大模型推理系统与实践

第 6 讲状态管理与记忆组织

张书豪

华中科技大学计算机学院

2027 · 研究生课程课件

教学目标

核心结论

本讲讨论推理系统中状态对象的保存、搬运、共享与复用问题。

- 理解 DistServe 的阶段解耦。
- 理解 RAGCache 的知识对象缓存。
- 建立 KV、知识块、会话记忆的统一状态视角。

为什么状态问题会升级

核心结论

当服务进入多轮交互、RAG 和智能体工作流后，状态不再是一次性中间结果。

- 状态会跨请求、跨轮次持续存在。
- 状态对象可能被搬运到不同设备和不同阶段。
- 状态复用收益和一致性代价需要一起考虑。

状态对象有哪些类型

核心结论

从系统角度看，不同状态对象的差别主要体现在生成方式、驻留时间和复用条件上。

- 引擎内部状态：KV、激活、调度元数据。
- 外部知识状态：检索结果、知识块、索引缓存。
- 长程交互状态：会话历史、智能体记忆和工具上下文。

DistServe 的启发

核心结论

DistServe 说明 prefill 和 decode 的分离可以带来收益，但状态传输会成为新的系统代价。

- prefill 偏重算力，decode 偏重低排队时延和状态驻留。
- 解耦后可分别扩容和优化。
- 状态传输与协调复杂度随之出现。

DistServe 的系统启示

核心结论

DistServe 的重要意义在于将阶段差异重新建模为资源配置与状态传递问题。

- prefill 和 decode 不应强迫共享同一资源池。
- 一旦分离，状态搬运和阶段协同就进入主路径。
- 因此“解耦”本身不会免费带来收益。

RAGCache 的启发

核心结论

RAGCache 说明外部知识一旦进入在线推理链路，就会变成系统内部必须管理的缓存对象。

- 知识块注入会扩大 prefill 和 KV 压力。
- 热知识块存在跨请求复用机会。
- 检索与推理之间需要新的状态组织和重叠机制。

RAGCache 的系统启示

核心结论

RAGCache 将知识注入过程转化为状态缓存与流水重叠问题。

- 热知识块可以跨请求复用。
- 检索与推理需要联合优化而非串行拼接。
- 外部知识对象必须进入运行时视角。

统一状态对象视角

核心结论

KV、知识缓存和会话记忆虽然服务不同语义目标，但在系统上都属于长期存在且可复用的状态对象。

- 不同点在于复用条件、正确性要求和驻留时间。
- 相同点在于都需要考虑存储、调度、访问与回收。
- 统一视角有助于形成更强的系统抽象。

统一视角的研究价值

核心结论

如果我们能把这些对象放进同一抽象框架，就有机会统一讨论存储、调度、迁移和复用。

- 统一度量对象价值与代价。
- 统一考虑驻留层级和回收策略。
- 统一分析对调度器和执行路径的影响。

源码穿插：在 nano-vllm-hust 中观察状态对象

核心结论

nanovllm/engine/sequence.py 把请求状态、token 序列和 KV 相关元数据收在了同一个运行时对象里。

- Sequence 同时包含请求状态、token 序列和采样配置。
- status、num_cached_tokens 与 block_table 体现了运行时状态的不同维度。
- append_token()、__getstate__() 与 __setstate__() 说明状态对象会随阶段推进和跨进程序列化而变化。

源码穿插：Sequence 如何承载状态对象

核心结论

通过 Sequence 可以把“状态对象”从抽象概念落实到运行时最小承载单元。

- 它既保存 prompt 与 completion 的边界，也保存调度阶段与缓存进度。
- num_blocks、last_block_num_tokens 等属性把逻辑请求与物理块管理关联起来。
- 这一设计有助于学生理解状态对象为何会同时受到调度、KV 管理和执行路径的共同影响。

讲解：统一状态视角到底统一了什么

核心结论

这一讲不是把所有缓存混成一个词，而是把不同对象放到同一分析坐标里。

- KV、检索知识块、会话记忆都具有跨时间存在、可能复用、需要回收这三个共同特征。
- 它们的差别在于生成方式、正确性要求、搬运代价和复用条件不同。
- 所谓“统一状态管理”，不是让它们实现完全一样，而是用同一套语言讨论驻留、迁移、命名和价值。

讲解：第六讲应形成的研究意识

核心结论

状态一旦进入主路径，系统问题就不再只是计算快慢，而变成“对象如何持续存在”。

- 只谈计算资源而不谈状态落点，很多机制会在真实系统里失效。
- 只谈潜在复用而不谈搬运与一致性，很多收益会停留在理想叙述里。
- 因此这一讲要让学生意识到：推理系统正在从执行系统走向状态化在线系统。

