

LightLLM v1.0.0 介绍

主讲人：白世豪

日期：2025年3月8日

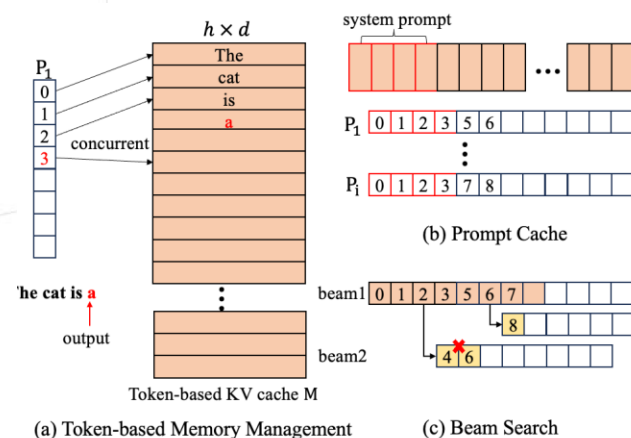
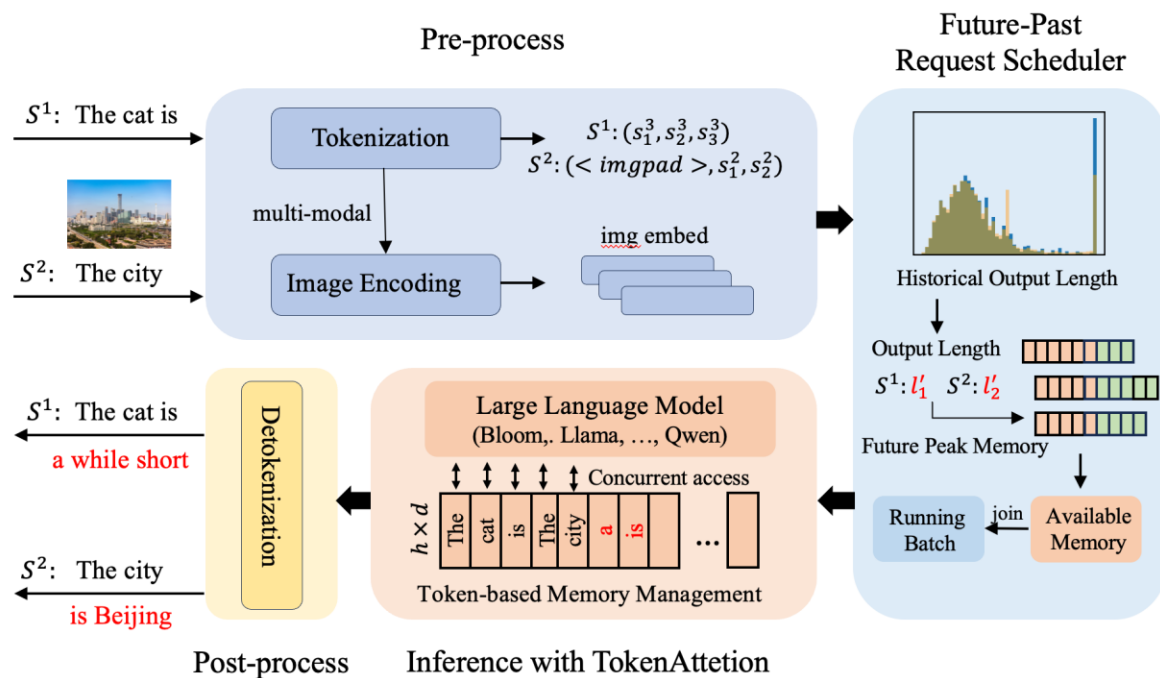
目录

