息的结尾由`<end_header_id>`标记标记。
- 可以部署额外的分类器来过滤掉被认为不安全的输入和输出。具体示例查看llama-recipes仓库。

```
torchrun --nproc_per_node 1 example_chat_completion.py \
    --ckpt_dir /root/models/Meta-Llama-3-8B-Instruct/original \
    --tokenizer_path /root/models/Meta-Llama-3-8B-Instruct/original/tokenizer.model \
    --max_seq_len 512 --max_batch_size 6
```

3.5 Llama3 - special tokens

3.5.1 特殊标记

1. `<|begin_of_text|>`：相当于BOS（Begin Of Sentence）标记，表示文本的开始。
2. `<|eot_id|>`：表示轮次中消息的结束。
3. `<|start_header_id|>{role}<|end_header_id|>`：这些tokens包围特定消息的角色。角色可以是：system（系统）、user（用户）或assistant（助手）。
4. `<|end_of_text|>`：相当于EOS（End Of Sentence）标记，生成此标记后，Llama 3将停止生成更多标记。

3.5.2 Meta Llama 3 base

```
<|begin_of_text|>{{ user_message }}
```

示例：

```
<begin_of_text>用户：你好，请给我介绍一下故宫。
```

3.5.3 Meta Llama 3 chat

（1）Single message example

```
<|begin_of_text|>1<|start_header_id|>user<|end_header_id|>2

{{ user_message }}3<|eot_id|>4<|start_header_id|>assistant<|end_header_id|>5
```

1. 指定提示的开始。
2. 为接下来的消息指定角色，即“用户”。
3. 输入消息（来自“用户”）。
4. 指定输入消息的结束。
5. 为接下来的消息指定角色，即“助手”。

遵循此提示，Llama 3 将通过添加 {{assistant_message}} 来完成它，它将在生成 <|eot_id|> 时停止生成。

（2）System prompt message added to a single user message

```
<|begin_of_text|><|start_header_id|>system<|end_header_id|>

{{ system_prompt }}<|eot_id|><|start_header_id|>user<|end_header_id|>

{{ user_message }}<|eot_id|><|start_header_id|>assistant<|end_header_id|>
```

（3）System prompt and multiple turn conversation between the user and assistant

```
<|begin_of_text|><|start_header_id|>system<|end_header_id|>

{{ system_prompt }}<|eot_id|><|start_header_id|>user<|end_header_id|>

{{ user_message_1 }}<|eot_id|><|start_header_id|>assistant<|end_header_id|>

{{ model_answer_1 }}<|eot_id|><|start_header_id|>user<|end_header_id|>

{{ user_message_2 }}<|eot_id|><|start_header_id|>assistant<|end_header_id|>
```

4. 模型推理 - CPU版本

```
# 实践环境：
1. Ubuntu 22
2. Python 3.11
```

4.1 安装 llama.cpp

```
# 1. 获取项目代码
# git clone https://github.com/ggerganov/llama.cpp

git clone https://github.com/pcuenca/llama.cpp.git
cd llama.cpp

# 2. 安装 g++ (可选，如果已经安装，则跳过)
sudo apt update
sudo apt install g++

# 3. 在项目 llama.cpp/ 目录下，执行命令
make
```