例子：为什么 DistServe 要把 prefill 和 decode 拆开（1）

核心结论

状态管理不是“把对象存起来”，而是决定对象该在什么地方、什么时机被持有和搬运。

单机一体化路径

prefill 和 decode 共用一套资源，简单但容易让两类阶段互相干扰。

例子：为什么 DistServe 要把 prefill 和 decode 拆开（2）

核心结论

状态管理不是“把对象存起来”，而是决定对象该在什么地方、什么时机被持有和搬运。

阶段解耦路径

prefill 更靠近高吞吐算力，decode 更靠近低抖动稳态资源，状态搬运因此成为主问题。

例子：为什么 DistServe 要把 prefill 和 decode 拆开（3）

核心结论

状态管理不是“把对象存起来”，而是决定对象该在什么地方、什么时机被持有和搬运。

课堂结论

一旦阶段被拆开，系统研究的重点就从“存没存”转到“搬得动、对不对、值不值”。

例子：RAGCache 为什么让状态视角进一步扩张（1）

核心结论

课程中的“状态”不只包括 KV，还包括检索知识块、会话记忆和调度元数据。

传统理解

缓存常被理解成底层实现优化，像一个被动的加速器。

例子：RAGCache 为什么让状态视角进一步扩张（2）

核心结论

课程中的“状态”不只包括 KV，还包括检索知识块、会话记忆和调度元数据。

统一状态视角

检索结果、前缀块和会话历史都可能跨请求复用，因此它们都值得被当成一等状态对象建模。

例子：RAGCache 为什么让状态视角进一步扩张（3）

核心结论

课程中的“状态”不只包括 KV，还包括检索知识块、会话记忆和调度元数据。

系统含义

一旦对象具有跨时间和跨请求价值，系统就必须管理它的命名、引用、迁移和回收边界。

代码例子：Sequence 的序列化为什么值得课堂关注（1）

核心结论

状态管理课程里的代码例子，应让学生看到对象不只是运行时临时变量，还需要可传递与可恢复。

代码片段

```
def __getstate__(self): ...
def __setstate__(self, state): ...
```

代码例子：Sequence 的序列化为什么值得课堂关注（2）

核心结论

状态管理课程里的代码例子，应让学生看到对象不只是运行时临时变量，还需要可传递与可恢复。

课堂应追问

哪些字段如果丢失，请求将无法继续？哪些字段如果错传，会直接造成状态不一致？

代码例子：Sequence 的序列化为什么值得课堂关注（3）

核心结论

状态管理课程里的代码例子，应让学生看到对象不只是运行时临时变量，还需要可传递与可恢复。

系统含义

只要对象可能跨模块、跨节点或跨阶段流动，序列化语义就是系统正确性的一部分。

代码例子：统一状态管理器会暴露哪些接口（1）

核心结论

课程项目里一个常见扩展，是想象如果把 KV、知识块和会话记忆统一管理，需要哪些最小接口。

接口草图

```
state_id = state_store.put(obj, metadata)
state_store.pin(state_id, device="gpu")
state_store.evict(policy="lru")
```

代码例子：统一状态管理器会暴露哪些接口（2）

核心结论

课程项目里一个常见扩展，是想象如果把 KV、知识块和会话记忆统一管理，需要哪些最小接口。

课堂应追问

这些接口背后分别对应命名、驻留、迁移还是回收问题？

代码例子：统一状态管理器会暴露哪些接口（3）

核心结论

课程项目里一个常见扩展，是想象如果把 KV、知识块和会话记忆统一管理，需要哪些最小接口。

结论

统一状态管理的难点不是接口名字，而是不同状态对象的价值函数和一致性需求并不相同。

例子：状态搬运为什么经常吞掉理论收益（1）

核心结论

很多机制论文的理论收益很强，但一旦进入真实系统，搬运与同步代价就会迅速显形。

理想世界

把更适合 prefill 的资源和更适合 decode 的资源拆开，看起来可以两头占优。

例子：状态搬运为什么经常吞掉理论收益（2）

核心结论

很多机制论文的理论收益很强，但一旦进入真实系统，搬运与同步代价就会迅速显形。

真实世界

KV 或知识对象一旦要跨节点移动，带宽、延迟、重组和一致性全都会冒出来。

例子：状态搬运为什么经常吞掉理论收益（3）

核心结论

很多机制论文的理论收益很强，但一旦进入真实系统，搬运与同步代价就会迅速显形。

教学重点

讲状态复用时必须同时讲状态搬运，否则学生会高估机制落地的直接性。

课堂练习：把三类状态放到同一张表里（1）

核心结论

学生需要学会比较不同状态对象，而不是只会孤立讨论 KV。

练习一

比较 KV、RAG 检索块、会话记忆三者在生成方式、驻留时间和复用条件上的差异。

课堂练习：把三类状态放到同一张表里（2）

核心结论

学生需要学会比较不同状态对象，而不是只会孤立讨论 KV。

练习二

说明哪一类状态最适合先做 observe-only 探针，哪一类最需要严格正确性边界。

课堂练习：把三类状态放到同一张表里（3）

核心结论

学生需要学会比较不同状态对象，而不是只会孤立讨论 KV。

练习三

用一句话回答：为什么第 6 讲的关键词不是“缓存”，而是“状态组织”？

课堂讨论

核心结论

状态管理的关键难点，在于缓存对象如何稳定纳入系统控制环。

- 为什么知识缓存不能被简单看成外挂组件？
- 当状态搬运进入主路径后，如何权衡收益与代价？
- 会话记忆在系统层面与 KV 有哪些相同点和差异？

本讲总结

核心结论

DistServe 与 RAGCache 都要求系统显式管理状态，但 KV、检索文档和长期记忆的复用键、失效条件与安全边界并不相同。

- 状态管理是推理基础设施的核心问题之一。
- 后续架构与异构执行讨论都要继续围绕状态展开。

本讲参考资料

- **DistServe**（OSDI 2024） 原文链接
- **RAGCache**（arXiv:2404.12457） 原文链接
- **Mooncake**（FAST 2025） 原文链接
- **NEO**（MLSys 2025） 原文链接

课件中的论文数字均指作者在特定模型、硬件和工作负载下报告的结果，不代表无条件性能。

对应 Tutorial

核心结论

本讲对应 Tutorial 5。先独立作答，再使用对应 Tutorial 的教师讨论要点核对。

- 文件：build/tutorials/Tutorial_05_ 状态管理与记忆组织.pdf
- 动手运行、日志记录与提交要求统一查看 build/experiments/ 和对应 experiment 源目录下的 assignment_spring-2027/。