机器学习系统2026春

第七章 模型剪枝

张燕咏 讲席教授 yanyongz@ustc.edu.cn
张午阳 特任教授 wuyangz@ustc.edu.cn

中国科学技术大学
University of Science and Technology of China

第一节课

剪枝基础: 从概念到硬件落地

建立剪枝的系统判据: (a) GEMM 形状缩小 (b) 硬件 sparse pattern

本课路线图

主线问题: 剪枝如何真正实现 LLM 推理加速? 本堂课分 5 个 Part 层层推进, 最终给出两条系统判据。

主线问题: 剪枝如何真正兑现 LLM 推理加速?

Part A / B / C: 从概念到质疑

A 剪枝概念 (8页)
- 定义、三维度框架 (Which × How × When)、粒度、评分 (magnitude/Hessian)、one-shot vs iterative

B 历史代表 (5页)
- OBD/OBS 前史、Deep Compression、EIE、Lottery Ticket、Liu 2019 重新评估

C 系统问题 (5页)
- 带宽瓶颈 → 实测压缩≠加速 → **提出判据 (a)(b)**

Part D / E: 硬件答案与总结

D 硬件答案 (9页)
- NVIDIA 2:4 Sparse TC: 规则 → 存储 → MMA → 实测 → 训练
- 同时讨论 N:M 泛化、Blackwell 下一代 GPU

E 补充+总结 (4页)
- 块稀疏 (attention mask)、其他硬件生态、对比表
- L1 小结 + 过渡 L2 (static LLM 统一配方)

关键要点: 路线图: A 框架 → B 历史 → C 问题 → D 答案 → E 总结; 两条判据 (a)(b) 是全章锚点。

PART A

剪枝概念

定义 · 权重冗余 · 三维度框架 (Which × How × When) · 粒度 · 评分 · 流程

本课大纲 (进度)

▸ Part A	Part B	Part C	Part D	Part E
剪枝概念	历史代表	系统问题	硬件答案 2:4	总结 + 过渡

A.1 什么是剪枝?

定义与两种操作形式

- **剪枝 (Pruning)** = 把训练好的神经网络中 "不重要" 的权重/神经元移除
- **Soft mask (软剪枝)**: 将权重置零, 保留矩阵形状; 需硬件/kernel 能识别并跳过零值 (2:4, 非结构化)
- **Structural removal (硬剪枝)**: 直接删除整个 channel / head / layer, 矩阵形状变小; 任意后端都能加速
- **目标**: 在精度几乎无损 (<1% drop) 的前提下减少 参数 / FLOPs / DRAM 流量 / 推理延迟

与其他模型压缩技术的关系

- **量化 (Quantization)**: 降低权重 bit 数 (FP32 → INT8 → INT4); 保留连接但降低精度
- **蒸馏 (Distillation)**: 用 teacher 模型指导 student; 改变模型规模但不改结构
- **低秩分解 (Low-rank)**: W ≈ U V^T, 用低秩近似原矩阵
- 四者独立, 生产上常组合 (如 Minitron = 剪枝+KD; SparseGPT-2:4 = 剪枝+GPTQ 量化)

本节要回答的三个问题

- **Which (剪什么)**: 粒度 (单权重 / head / 层) × 评分 (magnitude / activation / Hessian)
- **How (如何加速)**: 硬件认不认该 pattern? 零值真跳过吗? 带宽真节省吗?
- **When (何时剪)**: one-shot (几小时) vs iterative (数天到数周)
- 一个方法 = Which × How × When 的组合

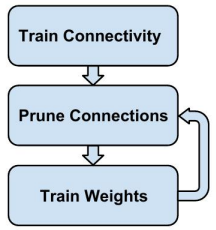


Figure 2: Three-Step Training Pipeline.

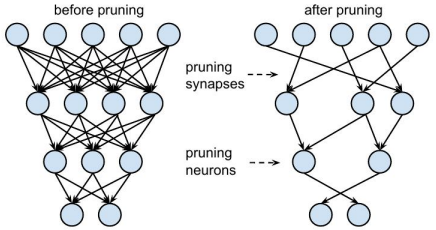


Figure 3: Synapses and neurons before and after pruning.

关键要点: 剪枝是 "发现并去除冗余"; Soft/Hard 两种操作 + 三个维度框架, 是本章的语言。

图: Han et al., Learning both Weights and Connections, NeurIPS 2015 (arxiv:1506.02626)

A.2 权重冗余: 为什么剪枝是可能的?

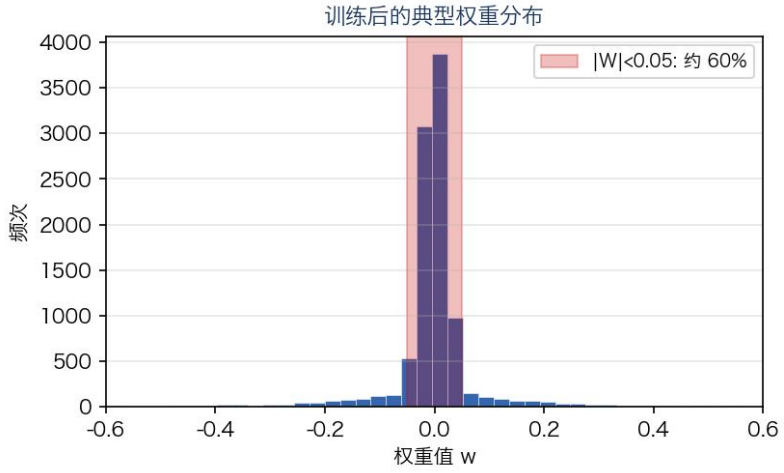
过参数化 (over-parameterization) 是训练的需要, 不是推理的需要。剪枝利用的是这个差异。

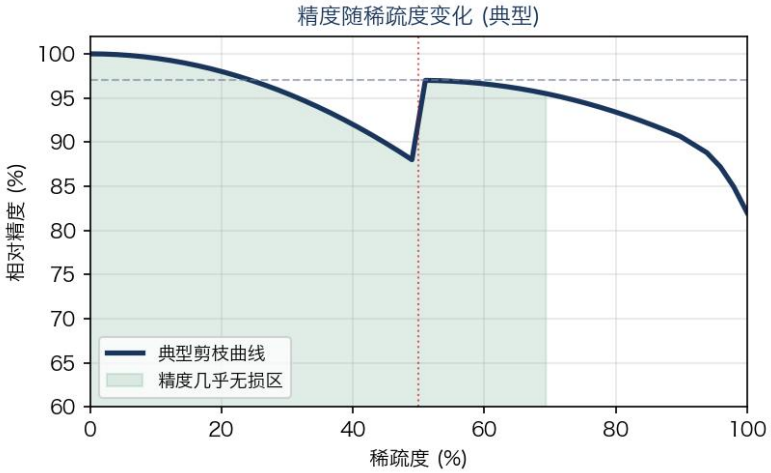
实证观察

- 训练后的权重 **呈长尾分布**:
  - 多数权重接近 0 (对输出贡献小)
  - 少数权重承担主要激活能量
- ResNet-50 的 Conv 权重: **60-70% 的 |w| < 0.05**
- Transformer FFN 权重: 分布更分散, 但仍长尾
- 直觉: 把 "尾巴" 剪掉, 对输出影响很小

理论视角: Lottery Ticket

- **Frankle & Carbin 2019**: dense 网络中早就存在一个稀疏子网 (winning ticket)
- 用原始初始化重新训练这个子网 ⇒ 可达原网络精度
- 意味着: **剪枝 = 找到, 不是创造**
- 也解释了为什么剪枝不会把模型变 "废"
- S1.14 会详细介绍这一观察





关键要点: 训练的冗余 = 推理的优化空间; 剪枝是对 over-parameterization 的合理回应。

A.3 剪枝的三个维度: Which × How × When

一个具体的剪枝方法 = Which × How × When 三个维度的组合; 后 5 页分别展开。

Which (剪什么)

① 粒度 (granularity)
- 单权重 (非结构化)
- head / channel (结构化)
- 2:4 block (半结构化)
- 决定硬件友好度

② 评分 (importance score)
- magnitude: |W|
- activation: |W| · ‖X‖
- gradient: |g · W|
- Hessian: OBS/OBD 家族
- 决定精度上限

How (如何加速)

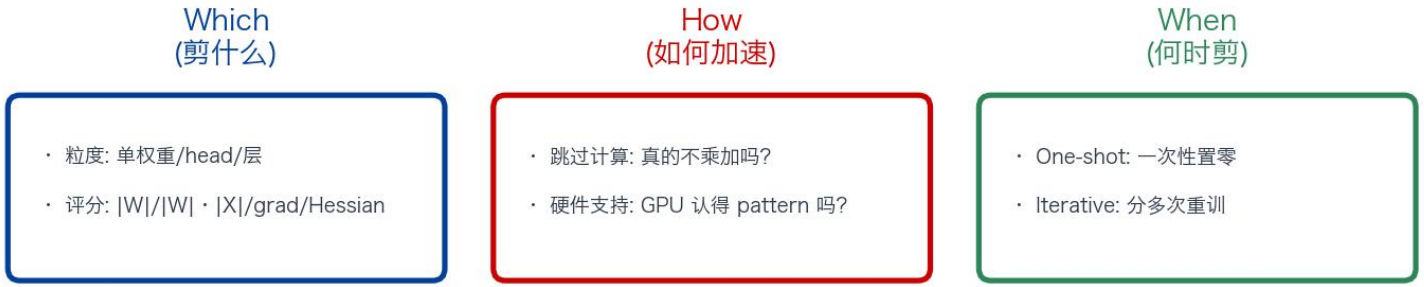
① 三个 "真"
- **真节省**: DRAM 不读?
- **真跳过**: 零值不乘加?
- **真识别**: GPU 认 pattern?

决定系统收益
- 同样 50% 稀疏度
- 非结构化 GPU 几乎无加速
- 2:4 能 2× 吞吐
- 结构化通用加速

When (何时剪)

① 一次性
- 成本低, 分钟-小时级
- SparseGPT, Wanda

② 迭代
- 剪 → 重训 → 再剪
- 精度最好 (可达 99%)
- 成本高 (天-周)
- Lottery Ticket, Slimming



关键要点: Which×How×When 是本堂课的框架语言; L2/L3 每篇论文都在这个三维空间的某一点。

A.4 Which 之一: 粒度 (三轴分类)

粒度越粗, 硬件越友好, 单次精度损失也越大; 下面三种粒度覆盖从精度优先到硬件优先的整个光谱。

① 非结构化 (Unstructured)

粒度
- 每个权重独立判断
- 可达 >95% 稀疏

代表方法
- Han 2015/2016
- Lottery Ticket 2019
- SparseGPT (非结构变体)

优点
- 精度损失最小
- 压缩上限最高

局限
- 零值位置散乱
- dense TC 无法利用 → GPU 无加速

② 结构化 (Structured)

粒度
- 整行 / channel / head / layer
- 或 hidden-dim slice

代表方法
- Filter Pruning (Li 2016)
- Network Slimming (Liu 2017)
- LLM-Pruner, SliceGPT, Minitron

优点
- GEMM 形状直接变小
- 任意 GPU 后端通用加速

局限
- 粒度粗, 精度下降快
- 通常需 fine-tune / KD

③ 半结构化 (N:M, Block)

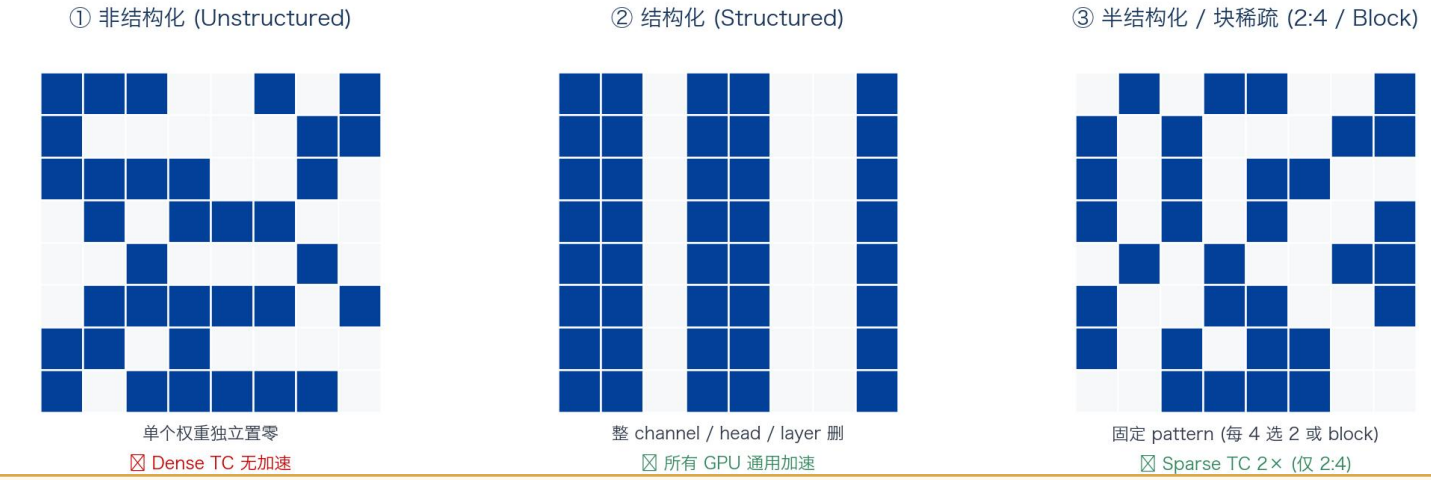
粒度
- 每 N 个连续权重恰好 M 个为零
- 或固定 block (16×16)

代表方法
- 2:4 (NVIDIA A100+ 主流)
- OpenAI Block-Sparse
- SparseGPT-2:4, Wanda-2:4

优点
- 硬件 2× MMA 吞吐
- 2-bit metadata 解码

局限
- 受硬件 pattern 约束

剪枝三轴: 同样 50% 稀疏度, 硬件待遇天差地别



关键要点: 三种粒度非互斥, 实际方法常在两轴间 (如 2:4 是非结构化与结构化的折中)。

A.5 Which 之二: 重要性评分 (1/2)

评分 (importance score): 对同一粒度, 如何判断 "哪些不重要"? 两种无训练成本的评分。

① Magnitude: $s = |W_{ij}|$

直觉
- 权重绝对值越小, 贡献越弱
- 把 bottom-k% 置零

成本
- **零**: 无训练, 无 calibration; 只看权重本身

代表方法
- Han 2015 Pruning + Deep Compression
- Network Slimming (Liu 2017): BN γ 作为 scaling

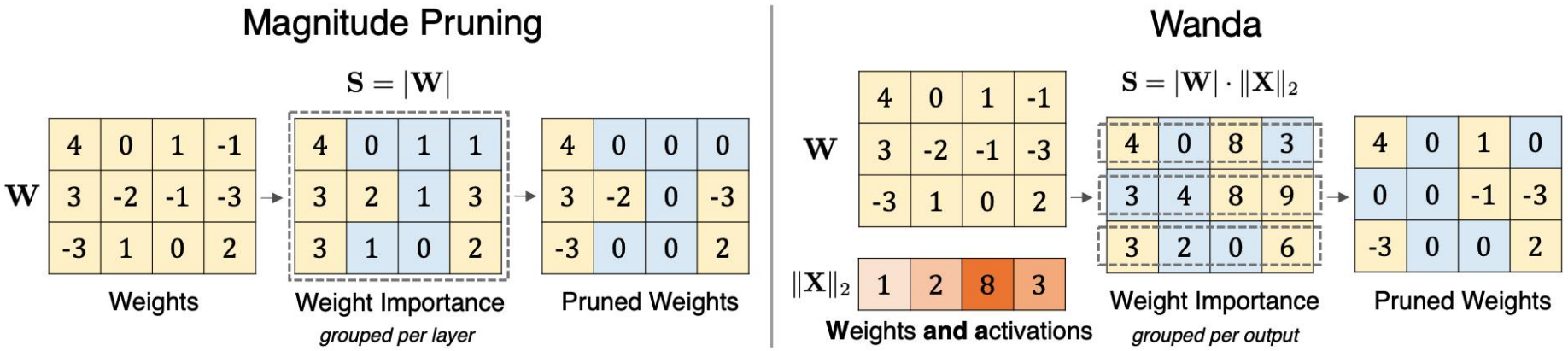
局限
- 忽略 activation: 同 |W| 若激活小其实可剪
- 对未 normalize 的模型效果差
- LLM 规模用 magnitude 精度下降快

② Activation-based: $s = |W_{ij}| \cdot \|X_j\|_2$

直觉
- 权重 × 激活 = "实际贡献"
- activation 大时小权重也重要
- 天然处理 LLM outlier 现象

成本
- 仅需 128 条 calibration 数据
- 无梯度, 无反向; Wanda 几分钟完成

代表方法
- **Wanda** (Sun et al., ICLR 2024)
- 3 行伪代码; LLaMA-7B 接近 SparseGPT 精度



关键要点: 无训练评分 (magnitude / activation) 是 LLM 规模首选。

图: Sun et al. 'Wanda: A Simple and Effective Pruning for LLMs', ICLR 2024 Fig 1

A.6 Which 之二: 重要性评分 (2/2)

二阶泰勒展开的核心想法

Taylor 二阶展开:

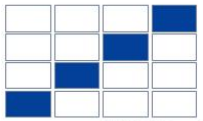
$$\Delta L \approx g^T \Delta W + \frac{1}{2} \Delta W^T H \Delta W$$

收敛后 $g \approx 0$, 用 Hessian H 估计

把哪个权重置零, 损失变化最小

从 OBD/OBS 到 SparseGPT

OBD (1990)



对角 Hessian

简化, 成本低

OBS (1993)

完整 Hessian

精度高, 成本高

↓

现代复兴:

SparseGPT (2023)

layer-wise 近似 OBS

③ Gradient (Taylor 一阶)

s = |g · W|

- 需反向计算梯度
- LLM-Pruner 2023 使用
- 多用于结构化剪枝

(按 head/channel 打分)

④ OBD (Hessian 对角)

LeCun et al., 1990

- 二阶 Taylor + 对角近似
- $\Delta L \approx \frac{1}{2} h_{ii} \Delta W_i^2$
- 对角矩阵, 成本低
- 但忽略权重间相关性

④ OBS (Hessian 完整)

Hassibi et al., 1993

- 完整 Hessian 求逆
- 同时更新余权重补偿
- SparseGPT 2023 复兴

(layer-wise 分块 XX^T)

关键要点: 评分成本 ↑ → 精度 ↑; LLM 规模下 Wanda (activation) 已接近 SparseGPT (Hessian) 的精度。

OBD: LeCun 1990; OBS: Hassibi 1993; SparseGPT: Frantar 2023 arxiv:2301.00774

A.7 When: One-shot vs 迭代剪枝

流程 (schedule): 时间轴上剪枝可一次完成或逐步进行, 成本与精度有数量级差别。

One-shot Pruning (LLM 主流)

流程 (3 步)
- 1. 所有权重打分 (magnitude / activation / Hessian)
- 2. 按阈值一次性置零
- 3. (可选) 少量 fine-tune 恢复精度

代表方法
- SparseGPT (Hessian): OPT-175B 约 4h
- Wanda (activation): 几分钟完成
- LLM-Pruner (gradient): 数小时

优点
- 成本极低, 128 条 calibration 够用
- 不处理原训练 pipeline

局限
- 稀疏度 >70% 时精度下降较快
- 需与 heal (见 L2) 组合

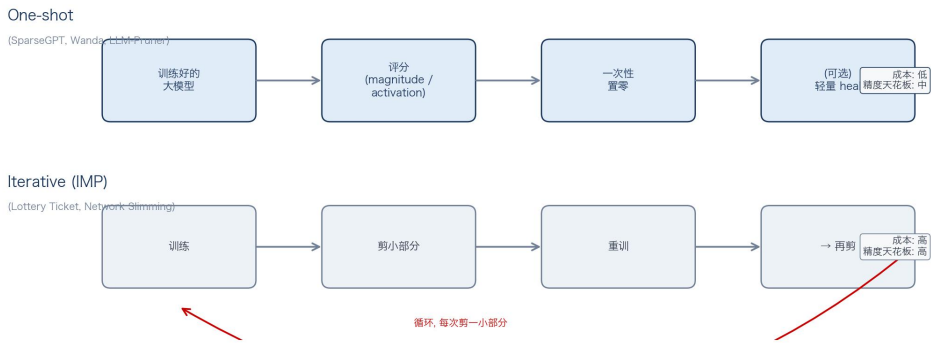
Iterative Magnitude Pruning (IMP)

流程 (4 步循环)
- 1. 训练至收敛
- 2. 剪去少量权重 (如 20%)
- 3. 重训至恢复精度
- 4. 回到 2, 反复多轮 (如 5-20 轮)

代表方法
- Lottery Ticket: IMP + rewind
- Network Slimming: BN γ + iterative prune

优点
- 可达 >95% 稀疏, 精度最好
- Lottery Ticket 95% sparse, MNIST/CIFAR 几乎无损

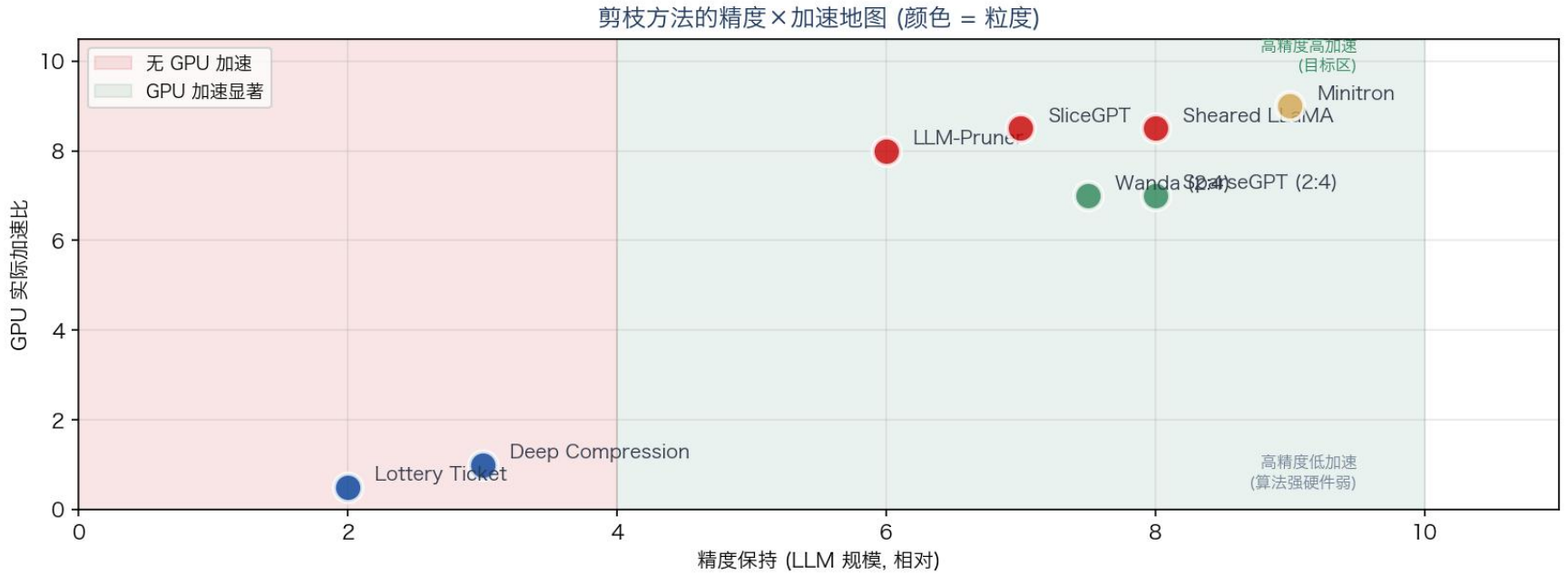
局限
- 每轮重训, 总成本 5-20× dense training
- LLaMA-70B 单次训练就需数月, 迭代不现实



关键要点: LLM 规模下只有 one-shot 可行: 这是 L2 所有统一配方的前提条件。

A.8 Part A 小结: 剪枝方法的二维地图

把代表方法按 (精度, 加速) 两维投影, 可以看到各方法的定位和剩余空白。



方法	粒度	评分	流程
Han 2016	非结构化	magnitude	iterative
Lottery Ticket	非结构化	magnitude	iterative + rewind
SparseGPT 2023	非结构化 / 2:4	Hessian (OBS)	one-shot
Wanda 2024	非结构化 / 2:4	activation	one-shot
LLM-Pruner	结构化 (head/ch)	gradient (Taylor)	one-shot
Minitron 2024	结构化 (w/d/h/e)	importance + KD	one-shot + heal

关键要点: Which×How×When 三维框架 + 方法地图: 这是 L2/L3 所有论文的通用语言。

PART B

历史代表

1990 前史 (OBD/OBS) · Deep Compression · EIE · Lottery Ticket · Liu 2019

本课大纲 (进度)

√ Part A	▸ Part B	Part C	Part D	Part E
剪枝概念	历史代表	系统问题	硬件答案 2:4	总结 + 过渡

B.1 前史: 剪枝的早期工作 (1990–2017)

剪枝不是新技术: 1990 就有系统研究; 但大规模落地要等到 2015 深度学习兴起。

经典二阶方法 (1990s)

OBD (LeCun 1990)
- 二阶对角 Hessian 近似
- $\Delta L \approx \frac{1}{2} h_{ii} \Delta W_i^2$
- 删使 loss 变化最小的权重
- 当时在 MLP 上验证

OBS (Hassibi 1993)
- 完整 Hessian 求逆
- 同时更新余权重补偿
- 精度更好, 成本高 $O(d^3)$
- **2023 SparseGPT 复兴**

非结构化复兴 (2015)

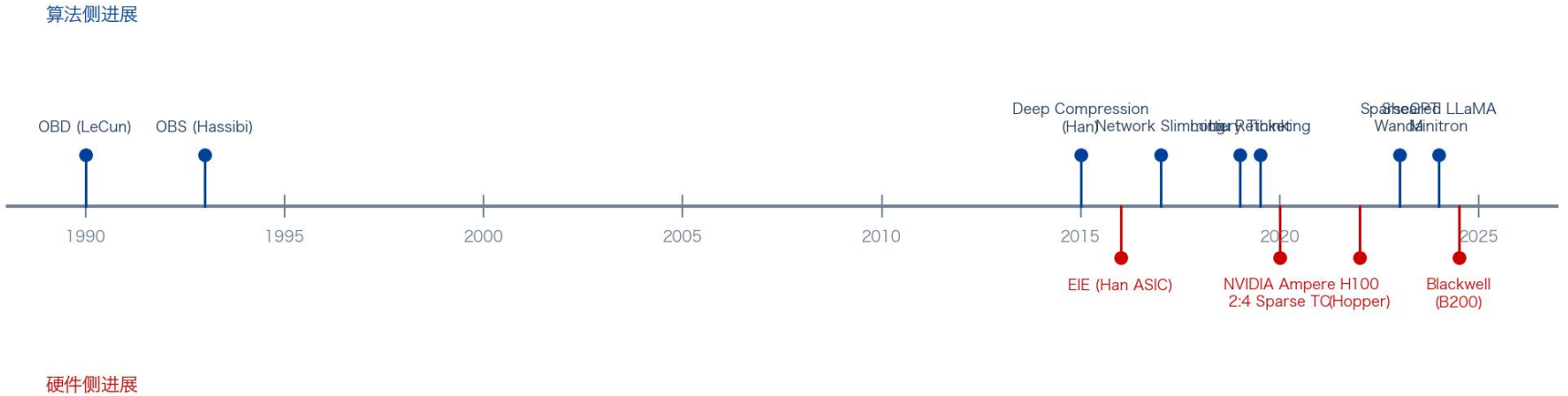
Han 2015 Pruning
- arxiv:1506.02626
- 非结构化 magnitude 剪枝
- AlexNet 压缩 9×
- 重训恢复精度
- 配套 Deep Compression (B.2)