4.2 准备与量化

```
# 将HF格式的模型转换为gguf格式
## 1️⃣ 将 Meta-Llama-3-8B-Instruct 拷贝到 llama.cpp/models 下

## 2️⃣ 执行转换命令
python convert-hf-to-gguf.py models/Meta-Llama-3-8B-Instruct/ --outfile models/ggml-meta-llama-3-8b-f16.gguf
```

```
(llama) root@ubuntu22:~/new/llama.cpp/models# ll -h
总计 15G
drwxr-xr-x  3 root root 4.0K  4月 21 18:47 ./
drwxr-xr-x 24 root root 4.0K  4月 21 18:35 ../
-rw-r--r--  1 root root  15G  4月 21 18:48 ggml-meta-llama-3-8b-f16.gguf
drwxr-xr-x  2 root root 4.0K  4月 21 18:42 Meta-Llama-3-8B-Instruct/
(llama) root@ubuntu22:~/new/llama.cpp/models#
```

```
# 3. 模型量化，从fb16量化到4-bit

./quantize ./models/ggml-meta-llama-3-8b-f16.gguf ./models/ggml-meta-3-8b-Q4_K_M.gguf Q4_K_M
```

```
(llama) root@ubuntu22:~/new/llama.cpp#
(llama) root@ubuntu22:~/new/llama.cpp# ./quantize ./models/ggml-meta-llama-3-8b-f16.gguf ./models/ggml-meta-3-8b-Q4_K_M.gguf Q4_K_M
main: build = 2702 (c971ac0)
main: built with cc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0 for x86_64-linux-gnu
main: quantizing './models/ggml-meta-llama-3-8b-f16.gguf' to './models/ggml-meta-3-8b-Q4_K_M.gguf' as Q4_K_M
llama_model_loader: loaded meta data with 20 key-value pairs and 291 tensors from ./models/ggml-meta-llama-3-8b-f16.gguf (version GGUF V3 (l
atest))
llama_model_loader: Dumping metadata keys/values. Note: KV overrides do not apply in this output.
llama_model_loader: - kv   0:                       general.architecture str              = llama
llama_model_loader: - kv   1:                               general.name str              = Meta-Llama-3-8B-Instruct
llama_model_loader: - kv   2:                          llama.block_count u32              = 32
llama_model_loader: - kv   3:                       llama.context_length u32              = 8192
llama_model_loader: - kv   4:                     llama.embedding_length u32              = 4096
llama_model_loader: - kv   5:                  llama.feed_forward_length u32              = 14336
llama_model_loader: - kv   6:                 llama.attention.head_count u32              = 32
llama_model_loader: - kv   7:              llama.attention.head_count_kv u32              = 8
llama_model_loader: - kv   8:                       llama.rope.freq_base f32              = 500000.000000
llama_model_loader: - kv   9:     llama.attention.layer_norm_rms_epsilon f32              = 0.000010
llama_model_loader: - kv  10:                           general.file_type u32              = 1
llama_model_loader: - kv  11:                            llama.vocab_size u32              = 128256
llama_model_loader: - kv  12:                  llama.rope.dimension_count u32              = 128
llama_model_loader: - kv  13:                       tokenizer.ggml.model str              = gpt2
llama_model_loader: - kv  14:                      tokenizer.ggml.tokens arr[str,128256]  = ["!", "\"", "#", "$", "%", "&", "'", ...
llama_model_loader: - kv  15:                  tokenizer.ggml.token_type arr[i32,128256]  = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
llama_model_loader: - kv  16:                      tokenizer.ggml.merges arr[str,280147]  = ["Ġ Ġ", "Ġ ĠĠĠ", "ĠĠ ĠĠ", "...
llama_model_loader: - kv  17:                tokenizer.ggml.bos_token_id u32              = 128000
llama_model_loader: - kv  18:                tokenizer.ggml.eos_token_id u32              = 128001
llama_model_loader: - kv  19:                    tokenizer.chat_template str              = {% set loop_messages = messages %}{% ...
llama_model_loader: - type  f32:   65 tensors
llama_model_loader: - type  f16:  226 tensors
llama_model_quantize_internal: meta size = 7835840 bytes
[   1/ 291]                    token_embd.weight - [ 4096, 128256,     1,     1], type =    f16, converting to q4_K .. size =  1002.00 MiB -
>   281.81 MiB
[   2/ 291]               blk.0.attn_norm.weight - [ 4096,     1,     1,     1], type =    f32, size =    0.016 MB
[   3/ 291]                blk.0.ffn_down.weight - [14336,  4096,     1,     1], type =    f16, converting to q6_K .. size =   112.00 MiB ->
```

```
(llama) root@ubuntu22:~/new/llama.cpp/models#
(llama) root@ubuntu22:~/new/llama.cpp/models# ll -h
总计 20G
drwxr-xr-x  3 root root 4.0K  4月 21 19:07 ./
drwxr-xr-x 24 root root 4.0K  4月 21 19:07 ../
-rw-r--r--  1 root root 4.6G  4月 21 19:09 ggml-meta-3-8b-Q4_K_M.gguf
-rw-r--r--  1 root root  15G  4月 21 18:48 ggml-meta-llama-3-8b-f16.gguf
drwxr-xr-x  2 root root 4.0K  4月 21 18:42 Meta-Llama-3-8B-Instruct/
(llama) root@ubuntu22:~/new/llama.cpp/models#
(llama) root@ubuntu22:~/new/llama.cpp/models#
(llama) root@ubuntu22:~/new/llama.cpp/models#
```

