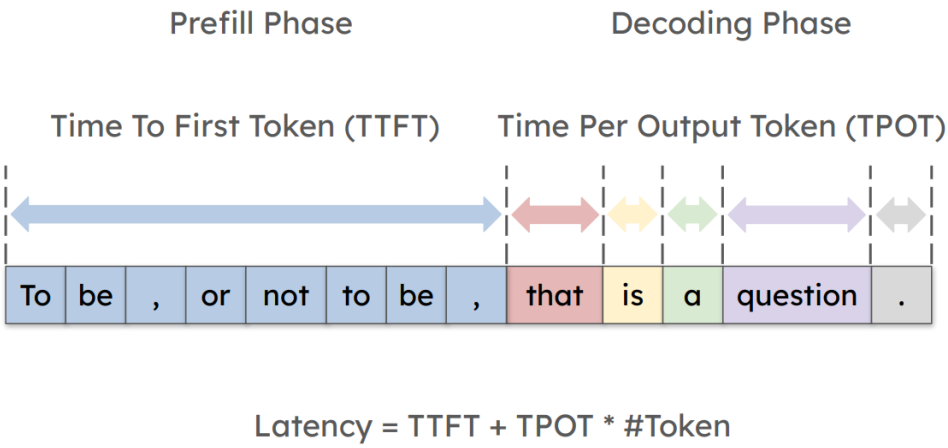
DistServe：预填充和解码解耦

LLM 推理分为两阶段：

1. **Prefill 阶段**：在这个阶段，模型并行处理用户输入的Prompt，并生成第一个输出 token。这个过程的延迟一般用 **TTFT** 来衡量。Prefill阶段是计算密集型任务，对计算资源的消耗很大。即使batch-size不大，但如果用户的输入很长，依然可能使得GPU的计算能力达到极限。
2. **Decoding 阶段**：进入这个阶段后，模型会基于已生成的 token ，逐步生成接下来的每一个token。这个阶段的延迟通常用 **TPOT** 来衡量。Decoding计算是通信密集型任务，每步只处理一个新token，只有更大的batch-size才能充分利用GPU算力，但受到GPU内存带宽的制约（因为KV-Cache的读取与存储）。

