大模型原理与应用

第 11 课：学术前沿 | 长上下文、推理与记忆

看见研究热点背后的共同问题

看什么

- 推理能力到底来自哪里
- 长上下文与 KV cache 为何始终卡脖子
- token、表示和记忆还有哪些重做空间

先抓住的总线索

今天的大模型前沿研究，已经不只是“堆更大模型”：

- 有人研究 RL 到底带来了什么
- 有人研究长上下文的真实极限
- 有人研究如何压缩 KV cache
- 有人研究 tokenization 是否该被替换
- 有人研究不同模型是否能直接共享内部表示
- 有人研究模型能否持续学习而不遗忘

这说明：

> 大模型研究正在从“规模扩张”转向“能力机制与系统瓶颈”

热点一：RL 与推理能力

- 现在很多人关心的，不只是“RL 能不能把分数做高”
- 而是：RL 到底让模型学会了什么？
- 它是真的教会了模型更会推理
- 还是只是把 base model 里原本就有的能力放大了？

换句话说：

- RL 到底是在“造新能力”
- 还是在“激活旧能力”

DAPO：RL for reasoning 的工程化路线

- clip higher
- dynamic sampling
- token-level policy gradients
- overlong reward shaping

这些词看起来有点硬，但它们其实都在解决几个很实际的问题：

- entropy collapse
- reward noise
- training instability

它的重要意义在于：

- 把“长链条推理 RL”从黑箱经验转成更公开的系统设计
- 也就是说，研究者开始把 RL 当成一个可以拆开分析、逐项优化的工程系统

RL 是否真的带来新能力

- 这个问题现在特别热，因为它关系到后面的整个训练路线。
- 一种看法是：推理能力本来就埋在 base model 里
- RL 只是把它“逼出来”“放大出来”
- 另一种看法是：RL 的确会让模型形成新的推理习惯

这些报告提出的几种假设包括：

- RL 产生全新的推理机制
- RL 重用了已有表征
- RL 放大了 base model 中本已存在的 reasoning pattern

这类问题非常重要，因为它影响：

- 我们要不要继续重度依赖 RL
- 还是更应该挖掘 base model 本身潜力

RL 的泛化与遗忘

- 另一个很现实的问题是：
- 用 RL 调过以后，模型会不会只会做这一类题？
- 它会不会在变强的同时，把原来的一些能力弄丢？

关键观点：

- catastrophic forgetting 是持续适配的核心难题
- on-policy RL 可能比 SFT / off-policy RL 更稳
- KL divergence 与遗忘关系密切

这给我们的启发是：

- 后训练不只是提高某个 benchmark
- 也要看是否伤害别的能力

Alignment 会不会压缩创造力

- 还有一条很有意思的研究线：
- 对齐以后，模型会不会更“听话”，但也更“保守”？
- 也就是：alignment 会不会 shrink the generative horizon?

核心分析工具是：

- branching factor
- token-level entropy
- prefix-level entropy

结论方向是：

- 对齐后的模型常常分支数更低
- 输出空间被压缩
- 推理可以更稳定，但创造力或多样性可能下降

这提示我们：

- alignment 与 creativity 并不总是同向

热点二：长上下文为什么总是不够长

- 模型标称 128k、1M context，不等于真实可用
- 多轮对话、长文档推理、超长输入常出现“context rot”

也就是说：

> 长上下文问题既是建模问题，也是评测问题

多轮对话中的“失联”

- LLMs get lost in multi-turn conversation
- 平均性能下降约 39%
- aptitude 下降
- unreliability 上升

