

加餐一：LLM 高性能推理技术方案

王炜/前腾讯云 CODING 高级架构师

目录

1 llama.cpp 量化和推理实战

2 vLLM 实战

1. llama.cpp 量化和推理实战

llama.cpp 简介

llama.cpp 是一个高性能大模型推理框架，主要特点：

- 纯 C++ 实现，无任何依赖项
- 支持 ARM（对于 Apple 芯片做了优化）、AVX、AVX2、AVX512 和 AMX 支持 x86 架构
- 支持 1.5 位、2 位、3 位、4 位、5 位、6 位和 8 位整数量化，可加快推理速度并减少内存使用
- 支持 GPU/CPU 混合推理，能加速超过总 VRAM 容量的模型

支持的模型

- LLaMA
- LLaMA 2
- LLaMA 3
- LLaMA Chinese
- Mistral 7B
- Deepseek
- Qwen
- ChatGLM

llama.cpp 实战

1. 准备模型文件

1. <https://huggingface.co/Qwen/Qwen2.5-0.5B-Instruct>

2. 安装 huggingface-cli 工具

1. `pip install -U "huggingface_hub[cli]"`

2. `export PATH=$PATH:/home/ubuntu/.local/bin`

3. 下载 GGUF 模型文件: `huggingface-cli download Qwen/Qwen2.5-0.5B-Instruct --local-dir /home/ubuntu/Qwen2.5-0.5B-Instruct`

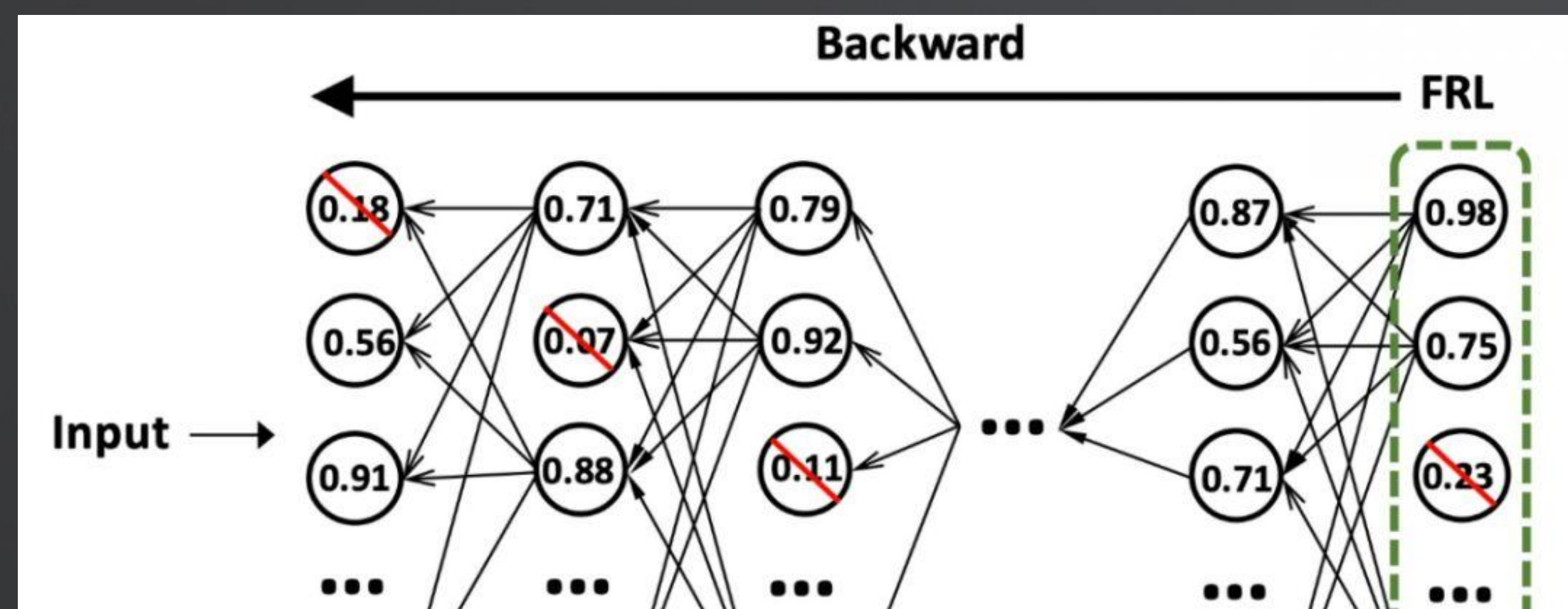
4. safetensors 格式的模型需要转化为 llama.cpp 的 GGUF 格式