观念突破
- 深度学习时代第一次大规模做剪枝
- 证明 "DNN 可以剪"

结构化剪枝 (2016–17)

Filter Pruning (Li 2016)
- L1 范数对 conv filter 打分
- 结构化剪枝的早期代表
- ResNet/VGG 验证

Network Slimming (Liu 2017)
- ICCV 2017
- 用 BN γ 作为 channel scale
- 训练中 L1 正则化 γ
- γ 小 ⇒ 剪 channel
- **为 L2 结构化剪枝奠基**



关键要点: 剪枝 30+ 年历史: 1990s 理论 (OBD/OBS) → 2015 Han 复兴 → 2017 结构化 (Slimming)。

OBD: LeCun 1990 NeurIPS; OBS: Hassibi 1993 NeurIPS; Han 2015 arxiv:1506.02626; Slimming: Liu 2017 ICCV

B.2 Deep Compression (Han et al., ICLR 2016) (Best Paper Award Recipient)

首次将 **剪枝 + 量化 + Huffman 编码** 串联起来, 并配套设计 EIE 加速器兑现为硬件加速。

三阶段压缩流程

① Pruning (剪枝)
- 按 magnitude 剪除
- 迭代 + 重训

② Quantization (量化)
- 32-bit → 5-bit
- K-means 聚类 codebook

③ Huffman 编码
- 对量化后的结果做无损压缩
- 高频值用短 code

核心数字
- AlexNet **35×** (240→6.9 MB)
- VGG-16 **49×**, 精度无损

对 MLSys 的意义

算法-硬件协同设计
- 同作者 **EIE** (ISCA 2016)
- 见下页 B.3 详解

影响
- 剪枝从 "研究" 进入 "系统"
- 奠定 tinyML / edge DL
- 后续稀疏硬件研究

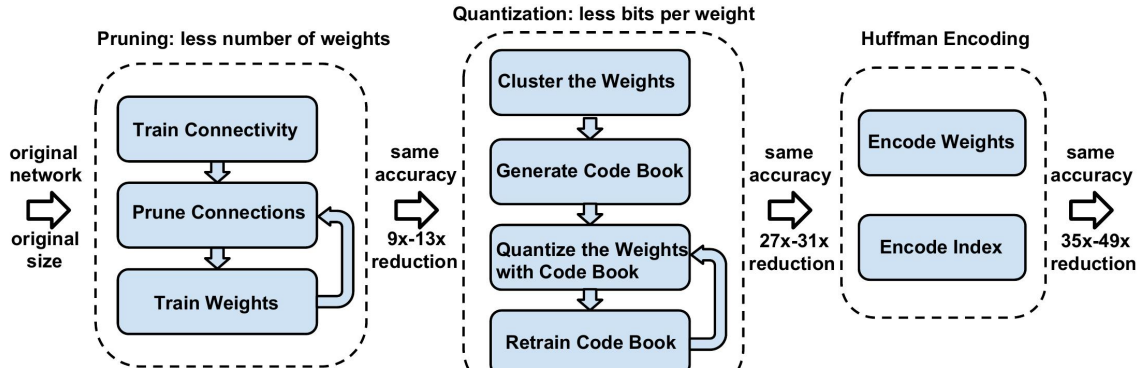
后续工作谱系
- NVIDIA 2:4 (Ampere, 2020)
- Cerebras / Graphcore 稀疏
- SparseGPT (2023) 直接继承

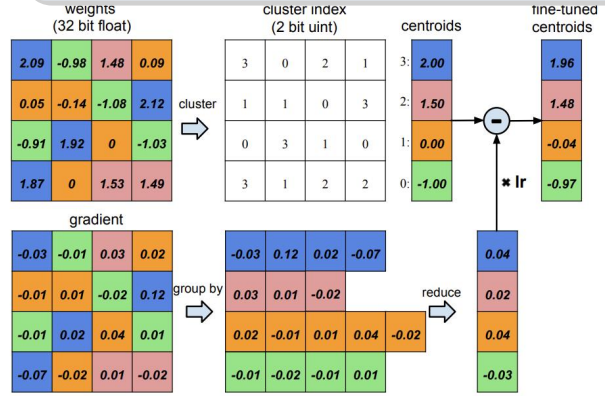
局限 (推动后续工作)

粒度局限
- 非结构化稀疏
- dense GPU 无加速
- 真实加速仅在 EIE / FPGA

训练局限
- 需多次剪-重训循环
- 单模型数天到数周
- LLM 70B 规模不现实

硬件局限
- EIE 是 ASIC, 通用性差
- 一个网络一套参数
- Part D 讲 GPU 如何跟进



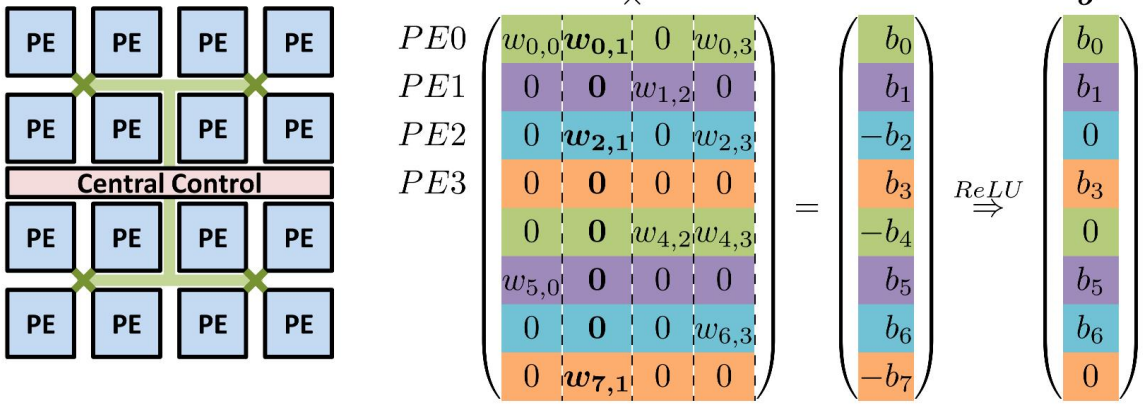


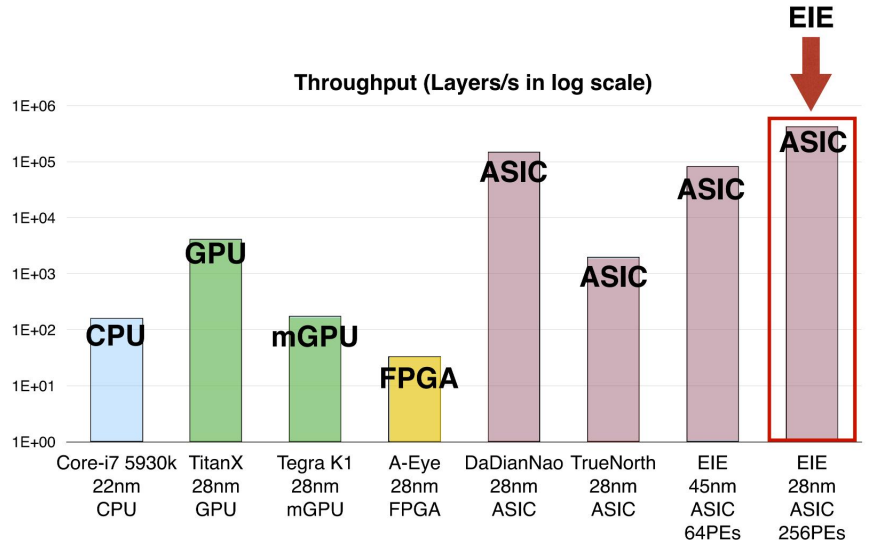
关键要点: Deep Compression 是剪枝进入 MLSys 的起点; EIE 是第一个稀疏 DNN 硬件。

Han et al., Deep Compression, ICLR 2016 (arxiv:1510.00149)

B.3 EIE: 首个稀疏 DNN 加速器 (ISCA 2016)

EIE = Efficient Inference Engine: 在 45nm / 28nm ASIC 上把稀疏 + 量化的收益兑现为真实推理加速。





架构 (左图)

空间并行
- 16×16 = 256 个 PE 矩阵排布
- 每个 PE 负责矩阵几行

PE 内部
- 本地 SRAM + FIFO
- 跳过零 activation (input)
- 跳过零 weight (via CSR)
- 权重共享解码 (5-bit codebook)

吞吐对比 (右图, log)

对比 CPU (i7-5930k)
- 速度 189×
- 能效 24,000×

对比 GPU (Titan X)
- 速度 13×
- 能效 3,400×

对比其他 ASIC
- 超过 DaDianNao, TrueNorth

历史定位

突破点
- 首次把稀疏做到硬件级
- 4 个零值处理通道

影响
- 直接启发 NVIDIA 2:4
- Graphcore, Cerebras 跟进
- 但 ASIC 通用性差
- GPU 等 4 年才有 2:4

关键要点: EIE 是算法-硬件协同设计的里程碑; 但 ASIC 代价太高, 通用 GPU 的答案是 2:4 (Part D)。

Han et al., EIE, ISCA 2016 (arxiv:1602.01528)

B.4 Lottery Ticket Hypothesis (Frankle & Carbin, 2019)

dense 网络中早就存在一个稀疏子网, 用 **同一初始化** 从头训练它 ⇒ 能达到原网络精度。

核心假设与证据

假设
- Dense 随机初始化网络中存在 "winning ticket" 稀疏子网
- 用相同初始化从头训练 ⇒ 匹配原网络精度

寻找方法 (IMP + Rewind)
- 1. 训练 dense 模型
- 2. 剪去小权重
- 3. **将剩余权重 rewind 到初始化值**
- 4. 从 rewind 值重新训练

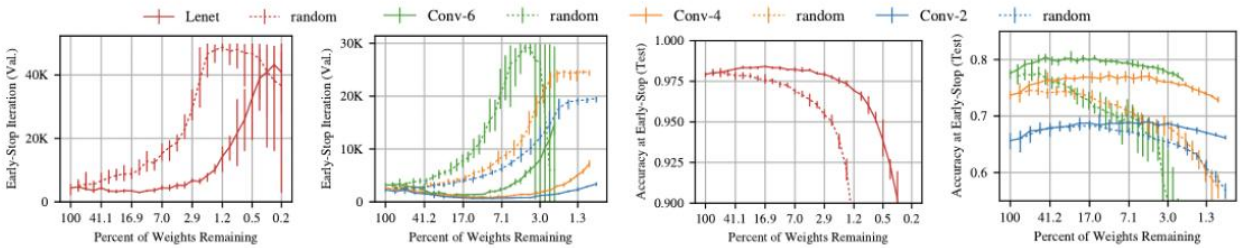
关键证据
- MNIST/CIFAR: 剪掉 95% 权重仍达 baseline
- Random reinit ⇒ 精度大幅下降

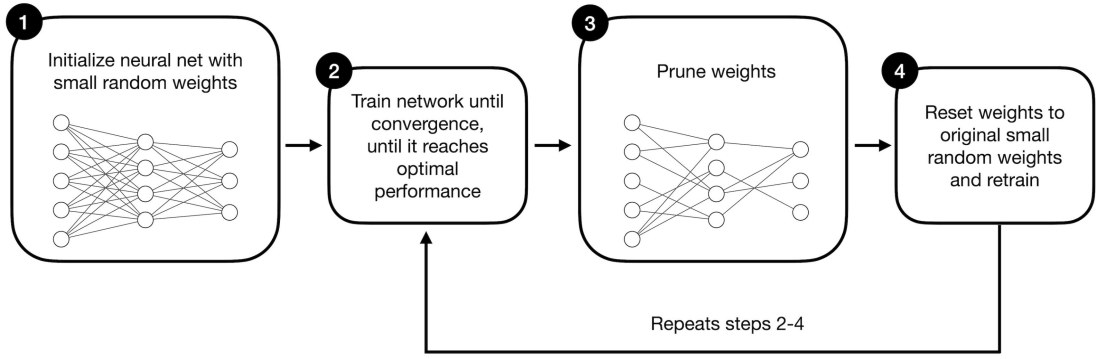
对本章的意义

观念更新
- 剪枝不是 "压缩", 而是 "发现"
- 稀疏结构本就存在, 只是被 dense 训练遮蔽

LLM 规模的问题
- IMP + rewind 需要多轮重训
- 70B 规模完全不现实
- L2 转向 one-shot + 轻量 heal

理论后续
- Lottery Ticket Hypothesis 分析的工具
- Stabilizing LTH (Frankle 2020)
- Early Bird Ticket (You 2020)





关键要点: 稀疏结构 "被发现" 不是 "被压出": 但 LLM 规模下发现成本过高, 促使 one-shot 方法兴起。

Frankle & Carbin, 'The Lottery Ticket Hypothesis', ICLR 2019 Best Paper (arxiv:1803.03635)

B.5 Liu 2019: 结构化剪枝的重新评估

结构化剪枝场景下, 从随机初始化重新训练小模型可以匹配甚至略超 fine-tune 的结果。

实验发现

四列对比 (在 CIFAR / ImageNet)

- Unpruned: 原始 dense 模型
- Fine-tuned: 传统剪枝 + 微调
- Scratch-E: 同结构从头训练 (同 epoch 数)
- Scratch-B: 同结构从头训练 (同 budget)

结果

- **Scratch-E ≈ Fine-tuned**
- **Scratch-B ⩾ Fine-tuned**
- 继承原权重 **并非必需**

重新定性: 剪枝 ≈ NAS

观念更新

- 结构化剪枝的本质是 **架构搜索 (NAS)**
- 价值在 "找到好的子结构", 不在 "继承权重"

对本章 L2 的影响

- **Sheared LLaMA / Minitron** 在此框架下工作
- 先用剪枝找结构, 再继续预训练 / KD 填权重
- L2 骨架中 "heal" 一格由此而来

Scratch-B slightly higher than Scratch-E in most cases. On ImageNet, both Scratch-B models are better than the fine-tuned ones by a noticeable margin.

Dataset	Model	Unpruned	Pruned Model	Fine-tuned	Scratch-E	Scratch-B
CIFAR-10	VGG-16	93.63 (±0.16)	VGG-16-A	93.41 (±0.12)	93.62 (±0.11)	**93.78** (±0.15)
	ResNet-56	93.14 (±0.12)	ResNet-56-A	92.97 (±0.17)	92.96 (±0.26)	**93.09** (±0.14)
			ResNet-56-B	92.67 (±0.14)	92.54 (±0.19)	**93.05** (±0.18)
	ResNet-110	93.14 (±0.24)	ResNet-110-A	93.14 (±0.16)	**93.25** (±0.29)	93.22 (±0.22)
			ResNet-110-B	92.69 (±0.09)	92.89 (±0.43)	**93.60** (±0.25)
ImageNet	ResNet-34	73.31	ResNet-34-A	72.56	72.77	**73.03**
			ResNet-34-B	72.29	72.55	**72.91**

Table 1: Results (accuracy) for L_1-norm based filter pruning (Li et al., 2017). "Pruned Model" is the model pruned from the large model. Configurations of Model and Pruned Model are both from the original paper.

ThiNet (Luo et al., 2017) greedily prunes the channel that has the smallest effect on the next layer's activation values. As shown in Table 2, for VGG-16 and ResNet-50, both Scratch-E and Scratch-B can almost always achieve better performance than the fine-tuned model, often by a significant margin. The only exception is Scratch-E for VGG-Tiny, where the model is pruned very aggressively from VGG-16 (FLOPs reduced by 15×), and as a result, drastically reducing the training budget for

读图要点

- 同一剪枝配置四列对比
- Scratch 列 ⩾ Fine-tuned 列
- 在结构化场景下

关键要点: 结构化剪枝 ≈ 架构搜索; 这一定性让 L2 的 prune-then-train 工业配方成为自然选择。

Liu et al., Rethinking the Value of Network Pruning, ICLR 2019 (arxiv:1810.05270)

PART C

系统问题

带宽瓶颈 · 算法 vs 硬件 gap · 压缩 ≠ 加速 · kernel 难 · 核心判据 (a)(b)

本课大纲 (进度)

√ Part A	√ Part B	▸ Part C	Part D	Part E
剪枝概念	历史代表	系统问题	硬件答案 2:4	总结 + 过渡

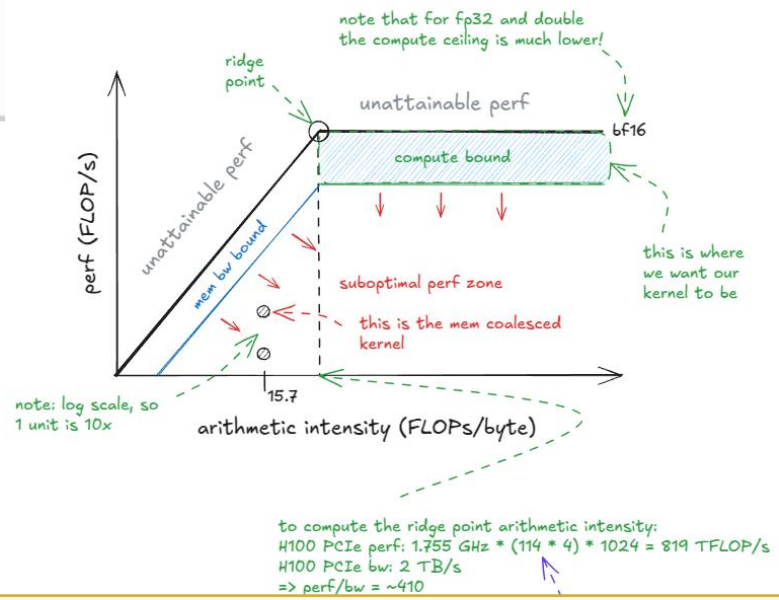
C.1 LLM 推理的瓶颈: 内存而非算力

LLM decode 是 memory-bound: 每 token 生成都要把整个权重张量流过一次内存层次。

70B 模型的数量级感知

权重大小
- LLaMA-70B FP16 = **140 GB**, 超过 H100 (80 GB)
- LLaMA-405B FP16 = 810 GB

Decode 内存流量
- 每生成 1 token, 整个权重需流过内存一次
- H100 HBM3 带宽: **3.35 TB/s**
- 理论下限 140 / 3.35 ≈ **42 ms/token**
- 即 **24 tok/s** 的物理上限 (单 GPU)



Roofline 视角: 两种 regime

Decode (batch=1, autoregressive)
- 算术强度 ~1 FLOP/byte
- **Memory-bound**, 瓶颈在 DRAM
- 优化目标: 减少字节数

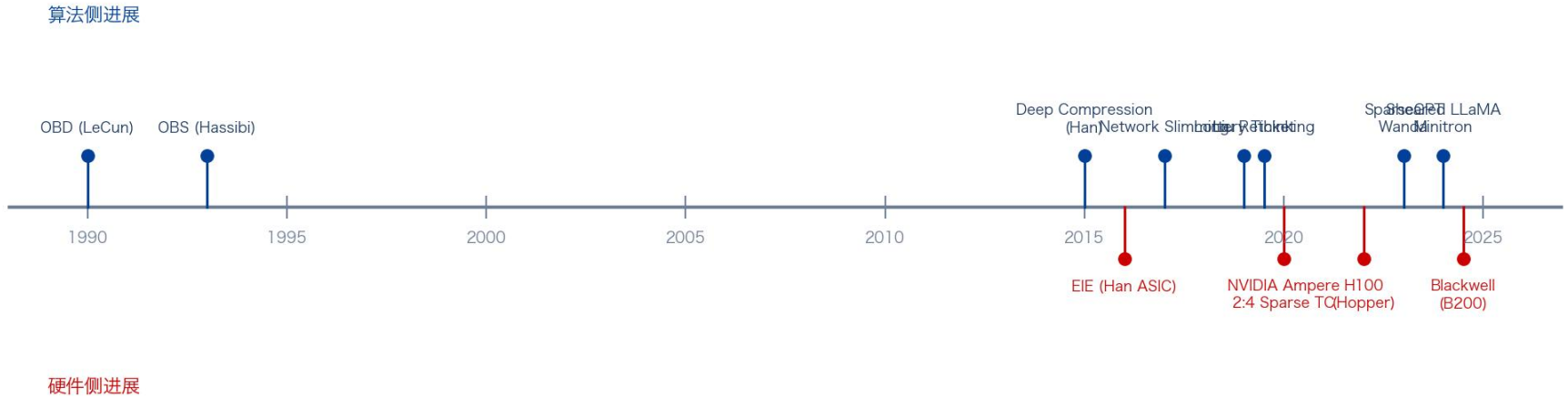
Prefill / Training
- 算术强度 >100 FLOP/byte
- **Compute-bound**, 瓶颈在 tensor core
- 优化目标: 提高 FLOP 利用率
- → 两种场景的剪枝收益完全不同

关键要点: LLM 推理不是算力不够, 而是带宽不够; 剪枝、量化、MoE 的共同价值是减少字节。

Roofline: Aleksa Gordic 'Inside NVIDIA GPUs' blog; 带宽: NVIDIA H100 whitepaper

C.2 算法侧 vs 硬件侧: 时间差与不匹配

算法侧 1990-2024 稀疏度/方法论持续演进; GPU 硬件直到 2020 才首次支持稀疏, 且只支持 2:4 一种 pattern。



算法端 (蓝色)

- 1990 OBD: 二阶 Hessian 剪枝
- 1993 OBS: 完整 Hessian
- 2015 Han Pruning: 95% 非结构化
- 2017 Slimming: channel 级结构化
- 2019 Lottery Ticket
- 2023 SparseGPT / Wanda: LLM one-shot
- 2024 Sheared / Minitron: 工业配方

硬件端 (红色)

- 2016 EIE: ASIC, 非通用
- 2020 NVIDIA A100: 首款 Sparse TC
  - 只支持 2:4 一种 pattern
- 2022 H100: 延续 2:4
- 2024 Blackwell B200
  - 仍以 2:4 为主, FP4 + 2:4 组合
- 其他: Cerebras/Graphcore 窄生态

Gap 的根本原因

硬件开发周期
- 架构设计 3-5 年
- 工艺流片 2-3 年

通用 GPU 约束
- 需要 “对多个工作负载都支持"
- 不能为单一算法定制
- → 必须选保守 pattern (2:4)

含义: 算法要“适应"硬件

关键要点: 算法超前硬件 5-10 年; 2:4 是硬件 "最小可行产品", 约束了所有后续方法的设计空间。

C.3 压缩率 ≠ 加速率: GPU 上的实测现实

直觉: 减权重 → 减字节 → 减延迟; **GPU 现实**: 非结构化稀疏 GEMM latency 几乎不变。

理论推演 (算法视角)

推理
- 50% 权重置零 ⇒ DRAM 字节数减半
- 若 memory-bound, 延迟应减半

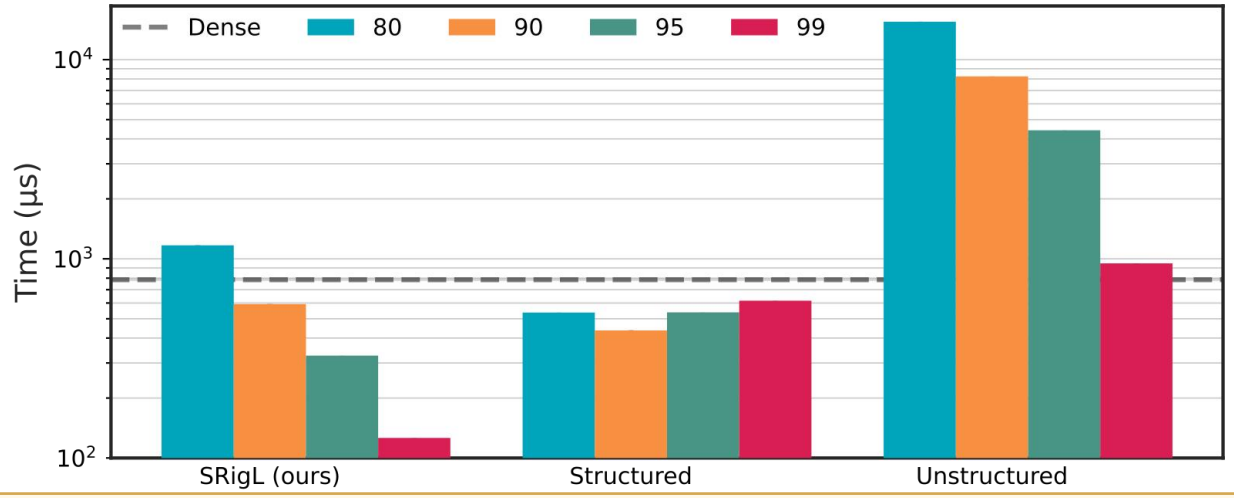
文献普遍报告
- 压缩率 10-50× (参数 / 存储)
- 许多论文只报告压缩率, 不报 latency

隐含假设
- "零值会被自动跳过" (实际不会)
- "带宽收益可自动兑现" (需硬件配合)

GPU 实测 (系统视角)

非结构化稀疏在 A100 实测
- 50% 稀疏: 延迟 **几乎不变**
- 80% 稀疏: 速度 ≈ dense
- 99% 稀疏: 勉强追上 dense

根本原因
- dense Tensor Core 不认零值
- 照样 16×16 MMA, 功耗也不降
- cuSPARSE 非结构化 kernel 有 metadata 开销
- 需要 99%+ 稀疏 metadata 才划算

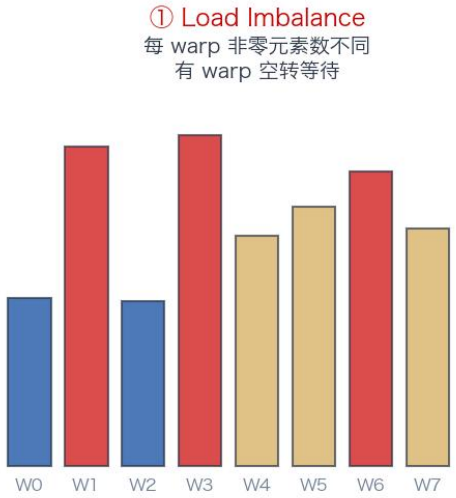


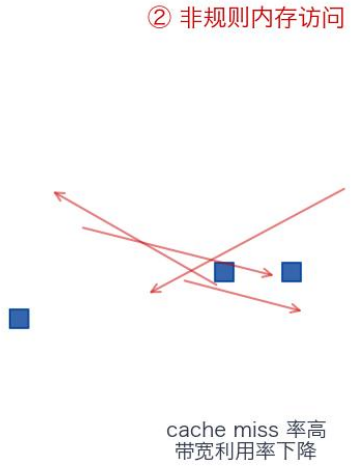
关键要点: "压缩比" vs "加速比" 不是同一量纲; 前者反映参数减少, 后者需要硬件能识别稀疏。

