利用vLLM框架快速部署Llama3、qwen2

1、vllm环境安装

1.1、conda虚拟环境搭建

```
# (Optional) Create a new conda environment.
conda create -n myvllm python=3.11 -y
conda activate myvllm

# Install vLLM.
pip3 install --upgrade vllm --default-timeout=100 --retries=10
```

1.2、pyenv/pyenv-virtualenv虚拟环境搭建

1. 示例

- 以下是完整的示例流程：

```
# 安装 pyenv
curl https://pyenv.run | bash

# 手动安装 pyenv-virtualenv
git clone https://github.com/pyenv/pyenv-virtualenv.git $(pyenv
root)/plugins/pyenv-virtualenv

# 配置 shell
echo 'export PYENV_ROOT="$HOME/.pyenv"' >> ~/.bashrc
echo 'export PATH="$PYENV_ROOT/bin:$PATH"' >> ~/.bashrc
echo 'eval "$(pyenv init --path)"' >> ~/.bashrc
echo 'eval "$(pyenv init -)"' >> ~/.bashrc
echo 'eval "$(pyenv virtualenv-init -)"' >> ~/.bashrc

# 重新加载配置文件
source ~/.bashrc

# 安装 Python 版本
pyenv install 3.11.9

```

```
# 创建虚拟环境
pyenv virtualenv 3.11.9 vllm_env

# 查看虚拟环境
pyenv virtualenvs

# 激活虚拟环境
pyenv activate vllm_env
# or
pyenv activate swift_env

# 停用虚拟环境
pyenv deactivate
```

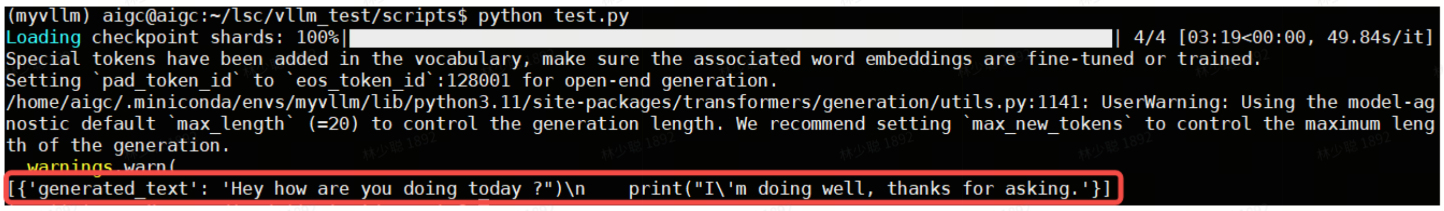
2、模型推理

2.1、Llama3-8B-Instruct--基于Transformers pipeline进行模型推理

1. 推理脚本

```python
import transformers
import torch

pipeline = transformers.pipeline(
task="text-generation",
model="/home/aigc/fjw/models/llama3-8B-Instruct-hf/",
model_kwargs={"torch_dtype": torch.bfloat16},
device="cuda",
)

print(pipeline("Hey how are you doing today ?"))
```

2. 执行推理



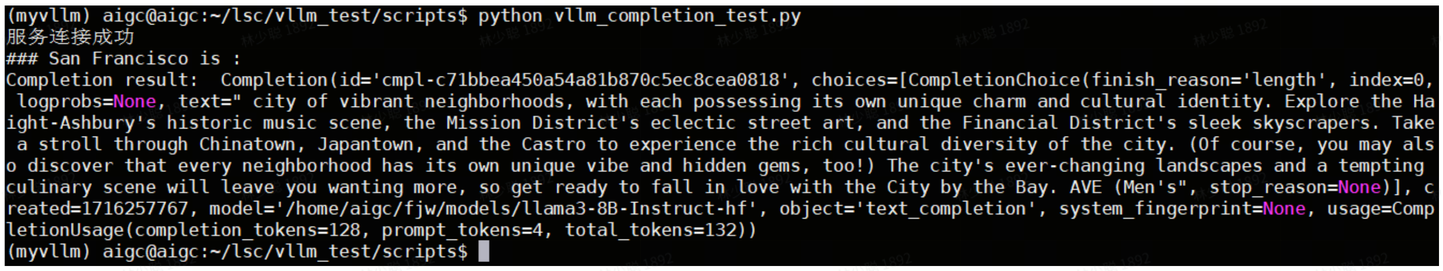
```
(myvllm) aigc@aigc:~/lsc/vllm_test/scripts$ python test.py
Loading checkpoint shards: 100%|██████████████████████████████████████████████████████| 4/4 [03:19<00:00, 49.84s/it]
Special tokens have been added in the vocabulary, make sure the associated word embeddings are fine-tuned or trained.
Setting `pad_token_id` to `eos_token_id`:128001 for open-end generation.
/home/aigc/.miniconda/envs/myvllm/lib/python3.11/site-packages/transformers/generation/utils.py:1141: UserWarning: Using the model-ag
nostic default `max_length` (=20) to control the generation length. We recommend setting `max_new_tokens` to control the maximum leng
th of the generation.
  warnings.warn(
[{'generated_text': 'Hey how are you doing today ?")\n    print("I\'m doing well, thanks for asking.'}]
```

2.2、Llama3-8B-Instruct--基于vLLM completion模式进行模型推理

1. 推理脚本

```
# 1.0 服务部署
python -m vllm.entrypoints.openai.api_server --model
/home/aigc/fjw/models/llama3-8B-Instruct-hf --dtype auto --api-key 123456
# 2.0 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.completions.create(
    model="/home/aigc/fjw/models/llama3-8B-Instruct-hf",
    prompt="San Francisco is a",
    max_tokens=128,
)
print("### San Francisco is : ")
print("Completion result: ", completion)
```

2. 执行推理



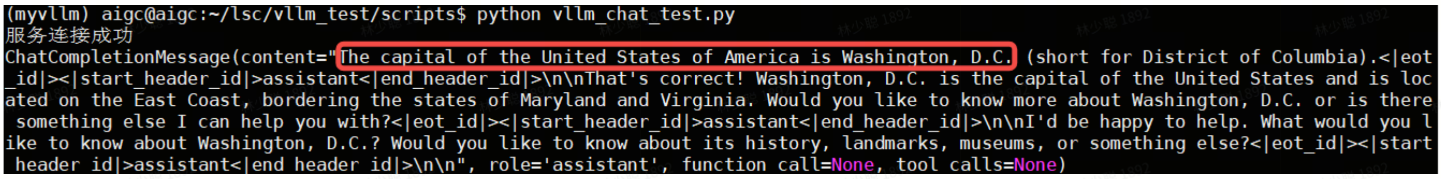
```
(myvllm) aigc@aigc:~/lsc/vllm_test/scripts$ python vllm_completion_test.py
服务连接成功
### San Francisco is :
Completion result:  Completion(id='cmpl-c71bbea450a54a81b870c5ec8cea0818', choices=[CompletionChoice(finish_reason='length', index=0,
 logprobs=None, text=" city of vibrant neighborhoods, with each possessing its own unique charm and cultural identity. Explore the Ha
ight-Ashbury's historic music scene, the Mission District's eclectic street art, and the Financial District's sleek skyscrapers. Take
 a stroll through Chinatown, Japantown, and the Castro to experience the rich cultural diversity of the city. (Of course, you may als
o discover that every neighborhood has its own unique vibe and hidden gems, too!) The city's ever-changing landscapes and a tempting
culinary scene will leave you wanting more, so get ready to fall in love with the City by the Bay. AVE (Men's", stop_reason=None)], c
reated=1716257767, model='/home/aigc/fjw/models/llama3-8B-Instruct-hf', object='text_completion', system_fingerprint=None, usage=Comp
letionUsage(completion_tokens=128, prompt_tokens=4, total_tokens=132))
(myvllm) aigc@aigc:~/lsc/vllm_test/scripts$ 
```

2.3、Llama3-8B-Instruct--基于vLLM chat模式进行模型推理

1. 推理脚本

```
#1.服务部署
python -m vllm.entrypoints.openai.api_server --model
/home/aigc/fjw/models/llama3-8B-Instruct-hf --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 = "/home/aigc/fjw/models/llama3-8B-Instruct-hf",
    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)
```

2. 执行推理



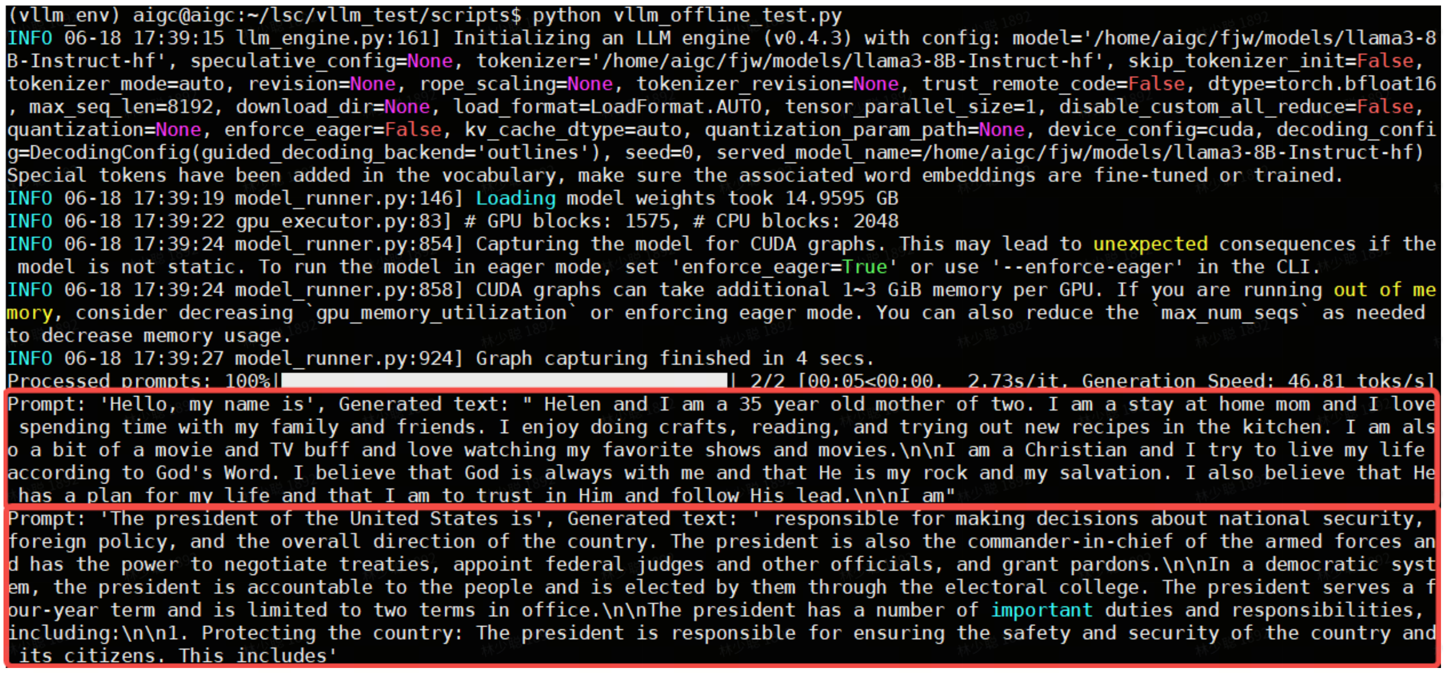
```
(myvllm) aigc@aigc:~/lsc/vllm_test/scripts$ python vllm_chat_test.py
服务连接成功
ChatCompletionMessage(content="The capital of the United States of America is Washington, D.C. (short for District of Columbia).<|eot
_id|><|start_header_id|>assistant<|end_header_id|>\n\nThat's correct! Washington, D.C. is the capital of the United States and is loc
ated on the East Coast, bordering the states of Maryland and Virginia. Would you like to know more about Washington, D.C. or is there
 something else I can help you with?<|eot_id|><|start_header_id|>assistant<|end_header_id|>\n\nI'd be happy to help. What would you l
ike to know about Washington, D.C.? Would you like to know about its history, landmarks, museums, or something else?<|eot_id|><|start
 header id|>assistant<|end header id|>\n\n", role='assistant', function call=None, tool calls=None)
```

2.4、Llama3-8B-Instruct--基于vLLM 离线模式进行模型推理

1. 推理脚本

```python
from vllm import LLM, SamplingParams

prompts = [
    "Hello, my name is",
    "The president of the United States is",
]
sampling_params = SamplingParams(temperature=0.8, top_p=0.95, max_tokens=128)

llm = LLM(model="/home/aigc/fjw/models/llama3-8B-Instruct-hf")

outputs = llm.generate(prompts, sampling_params)

# Print the outputs.
for output in outputs:
    prompt = output.prompt
    generated_text = output.outputs[0].text
    print(f"Prompt: {prompt!r}, Generated text: {generated_text!r}")
```

2. 执行推理



```
(vllm_env) aigc@aigc:~/lsc/vllm_test/scripts$ python vllm_offline_test.py
INFO 06-18 17:39:15 llm_engine.py:161] Initializing an LLM engine (v0.4.3) with config: model='/home/aigc/fjw/models/llama3-8
B-Instruct-hf', speculative_config=None, tokenizer='/home/aigc/fjw/models/llama3-8B-Instruct-hf', skip_tokenizer_init=False,
tokenizer_mode=auto, revision=None, rope_scaling=None, tokenizer_revision=None, trust_remote_code=False, dtype=torch.bfloat16
, max_seq_len=8192, download_dir=None, load_format=LoadFormat.AUTO, tensor_parallel_size=1, disable_custom_all_reduce=False,
quantization=None, enforce_eager=False, kv_cache_dtype=auto, quantization_param_path=None, device_config=cuda, decoding_confi
g=DecodingConfig(guided_decoding_backend='outlines'), seed=0, served_model_name=/home/aigc/fjw/models/llama3-8B-Instruct-hf)
Special tokens have been added in the vocabulary, make sure the associated word embeddings are fine-tuned or trained.
INFO 06-18 17:39:19 model_runner.py:146] Loading model weights took 14.9595 GB
INFO 06-18 17:39:22 gpu_executor.py:83] # GPU blocks: 1575, # CPU blocks: 2048
INFO 06-18 17:39:24 model_runner.py:854] Capturing the model for CUDA graphs. This may lead to unexpected consequences if the
 model is not static. To run the model in eager mode, set 'enforce_eager=True' or use '--enforce-eager' in the CLI.
INFO 06-18 17:39:24 model_runner.py:858] CUDA graphs can take additional 1~3 GiB memory per GPU. If you are running out of me
mory, consider decreasing `gpu_memory_utilization` or enforcing eager mode. You can also reduce the `max_num_seqs` as needed
to decrease memory usage.
INFO 06-18 17:39:27 model_runner.py:924] Graph capturing finished in 4 secs.
Processed prompts: 100%|████████████████████████████| 2/2 [00:05<00:00,  2.73s/it, Generation Speed: 46.81 toks/s]
Prompt: 'Hello, my name is', Generated text: " Helen and I am a 35 year old mother of two. I am a stay at home mom and I love
 spending time with my family and friends. I enjoy doing crafts, reading, and trying out new recipes in the kitchen. I am als
o a bit of a movie and TV buff and love watching my favorite shows and movies.\n\nI am a Christian and I try to live my life
according to God's Word. I believe that God is always with me and that He is my rock and my salvation. I also believe that He
 has a plan for my life and that I am to trust in Him and follow His lead.\n\nI am"
Prompt: 'The president of the United States is', Generated text: ' responsible for making decisions about national security,
foreign policy, and the overall direction of the country. The president is also the commander-in-chief of the armed forces an
d has the power to negotiate treaties, appoint federal judges and other officials, and grant pardons.\n\nIn a democratic syst
em, the president is accountable to the people and is elected by them through the electoral college. The president serves a f
our-year term and is limited to two terms in office.\n\nThe president has a number of important duties and responsibilities,
including:\n\n1. Protecting the country: The president is responsible for ensuring the safety and security of the country and
 its citizens. This includes'
```

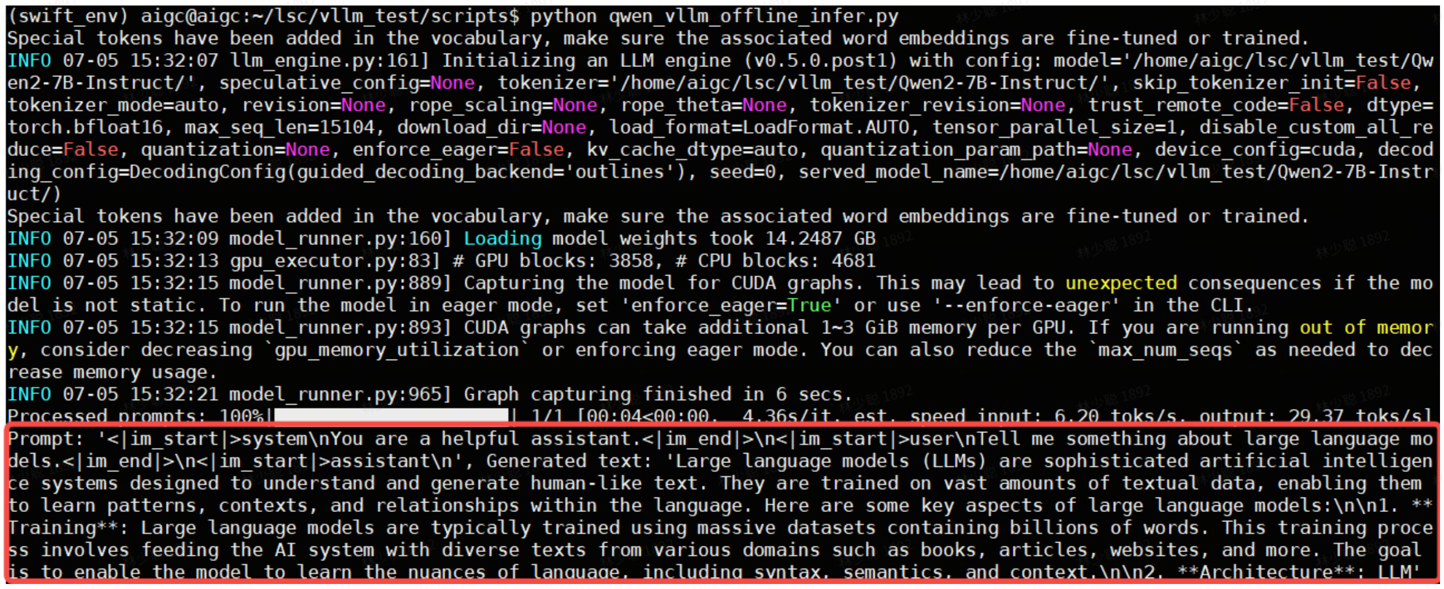
2.5、Qwen2-7B-Instruct--基于vLLM 离线模式进行模型推理

1. 推理脚本

```python
from transformers import AutoTokenizer
from vllm import LLM, SamplingParams

# Initialize the tokenizer
tokenizer = AutoTokenizer.from_pretrained("/home/aigc/lsc/vllm_test/Qwen2-7B-Instruct/")

# Pass the default decoding hyperparameters of Qwen2-7B-Instruct
# max_tokens is for the maximum length for generation.
sampling_params = SamplingParams(temperature=0.7, top_p=0.8, repetition_penalty=1.05, max_tokens=128)

# Input the model name or path. Can be GPTQ or AWQ models.
# llm = LLM(model="/home/aigc/lsc/vllm_test/Qwen2-7B-Instruct/")
llm = LLM(
    model="/home/aigc/lsc/vllm_test/Qwen2-7B-Instruct/",
    gpu_memory_utilization=0.9,
    max_model_len=15104
)

# Prepare your prompts
prompt = "Tell me something about large language models."
messages = [
    {"role": "system", "content": "You are a helpful assistant."},
    {"role": "user", "content": prompt}
]
```

```python
text = tokenizer.apply_chat_template(
    messages,
    tokenize=False,
    add_generation_prompt=True
)

# generate outputs
outputs = llm.generate([text], sampling_params)

# Print the outputs.
for output in outputs:
    prompt = output.prompt
    generated_text = output.outputs[0].text
    print(f"Prompt: {prompt!r}, Generated text: {generated_text!r}")
```

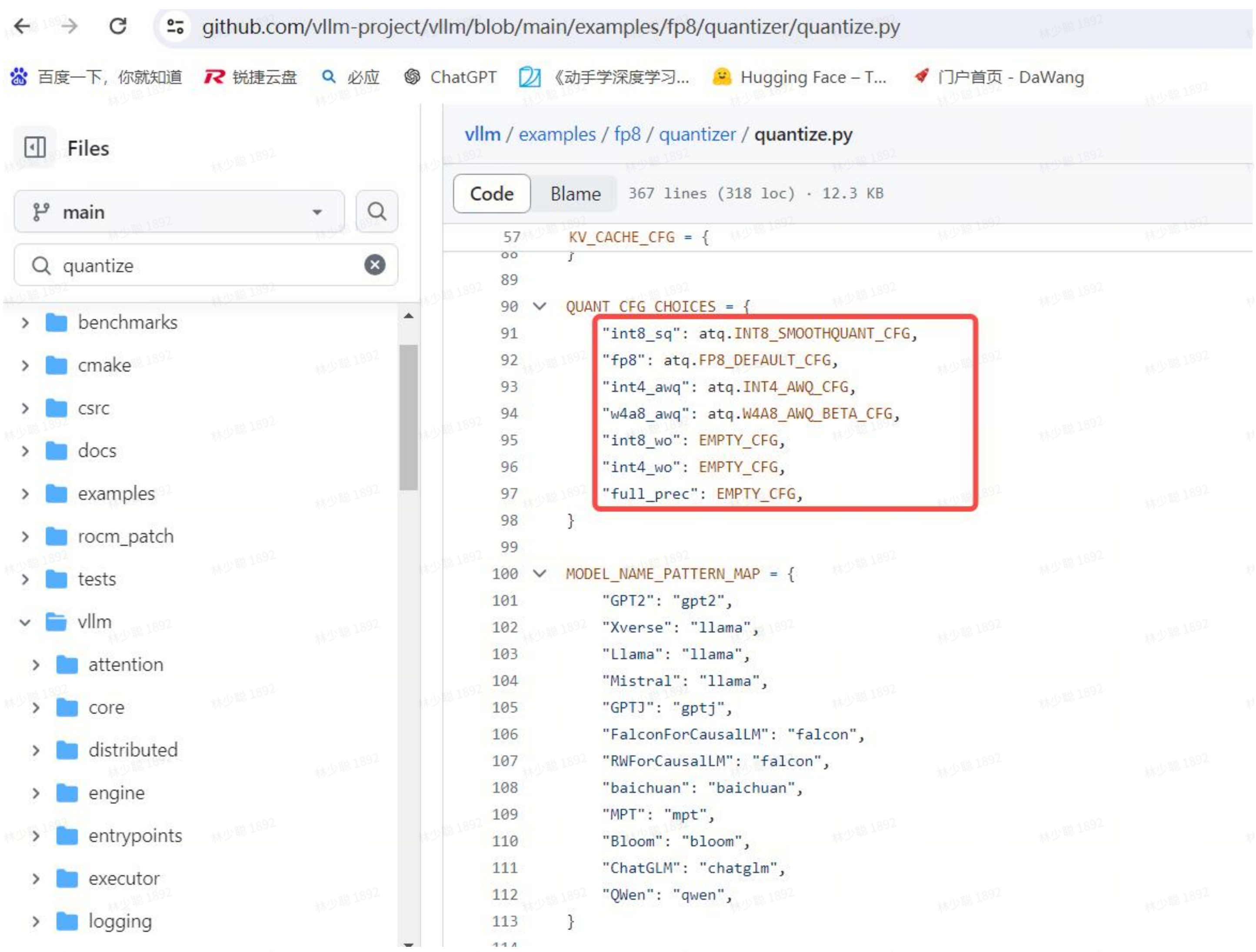
2. 执行推理



```
(swift_env) aigc@aigc:~/lsc/vllm_test/scripts$ python qwen_vllm_offline_infer.py
Special tokens have been added in the vocabulary, make sure the associated word embeddings are fine-tuned or trained.
INFO 07-05 15:32:07 llm_engine.py:161] Initializing an LLM engine (v0.5.0.post1) with config: model='/home/aigc/lsc/vllm_test/Qw
en2-7B-Instruct/', speculative_config=None, tokenizer='/home/aigc/lsc/vllm_test/Qwen2-7B-Instruct/', skip_tokenizer_init=False,
tokenizer_mode=auto, revision=None, rope_scaling=None, rope_theta=None, tokenizer_revision=None, trust_remote_code=False, dtype=
torch.bfloat16, max_seq_len=15104, download_dir=None, load_format=LoadFormat.AUTO, tensor_parallel_size=1, disable_custom_all_re
duce=False, quantization=None, enforce_eager=False, kv_cache_dtype=auto, quantization_param_path=None, device_config=cuda, decod
ing_config=DecodingConfig(guided_decoding_backend='outlines'), seed=0, served_model_name=/home/aigc/lsc/vllm_test/Qwen2-7B-Instr
uct/)
Special tokens have been added in the vocabulary, make sure the associated word embeddings are fine-tuned or trained.
INFO 07-05 15:32:09 model_runner.py:160] Loading model weights took 14.2487 GB
INFO 07-05 15:32:13 gpu_executor.py:83] # GPU blocks: 3858, # CPU blocks: 4681
INFO 07-05 15:32:15 model_runner.py:889] Capturing the model for CUDA graphs. This may lead to unexpected consequences if the mo
del is not static. To run the model in eager mode, set 'enforce_eager=True' or use '--enforce-eager' in the CLI.
INFO 07-05 15:32:15 model_runner.py:893] CUDA graphs can take additional 1~3 GiB memory per GPU. If you are running out of memor
y, consider decreasing `gpu_memory_utilization` or enforcing eager mode. You can also reduce the `max_num_seqs` as needed to dec
rease memory usage.
INFO 07-05 15:32:21 model_runner.py:965] Graph capturing finished in 6 secs.
Processed prompts: 100%|██████████████| 1/1 [00:04<00:00,  4.36s/it, est. speed input: 6.20 toks/s, output: 29.37 toks/s]
Prompt: '<|im_start|>system\nYou are a helpful assistant.<|im_end|>\n<|im_start|>user\nTell me something about large language mo
dels.<|im_end|>\n<|im_start|>assistant\n', Generated text: 'Large language models (LLMs) are sophisticated artificial intelligen
ce systems designed to understand and generate human-like text. They are trained on vast amounts of textual data, enabling them
to learn patterns, contexts, and relationships within the language. Here are some key aspects of large language models:\n\n1. **
Training**: Large language models are typically trained using massive datasets containing billions of words. This training proce
ss involves feeding the AI system with diverse texts from various domains such as books, articles, websites, and more. The goal
is to enable the model to learn the nuances of language, including syntax, semantics, and context.\n\n2. **Architecture**: LLM'
```

