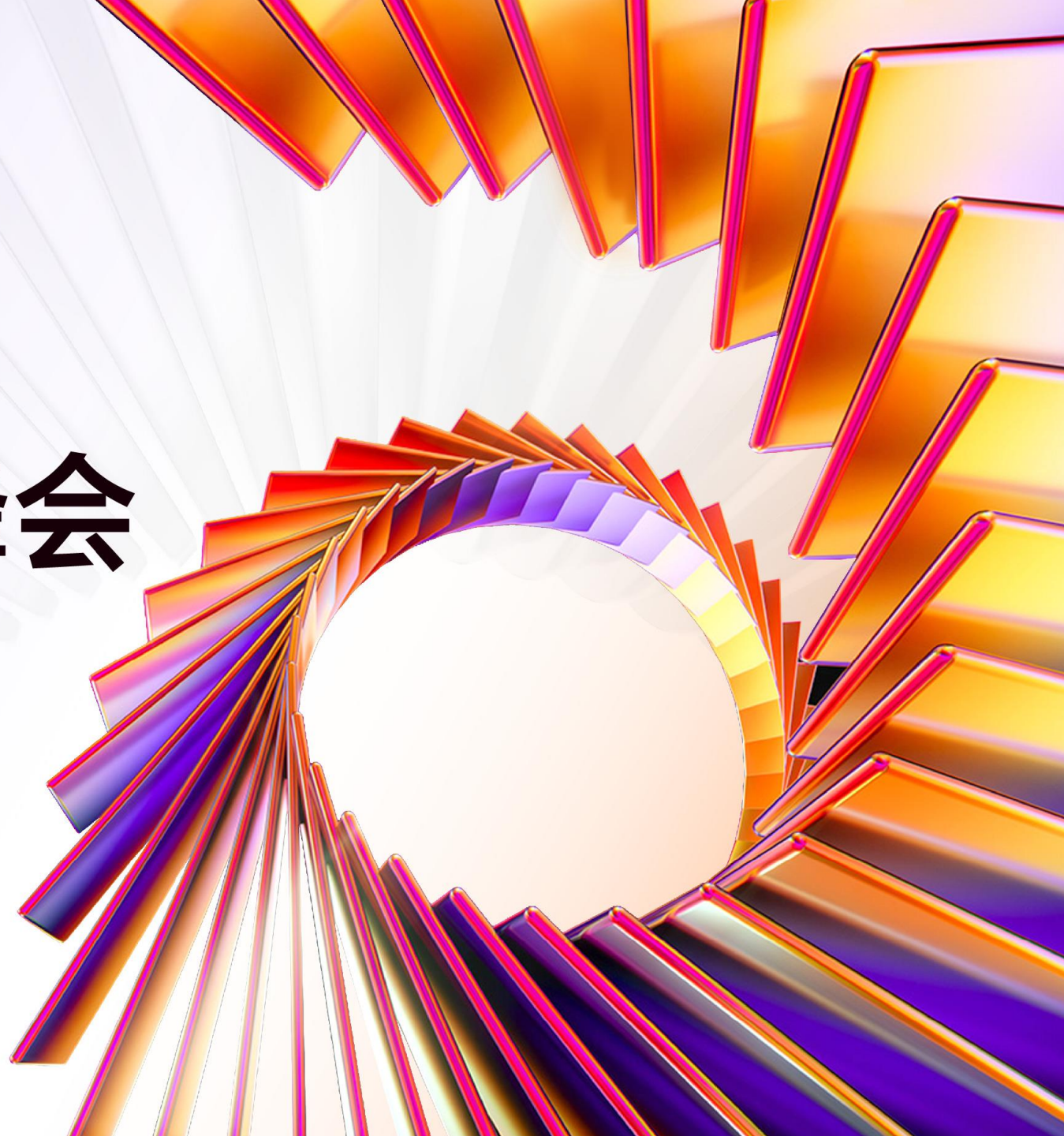
AiDD 6th 2025

2025 AI+ Development Digital Summit

AI+研发数字峰会

拥抱AI 重塑研发

05/23-24 | 上海站



AiDD 7th 2025

2025 AI+研发数字峰会

拥抱AI 重塑研发

AI+ Development Digital Summit

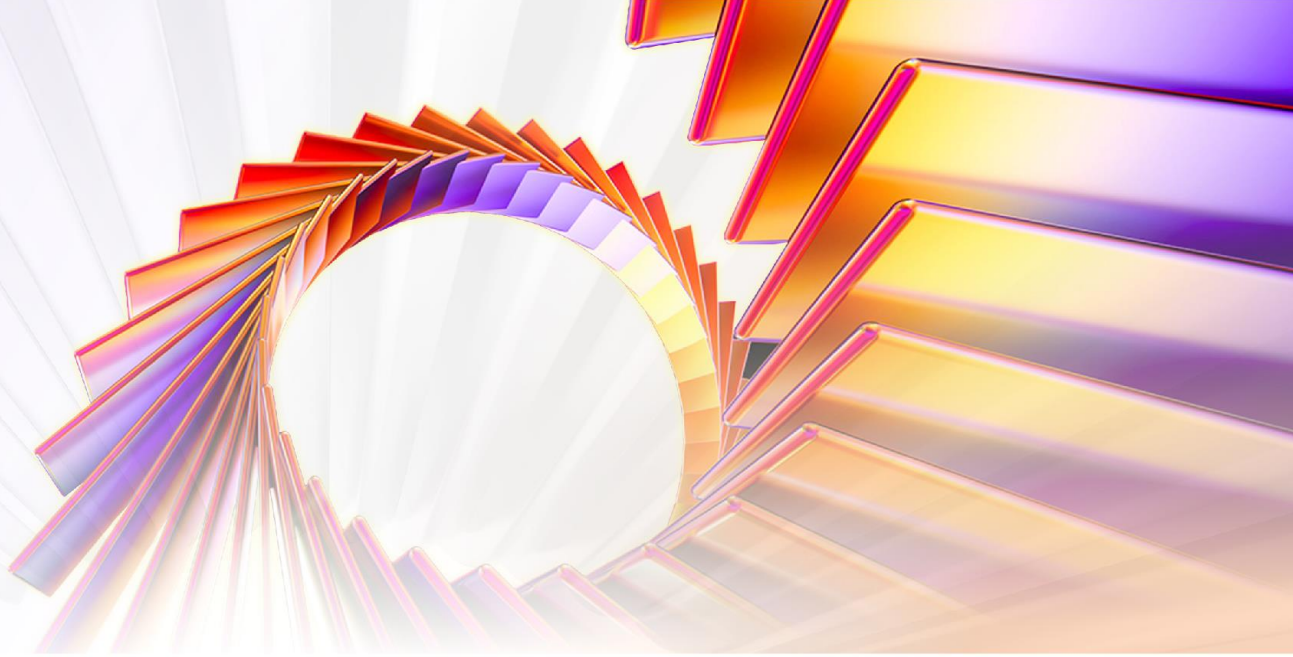
查看会议详情

下一站预告 08/08-09 | 北京站 11/14-15 | 深圳站

北京站论坛设置

- AI+ 金融业务创新
- 大模型安全与对齐
- 具身智能与机器人
- 大模型和 AI 应用评测
- 智能需求工程
- 大模型应用开发框架与实践
- 代码生成及其改进
- 智能存储与检索技术
- 智能体与研发效率工具
- 智能体经济（Agentic Economy）
- AI+ 新能源汽车
- 下一代知识工程
- AI 产品运营与出海策略
- 智能测试工具的开发与应用
- AI 前沿技术探索与实践

AiDD 6th 2025 | 05/23-24 | 上海站
2025 AI+ Development Digital Summit
AI+研发数字峰会
拥抱AI 重塑研发



大模型在得物部署优化实践

孟令公 | 得物

孟令公

得物 机器学习高级专家

得物机器学习高级专家，算法工程方向，主要负责得物算法平台的相关研发工作。在得物从0到1打造通用大模型训练和推理平台。曾就职于腾讯，阿里等多家互联网大厂。2022年加入得物，专注于大模型相关技术，包括推理加速与各应用场景落地。

目录
CONTENTS

背景

Deepseek-r1等大模型的火爆标志着本地部署大模型的需求日益增长。我们将探讨如何优化本地部署大模型的性能，并结合我们的实践进行评测分析。同时，我们还将分享如何在本地高效部署完整版本的Deepseek-r1大模型。

优化方法大多来源于开源社区，但我们希望大家能更多关注这些优化背后的**思路**。

背景

大模型推理性能的两个关键指标

吞吐量

- 传统上，我们用每秒请求数（QPS）来衡量吞吐量，即系统每秒能够处理多少请求。
- 大模型有一个重要指标——每秒Token数（tokens/s），它反映了系统每秒能处理的输入或输出Token数量。

响应时间

- 系统处理每个请求所需的时间。
- 大模型有一个指标——首个Token到达时间（TTFT: Time To First Token），即从开始处理请求到输出第一个Token所需的时间。

如何设计高性能的大模型推理引擎

性能足够高

- CPU与GPU分离设计

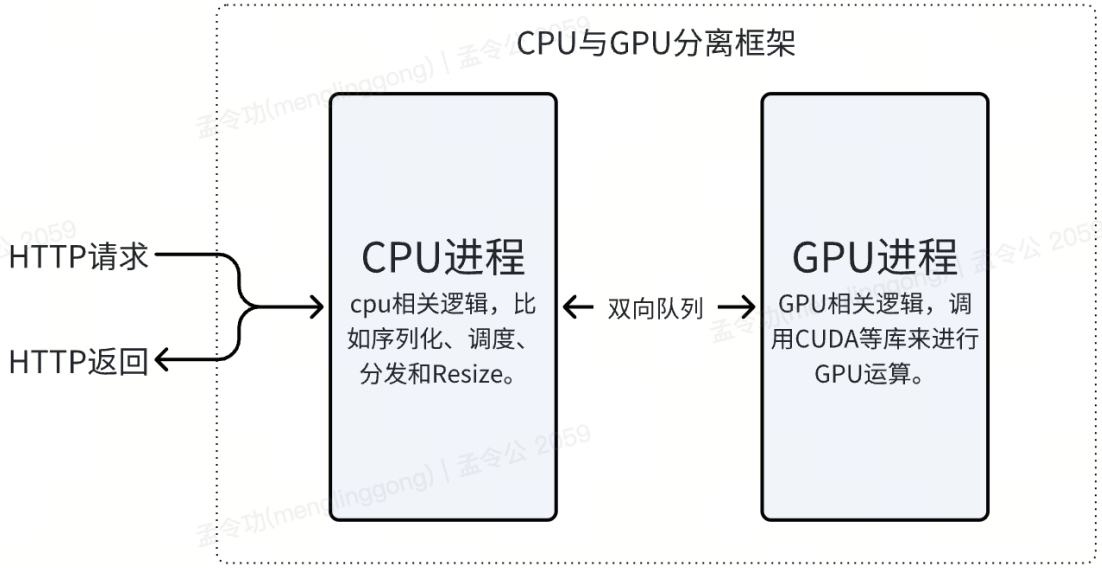
扩展性好

- 模块高内聚低耦合

如何设计高性能的大模型推理引擎

CPU与GPU分离设计

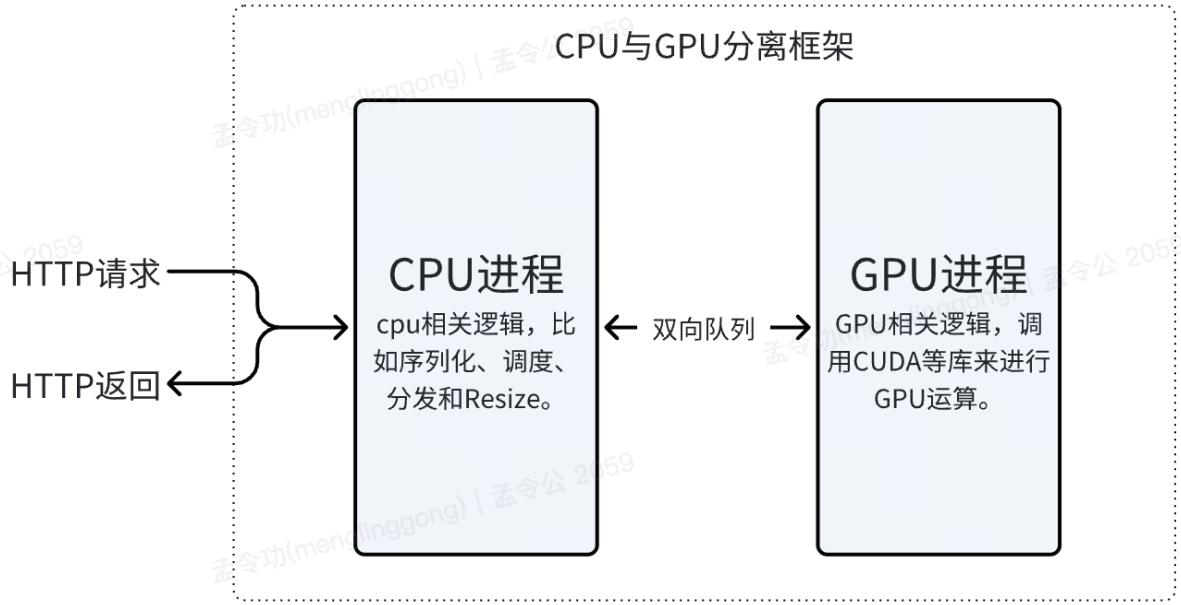
解决Python GIL锁带来的问题



1. 在传统的Python多线程环境中，CPU密集型任务与GPU任务会争夺GIL，导致GPU利用率低和高并发场景下响应速度差。
2. CPU与GPU分离解决了Python中全局解释器锁（GIL）带来的性能瓶颈问题。
3. 通过分离CPU与GPU，避免了GIL竞争，从而提升了GPU任务的执行效率和系统性能。

如何设计高性能的大模型推理引擎

CPU与GPU分离设计



性能提升

推理服务框架类型	QPS	耗时	GPU使用率
单进程设计(GPU与GPU任务分布多个线程)	4.5	1.05s	2%
CPU与GPU多进程分离设计	27.43	437ms	12%

如何设计高性能的大模型推理引擎

CPU与GPU分离设计

Performance Enhancements

To make sure we can keep GPUs busy, we made several enhancements:

Separating API server and inference engine into different processes (PR #6883)

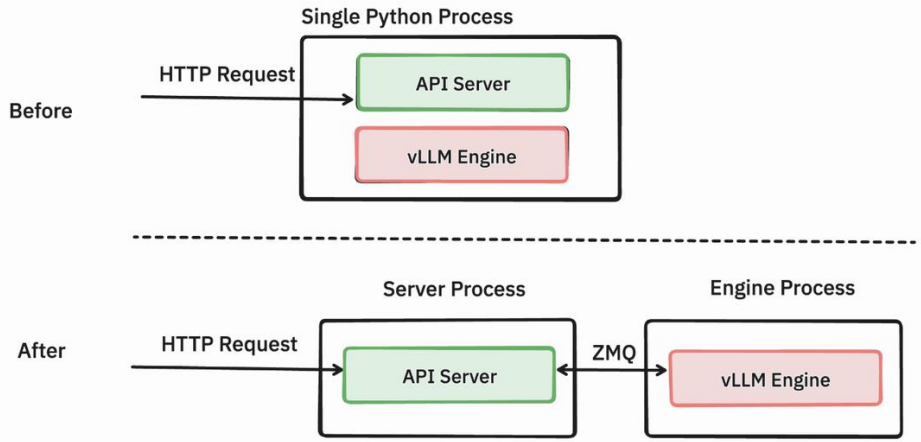


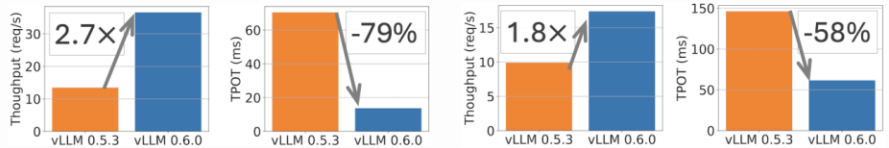
Illustration of the serving process architecture before and after. We separated the http serving component from the vLLM engine, and connected them with a ZMQ socket. This architecture ensures both CPU heavy components are isolated from each other.