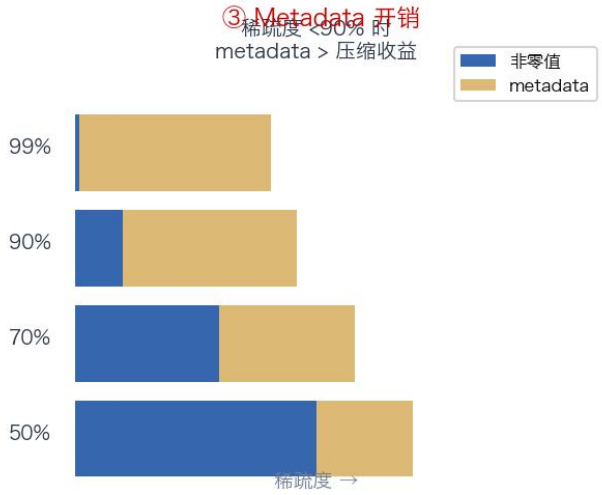
SRigL analysis, Calgary ML Lab (2024); cuSPARSE benchmarks

C.4 非结构化稀疏 kernel 为什么难加速?

dense GEMM 的三大优势在非结构化稀疏下全部被打破: 下图直观展示三个问题。







Dense GEMM 为什么快

① 数据重用
- Register tiling
- Shared memory reuse

② Warp 同构
- SIMD 执行, 无分支
- 32 线程同步

③ Cache-friendly
- Coalesced 访问

④ Tensor Core
- 16×16 专用硬件

非结构化的三问题 (上图)

① Load imbalance
- 每 warp 非零数不同
- 快 warp 等慢 warp

② 非规则访问
- 需 indirection 查位置
- Cache miss 率高

③ Metadata 开销
- 每元素 8-32 bit **位置索引**
- 稀疏度 <90%, 开销 > 收益

解决路径

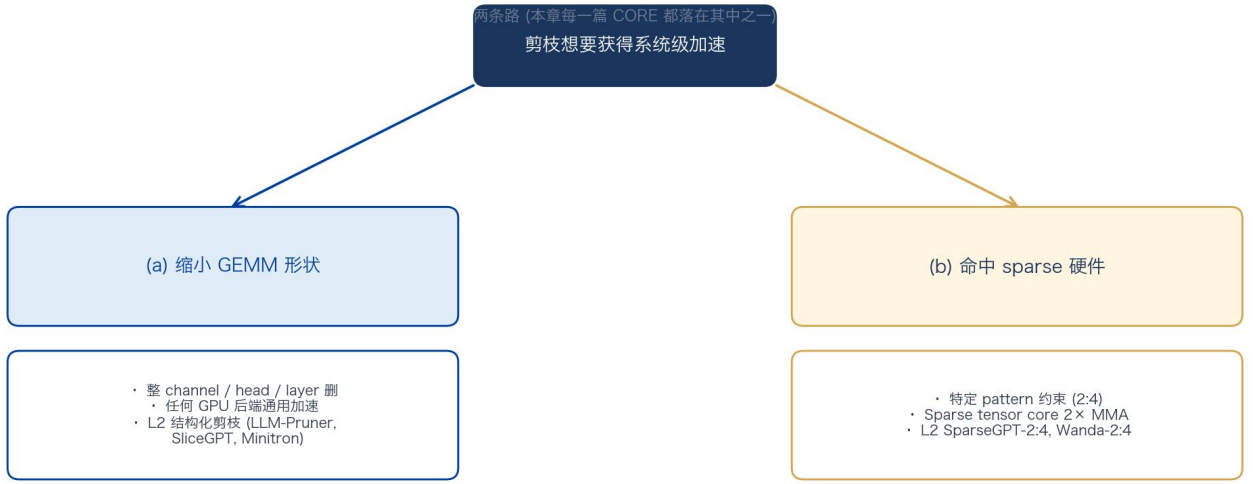
路径 A: 极高稀疏 (>99%)
- 学术研究, 生产难用

路径 B: 特殊硬件
- EIE ASIC, 但通用性差
- Cerebras, Graphcore

路径 C: 固定 pattern
- NVIDIA 2:4 (Part D)
- 约束换加速
- → **生产唯一可行**

关键要点: 非结构化稀疏在 GPU 上的困难是架构级的; 破解之道是引入 pattern 约束 (→ 2:4)。

C.5 本课的核心论点: 系统判据 (a) 或 (b)



(a) 缩小 GEMM 形状：更小的dense矩阵运算

做法
- 整 channel / head / layer 被物理删除
- GEMM 形状 M×N×K 对应变小

优点
- 任意 GPU 后端通用: cuBLAS / CUDA / PyTorch
- 不依赖特定硬件支持

代表方法
- LLM-Pruner (head/channel): Ma 2023 NeurIPS
- SliceGPT (hidden-dim): Ashkboos 2024
- Minitron (w/d/h/e): NVIDIA 2024, 3% FLOPs 超过 from-scratch
- → **L2 主线**

(b) 命中 sparse 硬件 pattern

做法
- 接受特定稀疏约束 (如 2:4: 每 4 选 2)
- 硬件层面 tensor core 识别 pattern 并 2× MMA

优点
- 不改变 GEMM 形状: 50% 带宽直接减半
- 硬件保证 2× throughput 上限

代表方法
- SparseGPT-2:4: Frantar 2023, OPT-175B 约 4h
- Wanda-2:4: Sun 2024, 几分钟级别完成
- NVIDIA ASP 训练 flow (D.8)
- → **Part D 全面展开**

关键要点: (a) 形状缩小 OR (b) 硬件命中: 至少满足一条, 剪枝才算 "有效"; 这是本章贯穿始终的评判标准。

PART D

硬件侧的稀疏支持: 2:4 为代表

NVIDIA Ampere 2:4 · Sparse Tensor Core · cuSPARSELt · N:M 泛化

本课大纲 (进度)

√ Part A	√ Part B	√ Part C	▸ Part D	Part E
剪枝概念	历史代表	系统问题	硬件答案 2:4	总结 + 过渡

D.1 NVIDIA Ampere 的 2:4 稀疏支持

2:4 半结构化稀疏 = 每 4 个连续权重中恰好 2 个为零; A100/H100/B200 均原生支持。

2:4 规则与 GPU 世代支持

规则
- 权重每 4 连续元素 **恰好** 2 个为零
- 记作 "2:4 semi-structured sparsity"
- 4 个元素有 C(4,2)=6 种合法模式

GPU 世代支持
- **A100 (Ampere 2020)**: 首次引入 Sparse TC
- **H100 (Hopper 2022)**: 延续 2:4, FP8 + 2:4
- **B200 (Blackwell 2024-25)**: FP4 + 2:4
- **未来**: NVIDIA 路线图 2:4 是 baseline

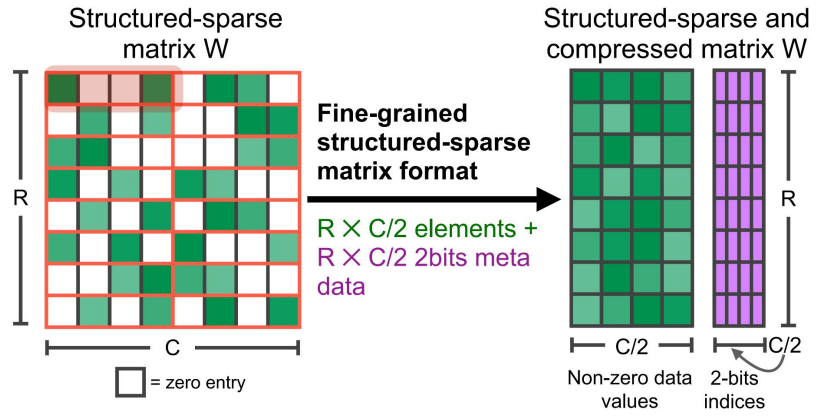
算法-硬件契约
- 算法: 接受 "每 4 选 2" 约束
- 硬件: 回馈 2× MMA throughput

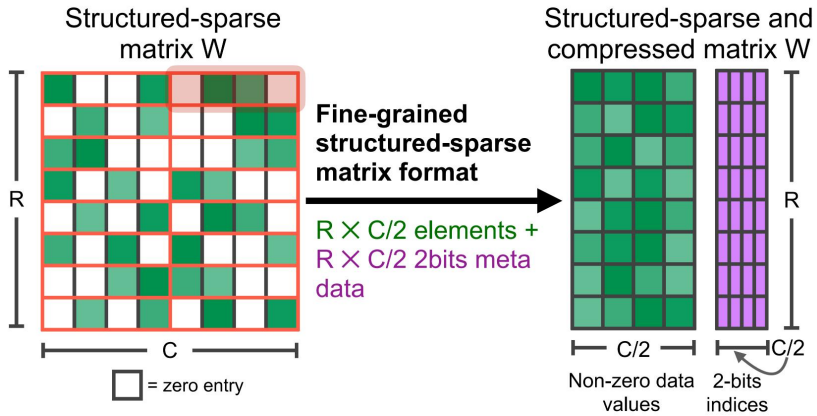
为什么 NVIDIA 选 2:4?

不是任意非结构化
- 零值位置任意 → metadata 开销显著增大
- 硬件解码逻辑复杂 (要在 MMA 前 select)

不是整列/整行 (完全结构化)
- 粒度太粗, 精度下降快
- 相当于 matrix 变小 (无需 sparse TC)

2:4 的三方面折中
- **硬件角度**: C(4,2)=6 种模式, 4-bit select 逻辑简单
- **存储角度**: 2-bit metadata, 开销低
- **算法角度**: 50% 稀疏, 精度可接受
- **生态角度**: 与 FP16/BF16/INT8/FP8 都兼容



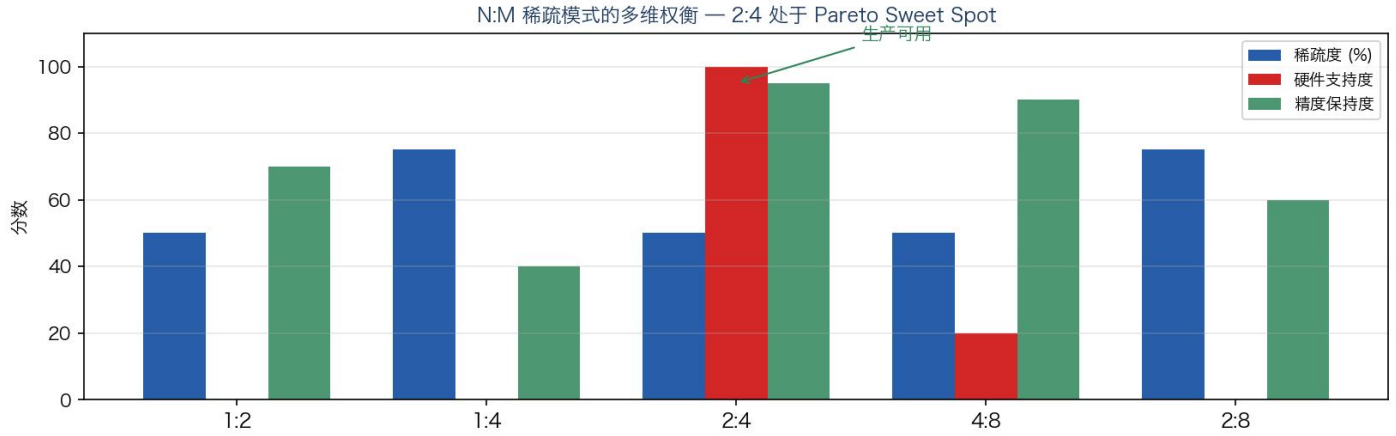


关键要点: 2:4 = NVIDIA 的 "硬件友好 + 算法可接受" 的三方折中; 下页深入权衡分析。

MIT TinyML 讲义 (Han Song); NVIDIA 'Accelerating Sparse DNNs' whitepaper

D.2 为什么是 2:4, 而不是其他 N:M?

NVIDIA 在 N:M 空间 (1:2 / 1:4 / 2:4 / 2:8 / 4:8) 做权衡分析, 2:4 是精度×硬件×生态的 Pareto sweet spot。



四个关键维度的权衡

稀疏度 (压缩比)
- 2:4 = 50%; 2:8 = 75%; 1:4 = 75%

Metadata 开销
- 2:4: 2-bit/非零元素, 合计 6.25% 开销
- 越细粒度 → metadata 开销越大

硬件解码复杂度
- 2:4 select 逻辑简单 (4 选 2 = C(4,2)=6 种)
- 1:4 有 4 种, 但稀疏度过高

精度保持
- 2:4 在大多数模型 <1% 精度掉
- 1:4 / 2:8 通常需要 fine-tune 补偿

NVIDIA 的选择逻辑

设计约束
- 通用 GPU 不能为单一 pattern 定制
- 需要在 FP16/BF16/INT8/FP8 多精度共存

4 是 Tensor Core 的自然对齐
- 16×16 MMA, K 维度以 4 为块
- 每 4 选 2 恰好对应半条 warp 计算

2:4 提供 2× 吞吐的关键约束
- 上限 2× MMA throughput
- 精度可工程化恢复
- 与已有 cuBLAS 兼容
- → **2:4 = 唯一生产可用的点**

关键要点: N:M 选择是 "稀疏度 × 硬件复杂度 × 精度" 的三方权衡; 2:4 是目前唯一生产可用的 Pareto 点。

NVIDIA: Mishra et al., 'Accelerating Sparse DNNs', arxiv:2104.08378

D.3 2:4 稀疏的存储格式

存储格式让硬件能快速定位非零位置, 是 2:4 的核心工程细节。

Fine-grained Structured-Sparse 格式

输入
- 稠密权重 W, 形状 R × C (如 4096 × 4096)

两个压缩数组
- **Values**: 非零值数组 shape R × (C/2)
- **Metadata**: 2-bit/元素, R × (C/2) × 2 bits
  (记录每 4 元素中哪 2 个保留)

总存储开销
- 权重: 50% + metadata: 6.25% = **53.1%**
- 几乎达到 2:1 压缩

Runtime 解码 (硬件层面)
- TC 读 metadata (2 bit)
- 动态 select 对应 activation 元素
- 只做 8 次 MMA 而非 16 次

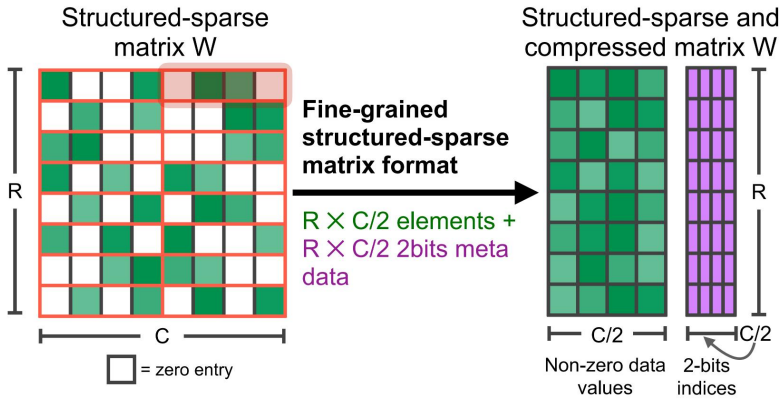
对内存层次的影响

DRAM 带宽
- 只读 50% 权重 + 6% metadata
- 带宽压力 **减半** (~47% 节省)

L2 Cache / SRAM
- 工作集减半, 命中率提升
- 减少 DRAM 二次访问

Register 使用
- TC A operand 减半
- 可支持更大 M×N tile

软件栈支持 (2024+)
- **cuSPARSELt** SDK (NVIDIA 官方)
- **PyTorch 2.1+** torch.sparse.to_sparse_semi_structured
- **NVIDIA ASP** 做 dense→2:4 转换



关键要点: 2:4 真正的价值在 DRAM 带宽减半; 2-bit metadata 让硬件能在 MMA 前快速 select。

Mishra et al., Accelerating Sparse DNNs; MIT TinyML 讲义 (Han Song)

D.4 Sparse Tensor Core 的 2× MMA 吞吐机制

Sparse Tensor Core 在硬件层面识别 2:4 pattern, 把 MMA 乘加次数减半从而 2× throughput。

Dense MMA (传统 Tensor Core)

输入
- A matrix (M×K), B matrix (K×N)

流程
- 每 cycle 做 16×16 MMA
- 计算全部 16 次乘加
- A 矩阵读全部元素

零值处理
- 零权重照样读入寄存器
- 照样参与乘加
- 带宽 / 算力都被浪费

PTX: mma.sync

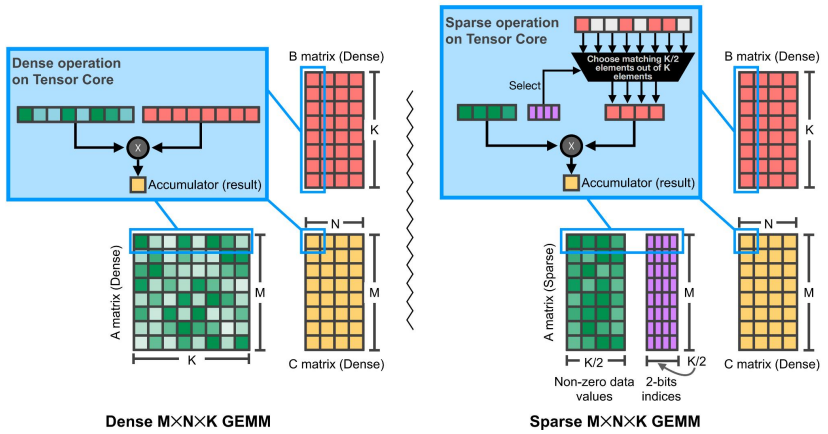
Sparse MMA (A100/H100/B200)

输入
- A = 2:4 稀疏 (Values + Metadata)
- B 仍 dense

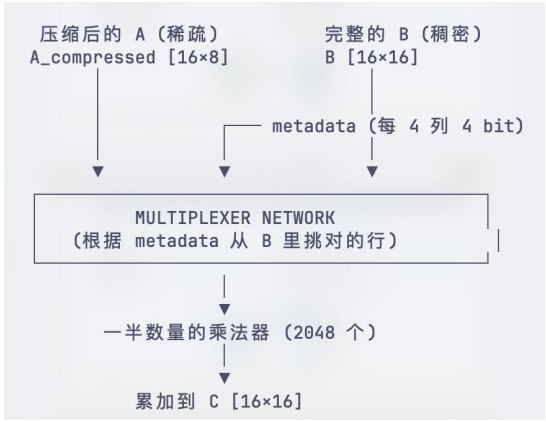
流程
- Select 单元读 metadata (2 bit)
- 从 B 中取 K/2 对应元素 (非零对应位置)
- 实际 MMA: 仅 8 次乘加 (跳过 2 个零)

结果
- 吞吐 = **2 × dense MMA**
- 支持 FP16 / BF16 / INT8 / FP8 / FP4
- FP32 不支持 (TC 只降精度累加)

PTX: mma.sp.sync.aligned.m16n8k32



Dense M×N×K GEMM　　Sparse M×N×K GEMM



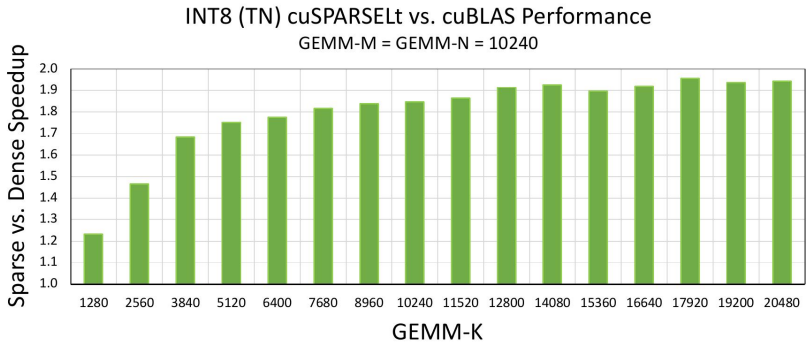
*读取 metadata,从 B 矩阵里**精准挑出** 那些"与 A 的非零元对应的行",送给乘法器。*

关键要点: Sparse Tensor Core 把 2:4 带宽优势在 GEMM 层面兑现为 2× 吞吐 (硬件保证)。

NVIDIA 'Accelerating Sparse DNNs' whitepaper (arxiv:2104.08378); MIT TinyML

D.5 cuSPARSELt 实测: 2× 是上限, 真实取决于 M/N/K

理论 2× 只在大 K 维度实现; 小 K 或小 M 时 metadata 与 kernel launch 开销占比上升。



ig. 3. Comparison of sparse and dense INT8 GEMMs on NVIDIA A100 Tensor Cor
arger GEMMs achieve nearly a 2× speedup with Sparse Tensor Cores.

A matrix (M×K), B matrix (K×N)

观察 (NVIDIA 自家实测)

大 K 场景
- K=20480: **接近 2.0×**
- K=10240: 约 1.8×

小 K 场景
- K=1280: 仅 **1.1-1.3×**
- K<128: 几乎无加速

精度选择
- FP16 / INT8 speedup 曲线相似
- FP32 不支持

为什么小 GEMM 加速不足?

kernel launch 固定开销
- 每次 GEMM 调用 ~10 µs
- 小 GEMM 本身也就几百 µs

metadata 解码开销
- 稀疏 MMA 比 dense 多一步 select
- K 小时摊销比例大

shared memory 效率
- 小 K 时 register tiling 摊销差

对 LLM 部署的指导

Linear 层 (K=d_model 4096+)
- 2:4 收益显著, 建议采用

Attention QKV (decode, K 小)
- 2:4 收益有限 (<1.3×)
- 带宽节省仍有价值

部署原则
- 先实测目标 M/N/K
- 不假设通用 2× 加速

关键要点: 2× 是上限, 实际收益取决于 M/N/K; LLM Linear 层收益大, 小 attention QKV 需实测。

NVIDIA: Mishra et al., 'Accelerating Sparse DNNs' arxiv:2104.08378 Fig 3

D.6 如何让已有模型达到 2:4?

给一个 dense 预训练模型, 有两条主流路径把它转成满足 2:4 约束。

路径一: NVIDIA ASP (有训练预算)

三步流程
- 1. Pre-train dense model (正常训练)
- 2. One-shot magnitude prune 到 2:4
  (每 4 权重保留 |W| 最大的 2 个)
- 3. Fine-tune with mask frozen
  (少量 epoch 恢复精度, 通常 10-20%)

工具
- NVIDIA `apex.contrib.sparsity.ASP`
- PyTorch 原生 sparse_semi_structured

适用场景
- ResNet/BERT 等中小型模型
- 精度损失 <0.1%

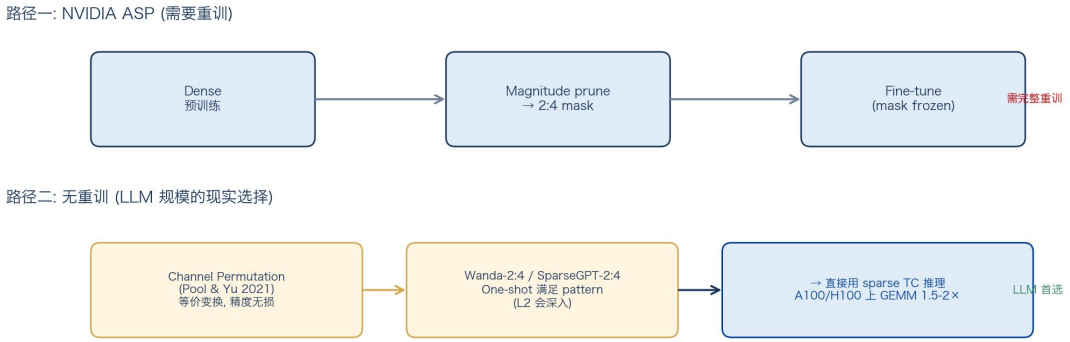
局限
- 70B LLM fine-tune 成本过高

路径二: 无重训方案 (LLM 主流)

① Channel Permutation
- Pool & Yu, NeurIPS 2021
- **重排列 channel, 每 4 连续位置自然满足 2:4**
- 等价变换, 精度无损, 无需重训

② SparseGPT-2:4 / Wanda-2:4
- 评分时加入 2:4 约束
- OBS/activation 框架下的投射
- One-shot, 无需训练数据
- OPT-175B 约 4h; LLaMA-7B 几分钟

为什么 LLM 都走这条?
- dense 70B 模型已经练出, 不想重训
- Calibration 数据 128 条就够

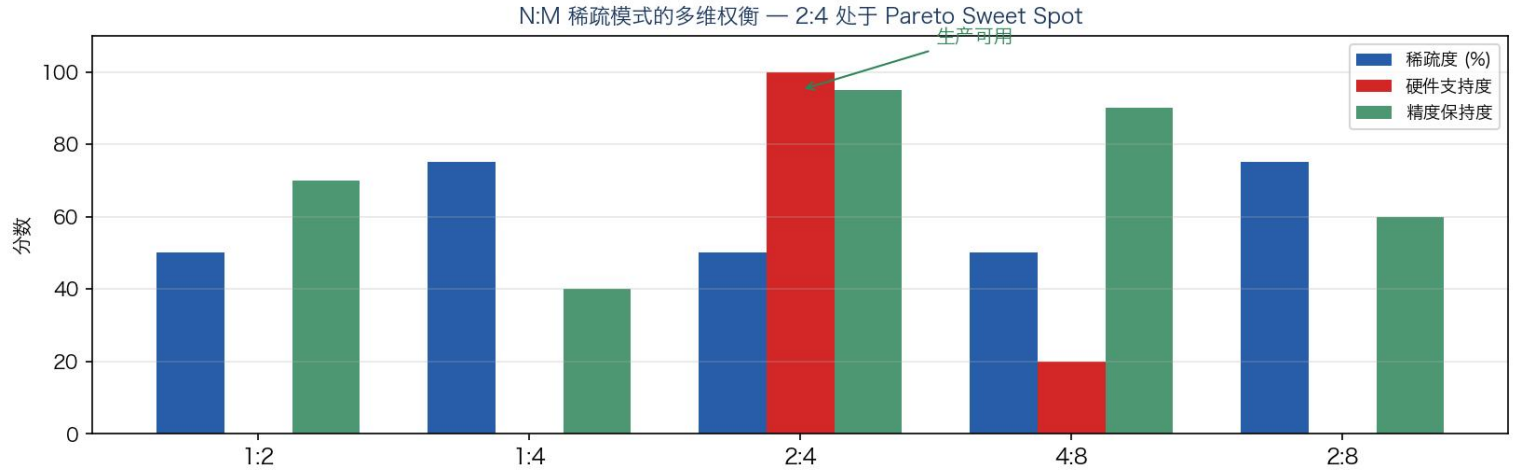


关键要点: LLM 规模下以无重训为主 (permutation + one-shot 评分), 这是 L2 配方的前置条件。

NVIDIA ASP library; Pool & Yu, NeurIPS 2021 (channel permutation); SparseGPT 2023

D.7 N:M 稀疏的泛化与下一代 GPU

2:4 只是 N:M 稀疏家族的一个点; 向前看还有 4:8 / 1:4 等选项, 但生产可用仍只有 2:4。



N:M 空间

4 是 TC 最小对齐单元
- 16×16 MMA 以 4 为块
- 2:4 = 50%, 2× 吞吐

其他学术点
- 4:8: 同 50%, 大 block
- 1:4: 75%, 理论 4×
- 2:8: 75%, 粗粒度

文献: Zhou 2021 ICLR

Blackwell (B200, 2024-25)

已确认
- 延续 2:4 sparse TC
- FP4 + 2:4 组合
- 对 Blackwell-GB200 NVL72 稀疏集成

未确认 (谨慎)
- 资料提到 4:8 / 8:16 支持
- 未见 NVIDIA 官方文档
- 本课以 2:4 为主

对本章 L2/L3 的含义

算法侧
- 2:4 是唯一稳定 target
- SparseGPT/Wanda 都有 2:4 变体

硬件演进方向
- 更大 block + 更低 N
- 更灵活的 pattern 支持

生态现状
- 生产只认 2:4
- 更激进留研究前沿

关键要点: 2:4 是今天的生产答案; 4:8/1:4 是未来方向; 算法-硬件协同设计持续演进。

D.8 训练中的 2:4: 从 ASP 到稀疏训练

一个 dense 预训练模型如何变成 2:4 部署? 工业主流、学术研究、未来探索三条路线。

路径一: NVIDIA ASP (需要重训)