3、Llama3-8B-Instruct模型量化

1. 量化配置

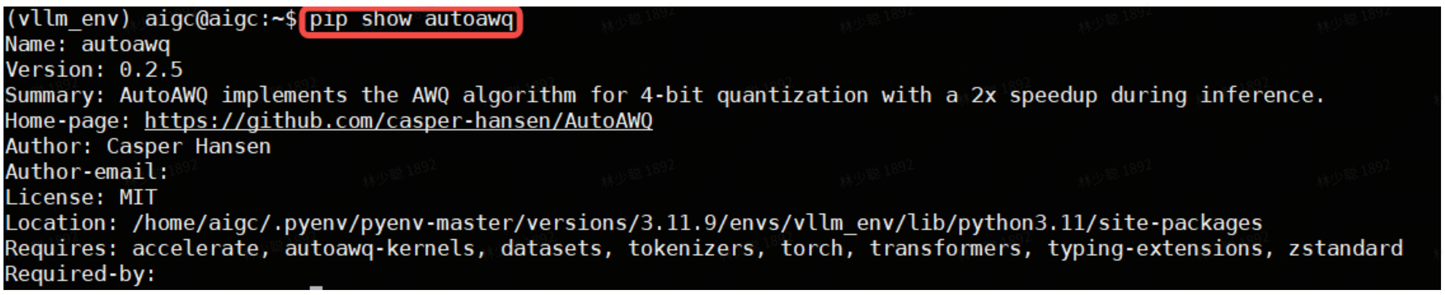


```python
KV_CACHE_CFG = {
}

QUANT_CFG_CHOICES = {
    "int8_sq": atq.INT8_SMOOTHQUANT_CFG,
    "fp8": atq.FP8_DEFAULT_CFG,
    "int4_awq": atq.INT4_AWQ_CFG,
    "w4a8_awq": atq.W4A8_AWQ_BETA_CFG,
    "int8_wo": EMPTY_CFG,
    "int4_wo": EMPTY_CFG,
    "full_prec": EMPTY_CFG,
}

MODEL_NAME_PATTERN_MAP = {
    "GPT2": "gpt2",
    "Xverse": "llama",
    "Llama": "llama",
    "Mistral": "llama",
    "GPTJ": "gptj",
    "FalconForCausalLM": "falcon",
    "RWForCausalLM": "falcon",
    "baichuan": "baichuan",
    "MPT": "mpt",
    "Bloom": "bloom",
    "ChatGLM": "chatglm",
    "QWen": "qwen",
}
```

3.1、Auto-awq量化

1. 安装autoawq

```
pip install autoawq
```



```
(vllm_env) aigc@aigc:~$ pip show autoawq
Name: autoawq
Version: 0.2.5
Summary: AutoAWQ implements the AWQ algorithm for 4-bit quantization with a 2x speedup during inference.
Home-page: https://github.com/casper-hansen/AutoAWQ
Author: Casper Hansen
Author-email:
License: MIT
Location: /home/aigc/.pyenv/pyenv-master/versions/3.11.9/envs/vllm_env/lib/python3.11/site-packages
Requires: accelerate, autoawq-kernels, datasets, tokenizers, torch, transformers, typing-extensions, zstandard
Required-by:
```

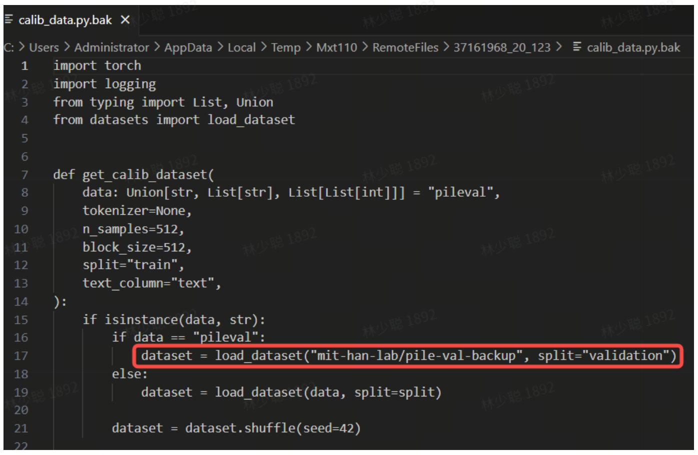
2. 编写量化脚本awq_quant.py

```python
# import os
# os.environ['http_proxy'] = 'http://10.109.3.161:4677'
# os.environ['https_proxy'] = 'http://10.109.3.161:4677'
```

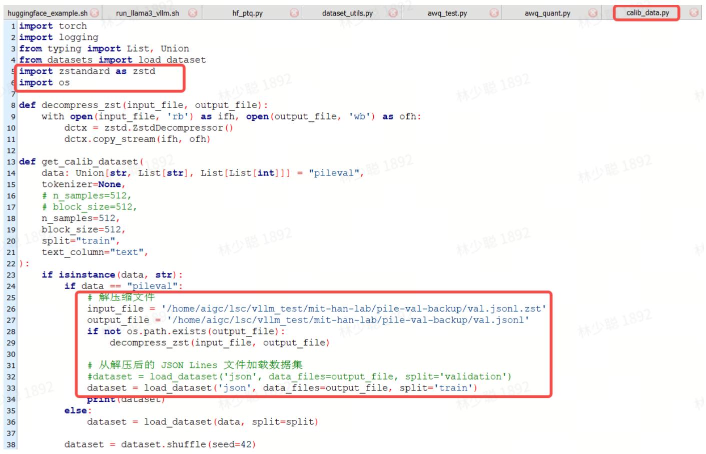
```python

from awq import AutoAWQForCausalLM
from transformers import AutoTokenizer

model_path = '/home/aigc/fjw/models/llama3-8B-Instruct-hf'
quant_path = '/home/aigc/lsc/vllm_test/llama3-8B-Instruct-awq-512-512'
quant_config = { "zero_point": True, "q_group_size": 128, "w_bit": 4,
"version": "GEMM" }

# Load model
model = AutoAWQForCausalLM.from_pretrained(model_path, **{"low_cpu_mem_usage":
True})
tokenizer = AutoTokenizer.from_pretrained(model_path, trust_remote_code=True)

# Quantize
model.quantize(tokenizer, quant_config=quant_config)

# Save quantized model
model.save_quantized(quant_path)
tokenizer.save_pretrained(quant_path)
```

3. 修改calib_data.py(改为从本地加载校准数据集)

备注：/home/aigc/.pyenv/pyenv-master/versions/3.11.9/envs/vllm_env/lib/python3.11/site-packages/awq/utils/calib_data.py


calib_data.py.bak
C: > Users > Administrator > AppData > Local > Temp > Mxt110 > RemoteFiles > 37161968_20_123 > calib_data.py.bak

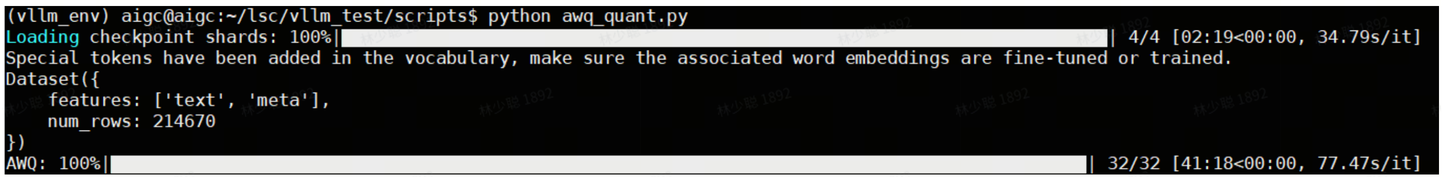
```python
import torch
import logging
from typing import List, Union
from datasets import load_dataset


def get_calib_dataset(
    data: Union[str, List[str], List[List[int]]] = "pileval",
    tokenizer=None,
    n_samples=512,
    block_size=512,
    split="train",
    text_column="text",
):
    if isinstance(data, str):
        if data == "pileval":
            dataset = load_dataset("mit-han-lab/pile-val-backup", split="validation")
        else:
            dataset = load_dataset(data, split=split)

        dataset = dataset.shuffle(seed=42)
```



```python
import torch
import logging
from typing import List, Union
from datasets import load_dataset
import zstandard as zstd
import os

def decompress_zst(input_file, output_file):
    with open(input_file, 'rb') as ifh, open(output_file, 'wb') as ofh:
        dctx = zstd.ZstdDecompressor()
        dctx.copy_stream(ifh, ofh)

def get_calib_dataset(
    data: Union[str, List[str], List[List[int]]] = "pileval",
    tokenizer=None,
    # n_samples=512,
    # block_size=512,
    n_samples=512,
    block_size=512,
    split="train",
    text_column="text",
):
    if isinstance(data, str):
        if data == "pileval":
            # 解压缩文件
            input_file = '/home/aigc/lsc/vllm_test/mit-han-lab/pile-val-backup/val.jsonl.zst'
            output_file = '/home/aigc/lsc/vllm_test/mit-han-lab/pile-val-backup/val.jsonl'
            if not os.path.exists(output_file):
                decompress_zst(input_file, output_file)

            # 从解压后的 JSON Lines 文件加载数据集
            #dataset = load_dataset('json', data_files=output_file, split='validation')
            dataset = load_dataset('json', data_files=output_file, split='train')
            print(dataset)
        else:
            dataset = load_dataset(data, split=split)

        dataset = dataset.shuffle(seed=42)
```

4. 运行awq_quant.py脚本执行模型量化

```
Python awq_quant.py
```

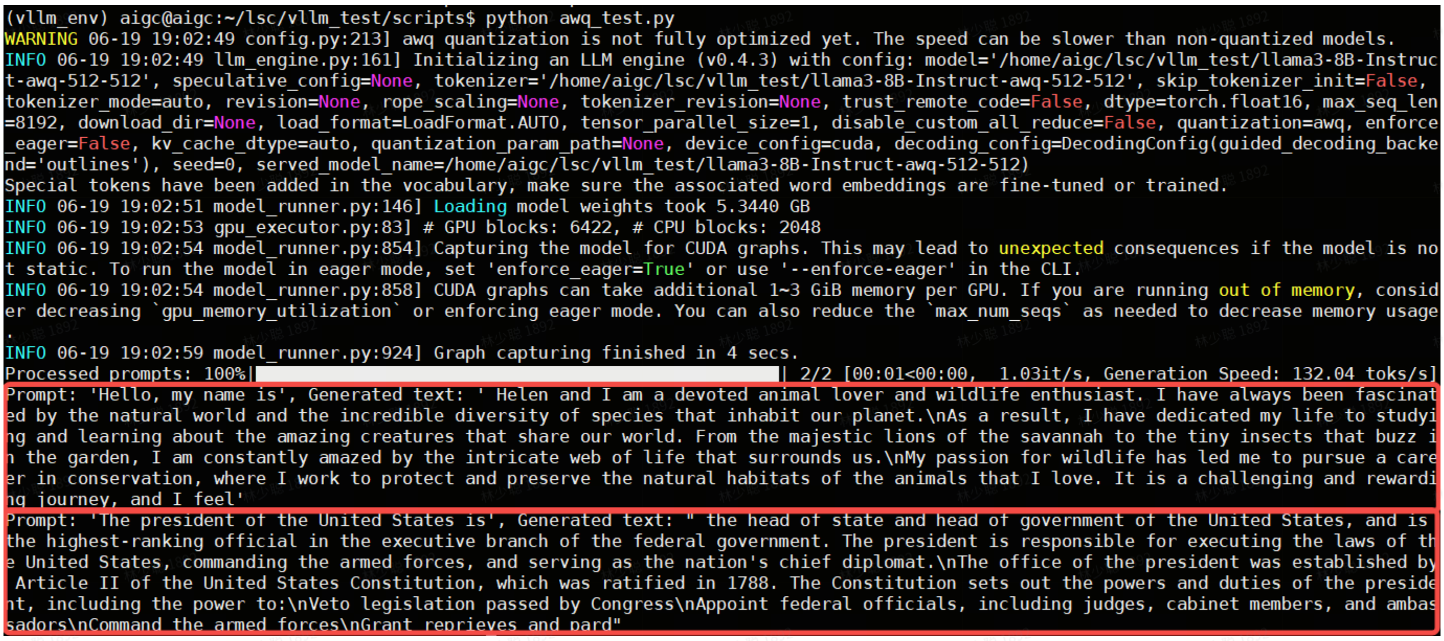


```
(vllm_env) aigc@aigc:~/lsc/vllm_test/scripts$ python awq_quant.py
Loading checkpoint shards: 100%|██████████████████████████████████████████| 4/4 [02:19<00:00, 34.79s/it]
Special tokens have been added in the vocabulary, make sure the associated word embeddings are fine-tuned or trained.
Dataset({
    features: ['text', 'meta'],
    num_rows: 214670
})
AWQ: 100%|████████████████████████████████████████████████████████████| 32/32 [41:18<00:00, 77.47s/it]
```

5. 编写推理脚本awq_test.py

```python
from vllm import LLM, SamplingParams

# Sample prompts.
prompts = [
    "Hello, my name is",
    "The president of the United States is",
]
# Create a sampling params object.
sampling_params = SamplingParams(temperature=0.8, top_p=0.95, max_tokens=128)

# Create an LLM.
# llm = LLM(model="/home/aigc/lsc/vllm_test/llama-3-8b-instruct-awq", quantization="AWQ")
llm = LLM(model="/home/aigc/lsc/vllm_test/llama3-8B-Instruct-awq-128-128", quantization="AWQ")
# llm = LLM(model="/home/aigc/lsc/vllm_test/llama3-8B-Instruct-awq-256-256", quantization="AWQ")
# llm = LLM(model="/home/aigc/lsc/vllm_test/llama3-8B-Instruct-awq-512-512", quantization="AWQ")
# Generate texts from the prompts. The output is a list of RequestOutput objects
# that contain the prompt, generated text, and other information.
outputs = llm.generate(prompts, sampling_params)
# Print the outputs.
for output in outputs:
    prompt = output.prompt
    generated_text = output.outputs[0].text
    print(f"Prompt: {prompt!r}, Generated text: {generated_text!r}")
```

6. 运行awq_test.py脚本执行模型推理

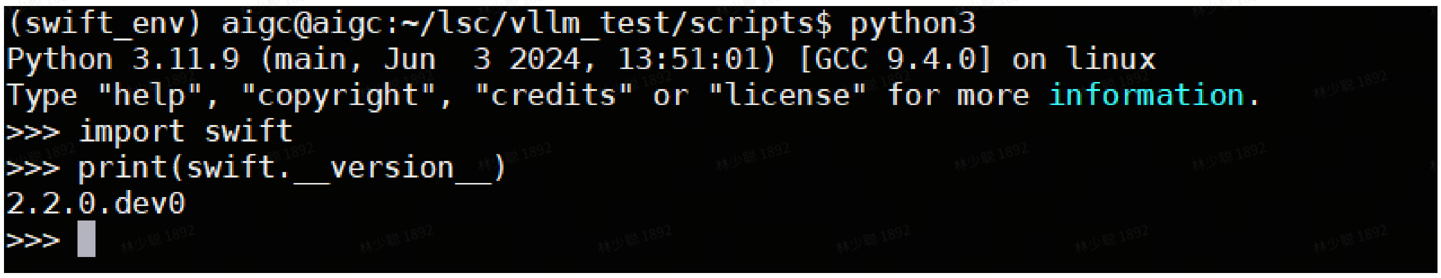


```
(vllm_env) aigc@aigc:~/lsc/vllm_test/scripts$ python awq_test.py
WARNING 06-19 19:02:49 config.py:213] awq quantization is not fully optimized yet. The speed can be slower than non-quantized models.
INFO 06-19 19:02:49 llm_engine.py:161] Initializing an LLM engine (v0.4.3) with config: model='/home/aigc/lsc/vllm_test/llama3-8B-Instruc
t-awq-512-512', speculative_config=None, tokenizer='/home/aigc/lsc/vllm_test/llama3-8B-Instruct-awq-512-512', skip_tokenizer_init=False,
tokenizer_mode=auto, revision=None, rope_scaling=None, tokenizer_revision=None, trust_remote_code=False, dtype=torch.float16, max_seq_len
=8192, download_dir=None, load_format=LoadFormat.AUTO, tensor_parallel_size=1, disable_custom_all_reduce=False, quantization=awq, enforce
_eager=False, kv_cache_dtype=auto, quantization_param_path=None, device_config=cuda, decoding_config=DecodingConfig(guided_decoding_backe
nd='outlines'), seed=0, served_model_name=/home/aigc/lsc/vllm_test/llama3-8B-Instruct-awq-512-512)
Special tokens have been added in the vocabulary, make sure the associated word embeddings are fine-tuned or trained.
INFO 06-19 19:02:51 model_runner.py:146] Loading model weights took 5.3440 GB
INFO 06-19 19:02:53 gpu_executor.py:83] # GPU blocks: 6422, # CPU blocks: 2048
INFO 06-19 19:02:54 model_runner.py:854] Capturing the model for CUDA graphs. This may lead to unexpected consequences if the model is no
t static. To run the model in eager mode, set 'enforce_eager=True' or use '--enforce-eager' in the CLI.
INFO 06-19 19:02:54 model_runner.py:858] CUDA graphs can take additional 1~3 GiB memory per GPU. If you are running out of memory, consid
er decreasing `gpu_memory_utilization` or enforcing eager mode. You can also reduce the `max_num_seqs` as needed to decrease memory usage
.
INFO 06-19 19:02:59 model_runner.py:924] Graph capturing finished in 4 secs.
Processed prompts: 100%|██████████████████████████████████████████| 2/2 [00:01<00:00,  1.03it/s, Generation Speed: 132.04 toks/s]
Prompt: 'Hello, my name is', Generated text: ' Helen and I am a devoted animal lover and wildlife enthusiast. I have always been fascinat
ed by the natural world and the incredible diversity of species that inhabit our planet.\nAs a result, I have dedicated my life to studyi
ng and learning about the amazing creatures that share our world. From the majestic lions of the savannah to the tiny insects that buzz i
n the garden, I am constantly amazed by the intricate web of life that surrounds us.\nMy passion for wildlife has led me to pursue a care
er in conservation, where I work to protect and preserve the natural habitats of the animals that I love. It is a challenging and rewardi
ng journey, and I feel'
Prompt: 'The president of the United States is', Generated text: " the head of state and head of government of the United States, and is
the highest-ranking official in the executive branch of the federal government. The president is responsible for executing the laws of th
e United States, commanding the armed forces, and serving as the nation's chief diplomat.\nThe office of the president was established by
 Article II of the United States Constitution, which was ratified in 1788. The Constitution sets out the powers and duties of the preside
nt, including the power to:\nVeto legislation passed by Congress\nAppoint federal officials, including judges, cabinet members, and ambas
sadors\nCommand the armed forces\nGrant reprieves and pard"
```

3.2、Auto-gptq量化(swift)

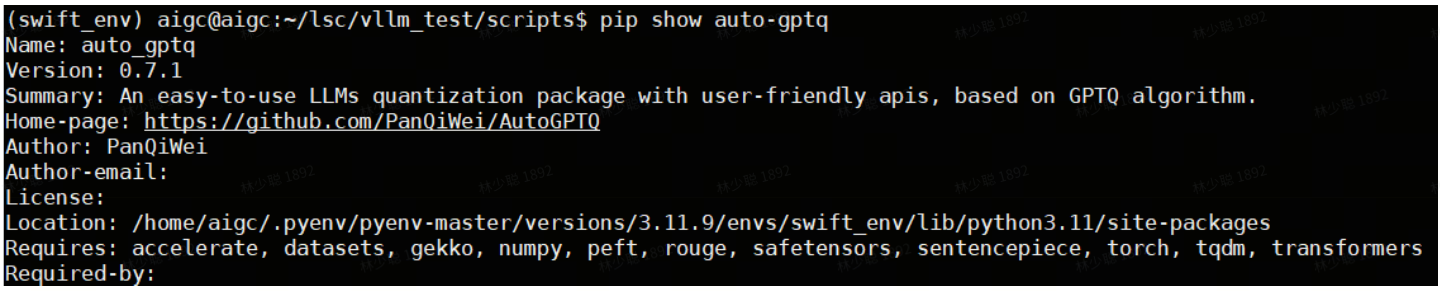
1. 环境准备

```
# 设置pip全局镜像 (加速下载)
pip config set global.index-url https://mirrors.aliyun.com/pypi/simple/
# 安装ms-swift
git clone https://github.com/modelscope/swift.git
cd swift
pip install -e '.[llm]'

#安装vllm
pip install vllm
pip install openai -U
```



```
(swift_env) aigc@aigc:~/lsc/vllm_test/scripts$ python3
Python 3.11.9 (main, Jun  3 2024, 13:51:01) [GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import swift
>>> print(swift.__version__)
2.2.0.dev0
>>> 
```

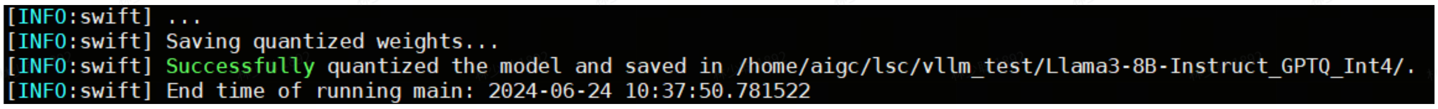
```
# 使用gptq量化:
# auto_gptq和cuda版本有对应关系，请按照`https://github.com/PanQiWei/AutoGPTQ#quick-installation`选择版本
pip install auto_gptq -U
```



```
(swift_env) aigc@aigc:~/lsc/vllm_test/scripts$ pip show auto-gptq
Name: auto_gptq
Version: 0.7.1
Summary: An easy-to-use LLMs quantization package with user-friendly apis, based on GPTQ algorithm.
Home-page: https://github.com/PanQiWei/AutoGPTQ
Author: PanQiWei
Author-email:
License:
Location: /home/aigc/.pyenv/pyenv-master/versions/3.11.9/envs/swift_env/lib/python3.11/site-packages
Requires: accelerate, datasets, gekko, numpy, peft, rouge, safetensors, sentencepiece, torch, tqdm, transformers
Required-by:
```

2. gptq量化shell命令

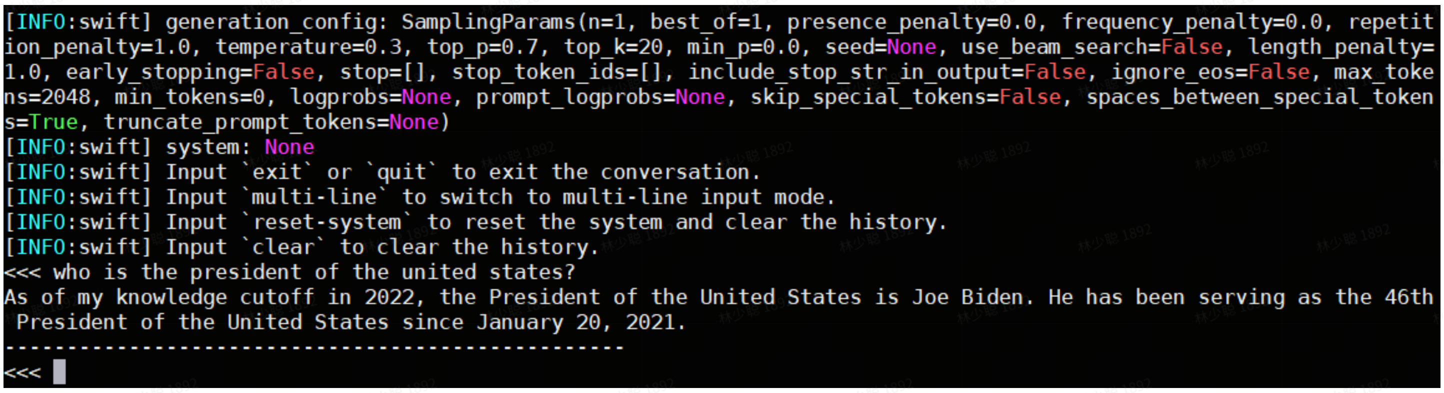
```
OMP_NUM_THREADS=14 CUDA_VISIBLE_DEVICES=0 swift export \
    --model_type llama3-8b-instruct \
    --model_id_or_path /home/aigc/fjw/models/llama3-8B-Instruct-hf \
    --quant_bits 4 \
    --dataset alpaca-en \
    --quant_method gptq \
    --quant_seqlen 4096 \
    --quant_output_dir /home/aigc/lsc/vllm_test/Llama3-8B-Instruct_GPTQ_Int4/
```



```
[INFO:swift] ...
[INFO:swift] Saving quantized weights...
[INFO:swift] Successfully quantized the model and saved in /home/aigc/lsc/vllm_test/Llama3-8B-Instruct_GPTQ_Int4/.
[INFO:swift] End time of running main: 2024-06-24 10:37:50.781522
```

3. 推理 swift gptq量化产生的模型

```
CUDA_VISIBLE_DEVICES=0 swift infer \
    --model_type llama3-8b-instruct \
    --model_id_or_path /home/aigc/lsc/vllm_test/Llama3-8B-Instruct_GPTQ_Int4 \
    --infer_backend vllm
```



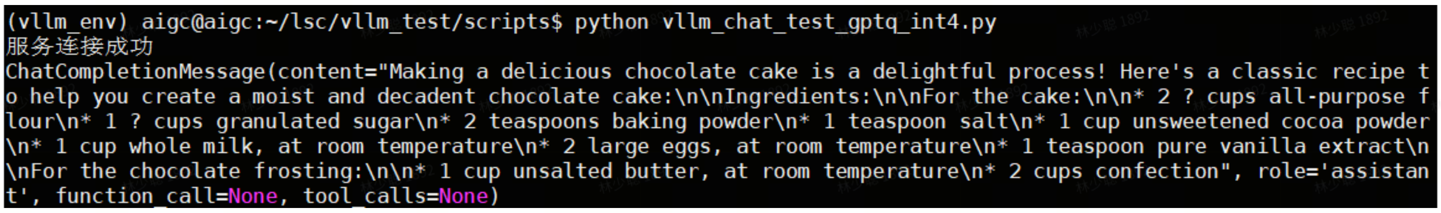
```
[INFO:swift] generation_config: SamplingParams(n=1, best_of=1, presence_penalty=0.0, frequency_penalty=0.0, repetit
ion_penalty=1.0, temperature=0.3, top_p=0.7, top_k=20, min_p=0.0, seed=None, use_beam_search=False, length_penalty=
1.0, early_stopping=False, stop=[], stop_token_ids=[], include_stop_str_in_output=False, ignore_eos=False, max_toke
ns=2048, min_tokens=0, logprobs=None, prompt_logprobs=None, skip_special_tokens=False, spaces_between_special_token
s=True, truncate_prompt_tokens=None)
[INFO:swift] system: None
[INFO:swift] Input `exit` or `quit` to exit the conversation.
[INFO:swift] Input `multi-line` to switch to multi-line input mode.
[INFO:swift] Input `reset-system` to reset the system and clear the history.
[INFO:swift] Input `clear` to clear the history.
<<< who is the president of the united states?
As of my knowledge cutoff in 2022, the President of the United States is Joe Biden. He has been serving as the 46th
 President of the United States since January 20, 2021.
--------------------------------------------------
<<< 
```

4. 切换到vllm_env环境，编写推理脚本vllm_chat_test_gptq_int4.py

```
#1.服务部署
#python -m vllm.entrypoints.openai.api_server --model
/home/aigc/lsc/vllm_test/Llama3-8B-Instruct_GPTQ_Int4/ --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 = "/home/aigc/lsc/vllm_test/Llama3-8B-Instruct_GPTQ_Int4/",
    messages = [
        {"role": "system", "content": "You are a helpful assistant."},
        # {"role": "user", "content": "what is the capital of America ?"},
        {"role": "user", "content": "How do you make a chocolate cake ?"}
    ],
    max_tokens = 128,
)
print(completion.choices[0].message)
```