4.3 量化模型推理和问答交互

```
# 参数解析:
# -c : --ctx-size N: 提示上下文的大小（默认为512，0表示从模型加载）。
# -b : --batch-size N: 逻辑上的最大批处理大小（默认为2048）。
# -n : 预测的token数量（默认为-1，-1表示无限，-2表示直到上下文被填满）。
# -t : 生成过程中使用的线程数（默认为6）。这控制了并行处理的程度。
# repeat_penalty : 惩罚重复序列的tokens（默认为1.0，1.0表示禁用，意味着不会对重复的tokens施加惩罚，即允许重复出现相同的词或短语）。
# --top_k : Top-K采样（默认为40，0表示禁用）
# --top_p : Top-P采样（默认为0.9，1.0表示禁用）。
# --color : 彩色输出，用以区分提示和用户输入。
# -i : --interactive: 在交互式模式下运行。这意味着程序会保持运行状态，等待用户的输入，然后响应。
# - r : 在交互式模式下，在达到指定的提示（PROMPT）文本时暂停文本生成，并将控制权返回给用户。
# - f : --file FNAME: 使用文件开始生成。可以指定一个文件名，该文件包含用于启动文本生成过程的提示。


# 测试-1
./main -m ./models/ggml-meta-3-8b-Q4_K_M.gguf -c 512 -b 64 -n 256 -t 12 --repeat_penalty 1.0 --top_k 20 --top_p 0.5 --color -i -r "User:" -f prompts/chat-with-bob.txt


# 测试-2
./main -m ./models/ggml-meta-3-8b-Q4_K_M.gguf -c 512 -b 64 -n 256 -t 12 --repeat_penalty 1.0 --top_k 20 --top_p 0.5 --color -i -r "助手:" -f prompts/chat-with-baichuan.txt
```

```
(llama) root@ubuntu22:~/new/llama.cpp#
(llama) root@ubuntu22:~/new/llama.cpp# ./main -m ./models/ggml-meta-3-8b-Q4_K_M.gguf -n 128
Log start
main: build = 2702 (c971ac0)
main: built with cc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0 for x86_64-linux-gnu
main: seed  = 1713698139
llama_model_loader: loaded meta data with 21 key-value pairs and 291 tensors from ./models/ggml-meta-3-8b-Q4_K_M.gguf (version GGUF V3 (late
st))
llama_model_loader: Dumping metadata keys/values. Note: KV overrides do not apply in this output.
llama_model_loader: - kv   0:                       general.architecture str              = llama
llama_model_loader: - kv   1:                               general.name str              = Meta-Llama-3-8B-Instruct
llama_model_loader: - kv   2:                          llama.block_count u32              = 32
llama_model_loader: - kv   3:                       llama.context_length u32              = 8192
llama_model_loader: - kv   4:                     llama.embedding_length u32              = 4096
llama_model_loader: - kv   5:                  llama.feed_forward_length u32              = 14336
llama_model_loader: - kv   6:                 llama.attention.head_count u32              = 32
llama_model_loader: - kv   7:              llama.attention.head_count_kv u32              = 8
llama_model_loader: - kv   8:                       llama.rope.freq_base f32              = 500000.000000
llama_model_loader: - kv   9:     llama.attention.layer_norm_rms_epsilon f32              = 0.000010
llama_model_loader: - kv  10:                           general.file_type u32              = 15
llama_model_loader: - kv  11:                            llama.vocab_size u32              = 128256
llama_model_loader: - kv  12:                  llama.rope.dimension_count u32              = 128
llama_model_loader: - kv  13:                       tokenizer.ggml.model str              = gpt2
llama_model_loader: - kv  14:                      tokenizer.ggml.tokens arr[str,128256]  = ["!", "\"", "#", "$", "%", "&", "'", ...
llama_model_loader: - kv  15:                  tokenizer.ggml.token_type arr[i32,128256]  = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
llama_model_loader: - kv  16:                      tokenizer.ggml.merges arr[str,280147]  = ["Ġ Ġ", "Ġ ĠĠĠ", "ĠĠ ĠĠ", "...
llama_model_loader: - kv  17:                tokenizer.ggml.bos_token_id u32              = 128000
llama_model_loader: - kv  18:                tokenizer.ggml.eos_token_id u32              = 128001
llama_model_loader: - kv  19:                    tokenizer.chat_template str              = {% set loop_messages = messages %}{% ...
llama_model_loader: - kv  20:              general.quantization_version u32              = 2
llama_model_loader: - type  f32:   65 tensors
llama_model_loader: - type q4_K:  193 tensors
llama_model_loader: - type q6_K:   33 tensors
llm_load_vocab: special tokens definition check successful ( 256/128256 ).
llm_load_print_meta: format           = GGUF V3 (latest)
llm_load_print_meta: arch             = llama
llm_load_print_meta: vocab type       = BPE
llm_load_print_meta: n_vocab          = 128256
llm_load_print_meta: n_merges         = 280147
```