路径二: 无重训 (LLM 规模的现实选择)

NVIDIA ASP (生产主流)

三步流程
- Dense pre-train (正常)
- One-shot magnitude prune
- Mask-frozen fine-tune

工具链
- cuSPARSELt SDK
- NVIDIA ASP library
- PyTorch 原生 torch.sparse

精度
- ResNet-50: <0.1% 精度掉
- BERT: 几乎无损

N:M 稀疏训练 (研究)

核心思想 (Zhou 2021)
- 从零开始强制 2:4
- Mask 动态更新 (每几步)
- STE gradient estimator

训练成本
- 吞吐 ≈ dense (硬件支持)
- 需调整 lr scheduler

验证现状
- ResNet / BERT 实测过
- LLaMA 70B 规模待验证

LLM 工业实践 (2024-2026)

LLM 预训练
- 仍以 dense 为主
- 稀疏稳定性未完全验证

LLM 推理部署
- dense 预训练 → SparseGPT/Wanda 无重训 → 2:4 部署

2025+ 探索方向
- 大厂内部: 预训练引入稀疏 mask
- 如 Google / Meta 闭源工作

关键要点: LLM 主流路径: dense 预训练 + 无重训稀疏化 (SparseGPT/Wanda); 稀疏训练仍是研究方向。

NVIDIA ASP; Zhou et al., Learning N:M Fine-grained Structured Sparsity, ICLR 2021 (arxiv:2102.04010)

PART E

总结 + 过渡 L2

块稀疏 · 其他硬件生态 · 三轴对比表 · L1 小结 · 预告 L2

本课大纲 (进度)

√ Part A	√ Part B	√ Part C	√ Part D	▸ Part E
剪枝概念	历史代表	系统问题	硬件答案 2:4	总结 + 过渡

E.1 块稀疏 (Block-Sparse): 另一条硬件友好的路径

2:4 主要服务 Linear 层的一般稀疏化; block-sparse 主要服务 attention mask 的结构化稀疏。

块稀疏的思想与优势

规则
- 权重矩阵分成 16×16 / 32×32 / 64×64 的 block
- 整块为零或整块保留
- 典型稀疏度: 50-90%

粒度定位
- 介于 2:4 (4 元素) 和整列剪枝之间

优势
- 块对齐 → 调用标准 dense GEMM (sub-block)
- 跳过整块 → 硬件利用率接近 dense
- 不需要 sparse tensor core
- 任何 GPU 后端都能跑 (CUDA/Triton)
- Block 越大, metadata 越便宜

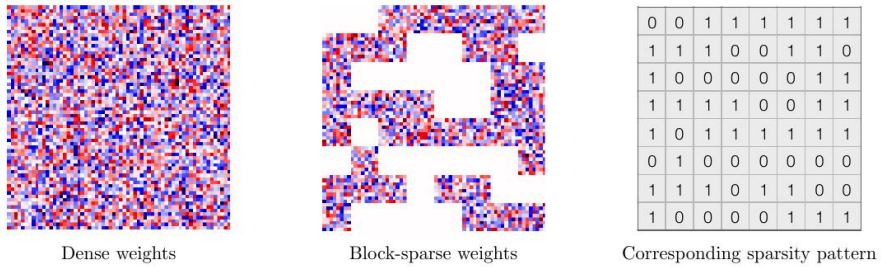
代表实现与应用

OpenAI Block-Sparse GPU Kernels (2017)
- Gray, Radford, Kingma
- LSTM/Transformer 时代首次
- 32×32 block, 手工优化 CUDA

Triton Block-Sparse Matmul
- OpenAI Triton DSL 编写
- 现代 PyTorch 生态首选
- 灵活 block size 支持

DeepSpeed Sparse Attention
- Long-context attention 的块稀疏 mask
- 固定 pattern: Local + Global + Random
- 用于 Megatron, GPT 长序列推理

Flash Attention 的变体
- Block-sparse attention (mask 稀疏)



0	0	1	1	1	1	1	1
1	1	1	0	0	1	1	0
1	0	0	0	0	0	1	1
1	1	1	1	0	0	1	1
1	0	1	1	1	1	1	1
0	1	0	0	0	0	0	0
1	1	1	0	1	1	0	0
1	0	0	0	0	1	1	1

Figure 1: Visualization of random dense and random block-sparse weight matrices, where white indicates a weight of zero. Our new kernels allow efficient usage of block-sparse weights in fully connected and convolutional layers, as illustrated in the middle figure. For convolutional layers, the kernels allow for sparsity in input and output feature dimensions; the connectivity is still dense in the

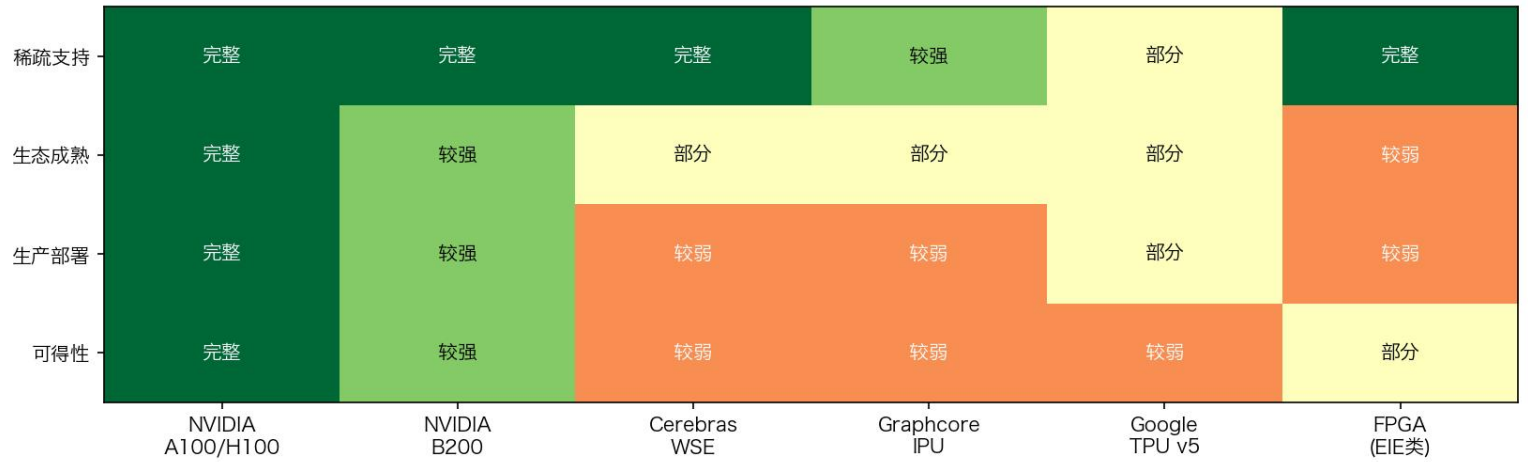
关键要点: 块稀疏补齐 2:4 做不到的场景 (long-context attention mask); 二者互补, 不是竞争。

Gray, Radford, Kingma, Block-Sparse GPU Kernels, OpenAI tech report 2017; DeepSpeed; Triton

E.2 其他硬件的稀疏支持现状

生态视角: 把 6 种稀疏硬件按 4 维度比较。为什么只有 NVIDIA 2:4 走通了?

稀疏硬件生产可用度矩阵 — 只有 NVIDIA 2:4 四项全绿



NVIDIA 2:4 (生产主流)

优势
- Pattern 固定, 解码简单
- cuSPARSELt + PyTorch 成熟

规模
- 95%+ 推理业务在 NVIDIA
- 文档 / 社区 / 生态全配套

代表客户
- OpenAI, Meta, Google, 大厂全接入

专用硬件 (Cerebras 等)

Cerebras WSE
- 细粒度稀疏支持
- 850K cores, 40 GB on-chip
- 主要大模型训练场景

Graphcore IPU
- Fine-grained sparsity
- 主攻 PyTorch/TensorFlow

局限
- 用户基础窄, 生态不全

TPU / ASIC / FPGA

Google TPU v5+
- 内部探索 N:M
- Google 自用为主

FPGA (Xilinx, Intel)
- 学术研究, 边缘设备

EIE / DaDianNao
- 历史贡献, 但通用性差
- 一模型一设计

关键要点: 稀疏硬件多样但生态窄; 本章聚焦 NVIDIA 2:4: 学生毕业后实际能用到的技术。

E.3 三轴 × 关键指标: 系统级对比表

把本堂课所有内容浓缩成一张对比表, 便于 L2/L3 参考。

	非结构化	2:4 半结构化	结构化 (channel/head/layer)
粒度	单权重	每 4 选 2	整行/通道/head/层
精度损失	最小	较小	较大, 需 heal
硬件加速 (GPU)	几乎无 (除非自定义 kernel)	2× (Ampere+ sparse TC)	所有后端通用
实现成本	低 (one-shot 即可)	中 (需 pattern 约束)	中-高 (依赖图 + heal)
适用场景	研究 / 极高稀疏	生产部署主力	生产 / L2 工业配方
LLM 代表	SparseGPT, Wanda (非结构变体)	SparseGPT-2:4, Wanda-2:4	LLM-Pruner, SliceGPT, Minitron
判据 (C.5)	都不满足 → 无真实加速	(b) 命中硬件	(a) 缩小 GEMM

关键要点: 生产部署常用 2:4 + 结构化; 非结构化主要用于研究或作为上限参考。

E.4 L1 小结: 两条判据与过渡 L2

剪枝要真正实现 LLM 推理加速, 需满足至少一条系统判据; 本章后续两个 lecture 是这条判据的应用。

L1 的三大成果

① 剪枝的算法框架
- Which × How × When 三维空间
- 3 种粒度 × 4 种评分 × 2 种流程
- 方法地图: Han / Wanda / SparseGPT / LLM-Pruner

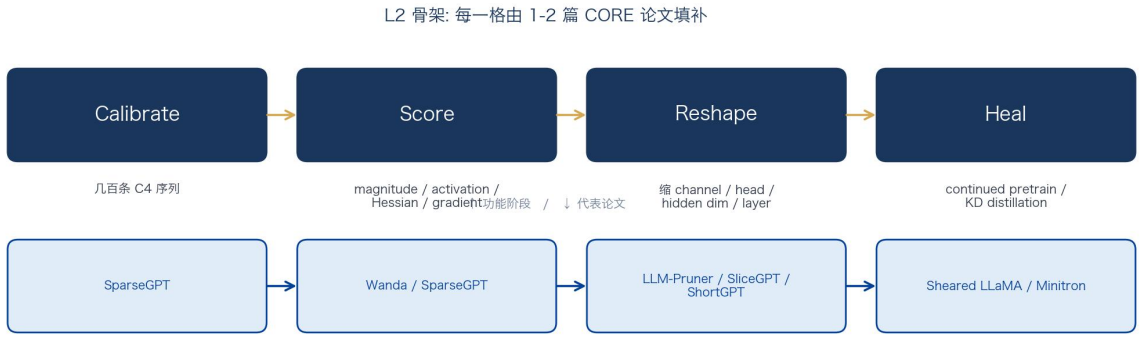
② 两条系统判据
- **(a)** 缩小 GEMM 形状
  任意 GPU 后端通用加速
- **(b)** 命中硬件 sparse pattern
  2:4 + Sparse TC, 2× MMA 吞吐

③ 硬件生态
- NVIDIA 2:4 生产可用
- Cerebras/Graphcore 研究为主
- EIE ASIC 是历史起点

L2 / L3 的定位

L2 静态 LLM 剪枝 (下一课)
- 70B 规模下的统一方法论
- 骨架: calibrate → score → reshape → heal
- 代表: SparseGPT / Wanda / LLM-Pruner / SliceGPT / Sheared LLaMA / Minitron
- 主要落判据 (a), 部分 (a)+(b)

L3 动态剪枝 (第三课)
- 2024-2026 增长点: input-dependence
- 方向 A: 动态权重 (Deja Vu, PowerInfer)
- 方向 B: KV-cache (H2O, StreamingLLM, Quest)
- 闭环: static + dynamic 组合



关键要点: L1 确立判据 (a)(b); L2 给出 70B 规模落地配方; L3 处理 input-dependent 增长点。

第二节课

LLM 规模下的统一方法论

骨架: calibrate → score → reshape → heal

本课路线图：LLM 规模的统一方法论

主线问题: 70B 模型无法重训, 是否存在一条统一配方?

主线问题: 70B 模型无法重训, 是否存在一条统一配方?

A/B 骨架前两个环节: Calibrate + Score

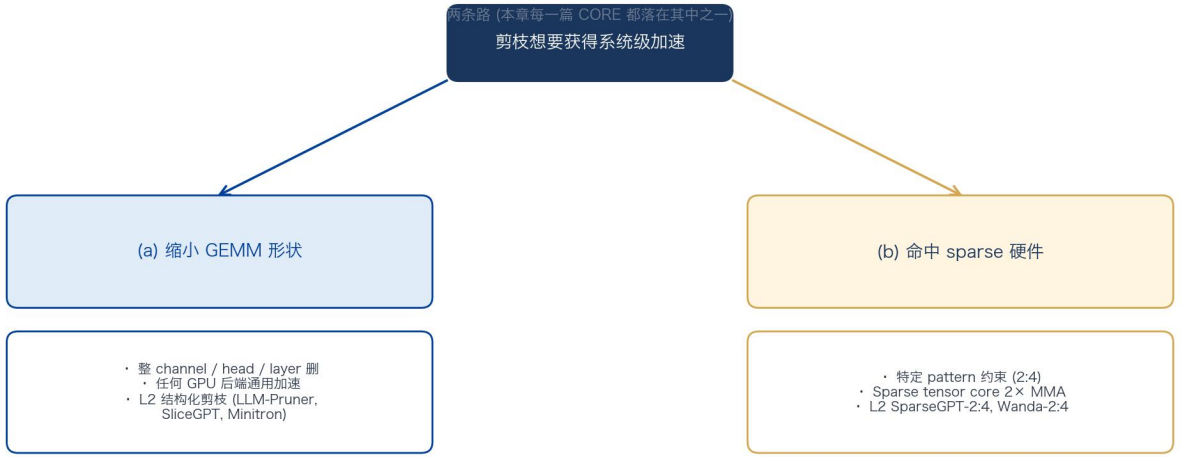
- 用 128 条 C4 数据做前向 calibration
- SparseGPT (Hessian OBS, 2023)
- Wanda (activation-based, 2024)
- OWL 层级稀疏 budget (2024)

C/D 后两个环节: Reshape + Heal

- Reshape: LLM-Pruner / SliceGPT / Minitron
- Heal: Sheared LLaMA / Minitron KD / EoRA
- 组合: SliM-LLM (prune+quant)
- 工业部署: Llama-3.1-Minitron-4B

L1 回顾: 两条判据 + LLM 的新约束

L1 建立判据 (a)(b); L2 要把它们具体落到 70B 规模的真实 LLM 上。



判据 (a): 缩小 GEMM 形状

- 删除整 channel / head / layer
- 任意 GPU 后端通用加速 (cuBLAS)
- 代表: LLM-Pruner, SliceGPT, Minitron
- **L2 的 reshape 环节 = (a) 的落地**

判据 (b): 命中 sparse 硬件

- 2:4 pattern + Sparse TC, 2× MMA
- DRAM 带宽减半 (硬件保证)
- 代表: SparseGPT-2:4, Wanda-2:4
- L2 多数方法都有 2:4 变体

L2 = 无重训 + 仅 calibration 约束下, 把 L1 两条判据落到 LLM 规模。

1.1 LLM 规模下的三大约束

这三条约束共同决定 L2 所有方法的设计空间。

① 无重训

- 70B 预训练 ≈ $10M
- 8K H100 × 数周
- IMP / 重训完全不可行
- 必选 one-shot

② 仅 calibration

- 128-2048 条 C4 数据
- 仅前向, 无反向
- 评分只能用 |W|, |X|
- 不用训练数据

③ Shape-axis 付房租

- 4 条 axis: w/d/h/e
- 每条 axis 都耗精度
- 无免费维度
- 组合使用最优

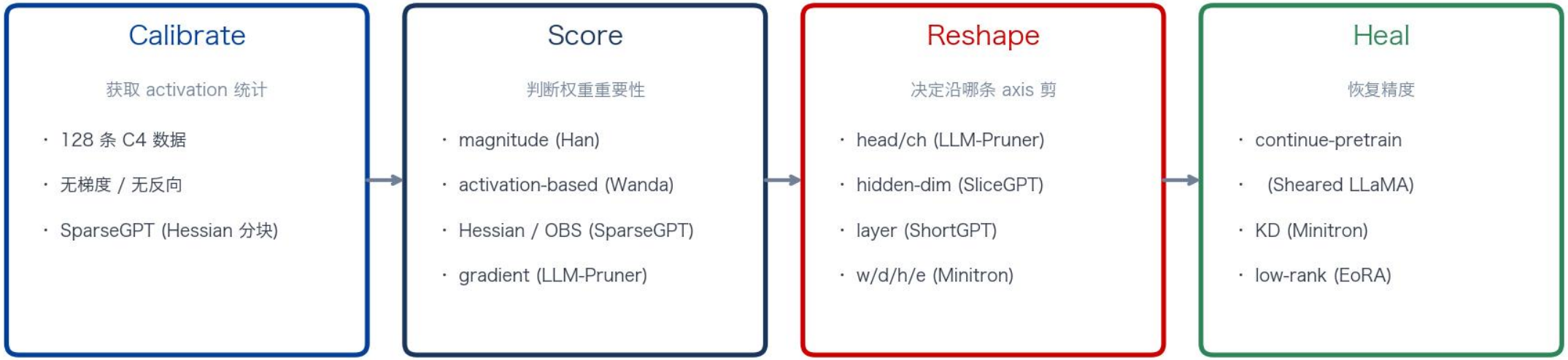
三约束的协同含义

- **one-shot 必选**: 70B 重训成本 $10M, IMP 不现实
- **forward-only 评分**: 128-2048 条 C4, 不能用梯度
- **组合式 shape axis**: 没有免费 axis, 组合 (width+depth) 最优

三约束 → one-shot + forward-only + 组合 shape axis: 定义 L2 方法空间边界。

1.2 L2 统一骨架: calibrate → score → reshape → heal

全课锚点: 每篇 CORE 论文填骨架的一个环节 (或跨多格)。



一条骨架, 4 个格子 — 每篇 CORE 论文填一格 (或多格)

每格要回答的问题

- Calibrate: 少量前向数据收集 XX^T 或 $\|X\|_2$ 统计
- Score: 哪些权重不重要? magnitude / activation / gradient / Hessian
- Reshape: 哪条 axis 物理缩小? width / depth / head / hidden-dim
- Heal: 如何守住精度? continue-pretrain / KD / 训练-free 低秩

PART A

Calibrate + Score

少量数据做 calibration · SparseGPT · Wanda · OWL (2024 层级预算)

本课大纲 (进度)

▸ Part A	Part B	Part C	Part D
Calibrate + Score	Reshape	Heal	组合 + 局限 + 收尾

A.1 Calibrate: 只用 128 条数据够吗?

答案: 够: 这是 L2 one-shot 剪枝能成立的技术基础。

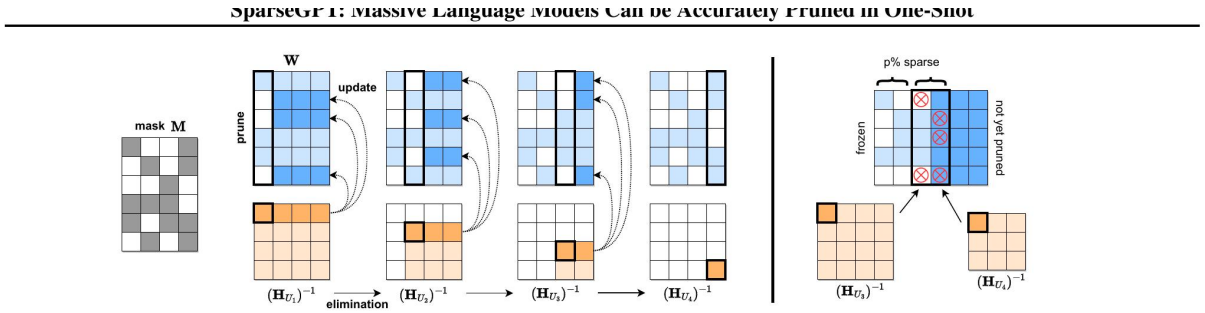


Figure 4. [Left] Visualization of the SparseGPT reconstruction algorithm. Given a fixed pruning mask $\mathbf{M}$, we incrementally prune weights in each column of the weight matrix $\mathbf{W}$, using a sequence of Hessian inverses $(\mathbf{H}_{U_j})^{-1}$, and updating the remainder of the weights in those rows, located to the "right" of the column being processed. Specifically, the weights to the "right" of a pruned weight (dark blue) will be updated to compensate for the pruning error, whereas the unpruned weights do not generate updates (light blue). [Right] Illustration of the adaptive mask selection via iterative blocking

为什么少量数据够?

- LLM activation 自带**低秩结构**
- FFN 激活 80% 被 ReLU 置零
- Attention 输出集中在少数 head
- 激活协方差 XX^T 主成分集中
- 128 条 ≈ 10K 条 统计差距 <1%

采集什么?

- **Forward-only**, 避 70B 反向开销
- Wanda: $\|X_j\|_2$ (column norm)
- SparseGPT: XX^T (covariance 分块)
- LLaMA-7B 采集: ~30s on A100

128 条 × forward-only = LLM one-shot 剪枝的技术基础, 全课都建立在此之上。

Wanda (arxiv:2306.11695); SparseGPT (arxiv:2301.00774)

A.2 SparseGPT: 把 OBS 复兴到 LLM 规模

1993 年 Hessian OBS 复兴: 分块求解, OPT-175B 4h, 填 calibrate+score 两格。

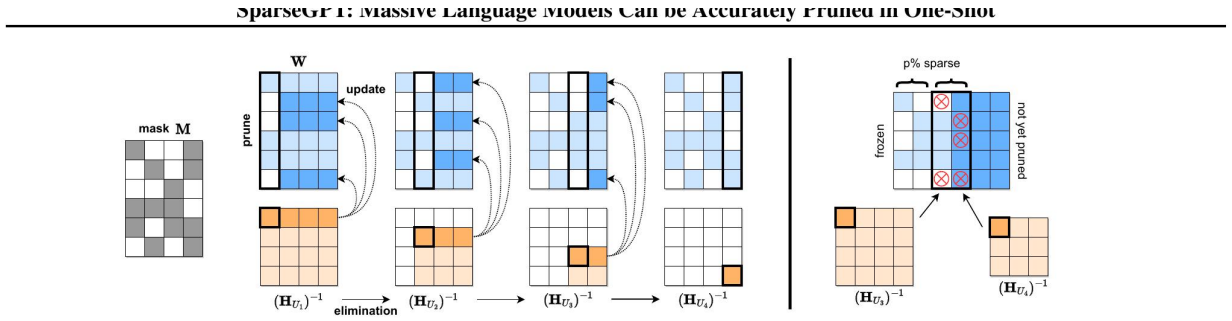


Figure 4. [Left] Visualization of the SparseGPT reconstruction algorithm. Given a fixed pruning mask $\mathbf{M}$, we incrementally prune weights in each column of the weight matrix $\mathbf{W}$, using a sequence of Hessian inverses $(\mathbf{H}_{U_j})^{-1}$, and updating the remainder of the weights in those rows, located to the "right" of the column being processed. Specifically, the weights to the "right" of a pruned weight (dark blue) will be updated to compensate for the pruning error, whereas the unpruned weights do not generate updates (light blue). [Right] Illustration of the adaptive mask selection via iterative blocking.

