LLMs部署与评测 v2

服务器

环境配置

已安装

nvidia-550.90.07

CUDA 12.4

Python 3.10.12

Miniconda 3

安装NVIDIA驱动程序

nvidia-smi命令不可用，需要安装NVIDIA驱动程序

```
sudo apt update  #更新apt软件包存储库
sudo apt upgrade
apt search nvidia-driver  #输出显示适用于GPU的可用驱动程序列表
sudo apt install nvidia-driver-550  #选择要安装的驱动程序
ls /usr/src | grep nvidia #查询驱动版本
```

安装cuda

查看支持的最高cuda版本

```
nvidia-smi
```

```
root@bms-3qab-0001-0002:/home/packages# nvidia-smi
Wed Aug  7 15:08:01 2024
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 550.90.07              Driver Version: 550.90.07      CUDA Version: 12.4     |
|-----------------------------------------+------------------------+----------------------+
```

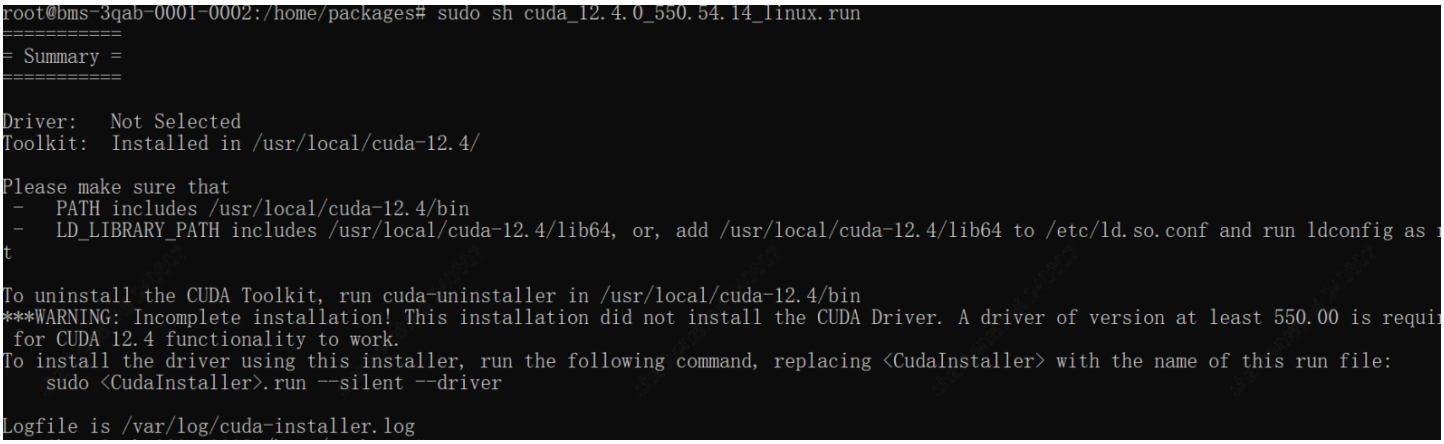
官网选择对应版本https://developer.nvidia.com/cuda-toolkit-archive

架构和Ubuntu版本可以通过命令查询：

```
uname -a
lsb_release -a
```

安装

```
wget https://developer.download.nvidia.com/compute/cuda/12.4.0/local_installers/cuda_12.4.0_550.54.14_linux.run
sudo sh cuda_12.4.0_550.54.14_linux.run
```



配置环境变量

```
vim ~/.bashrc
export PATH=/usr/local/cuda/bin:$PATH
export LD_LIBRARY_PATH=/usr/local/cuda/lib64:$LD_LIBRARY_PATH
source ~/.bashrc
```