性能提升

vLLM v0.6.0: 2.7x Throughput Improvement and 5x Latency Reduction

Sep 5, 2024 • vLLM Team

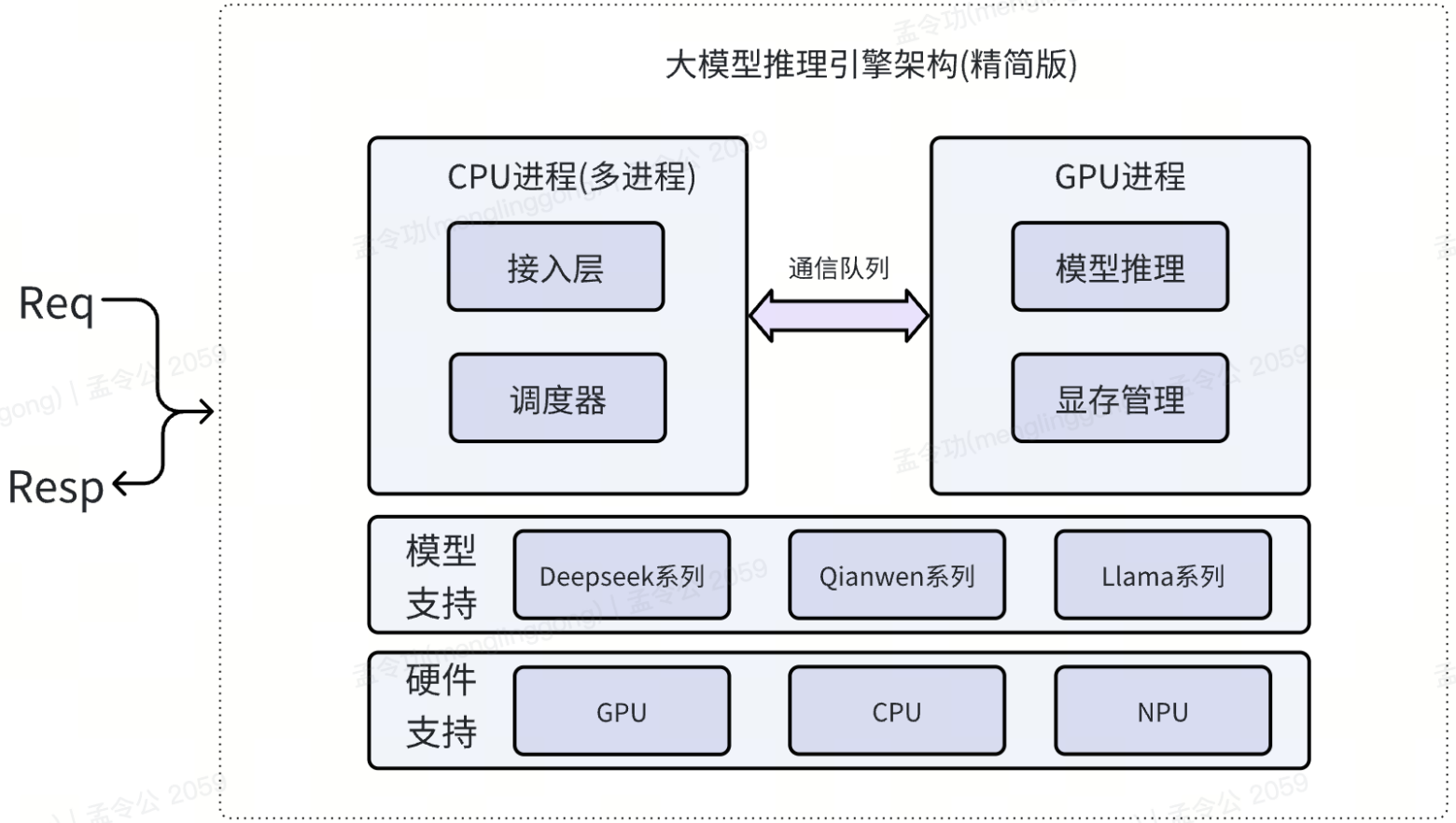
TL;DR: vLLM achieves 2.7x higher throughput and 5x faster TPOT (time per output token) on Llama 8B model, and 1.8x higher throughput and 2x less TPOT on Llama 70B model.



Performance comparison between vLLM v0.5.3 and v0.6.0 for Llama 8B on 1xH100 and 70B on 4xH100 on ShareGPT dataset (500 prompts). TPOT measured at 32 QPS.

如何设计高性能的大模型推理引擎

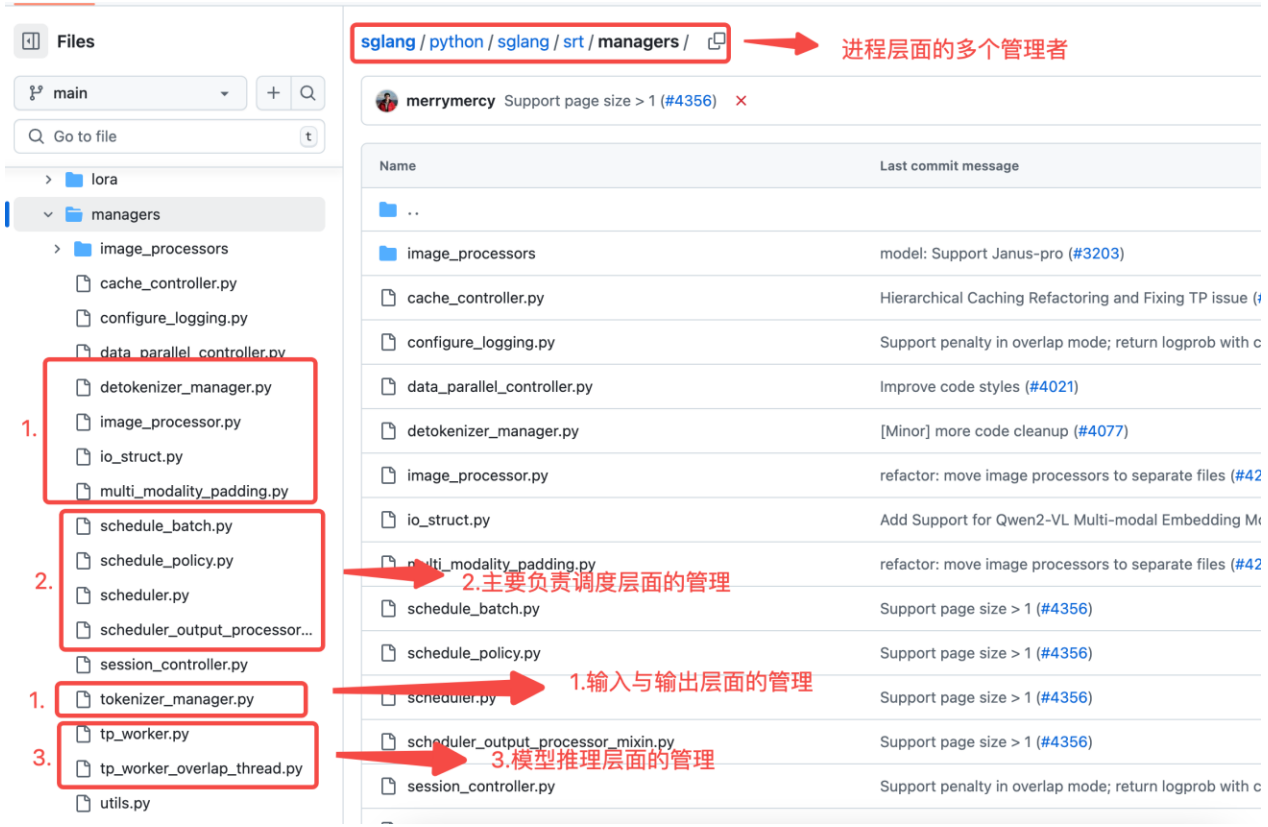
扩展性好的架构



如何设计高性能的大模型推理引擎

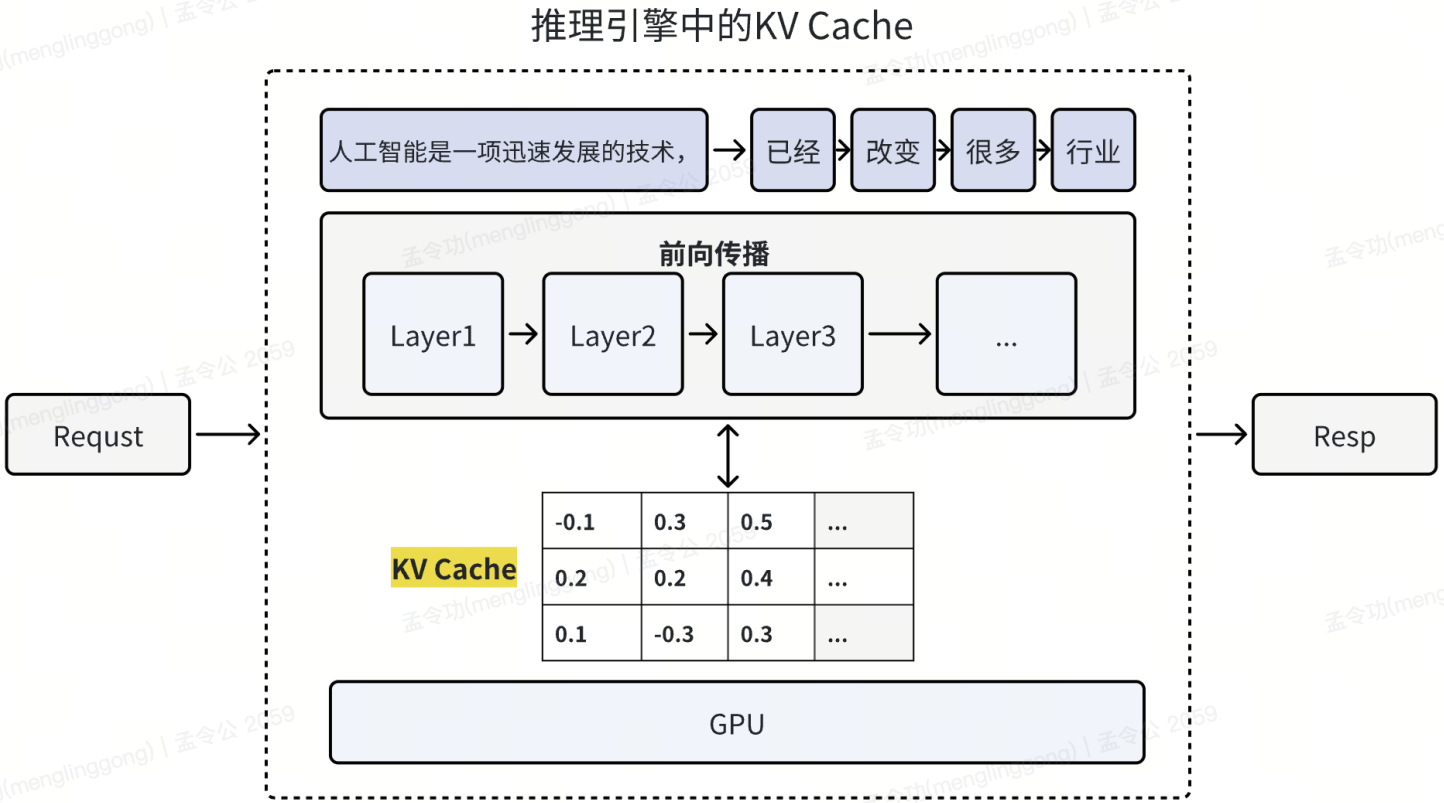
扩展性好的架构-sglang

sglang进程层面管理类



解决显存碎片问题，大幅提升吞吐—Paged Attention

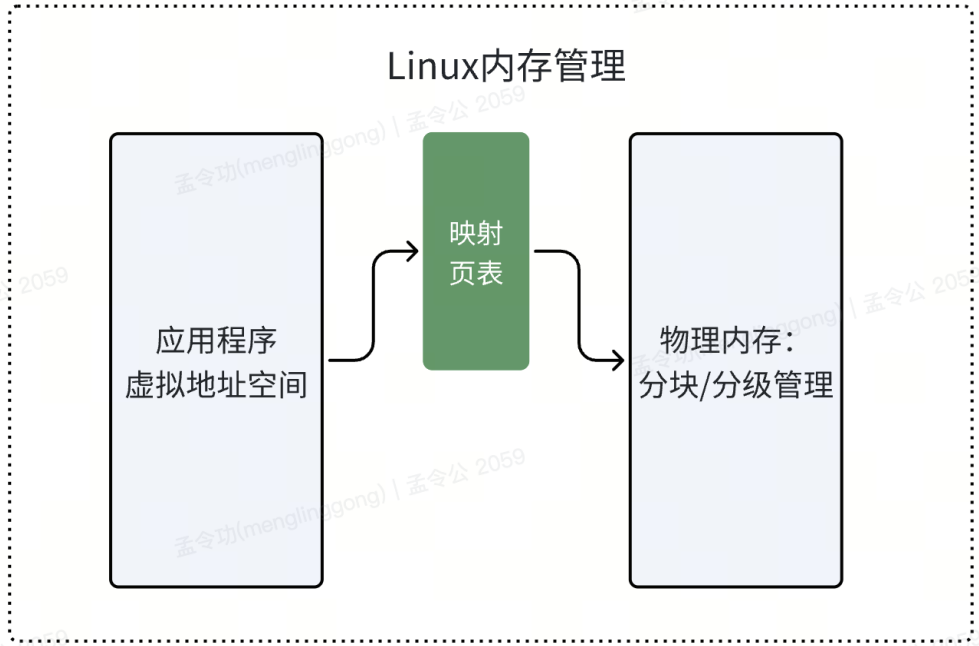
KV-Cache带来显存碎片问题

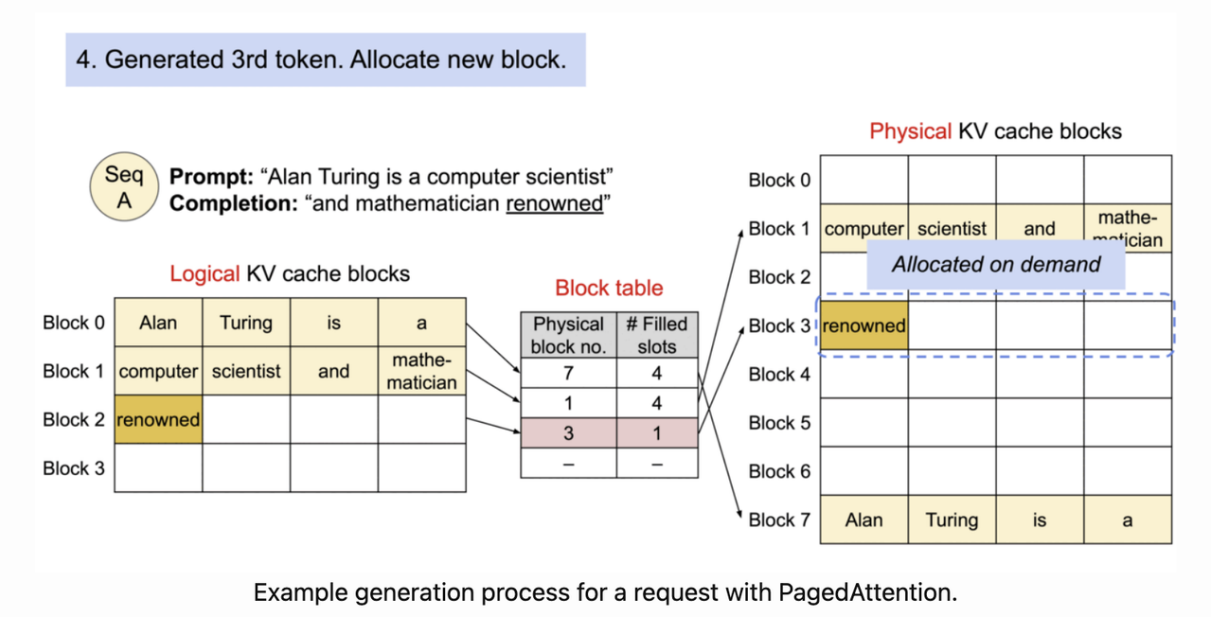


1. 大部分推理过程都涉及注意力计算（Attention）
2. 每次计算都需要申请并使用一个名为 kvcache 的缓存。
3. 随着请求的不断增加，kvcache 的大小与数量会逐步上升，而且它会被频繁地被申请和释放。
4. 如果不对 kvcache 使用的 GPU 显存进行有效管理，显存碎片将大量累积，最终可能导致系统性能下降甚至崩溃。

