Jingfeng

大语言模型

从张量、注意力到量化与推理系统

Jingfeng · 2026 年 10 月 22 日修订



从一个可手算的 block，走向可解释的推理系统

这本书把三个问题放在同一条路径上：模型究竟计算什么，数值表示改变了什么，计算与状态又怎样变成延迟和吞吐。

正文保留必要的线性代数与概率推导；可复算的矩阵例子连接公式与代码；推理资源账本和 CPU 实验用来检验数量级与实现假设。

原稿：2026 年 9 月。Jingfeng 书籍修订版：2026 年 10 月 22 日。

机制、算例、论文报告与本次实测在书中分别标明。

如何使用这本书

这本书沿着一条具体路径展开：先识别模型计算中的张量，再手算一个完整的 decoder block，随后追踪数值表示、持久状态与资源开销，最后让推导接受代码和评测的检验。希望读者面对一篇论文或一段推理代码时，能够说明它改变了什么、依赖什么条件，以及证据支持多大范围的结论。

全书分为四篇。第一篇从张量与条件概率出发，经过 token、embedding 和输出头，建立 Transformer、位置表示及注意力变体的共同语言。第二篇讨论存储、计算和累积精度，统一量化格点、输入相关误差与 KV 状态。第三篇连接训练目标、推理内存与计算账本、IO 和调度、推测采样、外部证据及评测。第四篇提供实际运行过的 CPU 工作坊；附录集中给出公式、练习提示与文献路线。

直觉与机制

阅读时持续追问：**这个张量是什么形状？什么时候产生？存多久？误差沿哪条路径传播？**再加一问：正在讨论的是数学恒等式、某个实现，还是带条件的实验结果？

按问题选择阅读路线

初次学习 顺读第 1 至 4 章，先完成第 3 章的小矩阵前向；再读数值与量化部分，把误差定义和例子对应起来。

已有实现经验 先用第 16 章检查形状、全局位置和多层 cache，再结合第 11、12 章复算字节与 FLOPs，解释 prefill/decode 的差别。

准备做研究 先读第 15 章固定比较协议，再按附录 C 的文献地图恢复方法的目标、变换与执行条件；从能够独立验证的最小假设开始。

本书要求基础矩阵乘法、导数与概率知识，必要概念均配有最小解释；代码使用 Python 与 NumPy。正文统一采用行向量右乘权重 $Y = XW$，不要求框架也按相同顺序存储。真实模型的 tensor layout、分组轴和 dtype 都应以代码为准。

证据类型与内容边界

本次修订与来源核验截至 **2026 年 10 月 22 日**。本书选取经过核对、适合解释机制的代表性工作，不以方法数量或最新排行榜作为完整性标准。文献采用首次 arXiv 预印本日期，链接固定到核验版本；条目键中沿用的年份不等同于会议出版年份。

书中把证据分为四类：**手算或合成教学数据**用于展示机制；**本次 CPU 实测**来自随书脚本与保存结果；**论文报告**只在对应来源和实验范围内引用；**未执行方案**给出所需配置、步骤及限制。解析内存账本是指定形状下的整数复算，不是设备显存峰值测量。浮点伪量化同样不构成低比特 kernel 加速的证据。

训练配方、Agent 工程和集群基础设施留给相应分册，本书保留解释模型机制所需的接口与边界。跨书指引采用主题名称。每章的练习可以独立推进；尚无 GPU 的读者也可完成全部 CPU 机制实验。

目录

如何使用这本书 i

第一部分　从符号到一个完整的模型计算 1

第 1 章　张量、概率与误差：共同的数学语言 2

1.1 形状是计算的接口 . . . 2
1.2 一次矩阵乘，三种不同的“乘” . . . 2
1.3 广播与归约：scale 究竟与谁共享 . . . 3
1.4 从 logits 到概率：稳定 softmax . . . 3
1.5 条件概率、熵、交叉熵与困惑度 . . . 4
1.6 把“误差很小”写成可检验的量 . . . 5
1.7 计算图与误差如何向后传播 . . . 5
1.8 练习与解题线索 . . . 6

第 2 章　从 token 到条件分布：模型的输入与输出 7

2.1 自回归分解从哪里开始，到哪里结束 . . . 7
2.2 Tokenization 决定离散坐标系 . . . 7
2.3 特殊 token 与聊天模板也是输入协议 . . . 8
2.4 Embedding、最终归一化与输出头 . . . 8
2.5 训练损失：哪些位置参与平均 . . . 9
2.6 采样分布与模型分布应分开记录 . . . 9
2.7 为何逐步贪心不等于全序列最大概率 . . . 10
2.8 从接口到机制的检查清单与练习 . . . 10

第 3 章　完整 decoder block：从形状到一次前向 11

3.1 两条子层和两次残差相加 . . . 11
3.2 RMSNorm 具体归一化什么 . . . 12
3.3 Q、K、V 是不同的可学习投影 . . . 12
3.4 因果注意力的四个计算步骤 . . . 12
3.5 多头拼接、输出投影与参数量 . . . 13
3.6 SwiGLU 的两条输入分支 . . . 13
3.7 一组可以完整复算的双头前向 . . . 14
3.8 从一次前向推到参数、计算与状态 . . . 16

3.9 练习与验证方法 16

第 4 章 位置与注意力家族：表示、可见图和缓存 17

4.1 位置与掩码承担不同的工作 17
4.2 RoPE：把相对位置写进 Q/K 点积 17
4.3 长上下文：可计算、可存储与可使用是三件事 18
4.4 MHA、MQA 与 GQA：哪些头共享历史 18
4.5 稀疏注意力：用可见图控制边数 19
4.6 线性注意力：归一化与前缀状态缺一不可 19
4.7 MLA：缓存低维 latent 与解耦的位置部分 20
4.8 把计算与状态放在同一张表中 21

第二部分 数值表示、误差与持久状态 22

第 5 章 数值格式：存储、乘法与累积是三种精度 23

5.1 先问张量在何时存在 23
5.2 浮点数怎样分配有限的位 23
5.3 Scale 并非整数独有 24
5.4 沿一条矩阵乘路径检查三种精度 24
5.5 舍入、饱和与累积应分开分析 25
5.6 建立可以检验的数值记录 25

第 6 章 统一量化语言：格点、分组与校准 26

6.1 编码与重构是一对映射 26
6.2 对称区间有一个常被省略的选择 26
6.3 尺度的共享轴决定矩阵乘怎样恢复 27
6.4 有效位宽与实际文件大小 27
6.5 校准是在选择承担哪一种误差 28
6.6 均匀噪声模型何时有帮助 28
6.7 一个可审查的伪量化接口 28

第 7 章 权重量化：从逐格舍入到输入感知的压缩 30

7.1 RTN：先建立一个可解释的基线 30
7.2 GPTQ：从一个坐标的约束补偿推导更新式 30
7.3 AWQ 与 SmoothQuant：同一个等式，不同的误差目标 32
7.4 旋转：改换坐标基，也必须守住计算图 33
7.5 统一写法中不能丢掉逆变换 33
7.6 非均匀标量码本与向量码本 34

7.7 低秩与稀疏补偿：额外精度有额外账单 34
7.8 二值、三值与两位：表示、训练与计算各问一次 35
7.9 验证一个方法：先证明等价，再比较误差，最后测系统 35

第 8 章 激活量化与端到端误差：从一个数到一次生成 36
8.1 先把权重、激活和累积误差放进同一等式 36
8.2 激活分布为什么比权重更难固定 36
8.3 共享尺度既是统计选择，也是矩阵乘接口 37
8.4 平滑、旋转、裁剪与高精度分支保护什么 37
8.5 QAT：前向离散，反向使用什么梯度 38
8.6 同样的 MSE，为何有不同的输出与生成 38
8.7 误差穿过多层时，局部校准能说明多少 39
8.8 局部指标的定义和零分母必须一致 39
8.9 从损失与 PPL 到任务质量 40
8.10 把诊断组织成可复查的比较 40

第 9 章 KV Cache：可复用状态、量化误差与长上下文 42
9.1 缓存递推与全量前向何时等价 42
9.2 “没有缓存”的复杂度取决于重算策略 42
9.3 容量、带宽与物理布局是三个账本 44
9.4 K 与 V 的误差沿不同路径进入输出 44
9.5 KIVI：不同轴、在线追加与高精度尾部 45
9.6 量化、淘汰与长上下文质量 46

第三部分 从训练目标到推理行为 47

第 10 章 从训练目标到模型行为 48
10.1 自回归损失是逐位置的概率账本 48
10.2 数据分布决定优化在强调什么 48
10.3 优化器状态、训练精度与计算预算 49
10.4 Scaling law 是带实验范围的经验模型 49
10.5 SFT：从续写语料到条件响应 50
10.6 偏好优化：从 KL 约束到 DPO 50
10.7 可验证奖励与测试时计算 51
10.8 MoE：总容量与每 token 工作量分离 51

第 11 章 推理的内存账本与计算账本 52
11.1 先固定单位与符号 52

11.2 参数量从矩阵形状相加 . . . 52
11.3 KV 是逐层、逐请求增长的持久状态 . . . 53
11.4 一个贯穿全章的虚构模型 . . . 53
11.5 激活与工作区：为什么不能一律乘层数 . . . 54
11.6 用 FLOPs 解释 prefill 与 decode . . . 54
11.7 Roofline 的适用对象是具体数据移动层级 . . . 55
11.8 量化与批处理改变哪个瓶颈 . . . 55

第 12 章 从注意力 IO 到服务执行 56

12.1 FlashAttention：不物化整个注意力矩阵 . . . 56
12.2 PagedAttention：逻辑序列不要求物理连续 . . . 56
12.3 连续批处理：在迭代边界更换活跃集合 . . . 57
12.4 分块 prefill：保持因果语义，改变执行粒度 . . . 57
12.5 前缀复用的条件比字符串相同更严格 . . . 58
12.6 模型并行从矩阵分解理解 . . . 58
12.7 分离 prefill 与 decode 也要支付状态传输 . . . 59

第 13 章 推测解码：用可回滚的猜测保持目标分布 60

13.1 先定义需要保持的分布 . . . 60
13.2 一个 token 的精确接受与拒绝 . . . 60
13.3 一轮验证多个候选：只提交连续前缀 . . . 61
13.4 把 logits 偏移与 KV 回滚写成状态事务 . . . 62
13.5 EOS、约束与有限精度的边界 . . . 62
13.6 为什么少调用 target 仍可能没有加速 . . . 63
13.7 自推测、贪心与候选树的适用条件 . . . 63

第 14 章 架构与外部世界：状态、证据与动作 65

14.1 从显式历史到有限维状态 . . . 65
14.2 递归误差：渐近稳定不等于短时不放大 . . . 66
14.3 视觉证据怎样进入语言计算 . . . 66
14.4 检索增强：潜在文档与拼接证据是不同建模方式 . . . 67
14.5 工具与 Agent：从概率输出到环境闭环 . . . 67

第 15 章 评测与可复现研究 68

15.1 先固定任务与被比较的概率 . . . 68
15.2 系统指标：从一条请求的时间线开始 . . . 68
15.3 污染、不确定性与生成判分 . . . 69
15.4 校准与解释：明确被预测的事件 . . . 70

15.5 安全行为需要在最终系统上重新验证 70
15.6 把论文主张变成可证伪实验 71
15.7 可选的十二周实践路线 71

第四部分 让机制接受独立检验 **73**

第 16 章 CPU 实验工作坊：把形状、缓存与量化跑起来 **74**
16.1 运行、配置与结果追踪 74
16.2 先验证注意力、头映射与位置 74
16.3 从预生成 QKV 到眞正的多层缓存 75
16.4 同一权重上的四种伪量化 76
16.5 参数、KV 与概率账本复算 76
16.6 哪些实验还没有运行 77

附录 A 符号、形状与公式速查 **78**
A.1 一套符号，明确的单位 78
A.2 行向量公式卡片 78
A.3 一眼判断精度标签还缺哪些信息 79

附录 B 账本使用与练习提示 **80**
B.1 资源估算以同一份矩阵清单为准 80
B.2 第 1 章：形状、损失与误差 80
B.3 第 2 章：输入输出协议 80
B.4 第 3、4 章：头、位置与完整前向 80
B.5 第 6 章：窄对称格点与尾组 81
B.6 第 9、11 章：长度与物理容量 81
B.7 第 14 章：递归状态与动作 81
B.8 第 15 章：一张速度表还缺什么 81

附录 C 文献地图与术语 **82**
C.1 沿问题阅读原始工作 82
C.2 常用术语的作用边界 82

参考文献 **84**

第一部分

从符号到一个完整的模型计算

第 1 章　张量、概率与误差：共同的数学语言

一段文字进入模型后，先变成整数，再变成向量；向量经过矩阵乘法、归一化与非线性，最后成为词表上的条件概率。后续的注意力、量化和推理系统都在改变这条计算链中的某些对象。理解一个方法，首先要能写出对象的形状、计算规则和误差度量。本章建立这些共同约定，不要求读者预先掌握张量库。

1.1　形状是计算的接口

标量 $a \in \mathbb{R}$ 只有一个数；向量有一个特征轴；矩阵有两个轴；张量则是任意多个有序轴的数组。轴的位置有意义。例如，$X \in \mathbb{R}^{B\times T\times d}$ 中，B 是批量中的序列数，T 是序列长度，d 是每个位置的隐藏维度。元素 $X_{b,t,j}$ 表示第 b 条序列、第 t 个 token、第 j 个特征的数值。把 T 与 d 交换，数据元素虽然没有增减，计算语义却变了。

本书统一采用**行向量乘右侧权重**：若 $x \in \mathbb{R}^{1\times d_{\text{in}}}$，线性层为 $y = xW$，其中 $W \in \mathbb{R}^{d_{\text{in}}\times d_{\text{out}}}$。同一个层应用于全部 token 时写成

$$Y = XW + b, \qquad Y \in \mathbb{R}^{B\times T\times d_{\text{out}}}. \tag{1.1}$$

这里矩阵乘作用在最后一个轴；$b \in \mathbb{R}^{d_{\text{out}}}$ 向每个 $[b,t,:]$ 广播。论文或框架可能存储转置后的权重，不能只比较变量名称，应比较其输入、输出轴。PyTorch 的线性层通常存储 $[d_{\text{out}}, d_{\text{in}}]$，前向实际使用其转置；这不改变式 (1.1) 的约定。

全书主要符号

N 为参数总数，B 为批量，T 为当前讨论的序列长度，V 为词表大小，L 为层数，d 为残差流宽度，d_{ff} 为前馈网络中间宽度。注意力使用 h_q 个 query 头、h_{kv} 个 KV 头，每头维度为 d_h；常见配置满足 $d = h_q d_h$。权重位宽写作 b_w，单位为 bit；存储量须区分十进制 GB 与二进制 GiB。量化的 scale、zero-point 分别写为 s, z。不同层、不同请求的长度不同时，会另行写出索引，不能默认它们相等。

把 X 的前两轴合并，可得 $X_{\text{flat}} \in \mathbb{R}^{(BT)\times d}$，这是线性层不混合位置的直接体现。重塑形状（reshape）只改变索引解释；交换轴（transpose/permute）改变轴的排列。先重排再重塑与直接重塑通常不是同一个操作。多头注意力从 $[B,T,h_q d_h]$ 变成 $[B,h_q,T,d_h]$ 时，必须先拆最后一轴，再交换头轴与位置轴。实际库还可能因内存步幅不连续而发生复制；逻辑形状与物理布局是两件事。

1.2　一次矩阵乘，三种不同的“乘”

设

$$X = \begin{bmatrix} 1 & 2 & -1 \\ 0 & 1 & 3 \end{bmatrix}, \qquad W = \begin{bmatrix} 1 & 0 \\ 2 & 1 \\ -1 & 2 \end{bmatrix}. \tag{1.2}$$

矩阵乘 $Y = XW$ 对输入特征求和，例如 $Y_{1,1} = 1 \cdot 1 + 2 \cdot 2 + (-1) \cdot (-1) = 6$，得到

$$Y = \begin{bmatrix} 6 & 0 \\ -1 & 7 \end{bmatrix}. \tag{1.3}$$

这里输出中的每个元素都依赖三个输入特征。逐元素乘 $A \odot G$ 则要求形状相同或能广播，不对特征求和；SwiGLU 用它让两条投影分支相互作用。行向量的点积 $qk^\top$ 返回一个标量，是注意力的匹配分数。三者的返回形状不同，不能把符号统一理解为“相乘”。

对于 $[m, k] \times [k, n]$，共有 mn 个长度为 k 的点积。按一次乘加计两次浮点操作的惯例，工作量约为 $2mkn$ FLOPs。若逐个计数而且不计从零开始的累加，则有 mkn 次乘法、$mn(k-1)$ 次加法，合计 $mn(2k-1)$；两种数字的区别来自计数约定。式 (1.3) 的精确计数为 12 次乘法与 8 次加法，常用 FLOPs 估算为 24。分析大模型时使用后者，但报告微小算例时应说清楚。

多头注意力的 $QK^\top$ 只转置 K 的最后两个轴，保留批量与头轴。若 Q 为 $[B, h_q, T_q, d_h]$、匹配后的 K 为 $[B, h_q, T_k, d_h]$，分数形状是 $[B, h_q, T_q, T_k]$。故意选 $T_q \neq T_k \neq d_h$ 调试，比让所有维度都等于 128 更容易发现轴错误。