查看cuda版本

```
nvcc -V
which nvcc  #查询安装目录
```

安装conda

miniconda只能在命令行中使用，比anaconda占用系统空间更小。从镜像源选择对应miniconda

https://mirrors.tuna.tsinghua.edu.cn/anaconda/miniconda/?C=M&O=A

选择Miniconda3-latest-Linux-x86_64.sh安装并激活

```
# wget 加网址，中间可以加-c参数，断点续传
wget https://mirrors.tuna.tsinghua.edu.cn/anaconda/miniconda/Miniconda3-latest-Linux-x86_64.sh
bash Miniconda3-latest-Linux-x86_64.sh
# 配置环境变量
vim ~/.bashrc
# 在最末尾添加以下内容，usrname替换为你自己的用户名，或者地址换为你的conda地址
export PATH="/home/usrname/miniconda3/bin:$PATH"
# 激活环境变量
source ~/.bashrc
# 查看是否安装成功
conda --help
```

配置conda镜像

```
# 下面这三行配置官网的channel地址
conda config --add channels r
conda config --add channels conda-forge
conda config --add channels bioconda
#下面这四行配置清华大学的conda的channel地址，国内用户推荐
conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main/
conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/bioconda/
conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/conda-forge/
conda config --set show_channel_urls yes
# 查看配置镜像结果
cat ~/.condarc
```

创建虚拟环境

```
conda create --name your_env_name python=xxx
```

目前虚拟环境

环境名	功能
llms	vllm部署
evalscope	压力测试

opencompass	能力测试

下载模型参数

本地下载，上传至服务器

模型参数路径

模型	版本	参数量	路径
Llama3.1	Instruct	8B	/home/llms/models/llama3.1-8B-Instruct
Qwen2		0.5B	/home/llms/models/qwen2-0.5B
		7B	/home/llms/models/qwen2-7B
	Instruct	7B	
	Instruct	57B	/home/llms/models/qwen2-57B-A14B-Instruct

vLLM实践

安装vllm库

一般先用conda安装，无法安装再使用pip

```
conda activate llms
# 进入虚拟环境后安装
pip3 install -i https://pypi.tuna.tsinghua.edu.cn/simple vllm
```

注意：pip安装的包可能用conda list命令无法查看，要用pip list

离线部署（Offline Batched Inference）

本地IDE连接服务器，Pycharm用SFTP方式连接服务器不稳定，修改/etc/ssh/sshd_config

```
Subsystem sftp /usr/libexec/openssh/sftp-server
# 改为
Subsystem sftp internal-sftp
```

用Python调用服务器上下载的模型，运行以下代码。调用llm.generate以生成输出，它将输入提示添加到vLLM引擎的等待队列中，并执行vLLM引擎以高吞吐量生成输出。输出以RequestOutput对象列表的形式返回。

推理

```python
from vllm import LLM, SamplingParams
prompts = [
    "Hello, my name is",
    "The president of the United States is",
    "The capital of France is",
    "The future of AI is",
]
sampling_params = SamplingParams(temperature=0.8, top_p=0.95)
llm = LLM(model="/home/llms/models/llama3.1-8B-Instruct")
outputs = llm.generate(prompts, sampling_params)
for output in outputs:
    prompt = output.prompt
    generated_text = output.outputs[0].text
    print(f"Prompt: {prompt!r}, Generated text: {generated_text!r}")
```

Llama3.1-8B在temperature=0.8, top_p=0.95设定下的生成结果为

Prompt	Generated text
Hello, my name is	Helen and I am a devoted animal lover and owner of Angels of Mercy. I
The president of the United States is	often referred to as the "leader of the free world." As the head of
The capital of France is	known for its iconic landmarks, fashion, art, and cuisine. Here are some
The future of AI is	not just about technology, but about how we understand the world and our place in

服务器部署（OpenAI-Compatible Server）

vLLM可以作为实现OpenAI API协议的**服务器**进行部署，默认在http://localhost:8000启动服务器。可以使用--host和--port参数指定地址。目前服务器**一次只托管一个模型**，并实现了list models, create chat completion, create completion三个endpoints。

