【LLM推理加速】：VLLM最全参数详解

原创 方方 方方的算法花园 2024年10月28日 17:02 北京 标题已修改

点击蓝字 关注我们

本文详细解读了 VLLM 的参数，分析了 LLM()和SamplingParams() 的各项参数意义，同时文末还介绍了 VLLM 常用的调参方法，欢迎阅读～

VLLM的推理代码

「READING」

基于Qwen2.5的VLLM离线推理代码示例如下：

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

# Initialize the tokenizer
tokenizer = AutoTokenizer.from_pretrained("Qwen/Qwen2.5-7B-Instruct")

# Pass the default decoding hyperparameters of Qwen2.5-7B-Instruct
# max_tokens is for the maximum length for generation.
sampling_params = SamplingParams(temperature=0.7, top_p=0.8, repetition_penal

# Input the model name or path. Can be GPTQ or AWQ models.
llm = LLM(model="Qwen/Qwen2.5-7B-Instruct")

# Prepare your prompts
prompt = "Tell me something about large language models."
messages = [
    {"role": "system", "content": "You are Qwen, created by Alibaba Cloud. Yo
    {"role": "user", "content": prompt}
]
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}")
```

在上述代码中，可以发现有两个位置可用于配置参数，分别是 **LLM（）** 和 **SamplingParams（）**，下面将参考源码对这两个函数所涉及的参数逐一进行解读。

LLM（）参数解读

「READING」

参考vllm/entrypoints/llm.py代码，class LLM具有如下几种参数：（标红为常用参数）

1. ***<model>***: HuggingFace Transformers 模型的名称或路径，也可以是本地路径，比如"../Qwen2.5-7B-Instruct"。
2. ***<tokenizer>:*** HuggingFace Transformers 分词器的名称或路径，如果未指定，将使用model名称或路径。
3. ***<tokenizer_mode>***: 分词器模式。可选值：auto, slow, mistral。“auto”将在可用时使用快速分词器，而“slow”将始终使用慢速分词器,“mistral”将始终使用mistral_common标记器。默认值是 “auto”。4. ***<skip_tokenizer_init>:***如果为*True*，则跳过分词器和解分词器的初始化。期望从输入中得到有效的prompt token ID，且prompt为 None。
5. ***<trust_remote_code>:*** 在下载模型和分词器时信任来自远程的代码（例如来自HuggingFace 的代码）。
6. ***<tensor_parallel_size>:*** 用于张量并行分布式执行的 GPU 数量，默认为1.
7. ***<dtype>:*** 模型权重和激活的数据类型。取值范围：auto, half, float16, bfloat16, float, float32。“auto”将FP32和FP16模型使用FP16精度，BF16模型使用BF16精度。“half”用于FP16，推荐用于AWQ量化。“float16”和“half”一样。“bfloat16”用于精度和范围之间的平衡。“float”是FP32精度的简写。“float32”是FP32精度。如果是“auto”，我们将使用在模型配置文件中指定的“torch_dtype”属性。但是，如果配置中的“torch_dtype”是“float32”，我们将改为使用“float16”。默认为“auto”。
8. ***<quantization>:*** 用于量化模型权重的方法。取值范围：aqlm, awq, deepspeedfp, tpu_int8, fp8, fbgemm_fp8, modelopt, marlin, gguf, gptq_marlin_24, gptq_marlin, awq_marlin, gptq, compressed-tensors, bitsandbytes, qqq, experts_int8, neuron_quant, ipex, None。如果没有，我们首先检查模型配置文件中的quantization_config属性。quantization_config为None，我们假设模型权重没有量化，并使用dtype来确定权重的数据类型。