safetensors 转化为 GGUF

1. 克隆 llama.cpp 仓库: `git clone https://github.com/ggerganov/llama.cpp`
2. 安装依赖: `pip install -r requirements.txt`
3. 将模型格式转化为 GGUF
 1. `cd llama.cpp && ./convert_hf_to_gguf.py /home/ubuntu/Qwen2.5-0.5B-Instruct`
 2. Model successfully exported to `/home/ubuntu/Qwen2.5-0.5B-Instruct/Qwen2.5-0.5B-Instruct-F16.gguf`
4. 对于大部分主流模型, 在 huggingface 都有 GGUF 格式可以下载, 例如:
 1. <https://huggingface.co/Qwen/Qwen2.5-0.5B-Instruct-GGUF/tree/main>

了解模型精度

- 精度是对于模型神经网络的权重值而言的
- 量化指的是把权重值从原始的高精度浮点（如 32-bit float, FP32）转化为更小的比特数表示，例如 8-bit (INT8)
- 量化过程并不是简单的把小数点的精度降低



量化后模型精度和 BPW 的关系

Bits per Weight (BPW) 表示每个权值（Weight）在量化后的存储大小，是按比特（bits）为单位的衡量指标，可以用来评估模型的存储率和推理效率，常见的精度如下所示，一般 GGUF 模型名中会包含精度值。

以 Llama2 7B 模型为例：

模型（Quantization）	BPW	解析
Q2_K	3.35	2-bit 基础方案，压缩率极高，但精度损失可能较大。
Q3_K_S	3.50	多了稍微复杂的策略，导致 BPW 略高，但仍接近 3-bit 水平。
Q3_K_M	3.91	中型变体，稍高的 BPW 支撑更精确的量化策略。
Q3_K_L	4.27	精度进一步提高，接近 4-bit 的存储消耗。
Q4_K_S	4.58	4-bit 量化的优化版本，仍在权重压缩基础上优化了分布。
Q4_K_M	4.84	牺牲较少位数换取了更高的精确度，适用于更多任务。
Q5_K_S	5.52	接近 5.5-bit 的存储，精确度较高。
Q5_K_M	5.68	稍微复杂，更高精度，存储效率和精度达成平衡。

模型精度的权衡

精度的差异体现了各个量化方案在压缩率与精度之间的权衡，通常：

1. L, S, M 表示量化细节的类型变体（小型、中型、大型），体现量化策略的差异。
2. 较低的 BPW（如 Q2_K, Q3_K）：适合存储敏感任务，比如在存储受限的边缘（移动）设备上运行，但有较大的精度影响。
3. 较高的 BPW（如 Q5_K, Q6_K）：适合需要较高精度的任务，但对存储/计算资源有更多需求。

模型量化一般实践

1. 原始模型一般是 32-bit (FP32) 精度

2. 常见的量化实践通常选择将模型从 (32-bit) (即FP32) 量化到 (16-bit) (如FP16、BF16) 或者 (8-bit) (如INT8)

3. 从 32 bit 到 16 bit: 模型效果基本保持

1. 平滑性和冗余性：大多数神经网络权重本身具有一定程度的冗余性
2. BF16 的灵活性：保留了与 FP32 相同的8位指数部分，而只减少了尾数（精确度部分）

4. 从 32 bit 到 8 bit: 进一步降低计算和存储成本，适合推理

1. 量化方法将浮点数（如 FP32）映射到一个较小的整数范围（如 INT8 $[-128, 127]$ ），每个权重或激活值被均匀地离散到该范围
2. 推理阶段对精度要求较低
3. 量化到 INT8 后，每个权重的存储需求减少为原来的1/4，可以显著提升推理速度