1.3 广播与归约：scale 究竟与谁共享

对 $X \in \mathbb{R}^{B \times T \times d}$ 沿最后一轴求均值，保留维度后得到 $[B, T, 1]$。它可以广播回 X，使每个 token 共享一个归一化因子。若沿批量与位置两轴归约，则得到 $[1, 1, d]$，表示每个特征通道各有一个统计量。二者可能包含同样多的数学符号，却代表完全不同的共享关系。

以 $B = 2, T = 3, d = 4$ 为例，per-token scale 有 6 个数，通常存成 $[2, 3, 1]$；per-channel scale 有 4 个数，通常存成 $[1, 1, 4]$；per-tensor scale 只有一个数。分组量化还要声明分组沿哪个轴以及不整除时如何处理尾组。仅写“group size 128”无法判断哪些权重共用 scale。后文的误差与缓存开销都依赖这个定义。

掩码也通过广播应用。$[T_q, T_k]$ 的因果掩码可以共享给全部批量和头；而 padding、不同请求长度、前缀可见性可能要求 $[B, 1, T_q, T_k]$。一个可广播的形状并不一定是语义正确的形状：把所有请求都按最长序列可见，程序可能正常执行，却会访问无效内容。

1.4 从 logits 到概率：稳定 softmax

实数向量 $a \in \mathbb{R}^V$ 称为 logits，softmax 定义为

$$p_i = \frac{e^{a_i}}{\sum_{j=1}^{V} e^{a_j}}, \qquad p_i > 0, \quad \sum_i p_i = 1. \tag{1.4}$$

概率取决于 logits 的差值，而不取决于共同偏移。对任意常数 c，$\mathrm{softmax}(a + c\mathbf{1}) = \mathrm{softmax}(a)$。因此实际计算先令 $m = \max_i a_i$，再计算

$$p_i = \frac{e^{a_i - m}}{\sum_j e^{a_j - m}}, \qquad \mathrm{LSE}(a) = m + \log \sum_j e^{a_j - m}. \tag{1.5}$$

减最大值使指数的输入不大于零，避免 e^{1000} 这样的上溢。它无法消除所有舍入与下溢，但能避免一个本来有限的概率因为中间计算溢出而变成非数。

例如 $a = [1000, 1001]$ 与 $[0, 1]$ 给出相同的 $[0.268941, 0.731059]$。若真实标签为第二类，负对数似然是

$$\ell = \mathrm{LSE}(a) - a_2 = \log(1 + e^{-1}) \approx 0.313262. \tag{1.6}$$

不必先算一个可能舍入成零的概率再取对数。融合的 log-softmax/交叉熵实现正是利用这一关系。

softmax 对 logits 的导数为

$$\frac{\partial p_i}{\partial a_j} = p_i(\delta_{ij} - p_j). \tag{1.7}$$

当某个概率非常接近 1，其余接近 0 时，很多局部导数会变小。注意力中的 $1/\sqrt{d_h}$ 缩放与后续的数值精度选择，都要考虑这一敏感性。但小的概率变化也可能跨过两个候选 token 的排序边界；小范数误差与生成字符串一致不是同一要求。

1.5 条件概率、熵、交叉熵与困惑度

条件概率 $p(y \mid x)$ 是在已知 x 后对 y 的分布。期望 $\mathbb{E}_{y\sim p}[f(y)] = \sum_y p(y)f(y)$ 对所有可能结果加权，而一次抽样只产生其中一个结果。语言模型的概率评测与实际生成分别对应这两种观察方式。

使用自然对数时，熵、交叉熵与 KL 散度为

$$H(p) = -\sum_i p_i \log p_i, \tag{1.8}$$

$$H(p,q) = -\sum_i p_i \log q_i, \tag{1.9}$$

$$D_{\mathrm{KL}}(p\|q) = \sum_i p_i \log \frac{p_i}{q_i} = H(p,q) - H(p). \tag{1.10}$$

$H(p)$ 是分布自身的不确定性；$H(p,q)$ 是用 q 编码来自 p 的样本的平均代价。它们以 nat 为单位，换用以 2 为底的对数才以 bit 为单位。KL 不对称；当 $p_i > 0$ 而 $q_i = 0$ 时，$D_{\mathrm{KL}}(p\|q)$ 为无穷大，不能简单跳过该项。

取 $p = [1/2, 1/2]$、$q = [3/4, 1/4]$，则

$$H(p) = \log 2 \approx 0.693147, \quad H(p,q) = -\frac{1}{2}\log(3/16) \approx 0.836988, \tag{1.11}$$

所以 $D_{\mathrm{KL}}(p\|q) \approx 0.143841$。$q$ 对第一类更自信，却并不适合这个均匀的真实分布。自信程度与预测准确性是不同的量。

对 n 个参与评测的目标 token，平均损失与困惑度（perplexity，PPL）定义为

$$\bar{\ell} = -\frac{1}{n}\sum_{t\in\mathcal{J}} \log p_\theta(x_t \mid x_{<t}), \qquad \mathrm{PPL} = e^{\bar{\ell}}, \qquad n = |\mathcal{J}|. \tag{1.12}$$

$\mathcal{J}$ 是有效目标集合，不包括 padding 或被明确排除的标签。若两个真实 token 的预测概率分别为 1/2 与 1/4，则 $\mathrm{PPL} = \sqrt{8} \approx 2.828427$，并不是概率的算术平均再取倒数。

哪些 PPL 可以直接比较

同一文本更换 tokenizer 后会改变目标 token 的数量和条件分解；更换滑窗、文档边界、聊天模板或 loss mask 也会改变条件与分母。因此跨模型比较 PPL，至少要固定分词器、评测文本、上下文协议与有效 token 规则。PPL 较低说明该协议下真实 token 的平均对数概率较高，不保证事实正确、复杂推理更强或对话偏好更好。

1.6 把“误差很小”写成可检验的量

向量的二范数为 $\|x\|_2 = (\sum_i x_i^2)^{1/2}$，矩阵的 Frobenius 范数为 $\|X\|_F = (\sum_{ij} X_{ij}^2)^{1/2}$。对参考张量 Y 与近似张量 $\widehat{Y}$，本书常用

$$\mathrm{MSE}(Y,\widehat{Y}) = \frac{\left\|Y-\widehat{Y}\right\|_F^2}{|Y|}, \qquad e_{\mathrm{rel}} = \frac{\left\|Y-\widehat{Y}\right\|_F}{\|Y\|_F + \epsilon_{\mathrm{den}}}. \tag{1.13}$$

$|Y|$ 是元素数；ϵ_{den} 只为参考接近零时避免除零，必须与归一化层的 ϵ 区分并报告取值。实现一致性检查还会使用最大绝对误差和逐元素的绝对、相对容差。不同 dtype、归约顺序可能造成非零误差；要求字节完全相同并非总是合理。

余弦相似度 $xy^\top/(\|x\|_2\,\|y\|_2)$ 主要比较方向。$[1,0]$ 与 $[100,0]$ 的余弦为 1，但二者尺度相差百倍；零向量的余弦还没有定义。所以只报余弦相似度无法说明量化后的层输出是否保真。

更关键的是，权重误差不能脱离输入分布。设 $\Delta W = \widehat{W} - W$，固定校准输入矩阵为 X，则线性层的误差恰为

$$E_Y = X\Delta W, \tag{1.14}$$

$$\|E_Y\|_F^2 = \mathrm{tr}\big(\Delta W^\top X^\top X \Delta W\big). \tag{1.15}$$

例如 $X=[10,0]$，$W=[1,1]^\top$。两个同样具有 0.1 权重二范数误差的扰动 $[0.1,0]^\top$ 与 $[0,0.1]^\top$，分别造成 1 与 0 的输出误差。若真实输入后来大量使用第二个通道，校准结论也会改变。

式 (1.15) 中的 $X^\top X$ 是输入 Gram 矩阵；它是固定输入的层重构二次目标的 Hessian 的组成部分。目标是否含 1/2 或均值系数，会改变 Hessian 的常数因子。GPTQ 一类方法利用的是这种可计算的结构，不应把它直接等同于整个语言模型训练损失的精确 Hessian。校准数据、阻尼和顺序都会影响其近似效果，量化章会继续展开。

1.7 计算图与误差如何向后传播

对 $Y=XW$，上游梯度 $G=\partial\mathcal{L}/\partial Y$ 与 Y 同形，则

$$\frac{\partial\mathcal{L}}{\partial X} = GW^\top, \qquad \frac{\partial\mathcal{L}}{\partial W} = X^\top G. \tag{1.16}$$

因此训练不能只保存权重：反向需要前向的 X，要么保存它，要么重算它。这解释了激活重计算如何用额外计算换内存。推理时不需要梯度，但因果生成还需保留供未来访问的 K/V；二者生命周期不同。

残差层 $h_{\ell+1} = h_\ell + f_\ell(h_\ell)$ 同时传播已有扰动与本层新误差。沿用行向量约定，定义局部 Jacobian 的元素为 $(J_\ell)_{ij} = \partial(f_\ell)_j/\partial(h_\ell)_i$，$e_\ell$ 是本层引入的近似误差，则一阶关系为

$$\delta h_{\ell+1} \approx \delta h_\ell (I + J_\ell) + e_\ell. \tag{1.17}$$

若 $\delta h_0 = 0$，最终误差为若干 e_ℓ 经后续层按计算顺序作用后的和。以三层为例，

$$\delta h_3 \approx e_0(I+J_1)(I+J_2) + e_1(I+J_2) + e_2. \tag{1.18}$$

矩阵乘积通常不交换，误差也未必独立、零均值或持续变大。真实网络还可能跨越非线性区间，使一阶近似失效。故局部误差用于诊断，模型级损失与任务评测用于确认影响，二者不能相互替代。

1.8 练习与解题线索

1. 对 $X \in \mathbb{R}^{2\times3\times5}$、$W \in \mathbb{R}^{5\times7}$ 写出 XW 的形状和常用 FLOPs 估算。若再加 $[2,3,1]$ 的张量，解释它沿哪个轴广播。线索：合并批量与位置后是 6×5 乘 5×7，结果为 $[2,3,7]$，估算 420 FLOPs。
2. 设 logits 为 $[1002, 1000, 999]$，用减最大值的方法写出概率与真实标签为第三类时的损失。线索：归一化常数为 $1+e^{-2}+e^{-3}$，损失为 $3+\log(1+e^{-2}+e^{-3})$。
3. 数据集含两个序列，各有 2 与 8 个有效目标。为什么先求每条序列平均损失再平均，与按全部有效 token 求平均一般不同？线索：前者每条序列权重相同，后者每个 token 权重相同；须按评测目标选择。
4. 用 $X=[1,1]$ 比较 $\Delta W=[0.1,-0.1]^\top$ 与 $[0.1,0.1]^\top$ 的输出误差。线索：误差分别为 0 与 0.2；输入相关性决定哪些扰动方向被抵消。
5. 若残差层的 $J_\ell=0$，误差怎样传播？若 $J_\ell=-I$，一阶近似又如何？线索：前者保留已有误差并加新误差，后者会在这一局部线性模型中消除已有误差，说明“误差必然逐层放大”不是定律。

带着这些工具，下一章把文本分解为条件概率问题；再下一章则逐项写出产生这些概率所需的张量计算。后续遇到性能或压缩结论，可以回到三个问题：比较的对象同形吗，误差的分母相同吗，统计输入覆盖了实际使用条件吗？

第 2 章　从 token 到条件分布：模型的输入与输出

“模型读到一句话”包含三个不同步骤：把字符编码为离散 token，把 token ID 查表变成向量，再用网络产生下一个 token 的分布。分词规则决定预测问题的单位；训练目标决定怎样评价一个分布；采样规则决定怎样从分布得到文本。把这三层分开，才能解释同一权重为何在不同模板、解码设置和评测协议下表现不同。

2.1　自回归分解从哪里开始，到哪里结束

设 x_0 为序列开始标记 BOS，$x_1, \dots, x_T$ 为待建模的 token，末尾可以包含结束标记 EOS。条件概率的链式法则给出

$$p_\theta(x_{1:T} \mid x_0) = \prod_{t=1}^{T} p_\theta(x_t \mid x_{0:t-1}). \tag{2.1}$$

这里 θ 是全部模型参数。公式本身是概率恒等式；模型的能力限制来自如何用有限网络表示每个条件分布，以及训练数据是否覆盖需要的条件。decoder-only Transformer 使用因果可见图来计算这些分布 [1, 2]。

必须约定隐藏状态与标签的偏移：本书位置 t 的隐藏状态 h_t 已读入 x_t，用于预测 x_{t+1}。因此训练输入与标签可写为

$$\text{input} = [x_0, x_1, \dots, x_{T-1}], \qquad \text{target} = [x_1, x_2, \dots, x_T]. \tag{2.2}$$

若某个框架内部已经完成 shift，调用者再移一次会把任务变成“隔一个 token 预测”；若没有 shift，则模型可能直接复制当前输入获得很低的无效损失。阅读训练代码应追踪实际张量，而不凭接口名字判断。

EOS 是词表中的一种预测结果，控制模型何时认为序列结束；最大生成长度是外部截断条件；停止字符串则是解码后的文本规则。三者不能互相替代。一个答案恰好达到长度上限，并不说明模型赋予此处结束较高概率。数据集是否包含 EOS、多个文档如何拼接、文档之间是否允许互相注意，也都会改变式 (2.1) 中的条件。

2.2　Tokenization 决定离散坐标系

分词器将字符串映射为 ID 序列 $\mathrm{enc}(s) = (x_1, \dots, x_T)$；反向映射 dec 将 ID 序列还原为文本。token 可能对应字、词的一部分、若干字节或特殊控制符，不应把 token 数直接等同于汉字数或英文单词数。

BPE 从较小单元出发，依据训练得到的合并规则构造子词 [3]。例如一个教学用词表可将 `l,o,w` 先合成 `lo,w`，再合成 `low`；真实实现还需要合并优先级、空格/边界编码、预分词和字节映射等完整约定。Unigram 方法则从候选子词词表出发，使用概率模型选择切分。SentencePiece 提供不依赖预先按空格分词的处理流程，并支持 BPE 和 Unigram 等方法 [4]。WordPiece 与这些方法也不是同一个算法，不能仅因都输出子词而混用名称。

字节级覆盖或 byte fallback 可以提高未知字符串的可表示性，但是否能可逆地还原原始字节，取决于具体实现与规范化步骤。大小写转换、Unicode 规范化、空白清理可能使 $\mathrm{dec}(\mathrm{enc}(s)) \neq s$。中文、

代码缩进、组合字符、全角符号和不常见语言尤其值得做往返检查。分词前“看起来一样”的文本，也可能有不同字节表示。

为了理解成本差异，假设两个人工定义的分词器分别把“下雨了”编码成 [下, 雨, 了] 与 [下雨, 了]。它们对相同文本产生 3 与 2 个 token。这组构造用于说明：上下文窗口容纳的实际文本量、prefill 工作量、KV 缓存长度和每秒 token 的含义都依赖分词器。短 token 序列可能降低注意力与状态开销，较大词表却会增加 embedding 与输出头成本；不能只优化一个指标。

2.3 特殊 token 与聊天模板也是输入协议

基础语言模型接收一个 token 序列；聊天产品通常先把 system、user、assistant 等消息序列化。例如角色边界、消息结束、工具调用标记可能分别对应专用 token，也可能是普通文本片段。聊天模板规定角色顺序、换行和结尾是否附加 assistant 起始标记。

同一可见句子放在不同角色中，前面的控制 token 可能不同，因而条件分布也不同。把另一个模型的模板直接套过来，可能使模型继续补写用户消息、重复角色名或过早结束。评测与部署至少应记录 tokenizer 版本、特殊 token ID、模板文本或哈希、BOS/EOS 行为，以及是否允许输入文本被解释为特殊 token。模型权重和 tokenizer 必须配套；词表 ID 的交换并不是无害改名，因为 embedding 的每一行已有特定含义。

2.4 Embedding、最终归一化与输出头

令 embedding 表为 $E \in \mathbb{R}^{V\times d}$。ID x_t 的向量为

$$h_t^{(0)} = E[x_t, :] \in \mathbb{R}^{1\times d}. \tag{2.3}$$

这是查表，不要求对 ID 本身求梯度。查表算子的反向梯度只累加到被查到的权重行；共享输出头时，输出路径还可向其他行传递梯度，优化器状态及权重衰减也可能带来额外更新。不同 ID 的几何关系来自学习，而不是整数大小；ID 42 与 43 不会天然比 42 与 1000 更接近。

将全部 token 堆叠，输入张量是 $[B, T, d]$。经过 L 个 block 和架构规定的最终归一化后，得到 $\tilde{H} \in \mathbb{R}^{B\times T\times d}$。最终归一化与 block 内的归一化不是同一个操作，配置不同也不能擅自增删。语言模型头为

$$Z = \tilde{H}W_{\text{lm}} + b_{\text{lm}}, \qquad W_{\text{lm}} \in \mathbb{R}^{d\times V}, \quad Z \in \mathbb{R}^{B\times T\times V}. \tag{2.4}$$