解决显存碎片问题，大幅提升吞吐—Paged Attention

Paged Attention工作原理



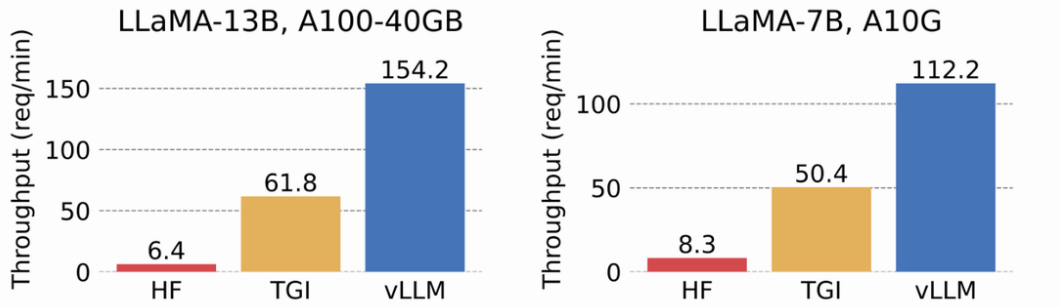


Example generation process for a request with PagedAttention.

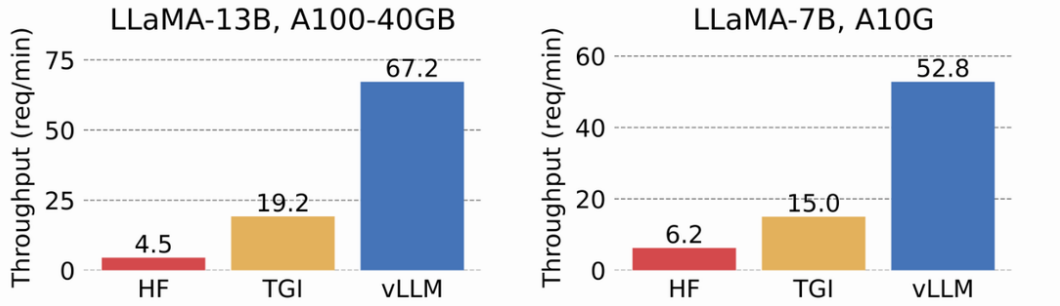
解决显存碎片问题，大幅提升吞吐—Paged Attention

性能提升

与 HuggingFace Transformers 相比，吞吐量可提升至 24 倍；与 HuggingFace TGI 相比，提升可达 3.5 倍。



Serving throughput when each request asks for *one output completion*. vLLM achieves 14x - 24x higher throughput than HF and 2.2x - 2.5x higher throughput than TGI.



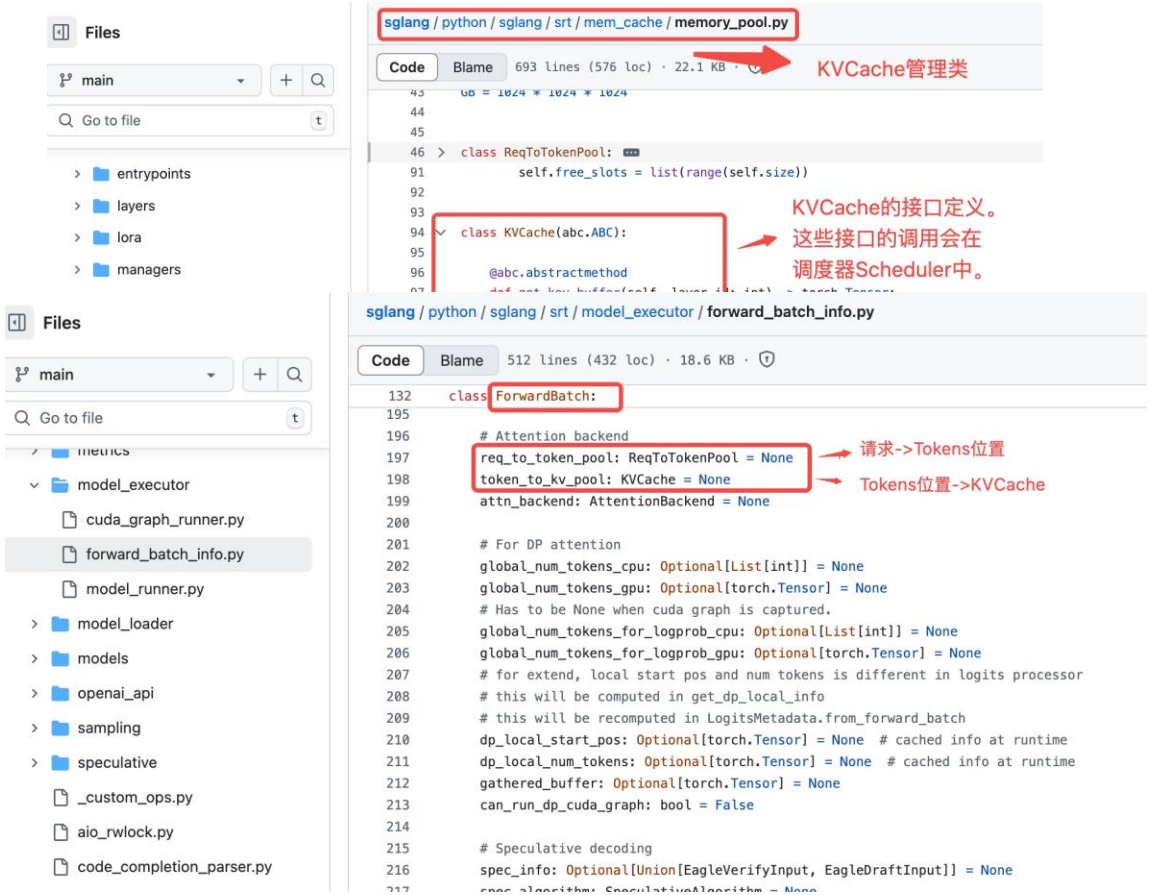
Serving throughput when each request asks for *three parallel output completions*. vLLM achieves 8.5x - 15x higher throughput than HF and 3.3x - 3.5x higher throughput than TGI.

图片来自 vLLM: Easy, Fast, and Cheap LLM Serving with PagedAttention

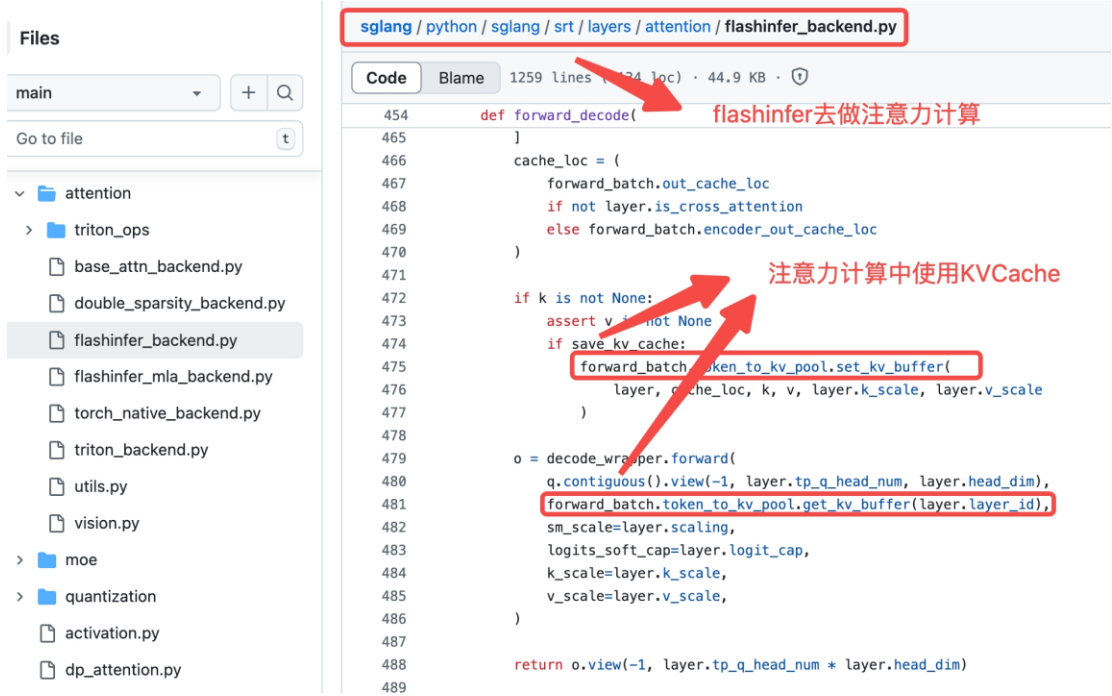
解决显存碎片问题，大幅提升吞吐—Paged Attention

Show Code

KVCache实现

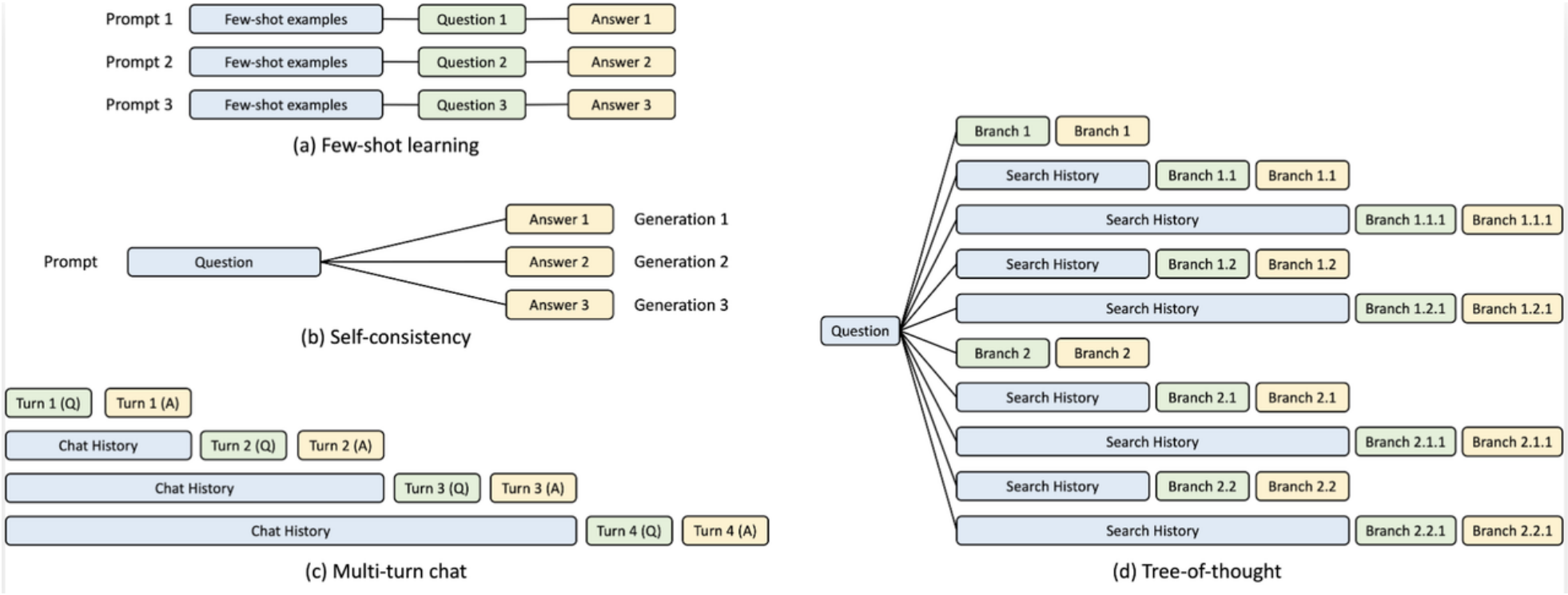


注意力计算使用KVCache



缓存之前请求的计算结果，减少重复计算—Radix Attention

这些场景还可以优化



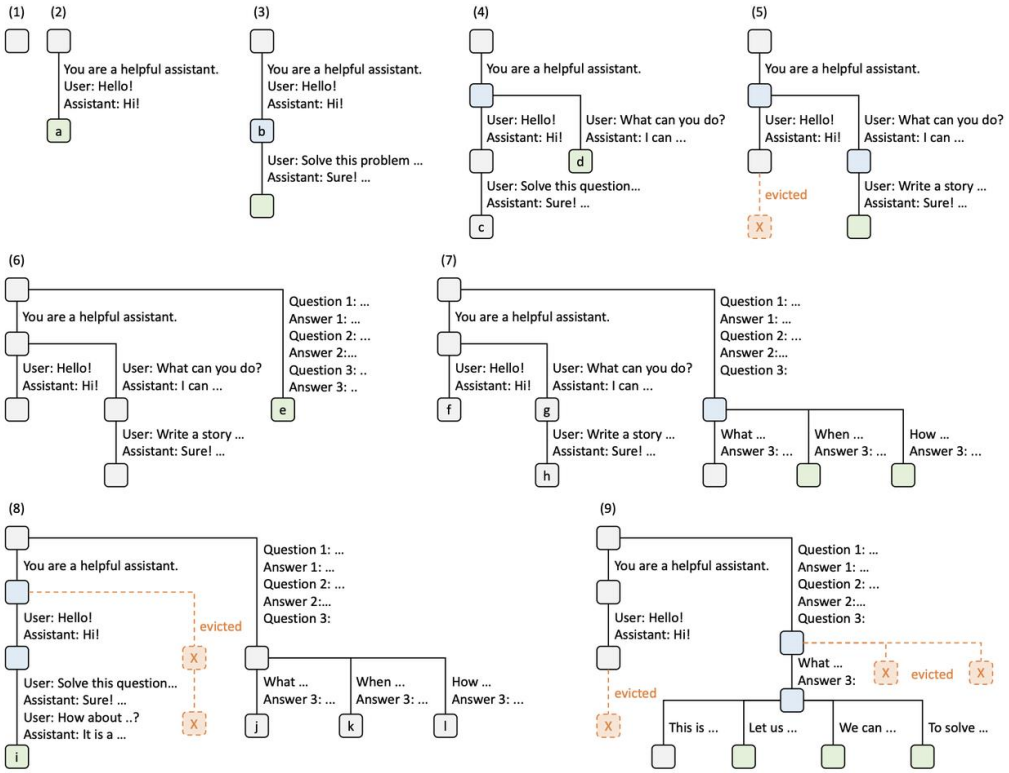
图片来自 Fast and Expressive LLM Inference with RadixAttention and SGLang

上图中可共享部分场景包括少量示例学习、自我一致性中的问题、多轮对话中的聊天历史，以及思维树中的搜索历史。在这些场景中，每次请求都会重复计算这些Prompt中可共享的部分，这些会造成大量的计算资源浪费。

有没有办法将这些重复部分的计算结果(KV Cache)缓存起来，下次请求时直接使用呢?