启动服务

```
# 查看命令参数
vllm serve -h
# 以下启动方式二选一
# 后台启动
CUDA_VISIBLE_DEVICES=3 nohup vllm serve /home/llms/models/llama3.1-8B-Instruct --served-model-name llama3.1-8B-Instruct > deployment.log 2>&1 &
CUDA_VISIBLE_DEVICES=3 nohup vllm serve /home/llms/models/qwen2-7B --served-model-name qwen2-7B > deployment.log 2>&1 &
# Python启动
CUDA_VISIBLE_DEVICES=0 python -m vllm.entrypoints.api_server --model /home/llms/models/llama3.1-8B-Instruct --served-model-name llama3.1-8B-Instruct
CUDA_VISIBLE_DEVICES=0 python -m vllm.entrypoints.openai.api_server --model /home/llms/models/llama3.1-8B-Instruct --served-model-name llama3.1-8B-Instruct

# 定期清空部署日志
truncate -s 0 deployment.log
# 或者
> deployment.log
```

--gpu-memory-utilization GPU_MEMORY_UTILIZATION

The fraction of GPU memory to be used for the model executor, which can range from 0 to 1. For example, a value of 0.5 would imply 50% GPU memory utilization. If unspecified, will use the default value of 0.9.

Route	/v1/models	/v1/completions	/v1/chat/completions	/v1/embeddings
Methods	GET	POST	POST	POST

查看模型

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

响应结果

```
{"object":"list","data":[{"id":"llama3.1-8B-
```

```
Instruct","object":"model","created":1723686286,"owned_by":"vllm","root":"llama
3.1-8B-Instruct","parent":null,"max_model_len":131072,"permission":
[{"id":"modelperm-
a1d8002e8831402884b4fd0ed5bfd29c","object":"model_permission","created":1723686
286,"allow_create_engine":false,"allow_sampling":true,"allow_logprobs":true,"al
low_search_indices":false,"allow_view":true,"allow_fine_tuning":false,"organiza
tion":"*","group":null,"is_blocking":false}]}]}
```


Completion请求

```
curl http://localhost:8000/v1/completions \
    -H "Content-Type: application/json" \
    -d '{
        "model": "llama3.1-8B-Instruct",
        "prompt": "San Francisco is a",
        "max_tokens": 7,
        "temperature": 0
    }'
```

响应结果


```
{"id":"cmpl-
8084f8ad8a134a34b17b027718096d44","object":"text_completion","created":17236863
19,"model":"llama3.1-8B-Instruct","choices":[{"index":0,"text":" top tourist
destination, and for
good","logprobs":null,"finish_reason":"length","stop_reason":null}],"usage":
{"prompt_tokens":5,"total_tokens":12,"completion_tokens":7}}
```


OpenAI Chat API

```
curl http://localhost:8000/v1/chat/completions \
    -H "Content-Type: application/json" \
    -d '{
        "model": "llama3.1-8B-Instruct",
        "messages": [
            {"role": "system", "content": "You are a helpful assistant."},
            {"role": "user", "content": "Who won the world series in 2020?"}
        ]
    }'
```

响应结果


```
{"id":"chat-ceaa585f215c47cdb69dd9f940457401","object":"chat.completion","created":1723686362,"model":"llama3.1-8B-Instruct","choices":[{"index":0,"message":{"role":"assistant","content":"The Los Angeles Dodgers won the 2020 World Series. They defeated the Tampa Bay Rays in the series 4 games to 2. It was the Dodgers' first World Series title since 1988.","tool_calls":[]},"logprobs":null,"finish_reason":"stop","stop_reason":null}],"usage":{"prompt_tokens":51,"total_tokens":94,"completion_tokens":43}}
```


Ollama实践

注意事项

- Ollama在加载模型时，如果GPU有空间，则会优先加载到GPU上，GPU不够了会加载到CPU（内存）中。
- ollama run命令如果本地没有模型参数会自动下载，可能是**量化**版本，安装最好去官网 https://ollama.com/library查看