许多模型省略 b_{lm}。权重共享（weight tying）令 $W_{\text{lm}} = E^{\top}$，因此在本书的行向量约定下必须有这个转置 [5]。不共享时两者合计约 $2Vd$ 个参数，共享时约 Vd；不能用“所有模型都有 Vd 个 embedding/head 参数”估算未共享模型。

取一个只有三个 token 的玩具词表，令

$$E = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ -1 & 0 \end{bmatrix}, \qquad \tilde{h} = [1, 2], \qquad W_{\text{lm}} = E^{\top}. \tag{2.5}$$

则 logits 是 $[1, 2, -1]$，原始模型概率约为 $[0.259496, 0.705385, 0.035119]$。如果真实下一 token 是第二个，损失约为 0.349012。embedding 的每一行在输入时充当表示，在输出时又作为匹配方向；这解释了共享的机制，但不意味着所有词的语义都可以从二维例子解释。

训练可能需要计算全部有效位置的词表分布，语言模型头的乘法约为 $2BTVd$ FLOPs。普通逐步

生成通常只需要当前最后位置，约为 $2BVd$。大词表时，输出头不是可以永远忽略的小层；而为了教学写出完整 softmax，也不意味着实现必须把所有历史位置的 logits 都长期保存。

2.5 训练损失：哪些位置参与平均

设 $y_{b,t}$ 是位置 $[b,t]$ 应预测的目标 ID，$m_{b,t} \in \{0,1\}$ 是损失掩码，$n = \sum_{b,t} m_{b,t}$。最常见的平均负对数似然为

$$\mathcal{L}(\theta) = -\frac{1}{n} \sum_{b,t} m_{b,t} \log p_\theta(y_{b,t} \mid x_{b,\le t}), \qquad n > 0. \tag{2.6}$$

padding 位置通常不参与损失；SFT 数据还可能只对 assistant 的目标 token 计损失。这里的**损失掩码**决定哪些预测接受监督，**注意力掩码**决定预测时能读哪些输入。把用户消息的 loss 设为零，并不意味着 assistant 不能注意用户消息。反过来，只在损失中忽略 padding，却让有效 token 注意随机 padding 内容，也可能改变结果。

对单个有效位置，令真实标签的 one-hot 向量为 y，原始 logits 为 a，则

$$\ell = -\sum_i y_i \log p_i, \qquad \frac{\partial \ell}{\partial a_i} = p_i - y_i. \tag{2.7}$$

真实 token 的概率低时，它的 logit 获得上升方向；其他 token 的概率共同被压低。平均损失还要乘 $m_{b,t}/n$。这是从条件概率到参数更新的联系，不需要把“语义理解”作为额外算子塞进计算图。

teacher forcing 在每个位置给模型真实前缀。对于序列 $[\mathrm{BOS}, a, b, c]$，可以并行计算三个条件 $p(a \mid \mathrm{BOS})$、$p(b \mid \mathrm{BOS}, a)$ 与 $p(c \mid \mathrm{BOS}, a, b)$；因果掩码保证第一个预测不能读取 a, b, c 的未来表示。输入张量一次包含完整序列与每个预测只能访问前缀并不矛盾。训练的并行性来自已知全部真实输入，而生成时下一个输入尚需通过模型选择。

生成使用模型自己产生的前缀，训练则通常使用数据中的前缀；两者的条件分布可能不同。早期一个错误 token 会改变后续条件，这与固定真实前缀下测得的平均损失有区别。不能据此断言 teacher forcing 无效，也不能认为低 PPL 足以保证长生成稳定；应同时检查适合任务的生成评测。

2.6 采样分布与模型分布应分开记录

原始模型分布是 $p_i = \mathrm{softmax}(a)_i$。温度 $\tau > 0$ 定义另一个分布

$$q_{\tau,i} = \frac{e^{a_i/\tau}}{\sum_j e^{a_j/\tau}}. \tag{2.8}$$

当 $\tau = 1$ 时二者相同；$\tau \to 0^+$ 时概率集中于最大 logit 所在集合；最大值并列时还需要明确打破平局规则。贪心解码直接取 $\arg\max_i a_i$，不必数值上把温度设成零再做除法。较高温度通常使分布更平坦，但“温度越高错误率一定越高”没有普遍保证，效果取决于任务、候选答案、重试机制与评分方式。

两类常用截断如下。

top-k 保留概率最高的 k 个 token，其余置零，再在保留集合内归一化。

top-p（nucleus）将概率降序排列，保留累积概率第一次达到或超过阈值 p 的最短前缀，再归一化。

边界 token 是否包含、至少保留几个候选，须以实际实现为准。

例如温度调整后的概率为 $[0.6, 0.25, 0.1, 0.05]$，top-k 取 $k = 2$ 与 top-p 取 $p = 0.8$ 都保留前两个，归一化后为 $[12/17, 5/17, 0, 0]$。但 top-p 的候选数会随每一步的分布改变，top-k 的候选数通常固定。两

者叠加、温度调整的先后顺序，以及重复惩罚施加在 logits 还是概率上，均可改变结果；仅记录一个 temperature 数字不够复现。

“采样没抽到”与“模型没有给高概率”

概率评测看真实答案在固定前缀下的概率；一次随机生成看某个样本；贪心看局部最高概率路径。三者检验的对象不同。增大采样次数可以提高至少命中一次某类答案的概率，却增加计算量，也不能自动改善单次回答质量。报告 pass@k 时，应同时给出 k、采样配置、预算和验证器，不能把它当成单次准确率。

2.7 为何逐步贪心不等于全序列最大概率

假设第一步 $p(a) = 0.6, p(b) = 0.4$，第二步 $p(\text{EOS} \mid a) = 0.2$、$p(\text{EOS} \mid b) = 0.9$，并假设任务只比较这两个以 EOS 结束的候选。贪心第一步选 a，但两个完整候选的概率分别是 0.12 与 0.36。最大概率的局部决定不必构成最大概率的完整序列。实际语言生成还包含更多继续路径、长度偏好与停止条件；beam search 扩展候选能缓解部分搜索问题，却不保证更符合开放式对话的偏好。

一个不使用缓存的最小生成过程可以写成：先对完整提示前缀前向，只取最后位置 logits；按指定规则选出下一 token；将该 ID 追加到前缀；若是 EOS 或达到外部限制则停止，否则重复。后续章节用 KV Cache 复用历史计算，目标是在确定性推理与一致的位置/掩码条件下保持同一个条件分布。采样器只应看到本步 logits，不应改变网络内部已计算的历史状态。

随机种子是复现的必要记录之一，但未必保证跨硬件、并行布局或不同库版本生成完全相同。接近的概率、不同归约顺序和随机数消耗次序都可能改变抽样结果。对模型数值一致性应先比较 logits 或概率，再比较离散生成；字符串差异本身不能定位是哪一步出了问题。

2.8 从接口到机制的检查清单与练习

开始解释一个模型的注意力前，先确认输入 ID 是否来自匹配的 tokenizer、消息模板是否一致、输出头是否共享、标签偏移在哪一层完成、loss mask 的有效分母为何。随后第 3 章会把“多层网络”展开成可手算的 decoder block，第 4 章再说明位置与缓存结构如何改变其计算。

1. 输入为 $[\text{BOS}, a, b]$，标签为 $[a, b, \text{EOS}]$。若只监督 b 与 EOS，写出损失掩码与 n。提示：$m = [0, 1, 1]$、$n = 2$；这不要求屏蔽对 a 的注意力。
2. 已有 $V = 32000, d = 4096$，计算不含 bias 时共享与不共享 embedding/head 的参数数目。提示：分别为 131072000 与 262144000，与 block 层数无关。
3. 两个 logits 为 $[0, \log 3]$。写出 $\tau = 1$ 与 $\tau = 2$ 的第二类概率。提示：分别为 3/4 与 $\sqrt{3}/(1+\sqrt{3})$；温度改变分布，不改变最大 logit 的位置。
4. 对 $[0.6, 0.25, 0.1, 0.05]$ 应用 top-$p = 0.95$ 时，需要保留几项？提示：数学上前三项恰达 0.95；实际浮点实现要注意累加误差和边界约定，记录代码行为。
5. 为什么比较压缩前后生成是否完全一致之前，应先固定聊天模板和解码器，再比较相同前缀下的 logits？提示：这样能分离输入协议变化、网络数值误差与抽样放大效应。

第 3 章　完整 decoder block：从形状到一次前向

上一章把模型写作从 token 到 logits 的函数。本章打开其中一个 block，不把注意力留在“检索”的比喻中，而是逐步写出输入、权重、掩码、非线性与残差的计算。使用一种常见的 pre-norm、无 bias、RMSNorm 与 SwiGLU 配置；它是明确的教学对象，不是所有 Transformer 的唯一结构。

3.1　两条子层和两次残差相加

令输入 $H \in \mathbb{R}^{B\times T\times d}$。一个 block 的计算图为

$$X = \mathrm{RMSNorm}_1(H), \tag{3.1}$$

$$A = \mathrm{MHA}(X), \qquad H' = H + A, \tag{3.2}$$

$$U = \mathrm{RMSNorm}_2(H'), \tag{3.3}$$

$$F = \mathrm{SwiGLU}(U), \qquad H^{\mathrm{next}} = H' + F. \tag{3.4}$$

注意力子层允许不同位置之间交换信息；前馈子层只在每个位置内部变换特征。参数在全部位置和批量样本间共享。两次相加要求残差分支的输出都回到 $[B, T, d]$，否则图上看似合理的“跳连”不能成立。

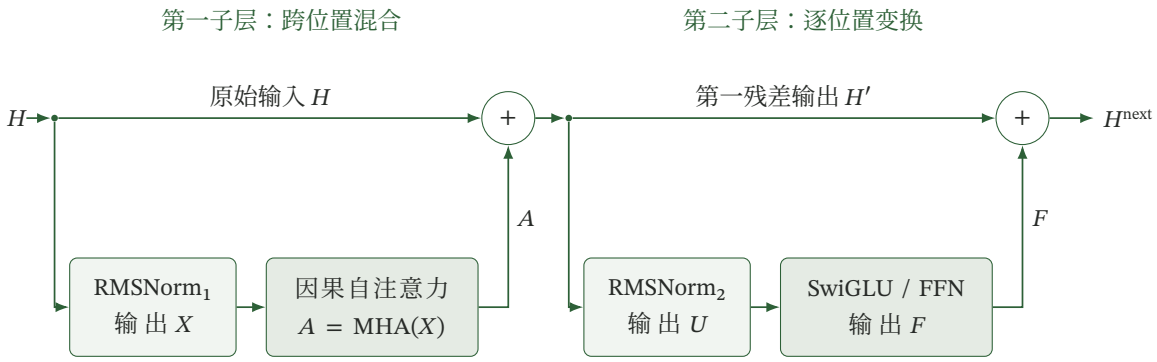


图 3.1　Pre-norm decoder block 的两条残差路径。$H, X, A, H', U, F, H^{\mathrm{next}}$ 均按式 (3.4) 定义，圆圈中的 + 表示逐元素相加；上方两条直路分别传递原始 H 与第一次相加后的 H'，不经过对应的 norm 或子层。根据归档原稿自绘图重排，未使用第三方插图。

Pre-norm 的归一化放在子层之前；post-norm 则在子层与残差相加之后归一化。两者的计算与梯度路径不同，不能靠移动同一个符号互换。常见 pre-norm 模型还在全部 L 个 block 后再放一个最终 norm，再接语言模型头。本文先说明推理前向；训练时的 dropout、初始化与残差缩放应按模型配置加入，不是为了让手算方便就能从真实模型中删除。

原始 Transformer 的 decoder 还包含对 encoder 输出的 cross-attention [1]。常见 decoder-only LLM 去掉该输入分支，使用因果自注意力进行序列生成；encoder-only 模型通常允许双向可见。架构选择与任务、数据、训练目标有关，decoder-only 的普遍使用不构成在所有任务上最优的定理。

3.2 RMSNorm 具体归一化什么

对单个 token 的行向量 $h \in \mathbb{R}^{1\times d}$，

$$\mathrm{RMSNorm}(h) = \frac{h}{r(h)} \odot \gamma, \qquad r(h) = \sqrt{\frac{1}{d}\sum_{j=1}^{d} h_j^2 + \epsilon}, \quad \gamma \in \mathbb{R}^d. \tag{3.5}$$

RMSNorm 沿特征轴归约，不跨不同 token 或不同请求；两个子层有各自学习的 γ [6]。LayerNorm 则先减均值，再除以标准差，通常还带平移参数 β。RMSNorm 不减均值，所以不能说“RMSNorm 后均值为零”，也不能说每个通道具有单位方差。

若 $\gamma = 1$、$\epsilon = 0$ 且 $h \neq 0$，输出的均方为 1。这不要求各坐标等大：$h = [2, 0, 0, 0]$ 的 RMS 是 1，归一化后仍为 $[2, 0, 0, 0]$。一般地，未乘 γ 的单个坐标绝对值至多为 $\sqrt{d}$，但当 d 很大时，这个界仍容许明显的通道极值。学习的 γ 又可改变各通道尺度，因此 norm 不等于去除量化异常值。

真实实现要为零或极小输入提供正的 ϵ，并说明平方、求和和乘除所用 dtype。低精度输入不意味着归约也必须低精度。Norm、softmax 与残差的数值误差会影响之后所有投影，后文的精度章会分别记账。

3.3 Q、K、V 是不同的可学习投影

先考虑标准 MHA，$h_q = h_{kv} = h$、$d = hd_h$。用归一化后的输入 X 计算

$$Q_0 = XW_Q, \qquad K_0 = XW_K, \qquad V_0 = XW_V, \qquad W_Q, W_K, W_V \in \mathbb{R}^{d\times hd_h}. \tag{3.6}$$

Q_0, K_0, V_0 均为 $[B, T, hd_h]$，拆分并重排成 $[B, h, T, d_h]$ 后，才进行每头的注意力。将 Q 理解为“想找的特征”、K 为“供匹配的特征”、V 为“匹配后返回的内容”有助于记忆；但三者没有固定的人类语义。尤其是 K 不需要等于 V，匹配依据与返回内容本来就可由不同投影编码。

以 $B = 2, T = 5, d = 12, h = 3, d_h = 4$ 为例，Q 的路径为

$$[2, 5, 12] \longrightarrow [2, 5, 3, 4] \longrightarrow [2, 3, 5, 4]. \tag{3.7}$$

分数形状是 $[2, 3, 5, 5]$，其最后两个轴才是 query 与 key 位置。若只把 $[2, 5, 12]$ 直接 reshape 成 $[2, 3, 5, 4]$，元素顺序通常错误，即使形状看似完全符合公式。

位置机制会在这里介入，例如 RoPE 旋转 Q/K 后再算分数；为了先分离各个操作，下面先令位置旋转为恒等，第 4 章再加入它。标准 RoPE 通常不旋转 V；历史 V 也不能统称为“位置旋转后的输入”。

3.4 因果注意力的四个计算步骤

对一个头，$Q, K, V \in \mathbb{R}^{T\times d_h}$，定义

$$S = QK^\top/\sqrt{d_h}, \tag{3.8}$$

$$M_{ij} = \begin{cases} 0, & j \le i, \\ -\infty, & j > i, \end{cases} \tag{3.9}$$

$$P_{ij} = \frac{\exp(S_{ij} + M_{ij})}{\sum_{k\le i}\exp(S_{ik})}, \qquad O = PV. \tag{3.10}$$

第一步算相似度，第二步限制可见位置，第三步沿 *key* 位置轴归一化，第四步对 V 加权。每个有效 query 行的权重和为 1，被屏蔽位置的权重为 0。这里的 P 是注意力权重，不是词表上的下一 token 概率。

$1/\sqrt{d_h}$ 的常见动机是：若各坐标的 q_j, k_j 独立、均值零、方差一，而且不同坐标间也独立，则 $\sum_j q_j k_j$ 的方差为 d_h；缩放后方差为一。训练后的 Q/K 不必满足这些理想假设，因此这个推导解释尺度设计，不保证训练中任何层的分数方差恰好等于一。QK-norm 等变体会进一步改变分数尺度。

因果掩码应按逻辑位置定义。一次完整序列的方阵中它是下三角；缓存后新 query 可能只有一行，却可见全部历史 keys，不能把局部行号 0 错当作全局位置 0。padding、滑窗、前缀和跨序列隔离还可能叠加额外可见性条件。

全部被屏蔽的行不是一个有效概率分布

若某个 query 没有任何可见 key，所有分数都是 $-\infty$，softmax 的减最大值会出现未定义运算。用一个很负的有限数替换全部分数，又可能得到无意义的均匀分布。正确做法是根据接口约定跳过无效 query、保证有效行至少有一个可见 key，或显式定义无效行输出，并验证它不会进入有效损失或缓存。不可只因程序没有报错就认为 mask 正确。

注意力输出是 value 的连续混合。例如 $q = [1, 0]$，$k_1 = [1, 0], k_2 = [0, 1]$，$v_1 = [2, 0], v_2 = [0, 4]$，使用 $d_h = 2$ 的真实缩放时，权重约为 $[0.669762, 0.330238]$，输出约为 $[1.339523, 1.320954]$。若为了直觉省略缩放，会得到另一组数字；不能把未缩放的手算结果拿去检验缩放版本代码。