4.4 llama3量化模型进行API服务部署

4.4.1 本地服务部署

```
# 1. 启动 API 服务
./server -m ./models/ggml-meta-3-8b-Q4_K_M.gguf -c 512 -b 64 -n 256 -t 12 --host 0.0.0.0 --port 8080

# 2. curl命令服务测试

# (1) 英文测试
curl --request POST \
    --url http://localhost:8080/completion \
    --header "Content-Type: application/json" \
    --data '{"prompt": "system\nYou are a helpful assistant\nuser\nHello,do you know where is the Great Wall located ? \nassistant\n","n_predict": 128}'


#  (2) 中文测试
curl --request POST \
    --url http://localhost:8080/completion \
    --header "Content-Type: application/json" \
    --data '{"prompt": "system\n你是一位智能助手！\nuser\n你好，请问太阳系有多少行星？请用中文回答。\nassistant\n","n_predict": 128}' \
    --stream true
```

```
(base) root@ubuntu22:~#
(base) root@ubuntu22:~# curl --request POST \
    --url http://localhost:8080/completion \
    --header "Content-Type: application/json" \
    --data '{"prompt": "system\n你是一位智能助手！ \nuser\n你好，请问太阳系有多少行星？  \n
assistant\n","n_predict": 128}'
{"content":"🌞\n\n太阳系中有八大行星，分别是：\n1. 水星\n2. 金星\n3. 地球\n4. 火星\n5. 木
星\n6. 土星\n7. 天王星\n8. 冥王星\n\n另外，还有多个小行星和其他星体，如彗星、火星二星等。
\n\nNote: 冥王星的官方名称为 Pluto，但是在 2006 年，国际天文学联合会将冥王星降级为矮行星
。\n\nHope that helps! 🌟\nuser","id_slot":0,"stop":true,"model":"./models/ggml-meta-3-8b
-Q4_K_M.gguf","tokens_predicted":128,"tokens_evaluated":28,"generation_settings":{"n_ctx"
:512,"n_predict":256,"model":"./models/ggml-meta-3-8b-Q4_K_M.gguf","seed":4294967295,"tem
perature":0.800000011920929,"dynatemp_range":0.0,"dynatemp_exponent":1.0,"top_k":40,"top_
p":0.949999988079071,"min_p":0.05000000074505806,"tfs_z":1.0,"typical_p":1.0,"repeat_last
_n":64,"repeat_penalty":1.0,"presence_penalty":0.0,"frequency_penalty":0.0,"penalty_promp
t_tokens":[],"use_penalty_prompt_tokens":false,"mirostat":0,"mirostat_tau":5.0,"mirostat_
eta":0.10000000149011612,"penalize_nl":false,"stop":[],"n_keep":0,"n_discard":0,"ignore_e
os":false,"stream":false,"logit_bias":[],"n_probs":0,"min_keep":0,"grammar":"","samplers"
:["top_k","tfs_z","typical_p","top_p","min_p","temperature"]},"prompt":"system\n你是一位
智能助手！ \nuser\n你好，请问太阳系有多少行星？  \nassistant\n","truncated":false,"stopped_
eos":false,"stopped_word":false,"stopped_limit":true,"stopping_word":"","tokens_cached":1
55,"timings":{"prompt_n":28,"prompt_ms":1831.656,"prompt_per_token_ms":65.4162857142857,"
prompt_per_second":15.286713225627519,"predicted_n":128,"predicted_ms":42486.558,"predict
ed_per_token_ms":331.926234375,"predicted_per_second":3.0121715752858117}}(base) root@ubu
ntu22:~#
```

```python
# 3. 基于 requests 进行服务测试
# gguf_api_test.py

import requests
url = "http://localhost:8080/completion"
headers = {"Content-Type": "application/json"}
data = {
    "prompt": "system\n你是一位智能助手！ \nuser\n你好，请介绍一下埃隆马斯克。请用中文回答。
\nassistant\n",
    "n_predict": 128
}
response = requests.post(url, headers=headers, json=data)
result = response.json()["content"]

for line in result.split("\n"):
    print(line)
```

```
(base) root@ubuntu22:~#
(base) root@ubuntu22:~# python gguf_api_test.py
😊
埃隆·马斯克（Elon Musk）是世界-renowned 的企业家和创新者。他出生于南非，1990年代移民到加
拿大，并在2002年移民到美国。马斯克是 PayPal 的共同创始人之一，是 Space Exploration Techno
logies Corporation（SpaceX）的 CEO 和首席设计师，是 Tesla, Inc. 的 CEO 和首席设计师。

马斯克在多个领域都有出色的成就。他是 PayPal 的共同创始人之一，帮助该公司在2002年被 eBay
收购，获得了数亿
```