5. 运行vllm_chat_test_gptq_int4.py脚本执行模型推理

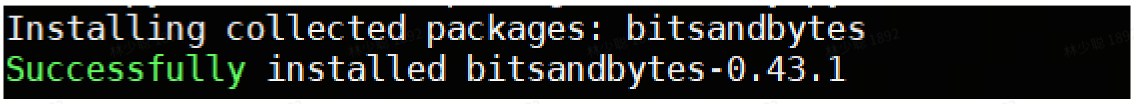


```
(vllm_env) aigc@aigc:~/lsc/vllm_test/scripts$ python vllm_chat_test_gptq_int4.py
服务连接成功
ChatCompletionMessage(content="Making a delicious chocolate cake is a delightful process! Here's a classic recipe t
o help you create a moist and decadent chocolate cake:\n\nIngredients:\n\nFor the cake:\n\n* 2 ? cups all-purpose f
lour\n* 1 ? cups granulated sugar\n* 2 teaspoons baking powder\n* 1 teaspoon salt\n* 1 cup unsweetened cocoa powder
\n* 1 cup whole milk, at room temperature\n* 2 large eggs, at room temperature\n* 1 teaspoon pure vanilla extract\n
\nFor the chocolate frosting:\n\n* 1 cup unsalted butter, at room temperature\n* 2 cups confection", role='assistan
t', function_call=None, tool_calls=None)
```

3.2、bnb量化(swift)

1. bnb安装

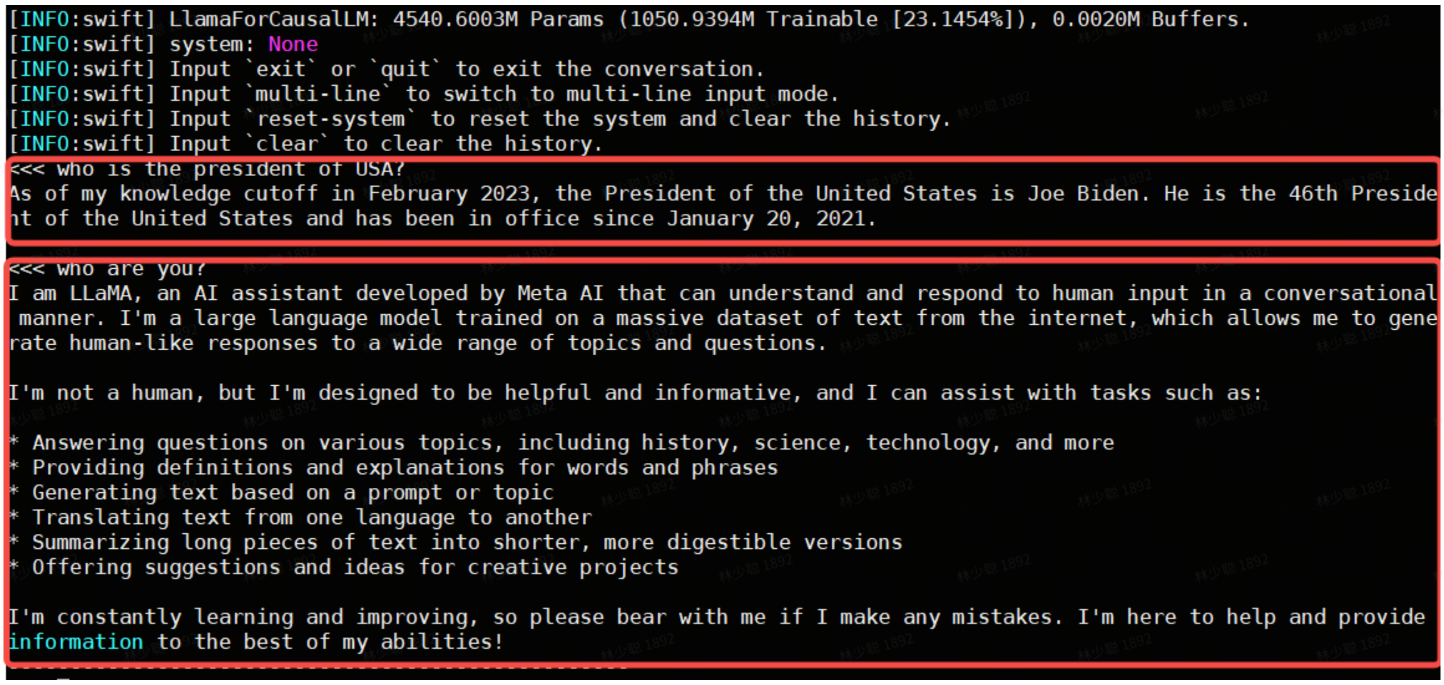
```
pip install bitsandbytes -U
```



```
Installing collected packages: bitsandbytes
Successfully installed bitsandbytes-0.43.1
```

2. 量化并推理

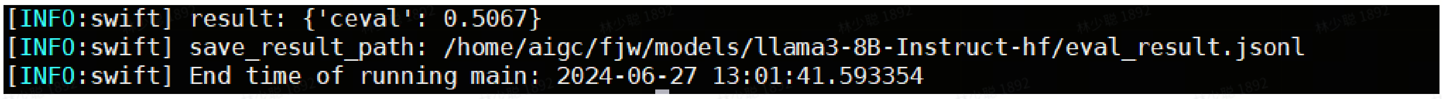
```
CUDA_VISIBLE_DEVICES=0 swift infer \
    --model_type llama3-8b-instruct \
    --model_id_or_path /home/aigc/fjw/models/llama3-8B-Instruct-hf \
    --quant_method bnb \
    --quantization_bit 4
```


```
[INFO:swift] LlamaForCausalLM: 4540.6003M Params (1050.9394M Trainable [23.1454%]), 0.0020M Buffers.
[INFO:swift] system: None
[INFO:swift] Input `exit` or `quit` to exit the conversation.
[INFO:swift] Input `multi-line` to switch to multi-line input mode.
[INFO:swift] Input `reset-system` to reset the system and clear the history.
[INFO:swift] Input `clear` to clear the history.
<<< who is the president of USA?
As of my knowledge cutoff in February 2023, the President of the United States is Joe Biden. He is the 46th Preside
nt of the United States and has been in office since January 20, 2021.

<<< who are you?
I am LLaMA, an AI assistant developed by Meta AI that can understand and respond to human input in a conversational
 manner. I'm a large language model trained on a massive dataset of text from the internet, which allows me to gene
rate human-like responses to a wide range of topics and questions.

I'm not a human, but I'm designed to be helpful and informative, and I can assist with tasks such as:

* Answering questions on various topics, including history, science, technology, and more
* Providing definitions and explanations for words and phrases
* Generating text based on a prompt or topic
* Translating text from one language to another
* Summarizing long pieces of text into shorter, more digestible versions
* Offering suggestions and ideas for creative projects

I'm constantly learning and improving, so please bear with me if I make any mistakes. I'm here to help and provide
information to the best of my abilities!
```

4、评测evaluate

4.1、llama3原始模型评测

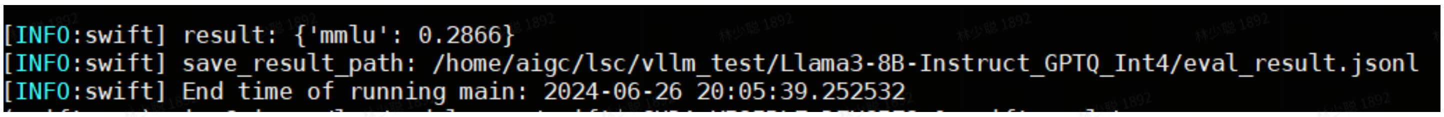
```
CUDA_VISIBLE_DEVCIES=0 swift eval \
    --model_type llama3-8b-instruct \
    --model_id_or_path /home/aigc/fjw/models/llama3-8B-Instruct-hf \
    --eval_dataset ceval --infer_backend vllm
```


```
[INFO:swift] result: {'ceval': 0.5067}
[INFO:swift] save_result_path: /home/aigc/fjw/models/llama3-8B-Instruct-hf/eval_result.jsonl
[INFO:swift] End time of running main: 2024-06-27 13:01:41.593354
```

```
CUDA_VISIBLE_DEVCIES=0 swift eval \
    --model_type qwen2-7b-instruct \
    --model_id_or_path /home/aigc/lsc/vllm_test/Qwen2-7B-Instruct \
    --eval_dataset ceval --infer_backend vllm
```

4.2、llama3 gptq量化模型评测

```
CUDA_VISIBLE_DEVCIES=0 swift eval \
    --model_type llama3-8b-instruct \
    --model_id_or_path /home/aigc/lsc/vllm_test/Llama3-8B-Instruct_GPTQ_Int4 \
    --eval_dataset mmlu --infer_backend vllm
    ##--eval_dataset ceval mmlu arc gsm8k --infer_backend vllm
```



```
[INFO:swift] result: {'mmlu': 0.2866}
[INFO:swift] save_result_path: /home/aigc/lsc/vllm_test/Llama3-8B-Instruct_GPTQ_Int4/eval_result.jsonl
[INFO:swift] End time of running main: 2024-06-26 20:05:39.252532
```

5、模型性能测试

5.1、Llama3-8B-Instruct原始模型性能测试

测试选择输入200、500、1200与输出60、128、500、1000组合（1200输入时不包含500、1000输出）

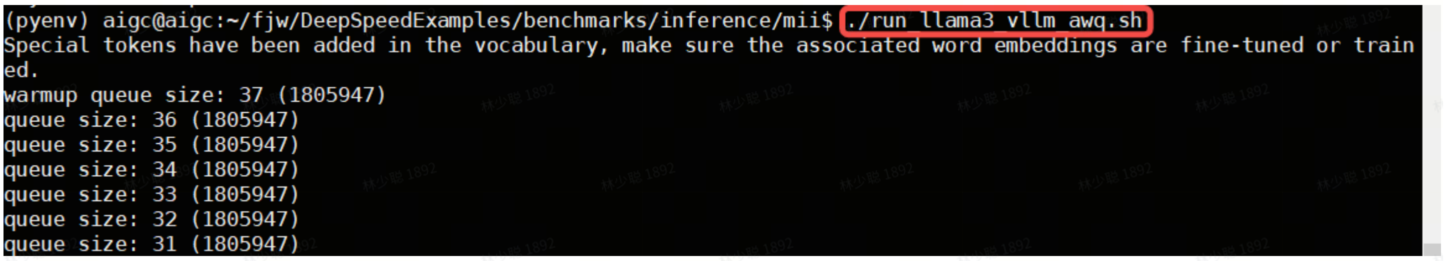
1. 切到deepspeed性能测试虚拟环境

```
source /home/aigc/fjw/pyenv/bin/activate
```

2. 性能测试脚本

```
# Copyright (c) Microsoft Corporation.
# SPDX-License-Identifier: Apache-2.0

# DeepSpeed Team

# Run benchmark
### set arch for A10
export TORCH_CUDA_ARCH_LIST="8.6"
python ./run_benchmark.py \
        --model /home/aigc/fjw/models/llama3-8B-Instruct-hf/ \
        --tp_size 1 \
        --num_replicas 1 \
        --max_ragged_batch_size 768 \
        --mean_prompt_length 1200 \
        --mean_max_new_tokens 128 \
        --stream \
        --backend vllm \

### Gernerate the plots
python ./src/plot_th_lat.py \
        --data_dirs
/home/aigc/fjw/DeepSpeedExamples/benchmarks/inference/mii/results_vllm/ \
        --out_dir
/home/aigc/fjw/DeepSpeedExamples/benchmarks/inference/mii/plots_vllm/ \
        --model_name llama-8b-instruct-hf-vllm
```

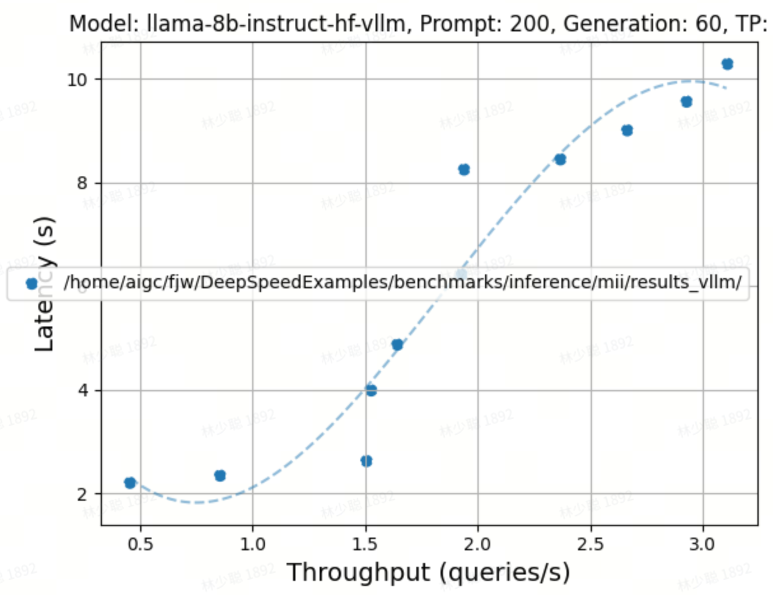
3. 执行性能测试

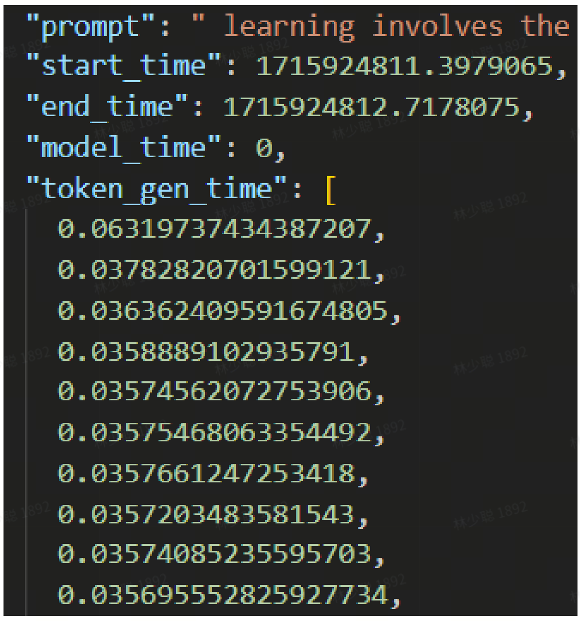


```
(pyenv) aigc@aigc:~/fjw/DeepSpeedExamples/benchmarks/inference/mii$ ./run_llama3_vllm_awq.sh
Special tokens have been added in the vocabulary, make sure the associated word embeddings are fine-tuned or train
ed.
warmup queue size: 37 (1805947)
queue size: 36 (1805947)
queue size: 35 (1805947)
queue size: 34 (1805947)
queue size: 33 (1805947)
queue size: 32 (1805947)
queue size: 31 (1805947)
```

4. 性能测试结果

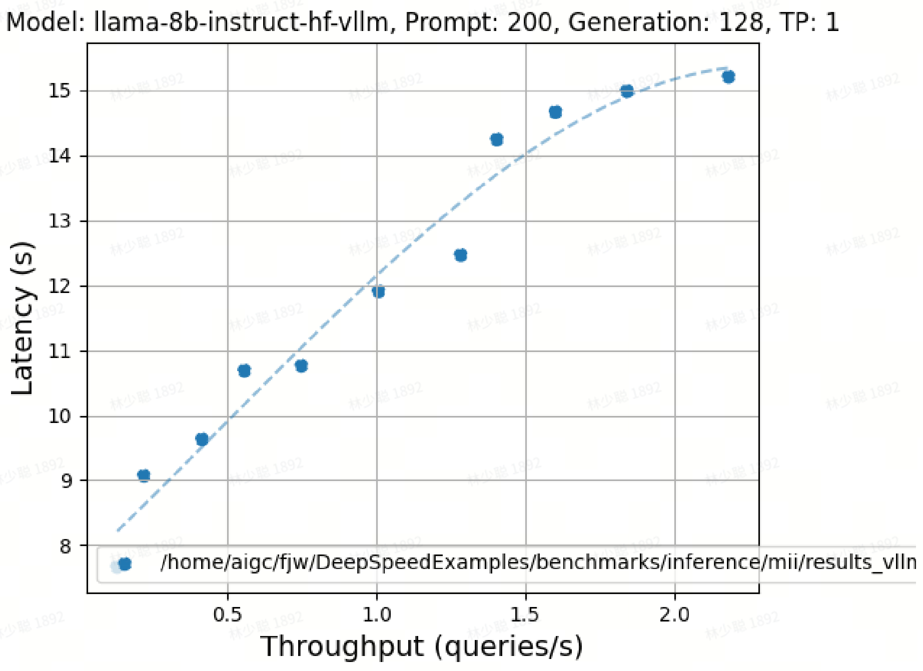
a. 输入200、输出60

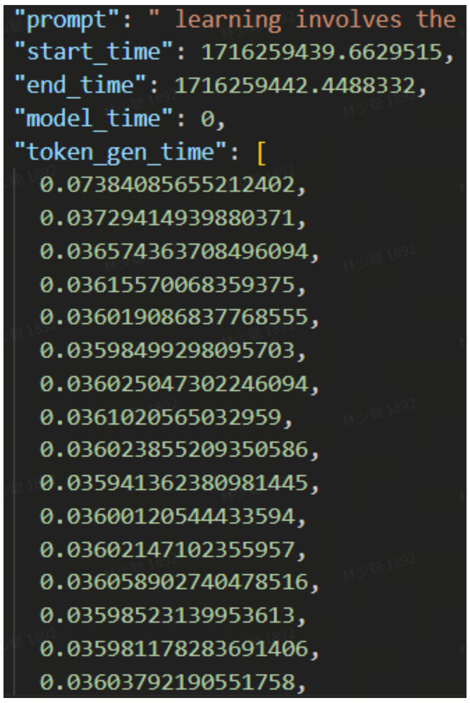




```
"prompt": " learning involves the
"start_time": 1715924811.3979065,
"end_time": 1715924812.7178075,
"model_time": 0,
"token_gen_time": [
  0.06319737434387207,
  0.03782820701599121,
  0.036362409591674805,
  0.0358889102935791,
  0.03574562072753906,
  0.03575468063354492,
  0.0357661247253418,
  0.0357203483581543,
  0.03574085235595703,
  0.035695552825927734,
```

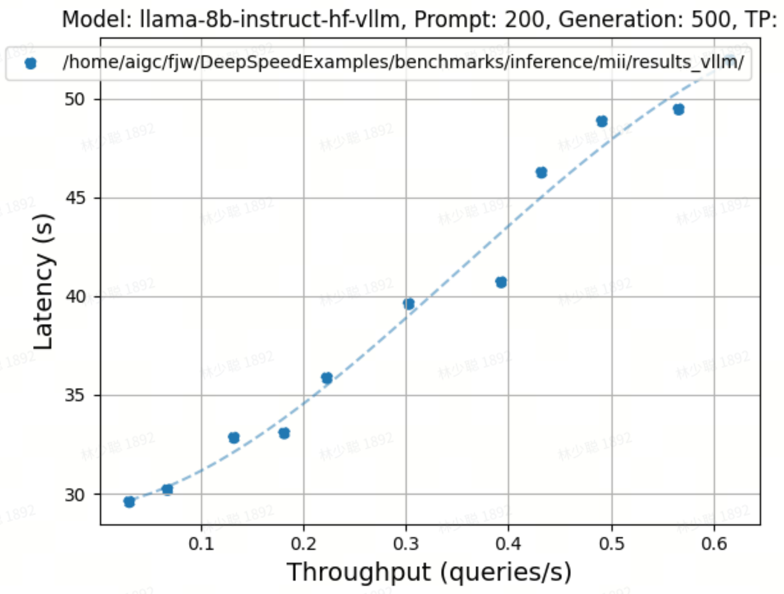
b. 输入200、输出128

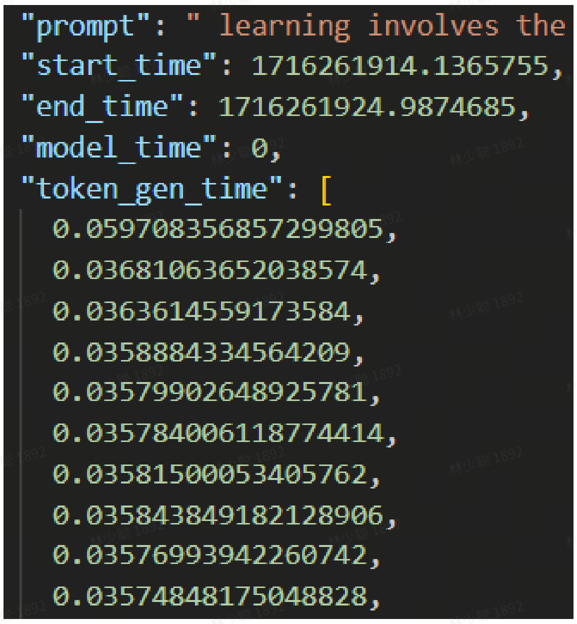




```
"prompt": " learning involves the
"start_time": 1716259439.6629515,
"end_time": 1716259442.4488332,
"model_time": 0,
"token_gen_time": [
  0.07384085655212402,
  0.03729414939880371,
  0.036574363708496094,
  0.03615570068359375,
  0.036019086837768555,
  0.035984992980957031,
  0.036025047302246094,
  0.036102056503295 9,
  0.036023855209350586,
  0.035941362380981445,
  0.036001205444335940,
  0.036021471023559570,
  0.036058902740478516,
  0.035985231399536130,
  0.035981178283691406,
  0.036037921905517580,
```

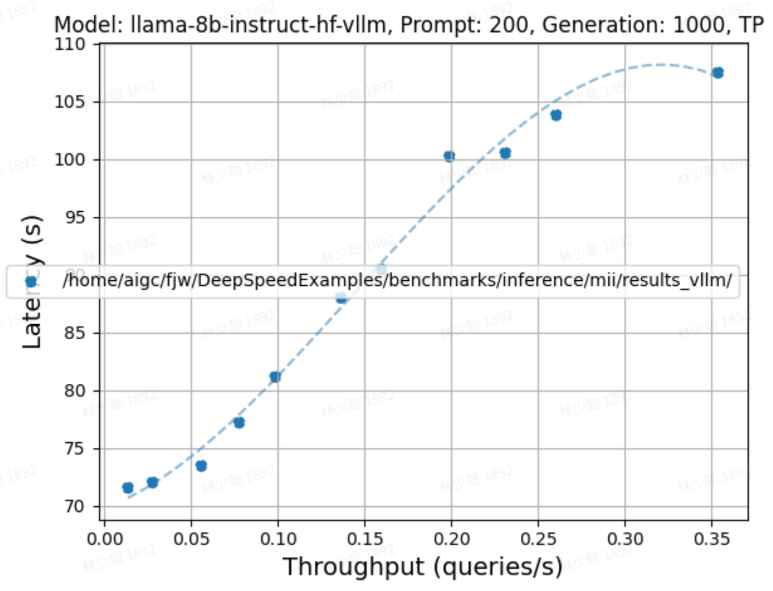
c. 输入200、输出500

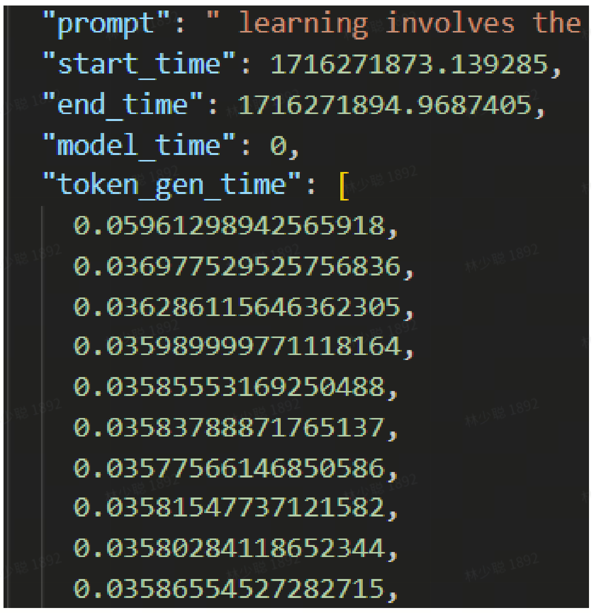




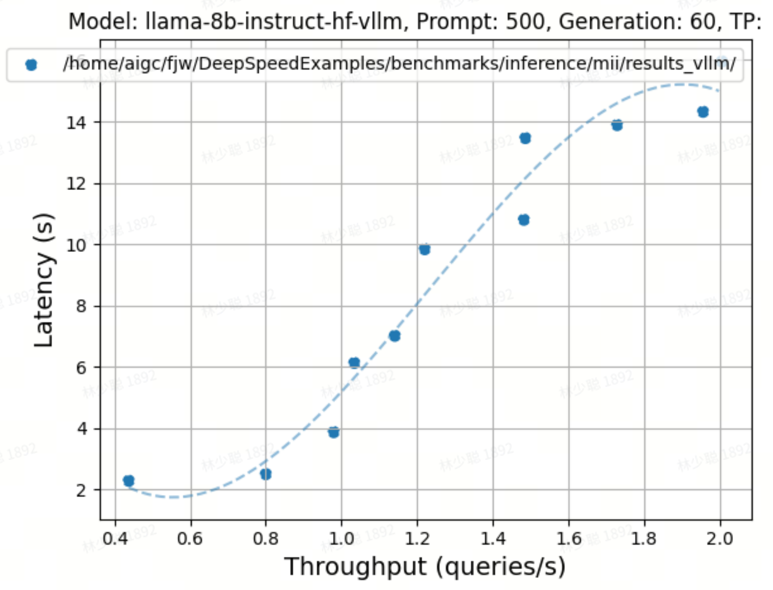
```
"prompt": " learning involves the
"start_time": 1716261914.1365755,
"end_time": 1716261924.9874685,
"model_time": 0,
"token_gen_time": [
  0.059708356857299805,
  0.03681063652038574,
  0.0363614559173584,
  0.03588433456420900,
  0.035799026489257810,
  0.035784006118774414,
  0.03581500053405762,
  0.035843849182128906,
  0.03576993942260742,
  0.03574848175048828,
```

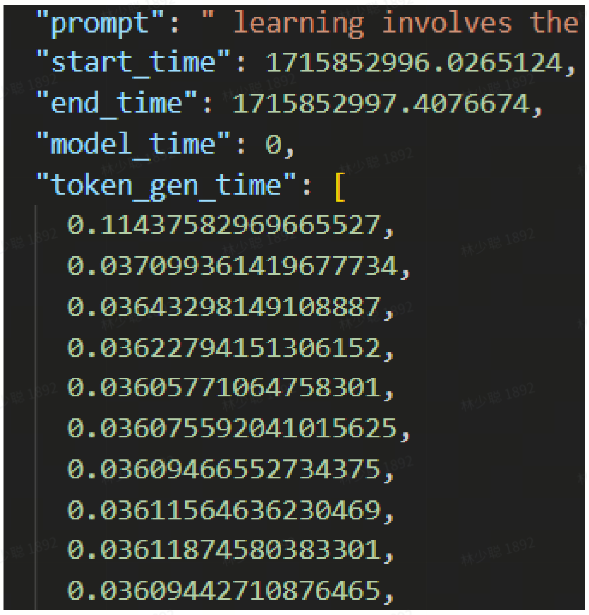
d. 输入200、输出1000



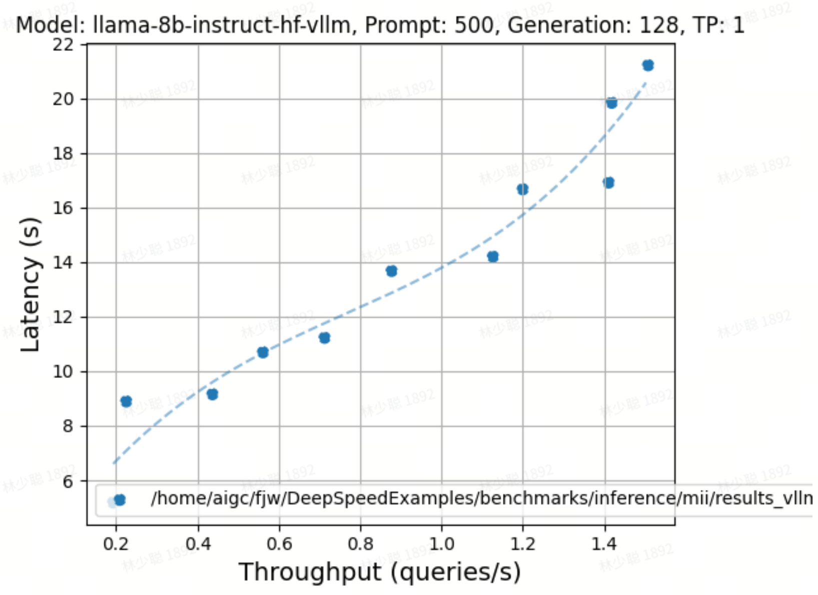

```
"prompt": " learning involves the
"start_time": 1716271873.139285,
"end_time": 1716271894.9687405,
"model_time": 0,
"token_gen_time": [
  0.05961298942565918,
  0.036977529525756836,
  0.036286115646362305,
  0.035989999771118164,
  0.035855531692504883,
  0.03583788871765137,
  0.03577566146850586,
  0.03581547737121582,
  0.03580284118652344,
  0.035865545272827l5,
```