3.5 多头拼接、输出投影与参数量

对头 $a = 1, \dots, h$ 分别得到 $O^{(a)} \in \mathbb{R}^{B \times T \times d_h}$，按特征轴拼接，再右乘输出矩阵：

$$A = \mathrm{Concat}(O^{(1)}, \dots, O^{(h)}) W_O, \qquad W_O \in \mathbb{R}^{h d_h \times d}. \tag{3.11}$$

每个头有自己的相似度度量和值投影；W_O 再混合来自不同头的结果。多头不保证每个头自动拥有清晰的“语法角色”，也不意味着在实现中将最后一轴切开就等于把原始特征永久分割：前面的 W_Q, W_K, W_V 已经可以混合全部输入特征。

MHA 的四个投影共有约 $4d^2$ 个参数。GQA 使用 $h_{kv} < h_q$，Q 的输出仍为 $h_q d_h$，K/V 只输出 $h_{kv} d_h$，因此在 $d = h_q d_h$ 时，投影参数为

$$N_{\text{attn}} = 2d^2 + 2d h_{kv} d_h. \tag{3.12}$$

GQA 的分组映射与缓存收益留到下一章；它不改变残差流宽度，也不把逻辑 query 头数减少为 h_{kv}。

3.6 SwiGLU 的两条输入分支

经典前馈网络先扩张、非线性、再投回：$\mathrm{FFN}(U) = \sigma(UW_1)W_2$。SwiGLU 改用两个投影，并通过逐元素乘形成门控 [7]：

$$G = UW_g, \qquad J = UW_u, \tag{3.13}$$

$$Z = \mathrm{SiLU}(G) \odot J, \qquad \mathrm{SiLU}(x) = \frac{x}{1 + e^{-x}}, \tag{3.14}$$

$$F = ZW_d. \tag{3.15}$$

$W_g, W_u \in \mathbb{R}^{d \times d_{ff}}$，$W_d \in \mathbb{R}^{d_{ff} \times d}$，$G, J, Z$ 均为 $[B, T, d_{ff}]$。SiLU 不是 sigmoid：它可以为负，大正值时近似线性，所以“门”不是永远位于 $[0, 1]$ 的开关。它与另一分支相乘还可能放大极值，激活量化需

表 3.1 本章 block 的主要张量；批量与位置轴不省略

对象	形状	操作或作用
H, X, H', U	$[B, T, d]$	残差与逐 token 归一化
Q	$[B, h_q, T, d_h]$	Q 投影后拆头
K, V	$[B, h_{kv}, T, d_h]$	K/V 投影；MHA 时 $h_{kv} = h_q$
S, P（逻辑）	$[B, h_q, T, T]$	可分块计算，不必完整物化
各头输出	$[B, h_q, T, d_h]$	由 P 聚合 V
拼接与 A	$[B, T, h_q d_h]$、$[B, T, d]$	重排、拼接、输出投影
G, J, Z	$[B, T, d_{ff}]$	两个前馈分支与逐元素门控
F, H^{next}	$[B, T, d]$	下投影与第二次残差

分别检查乘法前后分布。

不含 bias 时，SwiGLU 有 $3dd_{ff}$ 个参数。令 $d_{ff} \approx 8d/3$，即可与传统 $d_{ff} = 4d$、两个矩阵的 FFN 维持约 $8d^2$ 的参数规模；实际模型常为硬件对齐调整宽度。这是参数预算上的对应关系，不是 SwiGLU 最优宽度的数学证明。

3.7 一组可以完整复算的双头前向

下面不是训练过的模型，而是全部权重已指定的合成教学算例。取 $B = 1, T = 2, d = 4, h_q = h_{kv} = 2, d_h = 2, d_{ff} = 2$。省略唯一的批量轴，输入为

$$H = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix}. \tag{3.16}$$

两次 RMSNorm 的 γ 全为一。为使第一步可手算，仅本算例令 $\epsilon = 0$，各行均非零；真实实现使用正 ϵ。本算例也关闭 dropout，不加 RoPE，无任何 bias，$d_{ff} = 2$ 仅为了缩小表格，不代表真实模型的常见扩张比。

3.7.1 归一化、QKV 与每头分数

H 每行的 RMS 均为 1，所以 $X = H$。取

$$W_Q = W_K = I_4, \quad W_V = \operatorname{diag}(1, 2, 3, 4), \quad W_O = \tfrac{1}{2} I_4. \tag{3.17}$$

因此

$$Q_0 = K_0 = H, \qquad V_0 = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & -2 & 3 & -4 \end{bmatrix}. \tag{3.18}$$

第一个头取前两维，第二个头取后两维。两个头的 Q/K 恰好相同，均为 $\left[\begin{smallmatrix} 1 & 1 \\ 1 & -1 \end{smallmatrix}\right]$；V 分别为 $\left[\begin{smallmatrix} 1 & 2 \\ 1 & -2 \end{smallmatrix}\right]$ 与 $\left[\begin{smallmatrix} 3 & 4 \\ 3 & -4 \end{smallmatrix}\right]$。头不是同一个对象，只是在这组输入与权重下部分数值重合。

每个头未加 mask 的分数均为 $\left[\begin{smallmatrix} \sqrt{2} & 0 \\ 0 & \sqrt{2} \end{smallmatrix}\right]$。加因果 mask 后，第一行只可见第一个位置，故

$$P = \begin{bmatrix} 1 & 0 \\ p & 1-p \end{bmatrix}, \qquad p = \frac{1}{1 + e^{\sqrt{2}}} \approx 0.1955703175. \tag{3.19}$$

这一步同时检查缩放、softmax 的轴和未来位置屏蔽。若第一行给第二个位置任何非零权重，就不是这个因果前向。

3.7.2 聚合、拼接与第一次残差

各头的输出为

$$O^{(1)} = \begin{bmatrix} 1 & 2 \\ 1 & 2(2p-1) \end{bmatrix}, \qquad O^{(2)} = \begin{bmatrix} 3 & 4 \\ 3 & 4(2p-1) \end{bmatrix}. \tag{3.20}$$

拼接得到

$$C = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & -1.2177187300 & 3 & -2.4354374601 \end{bmatrix}. \tag{3.21}$$

注意力输出是 $A = C/2$，并非直接把 C 加入残差。相加得到

$$H' = \begin{bmatrix} 1.5 & 2 & 2.5 & 3 \\ 1.5 & -1.6088593650 & 2.5 & -2.2177187300 \end{bmatrix}. \tag{3.22}$$

输入 H 通过旁路保留了下来；而注意力分支的第一行虽只读取自身，其特征也因 W_V, W_O 被改变，不能把“只看自己”等同于“输出不变”。

3.7.3 第二次归一化、SwiGLU 与最终输出

H' 两行的 RMS 分别约为 2.3184046239 和 2.0004190075。逐行相除后，

$$U \approx \begin{bmatrix} 0.6469966392 & 0.8626621856 & 1.0783277320 & 1.2939932784 \\ 0.7498429051 & -0.8042611868 & 1.2497381752 & -1.1086271035 \end{bmatrix}. \tag{3.23}$$

给出全部前馈权重，避免只指定一个输出倒推过程：

$$W_g = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}, \quad W_u = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad W_d = \begin{bmatrix} \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \end{bmatrix}. \tag{3.24}$$

于是 G 取 U 的前两列，J 取后两列。逐元素计算 $Z = \mathrm{SiLU}(G) \odot J$，得到

$$Z \approx \begin{bmatrix} 0.4579069838 & 0.7849859285 \\ 0.6364312414 & 0.2756146693 \end{bmatrix}. \tag{3.25}$$

第二行第二个 gate 与 up 都为负，$\mathrm{SiLU}(G)$ 仍为负，乘积因而为正；这体现了不能把 SiLU 当成非负开关。最后 $F = ZW_d$，即把两列各乘 1/2 并分发回四个通道。再次加入未经第二次 norm 覆盖的 H'，得到

$$H^{\text{next}} \approx \begin{bmatrix} 1.7289534919 & 2.3924929643 & 2.7289534919 & 3.3924929643 \\ 1.8182156207 & -1.4710520303 & 2.8182156207 & -2.0799113954 \end{bmatrix}. \tag{3.26}$$

这就是完整 block 的结果，尚未经过后续 block、最终 norm 或语言模型头。若把 U 而不是 H' 加回去，或者漏掉 W_O，结果都会改变。

该算例的固定权重与高精度中间结果保存在项目的 `experiments/hand_block_case.json`。实验应先按这些数值逐步比较，再扩展到随机矩阵、不同头数与缓存场景。表中十位小数经过四舍五入，不能以零容差比较；应读取 JSON 中未手工截断的双精度参考并按实验协议判断。

3.8 从一次前向推到参数、计算与状态

不含 norm 与 bias 时，一个采用 GQA 的 SwiGLU block 参数约为

$$N_{\text{block}} = 2d^2 + 2dh_{kv}d_h + 3dd_{ff}. \tag{3.27}$$

这解释了 FFN 在许多模型中占较大权重存储，但具体比例须代入配置。每个 token 的投影与 FFN 工作量约为上述权重数的两倍 FLOPs；完整序列还需 QK 点积与 V 聚合。对 MHA，按计算完整方阵的惯例，两者合计约 $4BT^2d$ FLOPs。严格利用因果三角可减少约一半相关算术，但分块边界与 kernel 决定实际执行量。因此复杂度为 $\mathcal{O}(BTd^2) + \mathcal{O}(BT^2d)$，不代表任意长度下二次项都占主导。

前向中的 X, Q, G, Z 等是与输入有关的激活；模型权重在请求间共享；自回归生成还要把每层历史 K/V 保留给以后的位置。为何缓存 K/V 而通常不缓存 Q？新位置只需用自己的 query 查询历史 keys，并聚合历史 values；旧 query 对应的旧输出已经完成，不参与新位置的计算。在固定权重、固定位置/掩码和推理模式下，因果性保证追加未来 token 不改变历史 token 的逐层表示，所以历史 K/V 可复用。双向注意力、改写前缀、变更位置映射或依赖全序列统计的算子可能破坏这一条件。

注意力在公式中有 $T \times T$ 的逻辑权重，但不要求把整个权重矩阵写入显存。后续的 FlashAttention 利用分块与在线归约计算同一数学输出；它改变数据移动策略，不把稠密注意力的二次算术变成线性。分清模型定义、逻辑张量、物理实现，才不会从一张形状表直接推出峰值内存或真实延迟。

3.9 练习与验证方法

1. 修改手算案例中第二个输入 token，证明第一个 token 的 block 输出不变。提示：第一个位置的注意力只见自身，norm 与 FFN 也逐 token；这是检测未来泄露的独立性质，而不只是重跑同一公式。
2. 在手算案例中把 W_O 改成 I_4。哪些量保持不变，哪些必须重算？提示：QKV、分数与 P 不变；第一次残差及以后均改变。
3. 若 $d = 4096, d_{ff} = 11008, h_q = 32, h_{kv} = 8, d_h = 128$，写出三个注意力输入投影与三个 SwiGLU 权重的形状。提示：K/V 为 $[4096, 1024]$，Q 为 $[4096, 4096]$；gate/up 为 $[4096, 11008]$，down 为其转置形状。
4. 对 $[2, 0, 0, 0]$ 加一个很小的扰动，比较 RMSNorm 前后各坐标的尺度。能否由“均方接近一”推出逐通道易于量化？提示：不能；归约约束的是整体，极值通道和不同 token 的分布仍需检查。
5. 为什么仅验证缓存版与非缓存版输出一致，仍不足以证明因果 mask 正确？提示：二者可能共享同一种错误；应加入修改未来 token 不影响过去、第一行只见自身、手算固定案例等独立约束。

现在 block 已从黑箱变为完整计算链。下一章保持这套行向量与形状约定，逐项讨论位置、KV 共享与状态表示；之后的量化与系统章节则研究同一计算链如何在有限精度和有限带宽中实现。

第 4 章　位置与注意力家族：表示、可见图和缓存

上一章已经把一个 decoder block 写成了形状明确的计算。本章保持同样的行向量约定，加入位置，并比较不同注意力实际改变的对象：有的方法共享 K/V，有的方法限制可见位置，有的方法把历史聚合成状态，有的方法从训练架构上改变缓存的参数化。只有把这些差别写进公式，才能判断它节省的是参数、运算、持久状态，还是执行中的数据移动。

4.1　位置与掩码承担不同的工作

先忽略位置表示和掩码。设 $X \in \mathbb{R}^{T\times d}$，$P$ 是重排 token 行的置换矩阵。线性投影满足 $(PX)W = P(XW)$，而按行计算的 softmax 满足

$$\operatorname{softmax}(PSP^{\top}) = P\operatorname{softmax}(S)P^{\top}. \tag{4.1}$$

因此无位置、无掩码的自注意力具有置换等变性：重排输入行，输出也按同样方式重排。它能比较内容，却没有独立的距离坐标。这个结论不能原样用于固定因果掩码：若只重排 X 而保持原来的下三角可见图，通常不满足等变关系。因果掩码已经提供方向和边界信息，但它不等于为每对 token 显式提供精确距离。

几种可见图应先分开。双向注意力允许每个 query 访问左右两侧；因果注意力只允许位置 t 访问 $j \le t$；cross-attention 的 Q 来自一个序列、K/V 来自另一个序列；prefix 掩码可以让前缀内部双向可见，而续写部分仍保持因果。位置表示决定如何使用位置坐标，掩码决定哪些边合法。改变其中任一个，都可能改变模型函数。

绝对位置向量通常加在输入表示上；相对位置 bias 加在注意力 score 上；RoPE 作用于 Q/K 的二维子空间。它们可以与不同可见图组合。分析实现时，必须同时记录位置编号、旋转或 bias 规则、padding 规则和可见图，不能用一句“支持位置编码”代替这些接口。

4.2　RoPE：把相对位置写进 Q/K 点积

本书将每个 query/key 写成行向量。为使行向量右乘表示逆时针旋转，定义

$$R(\alpha) = \begin{bmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{bmatrix}, \qquad R(\alpha)^{\top} = R(-\alpha). \tag{4.2}$$

这与常见的列向量旋转矩阵互为转置。两种约定都可以使用，但不能把一方的矩阵与另一方的乘法顺序混用。固定一个频率 θ，令 $R_m = R(m\theta)$，则位置 m, n 的旋转表示为 qR_m, kR_n，满足

$$(qR_m)(kR_n)^{\top} = qR_mR_n^{\top}k^{\top} \tag{4.3}$$

$$= qR((m-n)\theta)k^{\top}. \tag{4.4}$$

所以对固定的未旋转内容向量 q, k，位置对点积的显式作用只依赖差值 $m - n$[8]。论文若采用列向量，常写成 $q^{\top}R((n-m)\theta)k$；这只是约定不同，不能据此判断某一实现符号反了。

若旋转维度为偶数 $d_r \le d_h$，可将前 d_r 个特征两两配对，使用块对角矩阵

$$\mathcal{R}_m = \operatorname{diag}\big(R(m\theta_0), \dots, R(m\theta_{d_r/2-1}), I_{d_h-d_r}\big), \qquad \theta_i = \beta^{-2i/d_r}. \tag{4.5}$$

β 是频率基数；全维旋转时 $d_r = d_h$，最后的恒等块省略。相邻特征配对和前后半轴配对是常见的两种布局；若同步重排权重与特征，它们可以表达同一计算，但在读取 checkpoint 时不能任意互换。部分模型还采用不同频率或缩放规则，应以配置和代码为准。

标准用法先计算内容投影，再对 Q/K 旋转：$\tilde{q}_m = q_m\mathcal{R}_m$、$\tilde{k}_n = k_n\mathcal{R}_n$；V 通常不旋转。旋转保持各向量的二范数，但改变不同位置 Q/K 的夹角。随后仍按第 3 章计算 score、掩码、softmax 和加权 V。旋转不是给每个 token 贴一个可直接读出的“位置标签”，也不保证不同相对距离具有不同分数。

一个二维算例与一个可检验恒等式

取 $q = [1,0]$、$k = [1,1]$、$\theta = \pi/2$。当 $m = 0, n = 1$ 时，$qR_0 = [1,0]$、$kR_1 = [-1,1]$，点积为 -1。同时右移一个位置，得到 $qR_1 = [0,1]$、$kR_2 = [-1,-1]$，点积仍为 -1。两次的相对差值都为 -1。若 $m = n = 0$，点积则为 1。

这里验证的是式 (4.4)。它没有证明整个 decoder 对平移不变：内容向量本身来自带边界和因果上下文的网络，padding、窗口和特殊 token 也可能改变可见条件。CPU 检查应固定内容向量，分别比较两种计算路径，而不是直接要求整段模型输出平移后相同。

4.3 长上下文：可计算、可存储与可使用是三件事

式 (4.5) 能在训练长度之外继续求正弦和余弦，并不意味着模型在这些长度上已学会可靠地检索或组合信息。频率、相位组合、训练中实际出现的相对距离、注意力分布和任务的数据分布，都影响外推。单个旋转分量具有周期性；多个频率共同作用也不构成任意长度上质量不变的保证。不能把“模型可接收某长度”直接写成“模型能利用该长度内所有内容”。

一种位置插值是把 m 换为 $f(m) = m/s$，$s > 1$。这将最大相位压到较短范围，但也把相邻位置的角度差缩小为原来的 $1/s$。改变基数或分频段调整频率，则对不同子空间分配不同尺度。它们都改变了训练时的几何关系，需要通过相应训练或验证判断收益，不能只凭公式宣称无损扩展。若 f 不是线性的，点积中的角度差为 $f(m) - f(n)$，未必只由 $m - n$ 决定。