量化原理

量化过程分成两步：缩放和映射

1. 分析数据分布：FP32 浮点值分布在 $[-2.0, 2.4]$ ，INT8 整数范围是 $[-128, 127]$

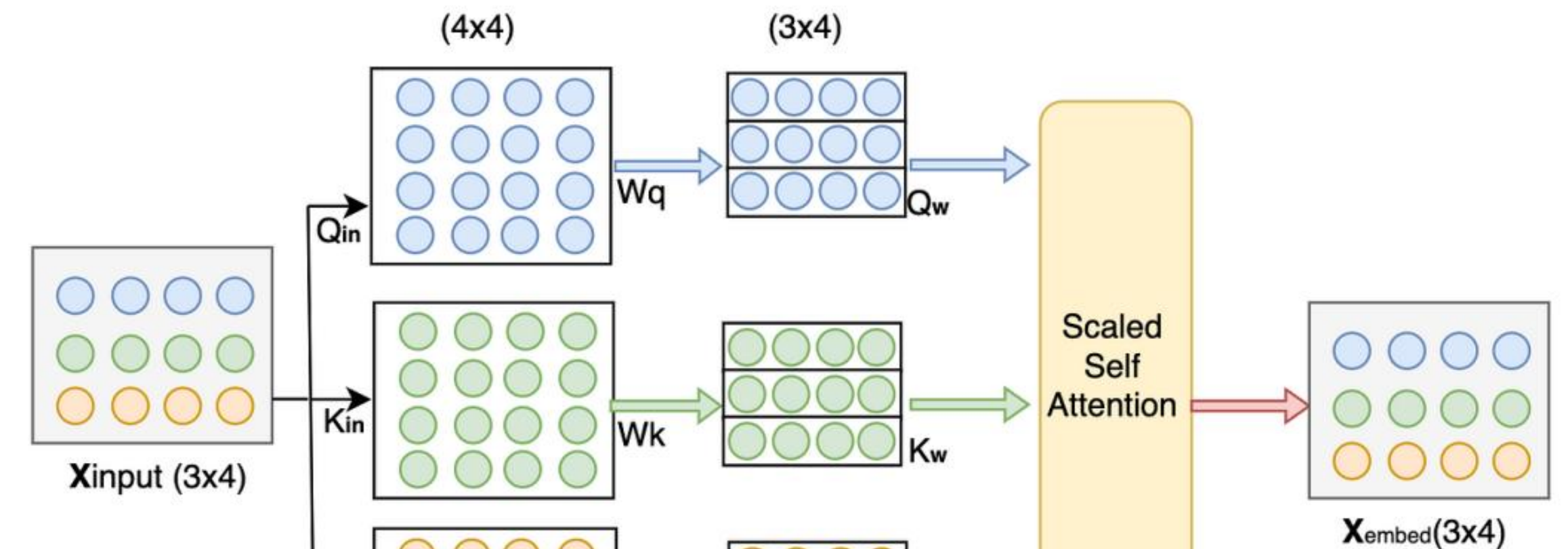
2. 计算缩放因子：
$$\text{Scale Factor} = \frac{\text{FP32值范围}}{\text{INT8值范围}}$$

$$\text{Scale Factor} = \frac{2.4 - (-2.0)}{127 - (-128)} = \frac{4.4}{255} \approx 0.01725$$

3. 缩放和映射，例如对于 $0.9125 \rightarrow 40$ 的转化过程

$$\text{INT8值} = \text{round}\left(\frac{0.9125 - 0}{0.01725}\right) = \text{round}(52.9) = 40$$

Self Attention Mechanism - Standard Transformer Architecture



Original Weight (FP32)

0.9125	-1.9589	-0.249	-0.6999
1.5274	-1.3953	0.1922	0.0445
1.4998	1.3379	0.2339	0.8801
-1.6090	2.3914	-0.1345	-0.9948

Maps (FP32: INT8)

Quantized Weight (INT8)

40	-128	-28	-54
77	-95	-2	-10
75	65	1	39
-107	127	-21	-71

- Each single value allocates: **32 bits in memory**
- Total weight parameters: $4 \times 4 = 16$
- **Memory allocations:** $16 \times 32 \text{ bit} = 512 \text{ bits}$

- Each single value allocates: **8 bits in memory**
- Total weight parameters: $4 \times 4 = 16$
- **Memory allocations:** $16 \times 8 \text{ bit} = 128 \text{ bits}$

llama.cpp 量化实战

1. 给虚拟机安装 CUDA 驱动

1. `wget https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2204/x86_64/cuda-keyring_1.1-1_all.deb`
2. `sudo dpkg -i cuda-keyring_1.1-1_all.deb`
3. `sudo apt-get update`
4. `sudo apt-get -y install cuda-toolkit-12-6`
5. `sudo apt-get install -y nvidia-open`
6. `sudo apt-get install -y cuda-drivers`
7. `export PATH=$PATH:/usr/local/cuda/bin`

llama.cpp 量化实战

1. 编译准备

1. <https://github.com/ggerganov/llama.cpp/blob/master/docs/build.md>

2. 推荐使用 cmake 编译

1. `sudo apt-get install software-properties-common`
2. `sudo add-apt-repository ppa:george-edison55/cmake-3.x`
3. `sudo apt-get update`
4. `sudo apt-get install cmake`