- LightLLM新特性介绍
- PD分离原型实现
- DeepSeek优化介绍
- DeepSeek R1/V3 部署实战

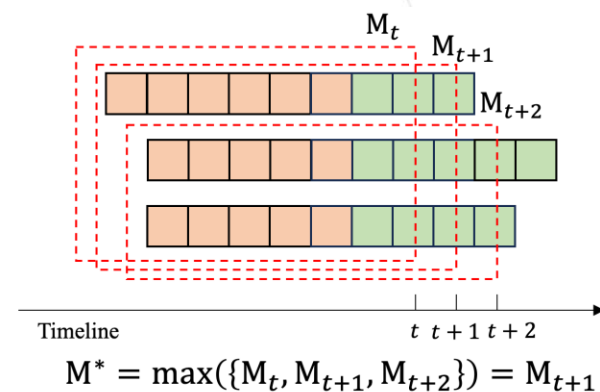
设计原则：轻量、可读性好、扩展性高

两大关键特性

- 多进程架构，异步cpu-gpu处理
- 高效的调度策略，被目前流行的llm推理框架采用



细粒度显存管理、零碎片化



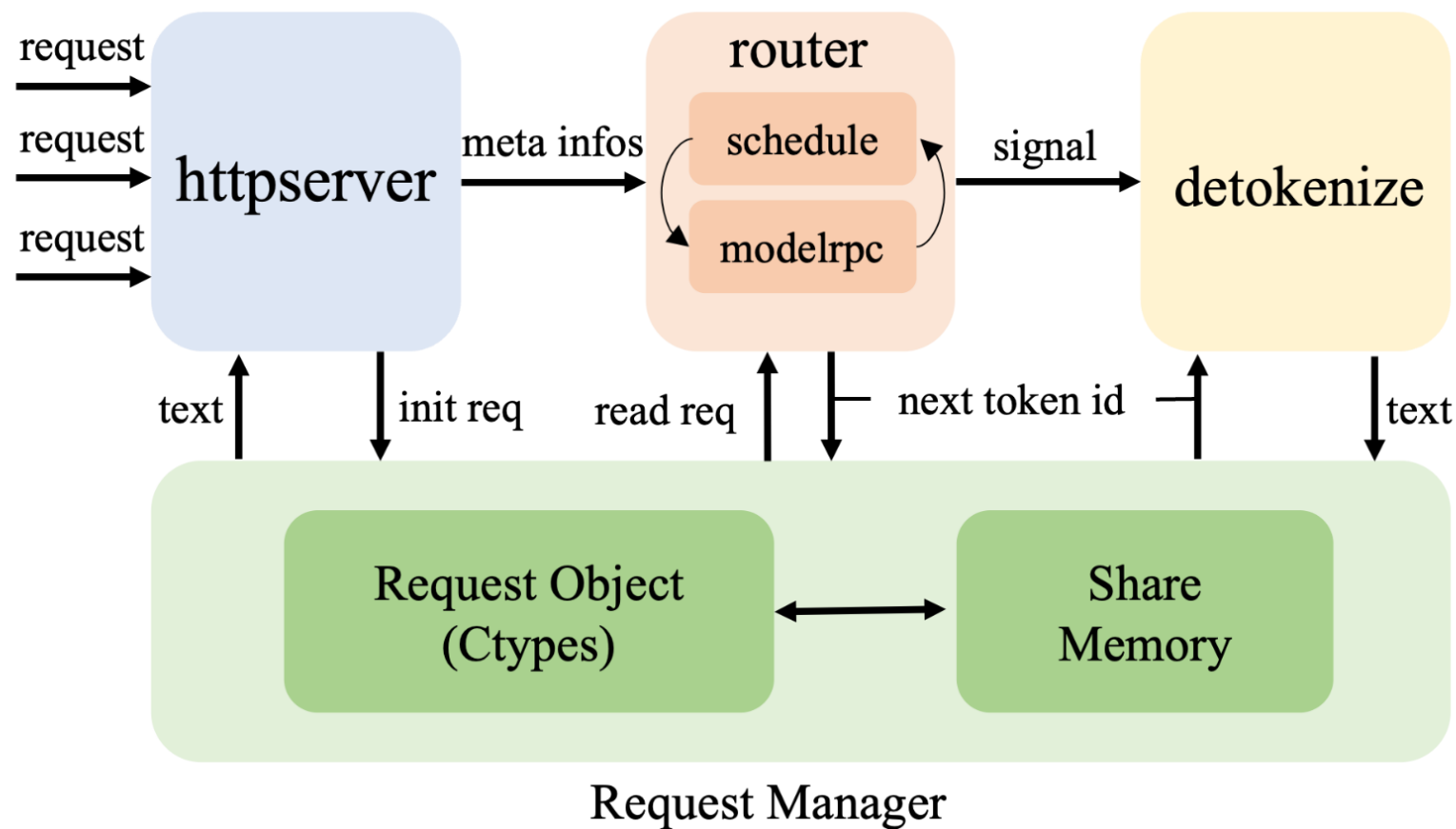
高效动态批次调度和扩缩容，最大化系统吞吐量

之前的问题

- 高并发时，模块间通信开销大
- 调度与模型推理串行，影响GPU利用率

V1.0.0

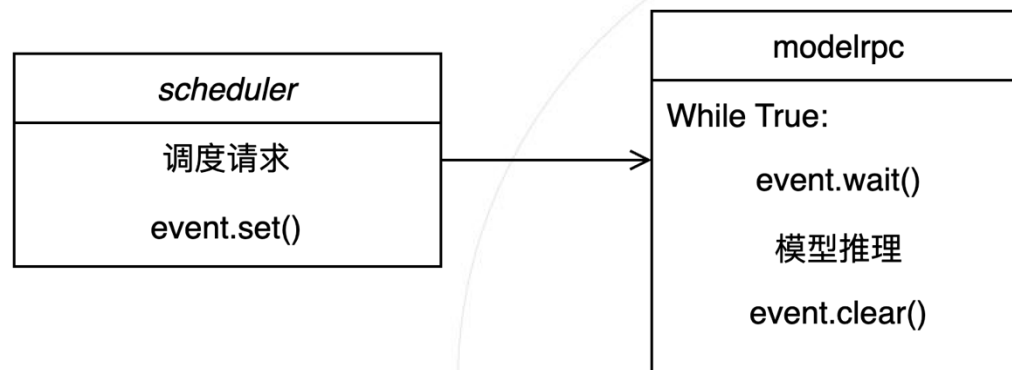
- 设计可跨进程直接读取的Req对象，降低通信开销
- 调度与模型推理折叠，提高GPU 利用率
- 面向SLA的带预测的调度策略
- 推理时torch的tensor管理器
- Layer级别的混合精度部署
- PD分离部署实现
- DeepSeek 专项优化