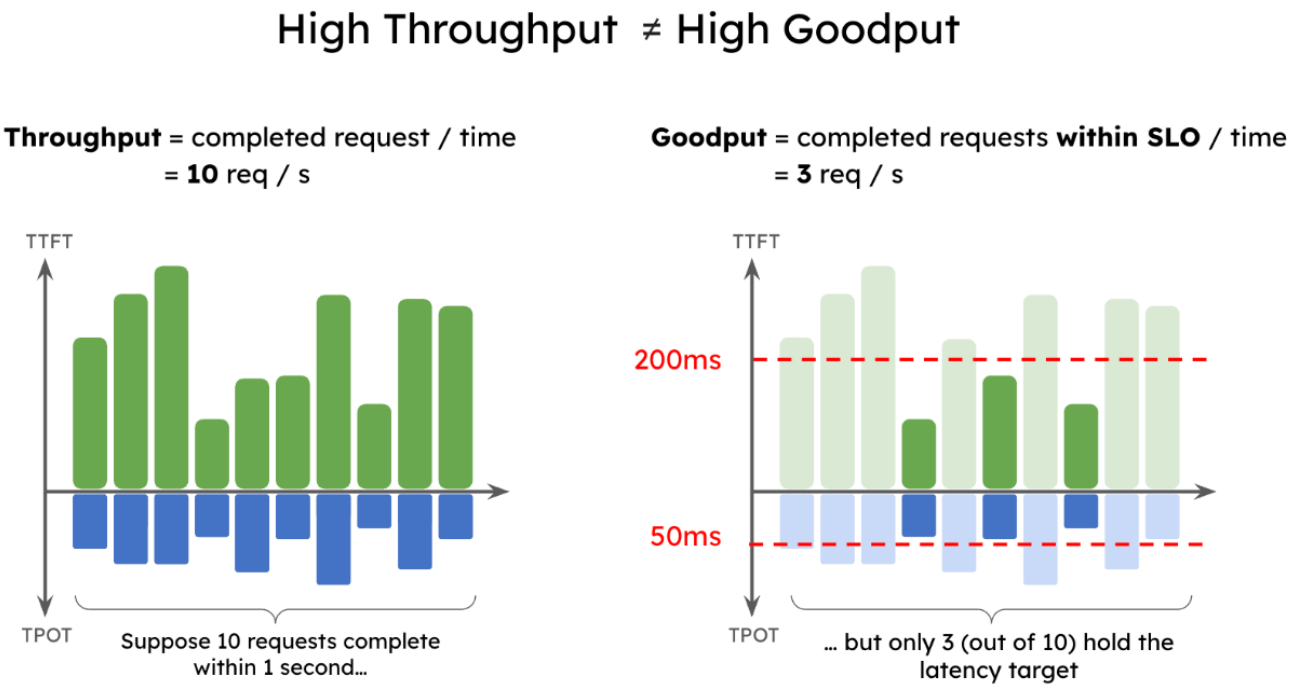
🍞 首先介绍一下Distserve这篇论文中评价LLM服务系统性能的几个指标：

- **时延（Latency）：** 时延是指用户从发起请求到收到响应所等待的时间，包括：
  - **首词元输出时间（TTFT，** time-to-first-token）：生成第一个 token 的等待时间。
  - **每词元输出时间（TPOT**，time-per-output-token）：连续生成 token 之间的平均时间间隔。
- **吞吐量（Throughput/rps）**：吞吐量常用 rps（Requests Per Second） 来衡量，指系统每秒能处理多少个用户请求。**吞吐量越高，单位成本越低**。但如果只追求吞吐量，可能会牺牲用户的实际体验。为此引入了SLO的概念：
- **服务等级目标（SLO**，Service level objective）： SLO 是针对 TTFT、TPOT 等时延指标设定的具体目标要求，比如“90% 的请求 TTFT < 200ms，TPOT < 50ms”。它用于保证大多数用户的体验不被影响。
- **有效吞吐量（Goodput）**：指在 SLO 要求下，系统每秒实际能处理的请求数量。只有同时保证高吞吐和良好体验的请求才计入 Goodput。



为什么会出现高Throughput而低Goodput?