2. 编译 llama.cpp

1. `cmake -B build_cuda -DLLAMA_CUDA=ON # 编译 CUDA 版本 (GPU 推理)`
2. `cmake --build build_cuda --config Release -j 12 -t llama-cli llama-server llama-quantize # 12 个线程加速编译, 只编译我们需要的组件`

llama.cpp 量化实战

1. cd build_cuda/bin

2. 将模型量化为 8-bit

- 1. ./llama-quantize /home/ubuntu/Qwen2.5-0.5B-Instruct-GGUF/Qwen2.5-0.5B-Instruct-F16.gguf /home/ubuntu/Qwen2.5-0.5B-Instruct-GGUF/Qwen2.5-0.5B-Instruct-Q8_0.gguf Q8_0

3. 模型量化后


- 1. 942.43 MB -> 500.79 MB

```
quantize.ts
1  Allowed quantization types:
2      2 or Q4_0 : 4.34G, +0.4685 ppl @ Llama-3-8B
3      3 or Q4_1 : 4.78G, +0.4511 ppl @ Llama-3-8B
4      8 or Q5_0 : 5.21G, +0.1316 ppl @ Llama-3-8B
5      9 or Q5_1 : 5.65G, +0.1062 ppl @ Llama-3-8B
6     19 or IQ2_XXS : 2.06 bpw quantization
7     20 or IQ2_XS : 2.31 bpw quantization
8     28 or IQ2_S : 2.5 bpw quantization
9     29 or IQ2_M : 2.7 bpw quantization
10    24 or IQ1_S : 1.56 bpw quantization
11    31 or IQ1_M : 1.75 bpw quantization
12    36 or TQ1_0 : 1.69 bpw ternarization
13    37 or TQ2_0 : 2.06 bpw ternarization
14    10 or Q2_K : 2.96G, +3.5199 ppl @ Llama-3-8B
15    21 or Q2_K_S : 2.96G, +3.1836 ppl @ Llama-3-8B
16    23 or IQ3_XXS : 3.06 bpw quantization
17    26 or IQ3_S : 3.44 bpw quantization
18    27 or IQ3_M : 3.66 bpw quantization mix
19    12 or Q3_K : alias for Q3_K_M
20    22 or IQ3_XS : 3.3 bpw quantization
21    11 or Q3_K_S : 3.41G, +1.6321 ppl @ Llama-3-8B
22    12 or Q3_K_M : 3.74G, +0.6569 ppl @ Llama-3-8B
23    13 or Q3_K_L : 4.03G, +0.5562 ppl @ Llama-3-8B
24    25 or IQ4_NL : 4.50 bpw non-linear quantization
25    30 or IQ4_XS : 4.25 bpw non-linear quantization
26    15 or Q4_K : alias for Q4_K_M
27    14 or Q4_K_S : 4.37G, +0.2689 ppl @ Llama-3-8B
28    15 or Q4_K_M : 4.58G, +0.1754 ppl @ Llama-3-8B
29    17 or Q5_K : alias for Q5_K_M
30    16 or Q5_K_S : 5.21G, +0.1049 ppl @ Llama-3-8B
31    17 or Q5_K_M : 5.33G, +0.0569 ppl @ Llama-3-8B
32    18 or Q6_K : 6.14G, +0.0217 ppl @ Llama-3-8B
33     7 or Q8_0 : 7.96G, +0.0026 ppl @ Llama-3-8B
34    33 or Q4_0_4_4 : 4.34G, +0.4685 ppl @ Llama-3-8B
35    34 or Q4_0_4_8 : 4.34G, +0.4685 ppl @ Llama-3-8B
36    35 or Q4_0_8_8 : 4.34G, +0.4685 ppl @ Llama-3-8B
37     1 or F16 : 14.00G, +0.0020 ppl @ Mistral-7B
38    32 or BF16 : 14.00G, -0.0050 ppl @ Mistral-7B
39     0 or F32 : 26.00G @ 7B
```


UI 量化界面

1. <https://huggingface.co/spaces/ggml-org/gguf-my-repo>, 无需配置

You must be logged in to use GGUF-my-repo.