缓存与位置方案还必须保持一致。常见实现缓存已经旋转的 K，因此其值绑定于当时的位置编号、频率和缩放配置。若服务在长度越过阈值后动态更改频率，却仍直接读取旧 K，就不再是同一个统一位置规则下的注意力。实现应固定请求的规则，或正确更新受影响状态；是否缓存未旋转 K 又会改变计算与存储路径。分块 prefill 也要延续全局位置，不能每块从零开始。

长上下文验证至少包含四个层次：输入协议能否表达，显存与执行能否承受，给定长度下的似然是否稳定，以及任务是否能使用远处的信息。除总体分数外，还应固定证据内容、移动其位置，改变干扰长度，并检查近距离能力是否退化。这里的长度既包括提示也包括生成过程中增长的上下文。第 11 章给出资源账本，KV 章节讨论状态压缩；两者都不能代替质量测试。

4.4 MHA、MQA 与 GQA：哪些头共享历史

设输入为 $X \in \mathbb{R}^{B\times T\times d}$，query 头数为 h_q、KV 头数为 h_{kv}，每头维度为 d_h。使用

$$W_Q \in \mathbb{R}^{d\times h_q d_h}, \qquad W_K, W_V \in \mathbb{R}^{d\times h_{kv} d_h}. \tag{4.6}$$

投影并重排后，Q 为 $[B, h_q, T, d_h]$，K/V 为 $[B, h_{kv}, T, d_h]$。MHA 取 $h_{kv} = h_q$；MQA 取 $h_{kv} = 1$；常用等组 GQA 要求 h_q 能被 h_{kv} 整除 [9, 10]。若 $g = h_q/h_{kv}$，从零开始编号的 query 头 i 对应

$$\pi(i) = \lfloor i/g \rfloor, \quad o_{t,i} = \sum_{j \le t} p_{t,i,j} v_{j,\pi(i)}, \quad p_{t,i,:} = \mathrm{softmax}_j\left(\frac{\tilde{q}_{t,i}\tilde{k}_{j,\pi(i)}^{\top}}{\sqrt{d_h}} + M_{t,j}\right). \tag{4.7}$$

其他分组映射也可能存在，但必须由架构或布局明确指定；并不是 K/V 头数较少时任意一次广播都正确。

取 $h_q = 4, h_{kv} = 2, d_h = 2, B = 1, T = 3$，头映射为 $[0, 0, 1, 1]$。Q 的形状是 $[1, 4, 3, 2]$，K/V 是 $[1, 2, 3, 2]$；逻辑 score 仍为 $[1, 4, 3, 3]$。头 0 和 1 虽共享 K/V，却有不同 Q，因此概率及输出一般不同。将 K/V 按映射显式复制到四个头，可以作为参考检查；高效 kernel 不必保存这份重复数据。若头顺序应为 $[0, 0, 1, 1]$ 却复制成 $[0, 1, 0, 1]$，形状完全相同，函数却已经改变。

每层每 token 的 K/V 元素数为 $2h_{kv}d_h$。固定 $h_q = 32, d_h = 128$，从 MHA 改为 $h_{kv} = 8$ 的 GQA，元素数从 8192 降至 2048；若都以两字节保存，则分别为 16 和 4 KiB。这是有效状态底数，不含 scale、页表或分配浪费。K/V 投影参数也随 h_{kv} 下降，但 Q、输出投影和 FFN 并未同时缩小。

GQA 没有把 query 头数减少为 h_{kv}。给定可见长度 S，逻辑上仍要计算 h_q 组注意力，QK 与 PV 的主要工作量仍约为 $4h_qSd_h$ FLOPs。实际 HBM 读取能否充分复用共享 K/V，取决于实现。把现有 MHA 权重的 K/V 头平均或合并，也不自动保持输出；原 GQA 工作研究了从多头 checkpoint 继续训练的转换方案，不能由此推断任意模型都能直接无损替换。

4.5 稀疏注意力：用可见图控制边数

令第 t 个 query 的合法 key 集合为 $\mathcal{E}_t$，注意力只在该集合上归一化：

$$p_{t,j} = \frac{\exp(s_{t,j})}{\sum_{u \in \mathcal{E}_t} \exp(s_{t,u})}, \quad j \in \mathcal{E}_t. \tag{4.8}$$

稀疏化既省掉某些点积，也改变了分母和信息路径。只看最后 w 个位置时，整段核心计算约为 $O(h_qTwd_h)$，单步约为 $O(h_qwd_h)$。若该层永远不再访问窗口外的位置，旧 K/V 可以删除，持久状态有望限制在 $2h_{kv}wd_h$ 个元素；如果未来的全局 query、检索规则或其他分支仍可能访问它们，就不能据窗口计算量直接推断缓存也有相同上界。

局部、块稀疏、少数全局 token 与混合全局层提供不同的可见图。局部层可以跨层传播远处信息，但需要多跳，不能等同于一次直接内容寻址；保留全局层则仍要为这些层保存相应历史。数据依赖的稀疏方案还要支付选边或检索成本。逻辑上只保留一小部分边，只有当 kernel 真正跳过它们并保持适合硬件的布局时，才可能转化为执行收益。

4.6 线性注意力：归一化与前缀状态缺一不可

考虑一种非负特征映射 $\phi : \mathbb{R}^{d_h} \to \mathbb{R}^r$，并以核 $\kappa(q, k) = \phi(q)\phi(k)^{\top}$ 代替 softmax 的指数相似度。某些方法把它作为近似，另一些直接训练这种注意力；固定的有限维映射通常不能据结合律就宣称与标准 softmax 完全相同 [11]。省略批量与头轴，令 V 的每头维度为 d_v，因果归一化输出为

$$o_t = \frac{\sum_{j \le t}[\phi(q_t)\phi(k_j)^{\top}]v_j}{\sum_{j \le t}\phi(q_t)\phi(k_j)^{\top}}. \tag{4.9}$$

定义两个前缀状态

$$A_t = A_{t-1} + \phi(k_t)^\top v_t, \qquad A_t \in \mathbb{R}^{r\times d_v}, \tag{4.10}$$

$$z_t = z_{t-1} + \phi(k_t), \qquad z_t \in \mathbb{R}^{1\times r}, \tag{4.11}$$

$$o_t = \frac{\phi(q_t)A_t}{\phi(q_t)z_t^\top}, \qquad A_0 = 0, \quad z_0 = 0. \tag{4.12}$$

外积 $\phi(k_t)^\top v_t$ 先把历史聚合，再由当前 query 读取。因果模型允许关注自身，所以先写入第 t 项再查询；若任务要求严格排除自身，则要改用 A_{t-1}, z_{t-1}。分母必须为正；对零分母的约定与额外 ϵ 都应明确，加入 ϵ 会改变上述精确函数。

取 $r = 2, d_v = 1$，已有 $\phi(k_1) = [1,0], v_1 = 2$ 与 $\phi(k_2) = [0,1], v_2 = 4$，得到 $A_2 = [2,4]^\top, z_2 = [1,1]$。对 $\phi(q) = [1,3]$，输出为 $(2+12)/(1+3) = 3.5$。若只写 $\phi(q)A_2$，得到的是未归一化的 14，并非同一个注意力。

每头固定状态含 $r(d_v+1)$ 个元素，单步核心更新与读取为 $O(rd_v)$，整段为 $O(Trd_v)$，另加特征映射成本。固定大小的状态是把历史信息聚合后的表示，不再保留可独立访问的全部 K/V；删除某个任意历史项、精确回滚或执行任意稀疏掩码可能需要额外状态。有限特征维数、归一化稳定性和长期累积误差都会限制效果。加法前缀和可以用于并行实现，但并行归约与逐步累加的浮点结果仍可能略有不同。更复杂的门控递推、SSM 和混合层见架构章节，不能把它们统称为同一种线性注意力。

4.7 MLA：缓存低维 latent 与解耦的位置部分

MLA 在训练架构中联合参数化 K/V。以下用行向量说明 DeepSeek-V2 中与缓存有关的机制，省略 query 的低秩投影与某些归一化细节 [12]。令输入行 $x_t \in \mathbb{R}^{1\times d}$，压缩表示为

$$c_t = x_t W_D \in \mathbb{R}^{1\times d_c}, \qquad k^C_{t,i} = c_t U_i^K, \qquad v_{t,i} = c_t U_i^V, \tag{4.13}$$

其中 $W_D \in \mathbb{R}^{d\times d_c}$、$U_i^K \in \mathbb{R}^{d_c\times d_h^C}$、$U_i^V \in \mathbb{R}^{d_c\times d_h^V}$。若实际模型在压缩后还有归一化，应把它包括在 c_t 的定义中；不能在迁移 checkpoint 时省掉。对已经如此参数化并训练的模型，上式定义其表示，不是对任意 MHA 的事后无损低秩分解。

若内容分量不加位置旋转，结合律给出

$$q^C_{t,i}(k^C_{j,i})^\top = \underbrace{q^C_{t,i}(U_i^K)^\top}_{1\times d_c} c_j^\top, \tag{4.14}$$

$$\sum_j p_{t,i,j} v_{j,i} = \left(\sum_j p_{t,i,j} c_j\right) U_i^V. \tag{4.15}$$

因此 decode 可以在 latent 空间匹配和汇聚，不必把每个历史 token 的全部各头 K/V 长期展开。U_i^V 还可以与该头对应的输出投影合并。代价是点积和汇聚此时围绕 d_c 工作，缓存 bytes 降低不等于每个算子的 FLOPs 都降低。

普通 RoPE 若直接旋转恢复的内容 K，会在内容 query 与上投影之间插入位置相关矩阵，通常不能将它一次性吸收到固定权重。DeepSeek-V2 采用解耦的 RoPE 部分：每头有额外旋转 query $\tilde{q}^R_{t,i} \in \mathbb{R}^{1\times d_R}$，各头共享一份旋转 key $\tilde{k}^R_j \in \mathbb{R}^{1\times d_R}$。score 可写为

$$s_{t,i,j} = \frac{q^C_{t,i}(U_i^K)^\top c_j^\top + \tilde{q}^R_{t,i}(\tilde{k}^R_j)^\top}{\sqrt{d_h^C + d_R}} + M_{t,j}. \tag{4.16}$$

每层每 token 缓存 c_j 和 $\tilde{k}_j^R$，共 $d_c + d_R$ 个元素。若实现为追求吞吐而另存展开状态，或使用不同缓存精度与布局，眞实分配量须另算。MLA 与 GQA、KV 量化的区别在于：前者改变缓存的训练参数化，GQA 改头共享，量化改每元素表示；三者不能只按名称推断质量与速度。

4.8 把计算与状态放在同一张表中

表 4.1 比较一层、一条序列的有效持久状态；统一 K/V 每头维度为 d_h，未计编码元数据和页表。S 为本步可见历史长度，w 为固定窗口，r 为线性核特征维数。运算只计注意力核心，省略输入/输出投影、位置变换与特征映射；MLA 一行指式 (4.15) 的 latent decode 路径。

表 4.1 注意力表示对持久状态和单步核心计算的影响

结构	持久状态元素数	单 query 位置的主要工作量
MHA	每历史 token 为 $2h_q d_h$	$O(h_q S d_h)$
GQA / MQA	每历史 token 为 $2h_{kv} d_h$	$O(h_q S d_h)$，共享是否复用由 kernel 决定
固定滑窗	全窗口为 $2h_{kv} w d_h$	$O(h_q w d_h)$，须保证窗口外不再访问
线性核注意力	全历史状态为 $h_q r(d_h + 1)$	$O(h_q r d_h)$，另有特征映射成本
MLA	每历史 token 为 $d_c + d_R$	$O(h_q S(d_c + d_R))$，另有 latent 输出投影

全序列 prefill 中，MHA、GQA、MQA 的核心二次项均为 $O(h_q T^2 d_h)$；固定滑窗为 $O(h_q T w d_h)$；线性核状态递推为 $O(h_q T r d_h)$。MLA 的展开实现与 latent 实现具有不同运算形状，不能仅凭缓存公式给它指定与 MHA 相同或更低的计算量。对于一般稀疏图，还应直接计算总合法边数 $\sum_t |\mathcal{E}_t|$；它不一定具有固定窗口的界。

把表中元素数乘上层数、批量和每元素 bytes，才得到持久状态底数。工作区峰值、可用带宽、batch、序列长度以及实际算法，决定它怎样影响延迟。FlashAttention 对注意力分块、在线归一化并减少中间 IO；PagedAttention 用页表管理物理 KV 块。它们分别改变执行与内存布局，通常保持所选注意力的数学定义，不会自动改变表中的有效缓存元素数。第 12 章将展开这两种机制，不在此把它们当作新的注意力建模家族。

沿着改变的对象检查新方法

1. 将式 (4.4) 中的 m, n 同加 a，验证固定内容点积不变；再解释为何这不能证明整段因果模型平移不变。
2. 对 $h_q = 6, h_{kv} = 2$ 写出连续等组映射；给 K/V 每头放不同常数，检查广播后的头顺序。提示：映射为 $[0, 0, 0, 1, 1, 1]$。
3. 设 $d_c = 512, d_R = 64, h_q = 32, d_h = 128$，比较 MLA、8 头 GQA 与 MHA 的每 token 缓存元素数。提示：分别为 $576, 2048, 8192$；这些是指定配置的算术比较，没有证明质量相同。
4. 在线性核算例中令 $\phi(q) = [0, 1]$，分别计算归一化与未归一化结果；再增加一个零核权重的 key，检查它是否应改变输出。
5. 一个模型每四层含一层全局注意力。为什么不能把全模型 KV 一律按窗口 w 截断？提示：逐层追踪未来 query 的合法访问集合。

本章把位置坐标、可见图、头共享与历史表示分开。下面转向数值：即使结构和有效状态元素数固定，同一张量用什么编码、怎样乘加和如何校准，也会改变误差、内存与执行路径。

第二部分

数值表示、误差与持久状态

第 5 章　数值格式：存储、乘法与累积是三种精度

前面的矩阵公式在实数域中成立，计算机却只能保存有限个数。同一个 XW，可以把权重压成四位整数，把输入保留为 BF16，在矩阵乘法前反量化权重，并用 FP32 累积。若只说“这是一个四位模型”，读者仍然不知道乘法怎样执行、误差从哪里进入，以及节省了哪一段内存。本章先建立数值语言，再讨论量化算法。

5.1　先问张量在何时存在

模型中的张量具有不同的生命周期。权重在一次请求期间固定，适合离线统计和打包；激活随输入、位置和层而改变；KV Cache 虽然来自激活，却要跨多个生成步骤保留。梯度和优化器状态只在训练中出现，优化它们不能直接说明推理服务节省多少内存。

表 5.1　张量的生命周期决定低精度方案的约束

对象	生命周期	范围由什么决定	主要收益与约束
权重	模型加载后持续保存	已训练参数	压缩文件、常驻内存与读取流量；可离线重排
激活	本次前向的局部阶段	当前输入与模型状态	降低部分计算和临时存储；需在线量化
KV Cache	跨多个解码步骤	已见前缀与位置	降低长期状态与历史读取；需支持追加
梯度	反向传播阶段	数据与损失曲面	降低训练计算或通信；误差改变优化轨迹
优化器状态	整个训练过程	梯度的历史统计	降低训练状态内存；需稳定更新小量

因此，W4、A4、KV4 和梯度四位表示是不同问题。即使都使用四位编码，数据分布、校准机会、读写模式及误差的后果也不同。尤其要避免把训练时保存的激活与推理时长期保存的 KV 当成同一份账。

5.2　浮点数怎样分配有限的位

对二进制正常数，符号位、指数位和尾数字段共同表示

$$x = (-1)^{\sigma} 2^{e-\beta}(1 + m2^{-f}), \tag{5.1}$$

其中 $\sigma \in \{0, 1\}$，f 为显式尾数字段位数，m 是该字段的整数值，β 为指数偏置。前导的 1 没有存入字段，却贡献了一位有效精度。指数控制相邻数量级之间的跨度，尾数控制同一数量级内格点的密度。

在 $[1, 2)$ 内，FP16 相邻正常数间隔为 2^{-10}，BF16 的间隔为 2^{-7}。采用最近邻舍入时，1.003 的 FP16 近似值是 1.0029296875，而 BF16 近似值是 1。这不表示 BF16 全面较差：它具有更宽的指数范围。若计算产生约 70000 的中间值，FP16 已超出最大有限值，BF16 则仍有足够范围。

表 5.2 常见二进制格式的范围与舍入尺度；正常数、最近邻舍入

格式	指数位	尾数字段位	最大有限值	单位舍入误差 u
FP32	8	23	$\approx 3.403 \times 10^{38}$	2^{-24}
FP16	5	10	65504	2^{-11}
BF16	8	7	$\approx 3.390 \times 10^{38}$	2^{-8}
FP8 E4M3FN	4	3	448	2^{-4}
FP8 E5M2	5	2	57344	2^{-3}

表中的 u 是正常数附近的相对舍入误差界所用常数，不是所有数值的绝对误差。FP16 的最小正常数是 2^{-14}，最小正次正常数是 2^{-24}；BF16 对应为 2^{-126} 和 2^{-133}。次正常数使用没有隐含前导 1 的编码，接近零时相对误差模型不再适用。具体硬件还可能把次正常数直接置零，必须核查执行模式。