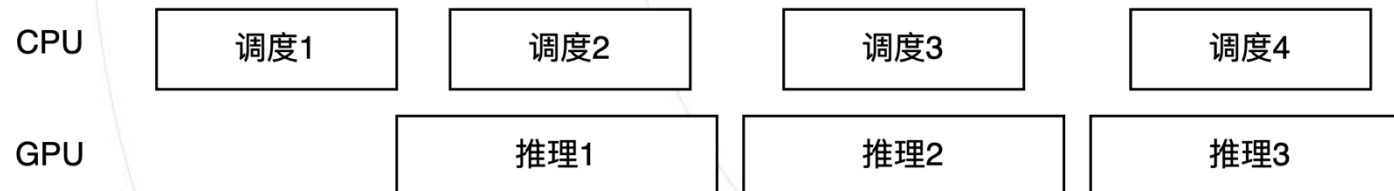
循环队列传递进程间消息



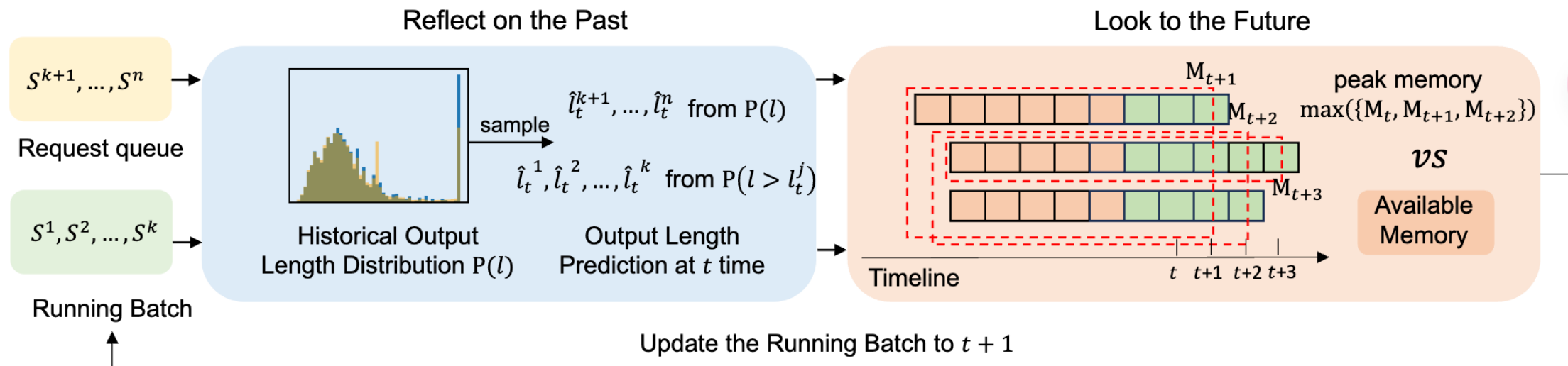
Router-modelrpc通信



调度推理折叠

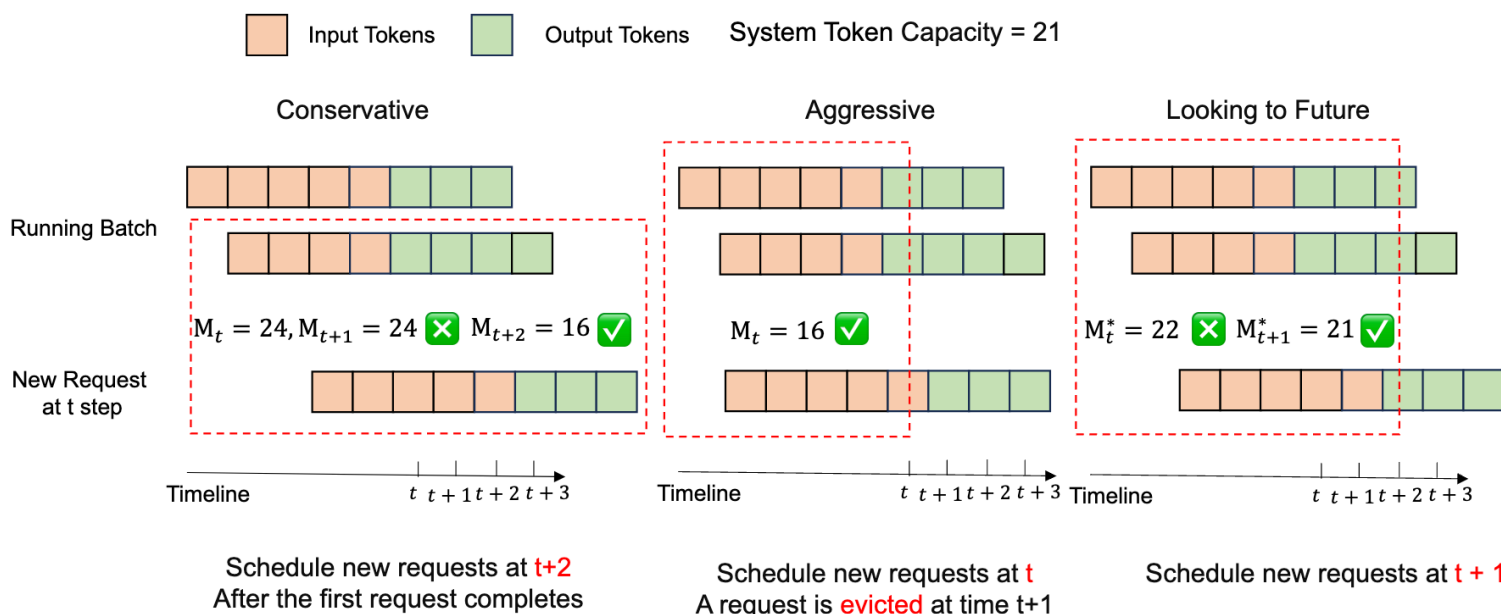


- SLA 约束保障了系统的首字延迟(TTFT), 包间延迟(TPOT)以及最大回复中断 (MTPOT)
- 在给定KV cache 容量的前提下, 合理的调度策略, 对于SLA保障下系统吞吐十分重要
- 提出了Past-Future scheduler, 包含两部分:
 1. 根据历史请求长度分布预测请求输出长度
 2. 计算Batch的峰值显存占用来决定是否调入新请求



