大模型推理基础设施

第 7 讲推理系统架构

张书豪

华中科技大学计算机学院

研究生课程课件

教学目标

核心结论

本讲从架构层面理解现代推理 runtime，并建立论文机制与真实代码边界之间的对应关系。

- 区分控制面与数据面。
- 识别调度器、执行器、KV 管理器和通信层。
- 理解为什么日志、脚本与回归也是系统组成部分。

典型推理系统的部件

核心结论

一个现代 serving runtime 至少包括请求入口、调度器、执行器、缓存层和监控层。

- API server 与 tokenizer
- scheduler 与 admission control
- model runner 与 kernel/runtime 层
- KV manager、通信层与 metrics/logging

为什么架构图不能只画模块名

核心结论

好的架构理解不仅要知道“有哪些模块”，还要知道请求、状态和控制信号如何在模块间流动。

- 请求路径回答“谁先被调用”。
- 状态路径回答“哪些对象长期存在”。
- 控制路径回答“谁在决定系统行为”。

控制面与数据面

核心结论

控制面决定做什么，数据面决定怎么做。

- 控制面关注准入、路由、优先级、配额和反馈。
- 数据面关注 token 计算、状态驻留、kernel 执行和通信。
- 不区分两者，问题定义就容易混乱。

控制面典型问题

核心结论

控制面的核心是资源治理和服务目标兑现。

- 什么时候接收新请求。
- 怎样分配配额和优先级。
- 什么时候扩容、限流或改变路由。

数据面典型问题

核心结论

数据面的核心是执行路径、状态驻留与通信组织。

- 哪些 token 被算、以什么批次被算。
- KV 和其他状态如何分配、迁移和释放。
- kernel、图执行和并行通信如何被组织。

为什么要进入真实代码库

核心结论

只有进入代码，才能真正理解论文机制在系统中落在哪些接口和模块上。

- 请求生命周期在哪里开始与结束。
- KV block 在哪里分配和回收。
- 指标输出如何与论文图表对应。

源码穿插：用 nano-vllm-hust 建立最小架构图

核心结论

nano-vllm-hust 可以把请求入口、调度、KV 管理和执行路径落到一套足够小的代码结构上。

- 请求入口：nanovllm/engine/llm_engine.py
- 调度与状态推进：nanovllm/engine/scheduler.py
- KV 块管理：nanovllm/engine/block_manager.py
- 执行与 KV 准备：nanovllm/engine/model_runner.py

源码穿插：从架构图到代码导读任务

核心结论

以 nano-vllm-hust 为例，代码导读可以先收束到一条最小主路径。

- 从 LLMEngine.generate() 进入全局驱动循环。
- 从 Scheduler.schedule() 观察控制面如何选择当前执行对象。
- 从 ModelRunner.run() 观察数据面如何准备输入、block table 与采样输出。

代码导读的最低要求

核心结论

一次合格的代码导读，应能够将关键函数与系统职责清晰对应起来。

- 哪个函数负责接收请求。
- 哪个循环负责组织批次。
- 哪个模块负责状态分配和统计输出。

辅助模块的重要性

核心结论

脚本、日志与回归流程共同决定系统结果能否被复现和比较。

- benchmark 配置决定实验边界。
- 日志与指标决定你是否真的理解系统行为。
- 回归脚本决定结果是否可持续维护。

课堂练习

核心结论

课堂练习可先做一次最小代码导读，建立架构与实现之间的映射关系。

- 定位请求入口。
- 定位调度循环。
- 定位 KV 分配和回收逻辑。
- 定位 metrics 输出点。

讲解：系统架构课为什么必须把三条路径画出来

核心结论

好的架构分析不只是列模块名，而是同时说明请求路径、状态路径和观测路径。

- 请求路径回答“系统怎么跑起来”。
- 状态路径回答“哪些对象在不同模块之间被持有、更新、回收”。
- 观测路径回答“我们靠什么知道系统真的按预期工作”。

讲解：为什么日志、benchmark 和回归也属于架构

核心结论

如果只把 runtime 主循环画进架构图，课程最后很容易出现“会讲代码，不会验证系统”的断裂。

- benchmark 决定 workload 如何进入系统，结果如何被汇总。
- 日志和指标决定问题是否可诊断，机制是否可解释。
- 回归脚本决定改动能否长期维护，因此它们不是外围工具，而是系统可持续性的组成部分。