算法核心

- 基于 OBS 公式: ΔW_rest = -W_M · (H⁻¹)_{M,rest}
- 按 column block (128) 分块求解
- 避开全局 d^3 求逆, $O(d \cdot col^3)$
- 同步更新余权重补偿误差
- 支持非结构化 + 2:4 投射

核心数字

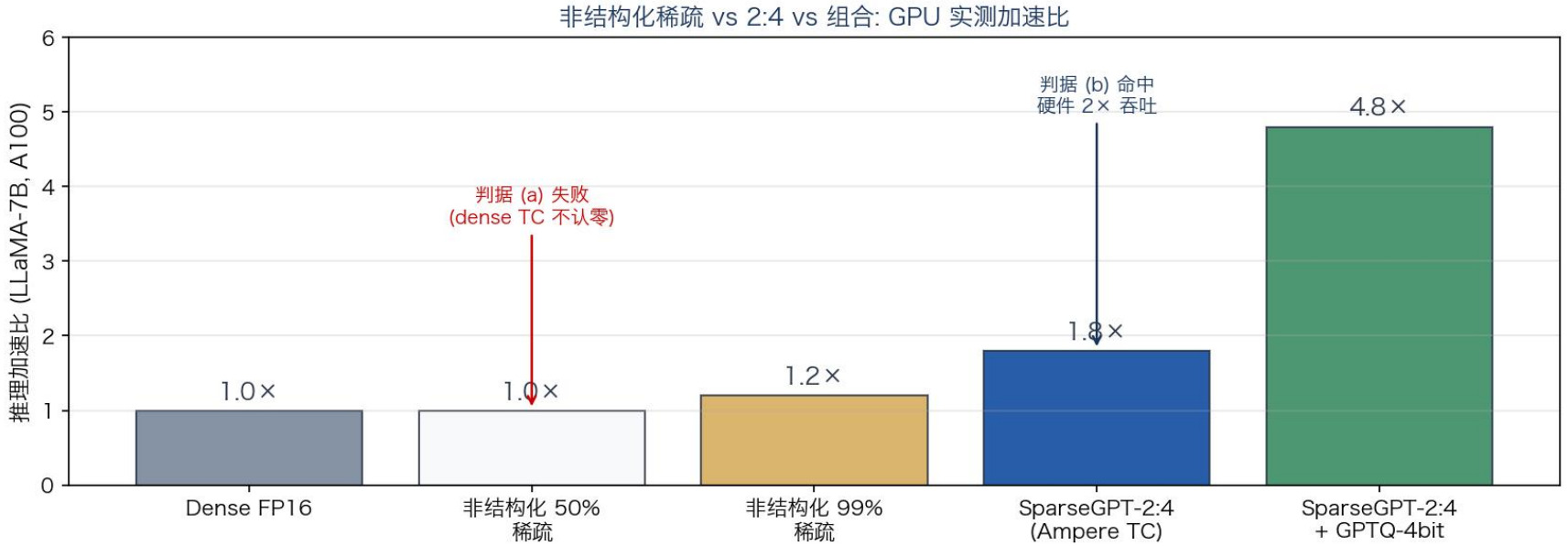
- OPT-175B 50% sparse: **~4h on A100**
- LLaMA-7B: ~30 min
- PPL 损失 <1% @ 50% 稀疏
- 2:4 变体: PPL +0.5-1
- 仅需 **128 条** calibration

SparseGPT 首次在 LLM 规模实现 OBS: calibrate + score 一次搞定, 为 L2 定型。

Frantar & Alistarh, SparseGPT, ICML 2023 (arxiv:2301.00774)

A.3 SparseGPT-2:4: L1 判据 (b) 的 LLM 实证

同一 OBS 框架投射到 2:4: 在 70B 模型上兑现 Sparse TC 的 2× 吞吐。



2:4 投射做法

- OBS 选 mask 时加入 2:4 约束
- 每 4 位 group 内从 C(4,2)=6 组合择优
- 对比非结构化: 多 0.5-1 PPL 代价

端到端加速 (LLaMA-70B)

- Linear 层 GEMM: **1.5-1.8× speedup**
- Prefill: 1.5×; Decode: ~45% 带宽节省
- 内存: 140 GB → 80 GB

SparseGPT-2:4 首次在 70B 闭合 L1 判据 (b); 2:4 从此成为 LLM 实际可用方案。

A.4 Wanda: 3 行 PyTorch 的答案

"不要 Hessian 也能做 LLM 剪枝": activation-based 评分, 接近 SparseGPT 精度 (ICLR 2024)。

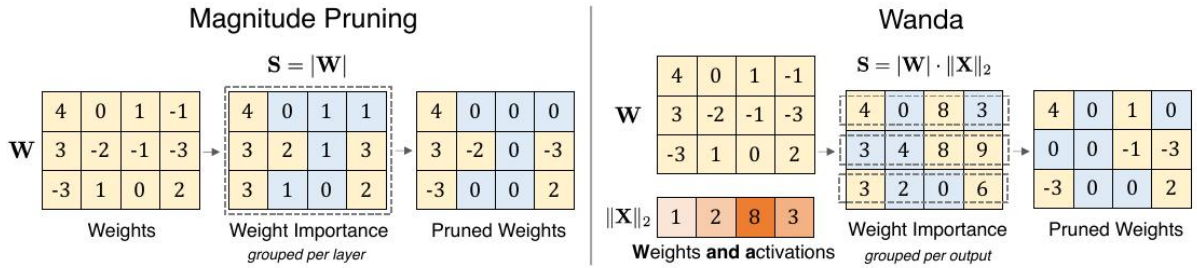


Figure 1: Illustration of our proposed method Wanda (Pruning by **W**eights **and** activations), compared with the magnitude pruning approach. Given a weight matrix $\mathbf{W}$ and input feature activations $\mathbf{X}$, we compute the weight importance as the elementwise product between the weight magnitude and the

评分公式 (见上图)

- 公式: $s_{ij} = |W_{ij}| \cdot \|X_j\|_2$
- 权重 × activation norm
- 对比 magnitude: 考虑了激活贡献
- 天然契合 LLM 的 activation outlier
- 分组: per output row (保稀疏度一致)

系统优势

- LLaMA-7B: **< 5 min** (SparseGPT ~30 min)
- 精度持平 (PPL 7.26 vs 7.22)
- 代码 ~30 行 (SparseGPT ~200 行)
- 无 Hessian 分块, 无余权重更新
- **2024 年 LLM 剪枝 baseline**

Wanda 极简化 calibrate + score: 成为 2024 年 LLM 剪枝的 baseline 方法。

图: Sun et al., Wanda, ICLR 2024 (arxiv:2306.11695) Fig 1

A.5 SparseGPT vs Wanda: 二选一

两种都可落地; 选型取决于 精度 / 成本 / 工程复杂度。

维度	SparseGPT	Wanda	胜者
评分方法	Hessian (OBS)	$\|W\| \cdot \|\|X\|\|_2$	平局
权重更新	同步更新余权重	不更新	SparseGPT 严
LLaMA-7B 耗时	~30 min	**~5 min**	Wanda
OPT-175B 耗时	~4 h	**~30 min**	Wanda 8×
PPL @ 50%	7.22	7.26	持平
PPL @ 2:4	12.0	12.8	SparseGPT 略好
易扩展	中	**高**	Wanda

Wanda 6-8× 快, 精度几乎持平, 代码简单: 2024 年 LLM 剪枝首选。

A.6 OWL: 非均匀层级稀疏度 (ICML 2024)

观察: 每层 outlier 密度不同 → 不同层应分配不同 sparsity budget。

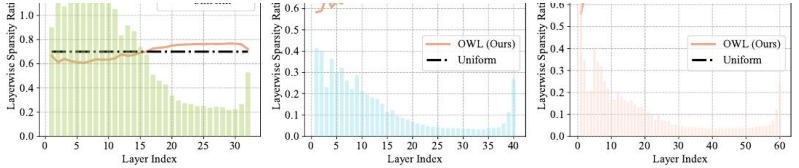


Figure 1: The demonstration of the OWL layerwise sparsity and Uniform layerwise sparsity at 70% sparsity. The bar chart in the background corresponds to the Layerwise Outlier Distribution (LOD), as elaborated in Section 3.2

dation (Frantar & Alistarh, 2023; Sun et al., 2023; Jaiswal et al., 2023b; Ma et al., 2023). SparseGPT (Frantar & Alistarh, 2023) addresses the challenge of LLM pruning from the perspective of layerwise reconstruction problem. In this context, the primary goal is to minimize the output discrepancy in terms of the reconstruction error between dense and sparse LLMs. It adopts an iterative strategy to handle the computational hurdle posed by the row-Hessian problem. Specifically, it employs the Optimal Brain Surgeon (OBS) algorithm (Hassibi et al., 1993) to selectively prune and update weights in a column-wise manner. Wanda (Sun et al., 2023), on the other hand, introduces a novel pruning metric that takes into account both the weight magnitudes and their corresponding input activations. Remarkably, it

Three reasons behoove us to pose the above research question: *First*, it is widely acknowledged that within Transformer architectures, certain components hold greater significance than others, and thus, they merit distinct treatment during the pruning process (Wang & Tu, 2020; Bhojanapalli et al., 2021); *Second*, a consensus view has been reached in computer vision that non-uniform layerwise sparsity typically achieves stronger results than uniform sparsity (Liu et al., 2022a; Lee et al., 2020); *More importantly*, LLMs demonstrate astonishingly emergent behaviors (Wei et al., 2022; Schaeffer et al., 2023; Dettmers et al., 2022) as model size continuously scales up, a phenomenon distinct from smaller-scale language models. These emergent behaviors offer fresh insights into the domain of LLM pruning. For

方法

- 测量每层 **LOD** (Layerwise Outlier Distribution)
- LOD 大 → 该层敏感, 给低 sparsity
- LOD 小 → 该层冗余多, 可高 sparsity
- 2-3 行代码插在 Wanda 前即可

效果

- LLaMA 70% 稀疏: **−4 PPL** vs 均匀 Wanda
- + SparseGPT: **−6.8 PPL**
- DeepSparse 部署加速 **2.6×**
- 可插入任意 scoring 方法

OWL 在 score 环节加 "层级预算" 维度: 2024 年对 Wanda 最实用增强。

Yin et al., OWL, ICML 2024 (arxiv:2310.05175)

PART B

Reshape: 沿哪条 axis 物理缩小模型?

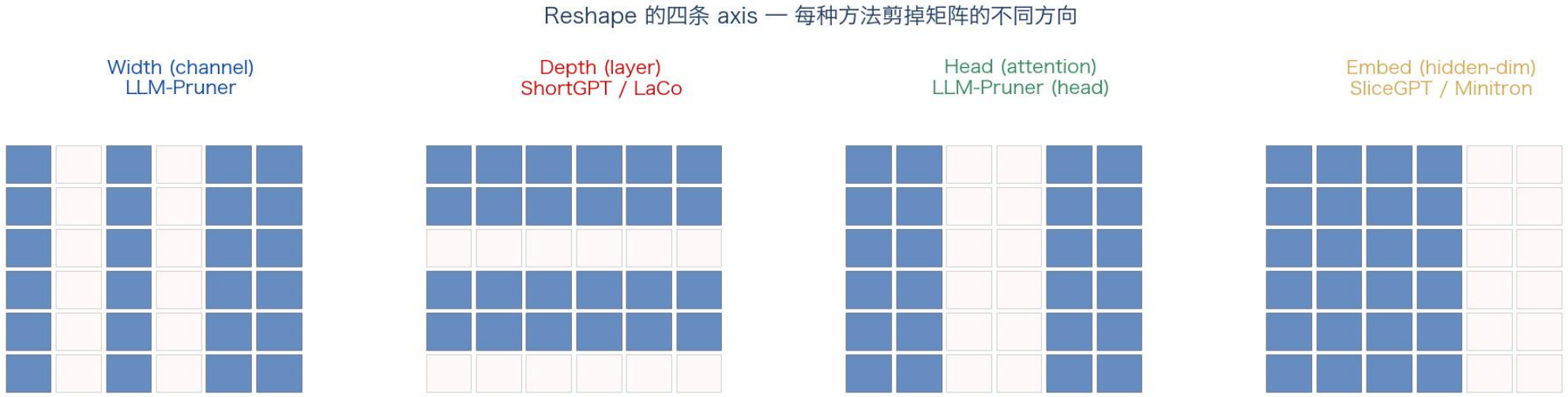
width / depth / head / embed 四条 axis · LLM-Pruner · SliceGPT · ShortGPT · LLM Surgeon · Bonsai

本课大纲 (进度)

√ Part A	▸ Part B	Part C	Part D
Calibrate + Score	Reshape	Heal	组合 + 局限 + 收尾

B.1 Reshape 的四条 axis

Reshape 不是单一选项: 四条 axis 各有精度 × 加速 Pareto 取舍。



四条 axis 的代表方法

- **Width**: LLM-Pruner, Bonsai, Minitron
- **Depth**: ShortGPT, LaCo
- **Head**: LLM-Pruner 变体
- **Embed**: SliceGPT, Minitron

系统收益差异

- Width → Linear GEMM 小
- Depth → 整层省, 端到端快
- Head → Attention 计算省
- Embed → **KV-cache 也缩** (long-context)

后 6 页展开每条 axis 的代表方法, 最终导出 Minitron 的组合式工业配方。

B.2 LLM-Pruner: 依赖图 + gradient (NeurIPS 2023)

首篇把结构化剪枝做到 LLM 规模: 依赖图解决耦合, gradient 做评分。

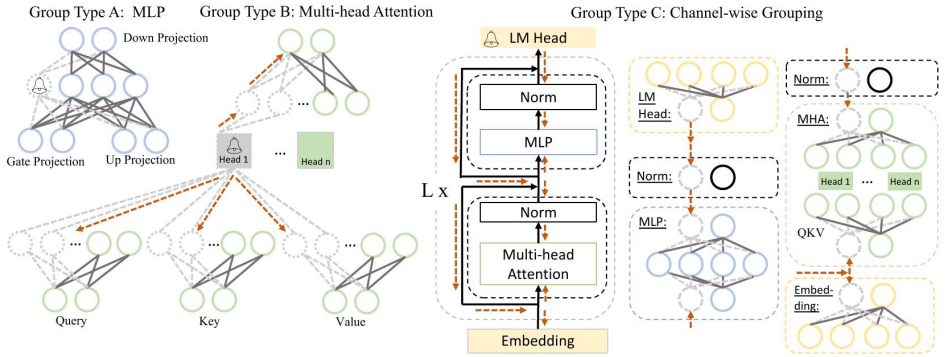


Figure 2: Illustration of the coupled structures in LLaMA. We simplify the neurons in each layer to make the dependent group clear. The trigger neuron, marked as a circle with a bell, cause weights with dependency pruned (dashed lines), which may propagate (red dashed lines) to coupled neurons (dashed circles). A group can be triggered by a variety of trigger neurons. Taking Group Type B as

依赖图 (Dependency Graph)

- 剪 head → Q/K/V/O 四矩阵同步剪
- 剪 channel → FFN up/down 同步
- 代码自动追踪 tensor 依赖
- 保证剪后模型结构一致

评分与数字

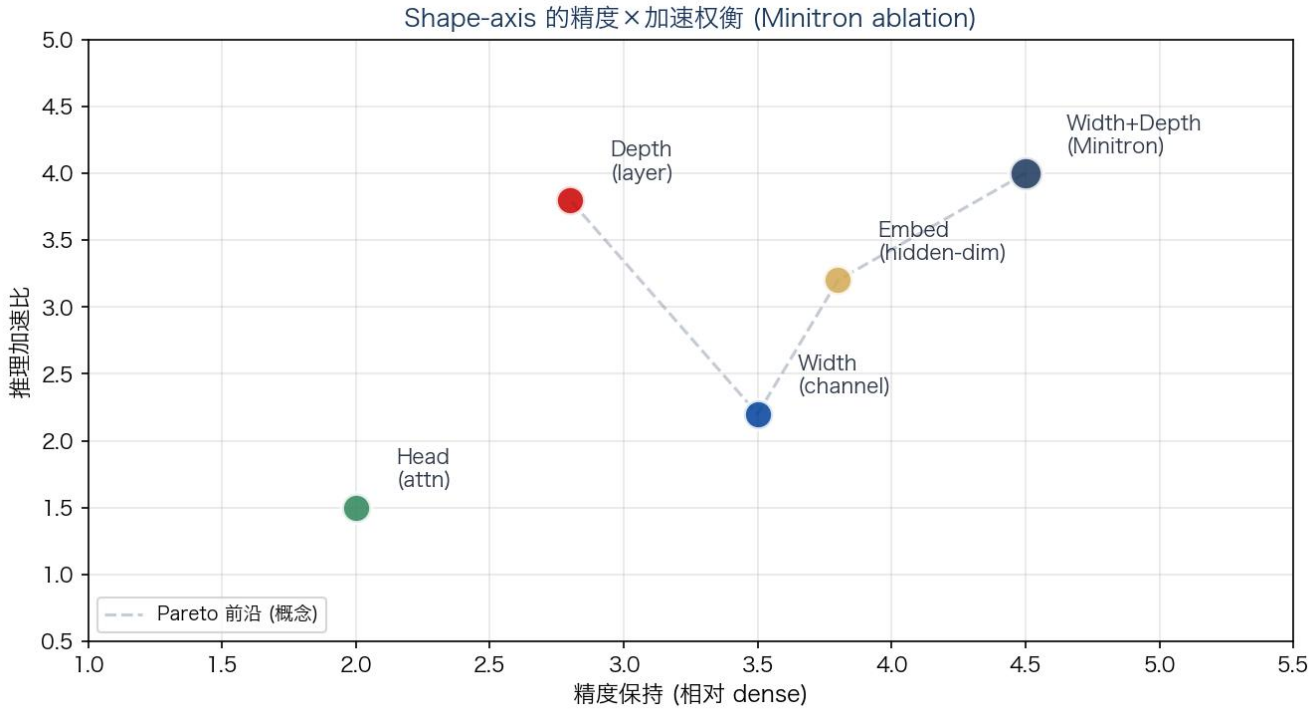
- Gradient Taylor: $s = |g \cdot W|$
- 仅 10 条 prompt 反向
- LLaMA-7B 剪 50%: 2h on 1×A100
- + LoRA fine-tune 5K 步

LLM-Pruner 是 L2 结构化剪枝开山之作: 依赖图 + gradient 成标准工具。

Ma et al., LLM-Pruner, NeurIPS 2023 (arxiv:2305.11627)

B.3 Shape-axis 权衡: Minitron 的 ablation

NVIDIA 在四条 axis 各做 ablation: 组合 (width+depth) 是 Pareto 最优。



核心发现

- **Width**: 温和, 精度好, 加速有限
- **Depth**: 加速大, 精度掉得快
- **Head**: 温和, 加速少
- **Embed**: 综合最好 (KV 连带)
- 没有免费 axis

组合:

- **width + depth** 左上角 Pareto
- 2024 工业最优

没有免费 axis; 组合式剪枝 (width+depth) 是 2024 工业最优解。

Muralidharan et al., Minitron, NeurIPS 2024 (arxiv:2407.14679)

B.4 SliceGPT: hidden-dim 切片 (ICLR 2024)

正交旋转 + 切 hidden-dim → 连带 KV-cache + 下游 GEMM 一起缩。

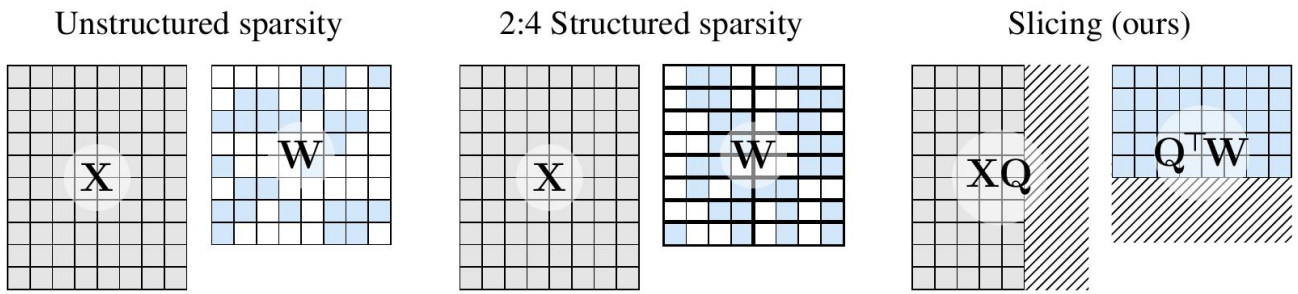


Figure 1: Matrix multiplication of the signal **X** and a weight matrix **W** under different types of sparsity. **Left**: unstructured sparsity, where some elements of **W** are zero, and **X** is dense. **Middle**: 2:4 structured sparsity, where each block of four weight matrix entries contains two zeros, and **X** is

核心方法

- PCA 分解 hidden activation
- 正交旋转对齐 principal directions
- 切掉 bottom-k dim (等价变换)
- 无需 calibration 数据
- 无需 fine-tune

系统连带收益

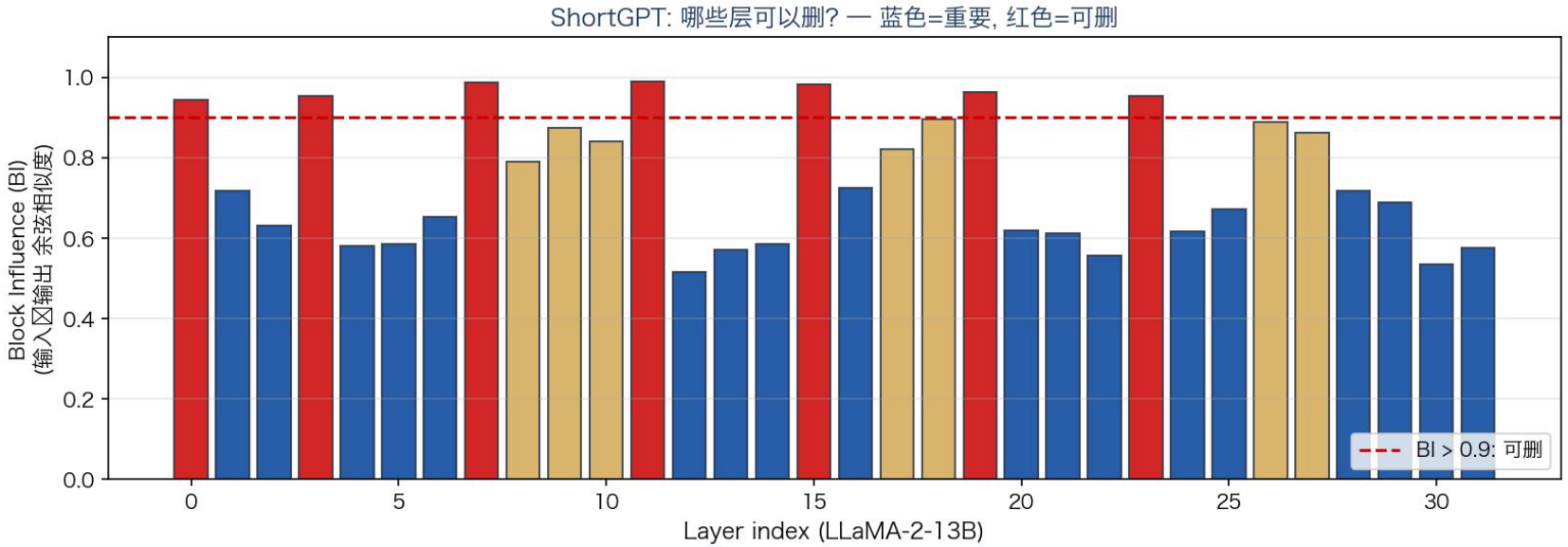
- d_model 缩 → **所有 Linear 层缩**
- **KV-cache 连带缩** (正比 d_model)
- LLaMA-2-70B 25% slice: 99% accuracy
- 端到端 1.4×; long context 2.2×
- 其他 axis 做不到 KV 连带

SliceGPT 是 hidden-dim axis 代表: long-context 场景 KV 连带缩是关键优势。

Ashkboos et al., SliceGPT, ICLR 2024 (arxiv:2401.15024)

B.5 ShortGPT / LaCo: 层级删除

LLM 存在冗余层: 一些层的输入 ≈ 输出, 可以直接删, 精度损失很小。



ShortGPT (2024)

- **Block Influence (BI)** = 输入↔输出余弦相似度
- BI ≈ 1 ⇒ 层近似跳过, 可删
- LLaMA-2-13B 删 20% 层: 精度仅 −2%

LaCo: Layer Collapse

- 合并相邻相似层 (取平均)
- 不是删, 而是 "合并保信息"
- LLaMA-2-7B 减 30% 层: PPL 仅 +1

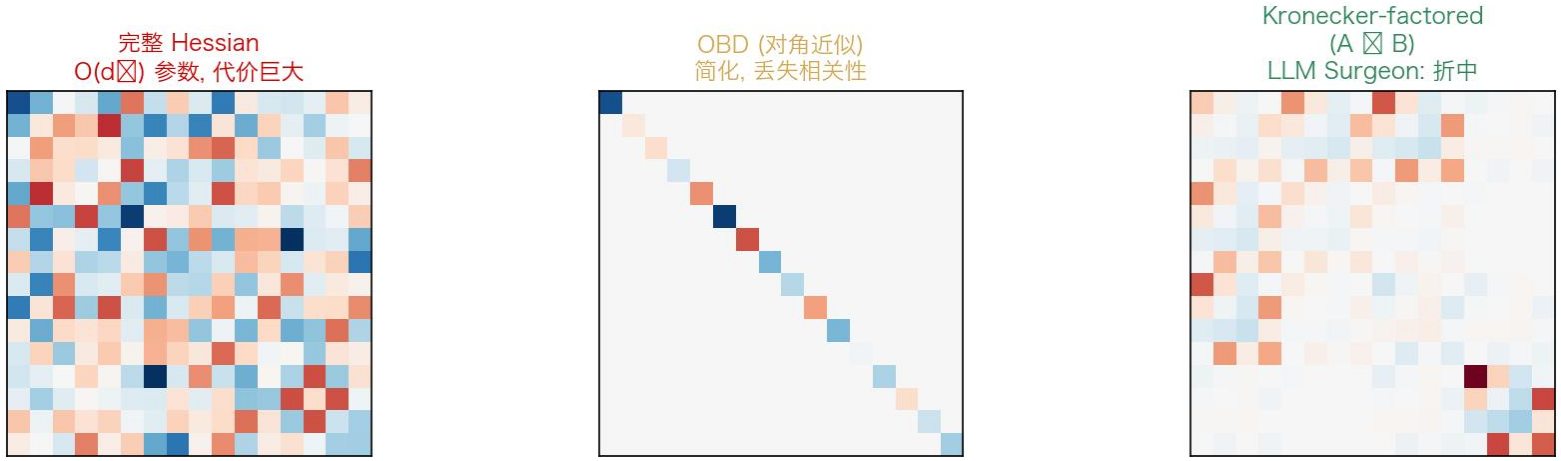
LLM 有显著层冗余; depth axis 最激进, 多与 width 组合使用。

Men et al., ShortGPT, 2024 (arxiv:2403.03853); Yang et al., LaCo, Findings of EMNLP 2024 (arxiv:2402.11187)

B.6 LLM Surgeon: Hessian 结构化剪枝 (ICLR 2024)

把 SparseGPT 的 OBS 扩展到结构化: Kronecker-factored Fisher 近似 Hessian。

LLM Surgeon: Kronecker Fisher 近似 — 保留相关性, 内存友好



核心方法

- Kronecker (A⊗B) 近似完整 Hessian
- 统一 unstructured / 2:4 / 结构化
- Multi-shot 权重更新迭代
- 动态 sparsity 分配: FFN 多, Attn 少

数字

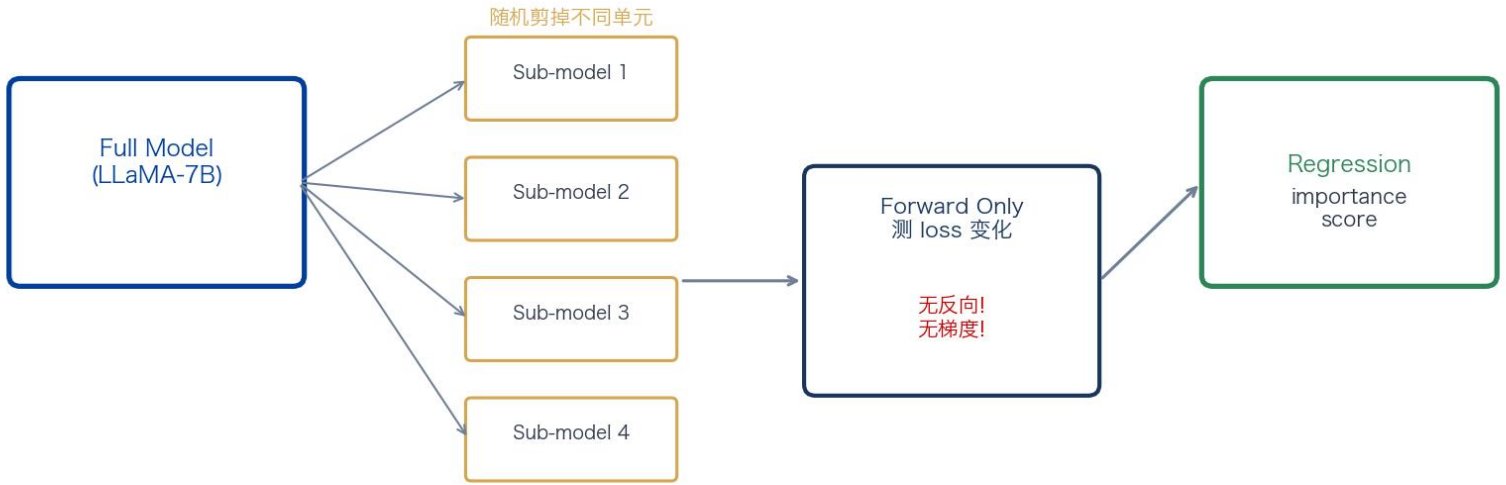
- LLaMA-2-7B @ 30%: PPL 7.95 (Wanda 8.41)
- 精度 SOTA
- 成本比 Wanda 高 ~10×
- 研究基准价值, Wanda 仍工业首选

LLM Surgeon 精度 SOTA, 成本 10× 高: 研究基准, Wanda 性价比仍首选。

van der Ouderaa et al., LLM Surgeon, ICLR 2024 (arxiv:2312.17244)

B.7 Bonsai: Gradient-free 结构化剪枝 (ICML 2024)

"没有梯度, 也能做结构化剪枝?": 扰动采样 + forward-only 评分。



Bonsai: Full-Model → 采样 Sub-Models → Forward-only 评分 → 回归求 importance

核心做法

- 对每个可剪单元采样 sub-model
- 每个 sub-model 少量 forward
- 回归求 importance score
- + informative prior 加速
- **无反向, 无梯度**

关键数字

- Phi-2 3B → 1.8B
- 4/6 Open-LLM tasks 超过其他 2B
- **单 A6000 (48GB)** 可跑 7B
- 比 LLM-Pruner 省 2-3× 内存

Bonsai 证明 forward-only 能做结构化剪枝: 小预算团队新选项。

Dery et al., Bonsai / Everybody Prune Now, ICML 2024 (arxiv:2402.05406)

B.8 Reshape 总结: axis × 方法 × 场景

把 B 段所有方法按 (axis, 评分, 成熟度) 整理。

Axis	代表方法	评分方式	系统收益	场景
Width (channel)	LLM-Pruner, Bonsai	grad / forward	Linear GEMM 小	FFN 加速
Depth (layer)	ShortGPT, LaCo	BI / 相似度	整层省	latency 敏感
Head (attention)	LLM-Pruner (head)	grad / activation	attn 计算省	memory-bound
Embed (hidden-dim)	SliceGPT, Minitron	PCA / importance	KV 连带缩	long-context
组合 (w+d+e)	Minitron	importance + KD	最优 Pareto	工业部署
Hessian 结构化	LLM Surgeon	Kronecker Fisher	精度 SOTA	研究基准

Reshape 各环节清晰: axis × 评分 × 成熟度: Minitron 组合 axis 是 2024 工业最优。

PART C

Heal: 剪完后如何守住精度?