 Sign in with Hugging Face

Create your own GGUF Quants, blazingly fast ⚡!

The space takes an HF repo as an input, quantizes it and creates a Public repo containing the selected quant under your HF user namespace.

Hub Model ID

Quantization Method

GGML quantization type

Q4_K_M

Use importance matrix for quantization.

☐ Use Imatrix Quantization

Create a private repo under your username.


☐ Private Repo

Shard the model using gguf-split.

☐ Split Model

Clear

Submit



llama.cpp 推理实战

- ./llama-cli 参数
 - -m: 指定一个本地模型
 - -i: 以交互模式运行
 - -n: 生成的 token 长度
 - -t: CPU 推理时要使用的线程数，建议设置为物理 CPU 核心数
 - -ngl: 使用 GPU 推理时设置，将模型的部分层装载到 GPU 加速推理
 - -p: 系统提示语
- 对话模式体验
 - ./llama-cli -m /home/ubuntu/Qwen2.5-0.5B-Instruct-GGUF/qwen2.5-0.5b-instruct-q4_k_m.gguf/qwen2.5-0.5b-instruct-q4_k_m.gguf -co -cnv -p "You are Qwen, created by Alibaba Cloud. You are a helpful assistant." -fa -ngl 256 -n 512

奇怪的现象

- 再次推理
 - `./llama-cli -m /home/ubuntu/qwen2.5-0.5b-instruct-q4_k_m.gguf -p "Once upon a time"`
 - 发现输出不会停止，为什么？
- 大型语言模型生成文本时，本质是通过当前提供的上下文预测下一个最可能的 token（单词、字符等更小单位）。这个过程是无状态的，模型对接下来应该生成的内容的唯一依据是输入的上下文或 prompt，而它并不会“主动”知道何时停止
- 为什么对话模式会主动停止？因为对我们隐藏了实现细节

如何让大模型知道什么时候停止输出？

- 生成一个特殊的终止标志（例如结束 token: eos token)
- 或者在输出过程中推测出接下来的角色切换到用户，不需要进一步生成

```
./llama-cli -m /home/ubuntu/qwen2.5-0.5b-instruct-q4_k_m.gguf \
-co -sp -i -if -p "<|im_start|>system\nYou are Qwen, created by Alibaba Cloud. You are a helpful
assistant.<|im_end|>\n" --in-prefix "<|im_start|>user\n" --in-suffix
"<|im_end|>\n<|im_start|>assistant\n" -fa -ngl 80 -n 512
```

```
<|im_start|>system
You are Qwen, created by Alibaba Cloud. You are a helpful assistant.<|im_end|>
<|im_start|>user
hi          用户输入
<|im_end|>  cli 追加的结束符
<|im_start|>assistant cli 追加的模型输出开始符
Hello! How can I assist you today?<|im_end|> 模型输出内容+结束符
<|im_start|>user 模型输出的开始符
```


一定要用 <|im_start|> <|im_end|> 吗?

- 不一定，只要你给模型特定的输出模式例子即可

```
./llama-cli -m /home/ubuntu/qwen2.5-0.5b-instruct-q4_k_m.gguf \  
  -n -1 \  
  -ngl 256 \  
  -t 12 \  
  --color \  
  -r "User:" \  
  --in-prefix " " \  
  -i \  
  -p \  
'User: Hi  
AI: 你好啊，我是你的助手，有什么可以帮你?  
User:'
```

- 缺点：正常输出内容里可能会包含 User 关键字，导致对话提前结束

llama.cpp 推理实战（一次性输出）

- 一次性输出，非对话模式
 - `./llama-cli -m /home/ubuntu/qwen2.5-0.5b-instruct-q4_k_m.gguf -co -sp -p "<|im_start|>system\nYou are Qwen, created by Alibaba Cloud. You are a helpful assistant.<|im_end|>\n<|im_start|>user\ngive me a short introduction to LLMs.<|im_end|>\n<|im_start|>assistant\n" -fa -ngl 80 -n 512`