缓存之前请求的计算结果，减少重复计算—Radix Attention

工作原理/一个例子



图片来自[5] Fast and Expressive LLM Inference with RadixAttention and SGLang

1.初始状态：基数树最初为空。

2.第一场聊天开始：用户发送“你好！”，助手回复“你好！”，系统提示、用户消息和助手回复被合并为基数树中的一条边，连接到新节点。

3.第一场聊天继续：同一会话中收到新消息，服务器在基数树中找到已有前缀并重用其KV缓存，新消息作为新节点附加到树上。

4.第二场聊天开始：另一个用户开始新的聊天会话，服务器分割现有节点，使两个会话共享公共前缀和KV缓存。

5.第二场聊天继续并进行淘汰：第二场聊天继续，因内存限制，第一场聊天中最少近期使用的节点被淘汰释放空间，新消息在共享节点后添加到树上。

6.处理少样本学习查询：服务器收到不与现有节点共享前缀的少样本学习请求，根节点被分割以容纳新序列，请求作为单独分支插入树中。

7.处理一批少样本学习查询：收到一批共享相同少样本示例的查询，分割第六步的节点以在这些查询间实现KV缓存共享，最大化重用并减少冗余计算。

8.第一场聊天继续并进行淘汰：第一场聊天继续，基于最近最少使用（LRU）策略，淘汰第二场聊天的节点以高效管理内存。

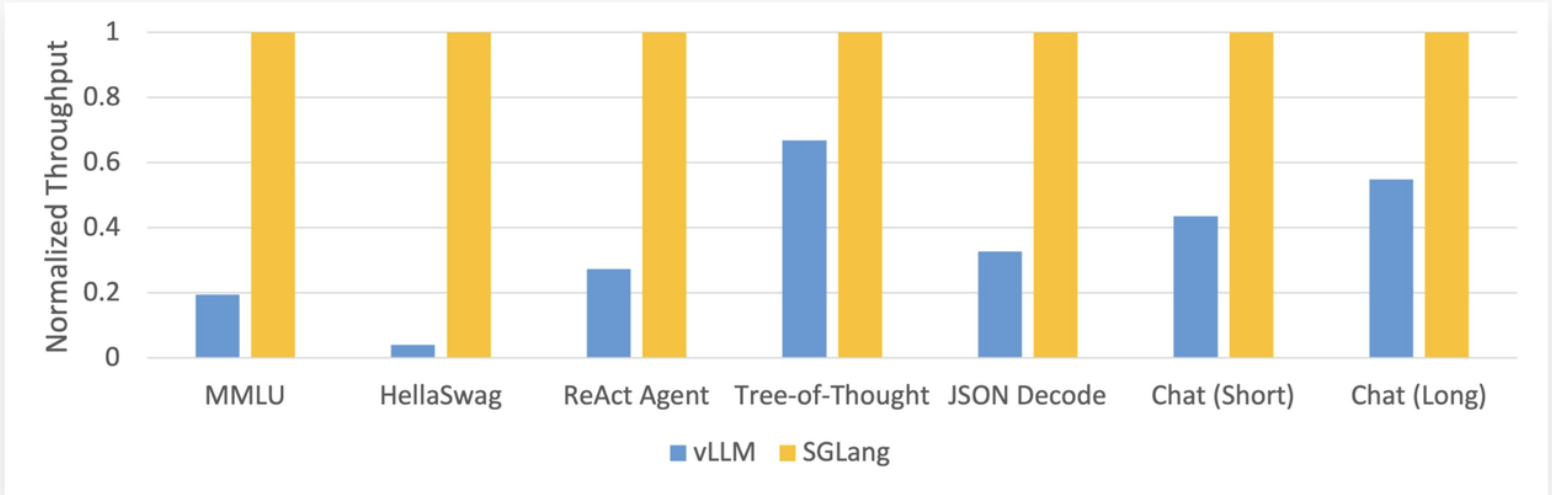
9.自一致性采样并进行淘汰：服务器收到生成多个答案的请求，为自一致性目的，按照LRU策略淘汰较不重要的节点以为新请求分配内存。

缓存之前请求的计算结果，减少重复计算—Radix Attention

性能提升

SGLang给出的Radix Attention性能对比效果，与当前系统相比，SGLang吞吐提升了5倍以上。

Figure 1: Throughput of Different Systems on LLM Tasks (Llama-7B on A10G, FP16, Tensor Parallelism=1)

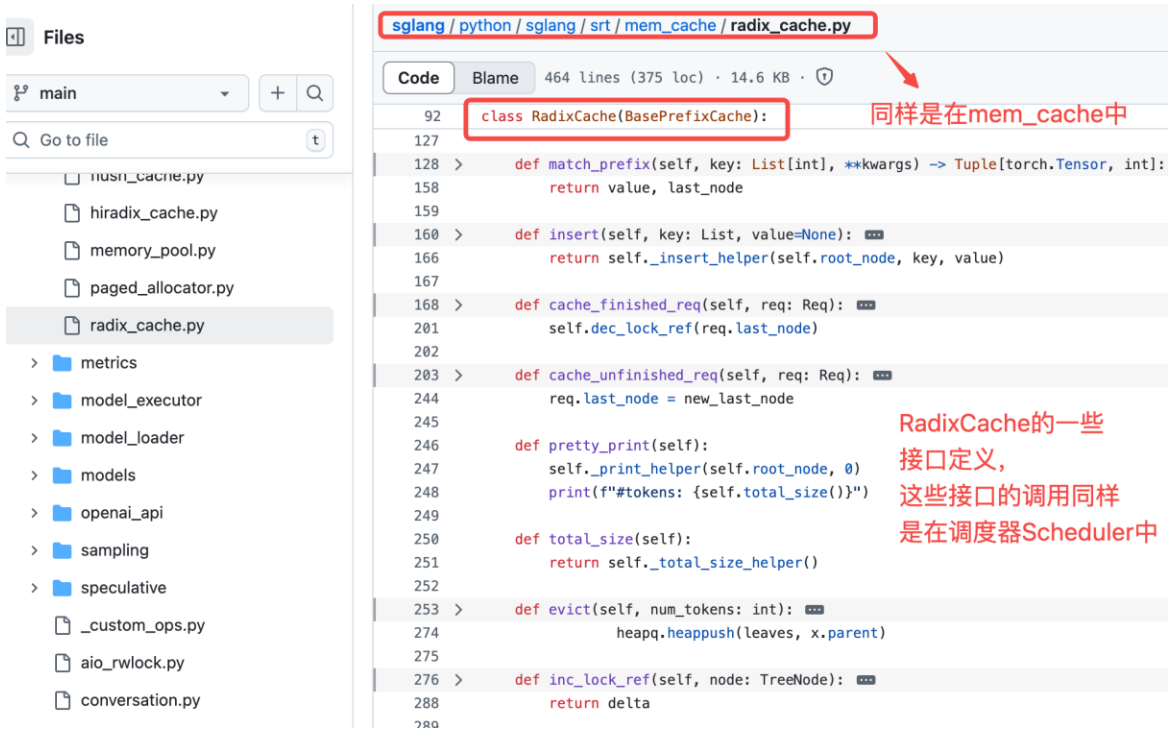


图片来自[5] Fast and Expressive LLM Inference with RadixAttention and SGLang

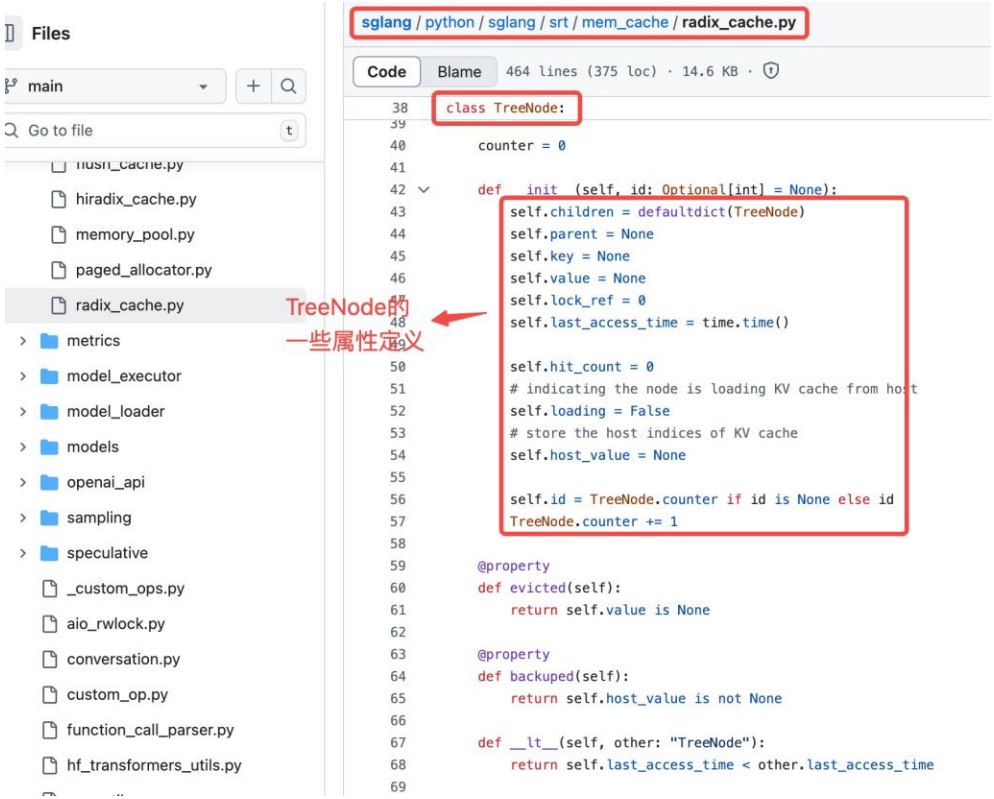
缓存之前请求的计算结果，减少重复计算—Radix Attention

Show Code

RadixCache定义



TreeNode节点定义



请求分块处理，避免单个请求卡顿—Chunked Prefill

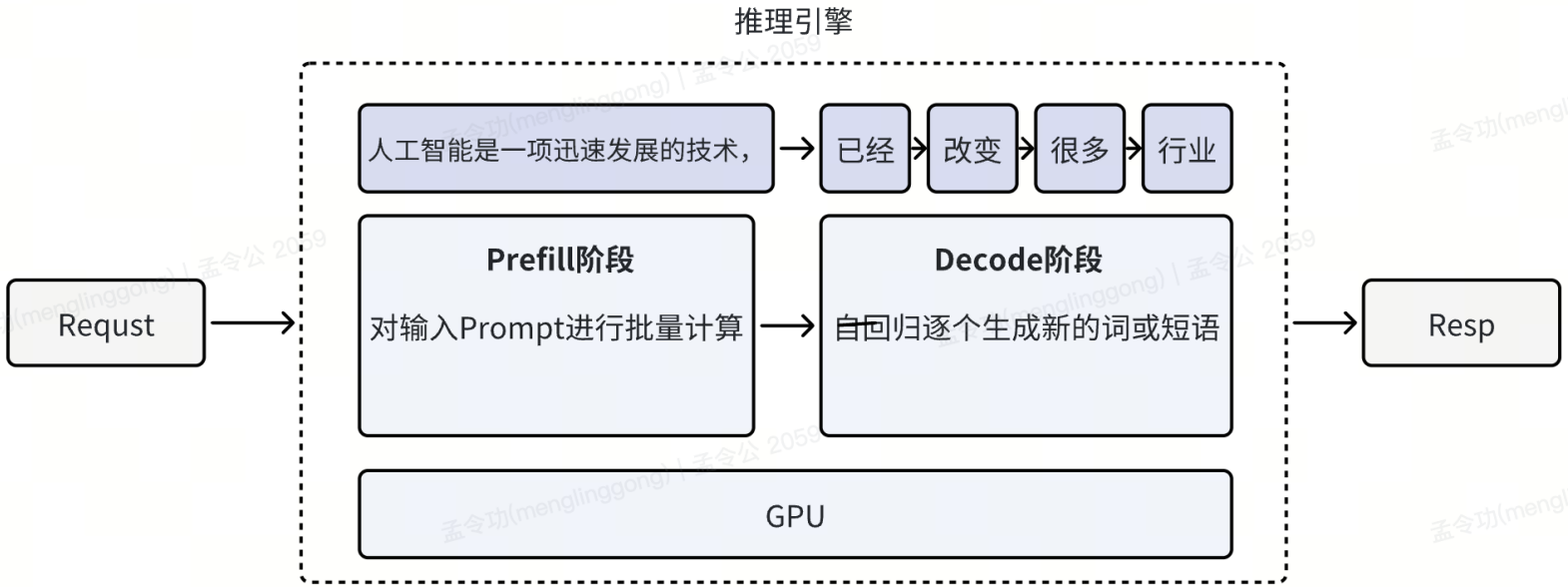
某个请求耗时过长

在将大模型应用于生产环境时，我们有时会遇到一种奇怪的现象：

某个请求的响应时间（RT）异常长，甚至出现卡顿，而系统的平均响应时间却依然正常。这是什么原因导致的呢？又如何解决呢？

请求分块处理，避免单个请求卡顿—Chunked Prefill

Prefill与D



假设我们输入了一个包含 1000 个 token 的 prompt，并希望模型生成 100 个 token 的响应。

1.Prefill 阶段

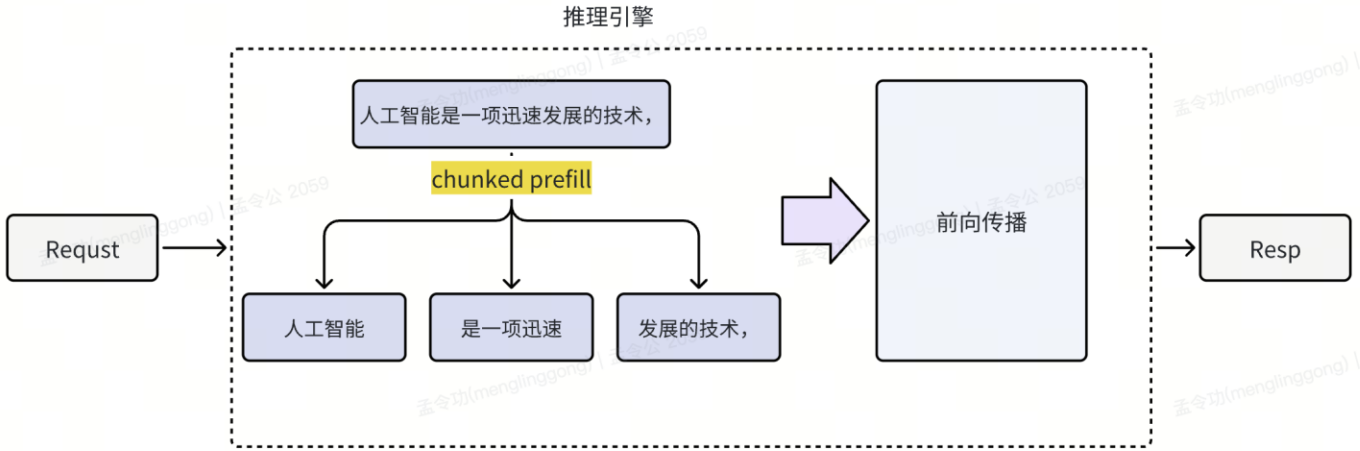
系统首先对这 1000 个 token 进行并行推理，这一步骤可以充分利用 GPU 的并行计算能力。