Past-Future Request Scheduler

- 不同时刻调入新请求，Batch会造成不同的系统峰值显存占用
- Batch的最大显存使用，一定发生在请求退出时。
- 根据请求输入长度，当前输出长度和目标输出长度，计算Batch的峰值显存占用



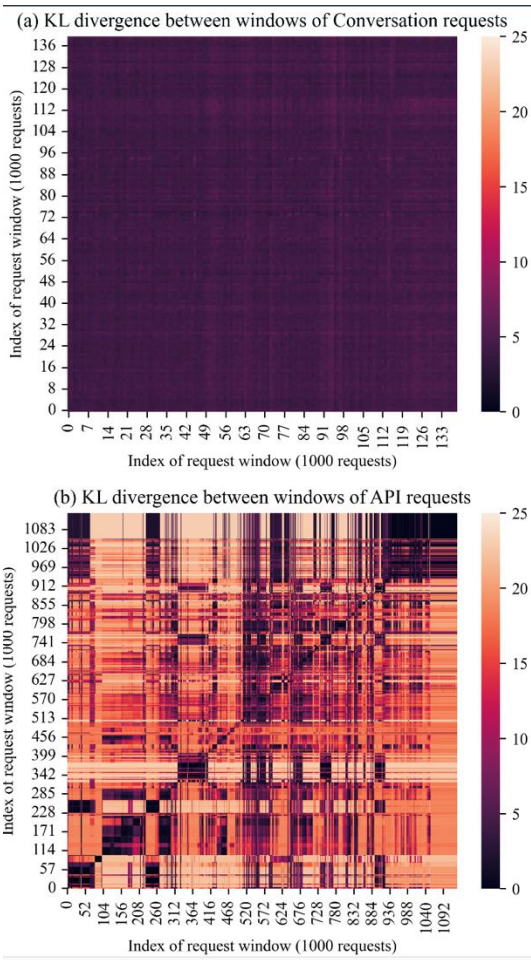
$$\{(l_p^i + l_t^i, \hat{l}_t^i - l_t^i) | S^i \in S\}$$

where $\hat{l}_t^i - l_t^i \geq \hat{l}_t^{i+1} - l_t^{i+1}, i = 1, 2, \dots, k$

$$M_i = \sum_{j=1}^i \{l_p^j + l_t^j\} + (\hat{l}_t^i - l_t^i) * i$$

$$M^* = \max(\{M_i | i = 1, 2, \dots, b\})$$

- 历史请求输出分布长度，在相邻的时间窗口，表现出一定的相似度
- 每个调度度，都对batch中的请求输出长度进行重新预测。结果从 $P(l > l_t^i)$ 中采样，保障请求不被逐出



Dataset	Method	Decoding Steps	Memory Utilization	Peak Memory	Evicted Reqs
Decode-heavy dataset-1	Theoretical optimum	300110	94.22%	98.19%	0%
	Past-Future	301730	92.67%	96.56%	6.26%
	Aggressive (water-level=99%)	289690	97.62%	102.18%	91.8%
	Conservative	490980	57.59%	60.66%	0%
Medium dataset-2	Theoretical optimum	692740	91.77%	97.75%	0%
	Past-Future	694080	91.32%	96.18%	6.06%
	Aggressive (water-level=99%)	651800	97.55%	103.06%	94.43%
Prefill-heavy dataset-3	Conservative	878260	72.38%	75.94%	0%
	Theoretical optimum	240270	96.11%	98.14%	0%
	Past-Future	247400	93.38%	95.55%	3.06%
	Aggressive (water-level=99%)	236650	97.58%	99.69%	33.66%
	Conservative	387720	59.56%	61.49%	0%