4.4.2 Docker服务部署

方式-1： CPU 推理

```
# docker执行命令

docker run -itd -p 8080:8080 -v /root/llama.cpp/models:/models --name llama3-cpu
ghcr.io/ggerganov/llama.cpp:server -m models/ggml-meta-3-8b-Q4_K_M.gguf -c 512 --host
0.0.0.0 --port 8080
```

```
(base) root@ubuntu22:~/llama.cpp/models# docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED
    STATUS         PORTS                                       NAMES
b70c1323ff5a   ghcr.io/ggerganov/llama.cpp:server   "/server -m models/g…"   4 seconds ag
o   Up 3 seconds   0.0.0.0:8080->8080/tcp, :::8080->8080/tcp   llama3-cpu
```

方式-2： GPU推理

```
docker run -itd -p 8080:8080 -v /root/llama.cpp/models:/models --name llama3-gpu --gpus
all ghcr.io/ggerganov/llama.cpp:server-cuda -m models/ggml-meta-3-8b-Q4_K_M.gguf -c 512 --
host 0.0.0.0 --port 8080 --n-gpu-layers 99
```

```
(base) root@ubuntu22:~/llama.cpp/models# docker ps
CONTAINER ID   IMAGE                                   COMMAND                  CREATED
         STATUS         PORTS                                       NAMES
2c5620b5195e   ghcr.io/ggerganov/llama.cpp:server-cuda   "/server -m models/g…"   5 minut
es ago   Up 5 minutes   0.0.0.0:8080->8080/tcp, :::8080->8080/tcp   llama3-gpu
```

5. 进行 llama3 模型中文语料微调

本周三更新

6. llama3核心源码解析

本周四更新

附录： >>> 踩坑记录 <<<

(1) docker gpu 创建容器报错

问题描述：

docker: Error response from daemon: could not select device driver "" with capabilities: [[gpu]].

解决方案:

```
# 添加NVIDIA包仓库
distribution=$(. /etc/os-release;echo $ID$VERSION_ID)
curl -s -L https://nvidia.github.io/nvidia-docker/gpgkey | sudo apt-key add -
curl -s -L https://nvidia.github.io/nvidia-docker/$distribution/nvidia-docker.list | sudo
tee /etc/apt/sources.list.d/nvidia-docker.list

# 安装nvidia-docker2并重启Docker服务
sudo apt-get update
sudo apt-get install -y nvidia-docker2
sudo systemctl restart docker
```