大模型推理系统与实践

第 8 讲执行优化与异构路径

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

本讲将执行优化问题从局部算子提速推进到执行路径与资源放置层面。

- 理解 I/O、放置与批处理如何共同决定执行收益。
- 分析 FlexGen 与 PowerInfer 的系统意义。
- 理解异构执行路径为何需要预测、放置与执行协同。

执行优化不只是算子提速

核心结论

执行优化的对象是整条数据路径，而不是单个 kernel 的局部时间。

- 算子融合与图执行。
- launch 开销与动态形状管理。
- 内存布局、带宽限制与通信重叠。

执行优化中的常见误判

核心结论

仅依据 GPU 利用率或 kernel 时间判断优化方向，往往会忽略路径组织对 serving 系统性能的主导作用。

- 数据放置方式会改写 I/O 代价。
- 动态 batch 会改变形状和 launch 模式。
- 通信和计算是否重叠会直接影响端到端表现。

FlexGen 的启发

核心结论

FlexGen 把“单卡运行超大模型”重新定义为 I/O-aware 的系统设计问题。

- GPU、CPU、磁盘之间的数据放置共同决定收益。
- 仅仅把模型 offload 并不足够。
- 放置策略、访问顺序和压缩必须协同设计。

FlexGen 的关键贡献

核心结论

FlexGen 的关键贡献在于将资源层次利用转化为可系统优化的主体问题。

- 权重、KV 和中间状态可以采用不同层次策略。
- I/O 访问顺序本身决定性能上界。
- 批处理大小与存储层级设计相互耦合。

PowerInfer 的启发

核心结论

PowerInfer 表明，异构执行路径设计需要围绕预测、放置与执行协同展开。

- 热点神经元与冷神经元可采用不同放置策略。
- 收益依赖激活分布和跨设备代价。
- 异构执行需要结构性观察，而不是事后搬运。

PowerInfer 的关键贡献

核心结论

PowerInfer 将模型内部激活分布转化为可用于调度与放置决策的系统信息。

- 热点神经元留在 GPU，冷部分交给其他层级。
- 收益依赖预测质量和跨设备代价。
- 这说明模型结构特征可以进入系统路径设计。

为什么局部加速常常不能兑现

核心结论

如果没有看清主导瓶颈和全链路代价，再高的局部提速也可能被系统其他部分吃掉。

- kernel 更快不等于 P99 更低。
- 放置更省显存不等于成本更优。
- 必须把局部机制放回端到端链路验证。

执行优化应如何报告收益

核心结论

执行优化的报告方式应该同时交代局部机制、路径变化和端到端结果。

- 局部数据说明改了什么。
- 过程数据说明路径为什么变了。
- 端到端指标说明收益是否真的被用户感知到。

讲解：执行优化课真正想纠正什么误解

核心结论

很多人一谈执行优化就想到 kernel，但系统里真正决定收益兑现的往往是整条路径。

- 算子更快只说明局部时间变短，不说明等待、I/O 或同步是否一起下降。
- serving 场景里的动态形状、阶段切换和新请求插入，会让“训练里成立的最优执行”不再稳定。
- 所以这一讲真正要训练的是路径分解能力，而不是只会比较算子数字。

讲解：如何判断一次执行优化是否真的成立

核心结论

执行优化必须同时给出局部证据、路径解释和端到端结果，缺一不可。

- 局部证据说明改了什么，例如 I/O 重叠、编译命中、launch 减少。
- 路径解释说明为什么这些局部变化会传递到请求级时延或吞吐。
- 端到端结果说明收益有没有真的被用户感知到，而不是停留在 profiler 里。

例子：为什么“优化执行路径”不等于“找更快 kernel”（1）

核心结论

执行优化课最需要纠正的，是把所有性能改进都理解成单点算子提速。

算子层优化

可以降低单轮执行时间，但未必解决主路径上的等待、搬运和重组代价。

例子：为什么“优化执行路径”不等于“找更快 kernel”（2）

核心结论

执行优化课最需要纠正的，是把所有性能改进都理解成单点算子提速。

执行路径优化

关注的是数据何时准备好、图何时稳定、I/O 是否与计算重叠、资源是否被正确放置。

例子：为什么“优化执行路径”不等于“找更快 kernel”（3）

核心结论

执行优化课最需要纠正的，是把所有性能改进都理解成单点算子提速。

课程结论

真正的执行优化，往往是路径组织与资源编排问题，而不只是更快的算子实现。

例子：图执行为什么在 serving 里更难（1）

核心结论

训练里稳定的图，不代表 serving 里也稳定，因为请求长度、batch 组成和阶段切换都在变化。

训练场景

形状相对稳定，图编译和缓存更容易复用。

例子：图执行为什么在 serving 里更难（2）

核心结论

训练里稳定的图，不代表 serving 里也稳定，因为请求长度、batch 组成和阶段切换都在变化。

serving 场景

prompt 长度、decode 轮数和新请求插入都会破坏图的稳定性。

例子：图执行为什么在 serving 里更难（3）

核心结论

训练里稳定的图，不代表 serving 里也稳定，因为请求长度、batch 组成和阶段切换都在变化。

教学重点

所谓 compile stabilization，本质是在给动态服务路径创造“足够稳定”的形状类。

代码例子：最小编译开关为什么值得上课讲（1）

核心结论

即使课堂不深入编译器实现，也应该让学生看到“执行路径优化”如何落到运行时开关。

代码片段

```
llm = LLM(path, enforce_eager=False, max_model_len=4096)
// eager / compiled path changes runtime behavior
```