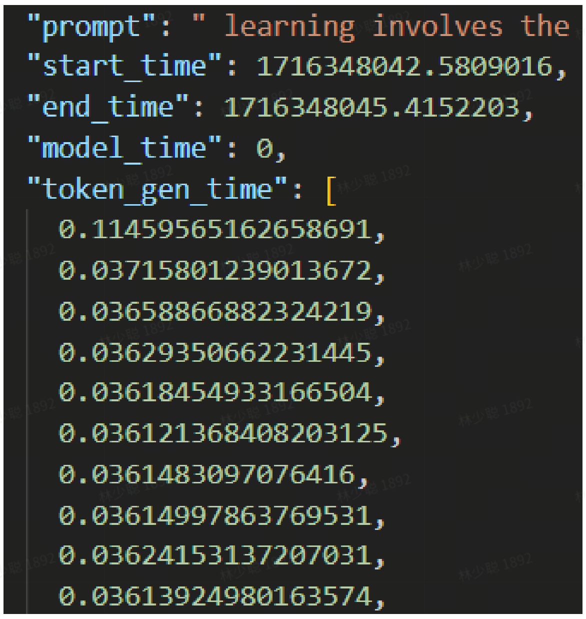
e. 输入500、输出60




```
"prompt": " learning involves the
"start_time": 1715852996.0265124,
"end_time": 1715852997.4076674,
"model_time": 0,
"token_gen_time": [
  0.11437582969665527,
  0.037099361419677734,
  0.03643298149108887,
  0.03622794151306152,
  0.03605771064758301,
  0.036075592041015625,
  0.0360946655273437 5,
  0.036115646362304 69,
  0.03611874580383301,
  0.03609442710876465,
```

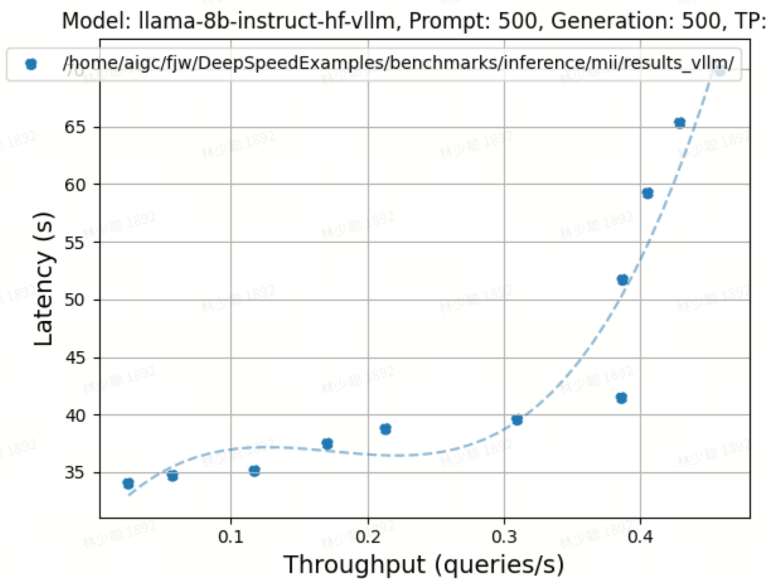
f. 输入500、输出128

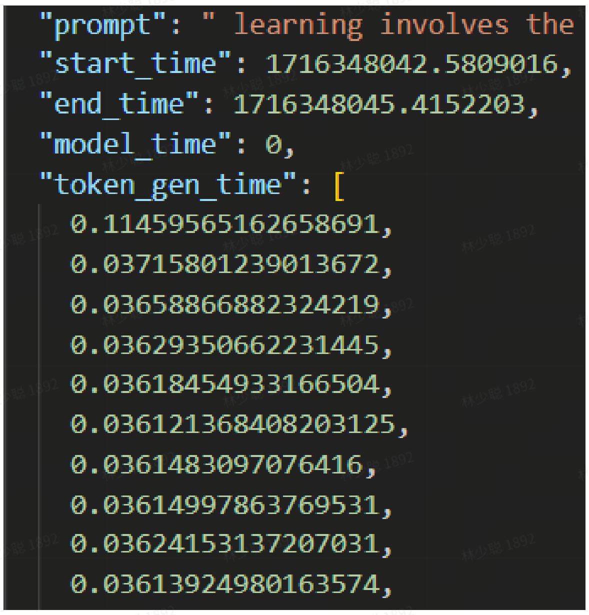




```
"prompt": " learning involves the
"start_time": 1716348042.5809016,
"end_time": 1716348045.4152203,
"model_time": 0,
"token_gen_time": [
  0.11459565162658691,
  0.03715801239013672,
  0.0365886688232421 9,
  0.03629350662231445,
  0.03618454933166504,
  0.036121368408203125,
  0.0361483097076416,
  0.03614997863769531,
  0.03624153137207031,
  0.03613924980163574,
```

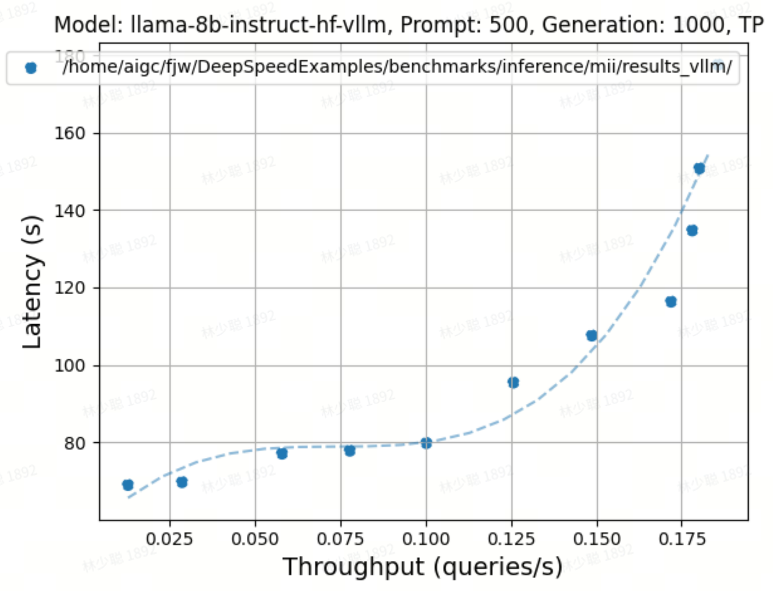
g. 输入500、输出500

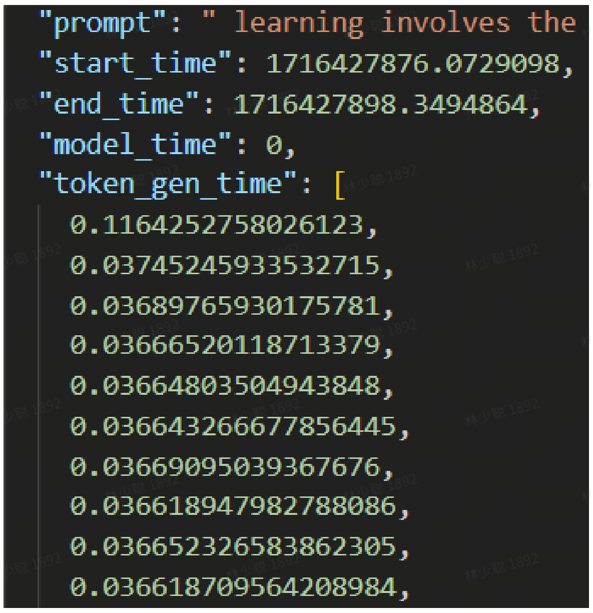




```
"prompt": " learning involves the
"start_time": 1716348042.5809016,
"end_time": 1716348045.4152203,
"model_time": 0,
"token_gen_time": [
  0.11459565162658691,
  0.03715801239013672,
  0.03658866882324219,
  0.03629350662231445,
  0.03618454933166504,
  0.036121368408203125,
  0.0361483097076416,
  0.03614997863769531,
  0.03624153137207031,
  0.03613924980163574,
```

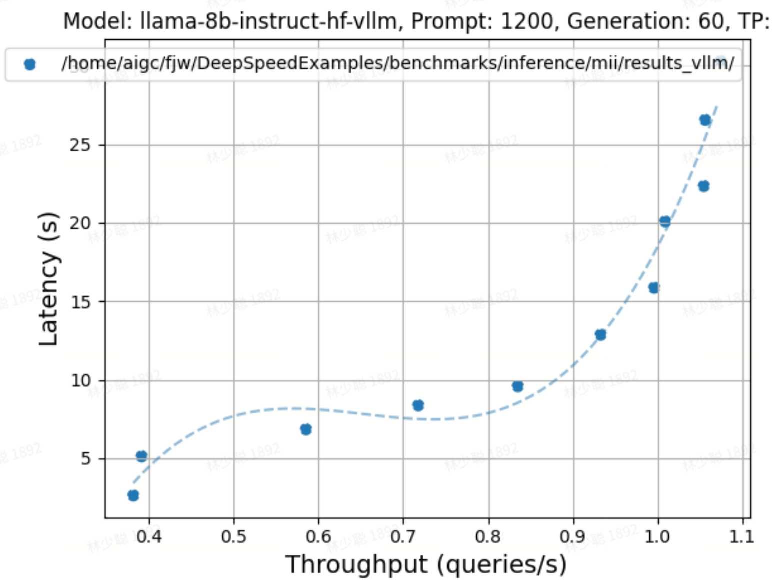
h. 输入500、输出1000

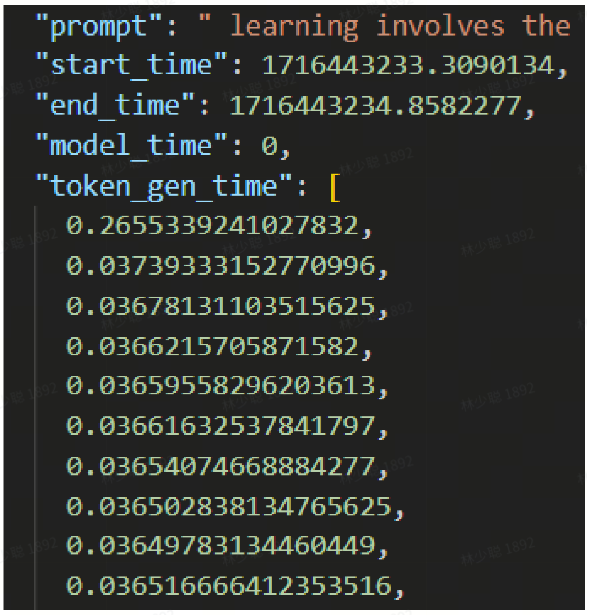




```
"prompt": " learning involves the
"start_time": 1716427876.0729098,
"end_time": 1716427898.3494864,
"model_time": 0,
"token_gen_time": [
    0.1164252758026123,
    0.03745245933532715,
    0.03689765930175781,
    0.03666520118713379,
    0.03664803504943848,
    0.036643266677856445,
    0.03669095039367676,
    0.036618947982788086,
    0.036652326583862305,
    0.036618709564208984,
```

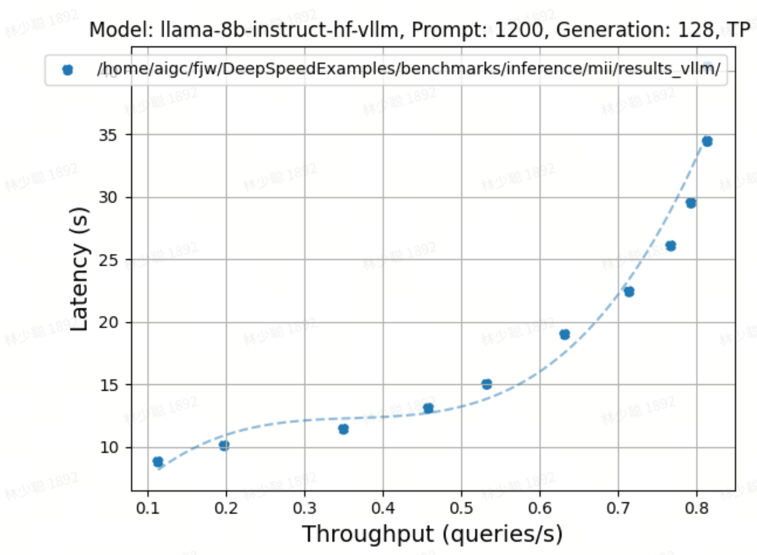
i. 输入1200、输出60

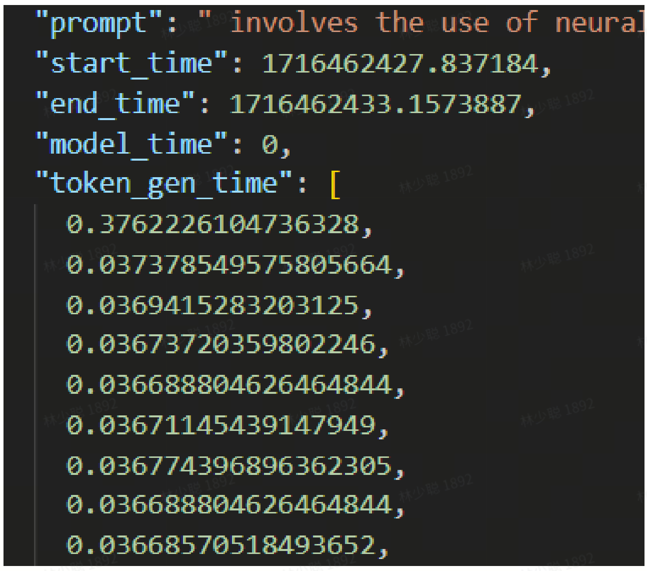




```
"prompt": " learning involves the
"start_time": 1716443233.3090134,
"end_time": 1716443234.8582277,
"model_time": 0,
"token_gen_time": [
    0.2655339241027832,
    0.03739333152770996,
    0.03678131103515625,
    0.0366215705871582,
    0.03659558296203613,
    0.036616325378417970,
    0.036540746688842770,
    0.036502838134765625,
    0.03649783134460449,
    0.036516666412353516,
```

j. 输入1200、输出128





```
"prompt": " involves the use of neural
"start_time": 1716462427.837184,
"end_time": 1716462433.1573887,
"model_time": 0,
"token_gen_time": [
  0.3762226104736328,
  0.037378549575805664,
  0.0369415283203125,
  0.036737203598002246,
  0.036688804626464844,
  0.036711454391479492,
  0.036774396896362305,
  0.036688804626464844,
  0.036685705184936523,
```

5.2、Llama3-8B-Instruct AWQ量化模型性能测试

1. 切到deepspeed性能测试虚拟环境

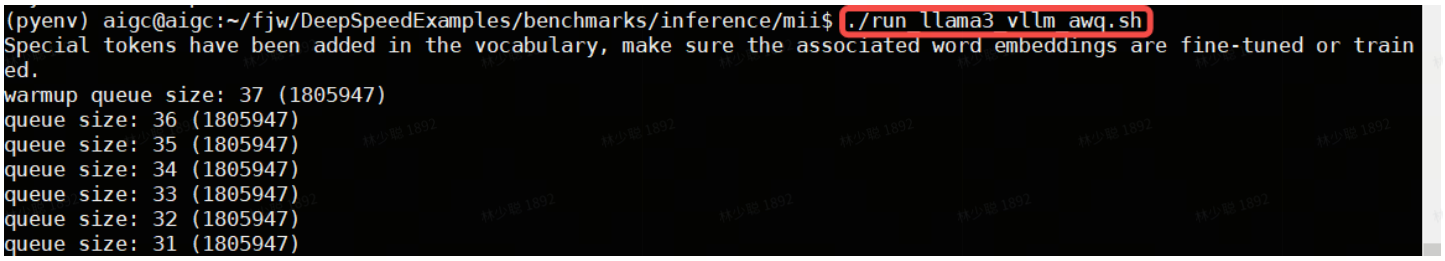
```
source /home/aigc/fjw/pyenv/bin/activate
```

2. 性能测试脚本

```
# Copyright (c) Microsoft Corporation.
# SPDX-License-Identifier: Apache-2.0

# DeepSpeed Team

# Run benchmark
### set arch for A10
export TORCH_CUDA_ARCH_LIST="8.6"
python ./run_benchmark.py \
        --model /home/aigc/lsc/vllm_test/llama-3-8b-instruct-awq-hf \
        --tp_size 1 \
        --num_replicas 1 \
        --max_ragged_batch_size 768 \
        --mean_prompt_length 1200 \
        --mean_max_new_tokens 128 \
        --stream \
        --backend vllm \
        --out_json_dir results_awq\

### Gernerate the plots
```

```
python ./src/plot_th_lat.py \
        --data_dirs
/home/aigc/fjw/DeepSpeedExamples/benchmarks/inference/mii/results_awq_vllm/ \
        --out_dir
/home/aigc/fjw/DeepSpeedExamples/benchmarks/inference/mii/plots_awq_vllm/ \
        --model_name llama-8b-instruct-hf-vllm-awq
```

3. 执行性能测试

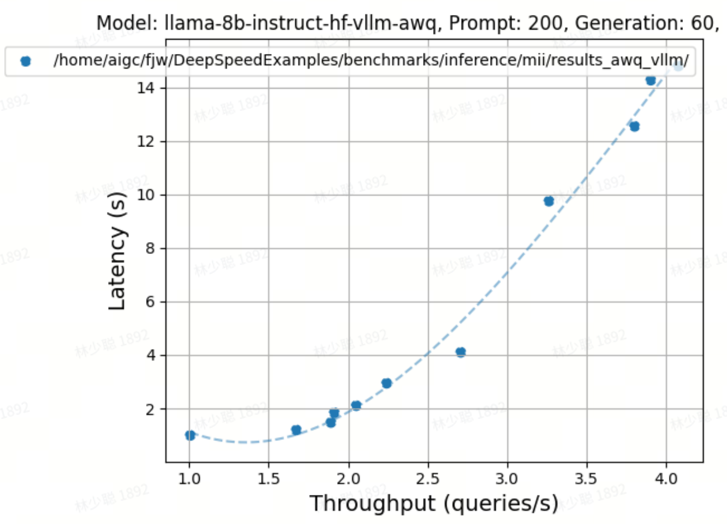


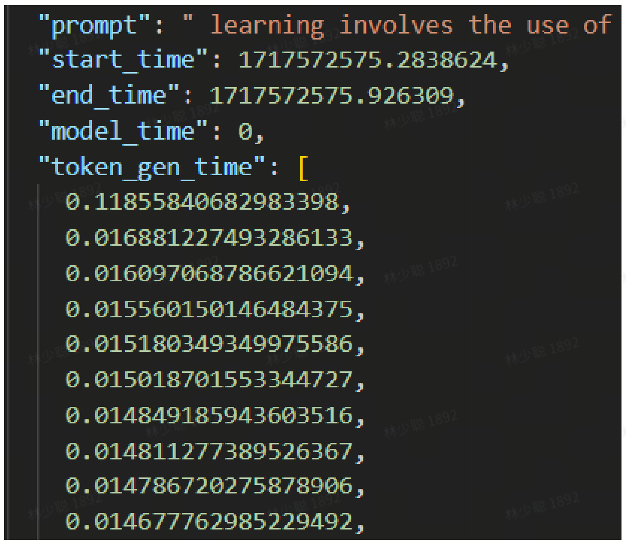
```
(pyenv) aigc@aigc:~/fjw/DeepSpeedExamples/benchmarks/inference/mii$ ./run_llama3_vllm_awq.sh
Special tokens have been added in the vocabulary, make sure the associated word embeddings are fine-tuned or train
ed.
warmup queue size: 37 (1805947)
queue size: 36 (1805947)
queue size: 35 (1805947)
queue size: 34 (1805947)
queue size: 33 (1805947)
queue size: 32 (1805947)
queue size: 31 (1805947)
```

4. 性能测试结果

测试选择输入200、500、1200与输出60、128、500、1000组合（1200输入时不包含500、1000输出）

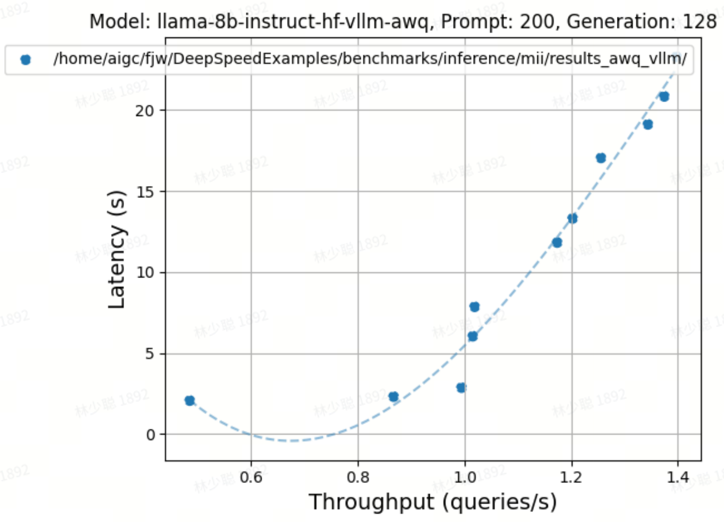
a. 输入200、输出60

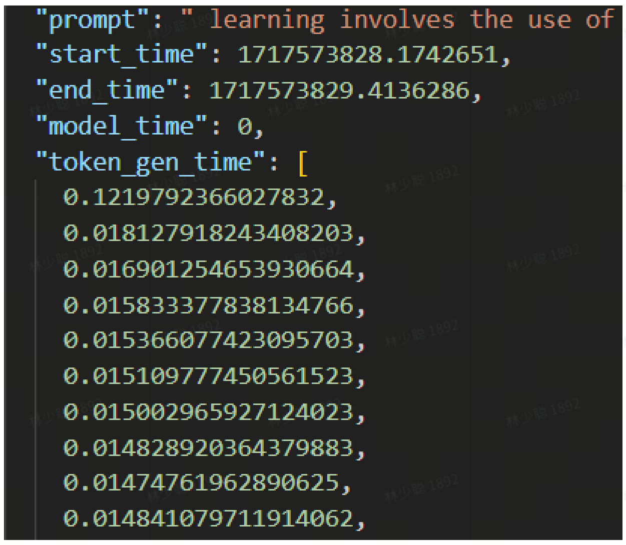




```
"prompt": " learning involves the use of
"start_time": 1717572575.2838624,
"end_time": 1717572575.926309,
"model_time": 0,
"token_gen_time": [
  0.11855840682983398,
  0.016881227493286133,
  0.016097068786621094,
  0.015560150146484375,
  0.015180349349975586,
  0.015018701553344727,
  0.014849185943603516,
  0.014811277389526367,
  0.014786720275878906,
  0.014677762985229492,
```

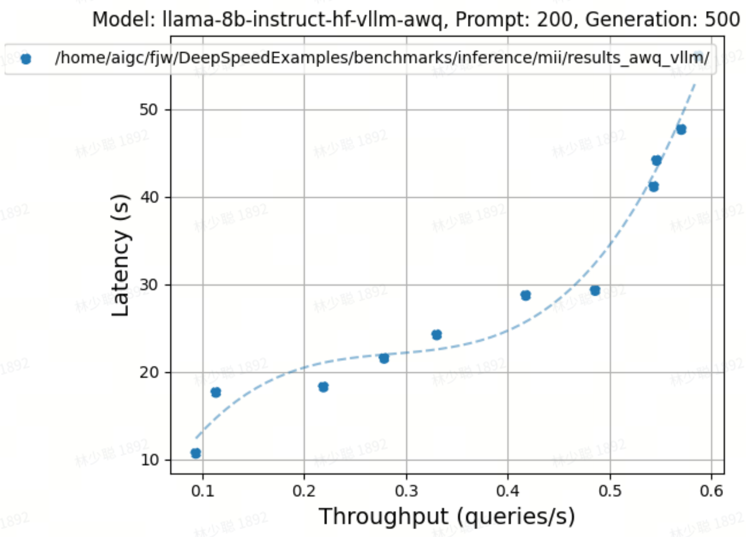
b. 输入200、输出128

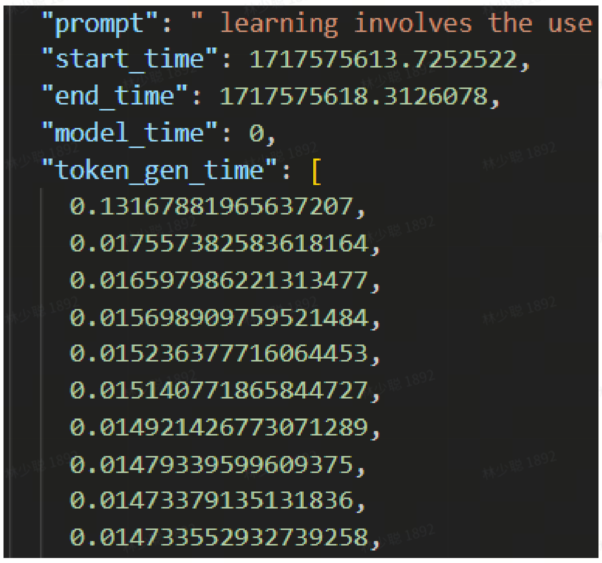




```
"prompt": " learning involves the use of
"start_time": 1717573828.1742651,
"end_time": 1717573829.4136286,
"model_time": 0,
"token_gen_time": [
  0.1219792366027832,
  0.018127918243408203,
  0.016901254653930664,
  0.015833377838134766,
  0.015366077423095703,
  0.015109777450561523,
  0.015002965927124023,
  0.014828920364379883,
  0.01474761962890625,
  0.014841079711914062,
```