下图举了一个例子，Throughput为10rps，但Goodput只有3 rps。

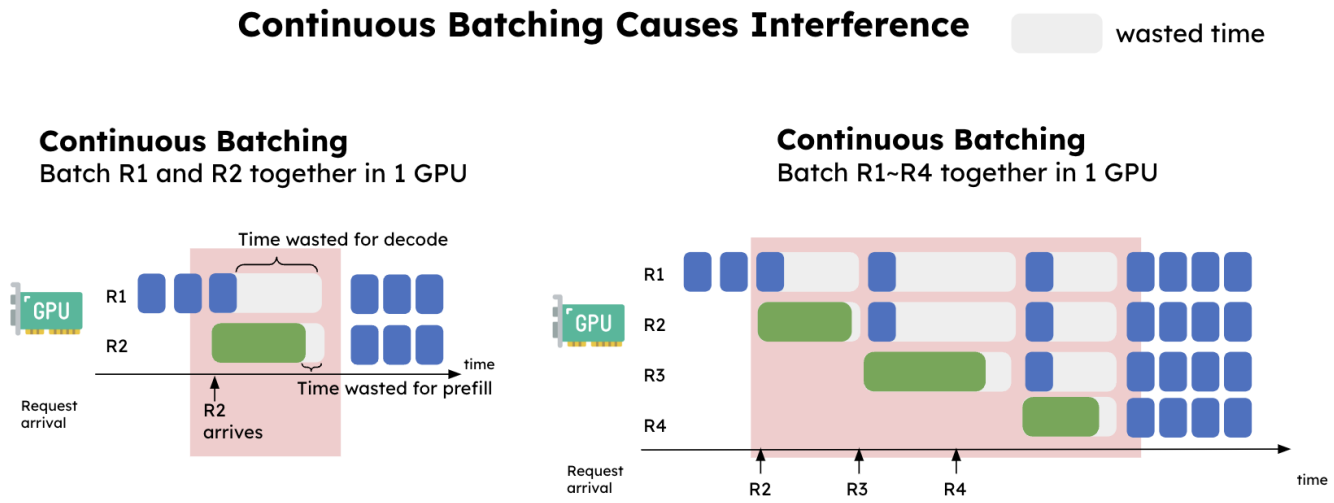


造成这一现象的原因：

1. **Continuous Batching带来的延迟累积**

为了提升整体吞吐量，现有系统通常采用continuous batching策略，对大量请求同时进行 prefill 与 decode。这种方式确实能最大化 GPU 利用率，但会导致部分请求在 decode 阶段等待过久，违反 SLO，从而不计入 goodput。如下左图中decode任务(R1)因为中途插入了prefill任务（R2），导致decode时间大幅拉长。

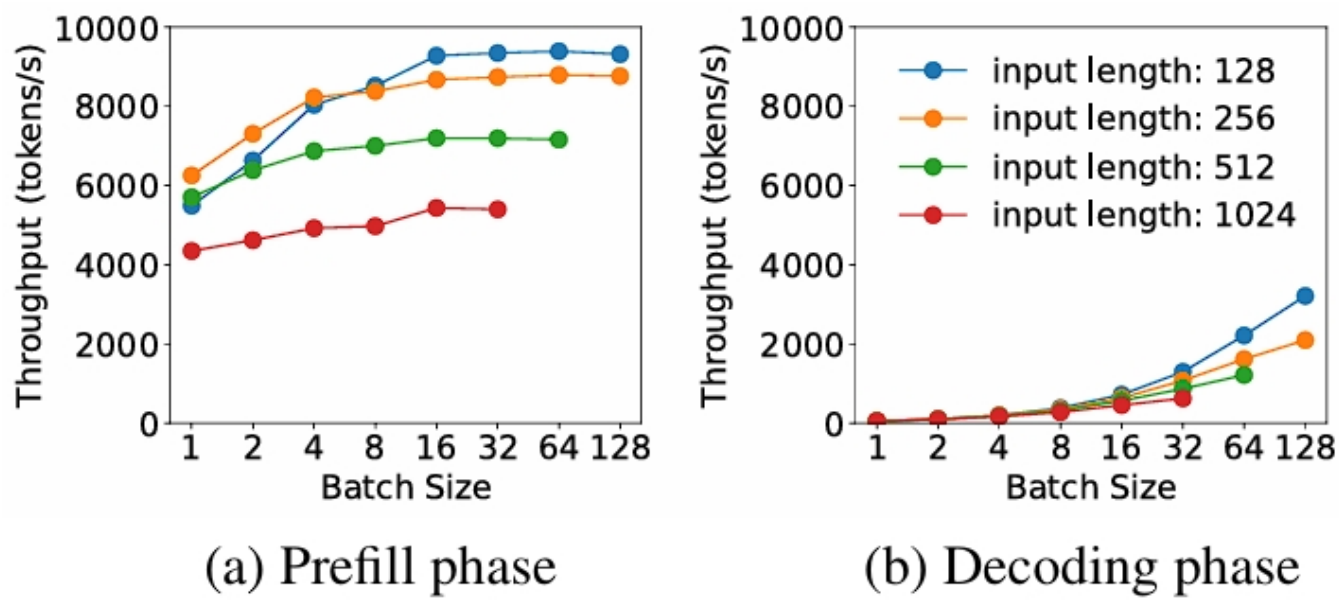
> Continuous batching为了实现并行计算，batch 内所有请求必须同步进行同一操作，哪怕其中部分请求只需 decode，也得等待 batch 内其它请求的 prefill 结束。



2. **简单拆分prefill和decode也难以解决**：decode任务可能需要等待进行中的prefill任务，专门做decode的batch的的GPU利用率也很低。
3. **计算资源和并行方式耦合，难以兼顾所有延迟需求**：因为prefill和decode部署在同一块GPU上，所以要共享计算资源和并行策略。但prefill阶段受到算力限制，为了满足TTFT要求，更需要做张量并行（TP）以缩短执行时间；decode阶段更适合大 batch 的流水线并行（PP），以提升整体 TPOT 和吞吐量。现行的推理系统中，为了同时满足两个SLO，往往需要配置过度的计算资源。