FP8 不是单一格式。E4M3FN 中的 FN 表示采用有限数及 NaN 的特定编码约定，不能套用 FP32 对全一指数位的解释；E5M2 则提供另一种范围和有效精度的折中。表中数值对应这些明确的格式，不涵盖所有厂商变体。FP8 格式设计及其训练应用可参见原始技术资料 [13]。这些格式用于说明数值表示机制。

5.3 Scale 并非整数独有

低精度浮点也常需要外部尺度。设 $\mathrm{cast}_{\mathrm{FP8}}$ 表示转换到指定 FP8 格式，可写成

$$\hat{x} = s\ \mathrm{cast}_{\mathrm{FP8}}(x/s), \qquad s > 0. \tag{5.2}$$

若一个张量的最大绝对值为 $a > 0$，用 $s = a/448$ 可以把它映到 E4M3FN 的最大有限范围；全零张量可取 $s = 1$，保证尺度为正。这只是最大值校准的例子，并不保证任务误差最小。尺度可按张量、行或块共享，其本身也有精度和元数据开销。若上一步极值用于下一步的尺度，还要处理分布随时间变化的问题。

整数表示的重构格点在给定尺度下等距，浮点表示的格点随指数分段改变。因而“FP8 范围比 INT8 大”不能脱离尺度、编码和所讨论的张量成立。INT8 的实数范围由尺度决定；FP8 的外部尺度也能平移其有效数量级。两者真正要比较的是：在相同内存、校准方式和运行路径下，哪些重要值被保留，哪些值被舍入或饱和。

5.4 沿一条矩阵乘路径检查三种精度

对 $X \in \mathbb{R}^{n \times d_{\mathrm{in}}}$、$W \in \mathbb{R}^{d_{\mathrm{in}} \times d_{\mathrm{out}}}$，考虑一个 W4A16 实现。磁盘上的权重被压成两个四位码共用一字节，旁边保存分组尺度；内核读入编码后，在寄存器或片上存储中重构为 FP16/BF16，再与十六位输入相乘；乘积汇入较高精度累加器，最后转换为输出精度。这里的“四位”直接描述存储，未必描述硬件乘法指令。

另一条路径把输入和权重都编码为 INT8，用整数乘加产生 INT32 结果，再乘尺度转换为浮点输出。两条路径需要的指令、反量化位置和误差模型不同。对一次点积，应明确记录

$$\underbrace{\text{编码与打包}}_{\text{存储}} \longrightarrow \underbrace{\text{乘法操作数}}_{\text{计算}} \longrightarrow \underbrace{\text{部分和及其合并}}_{\text{累积}} \longrightarrow \text{输出转换}. \tag{5.3}$$

“FP32 累积”也不等于每一个内部步骤都与逐项 FP32 参考求和一致。分块方式、部分和截断、融合指令和归约顺序都可能改变舍入。应以所用内核的数值约定和独立参考结果为依据。

有了这条路径，速度问题才能成立。W4 相对十六位权重的理想载荷缩小四倍，却还要读取尺度并执行解包；若计算受限、批量较大，或者反量化实现欠佳，延迟不必缩小四倍。若讨论的是训练，还需另外说明梯度、主权重副本与优化器状态的精度。精度标签只提供入口，不能代替执行图。

5.5 舍入、饱和与累积应分开分析

对于没有上溢、下溢的正常数，常用模型是

$$\mathrm{fl}(a \circ b) = (a \circ b)(1+\delta), \qquad |\delta| \le u, \tag{5.4}$$

其中 $\circ$ 为满足模型条件的一次基本运算。这个局部界并不能保证整个点积的相对误差很小。若大数相消，真实结果接近零，即使绝对误差不大，相对误差也可能很大。

一种常见求和界使用 $\gamma_n = nu/(1-nu)$（要求 $nu < 1$），把计算误差限制在与 $\gamma_n \sum_i |a_i|$ 同阶的绝对量内。它是给定归约模型下的最坏情况工具，不是观测到的平均误差，也不能直接用于解释所有 Tensor Core 的内部累积。其教学意义是：每次运算精度相同，项数与抵消程度仍会影响结果。

整数累积也会溢出。若两个操作数均取窄对称 INT8 区间 $[-127, 127]$，单个乘积最大为 16129。忽略偏置和零点修正时，n 项同号最大乘积需要满足

$$n \cdot 16129 \le 2^{31} - 1 \tag{5.5}$$

才能由有符号 INT32 容纳。实际矩阵乘通常远未达到这个极端，但零点修正、偏置及中间部分和也应计入核查，不能把“整数没有舍入”误写成“整数运算总是精确”。

对量化本身，可先在实数域裁剪到 $\tilde{x} \in [a, c]$，再舍入到该范围中的格点 $\hat{x}$，于是

$$\hat{x} - x = (\tilde{x} - x) + (\hat{x} - \tilde{x}). \tag{5.6}$$

第一项是裁剪误差，第二项是区间内舍入误差。减小量化步长通常有利于第二项，却可能增大第一项；两者可能相关，平方误差不能一般性地写成两个方差之和。此后，矩阵乘、非线性与残差再传播这些误差，这又是第三层问题。

用较高精度参考计算定位差异

若某层输出失真，先用同一份输入和重构权重计算 FP64 或可控 FP32 参考，比较量化前后的函数差异；再比较目标内核与该参考。前者主要诊断表示误差，后者主要诊断运算与实现误差。两项一起测量时，难以分辨尺度选错、维度广播错误和累积精度不足。

5.6 建立可以检验的数值记录

数值记录应包括编码格式、尺度的共享轴与类型、舍入和饱和规则，以及乘法、累积和输出精度。异常值需同时记录最大值、分位数及通道和位置分布，才能区分固定通道与偶发输入。

后续用 b_w, b_{kv} 表示每元素编码位数，bytes 表示字节数；低于八位需要打包，分组尺度另计。第 6 章统一量化公式，第 7 章再研究如何向敏感方向分配格点。

把精度标签展开成计算路径

为两个 W4A16 实现分别指定存储、解包、乘法、累积和输出类型，解释其输出与延迟为何可能不同。再说明 FP32 累积能否恢复四位编码已丢失的信息。

第 6 章　统一量化语言：格点、分组与校准

量化把连续或高精度数值映射到有限集合。若先把编码区间、尺度、共享轴和舍入规则写清楚，很多看似不同的方法便有了可以比较的共同基础。本章固定这些约定，也为后面的权重量化、激活量化和缓存量化建立独立参考实现。

6.1　编码与重构是一对映射

设整数编码属于 $[q_{\min}, q_{\max}]$，尺度 $s > 0$，整数零点为 z。本书使用

$$q(x) = \mathrm{clip}(\mathrm{round}(x/s) + z, q_{\min}, q_{\max}), \tag{6.1}$$

$$Q(x) = s\big(q(x) - z\big) \tag{6.2}$$

表示仿射量化及其重构值。$q(x)$ 是存储的整数，$Q(x)$ 是用于分析的实数重构，两者不可混用。除非特别说明，round 采用最近邻、恰好居中时取偶数的规则。若实现采用另一种 tie 规则，必须显式记录。

零点的含义是：实数零由整数 z 精确表示。这对带填充的卷积、稀疏结构以及需要精确保持零的张量很有价值。它不是要从浮点值中减去的平均值，也不等同于可学习的分布中心。

对观测范围，可先定义包含零的区间

$$a = \min(0, \min_i x_i), \qquad c = \max(0, \max_i x_i). \tag{6.3}$$

若 $c > a$，一种最小最大值校准为

$$s = \frac{c - a}{q_{\max} - q_{\min}}, \qquad z = \mathrm{clip}(\mathrm{round}(q_{\min} - a/s), q_{\min}, q_{\max}). \tag{6.4}$$

由于 z 必须为整数，实际重构端点 $s(q_{\min}-z)$、$s(q_{\max}-z)$ 可能与 a, c 略有偏移。把两端都恰好匹配且把零精确表示，一般不能同时做到。若整个组都是零，则不应除以零：可令 $s = 1$、$z = \mathrm{clip}(0, q_{\min}, q_{\max})$ 并保存相应零编码。非零常量组因已把零纳入范围，不会遇到同一个退化分支。

整数算术推理中的零点与重标定机制可参见 Jacob 等的工作 [14]。不同库可能使用不同舍入、饱和和偏置约定，核对公式比只核对函数名字更可靠。

6.2　对称区间有一个常被省略的选择

有符号 b 位整数的完整区间是 $[-2^{b-1}, 2^{b-1} - 1]$，负端比正端多一个编码。为了使正负重构严格对称，一些方案只用窄区间 $[-Q_b, Q_b]$，其中

$$Q_b = 2^{b-1} - 1, \qquad s = \frac{\max_i |x_i|}{Q_b}, \qquad z = 0. \tag{6.5}$$

此时有 $2^b - 1$ 个使用中的格点，一个编码闲置。另一些实现保留完整有符号区间，并接受不对称的负端点。两者都可能被叫作“对称 INT4”，却不是同一个量化器。本书的对称伪量化默认采用窄区间，且要求 $b \geq 2$；二值表示另行定义。

例如四个值 $[-0.9, -0.2, 0.4, 2]$ 采用三位窄对称量化时，$Q_b = 3$、$s = 2/3$，整数编码为 $[-1, 0, 1, 3]$，重构为

$$[-2/3, 0, 2/3, 2]. \tag{6.6}$$

这组手算数据的均方误差约为 0.04139。若把第一个数改为 -1，则 $x/s=-1.5$ 恰在两格中间，取偶舍入得到 -2。不写 tie 规则，参考答案便可能产生差异。

对偏斜分布，仿射编码能利用完整区间。两位无符号编码 $[0,3]$ 配合 $s=1,z=1$，可以精确重构 $[-1,0,1,2]$ 四个值；窄对称两位编码只有三个格点，无法同时表达这四个值。代价是后续点积需要处理非零的零点，且某些内核仅支持特定的零点约定。

6.3 尺度的共享轴决定矩阵乘怎样恢复

现在固定 $X\in\mathbb{R}^{n\times d_{\text{in}}}$、$W\in\mathbb{R}^{d_{\text{in}}\times d_{\text{out}}}$。所谓逐通道，必须进一步说明是输入通道还是输出通道。本书按输出通道量化权重时，每列 $W_{:,j}$ 有一个尺度；按输入轴分组时，每列被切成连续的 g 行，每组有独立尺度。实现库若保存 $W^{\top}$，同一设计在代码中的轴号会相反。

表 6.1 用形状替代含混的粒度名称

对象及共享方式	尺度形状示例	运算含义
整个权重矩阵	$[1,1]$	所有权重共享范围
权重逐输出通道	$[1,d_{\text{out}}]$	每个输出列恢复独立尺度
权重沿输入轴分组	$[\lceil d_{\text{in}}/g\rceil,d_{\text{out}}]$	点积需逐组求和并恢复各组尺度
激活逐行/逐 token	$[n,1]$	一行输入的全部特征共享尺度
激活逐输入通道	$[1,d_{\text{in}}]$	归约轴内存在不同尺度
KV 分块分通道	依头、token 块及特征组而定	必须说明 token 轴与特征轴如何分组

以对称量化为例，若 $X_{ij}\approx s_i^X q_{ij}^X$，$W_{jk}\approx s_k^W q_{jk}^W$，则

$$Y_{ik}\approx s_i^X s_k^W\sum_j q_{ij}^X q_{jk}^W. \tag{6.7}$$

尺度可以提到归约外面，一次整数点积后再恢复。若激活按输入通道缩放，则 s_j^X 留在求和内部；不能再用同一个未加权整数点积代替。分组量化介于两者之间：组内尺度相同，各组部分和分别重标定后合并。统计上更细的粒度可能减小重构误差，硬件上却需要更多元数据、乘法和归约。

若 $x_i\approx s_x(q_{x,i}-z_x)$、$w_i\approx s_w(q_{w,i}-z_w)$，且尺度与零点在这个点积中共享，展开得到

$$\sum_{i=1}^{m}x_iw_i\approx s_xs_w\Big(\sum_i q_{x,i}q_{w,i}-z_w\sum_i q_{x,i} \tag{6.8}$$

$$-z_x\sum_i q_{w,i}+mz_xz_w\Big). \tag{6.9}$$

权重编码的和可离线预计算，输入编码的和可能需要在线计算。若零点随组变化，修正也必须按组做。非对称表示是否划算，应同时比较质量收益和真实执行成本。

6.4 有效位宽与实际文件大小

对 N 个量化权重，若每 g 个权重共享一个 b_s 位尺度和 b_z 位零点，暂不计尾组与对齐，则

$$M_W\approx\frac{N}{8}\left(b_w+\frac{b_s+b_z}{g}\right),\qquad b_{\text{eff}}=b_w+\frac{b_s+b_z}{g}. \tag{6.10}$$

W4、组长 32、每组一个 FP16 尺度而没有零点时，$b_{\text{eff}} = 4.5$；组长 128 时为 4.125。若零点虽然只有四位逻辑信息，却用一个字节存储，账本应按眞实八位计算。文件头、张量对齐、未量化层以及为了内核重排而产生的填充也会增加实际大小。

当 d_{in} 不整除 g 时，可以让最后一组较短，或填充到整组。若采用填充，必须保证填入的是实数零对应的编码，而非盲目填整数零；输入 X 的归约轴也须同步补零，或由内核显式忽略超过原始 d_{in} 的项。输出维度仍是 d_{out}，不能靠裁剪输出修复输入轴不匹配。内存估算应包含填充，误差评价则只比较眞实元素。

6.5 校准是在选择承担哪一种误差

最小最大值校准保住极值，却可能让大量主体值只占少数格点。设有效裁剪范围为 $[-\alpha, \alpha]$，可定义量化器 Q_α 并搜索

$$\alpha^* = \arg\min_{\alpha>0} \frac{1}{m} \sum_i \big(x_i - Q_\alpha(x_i)\big)^2. \tag{6.11}$$

这是张量重构目标；若优化 $\left\|XW - X\widehat{W}\right\|_F^2$，则同样大小的权重误差会根据输入方向得到不同惩罚；若优化整个 block 的输出，还会纳入非线性、归一化和残差耦合。三个目标各有用途，不能把其中一个的最优解称作所有任务的最优量化器。

百分位裁剪对少量极值较不敏感，但其阈值依赖样本数量与分布。基于直方图的 KL 距离还依赖分箱和重分配规则。模型输入、普通隐藏激活和注意力概率不是同一种分布，不应把概率分布距离直接套给含负数的任意张量而不说明构造方式。

校准数据的作用也要分开。权重范围可直接从权重统计得到，不需要文本；输入敏感的重构和激活范围需要代表性输入；最终评测则应使用独立数据。若反复依据测试结果调组长和阈值，测试集实际上已经参与校准。数据来源、分词器、模板、序列长度和抽样种子应与量化结果一同保存。

静态量化离线确定激活尺度，在线省去部分统计，但未见输入可能超出范围。动态量化按当前行或块求尺度，能适应幅值变化，却要付出归约、同步、转换和写回成本。动态并不自动等于稳健：同一行中的一个极值仍可能决定整行步长，极低位宽的主体值仍可能归零。

6.6 均匀噪声模型何时有帮助

在没有饱和、输入相对于格点足够平滑、误差与输入近似不相关等条件下，可把舍入误差粗略建模为

$$\epsilon = Q(x) - x \sim \mathcal{U}(-s/2, s/2), \qquad \mathbb{E}[\epsilon] \approx 0, \quad \mathrm{Var}(\epsilon) \approx s^2/12. \tag{6.12}$$

这个模型解释了步长减半时局部误差方差约降为四分之一的直觉。但固定权重的量化误差不是每次前向重新抽样的白噪声；激活格点可能与输入相关，裁剪引入偏差，低位宽更容易违背高分辨率假设。跨层误差及长期复用的缓存误差尤其不能简单独立相加。

随机舍入则是眞正引入随机性：若 $x/s = k + r$、$r \in [0, 1)$，以概率 r 取 $k+1$，以概率 $1-r$ 取 k，在未饱和时重构期望等于原值。它牺牲单次最近邻误差来避免某些系统性偏差，并不保证每次推理更准确。可学习舍入又是另一类方案：用校准目标决定离散选择，应另计优化成本。

6.7 一个可审查的伪量化接口

下面用一维组展示参考逻辑。它保存整数列表与尺度，重构仍在浮点中执行；它不包含眞实低位打包或高效矩阵乘。

Listing 6.1 窄对称与仿射量化的独立参考逻辑

```
def quantize_group(values, bits, affine=False):
    assert bits >= 2 and len(values) > 0
    if affine:
        lo, hi = 0, 2**bits - 1
        a, c = min(0.0, min(values)), max(0.0, max(values))
        if c == a:
            scale, zero = 1.0, 0
        else:
            scale = (c - a) / (hi - lo)
            zero = max(lo, min(hi, round(lo - a / scale)))
    else:
        hi = 2**(bits - 1) - 1
        lo, zero = -hi, 0
        peak = max(abs(x) for x in values)
        scale = peak / hi if peak else 1.0
    codes = [max(lo, min(hi, round(x / scale) + zero))
             for x in values]
    restored = [scale * (q - zero) for q in codes]
    return codes, scale, zero, restored
```