能不能不这么费劲？

- 使用 llama.cpp 内置模板 `--chat-template` 参数
 - chatml
 - llama2/llama3
 - monarch
 - gemma
 - orion
 - deepseek
- `./llama-cli -m /home/ubuntu/Qwen2.5-0.5B-Instruct-GGUF/qwen2.5-0.5b-instruct-q4_k_m.gguf/qwen2.5-0.5b-instruct-q4_k_m.gguf -cnv --chat-template chatml -p '你是一个助理'`
 - https://qwen.readthedocs.io/zh-cn/latest/getting_started/concepts.html#control-tokens-chat-template

llama.cpp Server 实战

- 以 OpenAI 风格的接口对外提供服务
 - `./llama-server -m /home/ubuntu/qwen2.5-0.5b-instruct-q4_k_m.gguf --host "0.0.0.0" --port 8080 --ngl 256 --api-key "apikey"`
- 以 docker 的方式启动
 - `docker run -p 8080:8080 -v /path/to/models:/models --gpus all ghcr.io/ggerganov/llama.cpp:server-cuda -m models/7B/ggml-model.gguf -c 512 --host 0.0.0.0 --port 8080 --n-gpu-layers 99`
- 同理可以运行在 K8s 环境推理

性能测试

- 安装 Golang 和 K6 测试工具
 - `wget https://go.dev/dl/go1.23.3.linux-amd64.tar.gz`
 - `sudo tar -C /usr/local -xzf go1.23.3.linux-amd64.tar.gz`
 - `export PATH=$PATH:/usr/local/go/bin`
 - `export PATH=$(go env GOPATH)/bin:$PATH`
 - `go install go.k6.io/xk6/cmd/xk6@latest`
 - `xk6 build master --with github.com/phymbert/xk6-sse`

性能测试

- 下载 LLM 基准测试数据集

- wget

```
https://huggingface.co/datasets/anon8231489123/ShareGPT_Vicuna_unfiltered/resolve/main/ShareGPT_V3_unfiltered_cleaned_split.json
```

- 拷贝 aiops/extra/script/script.js 基准测试脚本到 CVM
- 运行基准测试：1 分钟内有 8 个并发用户进行 10 次对话请求
 - `./k6 run script.js --duration 1m --iterations 10 --vus 8`
 - 注意重启 llama-server，不需要 apikey，否则无法测试

性能测试报告

- http_req_duration: HTTP 请求的总体耗时，包括从发出请求到收到服务器完整响应的时间（也就是一次对话结束的时间）
- llamacpp_completion_tokens: 模型生成的 token 数量
- llamacpp_completion_tokens_total_counter: 生成的总 Token 数，以及每秒生成的 Token 数
- llamacpp_completions_truncated_rate: 表示生成是否因为达到最大长度 max_tokens 而被截断的比例
- sse_event: 每秒接收的 SSE（服务端事件）消息数

```
data_received.....: 452 kB 41 kB/s
data_sent.....: 6.0 kB 538 B/s
http_req_duration.....: avg=5.52s min=796.41ms med=5.85s max=10.14s p(90)=9.41s p(95)=9.78s
http_req_sending.....: avg=1.36s min=3.78ms med=6.68ms max=5s p(90)=5s p(95)=5s
http_reqs.....: 10 0.895681/s
iteration_duration.....: avg=5.52s min=796.55ms med=5.85s max=10.14s p(90)=9.42s p(95)=9.78s
iterations.....: 10 0.895681/s
llamacpp_completion_tokens.....: avg=214.8 min=45 med=177.5 max=512 p(90)=512 p(95)=512
llamacpp_completion_tokens_total_counter....: 2148 192.392186/s
llamacpp_completions_truncated_rate.....: 0.00% 0 out of 0
llamacpp_prompt_tokens.....: avg=80.4 min=28 med=48.5 max=293 p(90)=133.7 p(95)=213.35
llamacpp_prompt_tokens_total_counter.....: 804 72.012718/s
sse_event.....: 2159 193.377435/s
vus.....: 1 min=1 max=8
vus_max.....: 8 min=8 max=8
```

结论：平均 5.5s 响应完成，性能表现不错，可以考虑继续增加并发测试

2. vLLM 实战

vLLM 介绍

vLLM 特色：

- 支持单 GPU、单节点多 GPU、多节点多 GPU 并行推理
 - 模型太大无法放入节点中的单个 GPU
 - 模型太大无法放入单个节点的 GPU
- 使用 **PagedAttention** 高效管理 GPU 显存
 - <https://blog.vllm.ai/2023/06/20/vllm.html>
- 高吞吐量的服务
- 流式输出
- 高效的多流并行处理能力
- 生产级后端推理服务