预填充和解码分析

prefill阶段是计算密集型任务，一旦待处理请求的输入长度超过某个阈值 L_m，GPU 即成为计算瓶颈，此时继续增大batch-size不仅无法提升吞吐率，反而会按比例延长每个batch的处理时间，应保持较小batch-size以避免过度排队。



(a) Prefill phase　　(b) Decoding phase

📌 **论文中对不同并行策略平均TTFT的计算：**

假设预填充采用 M/D/1 排队系统，当请求到达率为 R、单请求执行时间为 D 时，$RD < 1$，TTFT的平均值为：

$$Avg\_TTFT = D + \frac{RD^2}{2(1-RD)}$$

引入2路张量并行后，设加速系数为 $K(1 < K < 2)$，因为通信开销会浪费一定的时间，TTFT的平均值为：

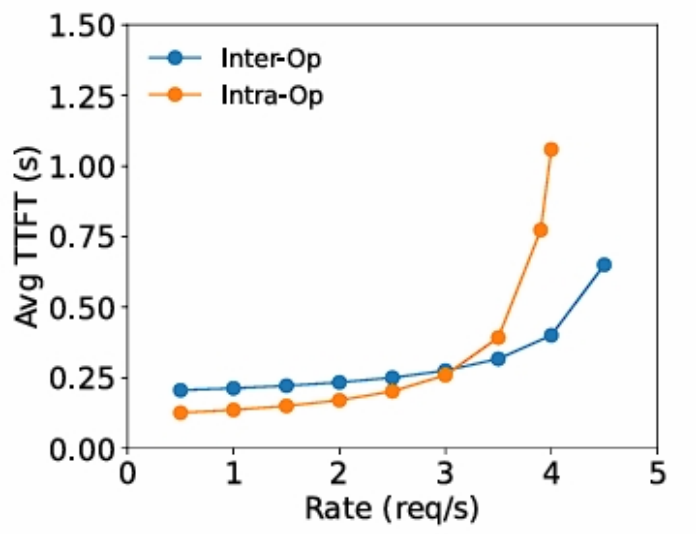
$$Avg\_TTFT = \frac{D}{K} + \frac{RD^2}{2K(K-RD)}$$

引入2路流水线并行后，忽略跨层通信的开销，设整体请求延迟为 D_s，最慢阶段耗时为 D_m，$D \approx D_s \approx 2 \times D_m$，TTFT的平均值为：

$$Avg\_TTFT \approx D + \frac{RD^2}{4(2-RD)}$$

结合以上分析可知：

- 在较低到达率时，执行时间主导延迟，张量并行（TP）更优。
- 随着到达率增高，排队延迟占比上升，流水线并行（PP）更优。
- 右图中的Inter-Op和Intra-Op类似于PP和TP。

