vLLM-HUST 实践案例与练习补充

LLM Inference Systems and Practice - 2026

这份材料不是独立专题，而是给各讲穿插使用的实践案例库。

BidKV、benchmark 异常、prefix reuse、异构后端适配、维护经验和 PR 工作流都会作为例子嵌入课程概念中。

课程项目鼓励学生进入 vLLM-HUST 开发流程，提交 PR、patch、benchmark 数据或可复现实验文档。

当前收录的例子

- KV 压力与调度策略
- benchmark 折线与实验可信度
- prefix reuse 与状态一致性
- 异构后端适配与环境记录
- vLLM-HUST 维护经验
- 从 issue 到 PR 的课程项目流程

使用方式

这份材料不是独立专题，而是给各讲穿插使用的实践案例库。每个例子都可以放进对应章节，用来帮助学生把课程概念和 vLLM-HUST 真实开发任务连起来。后续如果有新的论文、开源 PR、benchmark 发现或系统调优经验，也可以继续追加到这里。

课程项目不要求学生做一个脱离主线的 demo，而是要求他们进入 vLLM-HUST / vLLM-Ascend-HUST / benchmark / dev-hub 的真实开发流程：认领 issue、复现问题、定位代码路径、修改代码或实验配置、补充测试或 benchmark、提交 PR 或 patch，并解释 CI 与性能数据。

例子一：KV 压力为什么会改变调度策略

对应章节：第 4 讲 请求调度；第 5 讲 KV 缓存。

在普通连续批处理中，请求调度常被讲成“谁进入 batch、谁等待下一轮”。但在长上下文和高并发场景里，请求已经占用的 KV cache 也是重要状态。一个请求被暂停后，系统可能需要保留 KV、迁移 KV，或者重新 prefill，这些代价并不相同。

可以用 BidKV: Utility-Guided Preemption Scheduling for KV-Pressure LLM Serving（SC 2026）作为课堂例子：当 KV cache 成为紧张资源时，调度器不只是判断“哪个请求先跑”，还要判断“哪个请求的状态更值得保留”。这能引出 utility、preemption cost、tail latency 和 throughput 之间的权衡。

课堂练习：给定 6 个请求的输入长度、已生成 token 数、KV block 占用和预计剩余输出长度，让学生分别用 FIFO、最大 KV 占用优先抢占、最短剩余时间优先和 utility-guided 策略做一次手工调度，并比较重算成本。进阶任务可以进一步要求学生在 vLLM-HUST 中定位调度相关代码路径，写出可验证的修改方案。

例子二：同一条性能折线为什么不能直接下结论

对应章节：第 2 讲 工作负载与评价指标；第 10 讲 论文比较方法；第 11 讲 实验方法与验证。

在 vLLM-HUST benchmark 中，折线图看起来很直观，但如果实验条件没有对齐，很容易把参数变化误判成系统优化或性能退化。

需要检查的条件包括：

- workload 是否一致，例如 random、ShareGPT、agent、code、vision、throughput 和 latency 模式不能混在一起比较。
- 模型是否一致，例如 dense、MoE、VL、INT8 和 FP16 不能只看名字相近。
- 硬件是否一致，例如 910B2、910B3、多卡拓扑和驱动栈必须明确标注。

- 参数是否一致，例如 input / output token size、concurrency、max-num-seqs、max-model-len、block size、enforce-eager 等都会改变结果。
- baseline 是否一致。比较 vLLM-HUST PR 时，应和同一条件下的 vLLM / vLLM-Ascend baseline 对比，而不是混用两个历史优化版本。

课堂练习：给学生一组折线图和 JSON metadata，让他们标出哪些点可以比较，哪些点必须补 baseline，哪些点可能是参数不一致导致的异常。实践任务要求学生按 vLLM-HUST benchmark 规范补齐 metadata 或 workload 结果，而不是只在报告里截图说明。

例子三：Prefix reuse 不是“缓存命中率”那么简单

对应章节：第 5 讲 KV 缓存；第 6 讲 状态管理与记忆组织；第 7 讲 推理系统架构。

代码生成、文档问答、agent-research 和多轮对话经常共享 system prompt、文档上下文或历史对话。prefix reuse 可以减少重复 prefill，但它也会引入新的系统问题：prefix 如何组织、如何淘汰、如何跨请求复用、如何保证状态一致。

可以让学生比较三种负载：

- 所有请求完全随机，没有共享 prefix。
- 所有请求共享 system prompt，但用户输入不同。
- 请求共享 system prompt 和长文档上下文，但输出长度差异很大。

课堂练习：设计一个简单 prefix tree，记录每个节点的引用计数、KV block 数和最近访问时间。让学生实现两种淘汰策略，并观察 TTFT 和 memory pressure 的变化。

例子四：异构后端适配为什么会影响实验可信度

对应章节：第 8 讲 执行优化与异构路径；第 9 讲 异构平台适配；第 12 讲 开源系统实践。

在 Ascend / NPU 后端上，性能不只取决于模型和 batch。runtime 版本、torch-npu pin、CANN、attention backend、graph capture、kernel fallback、通信拓扑都会影响结果。

这个例子可以帮助学生理解：开源系统适配不是“让代码跑起来”就结束，还要让 benchmark、CI 和环境记录能解释性能变化。课程项目中，涉及 Ascend / NPU 的小组需要说明硬件型号、驱动栈、依赖 pin、启动脚本和 benchmark 参数。