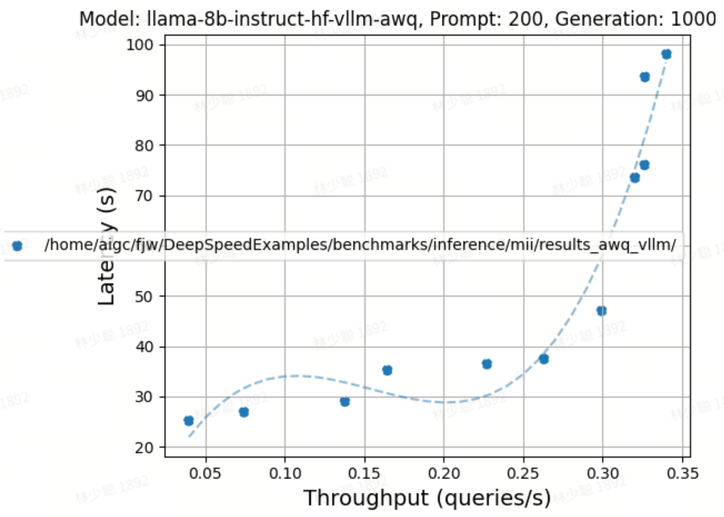
c. 输入200、输出500

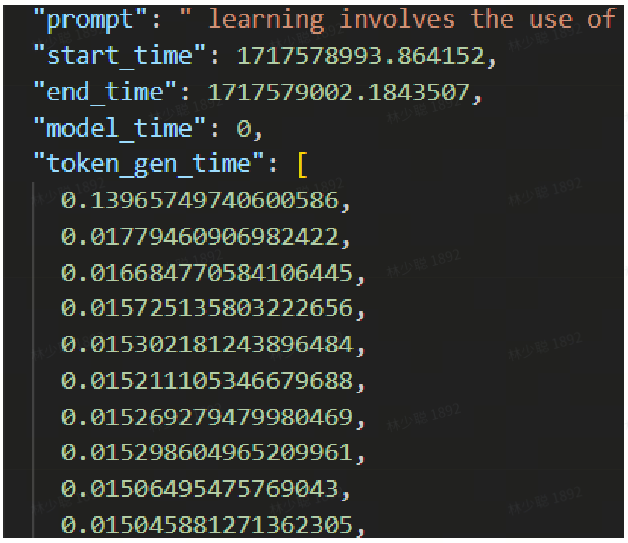




```
"prompt": " learning involves the use
"start_time": 1717575613.7252522,
"end_time": 1717575618.3126078,
"model_time": 0,
"token_gen_time": [
  0.13167881965637207,
  0.017557382583618164,
  0.016597986221313477,
  0.015698909759521484,
  0.015236377716064453,
  0.015140771865844727,
  0.014921426773071289,
  0.014793395996093750,
  0.014733791351318360,
  0.014733552932739258,
```

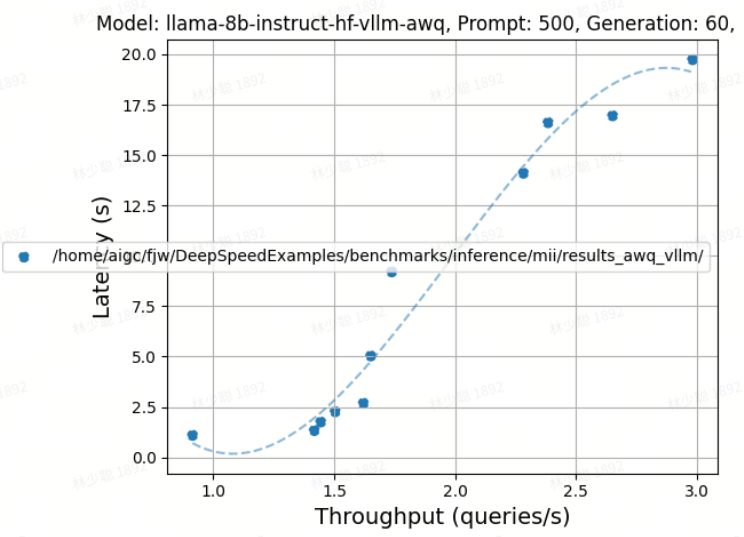
d. 输入200、输出1000

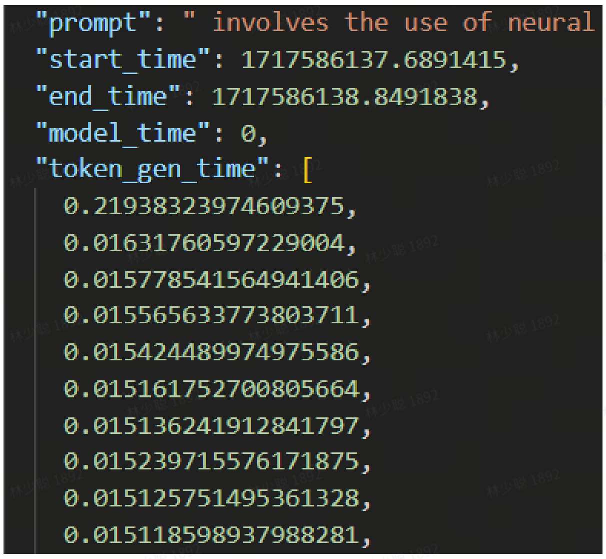




```
"prompt": " learning involves the use of
"start_time": 1717578993.864152,
"end_time": 1717579002.1843507,
"model_time": 0,
"token_gen_time": [
  0.13965749740600586,
  0.01779460906982422,
  0.016684770584106445,
  0.015725135803222656,
  0.015302181243896484,
  0.015211053466796880,
  0.015269279479980469,
  0.015298604965209961,
  0.01506495475769043,
  0.015045881271362305,
```

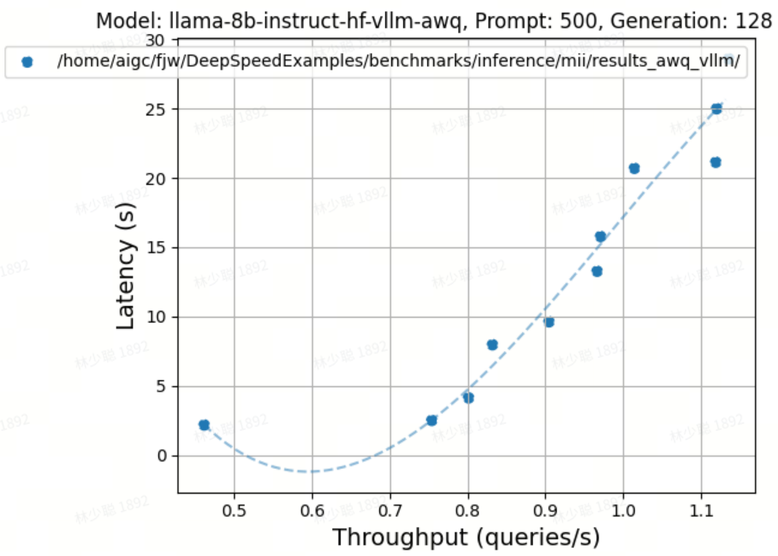
e. 输入500、输出60

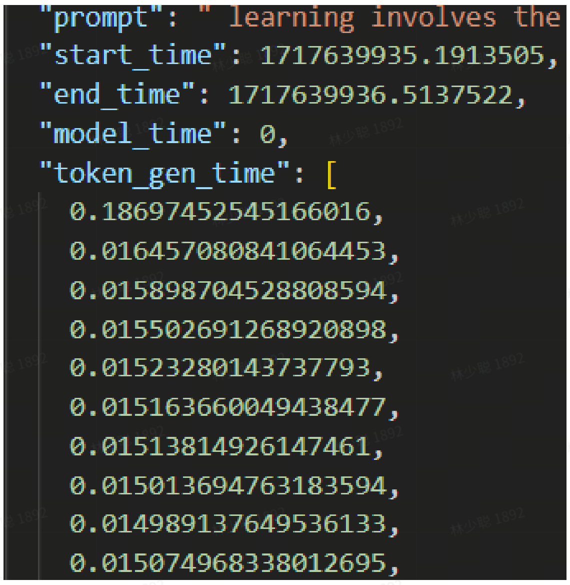




```
"prompt": " involves the use of neural
"start_time": 1717586137.6891415,
"end_time": 1717586138.8491838,
"model_time": 0,
"token_gen_time": [
  0.21938323974609375,
  0.016317605972290004,
  0.015778541564941406,
  0.015565633773803711,
  0.015424489974975586,
  0.015161752700805664,
  0.015136241912841797,
  0.015239715576171875,
  0.015125751495361328,
  0.015118598937988281,
```

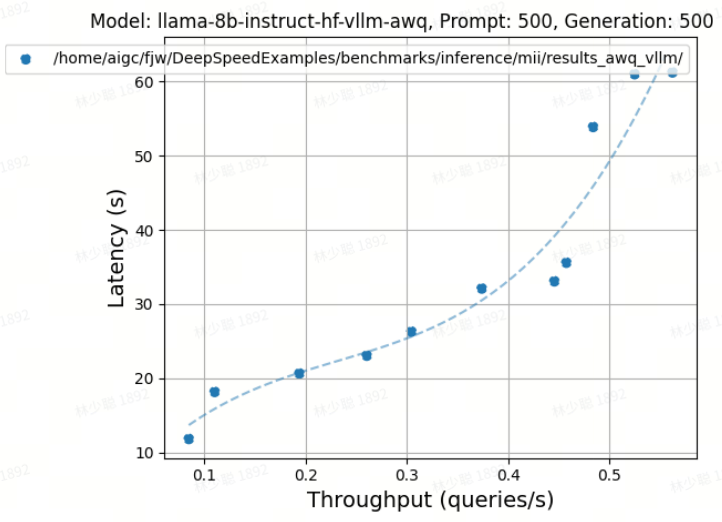
f. 输入500、输出128

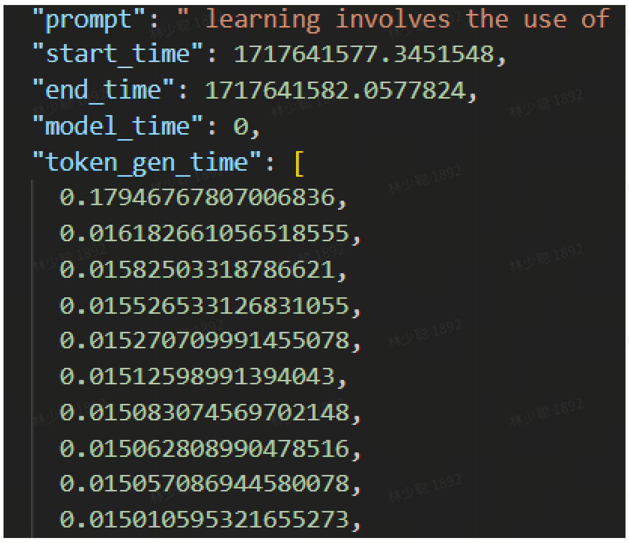




```
"prompt": " learning involves the
"start_time": 1717639935.1913505,
"end_time": 1717639936.5137522,
"model_time": 0,
"token_gen_time": [
  0.18697452545166016,
  0.016457080841064453,
  0.015898704528808594,
  0.015502691268920898,
  0.015232801437377930,
  0.015163660049438477,
  0.015138149261474610,
  0.015013694763183594,
  0.014989137649536133,
  0.015074968338012695,
```

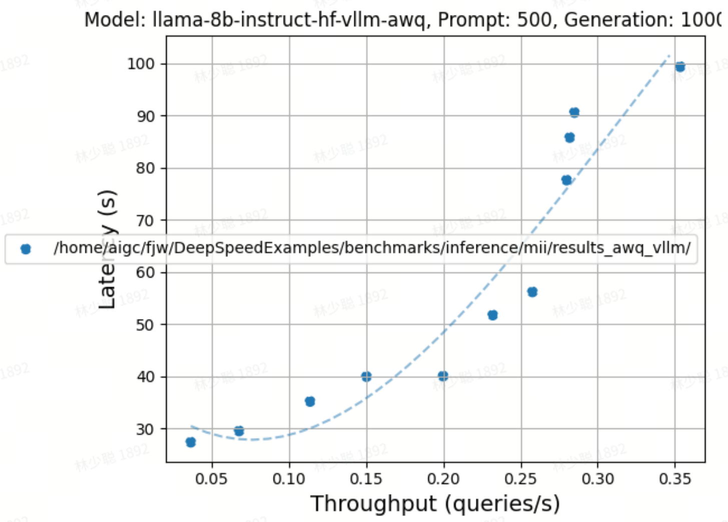
g. 输入500、输出500

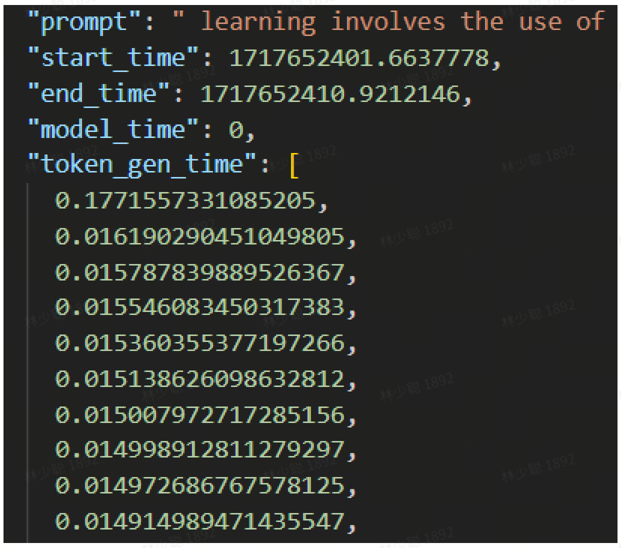




```
"prompt": " learning involves the use of
"start_time": 1717641577.3451548,
"end_time": 1717641582.0577824,
"model_time": 0,
"token_gen_time": [
  0.1794676780700683б,
  0.01618266105651855‌5,
  0.0158250331878662‌1,
  0.01552653312683105‌5,
  0.01527070991455078,
  0.01512598991394043,
  0.01508307456970214‌8,
  0.01506280899047851‌6,
  0.01505708694458007‌8,
  0.01501059532165527‌3,
```

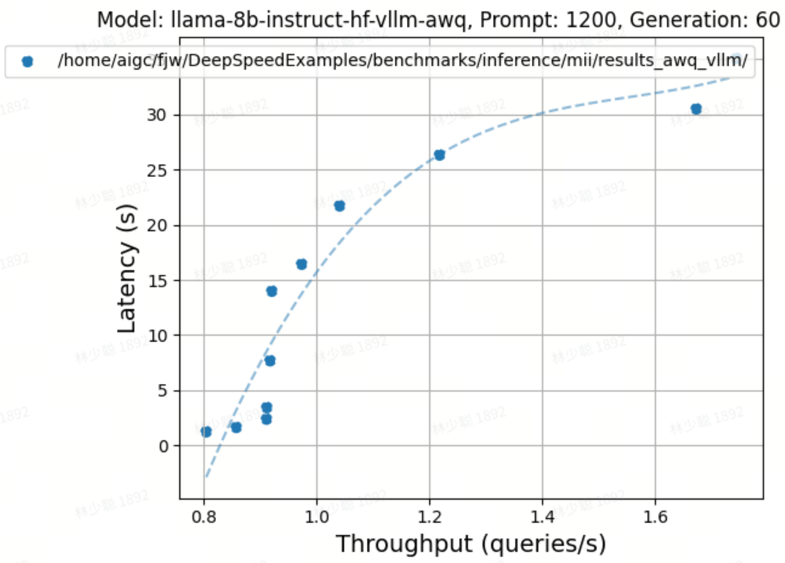
h. 输入500、输出1000

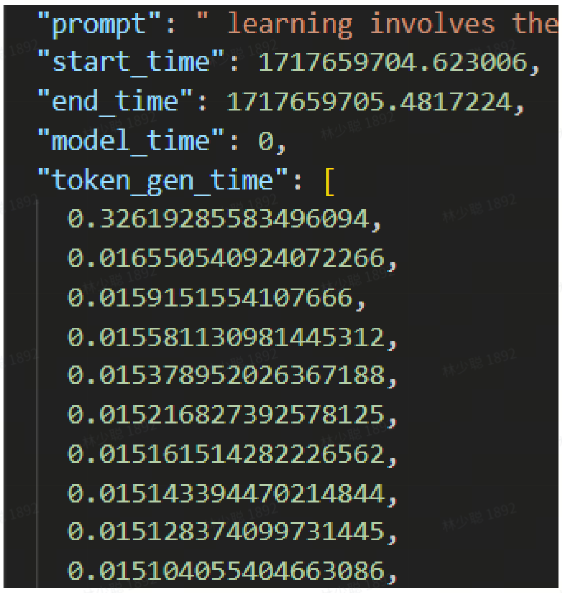




```
"prompt": " learning involves the use of
"start_time": 1717652401.6637778,
"end_time": 1717652410.9212146,
"model_time": 0,
"token_gen_time": [
  0.1771557331085205,
  0.016190290451049805,
  0.015787839889526367,
  0.015546083450317383,
  0.015360355377197266,
  0.015138626098632812,
  0.015007972717285156,
  0.014998912811279297,
  0.014972686767578125,
  0.014914989471435547,
```

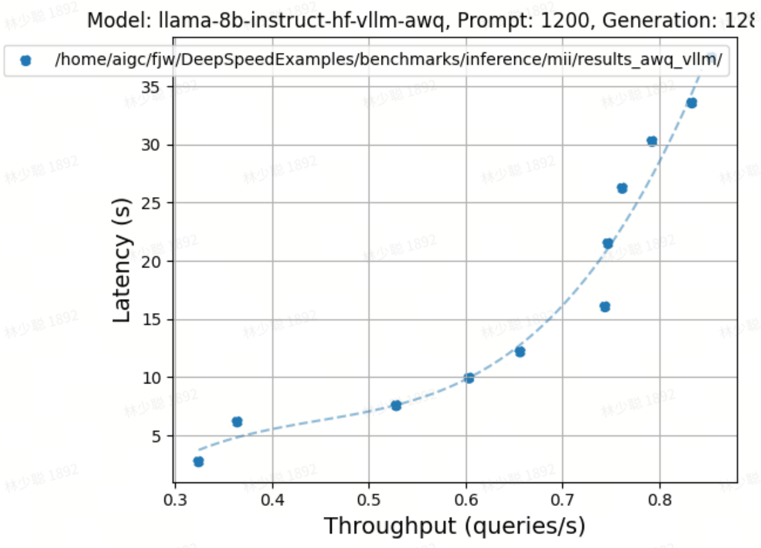
i. 输入1200、输出60

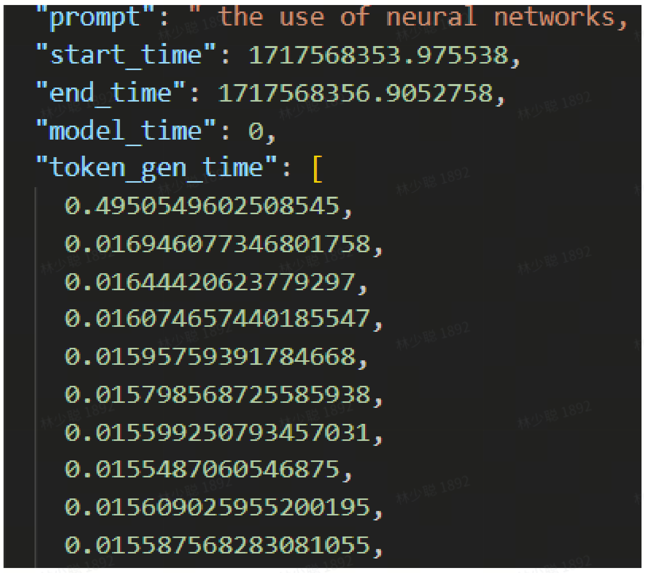




```
"prompt": " learning involves the
"start_time": 1717659704.623006,
"end_time": 1717659705.4817224,
"model_time": 0,
"token_gen_time": [
  0.32619285583496094,
  0.016550540924072266,
  0.015915155410766б,
  0.015581130981445312,
  0.015378952026367188,
  0.015216827392578125,
  0.015161514282226562,
  0.015143394470214844,
  0.015128374099731445,
  0.015104055404663086,
```

j. 输入1200、输出128



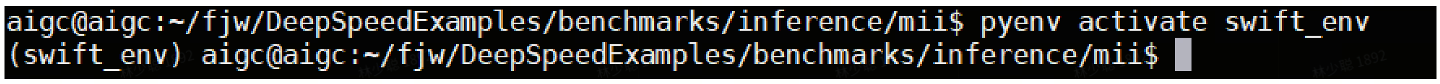


```
"prompt": " the use of neural networks,
"start_time": 1717568353.975538,
"end_time": 1717568356.9052758,
"model_time": 0,
"token_gen_time": [
  0.4950549602508545,
  0.016946077346801758,
  0.016444206237792297,
  0.016074657440185547,
  0.01595759391784668,
  0.015798568725585938,
  0.015599250793457031,
  0.0155487060546875,
  0.015609025955200195,
  0.015587568283081055,
```

5.3、Qwen2-7B-Instruct AWQ量化模型性能测试

1. 激活swift_env pyenv环境，带VLLM推理后端

```
pyenv activate swift_env

#关闭代理
unset http_proxy
unset https_proxy
```

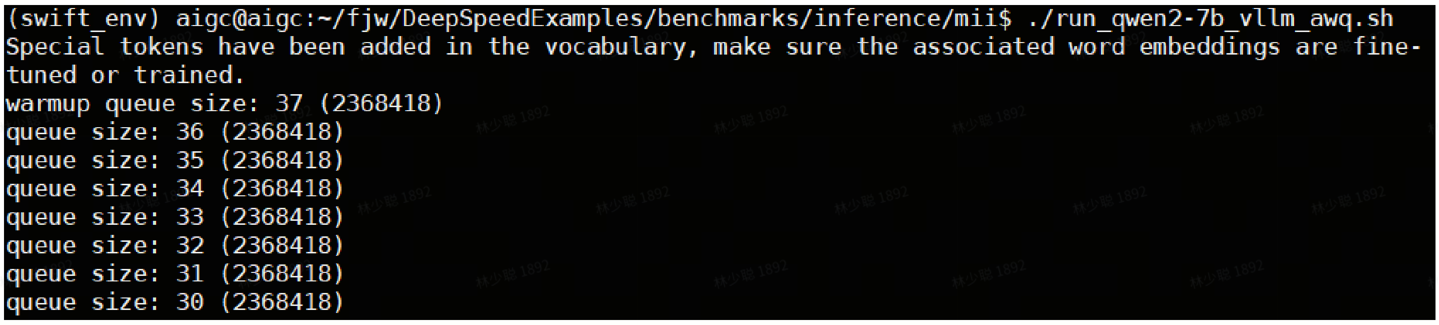


```
aigc@aigc:~/fjw/DeepSpeedExamples/benchmarks/inference/mii$ pyenv activate swift_env
(swift_env) aigc@aigc:~/fjw/DeepSpeedExamples/benchmarks/inference/mii$
```

2. 性能测试脚本

```
# Copyright (c) Microsoft Corporation.
# SPDX-License-Identifier: Apache-2.0

# DeepSpeed Team

# Run benchmark
### set arch for A10
export TORCH_CUDA_ARCH_LIST="8.6"
python ./run_benchmark.py \
        --model /home/aigc/lsc/vllm_test/Qwen2-7B-Instruct_AWQ_Int4 \
        --tp_size 1 \
        --num_replicas 1 \
        --max_ragged_batch_size 768 \
        --mean_prompt_length 200 \
        --mean_max_new_tokens 500 \
        --stream \
        --backend vllm \
        --out_json_dir results_qwen2_awq \

### Gernerate the plots
python ./src/plot_th_lat.py \
        --data_dirs
/home/aigc/fjw/DeepSpeedExamples/benchmarks/inference/mii/results_qwen2_awq_vllm/ \
        --out_dir
/home/aigc/fjw/DeepSpeedExamples/benchmarks/inference/mii/plots_qwen2_awq_vllm/ \
        --model_name qwen2-7b-vllm-awq
```

3. 执行性能测试

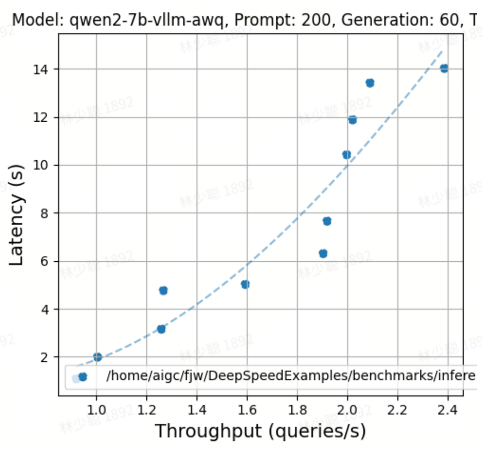


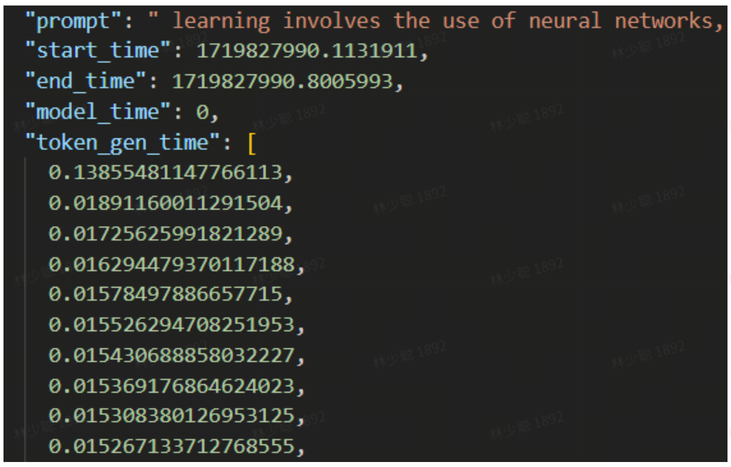
```
(swift_env) aigc@aigc:~/fjw/DeepSpeedExamples/benchmarks/inference/mii$ ./run_qwen2-7b_vllm_awq.sh
Special tokens have been added in the vocabulary, make sure the associated word embeddings are fine-
tuned or trained.
warmup queue size: 37 (2368418)
queue size: 36 (2368418)
queue size: 35 (2368418)
queue size: 34 (2368418)
queue size: 33 (2368418)
queue size: 32 (2368418)
queue size: 31 (2368418)
queue size: 30 (2368418)
```

4. 性能测试结果

测试选择输入200、500、1200与输出60、128、500、1000组合（1200输入时不包含500、1000输出）

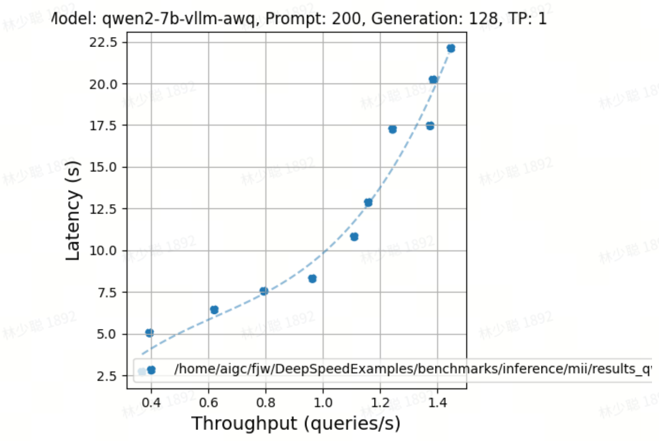
a. 输入200、输出60

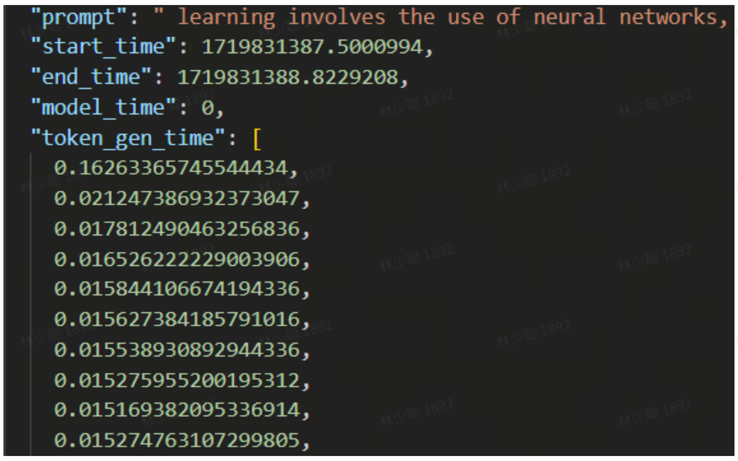




```
"prompt": " learning involves the use of neural networks,
"start_time": 1719827990.1131911,
"end_time": 1719827990.8005993,
"model_time": 0,
"token_gen_time": [
  0.13855481147766113,
  0.01891160011291504,
  0.01725625991821289,
  0.016294479370117188,
  0.015784978866577715,
  0.015526294708251953,
  0.015430688858032227,
  0.015369176864624023,
  0.015308380126953125,
  0.015267133712768555,
```