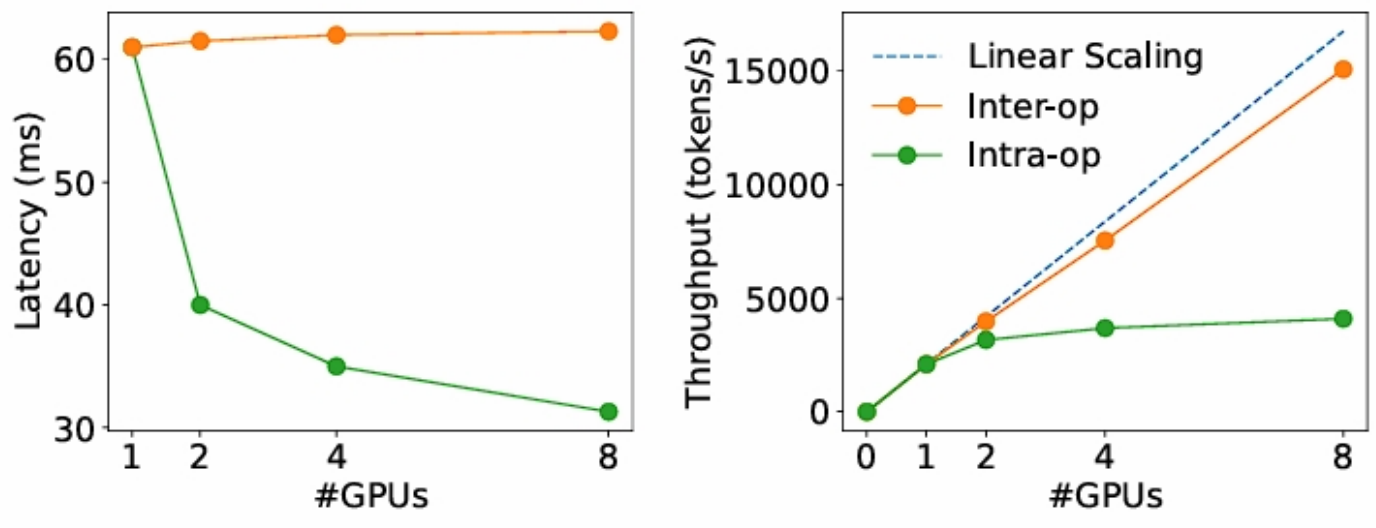


decode阶段是通信密集型任务，每步只生成一个新的token，需要频繁的读写KV cache，因此受限于内存的带宽。并且单个解码任务的GPU利用率很低，需要增大batch-size来提升吞吐量，如上上图(b)所示。

接下来讨论不同并行策略在decode阶段的影响，如下图所示。**张量并行**可以有效降低延迟，但随着GPU数量的增加，大量的通信导致收益降低。而**流水线并行**能够线性提升整体吞吐量，但不会影响单个请求的响应延迟。

在实际应用中：

- 为了满足TPOT的SLO，优先用用**张量并行**降低响应延迟。
- 通过**流水线并行**增大吞吐量。
- 在GPU显存充足的情况下，可以通过复制多个模型实例以增加吞吐量，并可等价地将到达率 R降至R/N，以减少排队延迟。