Table 1. Performance of different scheduling methods on multiple datasets

更丰富的量化功能支持

- 适配了多个架构的量化算子，例如vllm/torchao
- 支持加载浮点模型，在线量化
- 量化实现与模型推理部分解耦，可以高效轻松的加入各种新的量化方案
- 可通过配置文件设置layer级别的混合精度部署，便于做各种形式的量化实验验证

```
quant_type: vllm-w8a8
mix_bits:
  - name: "down_proj"
    quant_type: "none"
    layer_nums: [1, 2, 3] # Defaults to all layers, or you can specify a layer_num list.

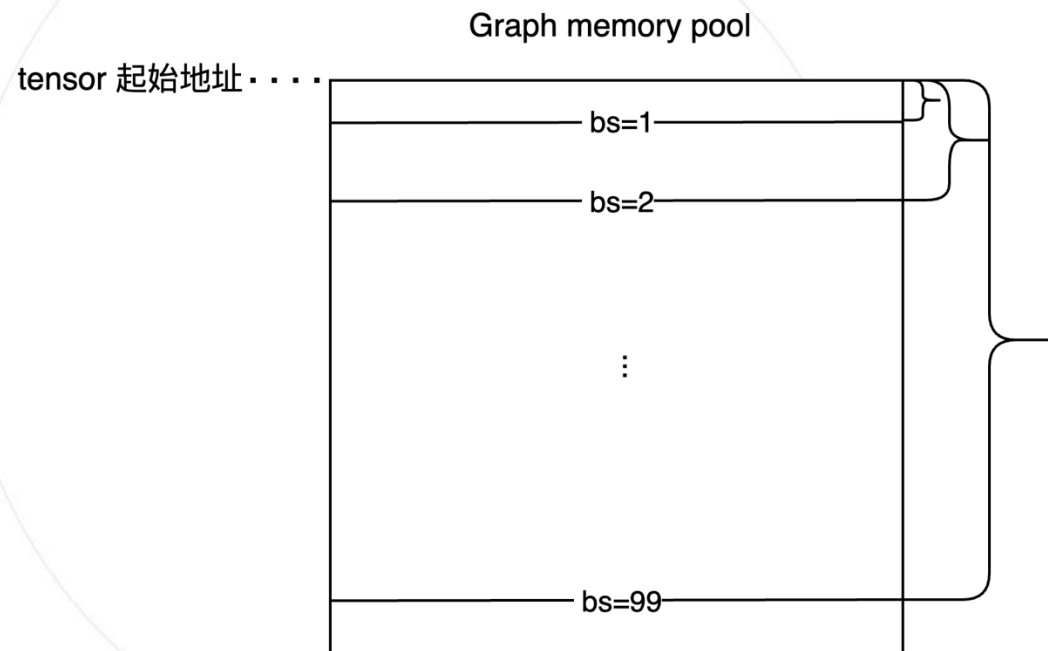
  - name: "o_proj"
    quant_type: "none"
    layer_nums: [1, 2, 3] # Defaults to all layers, or you can specify a layer_num list.
```

```
INFO 03-07 10:12:37 [mm_weight.py:175] Layer 1 down_proj is set to none
INFO 03-07 10:12:37 [mm_weight.py:175] Layer 2 q_proj is set to vllm-w8a8
INFO 03-07 10:12:37 [mm_weight.py:175] Layer 2 kv_proj is set to vllm-w8a8
INFO 03-07 10:12:37 [mm_weight.py:175] Layer 2 o_proj is set to none
INFO 03-07 10:12:37 [mm_weight.py:175] Layer 2 gate_up_proj is set to vllm-w8a8
INFO 03-07 10:12:37 [mm_weight.py:175] Layer 2 down_proj is set to none
INFO 03-07 10:12:37 [mm_weight.py:175] Layer 3 q_proj is set to vllm-w8a8
INFO 03-07 10:12:37 [mm_weight.py:175] Layer 3 kv_proj is set to vllm-w8a8
INFO 03-07 10:12:37 [mm_weight.py:175] Layer 0 q_proj is set to vllm-w8a8
INFO 03-07 10:12:37 [mm_weight.py:175] Layer 3 o_proj is set to none
INFO 03-07 10:12:37 [mm_weight.py:175] Layer 3 gate_up_proj is set to vllm-w8a8
INFO 03-07 10:12:37 [mm_weight.py:175] Layer 3 down_proj is set to none
INFO 03-07 10:12:37 [mm_weight.py:175] Layer 0 kv_proj is set to vllm-w8a8
INFO 03-07 10:12:37 [mm_weight.py:175] Layer 4 q_proj is set to vllm-w8a8
```

Tensor管理优化

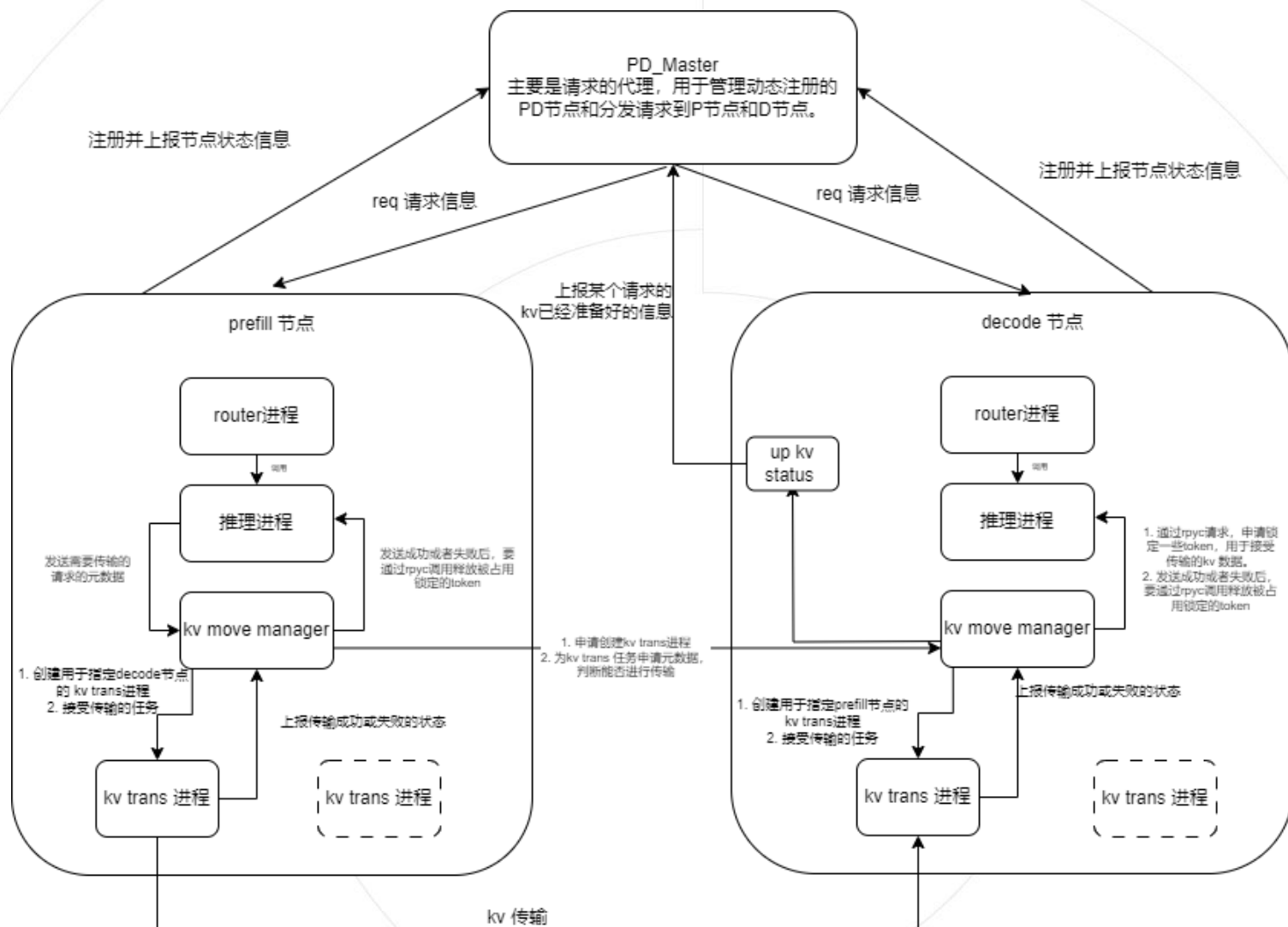
- 为了解决显存占用高且碎片化多时，torch 分配tensor 速度慢的问题，lightllm设计了一个推理时的tensor管理器，接管了torch的tensor释放
- decode时，bs小的cudagraph 和bs 大的 cudagraph共用同一块地址。

```
# 用于进行引用计数调整和判断
def custom_del(self: torch.Tensor):
    global g_cache_manager
    if hasattr(self, "storage_weak_ptr"):
        storage_weak_ptr = self.storage_weak_ptr
    else:
        storage_weak_ptr = self.untyped_storage()._weak_ref()
        UntypedStorage._free_weak_ref(storage_weak_ptr)
    if storage_weak_ptr in g_cache_manager.ptr_to_bufnode:
        g_cache_manager.changed_ptr.add(storage_weak_ptr)
    return
```