Python 的 `round` 在精确 tie 时取偶；十进制输入转换为二进制浮点后，某个看似居中的值可能并非精确 tie，因此边界测试宜选用二进制可精确表示的数。分组接口应沿指定输入轴调用这个函数，而不是先展平矩阵后误把相邻输出列混入同一组。

本书实验用同一份权重比较逐张量与分组伪量化，保存重构误差，并检查零组、常量组、尾组、tie 和异常值边界。这样的实验能验证公式和数值行为；即使重构误差下降，也没有测到低位内核加速。真实部署还需验证编码打包、尺度布局、算子覆盖率与端到端执行。

从量化器走到矩阵乘

对 $W \in \mathbb{R}^{130\times16}$ 沿输入轴每 64 行分组，计算尺度数、尾组长度，以及按完整组填充时的编码元素数。再解释为何把组长改为 32，不能仅凭尺度变多就断言模型质量一定提高。提示：先固定校准目标与分组范围，分清局部重构和下游任务。

第 7 章　权重量化：从逐格舍入到输入感知的压缩

第 6 章给出了量化器的编码、重构与共享尺度。本章继续问一个更具体的问题：若只能保存有限位数，应该让哪些权重承担误差，能否用其他权重补偿它，又该怎样把压缩结果交给推理程序？讨论始终采用

$$X \in \mathbb{R}^{n \times d_{\text{in}}}, \qquad W \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}, \qquad Y = XW. \tag{7.1}$$

这里 n 是收集到的有效输入行数，可以来自多个序列和位置。$Q(W)$ 表示量化后的重构值，低位整数或码本索引才是实际保存的编码。两者的区分贯穿整章。

权重固定，适合离线统计、校准和打包。W4A16 等 weight-only 配置以较低位宽保存权重，激活仍使用较高精度；某些内核在加载后反量化，再进行浮点乘法。它可能降低权重容量和数据移动量，却不自动把所有乘法、累积及临时张量都变成四位。是否获得端到端加速，还取决于反量化、访存复用、批量大小和实际内核。

7.1　RTN：先建立一个可解释的基线

Round-to-nearest（RTN）在确定尺度、零点、分组和裁剪区间后，把每个值映射到最近的合法格点。对固定格点集合、可逐元素独立选择的平方重构误差，最近邻选择确实使各元素的误差最小；但这只是条件很明确的局部结论。改变共享尺度、引入输入相关性，或改用任务损失，都会改变优化问题。因此 RTN 是有用的基线，不能称为质量、误差或速度的普遍“下界”。

沿用式 (7.1)，令 $\Delta W = \widehat{W} - W$。固定输入上的输出重构误差为

$$\mathcal{L}_{\text{layer}}(\widehat{W}) = \|X\Delta W\|_F^2 = \operatorname{tr}\big(\Delta W^\top G \Delta W\big), \qquad G = X^\top X. \tag{7.2}$$

权重 MSE 相同，并不意味着式 (7.2) 相同。G 的对角元刻画各输入通道的能量，非对角元刻画它们在校准输入上的相关性；两者都会影响补偿方向。这里的 G 是未减均值的 Gram 矩阵，不应不加条件地称为输入协方差。

一个有说服力的 RTN 对照，应与候选方法使用相同模型、被量化层、编码区间、分组轴、组大小、尺度 dtype 和评测协议。若一种方法保留输出头或敏感层为高精度，就不能把这些额外位数隐藏在“同为 W4”的标签中。对 RTN 搜索裁剪阈值后，也应记录这项校准成本，而不是把它与完全未经校准的结果混为一谈。

7.2　GPTQ：从一个坐标的约束补偿推导更新式

GPTQ 从固定校准输入的层重构目标出发，利用二次结构安排量化后的补偿 [15]。先只看一个输出通道，其权重为 $w \in \mathbb{R}^{d_{\text{in}} \times 1}$。从原始权重出发，改变量 δ 对输出造成的平方误差恰为

$$\|X\delta\|_2^2 = \frac{1}{2}\delta^\top H \delta, \qquad H = 2X^\top X. \tag{7.3}$$

这是该层固定输入重构损失的精确 Hessian，既不是整个模型训练损失的精确 Hessian，也没有要求整个模型处在训练损失的驻点。若重构目标除以样本数或带 1/2，常数因子相应改变；一致缩放不会改变下面的补偿比值。

现在把第 i 个坐标固定为格点 q_i。记当前浮点值与格点之差为 $e_i = \widetilde{w}_i - q_i$，于是约束是 $\delta_i = -e_i$。设当前仍可调整的坐标集合为 $\mathcal{F}$，$i \in \mathcal{F}$，且 $H_{\mathcal{F}\mathcal{F}}$ 正定，记 $C = (H_{\mathcal{F}\mathcal{F}})^{-1}$。求解

$$\min_{\delta_{\mathcal{F}}} \frac{1}{2}\delta_{\mathcal{F}}^{\top} H_{\mathcal{F}\mathcal{F}}\delta_{\mathcal{F}}, \qquad \delta_i = -e_i \tag{7.4}$$

时，拉格朗日条件给出 $H_{\mathcal{F}\mathcal{F}}\delta_{\mathcal{F}} + \mu e^{(i)} = 0$，其中 $e^{(i)}$ 是第 i 个坐标基向量。因此

$$\delta_{\mathcal{F}} = -e_i \frac{C_{:,i}}{C_{ii}}, \qquad \Delta\mathcal{L}_{\text{local}} = \frac{e_i^2}{2C_{ii}}. \tag{7.5}$$

负号来自“将当前值改成格点”这一约束。C_{ji} 可以为负，所以补偿不一定使其他权重减小。

把所有输出通道放回 W，同一次输入坐标选择对应**量化本书权重矩阵的一整行**。令 $E_i = \widetilde{W}_{i,:} - Q(\widetilde{W}_{i,:})$，则对尚未锁定的行 j，更新为

$$\widetilde{W}_{j,:} \leftarrow \widetilde{W}_{j,:} - \frac{C_{ji}}{C_{ii}} E_i. \tag{7.6}$$

取 $j = i$ 时，该行正好变成量化重构值。GPTQ 原论文使用转置的权重存储约定，因而称这一过程为逐列量化；若在本书的 XW 约定中仍照抄 $W_{:,j}$，就会把输入轴与输出轴混淆。

锁定 i 后，还应更新剩余坐标的逆矩阵信息，而不是从头到尾使用一个不变的 C。精确算术中的消元式为

$$C_{\text{next}} = C_{\mathcal{F}',\mathcal{F}'} - \frac{C_{\mathcal{F}',i} C_{i,\mathcal{F}'}}{C_{ii}}, \qquad \mathcal{F}' = \mathcal{F} \setminus \{i\}. \tag{7.7}$$

无阻尼、精确算术下，每次补偿都让剩余自由坐标继续处于此前约束下的二次目标最优点；此时才可以用无一阶项的局部二次式继续推导。实际实现则使用数值稳定的代理。逐步补偿使用已经更新过的当前权重，所以它的舍入结果可能与直接对原始 W 做 RTN 不同。

7.2.1 两维手算：相关输入决定补偿的符号

下面只用固定的整数格点 $\{-1, 0, 1\}$，尺度为 1，不发生裁剪。取

$$X = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \quad w = \begin{bmatrix} 0.4 \\ 0.4 \end{bmatrix}, \quad Xw = \begin{bmatrix} 0.8 \\ 0.4 \end{bmatrix}. \tag{7.8}$$

此时

$$H = \begin{bmatrix} 2 & 2 \\ 2 & 4 \end{bmatrix}, \qquad C = H^{-1} = \begin{bmatrix} 1 & -0.5 \\ -0.5 & 0.5 \end{bmatrix}. \tag{7.9}$$

先把 $w_1 = 0.4$ 舍入为 0，故 $e_1 = 0.4$。若第二个权重保持 0.4，输出误差是 $[-0.4, 0]^{\top}$，平方和为 0.16。式 (7.5) 则给出

$$\delta_2 = -0.4\frac{-0.5}{1} = 0.2, \qquad \widetilde{w}_2 = 0.6. \tag{7.10}$$

补偿后的输出是 $[0.6, 0.6]^{\top}$，误差为 $[-0.2, 0.2]^{\top}$，平方和降为 0.08，也等于 $e_1^2/(2C_{11})$。两项误差不再是一大一小，而是共同承担无法消去的约束。

接着把 0.6 舍入成 1，最终权重为 $[0, 1]^{\top}$，输出平方误差为 0.40；直接 RTN 得到 $[0, 0]^{\top}$，误差为 0.80。但是另一个合法权重 $[1, 0]^{\top}$ 的误差只有 0.20。这说明局部补偿可以改进此例的 RTN，却不证明贪心量化找到了全局最优离散解。以上是手算教学数据，不是语言模型评测结果。

7.2.2 阻尼、分块与校准的条件

实际的 $X^\top X$ 经常奇异或病态：有效输入行数可能少于输入维度，通道之间可能高度相关，也可能有通道在校准集上始终接近零。通常改用

$$H_\lambda = 2X^\top X + \lambda I, \qquad \lambda > 0. \tag{7.11}$$

它在实数算术中正定，相当于在局部二次目标中增加 $\lambda \|\delta\|_2^2/2$，限制过大的补偿。λ 可按平均对角元的比例选择，但若全部对角元为零，仍需单独定义正的保护值。阻尼过小可能数值不稳，过大则弱化相关性信息；参数必须随校准记录保存。

GPTQ 为了使大矩阵上的操作可行，采用共同的量化顺序、分块延迟更新和 Cholesky 形式的逆 Hessian 信息。实现里的上三角因子及误差归一化方式与式 (7.6) 的裸逆矩阵写法不同，不能把两套分母随意拼接。教学实现可以按式 (7.5)–(7.7) 验证小矩阵；生产实现还要核对阻尼、排序、组边界与打包后的逆置换。

校准数据只规定了在哪些输入方向上保护输出。用很少数据估计出一个可求逆矩阵，并不意味着已经覆盖多语言、代码或长上下文的激活。逐层校准还应说明输入来自原始模型还是先前各层已量化的模型；这会改变后续层接收到的分布。

7.3 AWQ 与 SmoothQuant：同一个等式，不同的误差目标

令 $S = \mathrm{diag}(s_1, \dots, s_{d_{\mathrm{in}}})$，各 $s_j > 0$。对行向量约定，有

$$XW = (XS^{-1})(SW). \tag{7.12}$$

左边与右边在实数精确计算中相等；右边被量化后，格点与数据的相对位置改变，误差通常也改变。若只量化权重，实际近似是

$$\widehat{Y} = (XS^{-1})Q(SW), \qquad \widehat{W} = S^{-1}Q(SW), \tag{7.13}$$

而不是把 $Q(SW)$ 直接当成原坐标系中的权重。

AWQ 利用校准激活的统计信息识别重要的输入通道，并搜索有利于权重量化的缩放 [16]。在本书约定中，放大输入通道 j 对应放大权重的第 j 行，并把激活第 j 列除以 s_j。变换坐标中的重构误差 $\Delta W'_{j,:}$，恢复后为 $\Delta W'_{j,:}/s_j$；这解释了适当放大可能保护重要通道的机制。但组尺度也会随放大改变，所以不能据此推出“s_j 越大越好”。AWQ 的重要性依据不是简单挑出绝对值最大的权重，也不等于必须把这些权重另存成 FP16。

SmoothQuant 使用同一个等式，目标更侧重把激活的通道幅值差异迁移给离线权重，使 W/A 联合量化更容易 [17]。一种尺度形式为

$$a_j = \max_r |X_{rj}|, \quad u_j = \max_k |W_{jk}|, \qquad s_j = \frac{\max(a_j, \epsilon_s)^\alpha}{\max(u_j, \epsilon_s)^{1-\alpha}}, \quad 0 \le \alpha \le 1. \tag{7.14}$$

$\epsilon_s > 0$ 是退化通道的数值保护，具体归一化和裁剪仍要按实现记录。变换后 $X' = XS^{-1}$、$W' = SW$，分别量化时得到 $Q_A(X')Q_W(W')$；激活变平滑的同时，权重可能更难量化。AWQ 的 weight-only 目标与 SmoothQuant 的联合目标，不能因为使用相同代数恒等式就视为相同方法。

缩放的逆操作有时可融合到前一个线性层、归一化的仿射参数或现有逐元素算子中，避免额外内存读写。多分支共享激活、门控和残差连接会限制这种融合；应沿真实计算图证明每条分支的输出不变，不能只验证单个 XW 等式。若选择运行时显式缩放，则其开销属于部署成本。

OmniQuant 进一步在校准数据上优化可学习裁剪与等价变换参数，以局部重构目标适应量化[18]。这不等同于全量重训练，但也有优化步数、峰值内存与校准过拟合的成本。选择尺度、补偿权重和微调参数这三类过程，应分别写清楚哪些变量被学习。

7.4 旋转：改换坐标基，也必须守住计算图

对正交矩阵 $R \in \mathbb{R}^{d_{\text{in}} \times d_{\text{in}}}$，有

$$XW = (XR)(R^{\top}W), \qquad R^{\top}R = RR^{\top} = I. \tag{7.15}$$

正交变换保持行向量的二范数，却改变能量分布在哪些坐标上。以归一化 Hadamard 矩阵

$$R = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{7.16}$$

为例，$[a, 0]R = [a/\sqrt{2}, a/\sqrt{2}]$，峰值减小而总能量不变。这可以改善某些共享尺度量化器的分辨率。任意旋转并不必然降低峰值；旋转还可能使原来较均匀的数据更集中。QuaRot 和 SpinQuant 分别展示了结构化旋转与学习旋转在特定量化流程中的用途[19, 20]，其适用性依赖完整的变换布局与校准设置。

线性等价不意味着可以穿过所有算子

逐元素非线性通常不满足 $f(xR) = f(x)R$。例如取 $x = [1, -1]$ 和上述 R，则 $\mathrm{ReLU}(xR) = [0, \sqrt{2}]$，而 $\mathrm{ReLU}(x)R = [1/\sqrt{2}, 1/\sqrt{2}]$。SiLU、GELU 及 SwiGLU 的逐元素门控也不能任意交换旋转次序。将旋转从矩阵乘一侧挪到非线性的另一侧，可能已经改变模型函数。

RMSNorm 也需要区分归一化本身与可学习增益。设 $N(x) = x/\sqrt{\|x\|_2^2/d + \epsilon}$，则正交性给出 $N(xR) = N(x)R$。但带逐通道增益 $D_\gamma = \mathrm{diag}(\gamma)$ 的输出是 $N(x)D_\gamma$，一般有

$$N(xR)D_\gamma = N(x)RD_\gamma \neq N(x)D_\gamma R. \tag{7.17}$$

只有在矩阵可交换，或先把增益按正确方向吸收到相邻权重并重新推导整条分支时，才能得到所需等价性。

RoPE 含位置相关的旋转。若行向量形式为 xP_t，把额外的 R 移到 RoPE 前后需要核对 $P_tR = RP_t$，任意 R 不满足这个条件。另一种合法构造是在既有 RoPE 之后，对同一个头的 query 和 key 同时施加同一个正交矩阵：$(qR)(kR)^{\top} = qk^{\top}$。这只说明该位置的点积保持，不说明可以跨头混合、随意移动 RoPE，或不转换旧 KV 就改变缓存坐标。残差流上若改变坐标，还需让所有相加分支处在同一坐标系中。

结构化 Hadamard 变换可以使用快速算法，但仍有读写、调度和融合成本。能离线吸收的旋转与必须在线执行的旋转应分别统计；不能仅凭权重大小减少，就推断额外旋转不会影响延迟。

7.5 统一写法中不能丢掉逆变换

令 $\mathcal{T}$ 是作用在权重表示上的可逆映射，Q 是该坐标系内的量化重构。恢复到原始坐标后的近似应写成

$$\widehat{W} = \mathcal{T}^{-1}(Q(\mathcal{T}(W))) + \Delta, \tag{7.18}$$

其中 Δ 是原始坐标中的补偿。若补偿 Δ' 定义在变换坐标，应改写为 $\mathcal{T}^{-1}(Q(\mathcal{T}(W)) + \Delta')$。对一般映射，不能省略逆变换，也不能默认它与加法可交换。

对缩放，$\mathcal{T}(W) = SW$；对式 (7.15) 的旋转，$\mathcal{T}(W) = R^\top W$。执行程序可以把逆操作融合进激活侧，而不显式恢复完整浮点权重，但数学函数必须仍对应式 (7.18)。GPTQ 的补偿通常在离线量化过程中改写尚未锁定的权重，最终不必额外保存一个稠密 Δ；低秩和稀疏补偿则往往真的增加运行时分支。抽象表达相似，不代表存储及执行相同。

7.6 非均匀标量码本与向量码本

标量量化也可以使用非均匀码本 $\{c_1, \dots, c_K\}$，存储索引 $a(w)$，重构为 $\widehat{w} = c_{a(w)}$。NF4 是 QLoRA 所采用的一类四位非均匀表示，其码值设计与近似正态的权重分布有关 [21]。它不是均匀 INT4，也不能仅凭四位标签就与某种 FP4 格式互换。归一化后的分布、块尺度和反量化计算都会影响误差；一种分布上的码本设计不保证适合所有层。