安装配置

```
# 安装ollama
sudo curl -fsSL https://ollama.com/install.sh | sh
# 配置文件
sudo vi /etc/systemd/system/ollama.service
# 在文件中找到[Service]，指定gpu
Environment="CUDA_VISIBLE_DEVICES=2"

# 在命令行中运行模型，如果本地没有模型参数会自动下载，默认目录/usr/share/ollama/.ollama/models
ollama run llama3.1
# 显示已下载模型
ollama list
# 查看模型CPU/GPU占用情况
ollama ps
```

Ollama在运行时，自动在本地的11434端口提供Rest服务，可以通过指定模型并传入输入来获取结果。Ollama支持普通风格的API以及OpenAI风格的API。

调用API

```
curl http://localhost:11434/v1/models
curl http://localhost:11434/v1/chat/completions -d '{
  "model": "qwen2",
  "messages": [
    { "role": "user", "content": "why is the sky blue?" }
  ]
}'
```

```python
import requests
import json

def send_message_to_ollama(message, port=11434):
    url = f"http://localhost:{port}/api/chat"
    payload = {
        "model": "mistral",
        "messages": [{"role": "user", "content": message}]
    }
    response = requests.post(url, json=payload)
    if response.status_code == 200:
        response_content = ""
        for line in response.iter_lines():
            if line:
                response_content += json.loads(line)["message"]["content"]
        return response_content
    else:
        return f"Error: {response.status_code} - {response.text}"

if __name__ == "__main__":
    user_input = "why is the sky blue?"
    response = send_message_to_ollama(user_input)
    print("Ollama's response:")
    print(response)
```

压力测试

测试工具

https://github.com/modelscope/evalscope/tree/main

评价指标

Metrics	Description
Total requests	总请求数
Succeed requests	成功请求数
Failed requests	失败请求数
Average QPS	平均每秒处理的请求数
Average latency	平均全部请求的延时
Throughput	每秒输出的token数量
Average time to first token	平均生成第一个token的时间
Average input tokens per request	平均每个请求的输入token数量
Average output tokens per request	平均每个请求的输出token数量
Average time per output token	平均输出每个token的时间
Average package per request	平均每个请求的包数
Average package latency	平均每包延迟
Percentile of time to first token (p50, ..., p99)	Percentile of time to first token
Percentile of request latency (p50, ..., p99)	Percentile of request latency

数据集

数据集	语言	大小（条）	长度	link
openqa	中文	3.29k	短	https://huggingface.co/datasets/Hello-SimpleAI/HC3-Chinese/blob/main/open_qa.jsonl
LongAlpaca	英文	12k	长	https://huggingface.co/datasets/Yukang/LongAlpaca-12k/blob/main/LongAlpaca-12k.json

EvalScope评测框架

```
conda create -n evalscope python=3.10
conda activate evalscope
pip install evalscope
```