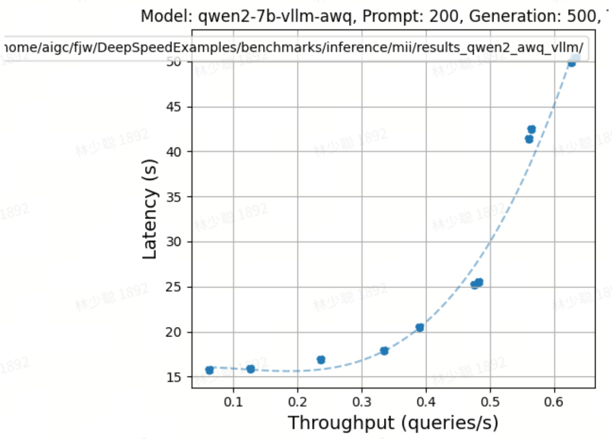
b. 输入200、输出128

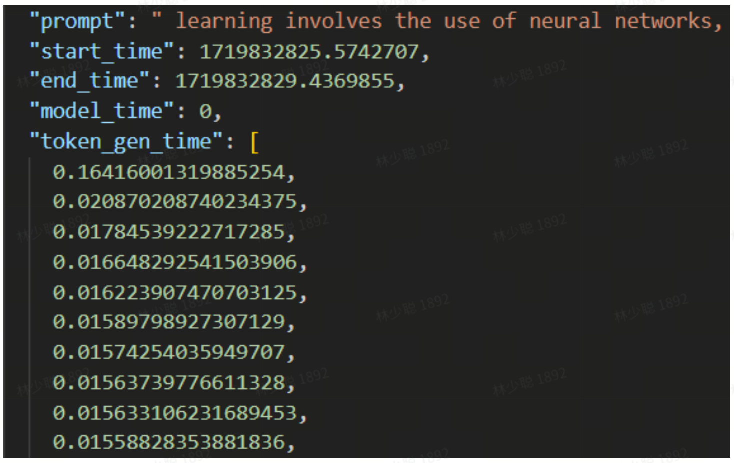




```
"prompt": " learning involves the use of neural networks,
"start_time": 1719831387.5000994,
"end_time": 1719831388.8229208,
"model_time": 0,
"token_gen_time": [
  0.16263365745544434,
  0.021247386932373047,
  0.017812490463256836,
  0.016526222229003906,
  0.015844106674194336,
  0.015627384185791016,
  0.015538930892944336,
  0.015275955200195312,
  0.015169382095336914,
  0.015274763107299805,
```

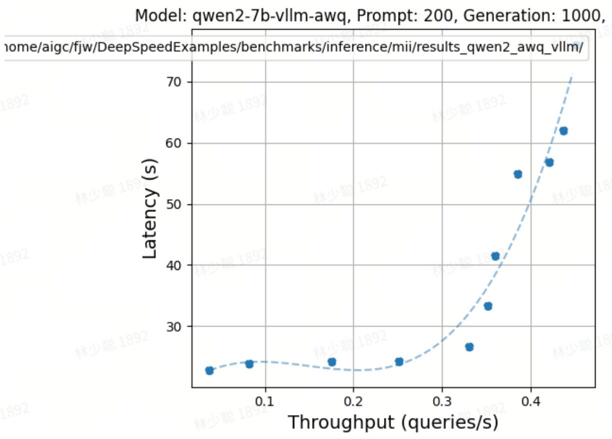
c. 输入200、输出500

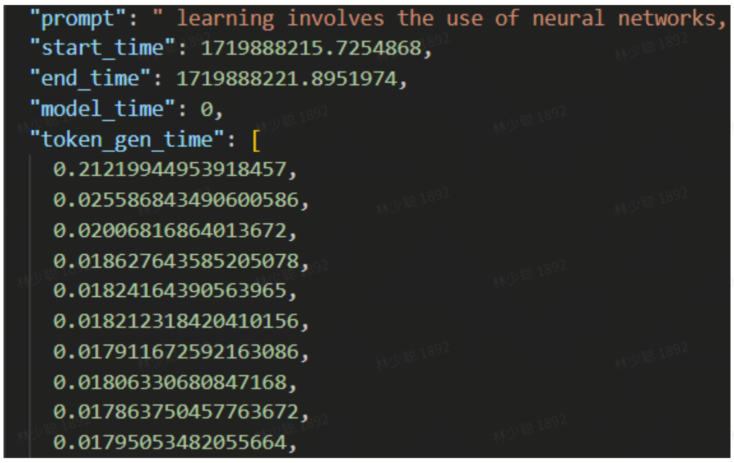




```
"prompt": " learning involves the use of neural networks,
"start_time": 1719832825.5742707,
"end_time": 1719832829.4369855,
"model_time": 0,
"token_gen_time": [
  0.16416001319885254,
  0.020870208740234375,
  0.017845392227172852,
  0.016648292541503906,
  0.016223907470703125,
  0.01589798927307129,
  0.01574254035949707,
  0.015637397766113228,
  0.015633106231689453,
  0.015588283538818836,
```

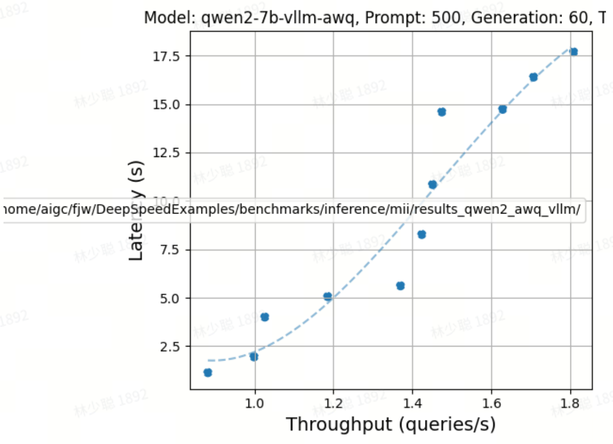
d. 输入200、输出1000

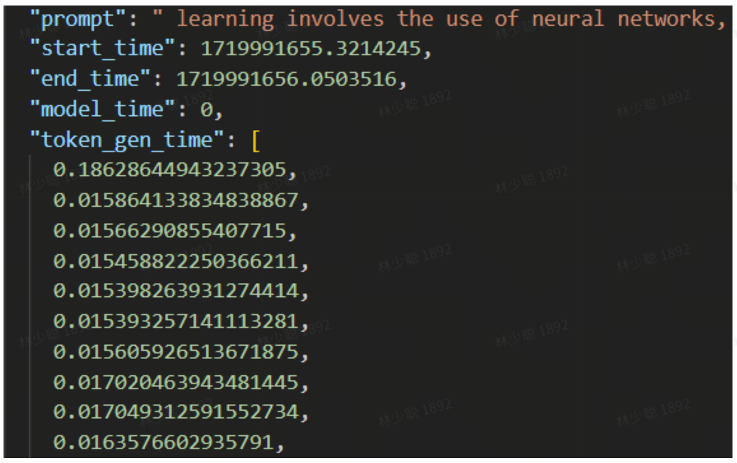




```
"prompt": " learning involves the use of neural networks,
"start_time": 1719888215.7254868,
"end_time": 1719888221.8951974,
"model_time": 0,
"token_gen_time": [
  0.21219944953918457,
  0.025586843490600586,
  0.02006816864013672,
  0.018627643585205078,
  0.01824164390563965,
  0.018212318420410156,
  0.017911672592163086,
  0.018063306808471680,
  0.017863750457763672,
  0.017950534820556640,
```

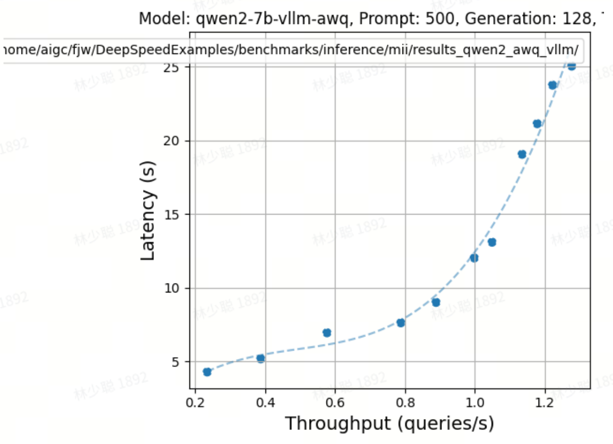
e. 输入500、输出60

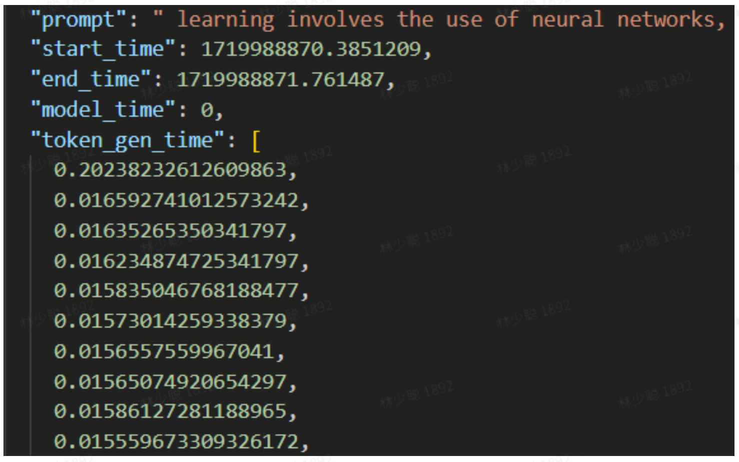




```
"prompt": " learning involves the use of neural networks,
"start_time": 1719991655.3214245,
"end_time": 1719991656.0503516,
"model_time": 0,
"token_gen_time": [
  0.18628644943237305,
  0.015864133834838867,
  0.01566290855407715,
  0.015458822250366211,
  0.015398263931274414,
  0.015393257141113281,
  0.015605926513671875,
  0.017020463943481445,
  0.017049312591552734,
  0.0163576602935791,
```

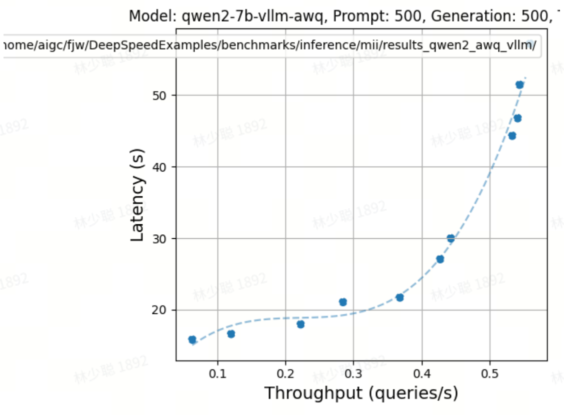
f. 输入500、输出128

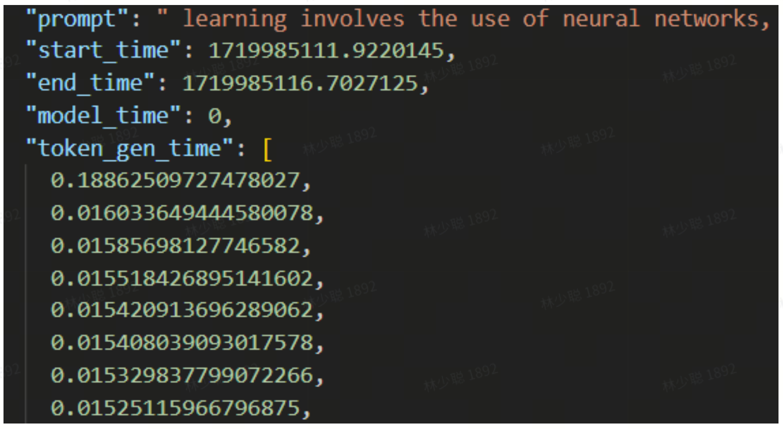




```
"prompt": " learning involves the use of neural networks,
"start_time": 1719988870.3851209,
"end_time": 1719988871.761487,
"model_time": 0,
"token_gen_time": [
  0.20238232612609863,
  0.016592741012573242,
  0.01635265350341797,
  0.016234874725341797,
  0.015835046768188477,
  0.01573014259338379,
  0.0156557559967041,
  0.01565074920654297,
  0.015861272811889648,
  0.015559673309326172,
```

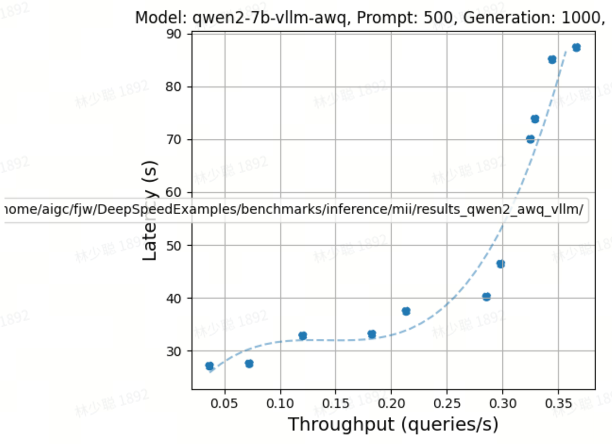
g. 输入500、输出500

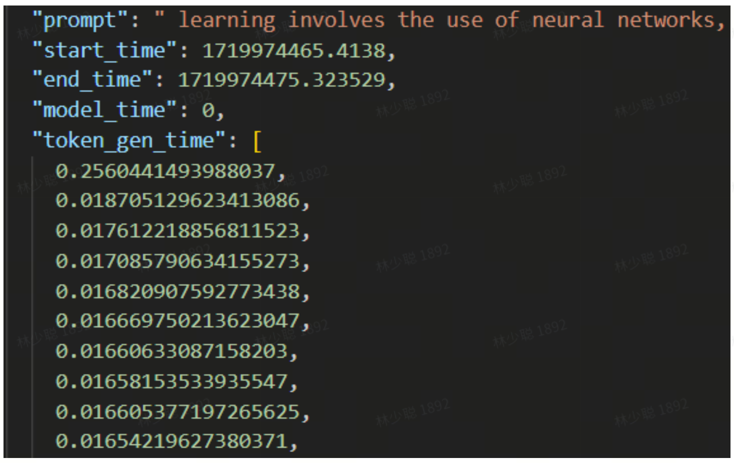




```
"prompt": " learning involves the use of neural networks,
"start_time": 1719985111.9220145,
"end_time": 1719985116.7027125,
"model_time": 0,
"token_gen_time": [
  0.18862509727478027,
  0.016033649444580078,
  0.015856981277465820,
  0.015518426895141602,
  0.015420913696289062,
  0.015408039093017578,
  0.015329837799072266,
  0.015251159667968750,
```

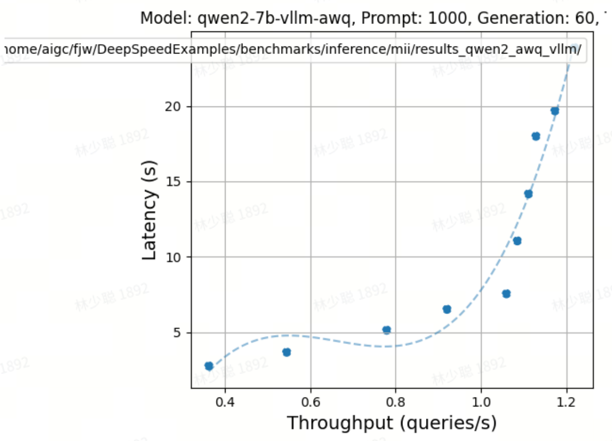
h. 输入500、输出1000

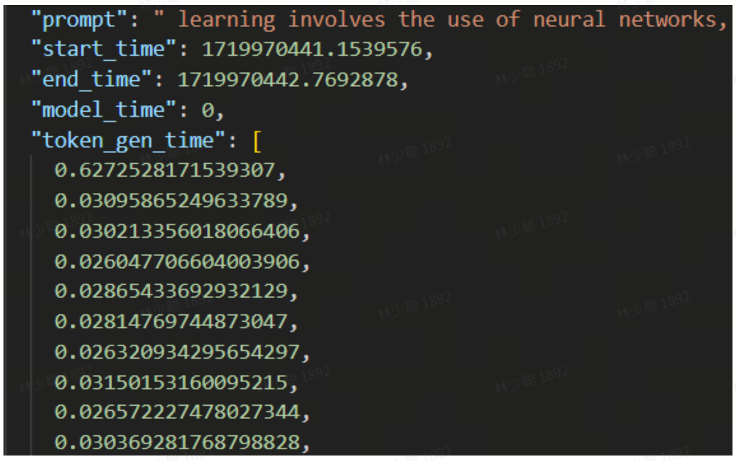




```
"prompt": " learning involves the use of neural networks,
"start_time": 1719974465.4138,
"end_time": 1719974475.323529,
"model_time": 0,
"token_gen_time": [
  0.2560441493988037,
  0.018705129623413086,
  0.017612218856811523,
  0.017085790634155273,
  0.016820907592773438,
  0.016669750213623047,
  0.01660633087158203,
  0.01658153533935547,
  0.016605377197265625,
  0.01654219627380371,
```

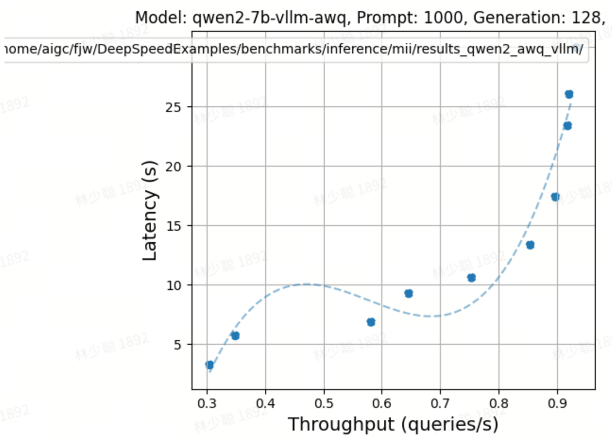
i. 输入1000、输出60（--num_clients无32）

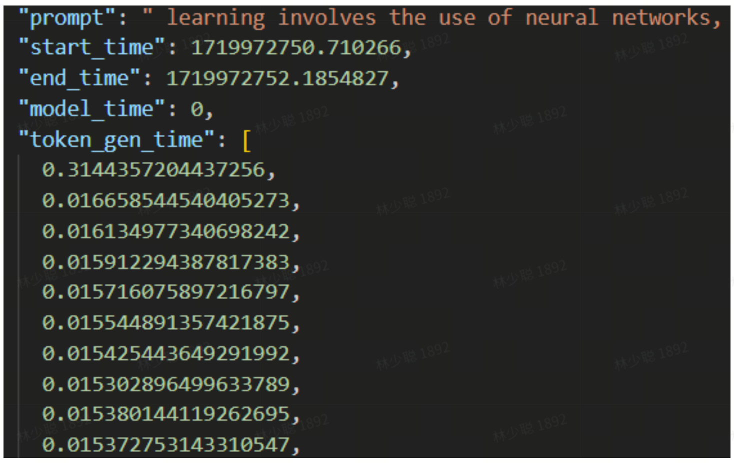




```
"prompt": " learning involves the use of neural networks,
"start_time": 1719970441.1539576,
"end_time": 1719970442.7692878,
"model_time": 0,
"token_gen_time": [
  0.6272528171539307,
  0.03095865249633789,
  0.030213356018066406,
  0.026047706604003906,
  0.02865433692932129,
  0.02814769744873047,
  0.026320934295654297,
  0.03150153160095215,
  0.026572227478027344,
  0.030369281768798828,
```

j. 输入1000、输出128（--num_clients无32）



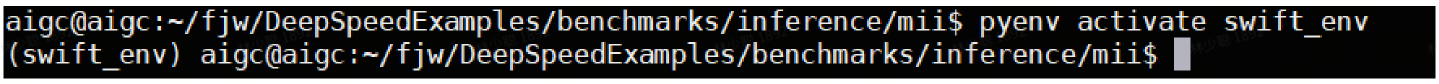


```
"prompt": " learning involves the use of neural networks,
"start_time": 1719972750.710266,
"end_time": 1719972752.1854827,
"model_time": 0,
"token_gen_time": [
  0.3144357204437256,
  0.016658544540405273,
  0.016134977340698242,
  0.015912294387817383,
  0.015716075897216797,
  0.015544891357421875,
  0.015425443649291992,
  0.015302896499633789,
  0.015380144119262695,
  0.015372753143310547,
```

5.4、Qwen2-7B-Instruct GPTQ量化模型性能测试

1. 激活swift_env pyenv环境，带VLLM推理后端

```
pyenv activate swift_env
```

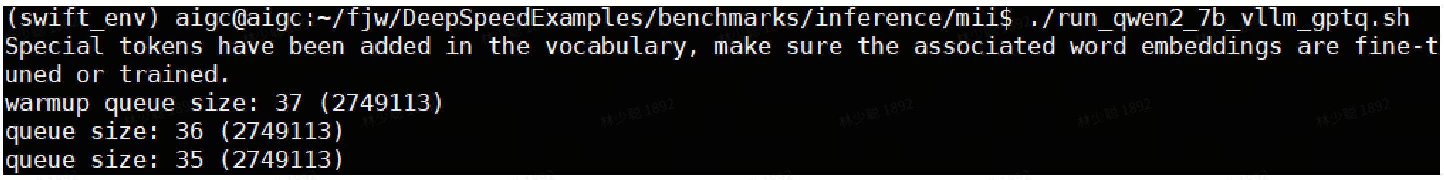


```
aigc@aigc:~/fjw/DeepSpeedExamples/benchmarks/inference/mii$ pyenv activate swift_env
(swift_env) aigc@aigc:~/fjw/DeepSpeedExamples/benchmarks/inference/mii$
```

2. 性能测试脚本

```
# Copyright (c) Microsoft Corporation.
# SPDX-License-Identifier: Apache-2.0

# DeepSpeed Team

# Run benchmark
### set arch for A10
export TORCH_CUDA_ARCH_LIST="8.6"
python ./run_benchmark.py \
        --model /home/aigc/lsc/vllm_test/Qwen2-7B-Instruct_ _Int4 \
        --tp_size 1 \
        --num_replicas 1 \
        --max_ragged_batch_size 768 \
        --mean_prompt_length 200 \
        --mean_max_new_tokens 60 \
        --stream \
        --backend vllm \
        --out_json_dir results_qwen2_gptq \

### Gernerate the plots
python ./src/plot_th_lat.py \
        --data_dirs /home/aigc/fjw/DeepSpeedExamples/benchmarks/inference/mii/results_qwen2_gptq_vllm/ \
        --out_dir /home/aigc/fjw/DeepSpeedExamples/benchmarks/inference/mii/plots_qwen2_gptq_vllm/ \
        --model_name qwen2-7b-vllm-gptq
```

3. 执行性能测试

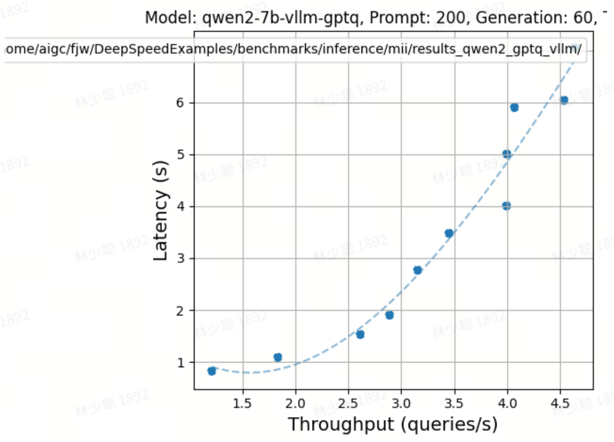


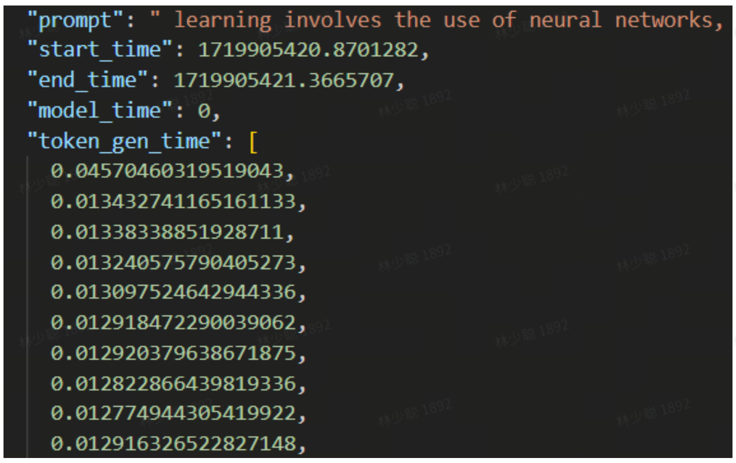
```
(swift_env) aigc@aigc:~/fjw/DeepSpeedExamples/benchmarks/inference/mii$ ./run_qwen2_7b_vllm_gptq.sh
Special tokens have been added in the vocabulary, make sure the associated word embeddings are fine-t
uned or trained.
warmup queue size: 37 (2749113)
queue size: 36 (2749113)
queue size: 35 (2749113)
```

4. 性能测试结果

测试选择输入200、500、1200与输出60、128、500、1000组合（1200输入时不包含500、1000输出）

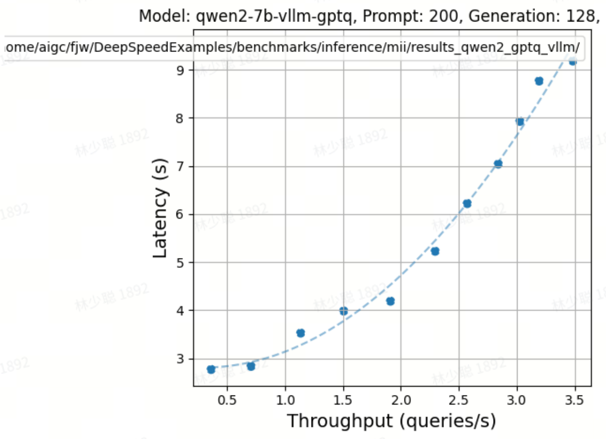
a. 输入200、输出60

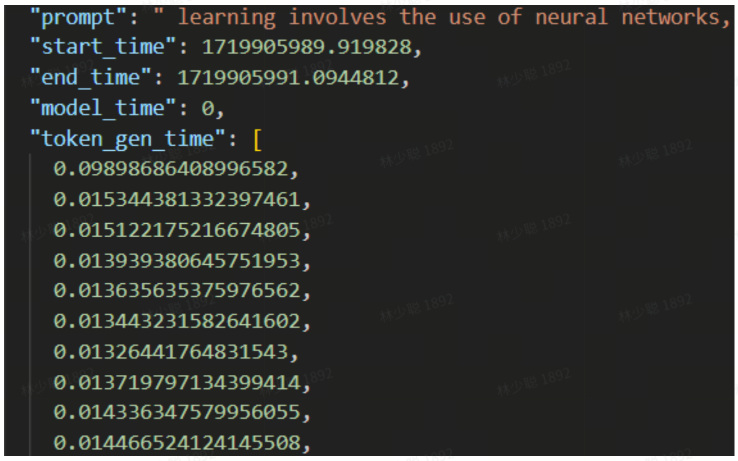




```
"prompt": " learning involves the use of neural networks,
"start_time": 1719905420.8701282,
"end_time": 1719905421.3665707,
"model_time": 0,
"token_gen_time": [
  0.04570460319519043,
  0.013432741165161133,
  0.01338338851928711,
  0.013240575790405273,
  0.013097524642944336,
  0.012918472290039062,
  0.01292037963867l875,
  0.012822866439819336,
  0.012774944305419922,
  0.012916326522827148,
```