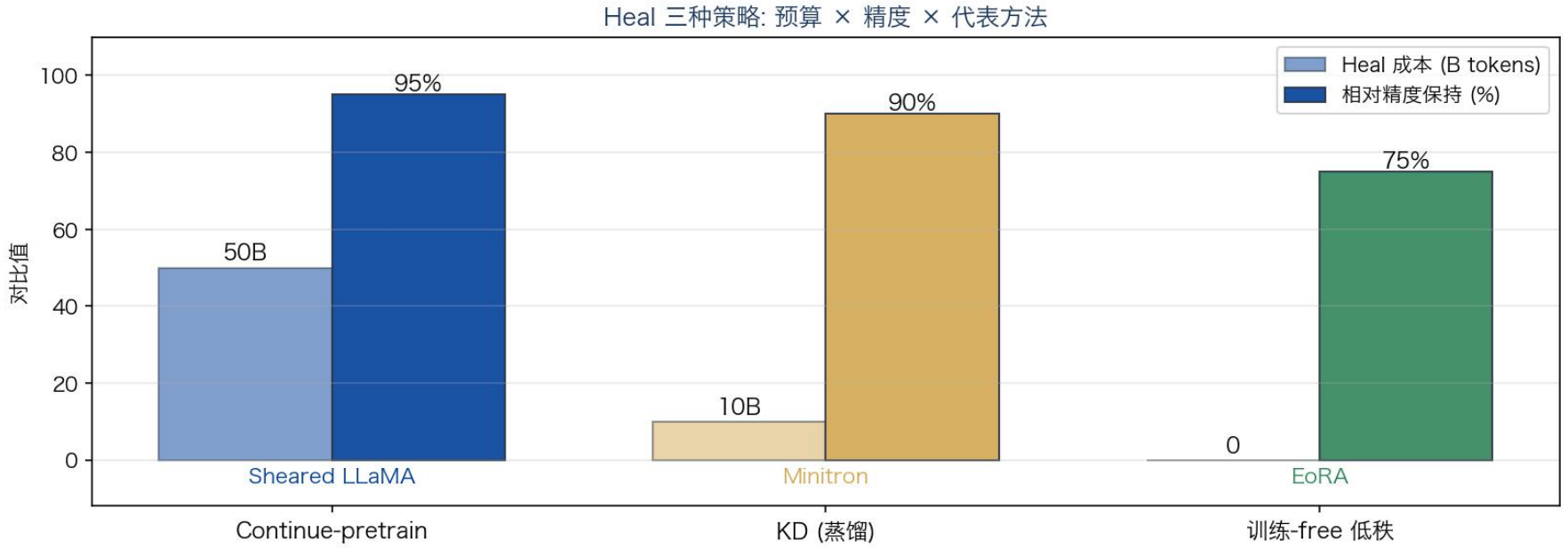
Continue-pretrain (Sheared) · KD (Minitron) · 训练-free 低秩 (EoRA) · 三种预算选择

本课大纲 (进度)

√ Part A	√ Part B	▸ Part C	Part D
Calibrate + Score	Reshape	Heal	组合 + 局限 + 收尾

C.1 为什么 Reshape 后需要 Heal?

Reshape 改变结构, 权重过时 → 精度必然掉; heal 用最小成本守住精度。



为什么精度会掉?

- 删结构后, 剩余权重对新结构过时
- 类似 NAS 找新架构但权重未重训
- Liu 2019: Scratch-E/B ⩾ Fine-tuned

Heal 三种策略

- Continue-pretrain → Sheared LLaMA (50B)
- KD 蒸馏 → Minitron (10B)
- 训练-free 低秩 → EoRA (0 tokens)

Heal 三种策略对应三种预算: 50B / 10B / 0 tokens: 按资源选择。

C.2 Sheared LLaMA: prune + continue-pretrain

把 heal 做成系统工程: Targeted pruning + Dynamic batch loading (ICLR 2024)。

We observe that training using the original pre-
reduction across different domains, compared to
his indicates that the pruned model retains vary-
(e.g., GitHub vs. C4) and simply using the pre-
cient use of data (Figure 4). To address these is-
hm consisting of the following two components:

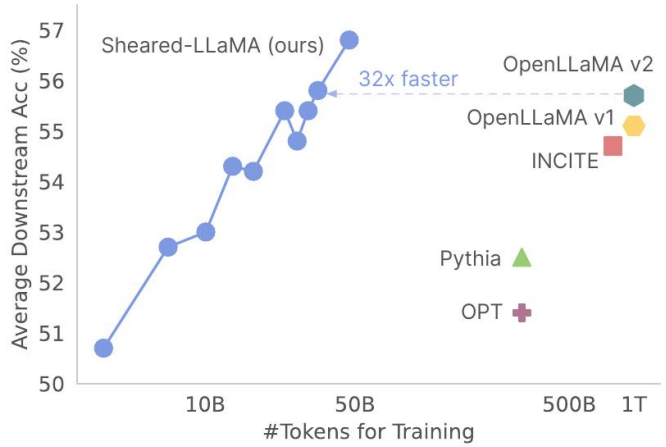


Figure 1: Sheared-LLaMA-2.7B surpasses a series of open-source models at a similar scale and only requires 1/32 (3%) of budget to achieve on-par performance with OpenLLaMA-3B-v2

两阶段方法

① Targeted Structured Pruning

- 定 target size (如 3B)
- 结构化剪 LLaMA2-7B → 3B
- 四 axis 同步

② Dynamic Batch Loading

- 继续预训练 **50B tokens**
- 按 domain 动态采样

结果

- Sheared-2.7B 超过 OPT-2.7B
- **40× token 效率提升**

Sheared LLaMA 把 heal 做成系统工程: 50B tokens 换 40× token 效率。

Xia et al., Sheared LLaMA, ICLR 2024 (arxiv:2310.06694)

C.3 Minitron: prune + KD (NVIDIA NeurIPS 2024)

NVIDIA 完整工业配方: width+depth 剪 + KD, 10B tokens 打败 from-scratch。

where μ and σ^2 represent the mean and variance across the embedding dimensions, ϵ is a small value for numerical stability, and γ and β are learnable parameters.

2.2 Importance Analysis

Estimating the importance or sensitivity of individual neural network components such as neurons, attention heads, and layers is a well-studied area [9, 13, 41]. In the context of LLMs, recent work has highlighted the ineffectiveness of traditional metrics such as weight magnitude for estimating importance [33]; instead, recent work on structured pruning of LLMs has focused on metrics such as gradient/Taylor [33], cosine similarity [34], and perplexity on a calibration dataset [26].

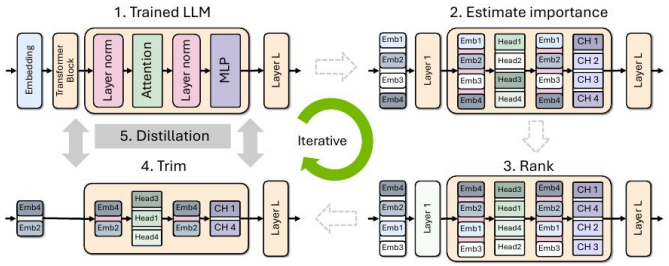


Figure 2: High-level overview of our proposed iterative pruning and distillation approach to train a family of smaller LLMs. On a pretrained LLM, we first evaluate importance of neurons, rank them,

完整配方

① **Width + Depth 剪枝**
- 四 axis 完整 ablation
- width+depth 组合最优

② **Importance + KD**
- Teacher = 原 LLaMA
- Loss: KL + MSE
- 仅 **10B tokens**

③ **数字**
- 40× fewer tokens vs scratch
- 打败 Phi-2, Gemma2, Qwen2-1.5B
- MMLU **+16%** vs scratch

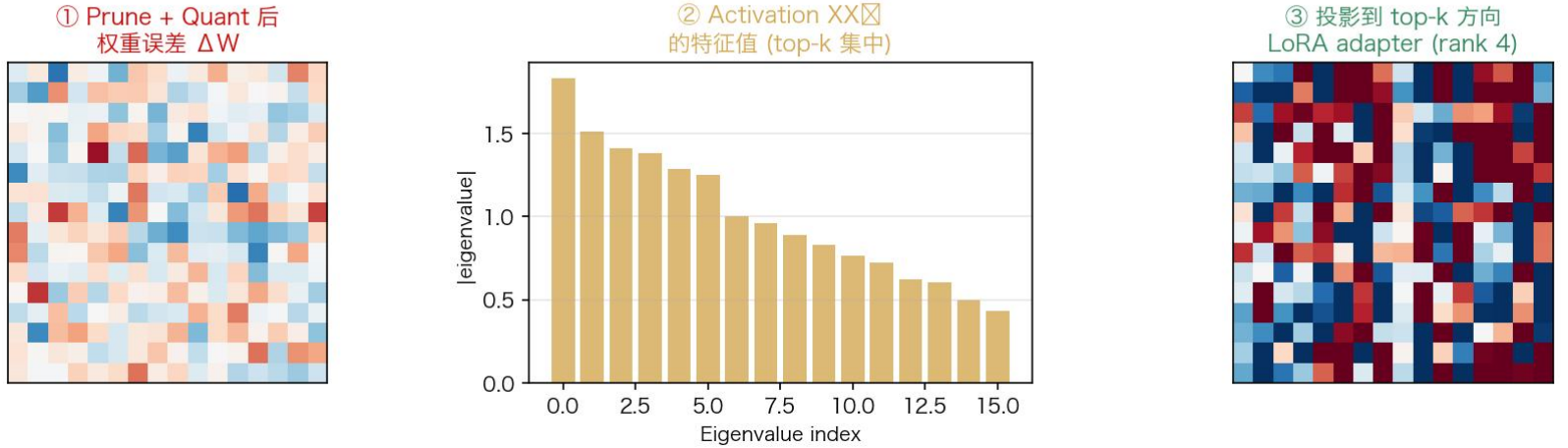
Minitron 是 L2 完整工业配方: 10B tokens 的 prune+KD 超过 scratch 训练。

Muralidharan et al., Minitron, NeurIPS 2024 (arxiv:2407.14679)

C.4 EoRA: training-free 低秩补偿 (ICLRW 2026)

最便宜的 heal: 无需 tokens, 无需梯度, 在 activation eigenspace 做 SVD 补偿。

EoRA: ΔW 投影到 activation 主方向, 保留 top-k 做低秩补偿



三步流程

- ① prune+quant 后权重误差 ΔW
- ② 计算 activation XX^T 特征分解
- ③ ΔW 投影到 top-k 特征方向
- 保留为 LoRA adapter (rank 4-16)

数字

- LLaMA3-8B @ 3-bit: ARC-C +10.8%
- @ 4-bit+2:4: ARC-E +31.3%
- 几分钟完成, 1.4× CUDA 加速
- 适合 prune+quant 多重误差场景

EoRA 填补 heal 环节的 "零成本" 角落, prune+quant 联合 heal 的利器。

Liu et al., EoRA, ICLRW 2026 (arxiv:2410.21271)

PART D

组合 + 局限 + 收尾

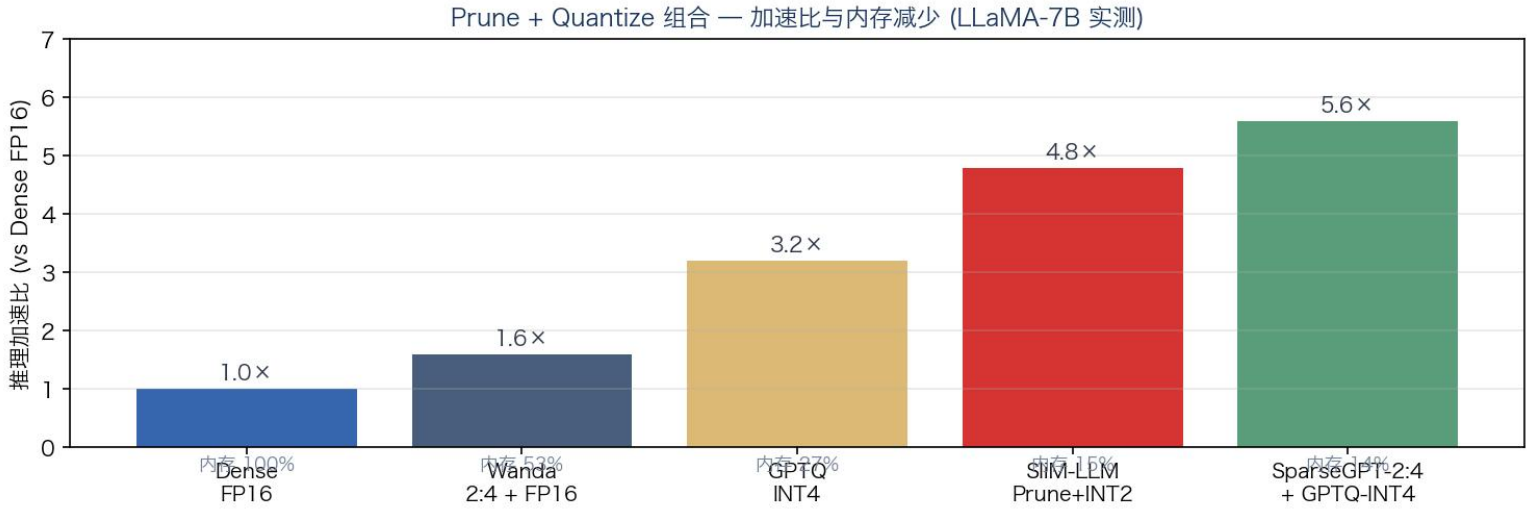
Prune+Quantize (SliM-LLM) · 工业部署 (Llama-3.1-Minitron-4B) · 局限 · 预告 L3

本课大纲 (进度)

√ Part A	√ Part B	√ Part C	▸ Part D
Calibrate + Score	Reshape	Heal	组合 + 局限 + 收尾

D.1 Prune + Quantize: SliM-LLM (ICML 2025)

同一 outlier 观察统一 prune 和 quant: salience 同时指导两者。



SliM-LLM 核心

- 3σ 找 outlier 权重 (重要)
- outlier 高 bit (8-bit)
- 其他权重低 bit (2-bit)
- Group-wise mixed-precision
- 与 2:4 剪枝同步

其他组合

- LLaMA-7B @ 2-bit: 6× 内存, −48% PPL
- SparseGPT-2:4 + GPTQ-4: 5.6×
- Wanda + AWQ: 常见组合
- 部署标配: prune + quant + heal 三合一

Prune + Quantize 是 2024-2026 工业标配; outlier 观察统一两者。

Huang et al., SliM-LLM, ICML 2025 (arxiv:2405.14917)

D.2 Llama-3.1-Minitron-4B: 完整部署案例

L2 配方的第一个端到端工业产品: LLaMA-3.1-8B → 4B, NVIDIA 2024。

where μ and σ^2 represent the mean and variance across the embedding dimensions, ϵ is a small value for numerical stability, and γ and β are learnable parameters.

2.2 Importance Analysis

Estimating the importance or sensitivity of individual neural network components such as neurons, attention heads, and layers is a well-studied area [9, 13, 41]. In the context of LLMs, recent work has highlighted the ineffectiveness of traditional metrics such as weight magnitude for estimating importance [33]; instead, recent work on structured pruning of LLMs has focused on metrics such as gradient/Taylor [33], cosine similarity [34], and perplexity on a calibration dataset [26].

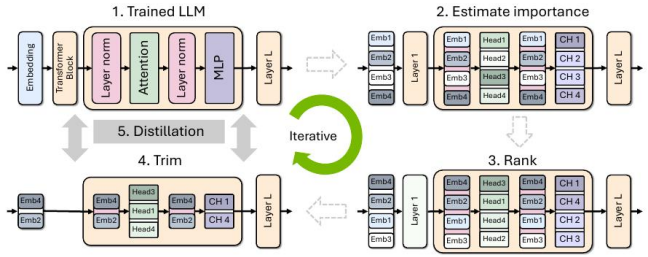


Figure 2: High-level overview of our proposed iterative pruning and distillation approach to train a family of smaller LLMs. On a pretrained LLM, we first evaluate importance of neurons, rank them,

完整 pipeline

① **Reshape: 8B → 4B**
- Depth: 32 → 16 layer (−50%)
- FFN: 14336 → 9216
- Hidden: 4096 → 3072

② **Heal: KD**
- Teacher = LLaMA-3.1-8B
- **10B tokens** 继续训练

③ **Deploy**
- FP8 on Hopper / Blackwell
- **2.7× throughput**
- 已上 HuggingFace

工程启示

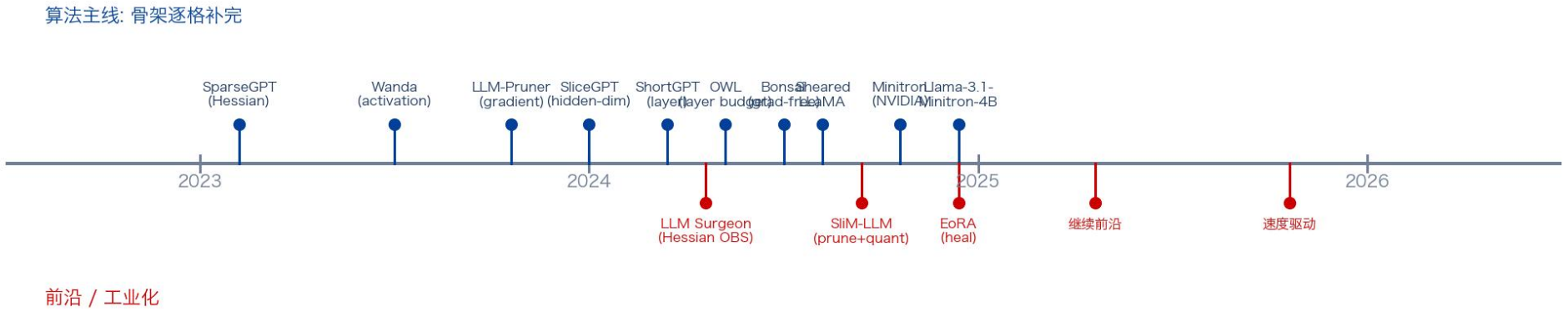
- Pruning = **prune + KD + quant + deploy** 完整链条
- 选型: 小模型结构化 > 非结构化; 部署要求 FP8 / 2:4 兼容
- 资源: 8B→4B 约 100-500 GPU-days, 比 scratch 训 8B 便宜 5-10×

Llama-3.1-Minitron-4B 是 L2 工业范本: prune+KD+FP8, 5-10× 训练成本节省。

Sreenivas et al., Llama-3.1-Minitron-4B, 2024 (arxiv:2408.11796)

D.3 2024-2026 其他前沿方向

骨架基本完备; 继续前沿在 "训练时稀疏" 和 "跨技术组合"。



LoRAPrune (ACL 2024)

- LoRA gradient 指导
- 不反向主模型
- LLaMA-65B 单 GPU
- 52.6% 内存节省

Compresso (2023 末)

- L0 正则 + LoRA
- 训练时稀疏化
- 不是 one-shot
- LLaMA-7B → 2.7B

未解决方向

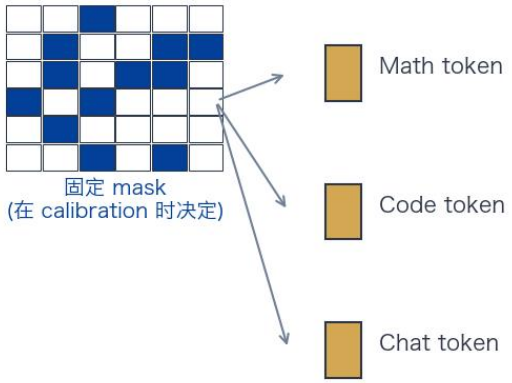
- 剪枝 + 推测解码 (VLM 仅 SpecVLM)
- 训练时稀疏 (Google/Meta 内部)
- 任务特定剪枝
- 超长 context → L3

L2 骨架基本完备; 继续前沿在训练时稀疏 + 跨技术组合。

D.4 L2 的根本局限: 为什么 L3 必然出现?

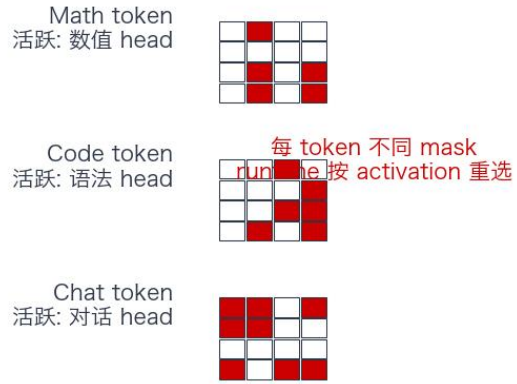
L2 解决 “一次性把模型变小”；两大局限推动 L3 的 input-dependent 方向。

L2: 静态剪枝 — 所有 token 共用一个 mask



每个 token 被相同剪枝结构处理
不一定最优

L3: 动态剪枝 — 每 token 按激活重选 mask



Input-dependent — L3 (Deja Vu, PowerInfer)

局限 1: 静态 mask

- 部署后 mask 固定
- 所有 token 共用结构
- Math / code / chat 其实需要不同权重
- → L3 动态权重 (Deja Vu, PowerInfer)

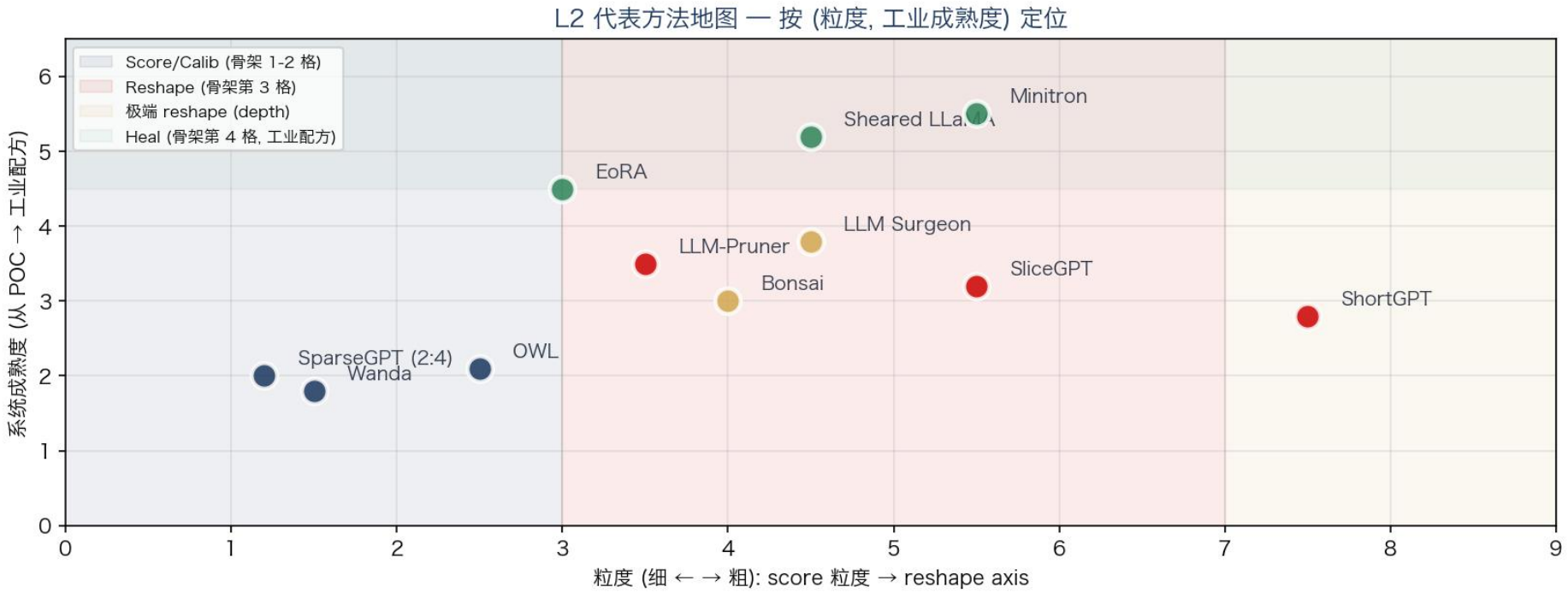
局限 2: 未压缩的 KV-cache

- LLaMA-70B @ 128k: KV 达 **40 GB**
- L2 剪权重但 KV 不处理
- Long context 瓶颈未解
- → L3 KV-cache (H2O, Quest)

L2 静态方法已收敛; 静态 + 未压缩的 KV 正是 L3 要补的两个方向。

D.5 L2 方法地图

按 (粒度, 工业成熟度) 把 L2 代表方法画在一张图上。



选型原则

- **左下简单**: Wanda / OWL: 性价比 baseline
- **中部 Reshape**: LLM-Pruner / SliceGPT / ShortGPT: 结构化加速
- **右上工业**: Sheared / Minitron: 完整 heal, 部署就绪

选型 = (预算, 精度需求, 部署约束); Minitron 是工业配方完整版。

E.1 从 L2 到 L3: 静态到动态的方法转换

L3 用 input-dependence 打破 L2 的静态限制: 2024-2026 最大增长点。

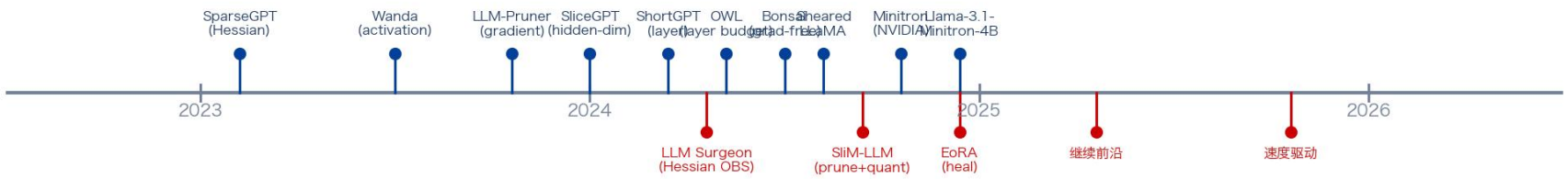
L3 Part A: 动态权重稀疏

- **Deja Vu** (ICML 2023):
  Predictor 预测 head/neuron 激活
- **PowerInfer** (SOSP 2024):
  Hot/cold neuron + GPU/CPU 异构
- **LLM in a Flash** (Apple 2024):
  闪存替 DRAM, 移动端