9. ***<revision>:*** 要使用的特定模型版本。它可以是分支名称、标签名称或commit ID。如果未指定，将使用默认版本。
10. ***<tokenizer_revision>:*** 要使用的特定分词器版本。它可以是分支名称、标签名称或commit id。如果未指定，将使用默认版本。
11. ***<seed>:*** 用于采样的随机数生成器的初始化种子，默认为0。
12. ***<gpu_memory_utilization>:*** 用于为模型权重、激活和键值缓存保留的 GPU 内存比例，范围从0到1。例如，0.5的值意味着50%的GPU内存利用率。较高的值将增加键值缓存的大小，从而提高模型的吞吐量。然而，如果该值过高，可能会导致内存不足（OOM）错误。如果未指定，将使用默认值0.9。
13. ***<swap_space>:*** 每个 GPU 使用的作为交换空间的 CPU 内存大小（单位为 GiB）。当请求的“best_of”采样参数大于 1 时，可用于临时存储请求的状态。如果所有请求的“best_of”都为 1，则可以安全地将此值设置为 0。否则，过小的值可能会导致内存不足（OOM）错误。默认值为4。
14. ***<cpu_offload_gb>:*** 每块 GPU 卸载到 CPU 的空间（单位为 GiB）。默认值为 0，表示不进行卸载。直观地说，这个参数可以看作是一种虚拟地增加 GPU 内存大小的方式。例

如，如果你有一块 24GB 的 GPU，并将其设置为 10，那么从虚拟的角度来看，你可以将其视为一块 34GB 的 GPU。然后，你可以加载一个使用 BF16 权重的 13B 模型，该模型至少需要 26GB 的 GPU 内存。请注意，这需要快速的 CPU-GPU 互连，因为在每次模型前向传递过程中，模型的一部分会实时地从 CPU 内存加载到 GPU 内存中。默认值：0。

15. ***<enforce_eager>:*** 总是使用eager模式的 PyTorch。如果设置为 “False”，则会在混合模式下使用eager模式和 CUDA 图，以获得最大性能和灵活性。

16. ***<max_context_len_to_capture>:*** CUDA 图覆盖的最大上下文长度。当序列的上下文长度大于此值时，我们回退到eager模式（已弃用。请改用`max_seq_len_to_capture`）。

17. ***<max_seq_len_to_capture>:*** CUDA 图覆盖的最大序列长度。当序列的上下文长度大于此值时，我们回退到eager模式。此外，对于编码器-解码器模型，如果编码器输入的序列长度大于此值，我们也回退到eager模式。默认值：8192。

18. ***<disable_custom_all_reduce>:*** 查看 ParallelConfig

19. ***<**kwargs>:*** 类`vllm.EngineArgs`的参数。（请参阅：`engine_args`的引用）。

<**kwargs> 「READING」

针对*<kwargs>*，在vllm/engine/llm_engine.py文件又进行了介绍，参数具体配置情况又可以从vllm/config.py文件中查看，具体如下：**

<model_config>: 与大语言模型相关的配置。

参数名称	解释
model	与class LLM一致
task	使用模型执行的任务。每个 vLLM 实例仅支持一个任务，即使同一模型可用于多个任务。当模型仅支持一个任务时，可以使用“auto”进行选择；否则，必须明确指定要使用的任务。
tokenizer	与class LLM一致
tokenizer_mode	与class LLM一致
trust_remote_code	与class LLM一致
dtype	与class LLM一致
seed	与class LLM一致
revision	与class LLM一致
code_revision	用于 Hugging Face Hub 上模型代码的特定修订版本。它可以是分支名称、标签名称或提交 ID。如果未指定，将使用默认版本。
rope_scaling	包含 RoPE 嵌入的缩放配置的字典。当使用此标志时，不要将“max_position_embeddings”更新为预期的新最大值。
tokenizer_revision	与class LLM一致
max_model_len	序列的最大长度（包括提示和输出）。如果为 None，则从模型中推导得出。
quantization	与class LLM一致
quantization_param_path	包含比例因子的 JSON 文件路径。用于在 ROCm（AMD GPU）上的键值缓存类型为 FP8_E4M3 时将键值缓存比例因子加载到模型中。未来，当在 ROCm 上的

	模型数据类型为 FP8_E4M3 时，这些也将用于加载激活和权重比例因子。