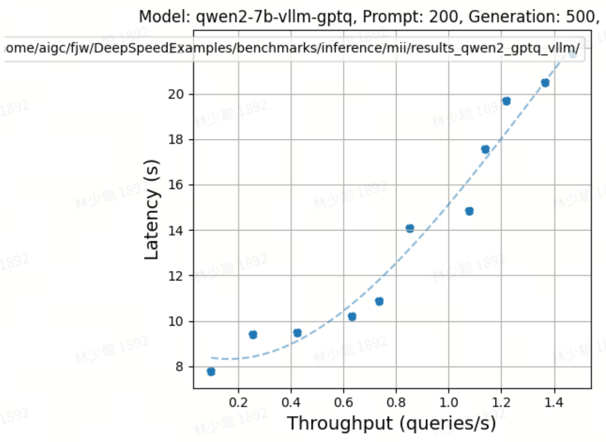
b. 输入200、输出128

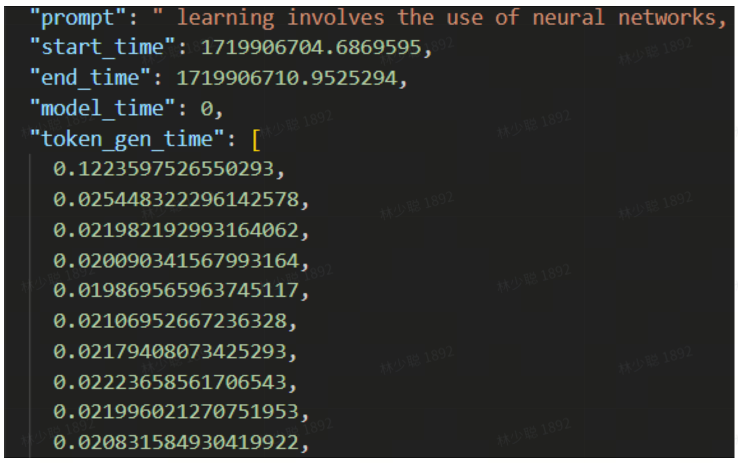




```
"prompt": " learning involves the use of neural networks,
"start_time": 1719905989.919828,
"end_time": 1719905991.0944812,
"model_time": 0,
"token_gen_time": [
  0.09898686408996582,
  0.015344381332397461,
  0.015122175216674805,
  0.013939380645751953,
  0.013635635375976562,
  0.013443231582641602,
  0.013264417648315430,
  0.013719797134399414,
  0.014336347579956055,
  0.014466524124145508,
```

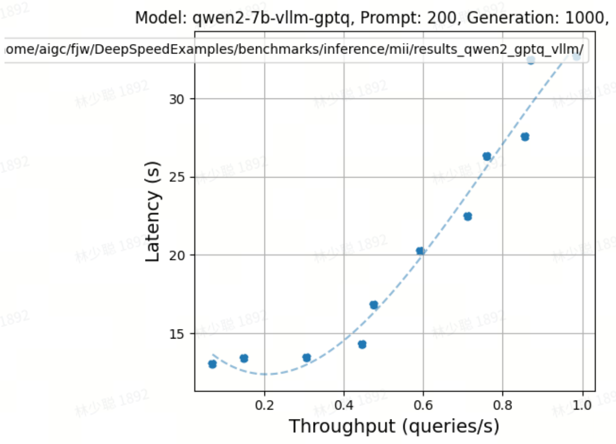
c. 输入200、输出500

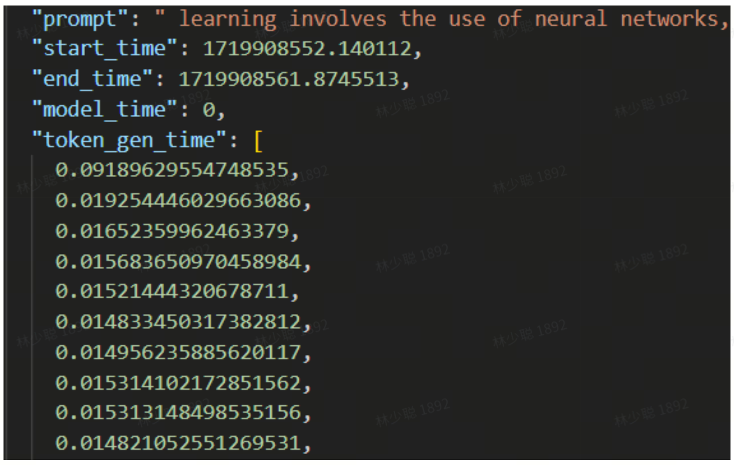




```
"prompt": " learning involves the use of neural networks,
"start_time": 1719906704.6869595,
"end_time": 1719906710.9525294,
"model_time": 0,
"token_gen_time": [
  0.1223597526550293,
  0.025448322296142578,
  0.021982192993164062,
  0.020090341567993164,
  0.019869565963745117,
  0.021069526672363280,
  0.021794080734252930,
  0.022236585617065430,
  0.021996021270751953,
  0.020831584930419922,
```

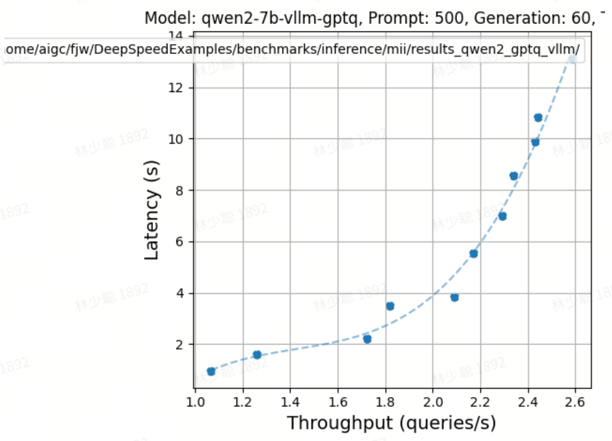
d. 输入200、输出1000

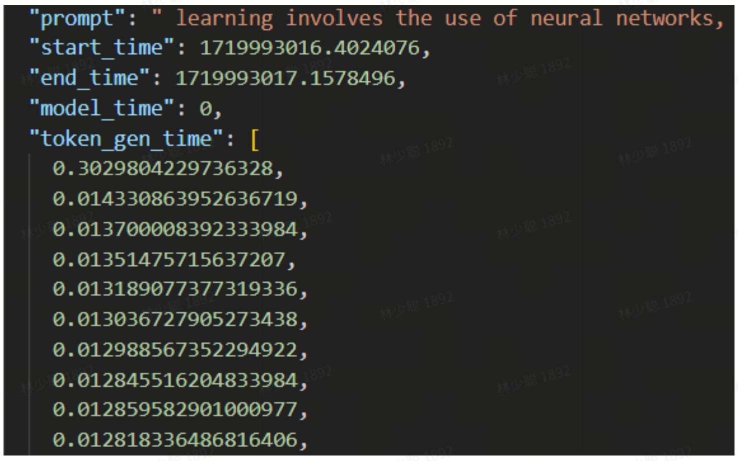




```
"prompt": " learning involves the use of neural networks,
"start_time": 1719908552.140112,
"end_time": 1719908561.8745513,
"model_time": 0,
"token_gen_time": [
  0.09189629554748535,
  0.019254446029663086,
  0.01652359962463379,
  0.015683650970458984,
  0.01521444320678711,
  0.014833450317382812,
  0.014956235885620117,
  0.015314102172851562,
  0.015313148498535156,
  0.014821052551269531,
```

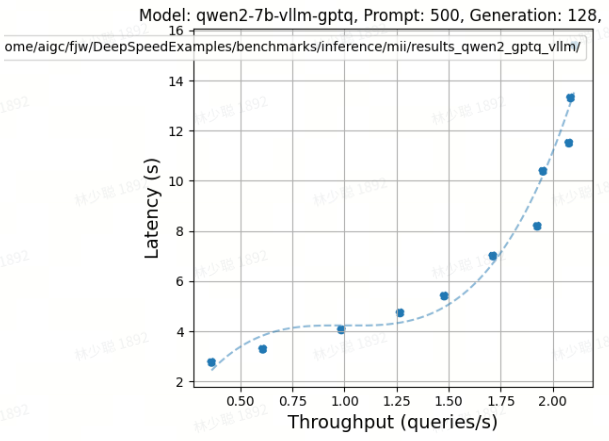
e. 输入500、输出60

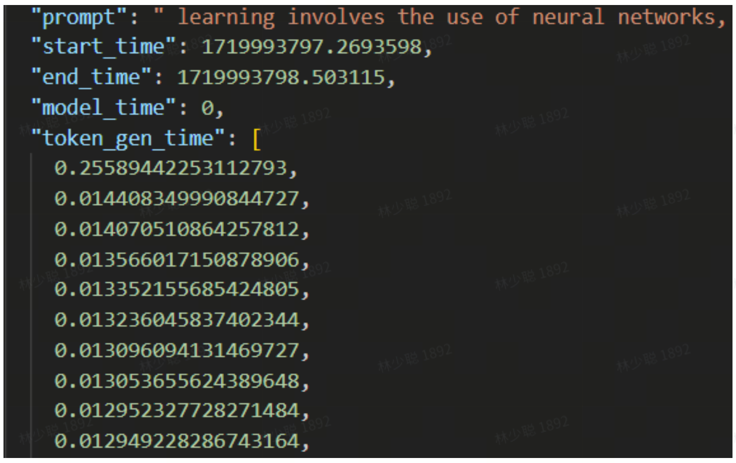




```
"prompt": " learning involves the use of neural networks,
"start_time": 1719993016.4024076,
"end_time": 1719993017.1578496,
"model_time": 0,
"token_gen_time": [
  0.3029804229736328,
  0.014330863952636719,
  0.013700008392333984,
  0.01351475715637207,
  0.013189077377319336,
  0.013036727905273438,
  0.012988567352294922,
  0.012845516204833984,
  0.01285958290100977,
  0.012818336486816406,
```

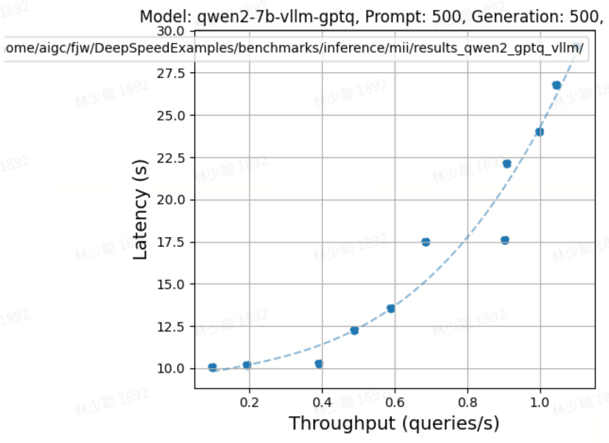
f. 输入500、输出128

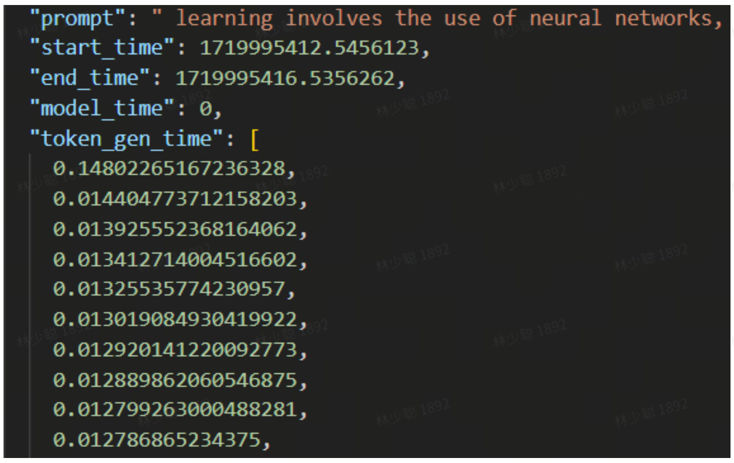




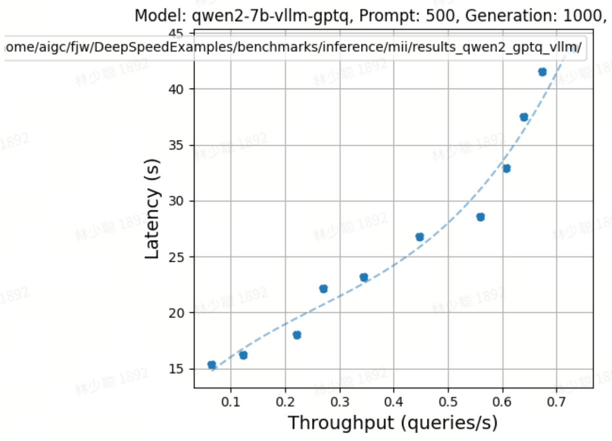
```
"prompt": " learning involves the use of neural networks,
"start_time": 1719993797.2693598,
"end_time": 1719993798.503115,
"model_time": 0,
"token_gen_time": [
  0.2558944225311279З,
  0.014408349990844727,
  0.014070510864257812,
  0.013566017150878906,
  0.013352155685424805,
  0.013236045837402344,
  0.013096094131469727,
  0.013053655624389648,
  0.012952327728271484,
  0.012949228286743164,
```

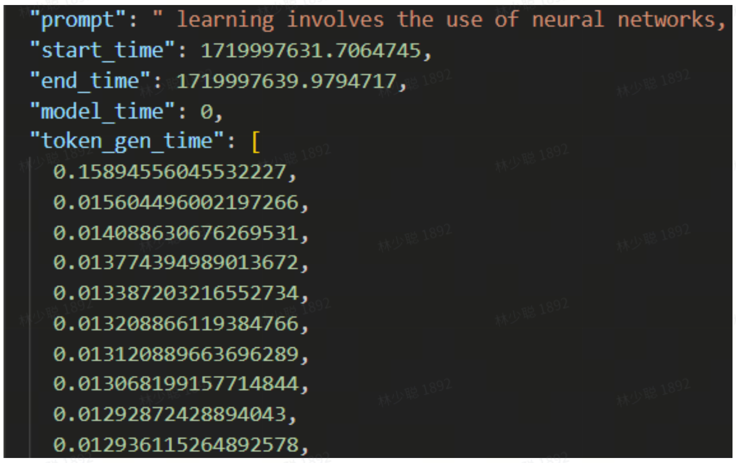
g. 输入500、输出500



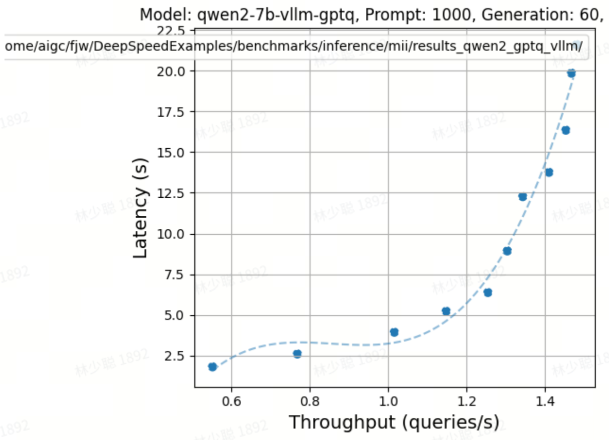

```
"prompt": " learning involves the use of neural networks,
"start_time": 1719995412.5456123,
"end_time": 1719995416.5356262,
"model_time": 0,
"token_gen_time": [
  0.14802265167236328,
  0.014404773712158203,
  0.013925552368164062,
  0.013412714004516602,
  0.01325535774230957,
  0.013019084930419922,
  0.012920141220092773,
  0.01288986206054687 5,
  0.012799263000488281,
  0.01278686523437 5,
```

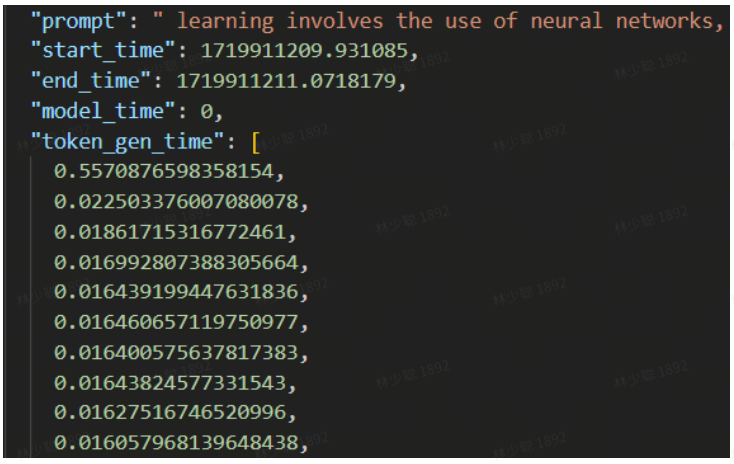
h. 输入500、输出1000



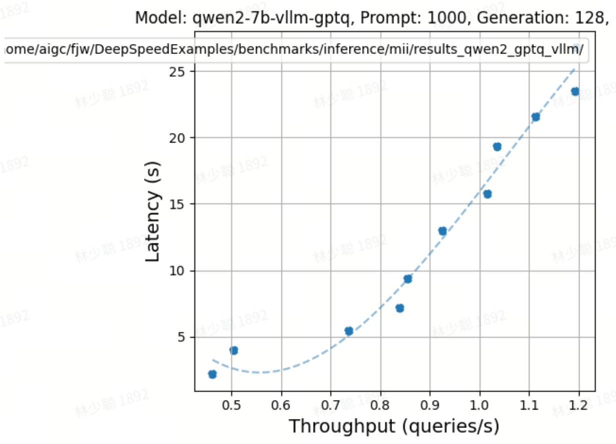

```
"prompt": " learning involves the use of neural networks,
"start_time": 1719997631.7064745,
"end_time": 1719997639.9794717,
"model_time": 0,
"token_gen_time": [
  0.15894556045532227,
  0.015604496002197266,
  0.014088630676269531,
  0.013774394989013672,
  0.013387203216552734,
  0.013208866119384766,
  0.013120889663696289,
  0.013068199157714844,
  0.012928724288940 43,
  0.012936115264892578,
```

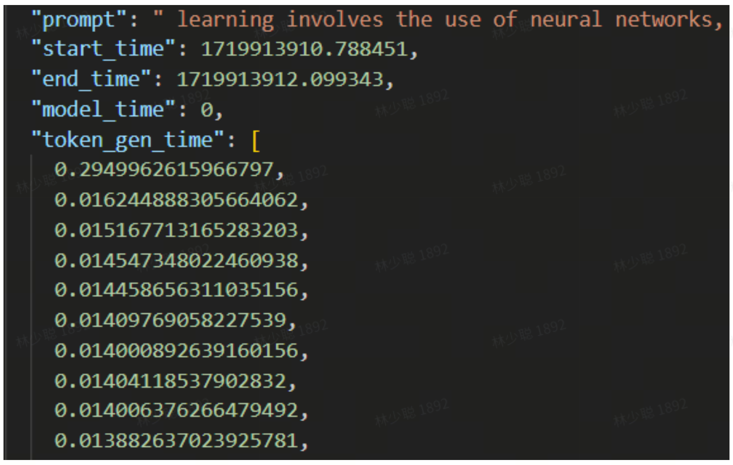
i. 输入1000、输出60




```
"prompt": " learning involves the use of neural networks,
"start_time": 1719911209.931085,
"end_time": 1719911211.0718179,
"model_time": 0,
"token_gen_time": [
  0.5570876598358154,
  0.022503376007080078,
  0.018617153167724 61,
  0.016992807388305664,
  0.016439199447631836,
  0.016460657119750977,
  0.016400575637817383,
  0.016438245773315 43,
  0.016275167465209 96,
  0.016057968139648438,
```

j. 输入1000、输出128



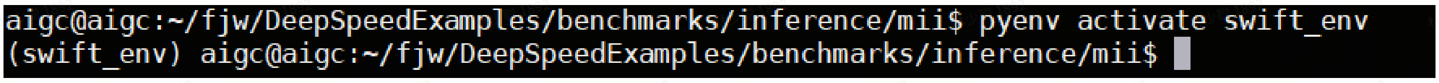


```
"prompt": " learning involves the use of neural networks,
"start_time": 1719913910.788451,
"end_time": 1719913912.099343,
"model_time": 0,
"token_gen_time": [
  0.2949962615966797,
  0.016244888305664062,
  0.015167713165283203,
  0.014547348022460938,
  0.014458656311035156,
  0.014097690582275390,
  0.014000892639160156,
  0.014041185379028320,
  0.014006376266479492,
  0.013882637023925781,
```

5.5、Qwen2-7B-Instruct 原始模型性能测试（deepspeed脚本）

1. 激活swift_env pyenv环境，带VLLM推理后端

```
pyenv activate swift_env
```



```
aigc@aigc:~/fjw/DeepSpeedExamples/benchmarks/inference/mii$ pyenv activate swift_env
(swift_env) aigc@aigc:~/fjw/DeepSpeedExamples/benchmarks/inference/mii$ 
```

2. 性能测试脚本

```
# Copyright (c) Microsoft Corporation.
# SPDX-License-Identifier: Apache-2.0

# DeepSpeed Team

# Run benchmark
### set arch for A10
export TORCH_CUDA_ARCH_LIST="8.6"
python ./run_benchmark.py \
        --model /home/aigc/lsc/vllm_test/Qwen2-7B-Instruct \
        --tp_size 1 \
        --num_replicas 1 \
        --max_ragged_batch_size 768 \
        --mean_prompt_length 200 \
        --mean_max_new_tokens 60 \
        --stream \
        --backend vllm \
        --out_json_dir results_qwen2 \

### Gernerate the plots
```

```
python ./src/plot_th_lat.py \
        --data_dirs
/home/aigc/fjw/DeepSpeedExamples/benchmarks/inference/mii/results_qwen2_vllm/ \
        --out_dir
/home/aigc/fjw/DeepSpeedExamples/benchmarks/inference/mii/plots_qwen2_vllm/ \
        --model_name qwen2-7b-vllm
```

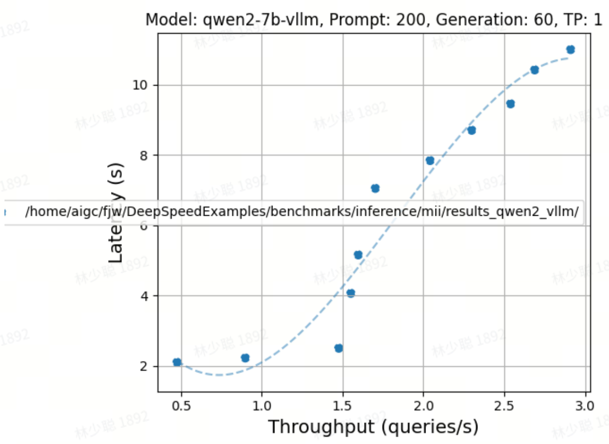
3. 执行性能测试

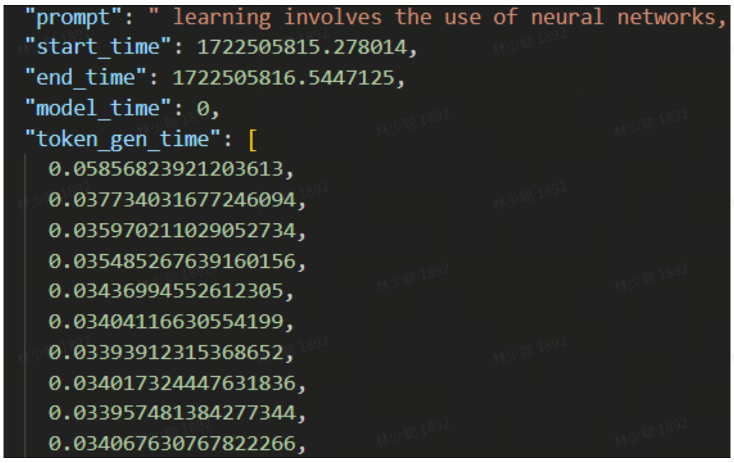
```
CUDA_VISIBLE_DEVICES=0 swift infer \
    --model_type qwen2-7b-instruct \
    --model_id_or_path /home/aigc/lsc/vllm_test/Qwen2-7B-Instruct \
    --infer_backend vllm \
    --max_model_len=16000
```

4. 性能测试结果

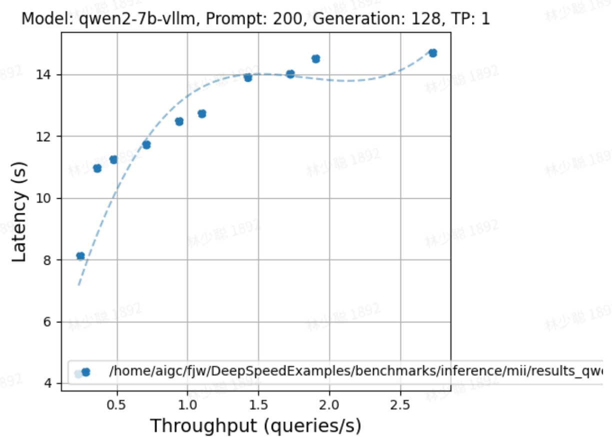
测试选择输入200、500、1200与输出60、128、500、1000组合（1200输入时不包含500、1000输出）

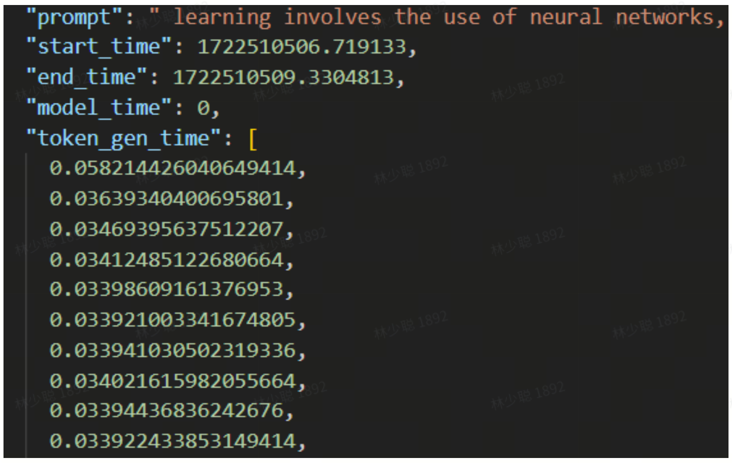
a. 输入200、输出60



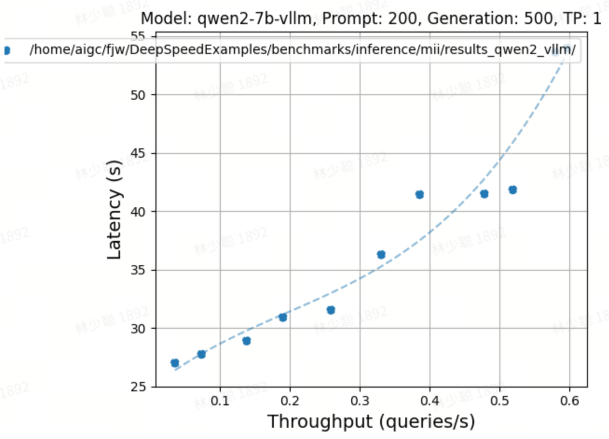

```
"prompt": " learning involves the use of neural networks,
"start_time": 1722505815.278014,
"end_time": 1722505816.5447125,
"model_time": 0,
"token_gen_time": [
  0.05856823921203613,
  0.037734031677246094,
  0.035970211029052734,
  0.035485267639160156,
  0.03436994552612305,
  0.03404116630554199,
  0.03393912315368652,
  0.034017324447631836,
  0.033957481384277344,
  0.034067630767822266,
```