L3 Part B: KV-cache 剪枝

- **H2O** (NeurIPS 2023):
  Heavy hitter 主导 attention
- **StreamingLLM** (ICLR 2024):
  Attention sink token 必保
- **Quest** (ICML 2024):
  Per-query 重选 KV page

算法主线: 骨架逐格补完



前沿 / 工业化

静态 (L2) + 动态 (L3) = production answer; L3 用 input-dependence 打破静态假设。

第三节课

Input-Dependence 与前沿 (2024-2026)

动态权重稀疏 + KV-cache 剪枝

PART A

动态权重稀疏: 每 token 只激活少量权重

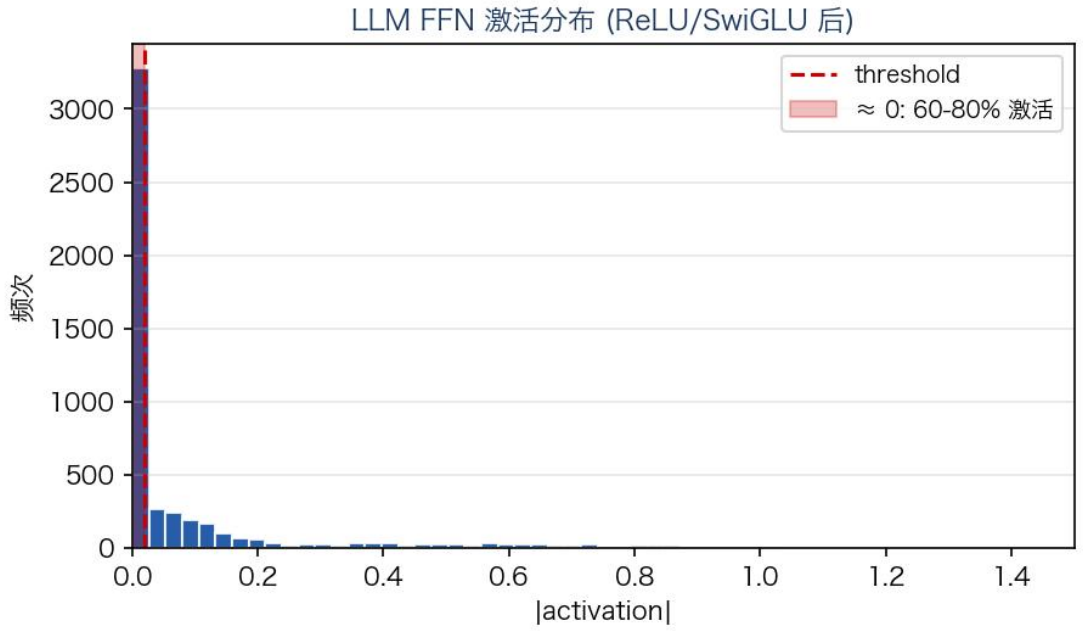
Deja Vu · PowerInfer · LLM in a Flash · NAEE · TEAL · Q-Sparse

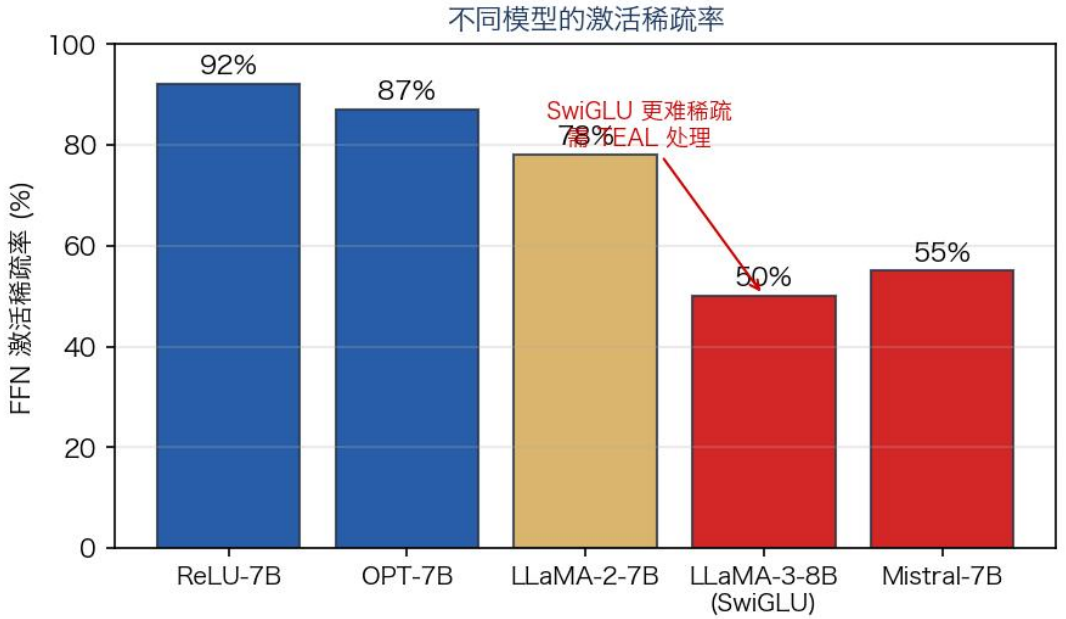
本课大纲 (进度)

▸ Part A	Part B	Part C
动态权重稀疏	KV-cache 剪枝	总结 + 过渡

A.1 激活稀疏现象: LLM 推理时多数 neuron 不激活

L3 的起点: 实测发现 FFN 激活大部分为零 (dynamic) → 只取激活的部分即可。





关键观察

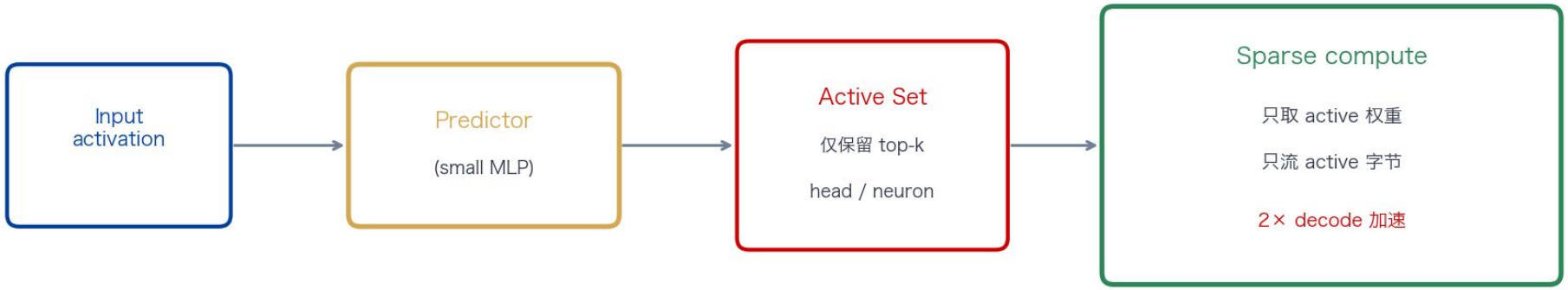
- ReLU-based LLM FFN: **85-95%** 激活为零 (ReLU 天然稀疏)
- SwiGLU-based (LLaMA-3/Mistral): 稀疏率降到 50-60%, 需 TEAL 这类方法
- 不同 token **激活的子集不同**: 这是 L3 的核心契机

FFN 激活天然稀疏 & input-dependent; 这是动态权重稀疏的物理基础。

Liu et al., Deja Vu (ICML 2023); TEAL (arxiv:2408.14690)

A.2 Deja Vu: predictor 预测 head/neuron 激活 (ICML 2023)

核心思想: 用小 MLP 预测当前 token 会激活哪些 head/neuron, 只算这部分。



Deja Vu: runtime 用 **predictor** 决定哪些 head/neuron 被激活, 只取相关权重做计算

(activation 进入 → predictor → 输出 top-k mask → 只计算 active 部分)

方法要点

- Small MLP predictor 附加在每层前
- 输出 top-k mask (head/neuron 级)
- 只 fetch active 权重到寄存器

系统收益

- OPT-175B decode: **~2× wall-clock**
- GPU 内存带宽利用率翻倍
- 精度几乎无损

Deja Vu 开创 runtime predictor 模式: 动态权重稀疏的第一代标准答案。

Liu et al., Deja Vu, ICML 2023 (arxiv:2310.17157)

A.3 Deja Vu 的限制 & PowerInfer 的动机

Deja Vu 只解决计算, 没解决内存: PowerInfer 把 “只存 active 权重" 做到极致。

Deja Vu 的边界

- 仅减少计算 (乘加次数)
- 所有权重仍需驻留 GPU 内存
- 70B 模型仍要 140 GB VRAM
- Decode 带宽瓶颈未完全解决
- 消费级 GPU 不现实 (VRAM 不够)

PowerInfer 的核心洞察

- 激活遵循 **幂律分布** (power law)
- **20% neuron 占 80% 激活** (hot)
- 其余 80% 是 cold
- Hot 放 GPU 常驻; cold 放 CPU 按需取
- → **消费级 GPU 也能跑 70B**

ory capacity but lower bandwidth. Nevertheless, each LLM inference iteration requires accessing the entire set of model parameters whose total size is too large for a single GPU, thus showing no locality at all and thus impeding efficient locality exploitation.

Recent works have identified activation sparsity in LLM inference [28, 34, 61]. During each inference iteration, only a limited number of neurons[1] are activated, significantly influencing token outputs. These sparse activations, which can be accurately predicted at runtime, allow for accelerated inference by computing only the activated neurons. However, the set of activated neurons varies across inputs and can only be determined at runtime, necessitating the entire model to be loaded into GPU memory. This requirement limits the approach's applicability in local deployment scenarios with constrained GPU VRAM.

Fortunately, we have observed that neuron activation in an LLM follows a **skewed power-law distribution** across numerous inference processes: a small subset of neurons *consistently* contribute to the majority of activations (over 80%) across various inputs (hot-activated), while the majority are involved in the remaining activations, which are determined based on the inputs at runtime (cold-activated). This observation suggests an inherent locality in LLMs with high activation sparsity, which could be leveraged to address the aforementioned locality mismatch.

maintaining their accuracy, thus freeing up GPU memory for LLM inferences.

Second, leveraging LLM sparsity requires the use of sparse operators. Conventional libraries like cuSPARSE [37] are not optimal due to their general-purpose design, which includes tracking each non-zero element and converting dense matrices into sparse formats [54, 63]. In contrast, PowerInfer designs neuron-aware sparse operators that directly interact with individual neurons, thereby bypassing operations on entire matrices. This approach enables efficient matrix-vector multiplication at the neuron level and removes the need for specific sparse format conversions.

Lastly, the optimal placement of activated neurons between the GPU and CPU in PowerInfer is a complex task. It involves evaluating each neuron's activation rate, intra-layer communication, and available hardware resources like GPU memory sizes. To effectively manage this, PowerInfer utilizes an offline phase to generate a neuron placement policy. This policy uses a metric that measures each neuron's impact on LLM inference outcomes and is framed as an integer linear programming problem. The policy formulation considers factors such as neuron activation frequencies and the bandwidth hierarchy of CPU and GPU architectures.

The online inference engine of PowerInfer was implemented by extending llama.cpp with an additional 4,200 lines of C++ and CUDA code. Its offline component, com-

Deja Vu 减计算; PowerInfer 减内存: 异构存储让消费级 GPU 跑 70B 成为可能。

Song et al., PowerInfer, SOSP 2024 (arxiv:2312.12456)

A.4 PowerInfer: Hot/Cold neuron + GPU/CPU 异构架构 (SOSP 2024)

Hot neuron 常驻 GPU, Cold neuron 按需从 CPU 取: 让消费级 GPU 跑 70B LLM。

better suited for CPUs, which provide more extensive memory capacity but lower bandwidth. Nevertheless, each LLM inference iteration requires accessing the entire set of model parameters whose total size is too large for a single GPU, thus showing no locality at all and thus impeding efficient locality exploitation.

Recent works have identified activation sparsity in LLM inference [28, 34, 61]. During each inference iteration, only a limited number of neurons[1] are activated, significantly influencing token outputs. These sparse activations, which can be accurately predicted at runtime, allow for accelerated inference by computing only the activated neurons. However, the set of activated neurons varies across inputs and can only be determined at runtime, necessitating the entire model to be loaded into GPU memory. This requirement limits the approach's applicability in local deployment scenarios with constrained GPU VRAM.

Fortunately, we have observed that neuron activation in an LLM follows a **skewed power-law distribution** across numerous inference processes: a small subset of neurons *consistently* contribute to the majority of activations (over 80%) across various inputs (hot-activated), while the majority are involved in the remaining activations, which are determined based on the inputs at runtime (cold-activated). This observation suggests an inherent locality in LLMs with high activation sparsity, which could be leveraged to address the aforementioned locality mismatch.

iterative process reduces the size of the predictors while maintaining their accuracy, thus freeing up GPU memory for LLM inferences.

Second, leveraging LLM sparsity requires the use of sparse operators. Conventional libraries like cuSPARSE [37] are not optimal due to their general-purpose design, which includes tracking each non-zero element and converting dense matrices into sparse formats [54, 63]. In contrast, PowerInfer designs neuron-aware sparse operators that directly interact with individual neurons, thereby bypassing operations on entire matrices. This approach enables efficient matrix-vector multiplication at the neuron level and removes the need for specific sparse format conversions.

Lastly, the optimal placement of activated neurons between the GPU and CPU in PowerInfer is a complex task. It involves evaluating each neuron's activation rate, intra-layer communication, and available hardware resources like GPU memory sizes. To effectively manage this, PowerInfer utilizes an offline phase to generate a neuron placement policy. This policy uses a metric that measures each neuron's impact on LLM inference outcomes and is framed as an integer linear programming problem. The policy formulation considers factors such as neuron activation frequencies and the bandwidth hierarchy of CPU and GPU architectures.

The online inference engine of PowerInfer was implemented by extending llama.cpp with an additional 4,200 lines of C++ and CUDA code. Its offline component, com-

架构核心

- 离线分析激活频率, 标记 hot/cold
- Hot neuron (20%) 常驻 GPU VRAM
- Cold neuron (80%) 在 CPU DRAM, 按需 fetch
- GPU/CPU 异步流水

产业冲击

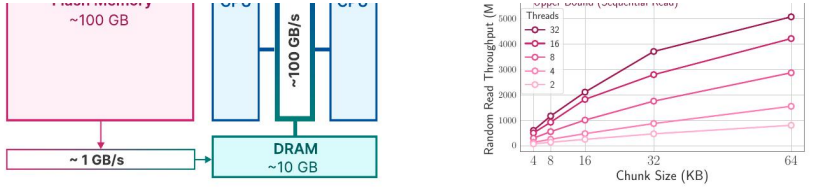
- **单 RTX 4090 跑 70B @ 11 tok/s**
- 消费级 GPU 跑大模型成为现实
- 稀疏 + 异构内存成为新范式
- 被后续 LLM in a Flash / llama.cpp 继承

PowerInfer 把稀疏 + 异构内存做成产品范式; 消费级硬件跑 70B 不再是假设。

Song et al., Song et al., PowerInfer, SOSP 2024 (arxiv:2312.12456)

A.5 LLM in a Flash: Apple 的移动端方案 (2024)

比 PowerInfer 更激进: 整个模型放 Flash 闪存, 按需加载: 面向手机/平板。



(a) Bandwidth in a unified memory architecture

(b) Random read throughput of flash memory

Figure 2: **(a)** Flash memory offers significantly higher capacity but suffers from much lower bandwidth compared to DRAM and CPU/GPU caches and registers. **(b)** The throughput for random reads in flash memory increases with the size of sequential chunks and the number of threads.

at least an order of magnitude larger than DRAM. Then, during inference, we directly load the required subset of parameters from the flash memory, avoiding the need to fit the entire model in DRAM. To this end, our work makes several contributions:

- First, we study the hardware characteristics of storage systems (e.g., flash, DRAM). We show that hardware constraints such as capacity and bandwidth limitations can have significant considerations when designing efficient algorithms for serving LLMs from flash (Section 2)

2.1 Bandwidth and Energy Constraints

While modern NAND flash memories offer high bandwidth and low latency, they fall well short of the performance levels of DRAM (Dynamic Random-Access Memory), in terms of both latency and throughput. Figure 2a illustrates these differences. A naive inference implementation that relies on NAND flash memory might necessitate reloading the entire model for each forward pass. This process is not only time-consuming, often taking seconds for even compressed models, but it also

核心技术

- 模型权重放 **flash 闪存**
- RAM 只保留 active neuron
- Row-column bundling: 减少 flash 随机读
- Window sliding: 复用上一步加载的权重

意义

- iPhone 16 Pro (8GB RAM) 可跑 7B LLM
- 推理延迟 **<0.5s/token** on A17
- 打开手机端 on-device LLM 时代
- 与 Apple Intelligence 集成

LLM in a Flash 把稀疏推向移动端; 端侧 LLM 从此有了系统答案。

Alizadeh et al., LLM in a Flash, ACL 2024 (arxiv:2312.11514)

A.6 NAEE: Mixtral 动态专家剪枝 (ACL 2024)

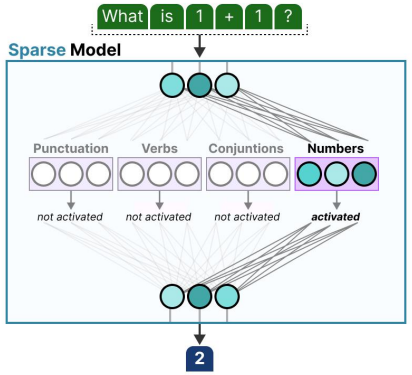
MoE 天然就是动态稀疏: 但还能更激进: 直接删不常用 expert + 动态 skip。

方法

- Mixtral 8×7B 的 8 个 expert per layer
- 每层用 reconstruction loss 枚举删除方案
- 剪掉最不重要的 2 个 expert
- + 动态 expert skipping (token-aware)

结果

- Mixtral 8×7B → 6×7B (25% 减)
- **单 80G GPU** 可跑 (原需 2 张)
- 1.2× 加速, task 平均 −2.9 分
- VRAM −25%



MoE 模型的 expert 也可动态剪枝; 与 Ch5 MoE 形成呼应。

Lu et al., NAEE, ACL 2024 (arxiv:2402.14800)

A.7 TEAL: Training-free 激活稀疏 (ICLR 2025)

把 Deja Vu 扩展到 SwiGLU LLM (LLaMA-2/3, Mistral): 全模型 40-50% 稀疏。

问题与突破

SwiGLU 激活不天然稀疏

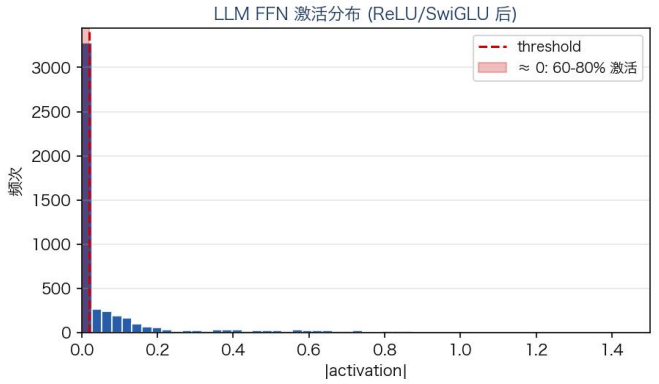
- ReLU: 60-80% 为零
- SwiGLU: 只有 50-60%
- Deja Vu 直接用效果差

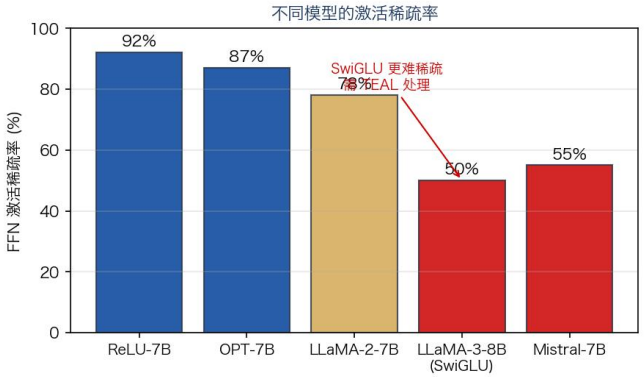
TEAL 方法

- Training-free magnitude threshold
- Attention + MLP 全部做
- 零均值单峰分布 → 阈值有效

数字

- **40-50% model-wide sparsity**
- LLaMA-2/3, Mistral 7B-70B 通用
- Decode **1.53×** (7B), **1.8×** (70B)
- 可叠加 4-bit AWQ
- 精度几乎无损





TEAL 把 Deja Vu 扩展到现代 SwiGLU LLM: 2024 最实用的 training-free 激活稀疏。

Liu et al., TEAL, ICLR 2025 (arxiv:2408.14690)

A.8 Q-Sparse: 稀疏 LLM 的 scaling law (Microsoft 2024)

Top-K + STE 让 LLM 全激活稀疏可训练: 首次给出稀疏 LLM 的 scaling law。

核心方法

- Top-K 选激活 (非量化 threshold)
- STE (straight-through estimator) 反向
- Attention + FFN 全 sparsity
- 可训练, 不是 post-hoc
- 可与 BitNet b1.58 叠加 (1-bit 权重)

Scaling Law 的意义

- 首次给出 sparse LLM inference-optimal scaling
- ~40% 稀疏匹配 dense baseline
- 更大模型 → 稀疏度可更高
- 对未来 edge 部署有理论指导
- 与 MoE scaling law 并列

Deja Vu: runtime 用 **predictor** 决定哪些 head/neuron 被激活, 只取相关权重做计算

(activation 进入 → predictor → 输出 top-k mask → 只计算 active 部分)

Q-Sparse 给出稀疏 LLM 的 scaling law: 未来训练时稀疏方向的里程碑。

Wang et al., Q-Sparse, Microsoft 2024 (arxiv:2407.10969)

A.8 动态权重剪枝的边界: 转折到 KV-cache

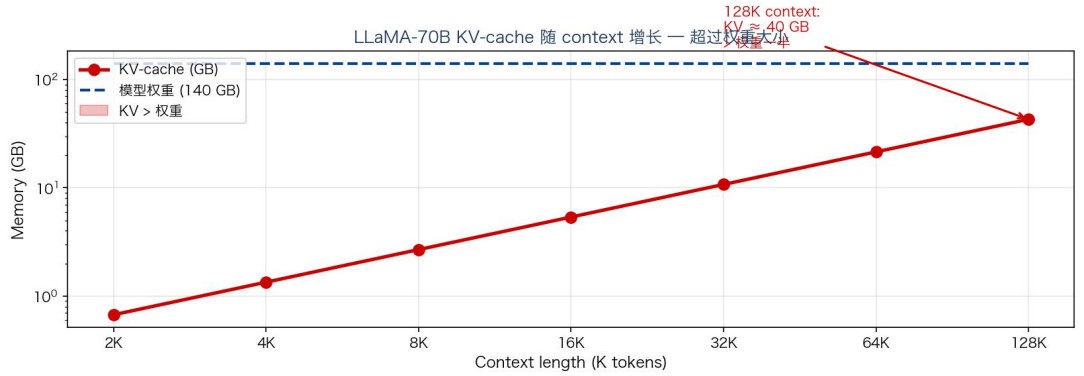
动态权重剪枝的收益有上限: decode 的内存占用最大的部分不是 W, 是 KV-cache。

动态权重剪枝的边界

- Predictor 本身有开销 (~5%)
- 激活稀疏预测精度限制实际收益
- 70B 模型权重 140 GB 仍要流
- Decode 带宽节省上限 ~50-60%
- 对长 context 帮助有限

真正瓶颈: KV-cache

- LLaMA-70B + 128k context: KV **40 GB**
- 超过权重的 28%; 长 context 下 > 50%
- 每步 decode 都要读全部 KV
- **这是剪权重无法解决的**
- → Part B 展开 KV-cache 剪枝



动态权重只减计算 + 权重字节; decode 长 context 的 KV 是剪权重无法解决的另一大户。

PART B

KV-cache 剪枝: 运行时动态裁剪

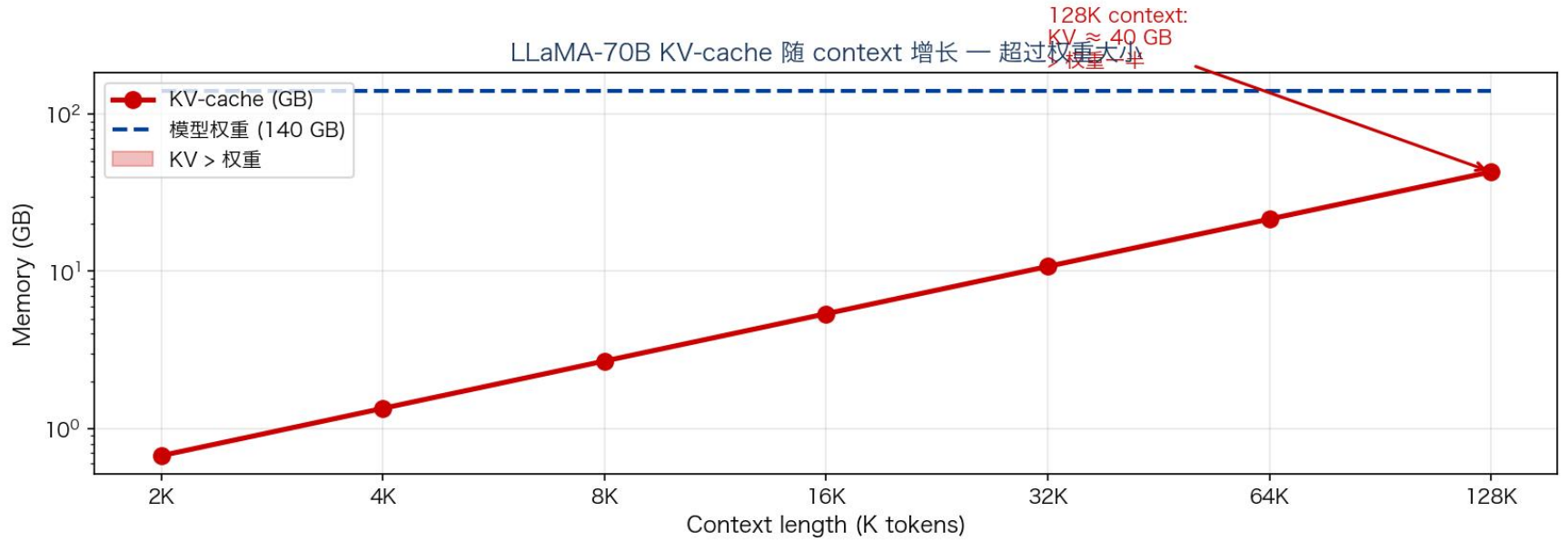
H2O · StreamingLLM · SnapKV · PyramidKV · Quest · RazorAttention · KIVI · LazyLLM

本课大纲 (进度)

√ Part A	▸ Part B	Part C
动态权重稀疏	KV-cache 剪枝	总结 + 过渡

B.1 KV-cache 的规模问题

KV-cache 随 context 线性增长, 长 context 时超过模型权重: 是 L3 的核心动机。



KV 大小公式

- size = B × L × H × d_h × 2 × bytes
- L=序列长; H=KV heads; d_h=head dim
- LLaMA-70B: 80 层 × 8 × 128 × 2 × 2 bytes = 640 KB/token