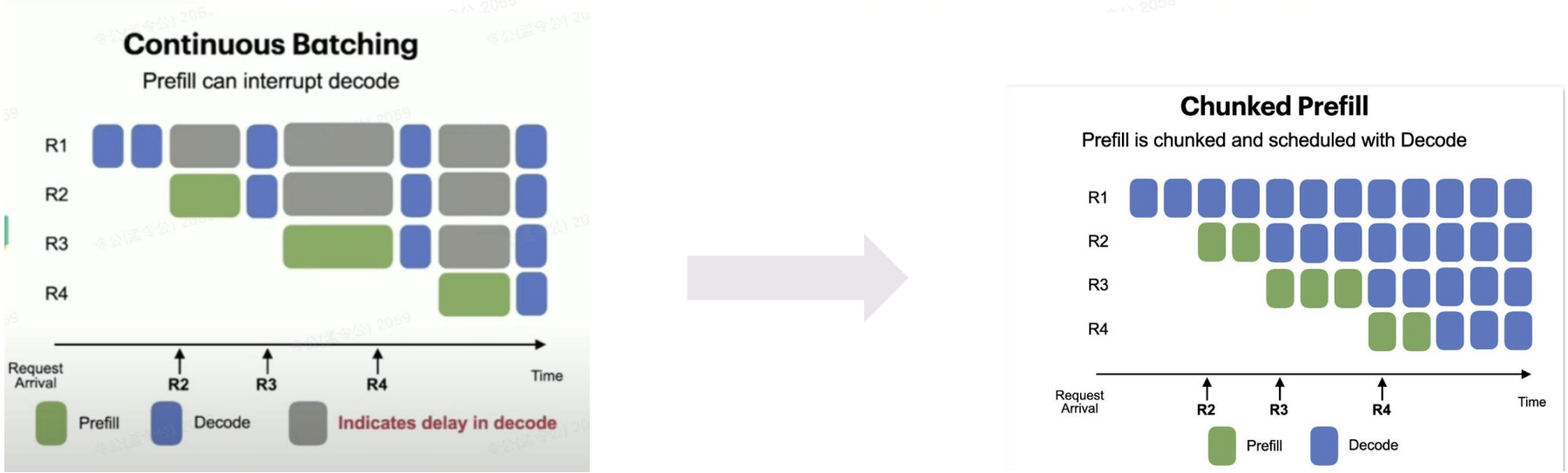
2.Decode 阶段

随后，系统会逐个生成后续的 100 个 token。由于每个新生成的 token 都依赖于之前的输出，因此这一阶段必须按顺序逐个生成。

请求分块处理，避免单个请求卡顿—Chunked Prefill

概念



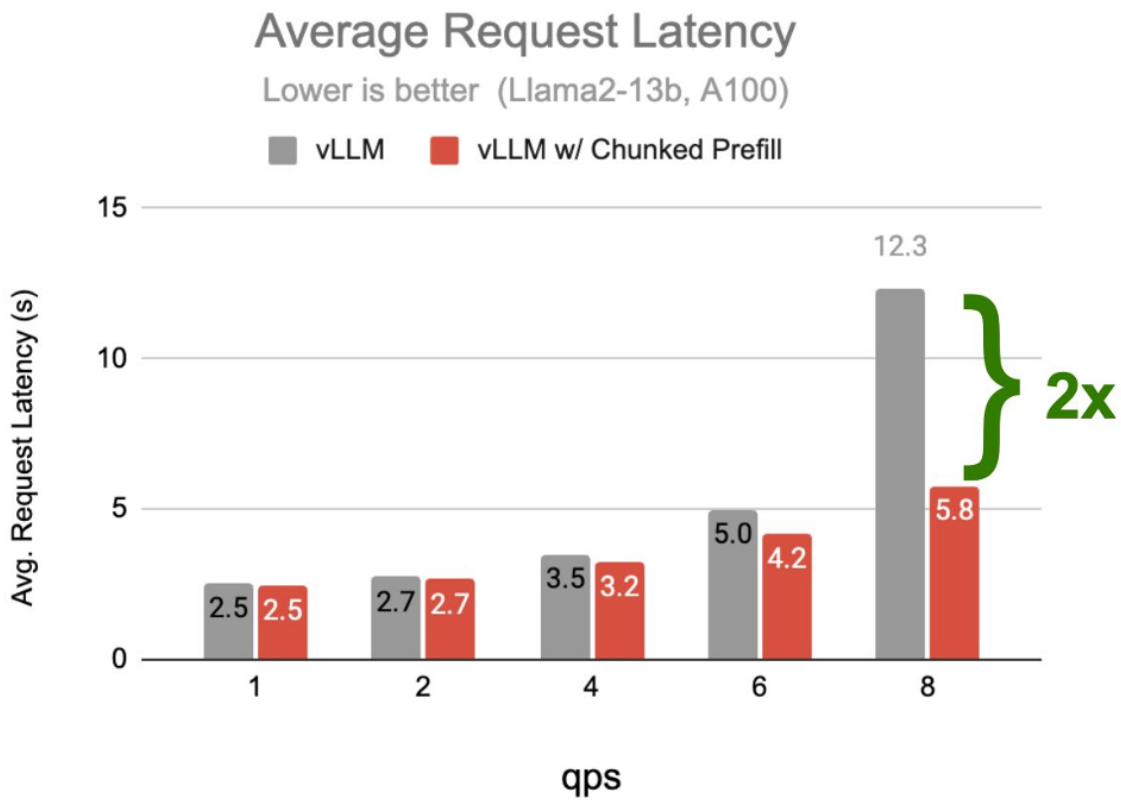


图片来自 Taming Throughput-Latency Tradeoff in LLM Inference with Sarathi-Serve vLLM @ Fourth Meetup (Public)

请求分块处理，避免单个请求卡顿—Chunked Prefill

性能提升

开启chunked prefill后，在高并发QPS下，平均RT提升了2倍。



图片来自 Taming Throughput-Latency Tradeoff in LLM Inference with Sarathi-Serve vLLM @ Fourth Meetup (Public)

使用多卡推理，推理速度翻倍

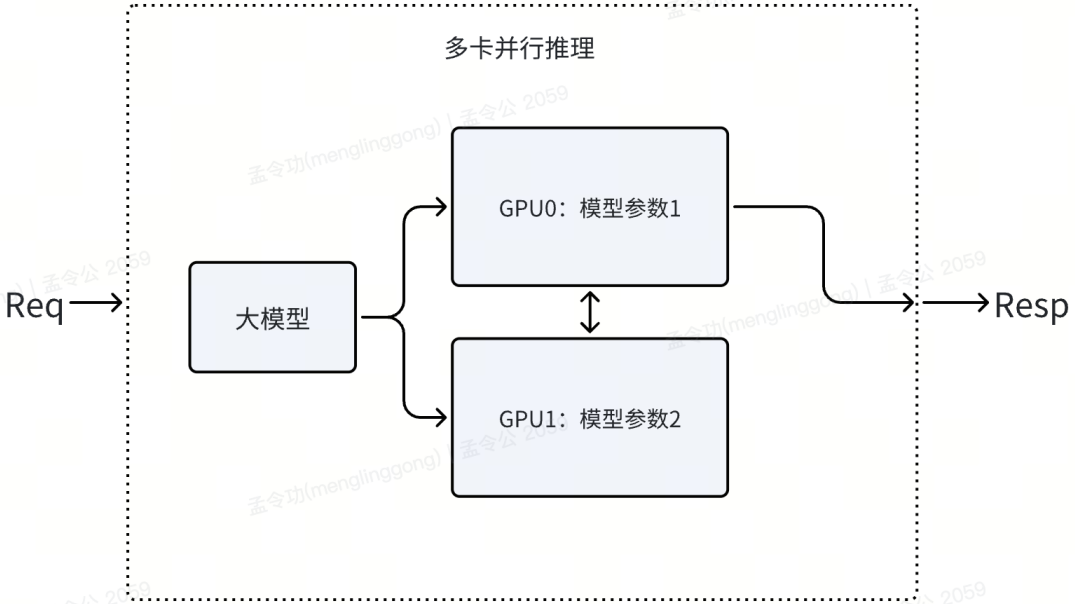
性能对比

场景	RT	QPS
单卡大模型推理	3s	1.2
双卡大模型推理	1.7s	2.4

推理中单卡变双卡可以显著提升大模型推理速度与QPS。
那为什么多卡可以提升大模型的推理速度呢?
主要原因多卡推理的优化是通过 张量并行(tensor parallelism)实现的。

使用多卡推理，推理速度翻倍

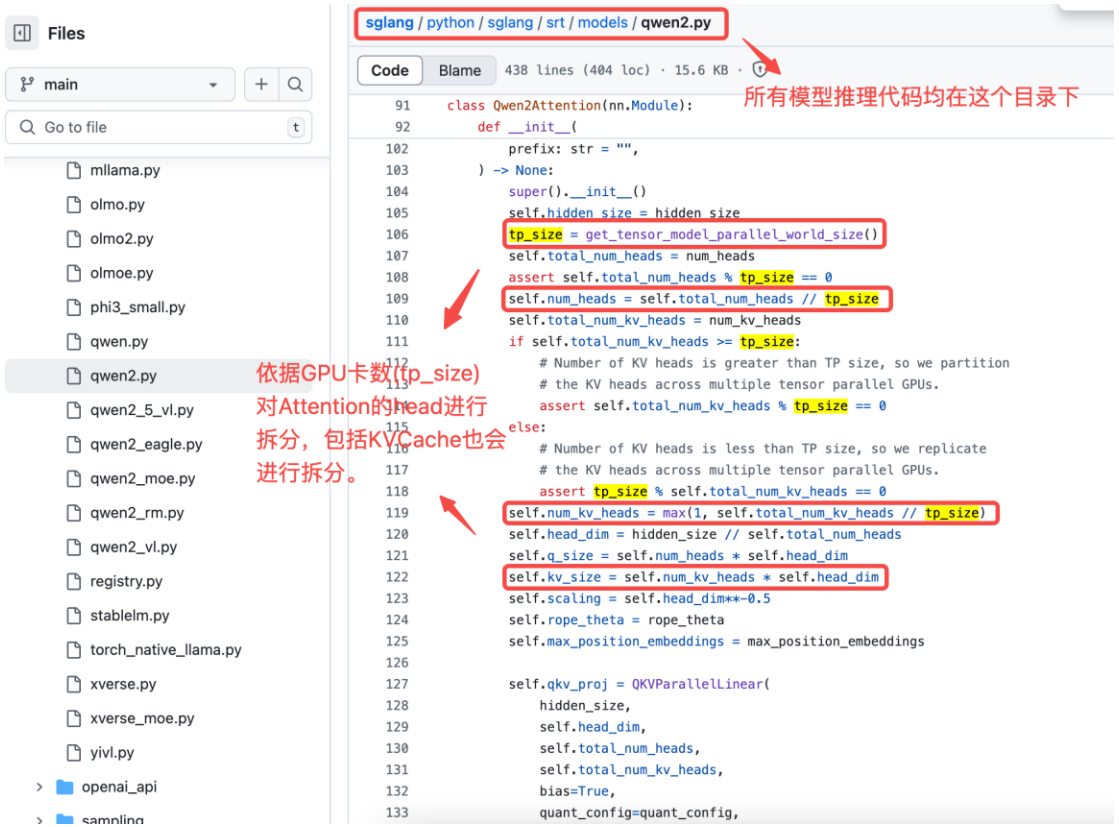
张量并行



假设你将 tensor parallel 设置为 2，意味着使用两张 GPU 来加速推理。在模型加载时，推理引擎会将大模型的 attention 参数的数量分为两组，分别加载到每张 GPU 上。然后，在推理过程中，两个 GPU 会并行计算注意力，最后再将结果聚合合并。

使用多卡推理，推理速度翻倍

张量并行-show code



1. 这个是qwen2大模型的加载逻辑，
2. 对于多头注意力机制，会依据GPU卡数对heads进行拆分。
3. 对应的KVCache也会基于GPU卡数进行拆分。

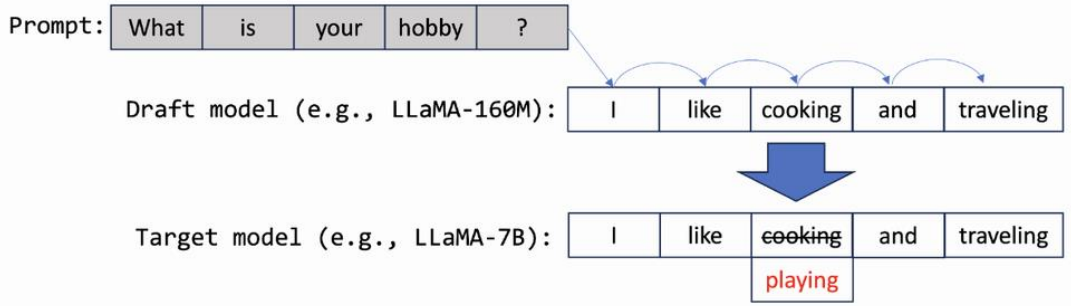
小模型推理+大模型验证 — 预测解码

工作原理

预测解码的加速技术备受关注，它能够在特定条件下显著提升大型模型（如72B大模型）的推理速度。

以下是该过程的工作原理：

1. **Draft Model**：一个更小、更高效的模型逐个提出代币。
2. **Target Model Verification**：较大的模型在单个前向传递中验证这些标记。它确认正确的令牌并更正任何不正确的令牌。
3. **一次传递多个 Token**：该方法不是每次传递生成一个 Token，而是同时处理多个 Token，从而减少延迟。



如上图所示，模型草案提出了五个标记：["I"， "like"， "cooking"， "and"， "traveling"]。然后将这些数据转发到目标模型进行并行验证。在这个例子中，第三个标记 "cooking" （应该是 "playing"） 被错误地提出。因此，此步骤中仅生成前三个标记 ["I"， "like"， "playing"]。