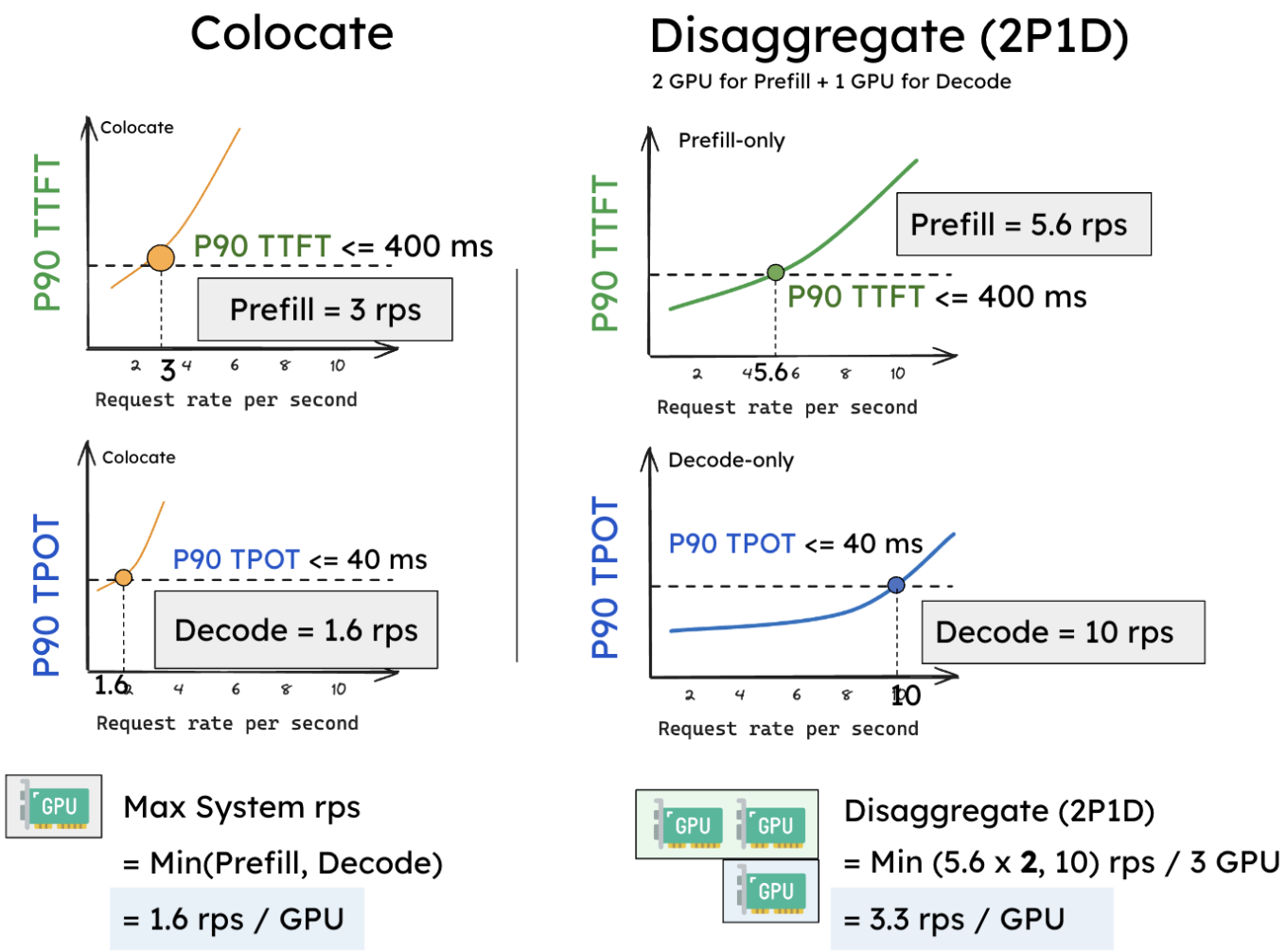


解耦预填充和解码阶段（PD分离）

通过将prefill和decode解耦到不同的GPU上，就可以避免这两个阶段互相干扰，都能更好的满足各自的SLO，并且可以为两个阶段各自指定并行策略。

下图的实验模拟了请求按泊松分布到达、用户输入长度为512、输出长度64、在单张A100-80GB上运行13B的LLM的情况下，逐步增加用户请求到达率时两个SLO的变化。两图中的水平线分别表示90%请求的TTFT < 0.4s和90%的TPOT < 0.04s，有以下观察：

- 左图：prefill和decode都在同一张显卡。为满足TTFT约束，吞吐量约为3 rps，为满足TPOT约束，吞吐量约为1.6 rps。因为要同时满足两个约束，所以有效吞吐量为1.6rps。
- 右图：每张显卡分别负责prefill和decode。prefill的GPU能达到5.6rps，decode的GPU能达到10rps。这样可以两个prefill任务搭配一个decode任务，需要三张GPU，平均每张显卡的有效吞吐量为3.3rps，在没有使用任何优化的情况下就实现了有效吞吐量翻倍。



论文中提出的优化算法

对于**高带宽集群**，跨节点 KV 缓存开销几乎可以忽略：

- 首先尝试不同的张量并行和流水线并行组合，找到prefill任务和decode任务在单 GPU 上的最佳 goodput。
- 根据拟合的模型预测每种配置下的 SLO 达成率和吞吐率，筛选出最佳方案。
- 通过复制多个模型实例确定所需实例数，以满足整体请求到达率 R。