enforce_eager	与class LLM一致
max_context_len_to_capture	与class LLM一致
max_seq_len_to_capture	与class LLM一致
disable_sliding_window	是否禁用滑动窗口。如果为真，我们将禁用模型的滑动窗口功能。如果模型不支持滑动窗口，则此参数将被忽略。
skip_tokenizer_init	与class LLM一致
served_model_name	指标标签“model_name”中使用的模型名称与通过 API 暴露的模型名称相匹配。如果提供了多个模型名称，则将使用第一个名称。如果未指定，则模型名称将与“model”相同。
limit_mm_per_prompt	每个提示中每种模态的数据实例的最大数量。仅适用于多模态模型。
config_format	应加载的配置格式。默认为“auto”，其默认值为“hf”。
override_neuron_config	初始化非默认的神经元配置或覆盖特定于神经元设备的默认神经元配置，这个参数将用于配置无法从 vllm 参数中收集到的神经元配置。
mm_processor_kwargs	初始化非默认的神经元配置或覆盖特定于神经元设备的默认神经元配置，这个参数将用于配置无法从 vllm 参数中收集到的神经元配置。

<cache_config>: 与键值缓存内存管理相关的配置。

参数名称	解释
block_size	以token数量表示的缓存块大小。
gpu_memory_utilization	与class LLM一致
swap_space	与class LLM一致
cache_dtype	用于键值缓存存储的数据类型。
num_gpu_blocks_override	要使用的 GPU 块数量。如果指定了此参数，则会覆盖分析得到的 num_gpu_blocks。如果为 None，则不执行任何操作。

<parallel_config>: 与分布式执行相关的配置。

参数名称	解释
pipeline_parallel_size	pipeline并行组数。
tensor_parallel_size	与class LLM一致
worker_use_ray	已弃用，请使用 distributed_executor_backend 代替。
max_parallel_loading_workers	当顺序加载模型时的最大并行加载批次数量。这是为了在使用张量并行和大型模型时避免内存溢出。
disable_custom_all_reduce	禁用自定义的all-reduce内核，并回退到 NCCL。

tokenizer_pool_config	分词器池的配置。如果为 None，则使用同步分词。
ray_workers_use_nsight	是否使用 nsight 分析 Ray 工作进程，详情请见 https://docs.ray.io/en/latest/ray-observability/user-guides/profiling.html#profiling-nsight-profiler。
placement_group	ray分布式模型工作节点放置组。
distributed_executor_backend	分布式模型工作节点使用的后端，可以是“ray”或“mp”（多进程）。如果流水线并行大小（pipeline_parallel_size）或张量并行大小（tensor_parallel_size）大于 1，且安装了 Ray，则默认为“ray”，否则为“mp”。

<scheduler_config>: 与请求调度器相关的配置。

参数名称	解释
task	使用模型执行的任务。
max_num_batched_tokens	在单个迭代中处理的最大token数量。默认值是2048。max_num_batched_tokens越大，能处理的tokens数量也就越大，但vllm内部会根据max_model_len自动计算max_num_batched_tokens，所以可以不设置这个值。
max_num_seqs	在单次迭代中要处理的最大序列数量。默认值是256。max_num_seqs越大，能处理的请求数量就会越大，但提升也会有上限，不一定是越大越好： 2卡时，max_num_seqs设置为1024，相较于256，速度提升19%。 4卡时，max_num_seqs设置为2048，相较于256，速度提升35%；max_num_seqs设置为4096，相较于256，速度提升33%。
max_model_len	序列的最大长度（包括提示和生成的文本）。
num_lookahead_slots	每步每个序列分配的前瞻槽位数，超出已知的token ID。这在推测解码中用于存储可能被接受也可能不被接受的token的键值激活。
delay_factor	在调度下一个prompt之前应用一个延迟（延迟因子乘以先前prompt的延迟时间）。
enable_chunked_prefill	如果为True，则预填充请求可以根据剩余的最大批处理token数进行分块。