这类工作提醒我们：

- 真正的难点不是“把一大段文本塞进去”
- 而是模型能不能在很多轮之后还知道：
- 我们现在在谈什么
- 前面已经说到哪了
- 哪些信息重要，哪些信息该忽略

长上下文为什么难

- 现在越来越多研究发现：
- 模型真正“能用好”的上下文长度，往往远小于它宣称支持的长度

原因之一是：

- 训练时，超远位置本来就见得少
- 所以模型对远距离 token 没那么熟
- 在 RoPE 这类位置编码下，这个问题会更明显

这意味着：

- 长上下文不足不只是 inference 技巧问题
- 还和训练数据中的位置分布直接相关

STRING / 位置重映射类思路

- 不一定硬训更长序列
- 也可以通过位置索引重映射
- 让长距离 token 使用模型更熟悉的位置区域

这类方法的启发是：

- 有些“长上下文改进”并不是再造一个更大的模型
- 而是想办法让老模型别在陌生位置上那么容易迷路

把文本当图像：OCR 路线

- Glyph
- DeepSeek-OCR

它们的共同问题是：

- 能不能把超长文本先渲染成图像
- 再用视觉模型压缩为更少的视觉 token

这样做的目标不是识别图片里的字本身，而是：

- 用“视觉压缩”换取更长有效上下文
- 也就是说，不是单纯扩窗口，而是换一种更省 token 的读入方式

OCR 压缩路线的核心思想

这类方法的假设是：

- 文本 token 太细
- 视觉系统可以把字符 / 单词 / 局部版面压缩成更少单元
- 于是可以从“扩窗口”改成“改表示”

这和传统长上下文方法很不一样：

- YaRN / 插值：改位置编码
- sparse / eviction：改 attention / cache
- OCR：改输入表示

这也是最近很新的一个方向：

- 与其一直想“怎么多塞一点 token”
- 不如改问“同样的信息，能不能用更少的 token 表达”

热点三：KV cache 不只是缓存

- KV cache 是不是可以从“推理中间产物”变成可操作的记忆与通信对象?

这是一类非常系统导向的研究。

为什么 KV cache 成为研究热点

因为长上下文和推理效率的核心瓶颈之一就是：

- KV cache 占显存大
- 随 context length 线性或更糟增长
- 对大模型和长推理非常昂贵

所以很多研究都在问：

- 该压缩什么？
- 该删什么？
- 该保留什么？

因为今天很多系统瓶颈，说到底不是模型不会答，而是太贵、太慢、太占显存。

PyramidKV：按层分配缓存预算

- 不同层的 attention / 信息聚合模式不一样
- 所以每层 KV cache 不该使用同样大小预算

这背后的研究问题是：

- 模型是否存在 recognizable pyramidal information funneling?
- 如果有，缓存压缩策略就应该 layer-aware

这比“统一裁剪”更精细。

- 说得直白一点：不是所有层都同样重要，所以不能一刀切地压。

KV eviction 的三条路线

1. Static eviction
    - prefill 阶段决定删什么
2. Dynamic eviction
    - decode 阶段根据运行时信号删什么
3. Redesigned training
    - 训练时直接限制注意范围

这是一个非常实用的研究分类框架。

你可以把它理解成三个问题：

- 先删，还是边生成边删？
- 靠规则删，还是靠运行时信号删？
- 还是干脆在训练阶段就让模型学会“少看一点”？

Cartridges：把 KV cache 变成可复用记忆

- 把长语料压成小型可交换 KV prefix
- 不改 base model 权重
- 像“插卡”一样按语料加载

它的优势在于：

- memory 显著下降
- throughput 显著提高
- 可组合多个 cartridge

这已经不是“缓存优化”，而是在做外接记忆模块。

- 也就是说，cache 开始像一个可以存、取、组合的知识插件。

Cache-to-Cache：模型间直接传内部语义

- LLM 之间能不能不通过文本说话
- 而是直接传 KV cache / hidden semantics

核心动机：

- 文本通信带来信息瓶颈
- 文本通信有歧义
- token-by-token 交流太慢

这类工作非常前沿，因为它重新定义了“多模型协作”。

- 过去模型之间像“发消息”
- 这里是在想：能不能直接“传脑内表征”

C2C 的意义

如果 cache-to-cache 真能稳定工作，意味着：

- 多个专长模型之间不必只靠自然语言接口
- 可以直接共享内部表示
- multi-LLM collaboration 可能更高带宽、更低延迟

这会改变：

- agent 系统设计
- 推理系统的模块协同方式

如果这条路走通，未来很多 agent 系统可能不再只是“文字对话编排”， 而会变成“内部表示协作”。

热点四：token 与表示是否该被重做

- 过去我们默认：先分词，再送进模型。
- 但最近越来越多人开始问：
- 这个默认步骤本身，会不会就已经把信息弄坏了？

BLT：字节级 patch 取代固定 token

- tokenization 是 biased heuristic preprocessing
- 会带来 input sensitivity、拼写问题、多语言不公平、模态限制
- 是否可以直接在 byte 层面工作，再动态 patching?