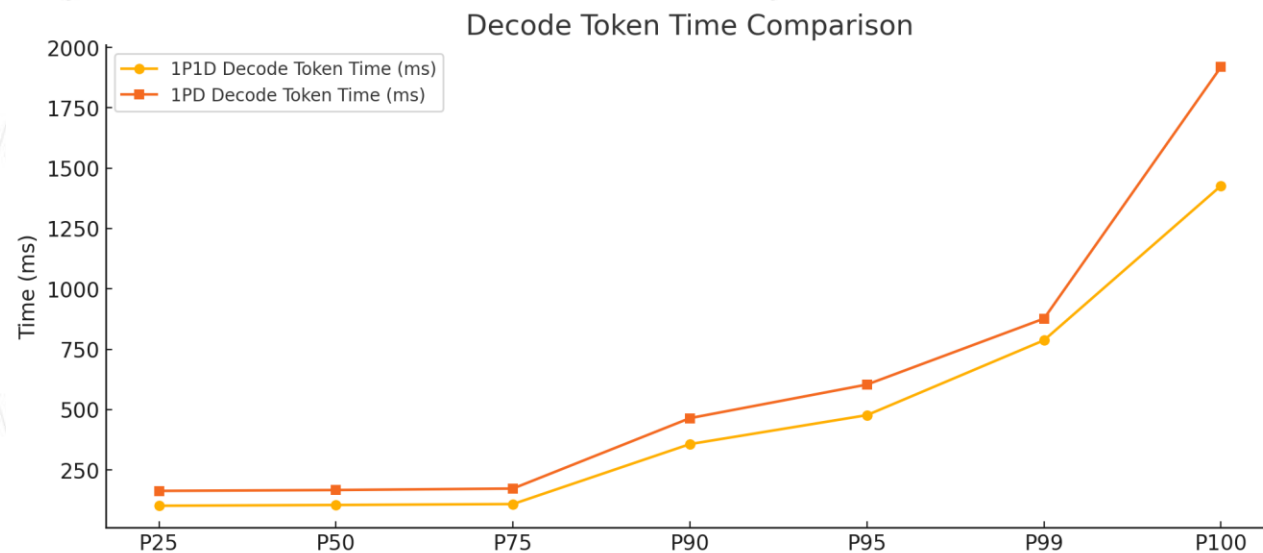
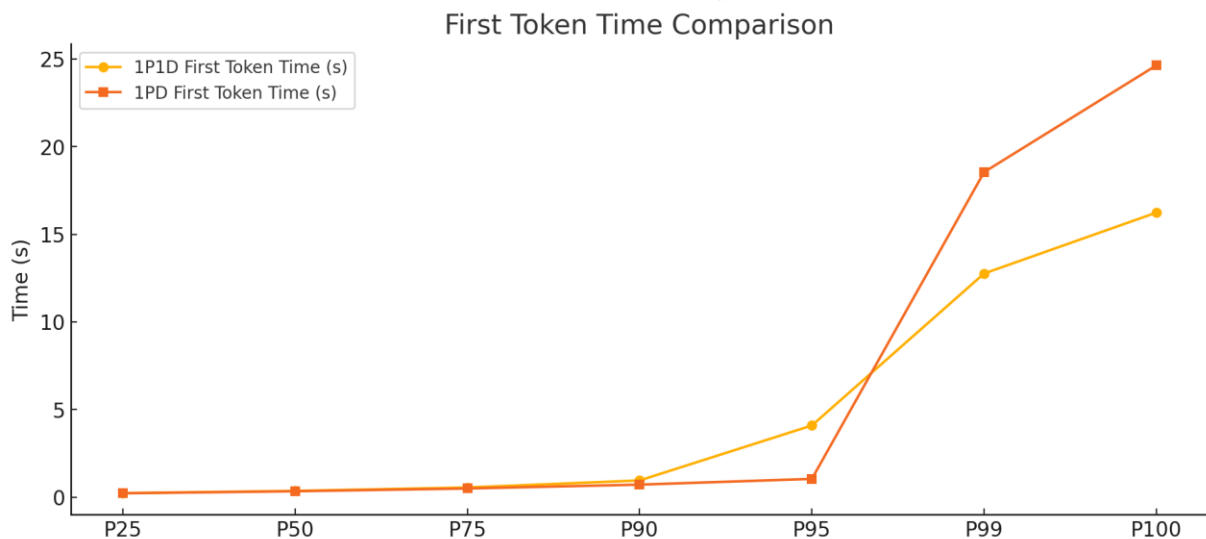