PagedAttention

原理：

- 在 Transformer 的推理过程中，会保存中间计算状态（如 key 和 value 矩阵）
- 一般方法是将这些存储为一个连续的大矩阵，这种方法会导致浪费大量的显存
- PagedAttention 将 key 和 value 矩阵按块分割成“小页”（pages），并以分页的形式存储，提高了显存使用率

优势：

- 通过分页机制，vLLM 很好地支持长上下文推理，而不会因为显存限制导致推理失败
- 在同时处理多个用户的生成任务时，vLLM 能够灵活调整页面以应对动态变化的上下文和多用户场景
- 以上优势使得 vLLM 特别适合处理大语言模型推理中的长上下文和多用户生成任务

支持的模型

- text-generation: https://docs.vllm.ai/en/latest/models/supported_models.html#text-generation
- text-embedding: https://docs.vllm.ai/en/latest/models/supported_models.html#text-embedding
- 多模态模型: https://docs.vllm.ai/en/latest/models/supported_models.html#multimodal-language-models

安装

要求：

- 操作系统：Linux
- Python：3.9 – 3.12
- GPU：V100、T4、RTX20xx、A100、L4、H100 及以上

安装：

```
$ pip install vllm
```

实战： 离线批量推理

```
batch.py
1  from vllm import LLM, SamplingParams
2
3  prompts = [
4      "你好, 你是谁? ",
5      "你在干什么? ",
6      "你有智商吗? ",
7      "你 IQ 多少? ",
8  ]
9
10 sampling_params = SamplingParams(temperature=0.8, top_p=0.95, max_tokens=1024)
11
12 model_path = "/home/ubuntu/Qwen2.5-0.5B-Instruct"
13
14 llm = LLM(model=model_path, dtype="half")
15
16 outputs = llm.generate(prompts, sampling_params)
17
18 for output in outputs:
19     prompt = output.prompt
20     generated_text = output.outputs[0].text
21     print(f"Prompt: {prompt!r}, Generated text: {generated_text!r}")
22
```


实战：启动推理服务

- 启动推理服务：vllm serve /home/ubuntu/Qwen2.5-0.5B-Instruct --host='0.0.0.0' --dtype=half --port=8080
 - 可添加 --api-key 指定密钥
- 以 OpenAI 风格访问推理服务：

```
request.py
1 import openai
2
3 client = openai.OpenAI(
4     base_url="http://localhost:8080/v1",
5     api_key="apikey",
6 )
7
8 completion = client.chat.completions.create(
9     model="/home/ubuntu/Qwen2.5-0.5B-Instruct", # 跟启动时保持一致
10    messages=[
11        {
12            "role": "system",
13            "content": "你是一个 AI 助手",
14        },
15        {"role": "user", "content": "你是谁? "},
16    ],
17 )
18
19 print(completion.choices[0].message)
```

Vision 多模态推理

- Vision 多模态：
 - `vllm serve microsoft/Phi-3.5-vision-instruct --task generate --trust-remote-code --max-model-len 4096 --limit-mm-per-prompt image=2`
- <https://docs.vllm.ai/en/latest/models/vlm.html#openai-vision-api>

Embedding 模式

- Embedding 模式：
 - `vllm serve TIGER-Lab/VLM2Vec-Full --task embedding --trust-remote-code --max-model-len 4096 --chat-template examples/template_vlm2vec.jinja`
- <https://docs.vllm.ai/en/latest/models/vlm.html#chat-embeddings-api>

Function Calling 函数调用

- https://docs.vllm.ai/en/latest/getting_started/examples/openai_chat_completion_client_with_tools.html

使用 Lora

- <https://docs.vllm.ai/en/v0.6.2/models/lora.html>
- 在启动指定 Lora
- 动态 Lora（在客户端请求时指定）

vLLM 性能优化

- 使用 `gpu_memory_utilization` 参数提高内存利用率
- 降低 `max_num_seqs` (默认 256) 或 `max_num_batched_tokens` (默认 512) , 减少请求并发数
- 使用 `tensor_parallel_size`, 将模型分片到多个 GPU (设备需要有多个 GPU, 例如2)

实验性功能:

启用 `enable_chunked_prefill` 分块预填充

```
llm = LLM(model="meta-llama/Llama-2-7b-hf", enable_chunked_prefill=True)
```

可以改进推理速度

THANKS