preemption_mode	预占模式是通过交换还是重新计算来执行。如果未指定，我们将按如下方式确定模式：默认情况下我们使用重新计算，因为它比交换产生的开销更低。然而，当序列组有多个序列时（例如，束搜索），目前不支持重新计算。在这种情况下，我们使用交换代替。
send_delta_data	调度器配置：私有 API。如果使用，调度器将向工作进程发送增量数据而非完整数据。仅当启用了 SPMD 工作进程架构时才应启用该功能。即，设置 VLLM_USE_RAY_SPMD_WORKER=1。

policy	要使用的调度策略。“先到先服务”（默认）或“优先级”。

<device_config>: 与设备相关的配置。

参数名称	解释
device	以token数量表示的缓存块大小。

<lora_config (Optional)>: 与服务multi-LoRA相关的配置。

参数名称	解释
max_lora_rank	最大的 LoRA 秩（rank），int
max_loras	最大的 LoRA 数量，int
fully_sharded_loras	是否为完全分片的 LoRA，默认为 False
max_cpu_loras	可选的最大 CPU 上的 LoRA 数量，默认None
lora_dtype	可选的 LoRA 数据类型（可以是 torch.dtype 或字符串表示的类型）。默认None
lora_extra_vocab_size	LoRA 额外的词汇大小，默认256
lora_vocab_padding_size	LoRA 词汇填充大小，这是一个类常量，默认256
long_lora_scaling_factors	可选的长 LoRA 缩放因子元组，默认None

<speculative_config(Optional)>: 与推测解码相关的配置。

参数名称	解释
target_model_config	目标模型的配置。
target_parallel_config	目标模型的并行配置。
target_dtype	目标模型使用的数据类型。
speculative_model	推测模型的名称（如果提供了的话）
speculative_model_quantization	用于量化推测模型权重的量化方法。如果为空，则假设模型权重未被量化。
speculative_draft_tensor_parallel_size	模型的张量并行度
num_speculative_tokens	推测tokens的数量（如果提供了的话）。如果未提供，将默认为草稿模型配置中的数量（如果存在），否则是必需的。
speculative_disable_mqa_scorer	是否为推测模型禁用 MQA 计分器，并回退到批量扩展进行计分。
speculative_max_model_len	推测模型的最大模型长度。用于测试跳过某些序列的推测能力。
enable_chunked_prefill	vLLM 是否配置为使用分块预填充。用于引发错误，因为它与推测解码尚不兼容。
speculative_disable_by_batch_size	当入队请求的数量大于此值时，对新传入的请求禁用推测解码。
ngram_prompt_lookup_max	ngram token的最大尺寸。
ngram_prompt_lookup_min	ngram token的最小尺寸。

draft_token_acceptance_method	用于接受draft token的方法。这可以取两个可能的值“rejection_sampler”和“typical_acceptance_sampler”，分别对应拒绝采样器（RejectionSampler）和典型接受采样器（TypicalAcceptanceSampler）。
typical_acceptance_sampler_posterior_threshold	一个阈值，为目标模型中的token后验概率设定一个下限，只有当使用典型接受采样器进行token接受时，该阈值才会被使用。
typical_acceptance_sampler_posterior_alpha	典型接受采样器中基于熵的阈值的缩放因子。
disable_logprobs	如果设置为 True，则在推测解码期间不返回token对数概率。如果设置为 False，则根据 SamplingParams 中的对数概率设置返回令牌对数概率。如果未指定，则默认为 True。

<prompt_adapter_config (Optional)>: 与服务prompt适配器相关的配置。

参数名称	解释
max_prompt_adapters	最大的prompt适配器数量
max_prompt_adapter_token	最大的prompt适配器token数量
max_cpu_prompt_adapters	可选的最大 CPU 上的prompt适配器数量。
prompt_adapter_dtype	prompt适配器的数据类型（可选的 torch.dtype

<executor_class>: 用于管理分布式执行的模型执行器类。
<log_stats>: 是否记录统计信息，Bool类型。