对于**低带宽集群**，跨节点带宽受限，只能依赖机内 NVLINK 做高速数据传输：

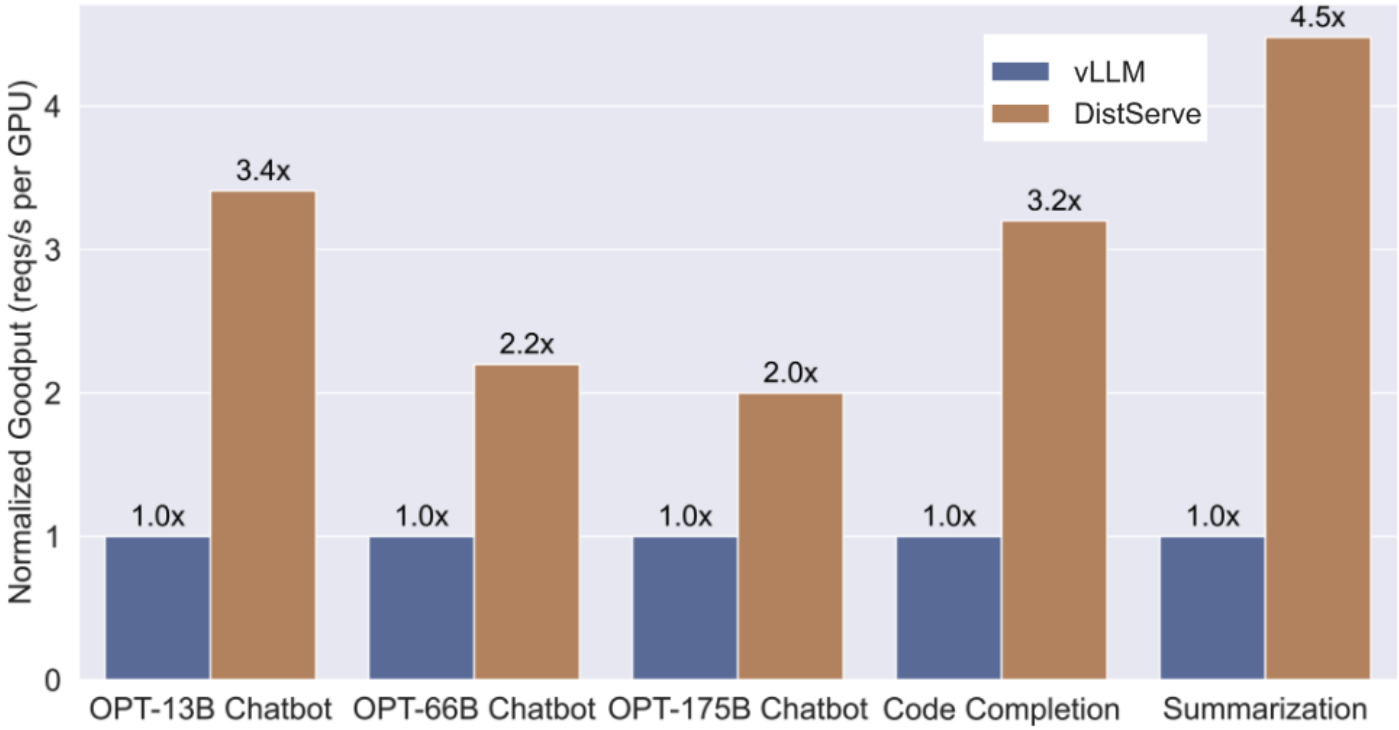
- 利用模型流水线并行把模型拆成多层，每层在不同卡上执行。
- 把同一层的prefill和decode任务都放在一台节点里，让 KV 缓存只需走 NVLINK，避免跨节点瓶颈。
- 对每个层，再枚举机内可用的张量并行度，模拟并选出最佳方案，然后按需复制模型实例。

最终的DistServe推理引擎遵循**FCFS**，新到的请求被投入等待队列最短的prefill引擎，执行完毕调度到最空闲的decode引擎。

实验结果

作者在三个SLO约束不同的数据集上进行了实验，实验结果如下图所示，可以看到DistServe的有效吞吐量显著高于vLLM框架。

LLM App	Data	TTFT	TPOT
Chatbot	ShareGPT	Tight	Medium
Code completion	HumanEval	Tight	Tight
Summarization	LongBench	Loose	Medium



目前vLLM、sglang等主流推理框架都已经实现了PD分离。

参考文章：

- 论文：《DistServe: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving》
- 作者团队的Blog：《Throughput is Not All You Need: Maximizing Goodput in LLM Serving using Prefill-Decode Disaggregation》