PD分离架构

- 可动态添加和删除P节点和D节点
- P, D节点通过websocket 定期向master 汇报节点状态, 便于支持各种调度和负载均衡策略。
- P节点申请传输进程之前, 查询D节点 KV cache命中情况, 仅传输缺失的内容减少传输开销 (尤其是长文本场景)



- PD 分离的最大收益来自于对SLA的提升
- 下图展示了在DeepSeek-V2 236B模型 在 8xH100机器上, 1P1D (100并发), 1PD(50并发)的对比结果
- PD分离可以持续保障更低的decode延迟和首字延迟, 带来更好的用户体验



DeepSeek 系列模型优化

KVCache 计算与存储策略对比

方案一：保存解压缩后的 KVCache，使用解压缩后的 KV 进行 Attention

方案二：计算并保存压缩的 KVCache，解压缩 KVCache 进行 Attention

方案三：计算保存并压缩 KVCache，使用压缩 KVCache 进行 Attention，q 矩阵吸收

方案四：计算保存并压缩 KVCache，使用压缩 KVCache 进行 Attention，q 矩阵吸收，vo 矩阵融合

$$\mathbf{q}_t^{(s)} \mathbf{k}_i^{(s)\top} = (\mathbf{x}_t \mathbf{W}_q^{(s)}) (\mathbf{c}_i \mathbf{W}_k^{(s)})^\top = \mathbf{x}_t (\mathbf{W}_q^{(s)} \mathbf{W}_k^{(s)\top}) \mathbf{c}_i^\top$$

bs= 1, length= 4000	方案一	方案二	方案三	方案四
PrefillMLA 计算量 (TFLOPS)	114.58	114.58	213.54	357.85
KVCache 显存占用 (GB)	18.75	0.263	0.263	0.263

Batch Size = 1, KV = 4000	方案一	方案二	方案三	方案四
DecodeMLA 计算量 (TFLOPS)	0.04	8.28	0.08	0.12
KVCache 显存占用 (GB)	18.75	0.263	0.263	0.263

Prefill: 方案二

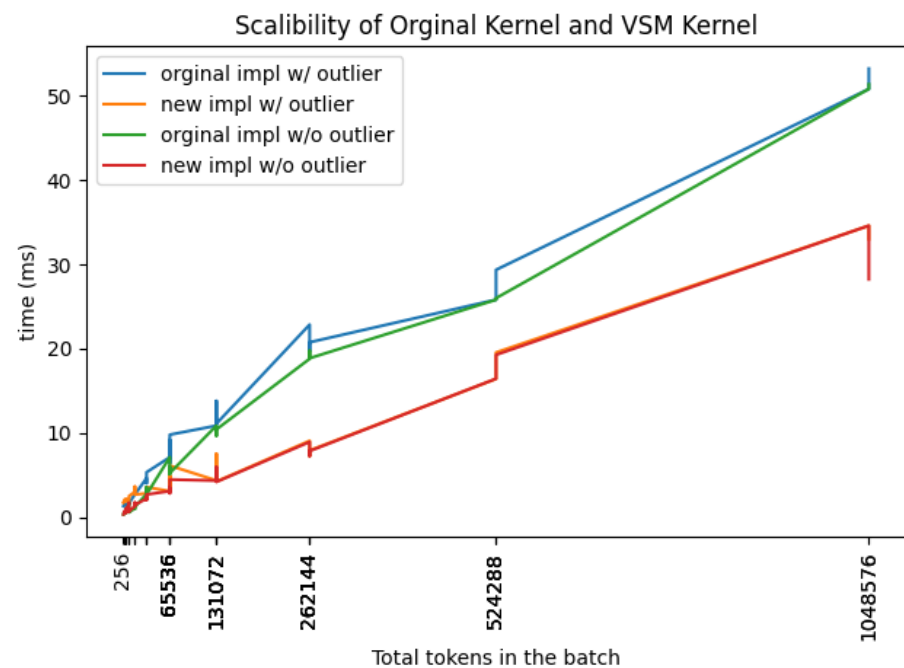
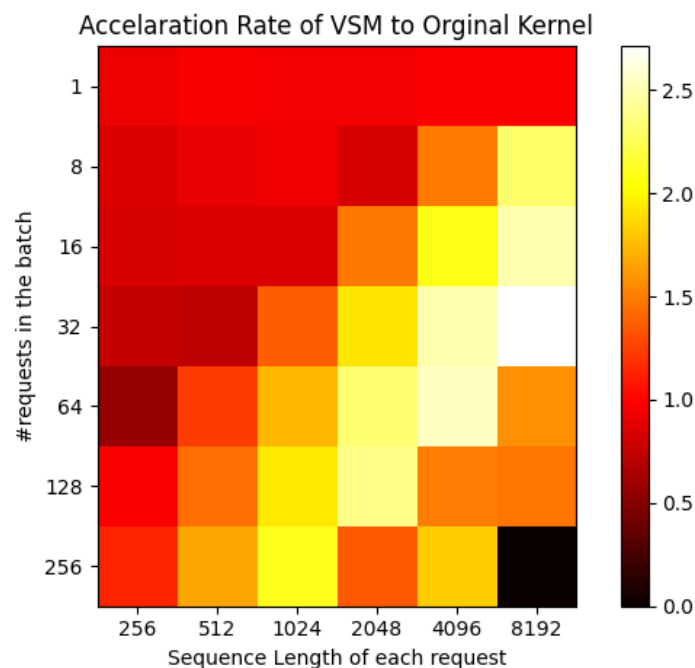
Decode: 方案三