BLT 的思想是：

- 不先固定 subword token
- 而是动态把 bytes 聚合成 patch
- 让模型按内容分配计算

这是一种很“end-to-end” 的语言建模方向。

它更容易懂的说法是：

- 不预先告诉模型“哪几个字母算一个词”
- 而让模型自己决定，哪里该切细一点，哪里可以合并得粗一点

热点五：推理过程如何并行化

未来 reasoning systems：

- self-reflection 很长
- tool use 很慢
- multi-agent 很贵
- 那么测试时推理能否并行展开?

这个问题越来越重要，因为今天很多模型不是“不会想”，而是“想得太慢”。

并行推理研究在问什么

多种 test-time computation 形式：

- tool use
- multi-agent debate
- judger guidance
- divergent thinking

难点在于：

- 有些推理天然顺序化
- 有些子问题可并行
- 有些 agent 之间又存在依赖

所以研究重点变成：

- 如何动态分配推理资源

说白了就是：

- 哪些步骤必须按顺序想

热点六：压缩会不会破坏推理完整性

- 蒸馏
- 剪枝
- 量化

这些效率手段会不会损伤 reasoning integrity?

这个问题很重要，因为真实部署几乎一定要压缩。

为什么这个问题重要

因为现实部署里你几乎不可能一直用最大模型、全精度、不压缩。

所以真正的问题不是：

- 压不压缩

而是：

- 哪种压缩最伤 reasoning？
- 哪些权重对 reasoning 特别关键？

也就是说：

- 不要只问“压完还能不能跑”
- 而要问“压完以后，模型是不是还真的会想”

热点七：模型能否持续学习

一条非常重要但经常被忽略的线：

- 模型上线后，能不能像学生一样持续学习？
- 学新知识时能不能不忘旧知识？

这件事现在越来越重要，因为很多模型不是训练完就结束了， 而是部署后还要不断吸收新任务、新知识、新偏好。

持续学习的稀疏路线

两层 sparsity 思想：

- structural sparsity：Memory Layers at Scale
- learning sparsity：Sparse Memory Finetuning

核心目标：

- 只动最少参数
- 提高知识容量
- 在更新时减少干扰

这类研究说明：

- 未来大模型“学新知识”未必总靠全量继续训练
- 可能更像稀疏、可定位的记忆更新

对学生来说，这条线最值得抓住的一点是：

- 未来更新模型，可能更像“给模型加一个小记忆模块”
- 而不是每次都把整台机器重新训练一遍

把这些热点串起来

共同趋势：

- 研究者正在拆解大模型系统中的每个关键接口

具体包括：

- 推理接口：RL、GRPO、DAPO、reasoning in base models
- 上下文接口：context length、conversation robustness、OCR compression
- 缓存接口：PyramidKV、eviction、cartridge、C2C
- 表示接口：byte-level patches、internal representation sharing
- 学习接口：continual learning、sparse memory updates

这就是“学术前沿”真正的含义：

> 不再只问模型多大，而是问系统每个部件还能如何被重构

为什么这部分值得看

虽然论文很多，但你真正要记住的是几种判断框架：

1. 一个方法是在改训练，还是改推理？
2. 它是在扩能力，还是在修系统瓶颈？
3. 它是增加参数，还是重用已有表示？
4. 它优化的是 accuracy、memory、latency，还是 robustness？

如果能用这四个问题看论文，你就不会被热点标题牵着走。

这部分你该带走什么

1. RL 研究在追问推理能力从哪来
2. 长上下文难点不只是窗口大小
3. KV cache 正在变成记忆和通信接口
4. token、表示、持续学习都在被重审
5. 前沿正在转向机制与系统设计

第 7 课（第一部分）收束

看前沿，关键不是记论文名，而是看它在改哪一层机制