小模型推理+大模型验证 — 预测解码

实验效果

训练：一个7B的小模型

预测解码：70B大模型+7B小模型

对比效果

推理方法	RT
70B大模型	11S
70B大模型+7B小模型 预测解码	2.8S

小模型推理+大模型验证 — 预测解码

show code

使用大模型+小模型的方式

```python
from vllm import LLM

llm = LLM(
    model="meta-llama/Meta-Llama-3.1-70B-Instruct",
    tensor_parallel_size=4,
    speculative_model="ibm-fms/llama3-70b-accelerator",
    speculative_draft_tensor_parallel_size=1,
)
outputs = llm.generate("The future of AI is")

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

大模型

小模型

使用n-gram的方式

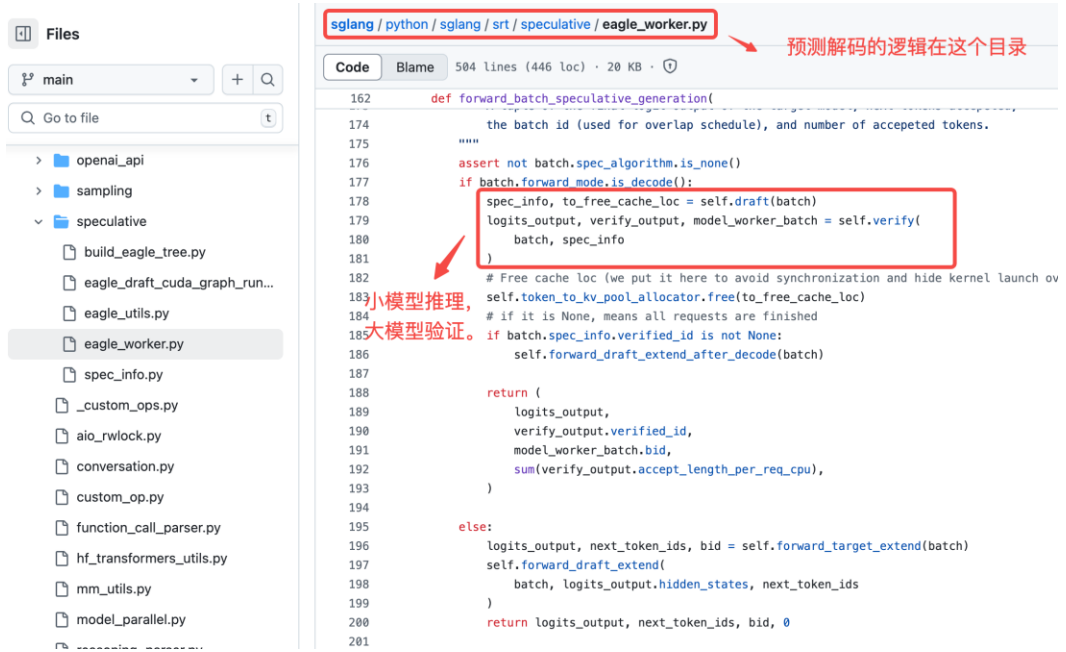
```python
from vllm import LLM

llm = LLM(
    model="facebook/opt-6.7b",
    speculative_model="[ngram]",
    num_speculative_tokens=5,
    ngram_prompt_lookup_max=4,
    ngram_prompt_lookup_min=1,
)
outputs = llm.generate("The future of AI is")

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

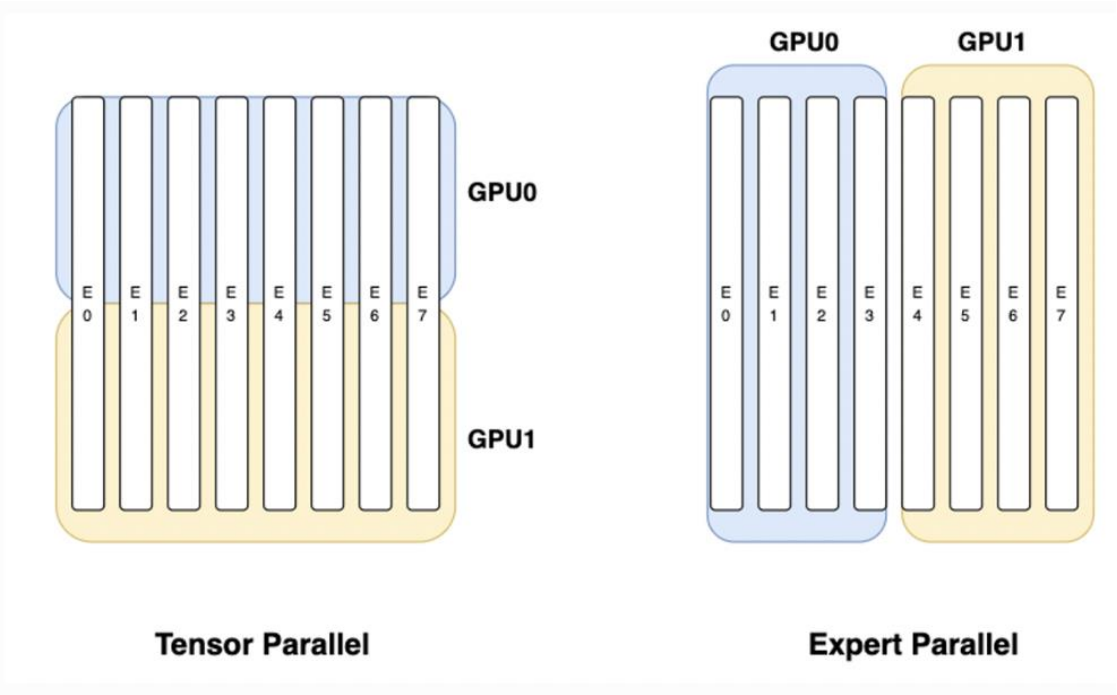
ngram检索及其配置

sglang预测解码相关代码



DeepSeek：专家并行 VS Tensor并行

工作原理

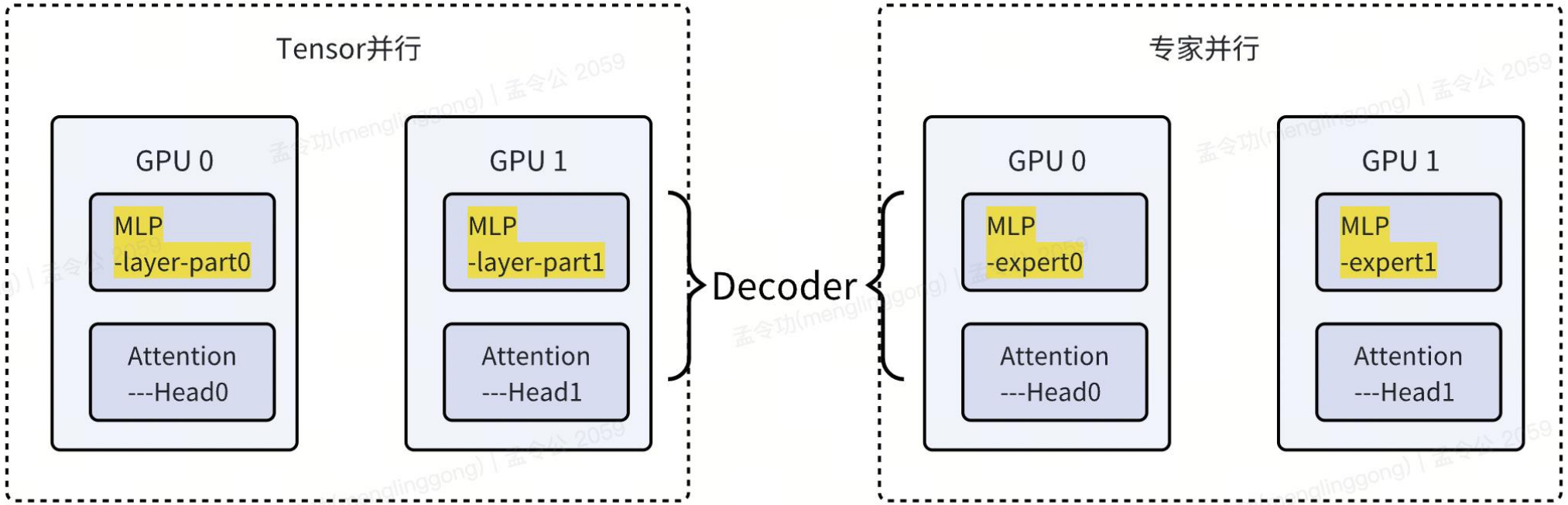


来自：https://nvidia.github.io/TensorRT-LLM/advanced/expert-parallelism.html

Tensor并行将每个专家的权重平均分配到不同的GPU上，每个GPU持有所有专家的部分权重；
专家并行则将部分专家的完整权重分配给不同的GPU，每个GPU仅持有特定专家的完整权重。

DeepSeek：专家并行 VS Tensor并行

工作原理



对于MOE结构的大模型来说，Tensor并行，每个token都需要经过所有GPU的计算，最后在聚合。
但是专家并行只需要部分GPU计算，可以避免大量的GPU聚合通信。

DeepSeek：专家并行 VS Tensor并行

效果

Prefill phase

A Layer Time(ms)	Attention	NCCL0	MoE	Other	NCCL1
Base(TP)	7.14	0.85	3.78	1.2	0.83
ThisPR(TP+EP)	7.14	0.85	0.70~1.3	1.2	1.0(avg)

来自：https://github.com/sgl-project/sglang/pull/2203

Decode phase

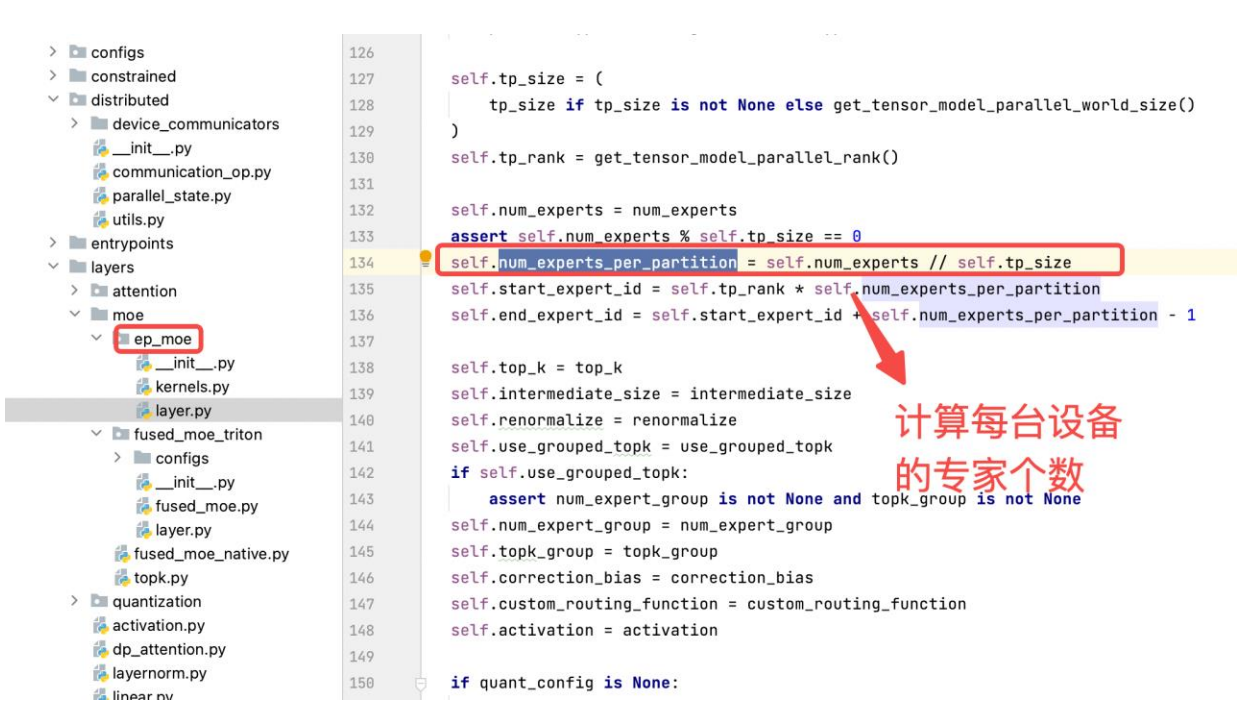
A Layer Time(ms)	Attention	NCCL0	MoE	Other	NCCL1
Base(TP)	2.5	0.24	0.98	0.4	0.24
ThisPR(TP+EP)	2.5	0.24	0.2~0.9	0.4	0.23~0.9