DeepSeek V2模型 Prefill/Decode 的计算量和 KV 大小

DeepSeek 系列模型优化

MLA decode triton kernel 优化

- 根据 q head_num 和 kv head_num 的比值，将attention的 dot操作放到tensor core
- 根据设备sm数量，对bs内不同长度的请求，做负载均衡，避免cudagraph模式下长度不均衡造成的浪费。



单机部署-H200*8

```
LOADWORKER=8 python -m lightllm.server.api_server --model_dir ~/models/DeepSeek-R1 --tp 8 --graph_max_batch_size 100
```

双机部署-H100*16

Node 0

```
LOADWORKER=8 python -m lightllm.server.api_server --model_dir ~/models/DeepSeek-R1 --tp 16 --graph_max_batch_size 100 --  
nccl_host master_addr --nnodes 2 --node_rank 0
```

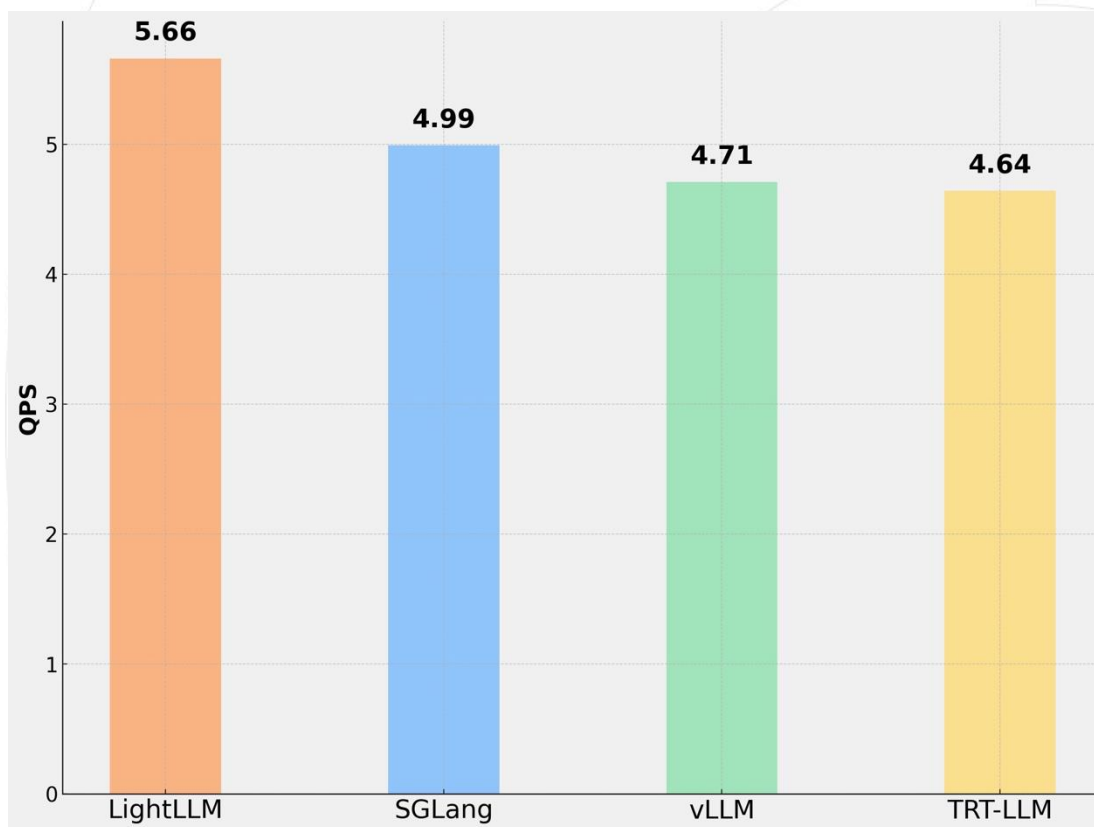
Node 1

```
LOADWORKER=8 python -m lightllm.server.api_server --model_dir ~/models/DeepSeek-R1 --tp 16 --graph_max_batch_size 100 --  
nccl_host master_addr --nnodes 2 --node_rank 1
```

DeepSeek-R1 单机H200*8 性能

对比框架: sglang==0.4.3, vllm==0.7.2, trtllm==0.17.0

实验设置: num_clients = 100. 输出长度: 1024, 输出长度: 均值为128的高斯分布



下一步开发计划

1. 开源DeepEP的接入，完成DeepSeek 跨机 DP/TP + EP的高效实现
2. 计算通信折叠，包括引入FLUX的MM和通信折叠，以及micro batch间的计算通信折叠等
3. PD 分离高阶调度方案的设计，达到更好的负载均衡
4. PD分离异构方案支持，PD采用不同的部署方案（不同parallelism，不同量化方案）
5. . . .



github 仓库地址



技术博客地址



欢迎加入!!!

感谢聆听!