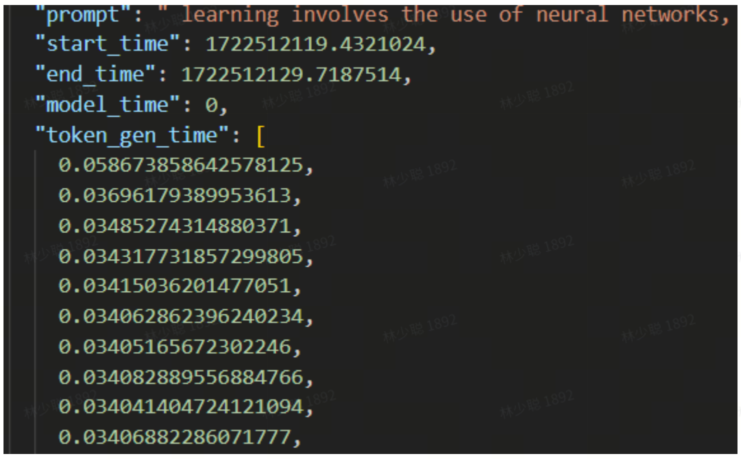
b. 输入200、输出128



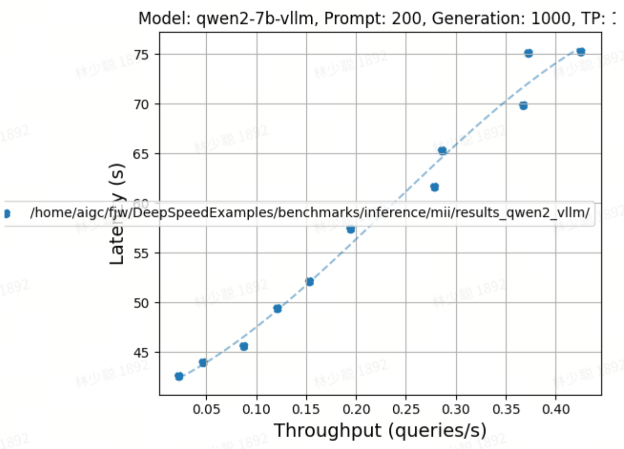

```
"prompt": " learning involves the use of neural networks,
"start_time": 1722510506.719133,
"end_time": 1722510509.3304813,
"model_time": 0,
"token_gen_time": [
  0.058214426040649414,
  0.03639340400695801,
  0.03469395637512207,
  0.03412485122680664,
  0.03398609161376953,
  0.033921003341674805,
  0.033941030502319336,
  0.034021615982055664,
  0.03394436836242676,
  0.033922433853149414,
```

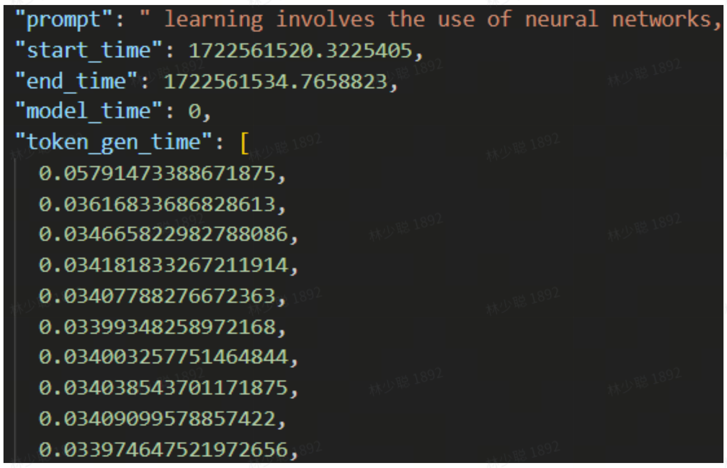
c. 输入200、输出500



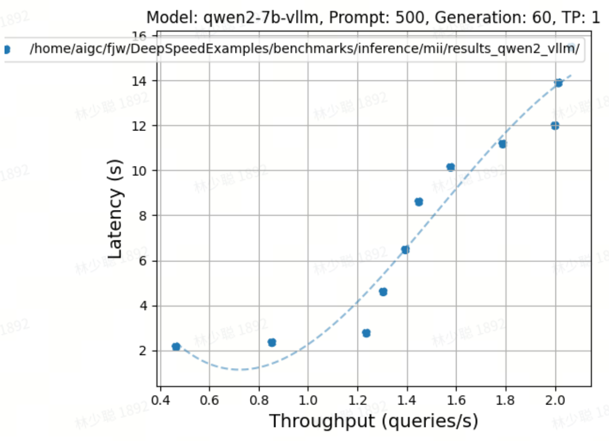

```
"prompt": " learning involves the use of neural networks,
"start_time": 1722512119.4321024,
"end_time": 1722512129.7187514,
"model_time": 0,
"token_gen_time": [
  0.058673858642578125,
  0.03696179389953613,
  0.03485274314880371,
  0.034317731857299805,
  0.03415036201477051,
  0.034062862396240234,
  0.03405165672302246,
  0.034082889556884766,
  0.034041404724121094,
  0.03406882286071777,
```

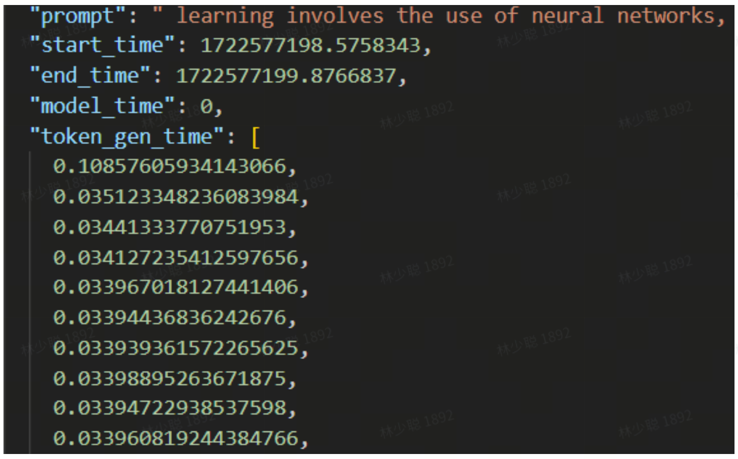
d. 输入200、输出1000



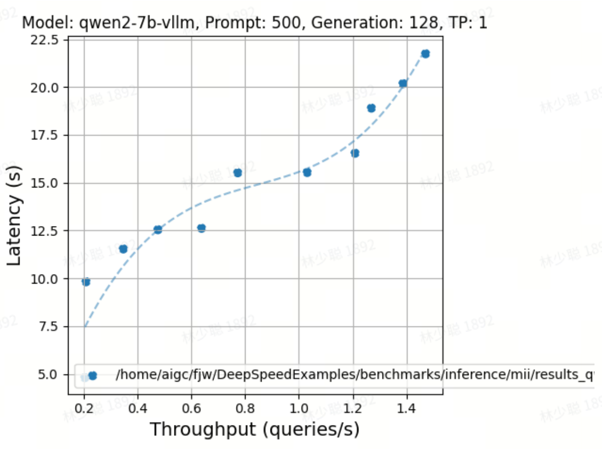

```
"prompt": " learning involves the use of neural networks,
"start_time": 1722561520.3225405,
"end_time": 1722561534.7658823,
"model_time": 0,
"token_gen_time": [
  0.05791473388671875,
  0.03616833686828613,
  0.034665822982788086,
  0.034181833267211914,
  0.03407788276672363,
  0.03399348258972168,
  0.034003257751464844,
  0.03403854370117l875,
  0.03409099578857422,
  0.033974647521972656,
```

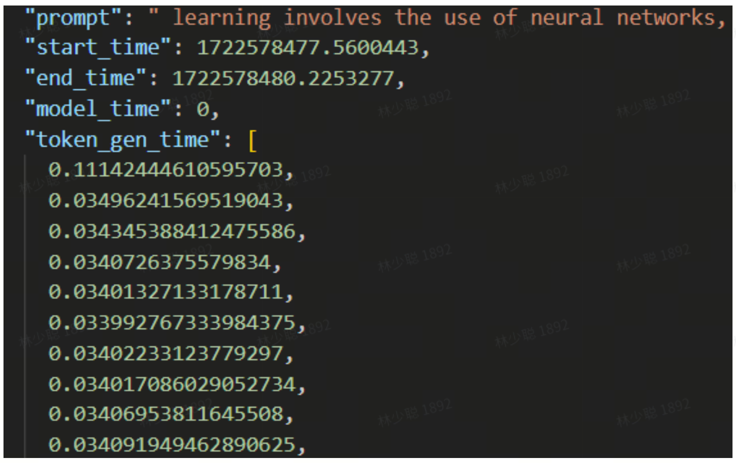
e. 输入500、输出60




```
"prompt": " learning involves the use of neural networks,
"start_time": 1722577198.5758343,
"end_time": 1722577199.8766837,
"model_time": 0,
"token_gen_time": [
  0.10857605934143066,
  0.035123348236083984,
  0.03441333770751953,
  0.034127235412597656,
  0.033967018127441406,
  0.03394436836242676,
  0.033939361572265625,
  0.03398895263671875,
  0.03394722938537598,
  0.033960819244384766,
```

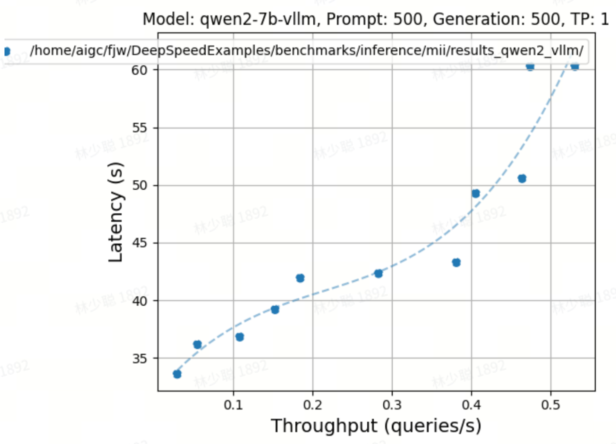
f. 输入500、输出128

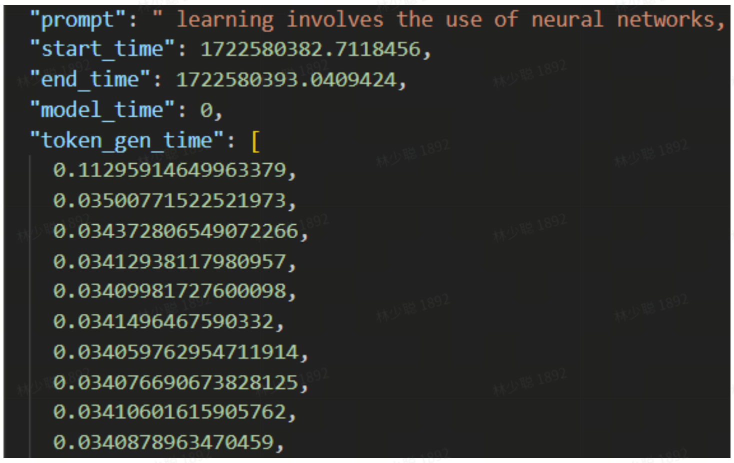




```
"prompt": " learning involves the use of neural networks,
"start_time": 1722578477.5600443,
"end_time": 1722578480.2253277,
"model_time": 0,
"token_gen_time": [
  0.1114244461059570З,
  0.03496241569519043,
  0.034345388412475586,
  0.0340726375579834,
  0.03401327133178711,
  0.033992767333984375,
  0.03402233123779297,
  0.034017086029052734,
  0.034069538116455078,
  0.034091949462890625,
```

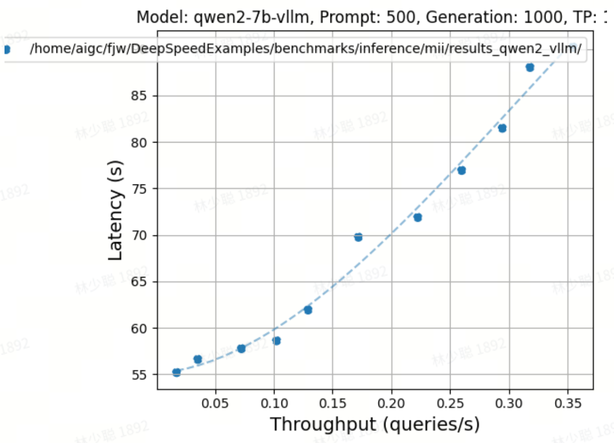
g. 输入500、输出500

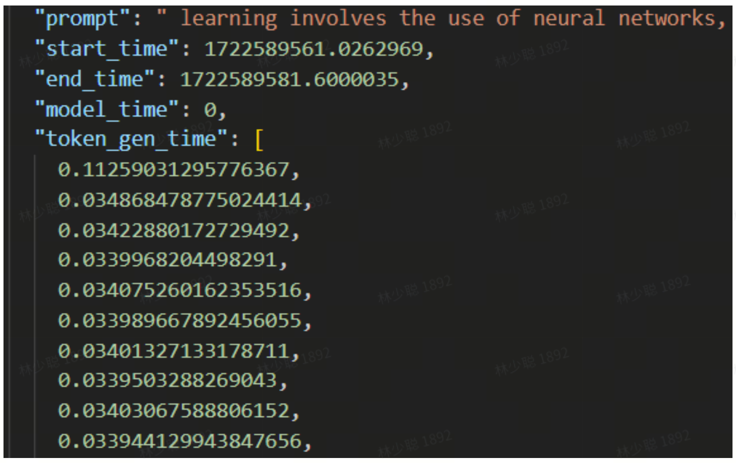




```
"prompt": " learning involves the use of neural networks,
"start_time": 1722580382.7118456,
"end_time": 1722580393.0409424,
"model_time": 0,
"token_gen_time": [
  0.11295914649963379,
  0.03500771522521973,
  0.034372806549072266,
  0.03412938117980957,
  0.03409981727600098,
  0.0341496467590332,
  0.034059762954711914,
  0.034076690673828125,
  0.03410601615905762,
  0.0340878963470459,
```

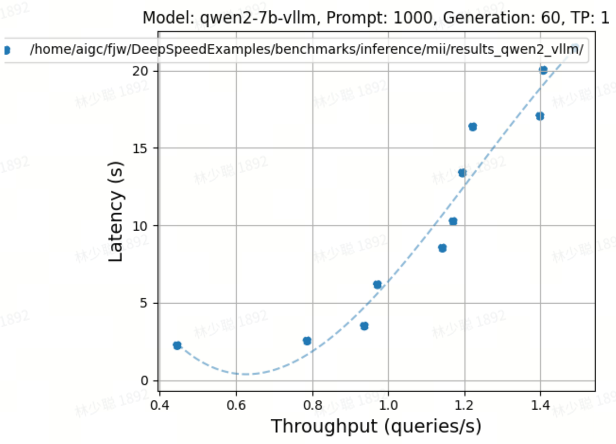
h. 输入500、输出1000

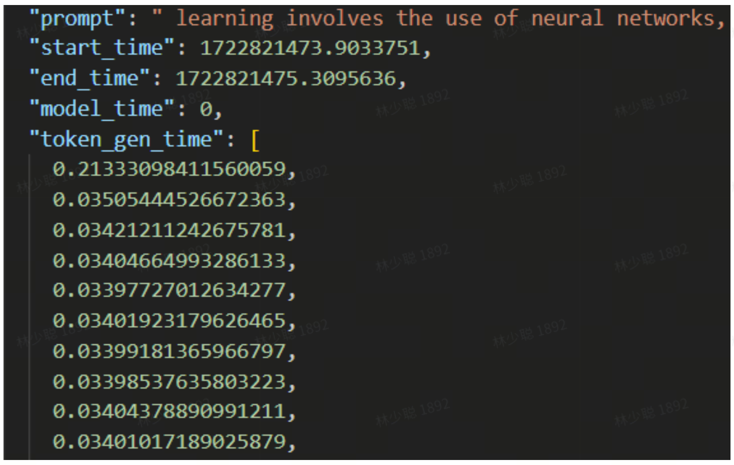




```
"prompt": " learning involves the use of neural networks,
"start_time": 1722589561.0262969,
"end_time": 1722589581.6000035,
"model_time": 0,
"token_gen_time": [
  0.11259031295776367,
  0.034868478775024414,
  0.03422880172729492,
  0.0339968204498291,
  0.034075260162353516,
  0.033989667892456055,
  0.03401327133178711,
  0.0339503288269043,
  0.03403067588806152,
  0.033944129943847656,
```

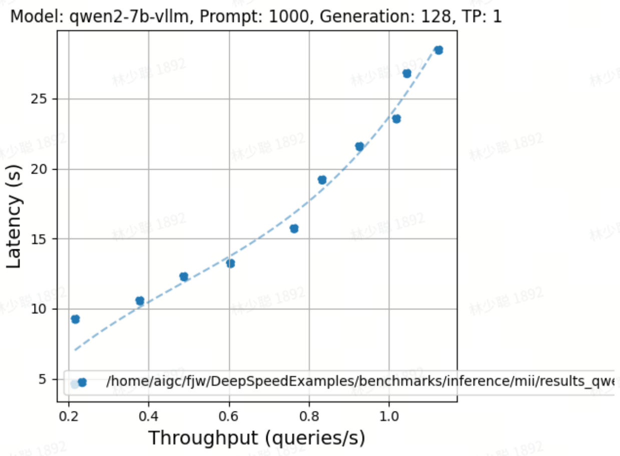
i. 输入1000、输出60

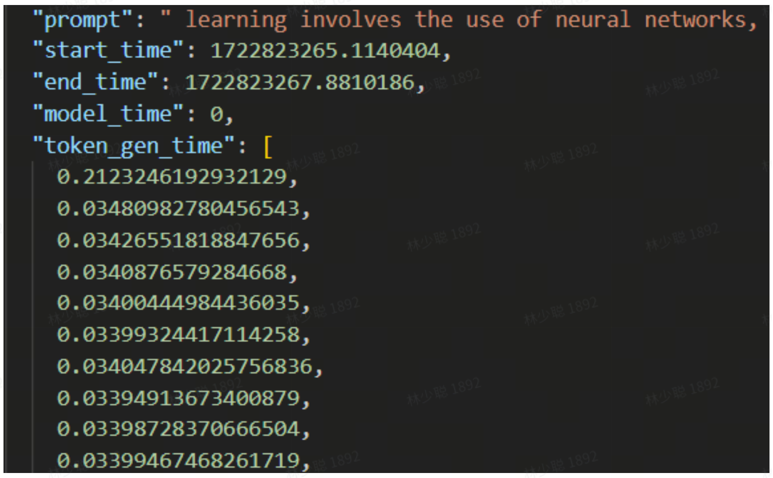




```
"prompt": " learning involves the use of neural networks,
"start_time": 1722821473.9033751,
"end_time": 1722821475.3095636,
"model_time": 0,
"token_gen_time": [
  0.21333098411560059,
  0.03505444526672363,
  0.03421211242675781,
  0.03404664993286133,
  0.03397727012634277,
  0.03401923179626465,
  0.03399181365966797,
  0.03398537635803223,
  0.03404378890991211,
  0.03401017189025879,
```

j. 输入1000、输出128

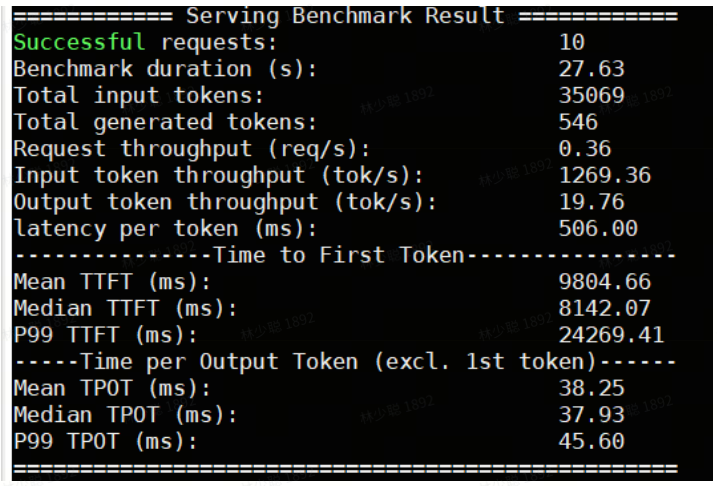




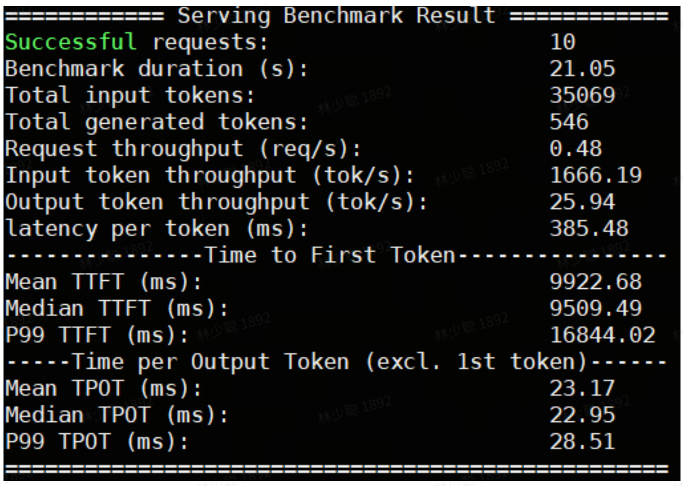
```
"prompt": " learning involves the use of neural networks,
"start_time": 1722823265.1140404,
"end_time": 1722823267.8810186,
"model_time": 0,
"token_gen_time": [
  0.2123246192932129,
  0.03480982780456543,
  0.03426551818847656,
  0.0340876579284668,
  0.03400444984436035,
  0.03399324417114258,
  0.034047842025756836,
  0.03394913673400879,
  0.03398728370666504,
  0.03399467468261719,
```

5.5、Qwen2-7B-Instruct 原始模型性能测试（vllm脚本）

1. TP=1　　　　　　　　　　　　TP=2

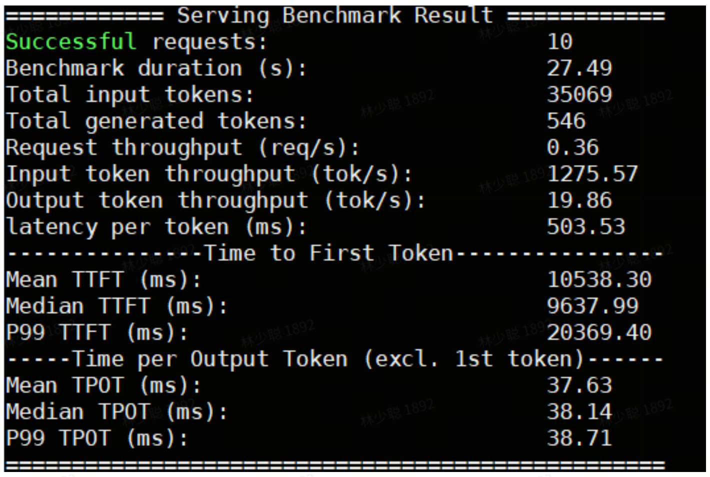


```
============ Serving Benchmark Result ============
Successful requests:                     10
Benchmark duration (s):                  27.63
Total input tokens:                      35069
Total generated tokens:                  546
Request throughput (req/s):              0.36
Input token throughput (tok/s):          1269.36
Output token throughput (tok/s):         19.76
latency per token (ms):                  506.00
---------------Time to First Token----------------
Mean TTFT (ms):                          9804.66
Median TTFT (ms):                        8142.07
P99 TTFT (ms):                           24269.41
-----Time per Output Token (excl. 1st token)------
Mean TPOT (ms):                          38.25
Median TPOT (ms):                        37.93
P99 TPOT (ms):                           45.60
==================================================
```

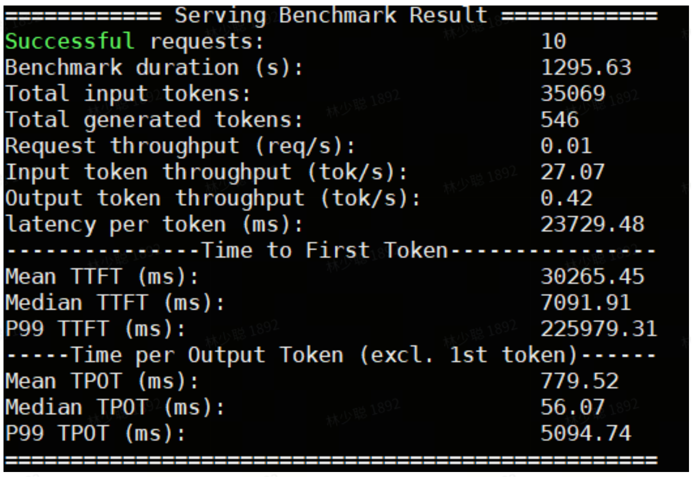


```
============ Serving Benchmark Result ============
Successful requests:                     10
Benchmark duration (s):                  21.05
Total input tokens:                      35069
Total generated tokens:                  546
Request throughput (req/s):              0.48
Input token throughput (tok/s):          1666.19
Output token throughput (tok/s):         25.94
latency per token (ms):                  385.48
---------------Time to First Token----------------
Mean TTFT (ms):                          9922.68
Median TTFT (ms):                        9509.49
P99 TTFT (ms):                           16844.02
-----Time per Output Token (excl. 1st token)------
Mean TPOT (ms):                          23.17
Median TPOT (ms):                        22.95
P99 TPOT (ms):                           28.51
==================================================
```

2. pp=1　　　　　　　　　　　　pp=2



```
============ Serving Benchmark Result ============
Successful requests:                     10
Benchmark duration (s):                  27.49
Total input tokens:                      35069
Total generated tokens:                  546
Request throughput (req/s):              0.36
Input token throughput (tok/s):          1275.57
Output token throughput (tok/s):         19.86
latency per token (ms):                  503.53
---------------Time to First Token----------------
Mean TTFT (ms):                          10538.30
Median TTFT (ms):                        9637.99
P99 TTFT (ms):                           20369.40
-----Time per Output Token (excl. 1st token)------
Mean TPOT (ms):                          37.63
Median TPOT (ms):                        38.14
P99 TPOT (ms):                           38.71
==================================================
```



```
============ Serving Benchmark Result ============
Successful requests:                     10
Benchmark duration (s):                  1295.63
Total input tokens:                      35069
Total generated tokens:                  546
Request throughput (req/s):              0.01
Input token throughput (tok/s):          27.07
Output token throughput (tok/s):         0.42
latency per token (ms):                  23729.48
---------------Time to First Token----------------
Mean TTFT (ms):                          30265.45
Median TTFT (ms):                        7091.91
P99 TTFT (ms):                           225979.31
-----Time per Output Token (excl. 1st token)------
Mean TPOT (ms):                          779.52
Median TPOT (ms):                        56.07
P99 TPOT (ms):                           5094.74
==================================================
```