<usage_context>: 指定的entry point，用于收集使用信息（from vllm.usage.usage_lib import UsageContext）

SamplingParams（）参数解读

「READING」

参考vllm/sampling_params.py代码，class SamplingParams 具有如下几种参数：

（1）***<n>：*** 要为给定提示返回的输出序列数量。

（2）***<best_of>：*** 根据prompt生成的输出序列数量。从这些best_of序列中，返回前n个序列。best_of必须大于或等于n。默认情况下，best_of设置为n。

（3）***<presence_penalty>：*** 根据新token是否出现在迄今为止生成的文本中对其进行惩罚的浮点数。值大于 0 鼓励模型使用新token，而值小于 0 鼓励模型重复token。

（4）***<frequency_penalty>***：根据新token在迄今为止生成的文本中的出现频率对其进行惩罚的浮点数。值大于 0 鼓励模型使用新token，而值小于 0 鼓励模型重复token。

（5）***<repetition_penalty>***：根据新token是否出现在prompt和迄今为止生成的文本中对其进行惩罚的浮点数。值大于 1 鼓励模型使用新token，而值小于 1 鼓励模型重复token。

（6）***<temperature>：*** 控制采样随机性的浮点数。较低的值使模型更具确定性，而较高的值使模型更具随机性。零表示贪婪采样。

（7）***<top_p>：*** 控制要考虑的前token的累积概率的浮点数。必须在（0，1] 中。设置为 1 以考虑所有token。

（8）***<top_k>：*** 控制要考虑的前token数量的整数。设置为 - 1 以考虑所有token。

（9）***<min_p>：***表示相对于最可能token的概率，token被考虑的最小概率的浮点数。必须在 [0，1] 中。设置为 0 以禁用此功能。

（10）***<seed>：***用于生成的随机种子。

（11）***<stop>：***停止生成的字符串列表。返回的输出将不包含停止字符串。

（12）***<stop_token_ids>：***停止生成的token列表。返回的输出将包含停止token，除非停止token是特殊token。

（13）***<include_stop_str_in_output>***：是否在输出文本中包含停止字符串。默认值为否。

（14）***<ignore_eos>：***是否忽略 EOS token并在生成 EOS token后继续生成token。

（15）***<max_tokens>：***每个输出序列生成的最大token数。

（16）***<min_tokens>：***在可以生成 EOS 或停止token之前，每个输出序列生成的最小token数。

（17）***<logprobs>***：每个输出token返回的对数概率数。当设置为 None 时，不返回概率。如果设置为非 None 值，则结果包括指定数量的最可能token的对数概率，以及所选token。请注意，该实现遵循 OpenAI API：API 将始终返回采样token的对数概率，因此响应中可能最多有logprobs+1个元素。

（18）***<prompt_logprobs>：***每个提示token返回的对数概率数。

（19）***<detokenize>：***是否对输出进行反tokenizer。默认值为是True。

（20）***<skip_special_tokens>：***是否跳过输出中的特殊token。

（21）***<spaces_between_special_tokens>：***在输出中的特殊token之间是否添加空格。默认值为True。

（22）***<logits_processors>：***根据先前生成的token以及可选地根据提示token修改对数概率的函数列表。

（23）***<truncate_prompt_tokens>：***如果设置为整数 k，则将仅使用提示的最后 k 个token（即左截断）。默认值为 None（即不截断）。

（24）***<guided_decoding>：***如果提供，则引擎将根据这些参数构造引导解码对数概率处理器。默认值为 None。

（25）***<logit_bias>：***如果提供，则引擎将构造一个应用这些对数偏差的对数概率处理器。默认值为 None。

（26）***<allowed_token_ids>：***如果提供，则引擎将构造一个仅保留给定令牌 ID 得分的对数概率处理器。默认值为 None。

VLLM常用调参方法

「READING」

「READING」

场景一： KV 缓存空间不足

由于transformer架构的自回归性质，有时 KV 缓存空间不足以处理所有批处理请求。vLLM 可以抢占请求，为其他请求腾出 KV 缓存空间。当再次有足够的 KV 缓存空间可用时，被抢占的请求会重新进行计算。当这种情况发生时，会打印以下警告：