EvalScope perf压力测试

```
evalscope perf --help
usage: evalscope <command> [<args>] perf [-h] --model MODEL [--url URL] [--connect-timeout CONNECT_TIMEOUT] [--read-timeout READ_TIMEOUT] [-n NUMBER] [--parallel PARALLEL] [--rate RATE]
                                         [--log-every-n-query LOG_EVERY_N_QUERY] [--headers KEY1=VALUE1 [KEY1=VALUE1 ...]] [--wandb-api-key WANDB_API_KEY] [--name NAME] [--debug] [--tokenizer-path TOKENIZER_PATH]
                                         [--api API] [--max-prompt-length MAX_PROMPT_LENGTH] [--min-prompt-length MIN_PROMPT_LENGTH] [--prompt PROMPT] [--query-template QUERY_TEMPLATE] [--dataset DATASET]
                                         [--dataset-path DATASET_PATH] [--frequency-penalty FREQUENCY_PENALTY] [--logprobs] [--max-tokens MAX_TOKENS] [--n-choices N_CHOICES] [--seed SEED] [--stop STOP] [--stream]
                                         [--temperature TEMPERATURE] [--top-p TOP_P]

options:
  -h, --help            show this help message and exit
  --model MODEL         The test model name.
  --url URL
  --connect-timeout CONNECT_TIMEOUT
                        The network connection timeout
  --read-timeout READ_TIMEOUT
                        The network read timeout
  -n NUMBER, --number NUMBER
                        How many requests to be made, if None, will will send request base dataset or prompt.
  --parallel PARALLEL   Set number of concurrency request, default 1
  --rate RATE           Number of requests per second. default None, if it set to -1,then all the requests are sent at time 0. Otherwise, we use Poisson process to synthesize the request arrival times. Mutual exclusion
                        with parallel
  --log-every-n-query LOG_EVERY_N_QUERY
                        Logging every n query.
  --headers KEY1=VALUE1 [KEY1=VALUE1 ...]
                        Extra http headers accepts by key1=value1 key2=value2. The headers will be use for each query.You can use this parameter to specify http authorization and other header.
  --wandb-api-key WANDB_API_KEY
```

```
                        The wandb api key, if set the metric will be saved to
wandb.
  --name NAME           The wandb db result name and result db name, default:
{model_name}_{current_time}
  --debug               Debug request send.
  --tokenizer-path TOKENIZER_PATH
                        Specify the tokenizer weight path, used to calculate
the number of input and output tokens,usually in the same directory as the
model weight.
  --api API             Specify the service api, current support
[openai|dashscope]you can define your custom parser with python, and specify
the python file path, reference api_plugin_base.py,
  --max-prompt-length MAX_PROMPT_LENGTH
                        Maximum input prompt length
  --min-prompt-length MIN_PROMPT_LENGTH
                        Minimum input prompt length.
  --prompt PROMPT       Specified the request prompt, all the query will use
this prompt, You can specify local file via @file_path, the prompt will be the
file content.
  --query-template QUERY_TEMPLATE
                        Specify the query template, should be a json string,
or local file,with local file, specified with @local_file_path,will will
replace model and prompt in the template.
  --dataset DATASET     Specify the dataset
[openqa|longalpaca|line_by_line]you can define your custom dataset parser with
python, and specify the python file path, reference dataset_plugin_base.py,
  --dataset-path DATASET_PATH
                        Path to the dataset file, Used in conjunction with
dataset. If dataset is None, each line defaults to a prompt.
  --frequency-penalty FREQUENCY_PENALTY
                        The frequency_penalty value.
  --logprobs            The logprobs.
  --max-tokens MAX_TOKENS
                        The maximum number of tokens can be generated.
  --n-choices N_CHOICES
                        How may chmpletion choices to generate.
  --seed SEED           The random seed.
  --stop STOP           The stop generating tokens.
  --stop-token-ids      Set the stop token ids.
  --stream              Stream output with SSE.
  --temperature TEMPERATURE
                        The sample temperature.
  --top-p TOP_P         Sampling top p.

```

openqa数据集评测

vllm评测

```
conda activate evalscope
# 对vllm服务器上的llama3.1-8B-Instruct进行评测,并发数为1
nohup evalscope perf --model 'llama3.1-8B-Instruct' --url 'http://127.0.0.1:8000/v1/chat/completions' --name 'llama3.1:8b-openqa-vllm' --dataset openqa --dataset-path '/home/llms/eval/datasets/HC3-Chinese/open_qa.jsonl' --read-timeout 120  --log-every-n-query 10 --wandb-api-key '6023b93bc77f64ec18780af37e6d972aba8ec5a7' --api openai --debug > vllm_eval.log 2>&1 &

# 并发测试，并发数通过--parallel指定
nohup evalscope perf --model 'llama3.1-8B-Instruct' --url 'http://127.0.0.1:8000/v1/chat/completions' --parallel 128 --name 'llama3.1:8b-openqa-vllm-p128' --dataset openqa --dataset-path '/home/llms/eval/datasets/HC3-Chinese/open_qa.jsonl' --read-timeout 120 --log-every-n-query 10 --wandb-api-key '6023b93bc77f64ec18780af37e6d972aba8ec5a7' --api openai --debug > vllm_eval.log 2>&1 &
```