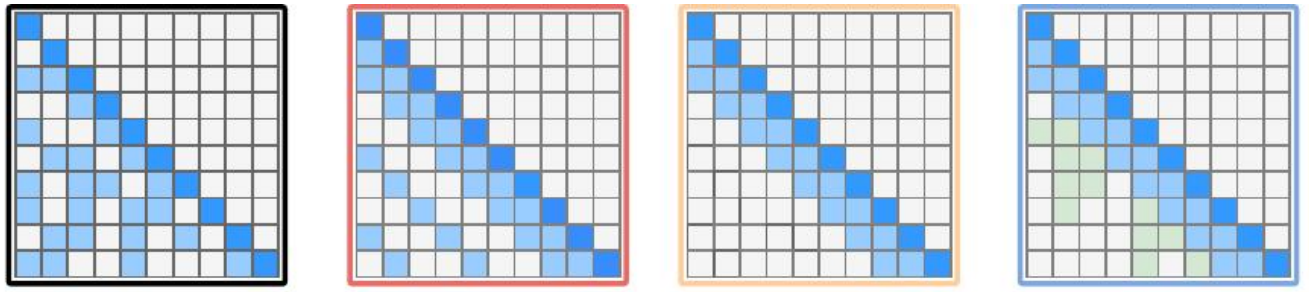
典型场景

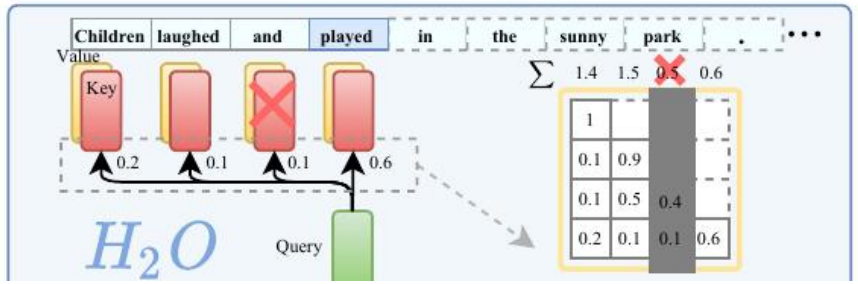
- 128k context, bs=1: **40 GB** KV
- 128k context, bs=8: 320 GB (1 张 H100 装不下)
- 长 context 应用: 文档 Q&A, 代码生成

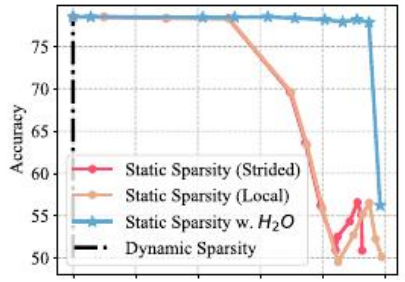
KV-cache 线性增长是 long-context 的核心瓶颈; 剪它是 2024-2026 最热方向。

B.2 H2O: Heavy Hitter Tokens (NeurIPS 2023)

观察: 少数 token (heavy hitter) 主导 attention 权重 → 只保留这些 token 的 KV。







Eviction 策略

- 每步累计 attention score
- 保留 top-k heavy hitter + 最近 R 个
- 淘汰 bottom-k
- **固定 KV 预算**, 永远 <= budget

数字

- KV 预算 = **20%** 时几乎无损
- OPT-30B 长 context 加速 5.5×
- vLLM、TGI 内置 H2O-like 支持
- 开创了 **eviction-based** KV 剪枝

H2O 开创 KV eviction 方法; heavy hitter 观察是后续所有 KV 剪枝的基础。

Zhang et al., H2O, NeurIPS 2023 (arxiv:2306.14048)

B.3 StreamingLLM: Attention Sink (ICLR 2024)

观察: 前几个 token (attention sink) 必须保留, 否则模型崩溃: 即使看起来不重要。

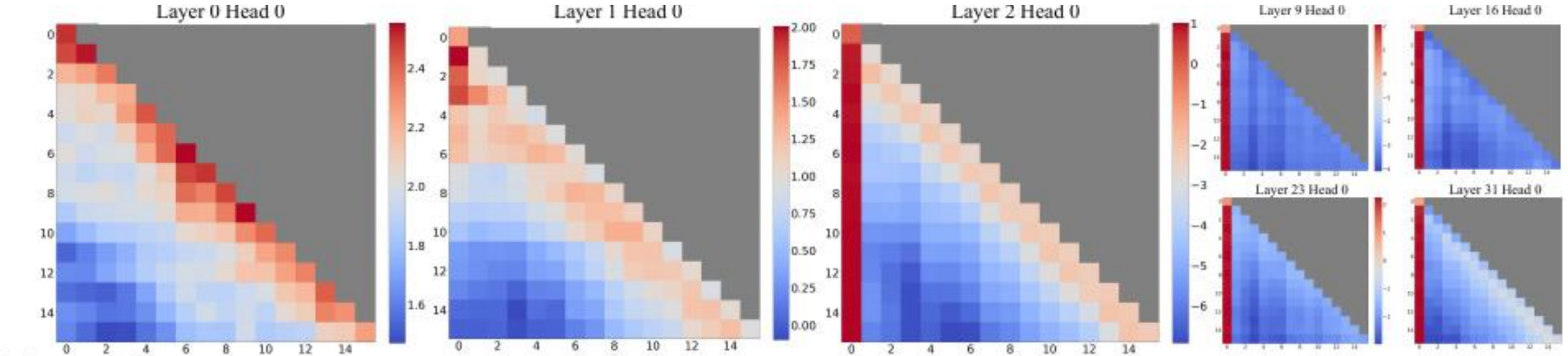


Figure 2: Visualization of the *average* attention logits in Llama-2-7B over 256 sentences, each with a le

核心观察

- 删掉最前面几个 token 后 PPL 显著增大
- 前几个 token 成为 "attention sink"
- 吸收无处可去的 attention mass
- 与内容相关性弱, 但功能必需

方案与影响

- 保留前 **4 个 sink** + 最近 **R 个** window
- **无限长 context** 不崩溃
- 已被 vLLM / TGI 纳入标配
- 与 H2O 可结合 (sink + heavy hitter)

StreamingLLM 的 sink 观察是 "必保留" 维度; 与 H2O 的 "按 attention" 形成互补。

Xiao et al., StreamingLLM, ICLR 2024 (arxiv:2309.17453)

B.4 H2O vs StreamingLLM: 两种保留策略对比

二者互补: H2O 保留 "内容重要" token; StreamingLLM 保留 "功能必需" token。

H2O

- 基于 attention score eviction
- 每步淘汰 bottom-k
- **fixed budget** (预算内不涨)
- 适合: generation / QA

StreamingLLM

- Sink (前 4) + sliding window (最近 R)
- **无限长 context**
- 不依赖 attention score
- 适合: streaming chatbot / 长对话

两个方向的观察互补; 现代 KV 剪枝常 sink + heavy hitter 混合。

B.5 SnapKV: Prompt-side 压缩 (NeurIPS 2024)

在 prefill 后就压完 KV: 用 prompt 末端的 observation window 选重要 KV。

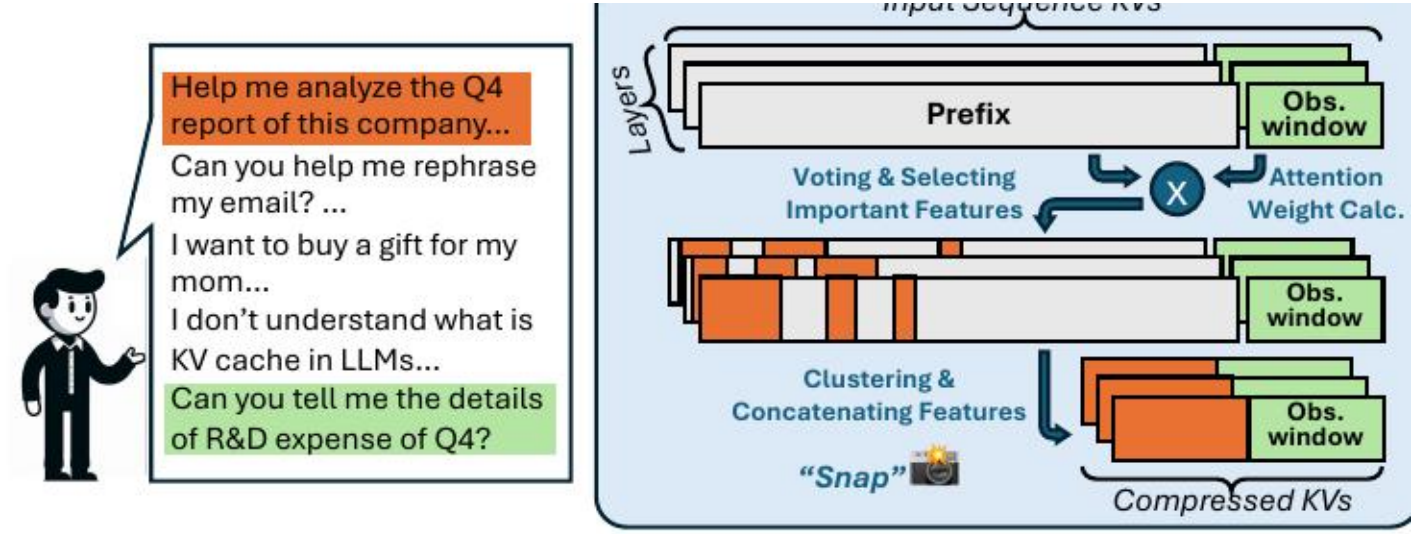


Figure 1: The graph shows the simplified workflow of SnapKV, where the

方法

- Prefill 完成后不是全存, 而是压缩
- 用 prompt 末端 window 观察 attention
- Max-pool 选簇状重要 KV cluster
- 只存这些选中的 token 的 KV

效果

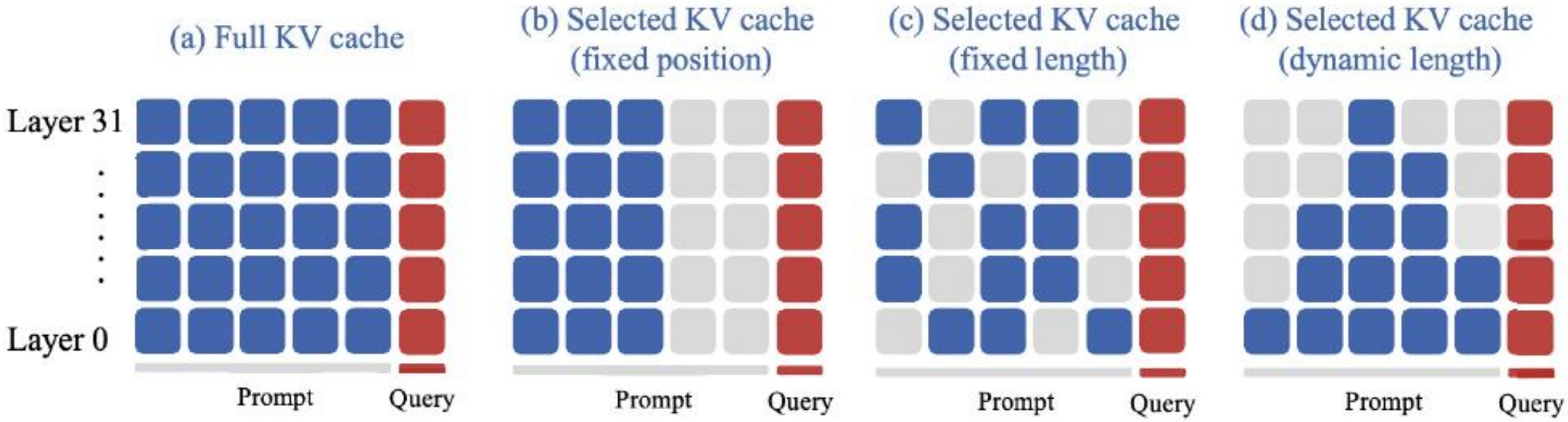
- **Decode 加速 3.6×** (16k, bs=2)
- **92% 压缩**, 几乎无精度损失
- 16k → 380k context on A100
- 对长 prompt 场景 (文档 Q&A) 尤其有效

SnapKV 从 decode-side 扩展到 prompt-side; 文档 Q&A 场景的首选方案。

Li et al., SnapKV, NeurIPS 2024 (arxiv:2404.14469)

B.6 PyramidKV: 层间预算金字塔 (ACL 2024)

观察: 底层 attention 分散, 高层集中 → 低层多给预算, 高层少给 (金字塔形状)。



核心洞察

- **Attention 信息漏斗**: 底层广, 高层窄
- 低层需要更多 KV 承载局部信息
- 高层已收敛到少数 token, 少量 KV 够
- → 按层分配不同 KV budget

效果

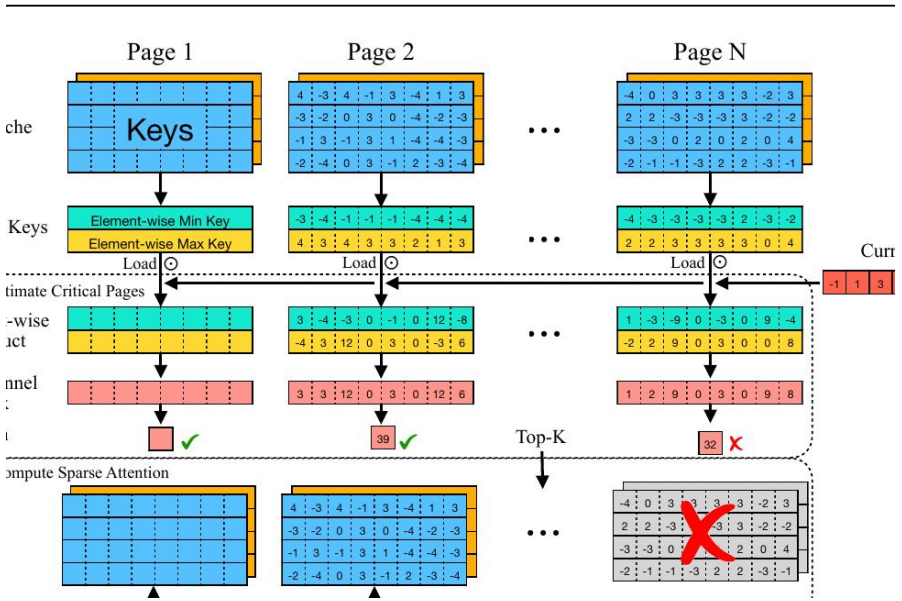
- **留 12% KV** 匹配 full KV 精度
- **0.7% KV**: TREC acc +20.5 vs H2O
- LLaMA-3-70B: **128 KV entries** 过 Needle
- 可与 H2O / SnapKV 组合

PyramidKV 首次按层分 budget: 与 OWL (L2) 对应, 是 KV 版的层级预算。

Cai et al., PyramidKV, ACL 2024

B.7 Quest: Per-query 重选 KV (ICML 2024)

范式转换: 前面方法都是"淘汰"; Quest 是"每步重选": 可以找回被淘汰的 KV。



方法

- KV 分 page (如 16 token/page)
- 每个 query 估算 page 重要性
- 只取 top-k page 参与 attention
- 不是 "淘汰", 而是 **每步重选**

意义 + 数字

- 避免 H2O 的 "淘汰误删" 问题
- 长 context retrieval-like 任务表现最佳
- 128k context: **7.3×** 加速
- 可与 FlashAttention 组合

Quest 从 "淘汰" 升级到 "每步重选": 信息可找回, 对 retrieval 类任务更友好。

Tang et al., Quest, ICML 2024 (arxiv:2406.10774)

B.8 RazorAttention: Retrieval Head 识别 (ICLR 2025)

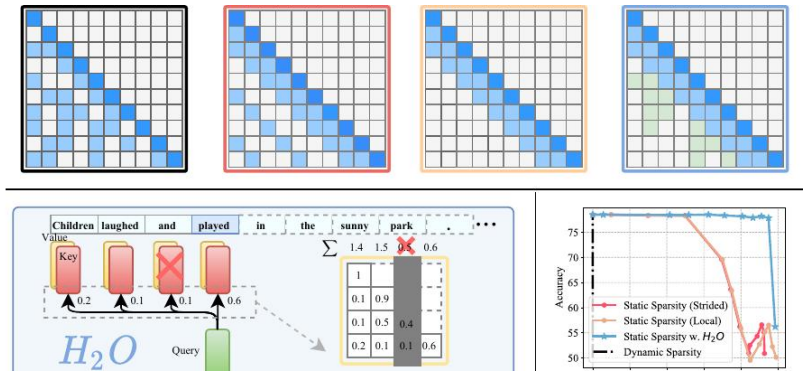
区分 retrieval head 和 local head: retrieval 给全 cache, local 截断 + 补偿 token。

核心洞察

- Retrieval heads: 负责从远距离 token 取信息
- Local heads: 只看附近 token
- Retrieval heads 需完整 KV
- Local heads 截断 + 1 compensation token 即可
- 离线识别 retrieval heads (一次分析)

优势 + 数字

- 压缩 70% KV 无可见降级
- 兼容 FlashAttention (H2O/SnapKV 不行)
- 不依赖 runtime attention map
- 可与 vLLM / TensorRT-LLM 集成
- 2024 产业最实用的 KV 剪枝之一



RazorAttention 从 head 维度剪 KV; 与 FlashAttn 兼容是重要工程优势。

Tang et al., RazorAttention, ICLR 2025 (arxiv:2407.15891)

B.9 KIVI: KV 量化 (2024)

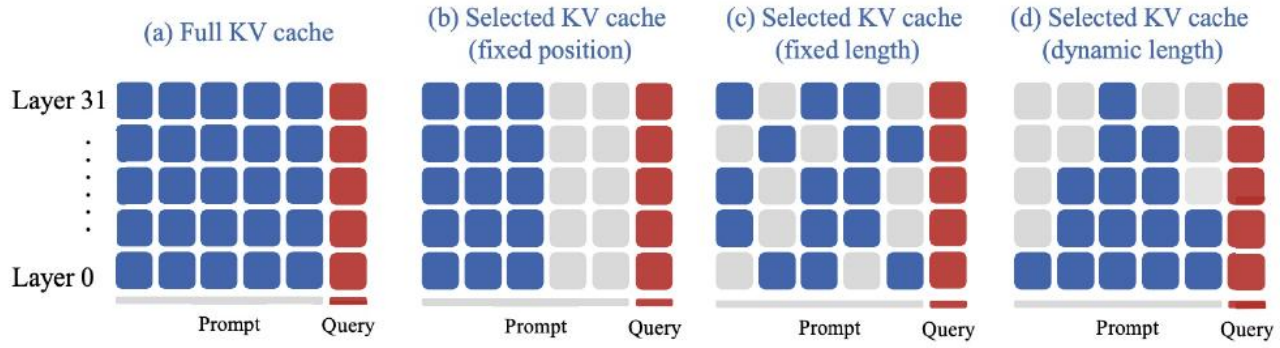
另一条路径: 不是剪 KV, 而是压 KV 的 bit 数: 2-bit KV 几乎无损。

核心方法

- K 和 V 分开量化
- K: per-channel quantization (dim 方向)
- V: per-token quantization (token 方向)
- 2-bit 即可保精度
- 需要 dequantize-on-the-fly

系统意义

- KV 内存 **4× 减少** (FP16 → INT2)
- 与剪枝 (H2O) **正交**, 可叠加
- 组合: PyramidKV + KIVI 可达 10× 减少
- 推理速度提升 2-3×
- 开源工具链集成



KIVI 从量化角度攻 KV; 与剪枝方法正交可叠加, 是现代 production 标配。

Liu et al., KIVI, ICML 2024 (arxiv:2402.02750)

B.10 2025 前沿: PQCache / Ada-KV / MiniCache

2025 继续前沿: learned predictor, per-head budget, 跨层合并: KV 剪枝持续演进。

Ada-KV (NeurIPS 2025)

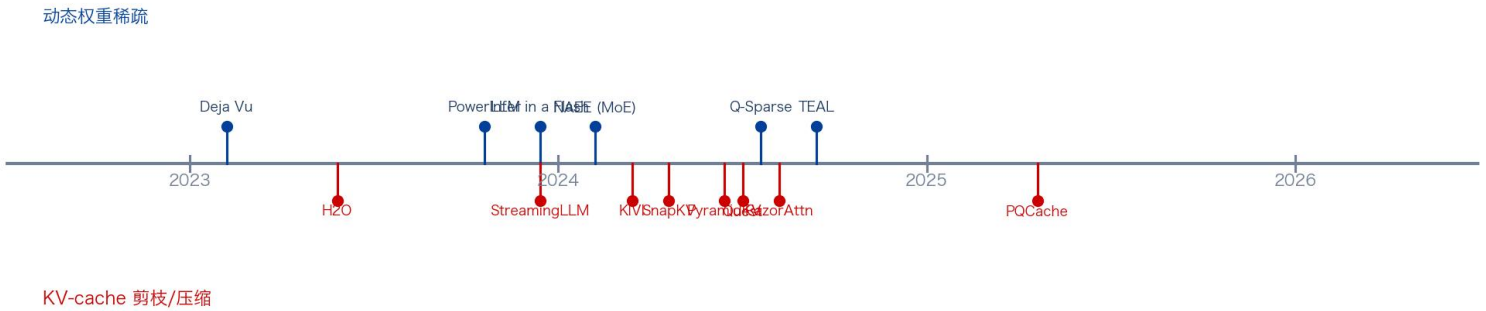
- Per-head 自适应 budget
- 比 PyramidKV 更细
- 每 head 不同预算
- 基于 attention 方差

MiniCache (NeurIPS 2024)

- **跨层合并** KV
- SLERP 插值相邻层
- **5×** 压缩
- 与 H2O / Quest 正交

PQCache (2025)

- Learned predictor for KV
- 类似 Deja Vu 但对 KV
- Runtime 选哪些 KV 重要
- 更精细 vs 启发式



KV 剪枝方向持续演进: head 级 → 层合并 → learned predictor; 还有增长空间。

Feng et al., Ada-KV, NeurIPS 2025 (arxiv:2407.11550); Liu et al., MiniCache, NeurIPS 2024 (arxiv:2405.14366)

B.11 LazyLLM: Prefill token 剪枝 (Apple 2024)

另一维度: 剪掉整个输入 token: 在 prefill 阶段跳过不重要的 token。

核心想法

- Prompt 中有些 token 对生成不重要
- 渐进式剪枝: 逐层越剪越激进
- 底层看全部, 高层只看重要的
- 支持 dynamic 的 prefill 长度

系统收益

- **2.3× prefill 加速**
- 长 context (8k+) 效果显著
- Apple 端侧 LLM 的关键优化
- 与 SnapKV 正交
- 对 retrieval/summary 场景友好

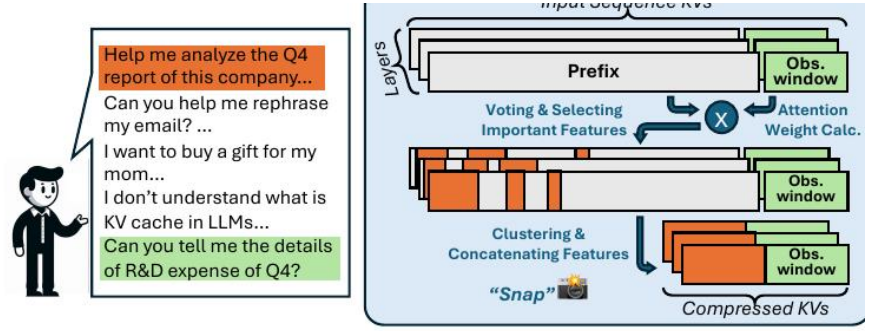


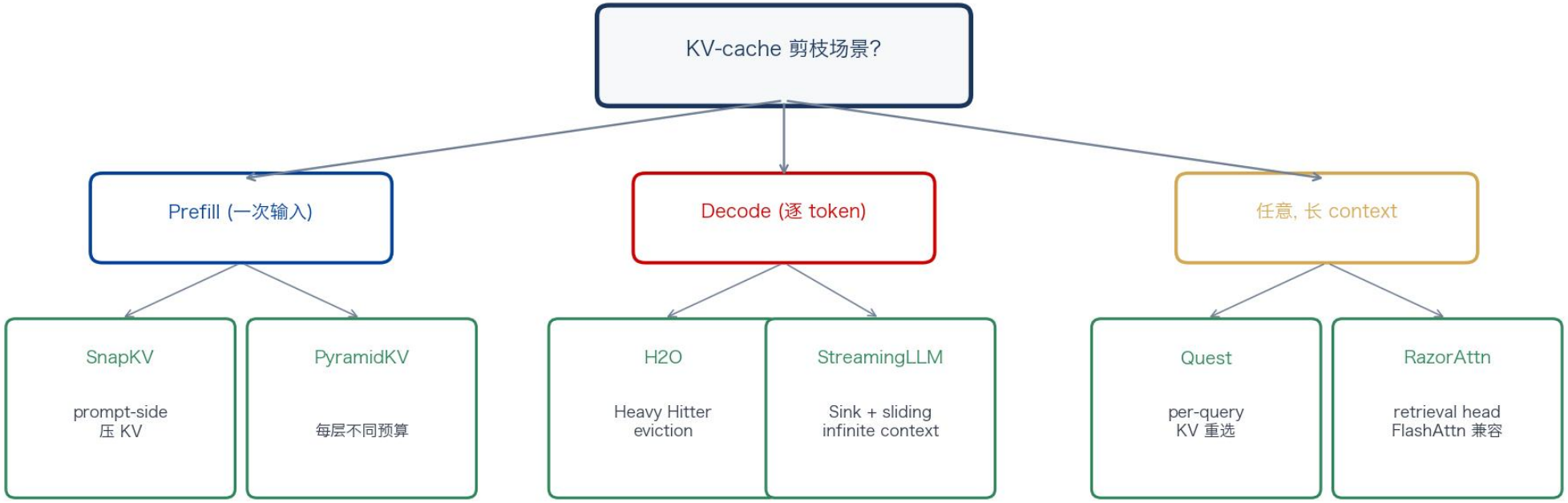
Figure 1: The graph shows the simplified workflow of SnapKV, where the

LazyLLM 从 prefill 端攻击长 context; 与 KV 剪枝形成完整 coverage。

Fu et al., LazyLLM, ICML 2024 (arxiv:2407.14057)

B.12 KV 策略决策树: 什么场景用什么?

工程视角: 根据 prefill/decode + context 长度 + 任务类型选择 KV 方法。



选型建议

- **通用 generation**: H2O + StreamingLLM (sink + heavy hitter)
- **长 prompt (文档 Q&A)**: SnapKV + PyramidKV

KV 方法选择 = 场景 × context 长度 × 任务类型; 组合使用往往最优。

B.13 KV-cache 方法总结: 维度 × 代表

把 Part B 所有方法按维度整理: 每个维度都有代表工作。

维度	代表方法	核心思想	适用场景
Heavy hitter 淘汰	H2O	按 attention 删 bottom-k	通用 generation
Sink + sliding	StreamingLLM	前 4 + 最近 R	streaming chat, 无限 context
Prompt 侧压缩	SnapKV	observation window 选 KV	文档 Q&A
层间预算	PyramidKV	低层多高层少	long-context 通用增强
Per-query 重选	Quest	每步按 query 选 page	retrieval-heavy
Head 识别	RazorAttention	retrieval vs local head	产业部署 (FlashAttn 兼容)
KV 量化	KIVI	2-bit quantize	正交, 可与任意方法叠加
Prefill 端 token	LazyLLM	逐层删 token	长 prompt, 端侧

KV 方法空间已广; 组合 (H2O + KIVI, Quest + RazorAttn 等) 是 production 常用配方。