```
` WARNING 05-09 00:49:33 scheduler.py:1057]
Sequence group 0 is preempted by PreemptionMode.SWAP mode because there is not en
ough KV cache space. This can affect the end-to-end performance.
Increase gpu_memory_utilization or tensor_parallel_size
to provide more KV cache memory. total_cumulative_preemption_cnt=1 `
```

虽然这种机制确保了系统的稳健性，但抢占和重新计算可能会对端到端延迟产生不利影响。
如果您经常遇到 vLLM 引擎的抢占，考虑采取以下参数调整：

- **增加*<gpu_memory_utilization>***：vLLM 通过使用 gpu_memory_utilization% 的内存预先分配 GPU 缓存。通过增加此利用率，您可以提供更多的 KV 缓存空间。
- **减少 *<max_num_seqs>* 或 *<max_num_batched_tokens>*:** 可以减少批处理中的并发请求数量，从而需要更少的 KV 缓存空间。
- **增加 *<tensor_parallel_size>*：** 这种方法对模型权重进行分片，因此每个 GPU 有更多可用于 KV 缓存的内存。
- 你还可以通过 vLLM 公开的 Prometheus 指标监控抢占请求的数量。此外，你可以通过设置 **disable_log_stats=False** 来记录抢占请求的累积数量。

「READING」

场景二： 分块预填充
分块预填充允许将大型预填充分成较小的块，并将它们与解码请求一起批处理，可以优化 TTFT（第一个token的时间）

- 设置**enable_chunked_prefill=True**启用分块预填充，比如*llm = LLM(model="xx", enable_chunked_prefill=True)*

默认情况下，vLLM 调度程序优先考虑预填充，并且不会将预填充和解码批处理到同一个批次。此策略优化了 TTFT（第一个token的时间），但会导致 ITL（token间延迟）变慢，并且 GPU 利用率低下。
启用分块预填充后，策略会更改为优先处理解码请求。它会将所有待处理的解码请求批处理到批次中，然后再调度任何预填充。当有可用的 token_budget（max_num_batched_tokens）时，它会调度待处理的预填充。如果最后一个待处理的预填充请求无法放入 max_num_batched_tokens 中，它会将其分块。
此策略有两个优点：
（1）它通过优先处理解码请求来提高 ITL 和生成解码。
（2）它通过将计算密集型（预填充）和内存密集型（解码）请求定位到同一个批次来帮助实现更好的 GPU 利用率。

- 更改 ***max_num_batched_tokens*** 来调整性能。默认情况下，它设置为 512，在初始基准测试（llama 70B 和 mixtral 8x22B）中，它在 A100 上具有最佳 ITL。较小的 max_num_batched_tokens 可以实现更好的 ITL，因为有更少的预填充会中断解码。较高的 max_num_batched_tokens 可以实现更好的 TTFT，因为你可以将更多预填充放入批次中。
  - 如果 ***max_num_batched_tokens*** 与 ***max_model_len*** 相同，几乎等同于默认调度策略（除了它仍然优先处理解码）。
  - ***max_num_batched_tokens***的默认值（512）针对 ITL 进行了优化，它的吞吐量可能低于默认调度程序。
  - 建议将 ***max_num_batched_tokens*** 设置为大于 2048 以获得更高的吞吐量。

参考链接

1. https://docs.vllm.ai/en/stable/models/engine_args.html
2. https://github.com/vllm-project/vllm/blob/main/vllm/entrypoints/llm.py
3. https://vllm-zh.llamafactory.cn/models/performance.html