课堂练习：给定两次实验的依赖列表和启动参数，让学生判断哪一次结果更适合写进性能报告，哪些信息缺失会让结论不可复现。

例子五：从 vLLM-HUST 维护看系统工程经验

对应章节：第 12 讲 开源系统实践；第 13 讲 课程项目工作坊。

vLLM-HUST 的维护工作不只是“改代码”。很多修改来自 CI、benchmark、环境依赖、硬件排队、参数记录、上游同步和文档规范。这些经验适合写进课程材料，因为学生如果只看一次成功的 demo，很难理解系统软件为什么需要长期维护。

可以把维护经验拆成六类：

- 基线优先：任何性能结论都要先确认 baseline。比较 vLLM-HUST 的 PR 时，应和同一条件下的 vLLM / vLLM-Ascend baseline 比较，不能把两个历史优化版本直接当成 preferred pair。
- 元数据优先：每个 benchmark 结果都要记录 commit、模型、硬件、驱动、依赖、启动参数、workload、输入输出长度、并发和 dtype。缺少这些信息，折线图再漂亮也很难解释。
- CI 不是附属工作：CI 失败经常暴露环境假设，例如 checkout transport、Node 版本、Python wheel、torch / torch-npu pin、linkcheck 容器等。维护 CI 是保证后续实验可复现的一部分。
- 小 patch 更容易评审：工程修改应尽量只解决一个问题。修 benchmark、修文档、修依赖、改调度路径最好拆开说明，避免把实验数据、格式化和功能改动混在一个 PR 里。
- 硬件资源要可调度：Ascend/NPU 资源有限，长时间 benchmark 需要预约、排队和记录资源占用。课程项目要区分本机可完成的文档/单测任务和需要服务器的性能实验。
- 上游同步要有取舍：从 vLLM、vLLM-Ascend、triton-ascend 等上游同步代码时，要先判断哪些是版本对齐，哪些是可抽取的通用修复，哪些会影响当前 benchmark 结论。

一个合格的课程项目 PR 至少应该说明：

- 它修改了哪个系统路径。
- 它如何被测试。
- 它和 baseline 的比较条件是什么。
- 如果性能变化明显，原因是否能被日志、profile 或 metadata 支持。
- 如果只是工程修复，它如何提升后续实验的可复现性。

课堂练习：给学生一个维护记录片段，例如“CI 因依赖版本失败”“benchmark 缺少 baseline”“某 workload 性能突降”“文档中硬件型号写错”。要求学生写出五段说明：问题现象、影响范围、排查路径、修复方案、验证方式。

项目要求：课程项目不能只提交最终结果截图。每组需要提交一份维护记录，说明问题从哪里来、为什么值得修、修了什么、如何验证、还有哪些风险没有解决。

例子六：课程项目如何从 issue 到 PR

对应章节：第 12 讲 vLLM-HUST 开源开发与 PR 工作流；第 13 讲 vLLM-HUST 课程项目工作坊；第 14 讲 PR 汇报与课程总结。

一个实践项目可以按下面流程推进：

- 选题：从 vLLM-HUST 相关仓库中选择一个清晰 issue，例如 benchmark 缺 baseline、某 workload metadata 不完整、CI 失败、文档缺少环境说明，或某个调度/KV 路径需要小修复。
- 复现：记录 commit、模型、硬件、驱动、依赖、启动脚本和失败现象。
- 定位：说明涉及哪些模块、函数、配置文件或实验脚本。
- 修改：提交最小必要 patch，避免无关重构。
- 验证：给出单元测试、smoke test、benchmark 或文档预览结果。
- 汇报：提交 PR 或 patch，并在课程汇报中解释 review 意见和后续风险。

例子七：把维护经验转化成教材页

对应章节：第 12 讲 vLLM-HUST 开源开发与 PR 工作流；第 14 讲 PR 汇报与课程总结。

教师可以把一次维护过程整理成一页教材，结构保持稳定：

- 背景：这次维护发生在哪个仓库、哪个模块、哪个 workload 或哪个 CI job。
- 现象：学生能观察到什么，例如失败日志、性能折线、缺失数据点、文档不一致。
- 判断：这是代码问题、参数问题、环境问题、硬件资源问题，还是实验设计问题。
- 修改：采用了什么最小修改，为什么不做更大范围重构。
- 验证：跑了哪些测试、benchmark 或文档预览，结果如何记录。
- 经验：这次维护提醒我们以后如何写参数、设计 CI、记录实验或拆分 PR。

课堂练习：每个小组选一个 vLLM-HUST 维护 PR 或 issue，按上面的结构整理成 1 页教材草稿。要求少用口号，多写证据；少写“提升效果显著”，多写“在哪个 workload、哪个 baseline、哪个参数下发生了什么变化”。

后续追加建议

后续课程中可以继续加入这些类型的例子：

- 新的 vLLM-HUST / vLLM-Ascend-HUST 优化 PR。
- 官方上游接收的修复或功能改进。

- 真实 benchmark 中发现的异常点和补数过程。
- vLLM-HUST 维护过程中形成的 CI、benchmark、文档和依赖管理经验。
- 新论文中的调度、缓存、内存管理或异构执行设计。
- 学生课程项目中形成的可复现实验片段。