ollama评测

```
# 对ollama服务器上的llama3.1-8B-Instruct进行评测,并发数为1
nohup evalscope perf --model 'llama3.1:8b-instruct-fp16' --url 'http://127.0.0.1:11434/v1/chat/completions' --name 'llama3.1:8b-openqa-ollama'  --dataset openqa --dataset-path '/home/llms/eval/datasets/HC3-Chinese/open_qa.jsonl' --read-timeout 120  --log-every-n-query 10 --wandb-api-key '6023b93bc77f64ec18780af37e6d972aba8ec5a7' --api openai --debug > ollama_eval.log 2>&1 &

# 并发测试
nohup evalscope perf --model 'llama3.1:8b-instruct-fp16' --url 'http://127.0.0.1:11434/v1/chat/completions' --parallel 64 --name 'llama3.1:8b-openqa-ollama-p64' --dataset openqa --dataset-path '/home/llms/eval/datasets/HC3-Chinese/open_qa.jsonl' --read-timeout 120  --log-every-n-query 10 --wandb-api-key '6023b93bc77f64ec18780af37e6d972aba8ec5a7' --api openai --debug > ollama_eval.log 2>&1 &
```

读取输出结果

```
import sqlite3
import base64
```

```
import pickle
import json
result_db_path = 'llama3.1-8B_openqa_vllm'
con = sqlite3.connect(result_db_path)
query_sql = "SELECT request, response_messages, prompt_tokens,
completion_tokens \
                FROM result WHERE success='True'"
# how to save
base64.b64encode(pickle.dumps(benchmark_data["request"])).decode("ascii"),
with con:
    rows = con.execute(query_sql).fetchall()
    if len(rows) > 0:
        for row in rows:
            request = row[0]
            responses = row[1]
            request = base64.b64decode(request)
            request = pickle.loads(request)
            responses = base64.b64decode(responses)
            responses = pickle.loads(responses)
            response_content = ''
            for response in responses:
                response = json.loads(response)
                print(response)
                response_content += response['choices'][0]['message']
['content']
            print('prompt: %s, tokens: %s, completion: %s, tokens: %s' %
(request['messages'][0]['content'], row[2], response_content, row[3]))
```

LongAlpaca-12k数据集评测

vllm评测

```
# 对vllm服务器上的llama3.1-8B-Instruct进行评测,并发数为1
nohup evalscope perf --model 'llama3.1-8B-Instruct' --url
'http://127.0.0.1:8000/v1/chat/completions' --name 'llama3.1:8b-longalpaca-
vllm' --dataset longalpaca --dataset-path '/home/llms/eval/datasets/LongAlpaca-
12k/LongAlpaca-12k.json' --read-timeout 120  --log-every-n-query 10 --wandb-
api-key '6023b93bc77f64ec18780af37e6d972aba8ec5a7' --api openai --debug >
vllm_eval.log 2>&1 &

#并发测试
nohup evalscope perf --model 'llama3.1-8B-Instruct' --url
'http://127.0.0.1:8000/v1/chat/completions' --parallel 128 --name 'llama3.1:8b-
longalpaca-vllm-p128' --dataset longalpaca --dataset-path
'/home/llms/eval/datasets/LongAlpaca-12k/LongAlpaca-12k.json' --read-timeout
```

```
120  --log-every-n-query 10 --wandb-api-key
'6023b93bc77f64ec18780af37e6d972aba8ec5a7' --api openai --debug >
vllm_eval.log 2>&1 &
```