来自：https://github.com/sgl-project/sglang/pull/2203

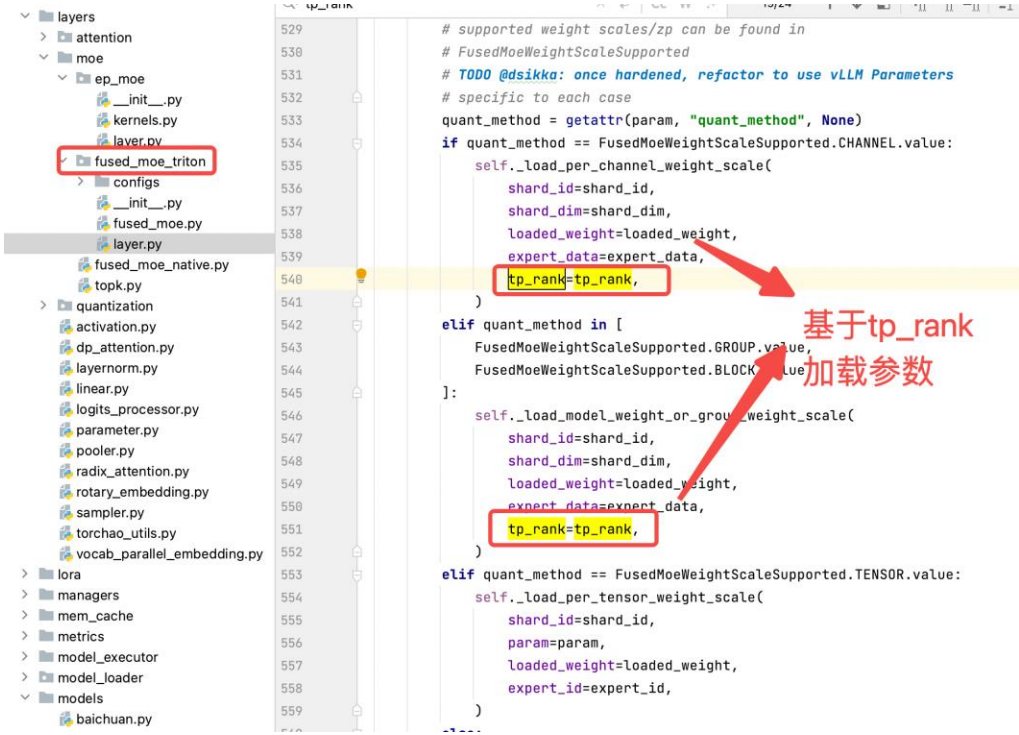
DeepSeek：专家并行 VS Tensor并行

Show Code

MOE：专家并行



MOE：Tensor并行



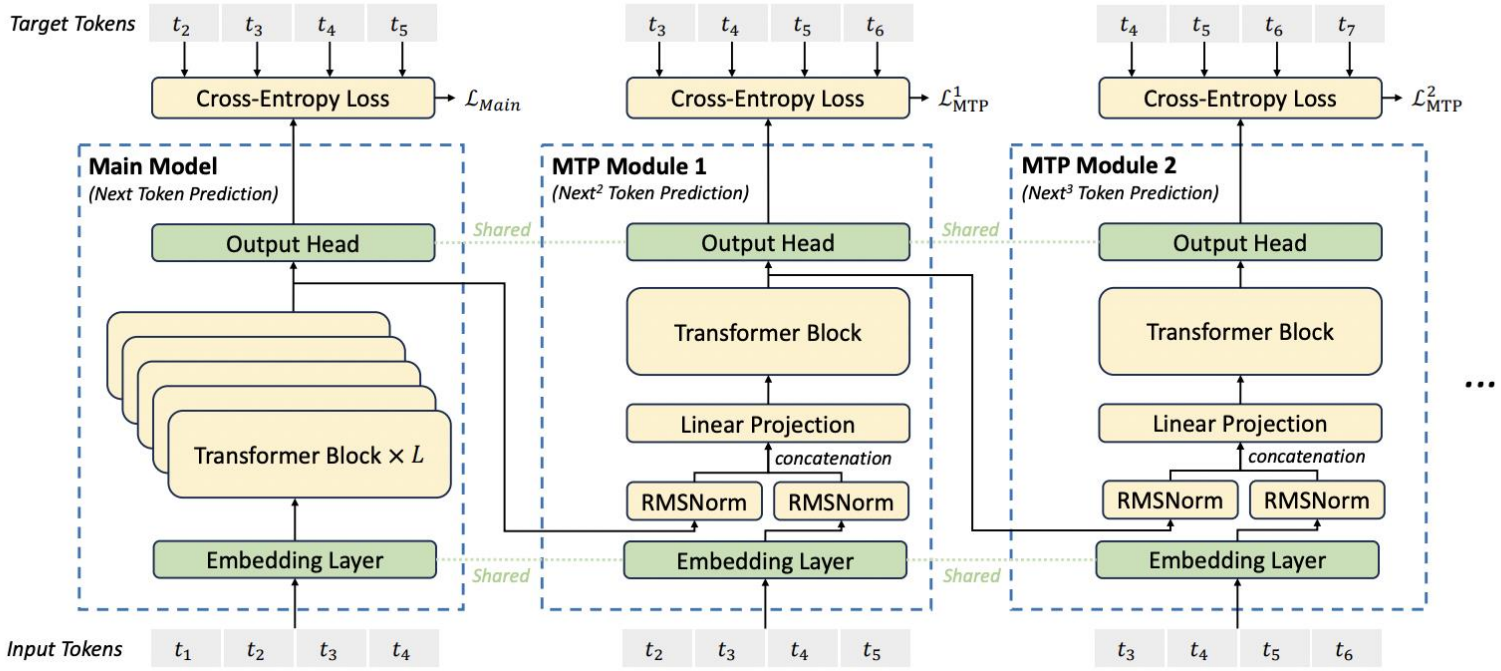
DeepSeek：MTP与推测解码

工作原理

通过优化解码阶段，将1-token生成转为multi-token生成，在训练时一次生成多个token，在推理时一次预测多个token，实现推理加速。

DeepSeek：MTP与推测解码

MTP结构



来自：https://arxiv.org/pdf/2412.19437

DeepSeek：MTP与推测解码

Show Code

DeepSeek-NextN模型

lmsys / DeepSeek-R1-NextN like 1 Follow Large Model Systems O.

Safetensors | deepseek_v3 | custom_code | fp8

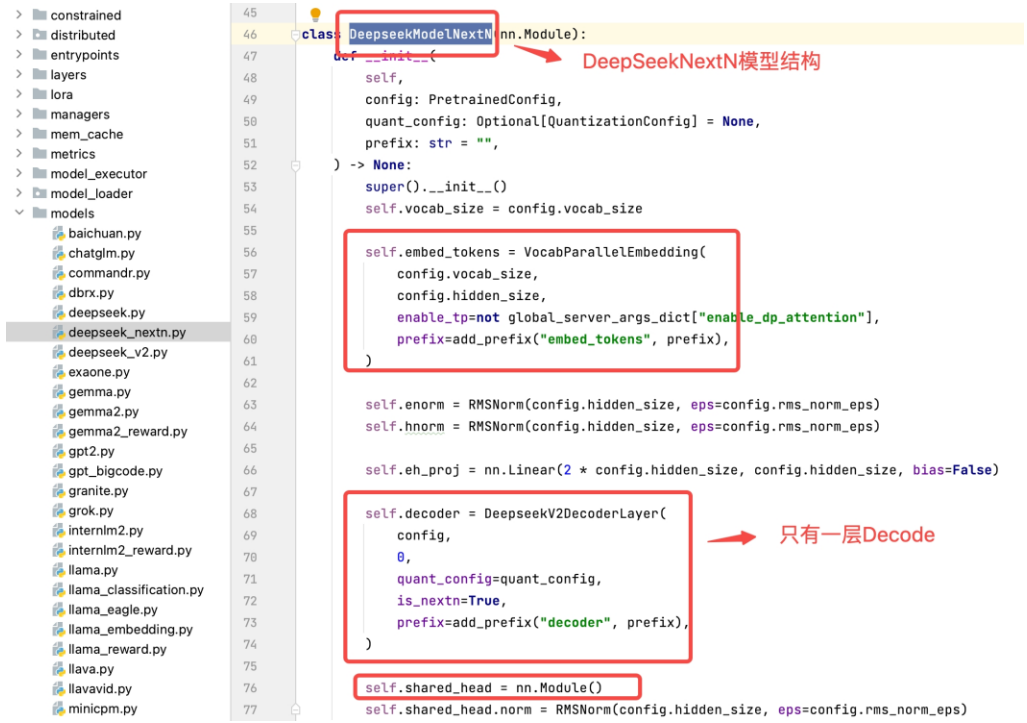
Model card | Files and versions | Community 1

main | DeepSeek-R1-NextN

ispobock Create README.md 972fa49 VERIFIED

.gitattributes Safe	1.59 kB	
README.md	444 Bytes	
config.json	3.1 kB	
configuration_deepseek.py Safe	10.6 kB	
model.safetensors.index.json	157 kB	
nextn_layer_parameters.safetensors	11.7 GB LFS	
tokenizer.json Safe	7.85 MB	
tokenizer_config.json	3.59 kB	

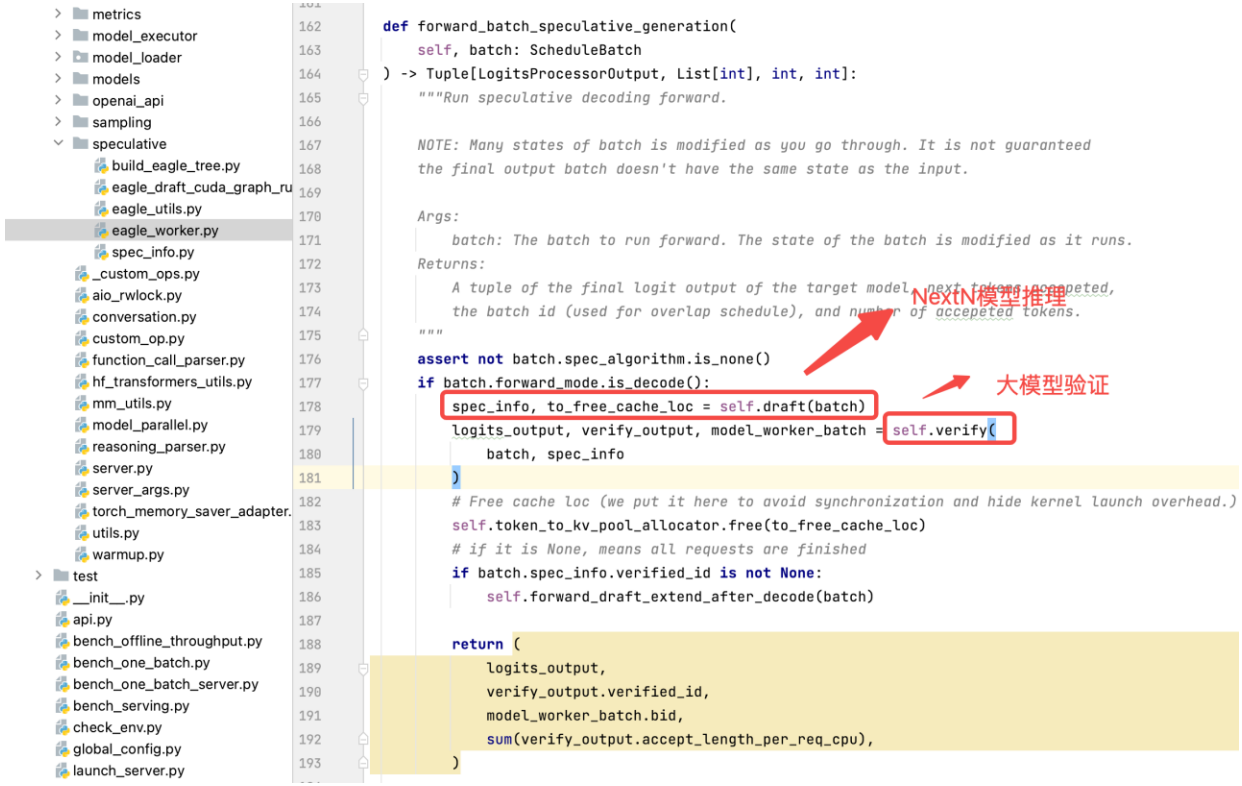
DeepSeek-NextN Model Code



DeepSeek：MTP与推测解码

Show Code

SGLang预测解码




```
def forward_batch_speculative_generation(
    self, batch: ScheduleBatch
) -> Tuple[LogitsProcessorOutput, List[int], int, int]:
    """Run speculative decoding forward.

    NOTE: Many states of batch is modified as you go through. It is not guaranteed
    the final output batch doesn't have the same state as the input.

    Args:
        batch: The batch to run forward. The state of the batch is modified as it runs.
    Returns:
        A tuple of the final logit output of the target model, next tokens accepted,
        the batch id (used for overlap schedule), and number of accepted tokens.
    """
    assert not batch.spec_algorithm.is_none()
    if batch.forward_mode.is_decode():
        spec_info, to_free_cache_loc = self.draft(batch)
        logits_output, verify_output, model_worker_batch = self.verify(
            batch, spec_info
        )
        # Free cache loc (we put it here to avoid synchronization and hide kernel launch overhead.)
        self.token_to_kv_pool_allocator.free(to_free_cache_loc)
        # if it is None, means all requests are finished
        if batch.spec_info.verified_id is not None:
            self.forward_draft_extend_after_decode(batch)

        return (
            logits_output,
            verify_output.verified_id,
            model_worker_batch.bid,
            sum(verify_output.accept_length_per_req_cpu),
        )
```

命令配置

```
python3 -m sglang.launch_server
--model deepseek-ai/DeepSeek-R1
--speculative-algo EAGLE
--speculative-draft lmsys/DeepSeek-R1-NextN
--speculative-num-steps 2
--speculative-eagle-topk 4
--speculative-num-draft-tokens 4
--trust-remote
--tp 8
```

DeepSeek：MTP与推测解码

性能提升

Config	并发	首token延迟(输入1k/2k)	平均tokens/s
Triton-backend	1	354ms/412ms	23.85
MTP: NextN	1	298ms/393ms	47.14

开启了MTP+预测解码后，在并发1的情况下，吞吐提升了接近1倍。

DeepSeek：单机部署与双机部署

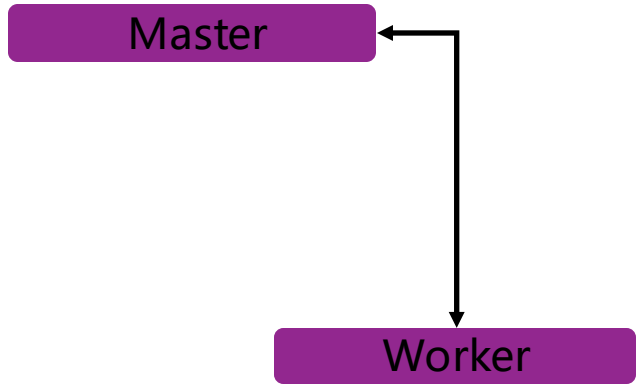
单机部署

硬件配置	8*H20*96G
启动命名	python3 -m sglang.launch_server --tp 8 --enable-p2p-check --host 0.0.0.0 --port 5000 --trust-remote-code --model-path /opt/deepseek/models/download/DeepSeek-V3 --max-running-requests 128 --context-length 32768 --mem-fraction-static 0.9 --enable-torch-compile
上下文长度	32K
最大tokens (10并发)	270 tokens/s

DeepSeek：单机部署与双机部署

双机部署

硬件配置	2*8*A100*80G
上下文长度	128K
最大tokens (16并发)	210 tokens/s
最大tokens (16并发) (IB网络)	280 tokens/s



IB网络，节点间双卡带宽可以达到12GB/s

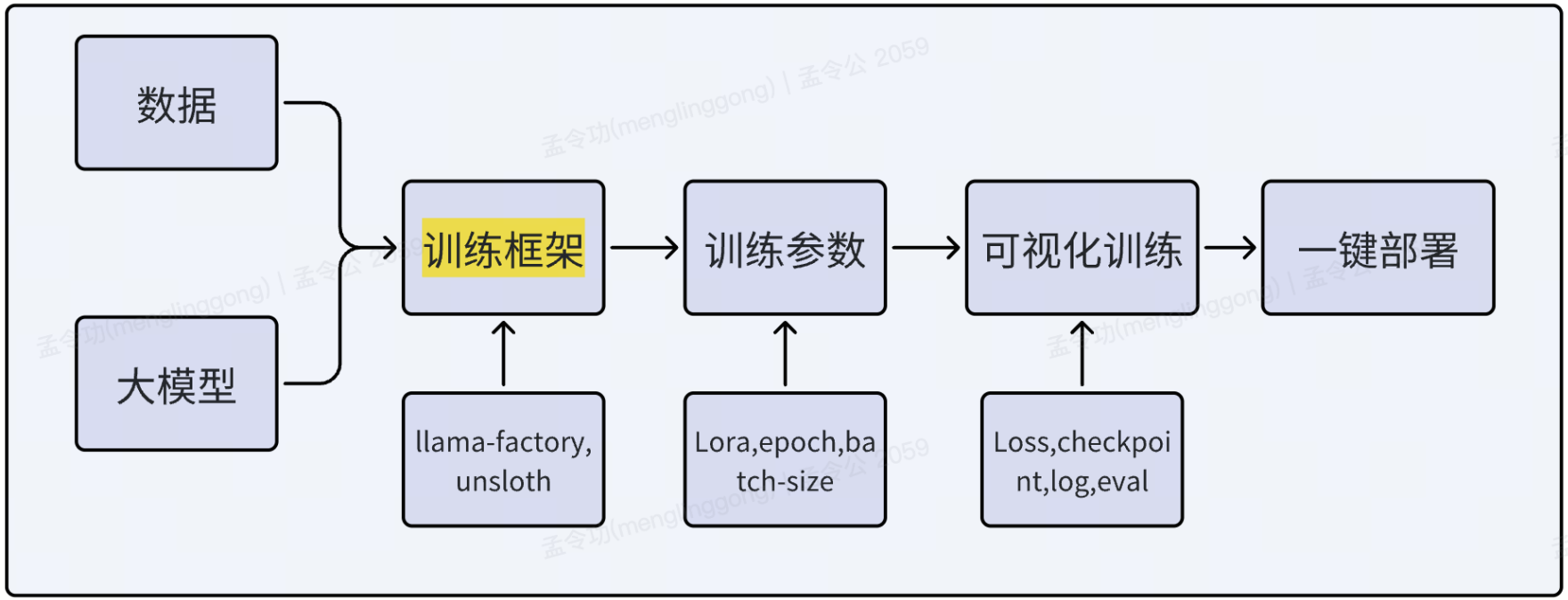
大模型训练平台

一键发起微调训练与推理部署

- 选择大模型，基于之前提到的大模型选择原则，在大型模型平台上选择您需要的大模型。
- 上传训练数据，按照上述数据准备方法，将您准备好的数据上传到大型模型平台。
- 配置训练参数，通常情况下，选择默认配置参数，如Lora，即可。这些参数通常经过优化以获得最佳的训练效果可。
- 训练，点击相应按钮，启动训练过程。大型模型平台将自动处理训练任务，以便您专注于业务应用的开发和部署。
- 一键部署，训练后，业务方可以点击按钮，一键部署，发布到生产环境。
- 效果体验，部署后，业务方可以快速训练体验效果。