例子：为什么控制面和数据面一定要分开讲（1）

核心结论

系统架构课的关键不是记模块名，而是知道谁负责决策、谁负责执行、谁负责携带状态。

控制面

接收请求、维护队列、决定调度策略、记录指标和做资源治理。

例子：为什么控制面和数据面一定要分开讲（2）

核心结论

系统架构课的关键不是记模块名，而是知道谁负责决策、谁负责执行、谁负责携带状态。

数据面

真正执行 tokenizer、prefill、decode、KV 访问和结果回传。

例子：为什么控制面和数据面一定要分开讲（3）

核心结论

系统架构课的关键不是记模块名，而是知道谁负责决策、谁负责执行、谁负责携带状态。

课程结论

如果控制面和数据面在讲授中混在一起，学生很难看出系统边界和可改控制点。

例子：分布式拓扑为什么会改变同一个问题的答案（1）

核心结论

单机上看起来只是“多开几个请求”的问题，进入分布式拓扑后会变成通信和协调问题。

单卡

主要矛盾是调度、KV 驻留和本地显存组织。

例子：分布式拓扑为什么会改变同一个问题的答案（2）

核心结论

单机上看起来只是“多开几个请求”的问题，进入分布式拓扑后会变成通信和协调问题。

多卡 TP / PP / DP

同一个 batch 决策会影响跨卡同步、负载倾斜和阶段间流水稳定性。

例子：分布式拓扑为什么会改变同一个问题的答案（3）

核心结论

单机上看起来只是“多开几个请求”的问题，进入分布式拓扑后会变成通信和协调问题。

教学意义

架构课要让学生意识到：系统模块关系会随着部署拓扑变化而重排。

代码例子：从 LLMEngine 到 ModelRunner 的调用链（1）

核心结论

系统架构课最适合补的代码页，是跨组件调用链而不是单函数细节。

调用链草图

```
LLM.generate -> LLMEngine.generate
LLMEngine.step -> Scheduler.schedule
LLMEngine.step -> ModelRunner.execute_model
```

代码例子：从 LLMEngine 到 ModelRunner 的调用链（2）

核心结论

系统架构课最适合补的代码页，是跨组件调用链而不是单函数细节。

课堂应追问

这一链路上，哪个位置属于控制面，哪个位置已经进入数据面？

代码例子：从 LLMEngine 到 ModelRunner 的调用链（3）

核心结论

系统架构课最适合补的代码页，是跨组件调用链而不是单函数细节。

系统含义

架构分析的第一步不是画框图，而是把真实调用关系翻译成模块职责图。

代码例子：为什么可观测性也属于架构的一部分（1）

核心结论

没有日志、指标和 trace，系统架构图只是静态想象，无法支持真实调试和研究。

最小可观测性接口

```
logger.info(stage, request_id, batch_size)
metrics.observe("ttft_ms", value)
trace.annotate("decode_step_begin")
```

代码例子：为什么可观测性也属于架构的一部分（2）

核心结论

没有日志、指标和 trace，系统架构图只是静态想象，无法支持真实调试和研究。

课堂应追问

哪些观测点属于控制面信号，哪些属于执行面信号？

代码例子：为什么可观测性也属于架构的一部分（3）

核心结论

没有日志、指标和 trace，系统架构图只是静态想象，无法支持真实调试和研究。

结论

真正可维护的架构，必须把观测路径和主执行路径一起设计。

例子：一个坏架构图通常坏在哪里（1）

核心结论

学生最常见的问题是画出很多框，但无法说明数据、状态和决策谁在流动。

坏图一

只列模块名，不说明调用方向和控制点。

例子：一个坏架构图通常坏在哪里（2）

核心结论

学生最常见的问题是画出很多框，但无法说明数据、状态和决策谁在流动。

坏图二

只画数据流，不画状态驻留和资源回收。

例子：一个坏架构图通常坏在哪里（3）

核心结论

学生最常见的问题是画出很多框，但无法说明数据、状态和决策谁在流动。

坏图三

把日志、benchmark、回归系统完全排除在架构之外，导致系统边界被画得过窄。

课堂练习：把架构图画成三层而不是一团（1）

核心结论

课程作业里更推荐“三层图”：入口与控制、执行与状态、观测与验证。

练习一

在你的架构图中分别标出请求入口、调度器、执行器、状态管理器和指标记录点。

课堂练习：把架构图画成三层而不是一团（2）

核心结论

课程作业里更推荐“三层图”：入口与控制、执行与状态、观测与验证。

练习二

指出哪一个模块最像“全局控制点”，哪一个模块最像“热路径执行点”。

课堂练习：把架构图画成三层而不是一团（3）

核心结论

课程作业里更推荐“三层图”：入口与控制、执行与状态、观测与验证。

练习三

用一句话回答：为什么日志和 benchmark 不应该被排除在系统架构课之外？

本讲总结

核心结论

架构视角是统一论文、代码与实验分析的关键桥梁。

- 系统研究不能只停留在机制名词。
- 理解模块边界，才能设计可落地项目。

对应 Tutorial

核心结论

本讲对应 Tutorial 6。先独立作答，再对照下一页参考答案。

- 文件：build/tutorials/Tutorial_06_ 推理系统架构.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-2026/。