代码例子：最小编译开关为什么值得上课讲（2）

核心结论

即使课堂不深入编译器实现，也应该让学生看到“执行路径优化”如何落到运行时开关。

课堂应追问

为什么同一个模型在 eager 与 compiled path 下，收益边界和失败模式都可能不同？

代码例子：最小编译开关为什么值得上课讲（3）

核心结论

即使课堂不深入编译器实现，也应该让学生看到“执行路径优化”如何落到运行时开关。

系统含义

执行路径优化必须伴随更细的可验证性要求，而不能只看一次跑得快不快。

代码例子：双缓冲与 I/O 重叠的伪代码（1）

核心结论

课程中可以用小段伪代码让学生理解“放置与重叠”比数学公式更接近系统实现。

伪代码

```
prefetch(next_weight_chunk)
compute(current_chunk, current_kv)
synchronize_if_needed()
```

代码例子：双缓冲与 I/O 重叠的伪代码（2）

核心结论

课程中可以用小段伪代码让学生理解“放置与重叠”比数学公式更接近系统实现。

课堂应追问

这里真正要隐藏的是什么：权重搬运、KV 访问，还是 launch 开销？

代码例子：双缓冲与 I/O 重叠的伪代码（3）

核心结论

课程中可以用小段伪代码让学生理解“放置与重叠”比数学公式更接近系统实现。

课程结论

所谓 I/O-aware，核心并不是“搬得快”，而是“搬运时尽量不阻塞主计算路径”。

例子：speculative decoding 为什么也是执行路径问题（1）

核心结论

speculative decoding 不只是算法策略，它还要求 draft 与 target 两条执行链在系统里协同。

理想收益

通过先草拟、再验证，减少 target model 的有效工作量。

例子：speculative decoding 为什么也是执行路径问题（2）

核心结论

speculative decoding 不只是算法策略，它还要求 draft 与 target 两条执行链在系统里协同。

真实代价

需要维护额外模型状态、验证回退逻辑和两条路径之间的同步成本。

例子：speculative decoding 为什么也是执行路径问题（3）

核心结论

speculative decoding 不只是算法策略，它还要求 draft 与 target 两条执行链在系统里协同。

教学意义

这类机制天然属于“执行路径组织”而不只是“模型技巧”。

课程研究案例（内部实验）：MLP Materialization

核心结论

profiler-driven optimization 的第一课：不要只优化一个看起来显眼、但不在主导路径上的算子。

- 实测 serving path 中反复出现 `MatMulV2 -> SwiGlu -> MatMulV2`。
- 内部 profile 显示矩阵乘法主导该路径，SwiGLU 只占较小比例；未归档实验条件前不引用具体百分比。
- 单算子或 inplace 优化未必带来端到端收益，下一步应检查 MLP 边界、materialization 与 writeback。

说明：本页为课程研究案例或教学假设，不代表已发表论文结论；只有在实验包给出模型、硬件、工作负载和原始记录时，数字才可作为结果引用。

执行路径优化的证据问题

核心结论

一个优化没有兑现时，课程里要训练学生沿路径定位，而不是简单说“实现不够好”。

1. 它减少的是 kernel time、launch、I/O、状态搬运，还是等待？
2. 这个局部变化是否落在 TTFT、TPOT 或 throughput 的主导路径上？
3. profiler、trace 和端到端结果是否互相解释，还是互相矛盾？

课堂练习：用路径语言解释一次优化（1）

核心结论

执行优化课最重要的能力，是把“优化点”翻译成“路径改变”。

练习一

任选一种优化，说明它减少的是 launch、I/O 等待、图不稳定，还是状态搬运。

课堂练习：用路径语言解释一次优化（2）

核心结论

执行优化课最重要的能力，是把“优化点”翻译成“路径改变”。

练习二

说明为什么“更快 kernel”不能自动推出“更低端到端时延”。

课堂练习：用路径语言解释一次优化（3）

核心结论

执行优化课最重要的能力，是把“优化点”翻译成“路径改变”。

练习三

用一句话回答：这讲里“执行路径”真正指的是什么？

课堂讨论

核心结论

执行优化的核心在于系统组织能力，而非局部编码能力本身。

- 为什么异构执行路径设计不等于简单 offload？
- 为什么 I/O-aware 设计会改变最优 batch 与放置策略？
- 什么时候局部加速最容易被全链路抵消？

本讲总结

核心结论

执行优化同时涉及路径组织、资源放置与实验验证。

- 真正重要的是端到端收益的稳定兑现。
- 异构执行讨论将自然通向下一讲的平台适配问题。

本讲参考资料

- **FlashAttention**（NeurIPS 2022） 原文链接
- **FlexGen**（ICML 2023） 原文链接
- **PowerInfer**（SOSP 2024 / arXiv:2312.12456） 原文链接
- **FlashInfer**（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 7。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_07_ 执行优化与异构路径.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/`。