大模型训练平台

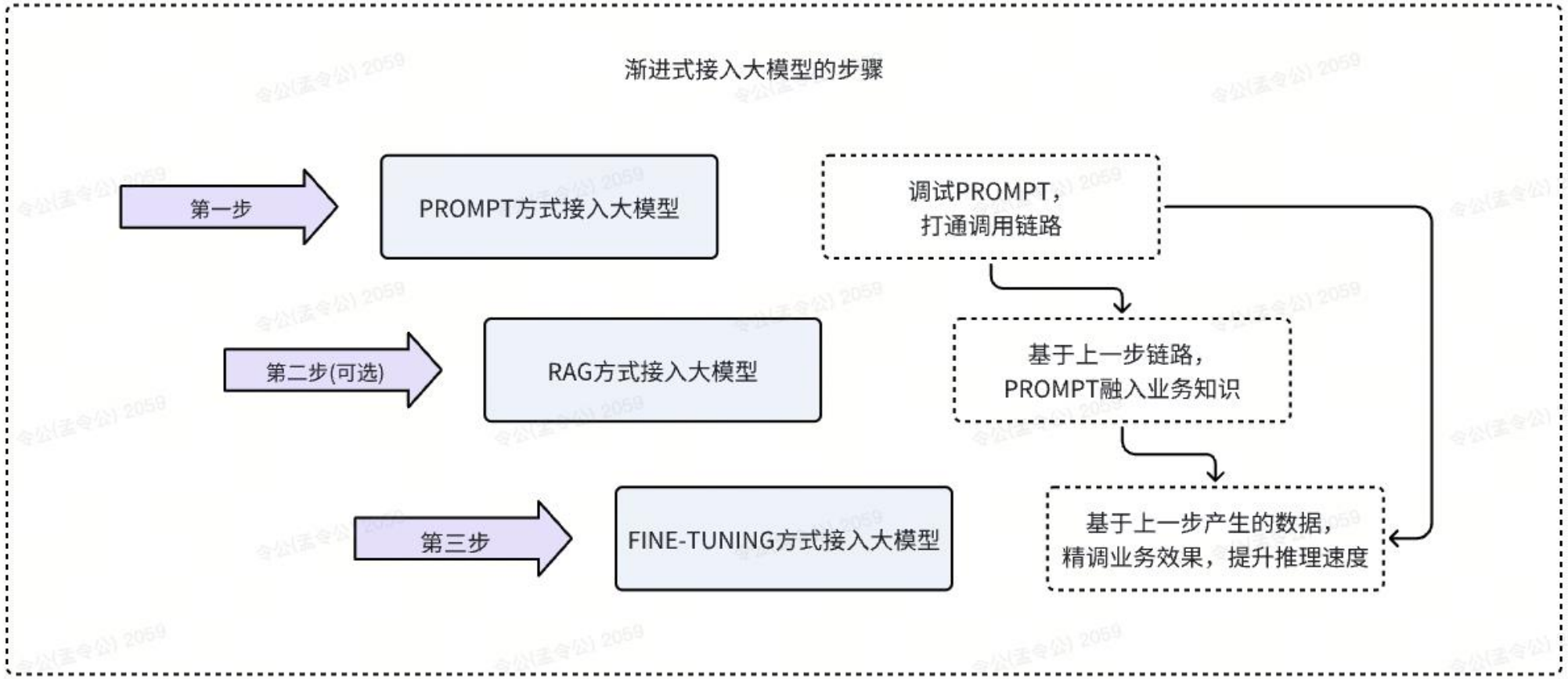
兼容任意训练框架



- 用户上传数据与大模型。
- 用户选择训练框架，比如llama-factory或unsloth。
- 配置训练参数，即可开始训练。
- 主要解决不同框架支持不同的训练方法的问题。

大模型训练平台

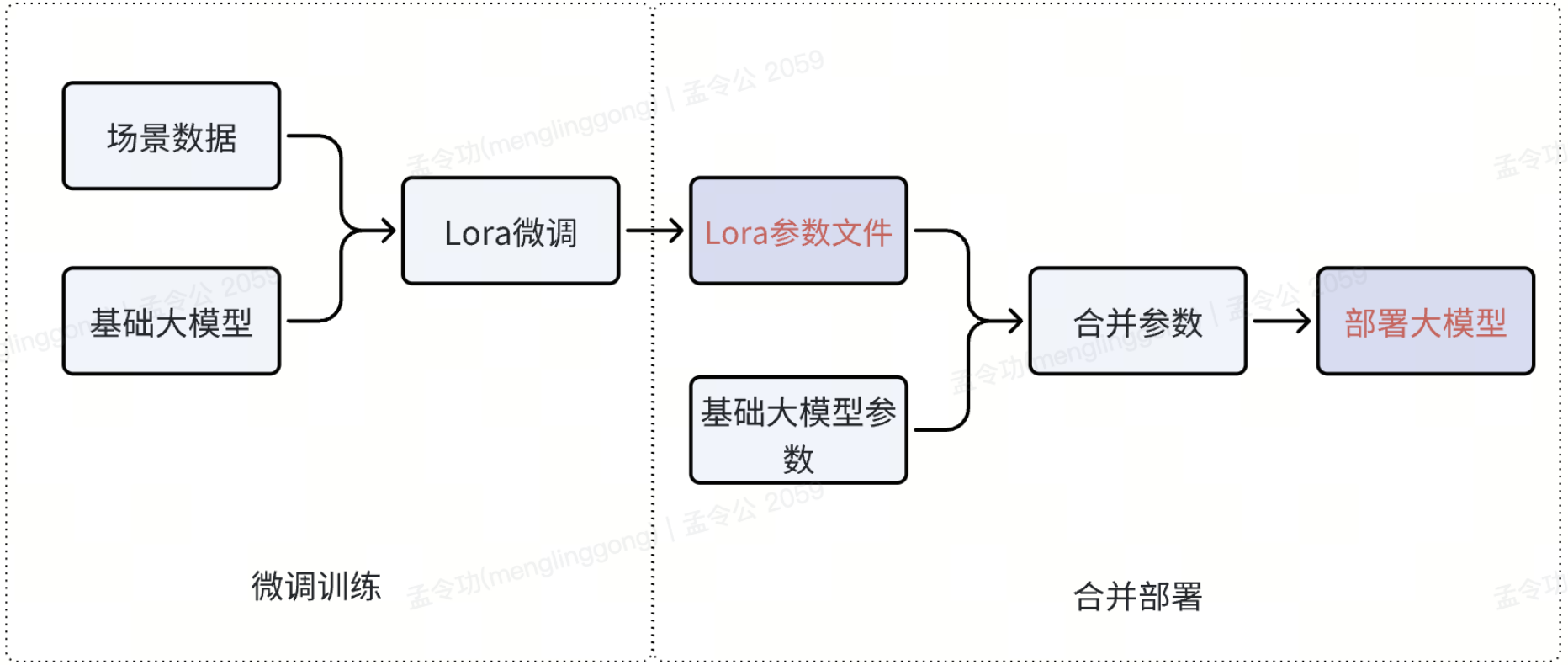
业务方训练数据如何来



大模型训练平台

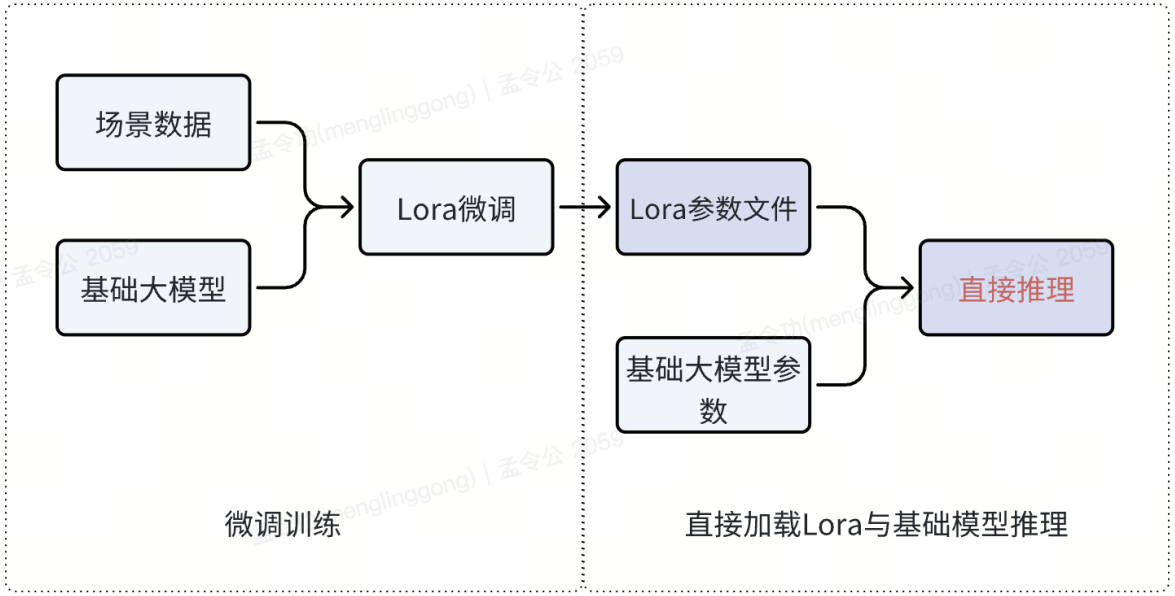
利用多Lora方式部署

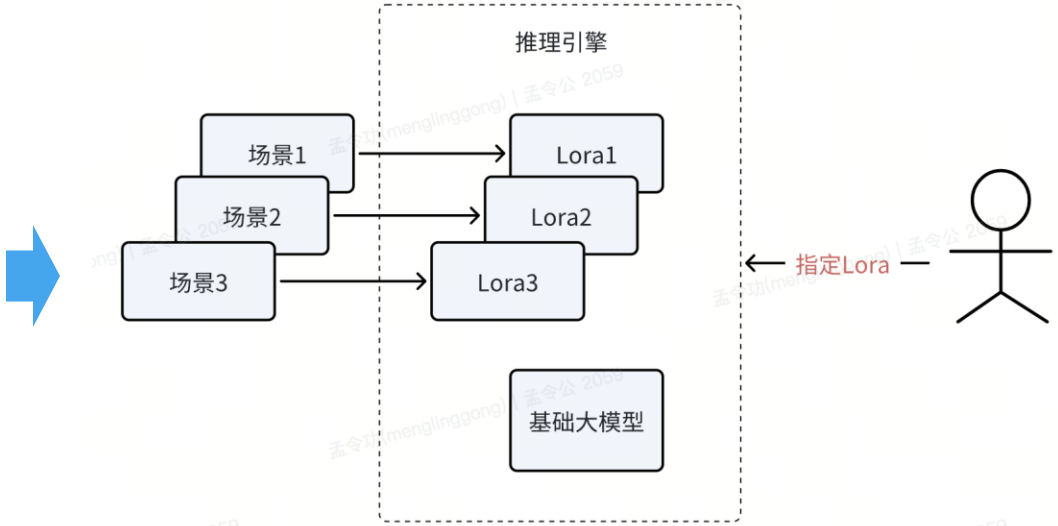
传统部署方式



大模型训练平台

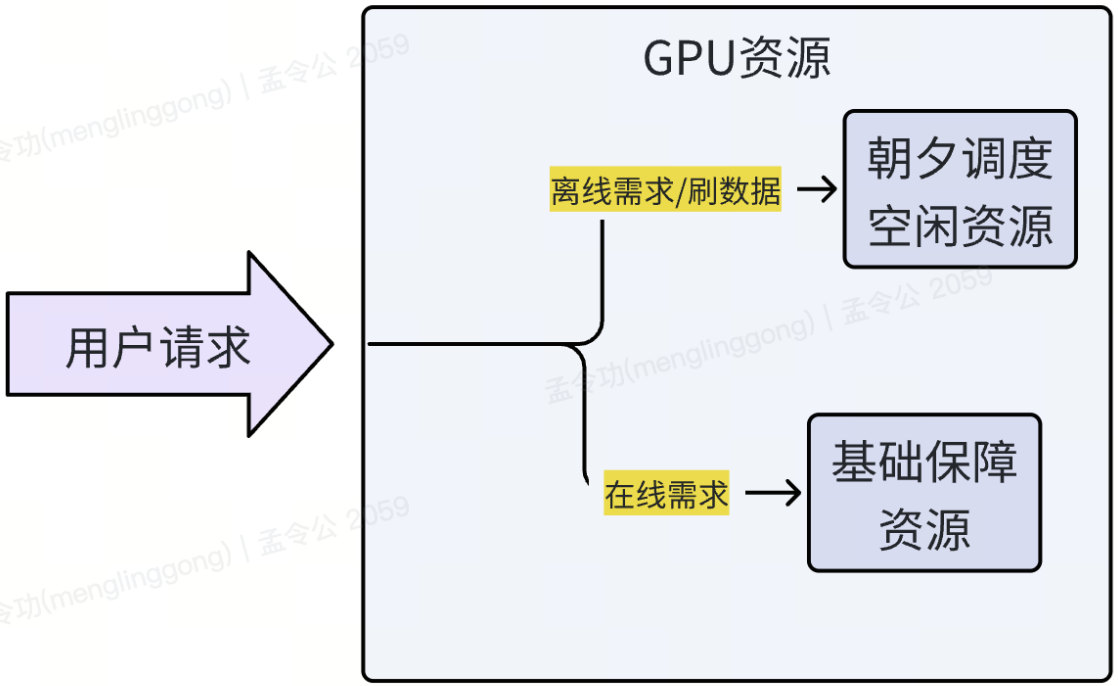
利用多Lora方式部署





大模型训练平台

利用空闲资源部署通用大模型



- 在线需求由基础保障资源提供
- 离线或刷数据需求，可以由朝夕调度空闲资源提供支持。

总结与展望

- **大模型推理加速技**巧：介绍了Paged Attention、Radix Attention、chunked prefill、多卡并行等方法。
- **验证与操作**：展示了相关加速方法的验证结果与操作步骤。
- **Deepseek-r1部署方**法：分享了最近火爆的Deepseek-r1大模型高效部署方法，并鼓励大家尝试。
- **优化思**路：重点关注大模型推理优化背后的思路。

AiDD 6th 2025

参与调研您将优先获得

AiDD定制版
《AI+软件研发精选案例》

专属学习顾问
1对1需求对接

AiDD会后小调研

AiDD峰会致力于协助企业利用AI技术深化计算机对现实世界的理解，推动研发进入智能化和数字化的新时代。作为峰会的重要共建者，您的真知灼见对我们至关重要。衷心感谢您的参与支持！

扫码参与调研

2025 AI+研发数字峰会

拥抱AI 重塑研发

科技生态圈峰会 + 深度研习

——1000 + 技术团队的共同选择

K+峰会

K+峰会 敦煌站
K+ 思考周®研习社
时间：2025.08.29-30

K+峰会 上海站
K+ 金融专场
时间：2025.09.26-27

K+峰会 香港站
K+ 思考周®研习社
时间：2025.11.17-18

K+峰会详情

AiDD峰会

AiDD峰会 上海站
AI+研发数字峰会
时间：2025.05.23-24

AiDD峰会 北京站
AI+研发数字峰会
时间：2025.08.08-09

AiDD峰会 深圳站
AI+研发数字峰会
时间：2025.11.14-15

AiDD峰会详情

AiDD 6th 2025

2025 AI+研发数字峰会

AI+ Development Digital Summit

感谢聆听！

扫码领取会议PPT资料