ollama评测

```
# 对ollama服务器上的llama3.1-8B-Instruct进行评测,并发数为1
nohup evalscope perf --model 'llama3.1:8b-instruct-fp16' --url
'http://127.0.0.1:11434/v1/chat/completions' --name 'llama3.1:8b-longalpaca-
ollama' --dataset longalpaca --dataset-path
'/home/llms/eval/datasets/LongAlpaca-12k/LongAlpaca-12k.json' --read-timeout
120  --log-every-n-query 10 --wandb-api-key
'6023b93bc77f64ec18780af37e6d972aba8ec5a7' --api openai --debug >
ollama_eval.log 2>&1 &

# 并发测试
nohup evalscope perf --model 'llama3.1:8b-instruct-fp16' --url
'http://127.0.0.1:11434/v1/chat/completions' --parallel 16 --name 'llama3.1:8b-
longalpaca-ollama-p16' --dataset longalpaca --dataset-path
'/home/llms/eval/datasets/LongAlpaca-12k/LongAlpaca-12k.json' --read-timeout
120  --log-every-n-query 10 --wandb-api-key
'6023b93bc77f64ec18780af37e6d972aba8ec5a7' --api openai --debug >
ollama_eval.log 2>&1 &
```

压力测试实验结果

可能影响结果的变量：

模型层面：量化

数据层面：数据集大小、数据集长度

开发层面：部署框架、并发数(短文本上做)

压力测试评测结果

💡 Benchmarking summary:

Time taken for tests: 16789.335 seconds

Expected number of requests: 3293

Number of concurrency: 1

Total requests: 3293

Succeed requests: 3291

Failed requests: 2

Average QPS: 0.196

Average latency: 5.028

Throughput(average output tokens per second): 71.431

Average time to first token: 5.028

Average input tokens per request: 66.524

Average output tokens per request: 364.409

Average time per output token: 0.01400

Average package per request: 1.000

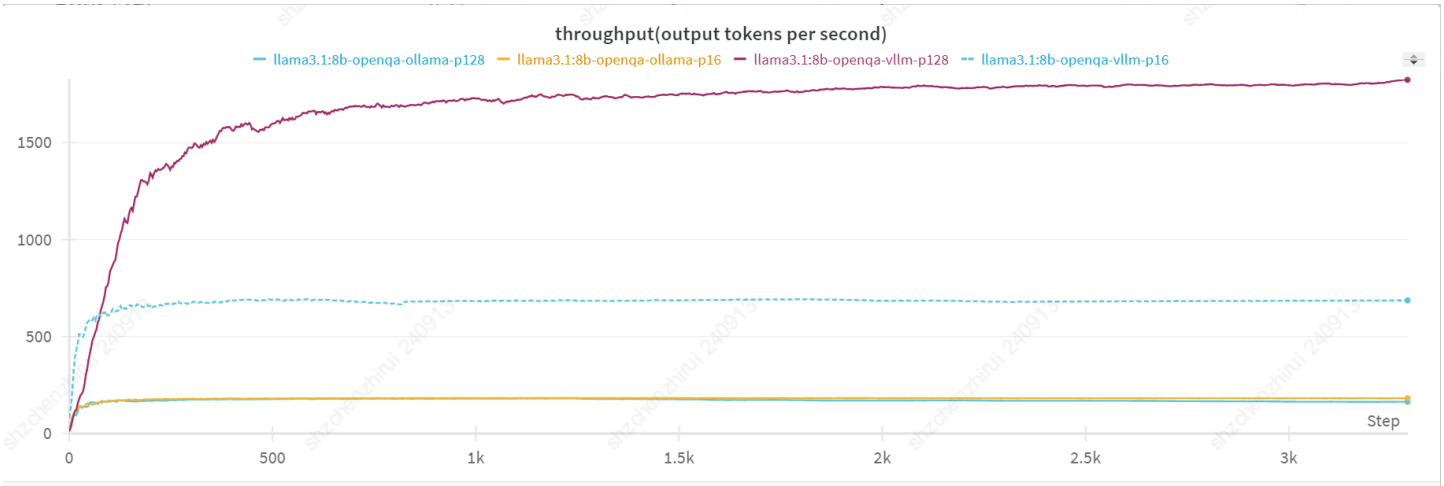
Average package latency: 5.028

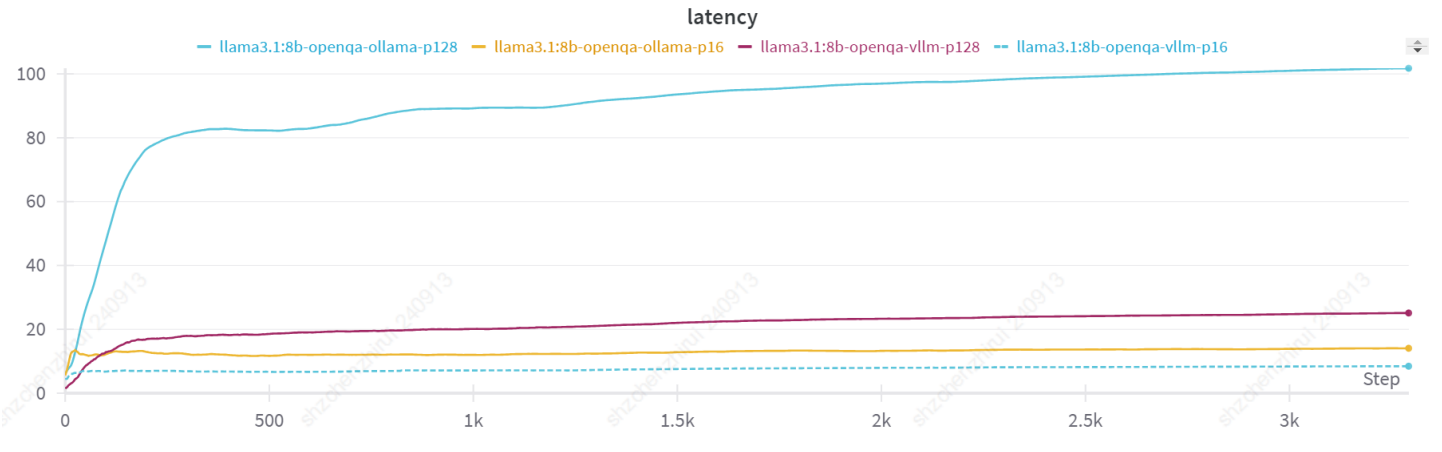
Percentile of time to first token:

- p50: 5.0402
- p66: 5.9084
- p75: 6.4606
- p80: 6.8322
- p90: 7.8796
- p95: 8.7338
- p98: 9.8858
- p99: 10.6578

Percentile of request latency:

- p50: 5.0402
- p66: 5.9084
- p75: 6.4606
- p80: 6.8322
- p90: 7.8796
- p95: 8.7338
- p98: 9.8858
- p99: 10.6578


throughput(output tokens per second)
llama3.1:8b-openqa-ollama-p128
llama3.1:8b-openqa-ollama-p16
llama3.1:8b-openqa-vllm-p128
llama3.1:8b-openqa-vllm-p16
1500
1000
500
0
Step
0
500
1k
1.5k
2k
2.5k
3k


latency
llama3.1:8b-openqa-ollama-p128
llama3.1:8b-openqa-ollama-p16
llama3.1:8b-openqa-vllm-p128
llama3.1:8b-openqa-vllm-p16
100
80
60
40
20
0
Step
0
500
1k
1.5k
2k
2.5k
3k