向量量化（VQ）则把 d_v 个相邻权重作为一个向量 w_v，用 K 个码字之一表示：

$$a(w_v) = \arg\min_{k \in \{1,\dots,K\}} \|w_v - c_k\|_2^2, \qquad \widehat{w}_v = c_{a(w_v)}. \tag{7.19}$$

码本可学习元素间的联合结构，但选择分块方式、距离函数和码本容量也增加了优化问题。欧氏距离小，不保证 XW 重构误差小；输入感知距离与联合优化可以改变这一取舍。Additive Quantization 使用多个码字之和表示一个向量，QuIP# 等方法则结合坐标变换与结构化码本 [22, 23]。这些是具体的表示与算法设计，不能概括成“VQ 总比标量量化准确”。

假设共有 M 个向量、每个 d_v 个元素，使用一个共享码本，索引按固定长度打包，每个码字分量用 b_c bit 保存。若每向量还保存 b_s bit 尺度，则

$$b_{\text{eff}} = \frac{M\lceil \log_2 K \rceil + K d_v b_c + M b_s + b_{\text{meta}}}{M d_v}. \tag{7.20}$$

b_{meta} 包括尾组、对齐和其他元数据；若有多个独立码本，码本项也要相应增加。多个加性码本不仅增加索引数，还增加码字存储和解码求和。

例如 $d_v = 8, K = 256, M = 8192, b_c = 16$，不保存向量尺度且忽略其他元数据时，索引为 1 bit/weight，码本另占 0.5 bit/weight，合计 1.5。若每向量再存 FP16 尺度，则另加 2 bit/weight，变为 3.5。这是编码账本算例，不是某个已训练模型的压缩结果。宣称“每权重一位”却漏掉码本和尺度，便掩盖了最主要的开销之一。

码本方法的速度取决于查表、解码和矩阵乘的组织方式。先解压出完整浮点权重再调用普通 GEMM，可能只减少磁盘文件；在每次推理前都重复解压，还可能增加延迟。融合解码、缓存码字或直接在码本表示上计算，需要对应的内核与布局支持。

7.7 低秩与稀疏补偿：额外精度有额外账单

设原始坐标中的量化残差为 $E = W - \widehat{W}_0$。若选择

$$E \approx UV^\top, \qquad U \in \mathbb{R}^{d_{\text{in}} \times r}, \quad V \in \mathbb{R}^{d_{\text{out}} \times r}, \tag{7.21}$$

则推理变成

$$\widehat{Y} = X\widehat{W}_0 + (XU)V^\top. \tag{7.22}$$

若 U, V 的元素各用 b_c bit，补偿至少增加 $r(d_{\text{in}} + d_{\text{out}})b_c$ bit，并增加约 $2nr(d_{\text{in}} + d_{\text{out}})$ FLOPs。保存 XU 还需要临时激活。残差按 Frobenius 范数有较好的低秩近似，不等于在输入加权的式 (7.2) 下也最合适；秩的选择仍需实际验证。

若残差集中在少量位置，可另存一个有 m 个非零元素的稀疏矩阵 S_{sp}，计算 $X\widehat{W}_0 + XS_{\text{sp}}$。例如直接保存行、列坐标时，除了 mb_c bit 数值，还需要约 $m(\lceil \log_2 d_{\text{in}} \rceil + \lceil \log_2 d_{\text{out}} \rceil)$ bit 索引，实际整数类型和对齐可能更多。稀疏数值本身很少，并不意味着稀疏分支便宜；随机访问、索引读取和负载不均衡可能使小规模稀疏运算不如稠密算子高效。

低秩分支适合某些相关误差，稀疏分支适合某些局部大误差，二者都要检验而不是预设。若要比较“相同位宽”，应把这些分支、保留层与尺度都计入总文件和峰值内存。

7.8 二值、三值与两位：表示、训练与计算各问一次

二值权重常写为 $\hat{w} \in \{-\alpha, +\alpha\}$；三值权重为 $\{-\alpha, 0, +\alpha\}$；两位均匀编码通常有四个格点。它们不是同一种表示。二值符号在真正按位打包后可用一位保存，但组尺度仍有开销；每 g 个符号共享一个 b_s bit 尺度时，理想账本为 $1 + b_s/g$ bit/weight，尚未包含其他参数。

三种状态的信息容量为 $\log_2 3 \approx 1.585$ bit/符号，并不表示一个独立随机访问的普通整数天然占 1.585 位。逐元素固定编码通常用两位；把五个三值符号打包到一个八位字节，因为 $3^5 = 243 \le 256$，可得到 1.6 bit/符号，但要处理分组、解码与尾部。更长块编码可逼近理论值，随机访问与内核成本也会改变。论文标题中的“1.58 bit”应结合其训练、尺度、非量化参数和实际存储解释 [24]。

必须分别问三个问题。第一，模型是否从头在低位约束或伪量化下训练，还是对成熟浮点权重做 PTQ？原生训练成功不证明可以无损压缩任意现有模型。第二，激活、累积、归一化和输出头实际采用什么精度？极低位权重不代表整个网络只有一两位计算。第三，目标硬件有没有适配的打包和矩阵乘内核？三值点积在代数上可以转为带符号加法，但尺度恢复、激活读取和最终累积仍有成本，普通稠密硬件未必因此更快。

7.9 验证一个方法：先证明等价，再比较误差，最后测系统

在引入量化之前，先用同一组随机小矩阵验证缩放或旋转前后的输出在给定浮点容差内一致；若这里已不一致，应先检查逆变换、残差分支和算子位置。对 GPTQ，应分别测试满秩二维例子、重复输入列与零输入通道，确认补偿符号以及阻尼分支。随后才比较 RTN、补偿方法在独立输入上的 $\left\|XW - X\widehat{W}\right\|_F^2$ 和模型损失。

保存结果时，至少应能还原模型与 tokenizer 版本、被量化层、位宽与格点、分组轴及尾组规则、尺度和零点 dtype、校准样本与种子、优化或搜索成本。系统报告还要给出真实打包文件、加载后常驻内存、峰值内存，以及硬件、内核、批量、输入长度和输出长度。只用浮点伪量化比较重构误差，不能据此声称已经测得低位内核的速度。

把公式变成独立检验

1. 重算二维补偿例的五组输出，并枚举 $\{-1, 0, 1\}^2$，找出该例的全局最优离散解。提示：贪心最优和整体最优是两个问题。
2. 设 W 的一个输入通道在校准集上全为零。解释为什么无阻尼 Gram 矩阵可能不可逆，以及为什么这不证明真实请求永远不用该通道。
3. 取带不同增益的 $D_\gamma = \text{diag}(1, 2)$，用本章的二维 R 验证 $RD_\gamma \neq D_\gamma R$。再说明如何在一个确定的线性后继分支中吸收增益。
4. 给一个加性码本或低秩补偿配置写出总 bit 账本。若同一配置在小批 decode 与大批 prefill 中速度不同，应检查哪几项数据移动和计算？

第 8 章　激活量化与端到端误差：从一个数到一次生成

离线权重量化只需保存一份固定编码，激活却在每次请求、每层和每个位置重新产生。它既是后续矩阵乘的输入，也是先前所有近似误差的结果。因而 W/A 联合量化不能拆成两个彼此独立的压缩任务：缩放在两侧转移范围，激活统计随上下文改变，误差还会经过门控、注意力和自回归反馈。本章先建立误差的可计算分解，再说明如何判断一次量化是否保留了需要的行为。

8.1　先把权重、激活和累积误差放进同一等式

仍设 $Y = XW$，令 $\hat{X} = X + \Delta X$、$\hat{W} = W + \Delta W$。假定矩阵乘本身暂用实数精确计算，则

$$\hat{Y} - Y = \Delta X\, W + X \Delta W + \Delta X \Delta W. \tag{8.1}$$

第一项来自激活重构，第二项来自权重重构，第三项是两者的交互。只把“单独量化激活的误差”和“单独量化权重的误差”相加，会漏掉第三项；它们也可能相互抵消，不能假设误差能量一定直接相加。

例如标量 $x = w = 1$，令 $\Delta x = \Delta w = 0.1$，则输出误差为 0.21，而不是 0.20。若 $\Delta w = -0.1$，输出变为 0.99，误差只有 -0.01。这是代数算例，不能用后一种偶然抵消为量化器提供普遍保证。

若要给保守的范数估计，可用谱范数 $\|\cdot\|_2$ 与 Frobenius 范数写成

$$\begin{aligned} \left\|\hat{Y} - Y\right\|_F \le\ & \|\Delta X\|_F \|W\|_2 + \|X\|_2 \|\Delta W\|_F \\ & + \|\Delta X\|_F \|\Delta W\|_2 . \end{aligned} \tag{8.2}$$

这不是实际误差必然达到的值，只说明大的权重算子范数或输入能量会放大特定方向的扰动。真实实现还需加上矩阵乘/累积误差 E_{arith}，若偏置被量化，还要加 Δb。输入编码正确但累积溢出，与量化格点过粗，是不同故障。

同样地，某层的 ΔX 不一定只由该层的激活量化器产生；它可能已经包含此前的权重、激活及 KV 误差。比较单层方法时，应说明喂入的是原始模型激活还是量化前缀产生的激活。

8.2　激活分布为什么比权重更难固定

一层激活的范围会随输入内容、语言、token 位置、上下文长度和 prefill/decode 阶段变化。一些大值持续集中在特征通道，另一些只出现在少数 token 或请求。校准集上见到的最大值既可能漏掉罕见输入，也可能被一个偶然极值支配；它不能替代明确的服务分布假设。

取多数数值位于 $[-1, 1]$、另有一个值为 20 的教学张量。采用窄区间对称 INT8 时，$s = 20/127 \approx 0.1575$，主体区间宽度内约有 12.7 个格点间隔；改为窄区间对称 INT4，$s = 20/7 \approx 2.8571$，主体中的所有值都可能舍入成零。保留极值让大量小值失去分辨率；裁剪极值则交换为少数位置的大误差。选择哪一边，需要看这些位置对输出的贡献。

因此“异常值”（outlier）应配有可复查定义，例如相对通道分位数的倍数或超过某个阈值的比例。幅值大不等于无用噪声，幅值小也不等于不重要。诊断时除了最大值，还可记录各层/各通道的

分位数、饱和比例、重构后零值比例以及输入加权输出误差，并区分不同阶段和长度。对不同分布只报告一个全模型最大值，往往无法定位问题。

8.3 共享尺度既是统计选择，也是矩阵乘接口

把批量与位置展平成 n 行。若激活逐行共享尺度 s_r^X，权重逐输出列共享尺度 s_k^W，对称编码满足

$$\widehat{X}_{rj} = s_r^X q_{rj}^X, \quad \widehat{W}_{jk} = s_k^W q_{jk}^W, \quad \widehat{Y}_{rk} = s_r^X s_k^W \sum_j q_{rj}^X q_{jk}^W. \tag{8.3}$$

尺度在归约轴 j 上不变，故可以先做整数点积，再恢复一次尺度。这解释了逐 token 激活与逐输出通道权重为何是常见组合，而不是说它们对所有数据误差最小。

若激活每个输入特征 j 有不同尺度 s_j^X，它就留在 $\sum_j s_j^X q_{rj}^X \widehat{W}_{jk}$ 内，不能从整个点积提出。分组尺度可以在组内提取、组间再求和，但内核必须支持对应的分块与重标定。把逐特征的伪量化结果直接接到只接受逐行尺度的整数 GEMM，数值协议便不一致。

仿射量化还需处理零点。以一行激活和一列权重为例，长度为 d_{in} 的整数点积应恢复

$$\begin{aligned}\sum_j (q_j^X - z_X)(q_j^W - z_W) = &\sum_j q_j^X q_j^W - z_W \sum_j q_j^X \\ &- z_X \sum_j q_j^W + d_{\text{in}} z_X z_W.\end{aligned} \tag{8.4}$$

零点不为零却漏掉修正项，得到的不是同一个近似矩阵乘。测试应包含非零零点、负值、全零行和非整除尾组，不能只用恰好对称的随机张量。

动态逐行量化还要先归约范围、计算尺度、舍入和打包。小批 decode 的矩阵乘可能很短，这些步骤的启动与读写占比就可能明显增加。把量化与矩阵乘融合有机会减少中间写回，但收益必须用实际内核测量；浮点伪量化程序里的耗时不代表最终低位执行时间。

8.4 平滑、旋转、裁剪与高精度分支保护什么

第 7 章的等价缩放使激活变成 XS^{-1}、权重变成 SW。它可能缓解激活通道范围差异，同时增加权重量化难度。旋转改变能量在坐标上的分布，裁剪主动放弃部分极值。这些方法都在改变低位格点与数据分布的关系，必须在完整 W/A 输出上比较，而不能只挑某一侧变小的 MSE。

另一种方法是把输入通道分成普通集合 $\mathcal{J}_b$ 和高精度集合 $\mathcal{J}_o$，两者不相交且覆盖全部输入轴：

$$XW = X_{:,\mathcal{J}_b} W_{\mathcal{J}_b,:} + X_{:,\mathcal{J}_o} W_{\mathcal{J}_o,:}. \tag{8.5}$$

前一项可走低精度路径，后一项保留较高精度。LLM.int8() 的混合分解体现了这类思路 [25]。选择哪些通道、阈值是否动态、额外索引和权重是否重复存储，都是实现的一部分；高精度集合扩大时，低位路径的覆盖率与收益也会改变。

因此精度标签还应附上执行说明。W8A8 需要明确是整数还是特定 FP8 格式、如何缩放、乘法与累积使用什么 dtype；W4A8 或 W4A4 也不能只凭位数推断误差与速度。归一化、softmax、SiLU、门控和残差相加可能仍在 BF16/FP16/FP32 完成，随后再量化。对这些算子使用高精度是数值设计的一部分，不应隐藏在“全四位”之类概括中。

8.5 QAT：前向离散，反向使用什么梯度

量化感知训练（QAT）在训练前向中模拟目标编码与重构，让参数适应这些约束 [14]。对固定 s, z 的标量量化器

$$Q(w) = s[\mathrm{clip}(\mathrm{round}(w/s) + z, q_{\min}, q_{\max}) - z], \tag{8.6}$$

Q 是分段常数，除不连续点外，其对 w 的真实导数几乎处处为零。直接使用这个导数，通常无法给权重提供有用的更新信号。

直通估计（straight-through estimator，STE）在反向中人为指定代理导数。例如令 $a = s(q_{\min} - z)$、$c = s(q_{\max} - z)$，一种常见约定为

$$\frac{\partial Q(w)}{\partial w} \mathrel{\widehat{=}} \begin{cases} 1, & a \le w \le c, \\ 0, & \text{其他情况.} \end{cases} \tag{8.7}$$

符号 $\mathrel{\widehat{=}}$ 表示算法采用的近似反向规则，不是导数恒等式。也有实现不屏蔽裁剪区外的梯度，或对可学习尺度采用专门归一化；边界规则和尺度梯度必须分别给出。STE 不保证是某个平滑目标的无偏梯度，也不保证训练必然稳定。

Listing 8.1 带显式 STE 约定的训练结构示意；不是低位推理内核

```
# Parameters and optimizer states may remain floating point.
w_master = parameter()
for batch in training_batches:
    w_fake = quantize_dequantize_with_ste(w_master, config)
    y = forward_with_fake_quantized_weights(batch, w_fake)
    loss = objective(y, batch.targets)
    loss.backward()       # backward uses the declared surrogate
    optimizer.step()
    optimizer.zero_grad()
# Deployment still needs packing and a compatible inference path.
```

QAT 可以更新原始权重、少量适配参数、尺度或裁剪阈值，但每种方案的训练开销与表达能力不同。浮点主权重、梯度和优化器状态不因前向使用 W4 而自动缩成四位；反向仍需保存或重算激活。微调结束后还要导出整数/码本表示，并验证导出模型与训练时伪量化模型一致。仅得到一个带 fake quantizer 的训练检查点，还没有证明真实低位部署完成。

8.6 同样的 MSE，为何有不同的输出与生成

先看输入敏感度。令 $X = [10, 0]$、$W = [1, 1]^\top$，两种权重扰动为

$$\Delta W_a = [\varepsilon, 0]^\top, \qquad \Delta W_b = [0, \varepsilon]^\top. \tag{8.8}$$

它们的权重 MSE 都是 $\varepsilon^2/2$，输出误差却分别为 10ε 与 0。第二种方向在这个输入上没有被使用，并不意味着它在其他输入上也不重要。这是输入加权重构与纯权重重构不同的最小例子。

即便到达 logits，相同 MSE 仍不足以解释行为。取二分类 logits $z = [1, 0]$，扰动 $\Delta z_a = [0.6, 0.6]$ 与 $\Delta z_b = [-0.6, 0.6]$ 的 MSE 都是 0.36。第一种只加共同偏移，softmax 分布完全不变；第二种把 logits 变成 $[0.4, 0.6]$，最大项已从第一类变为第二类。输出误差的方向和候选间距，与误差大小同样重要。

这种现象可写成一个充分条件。设原 logits 有唯一最大项 i^*，间距为

$$m = z_{i^*} - \max_{j \neq i^*} z_j > 0. \tag{8.9}$$

若 $\|\Delta z\|_\infty \le \varepsilon$，则任意其他类别 j 都满足

$$\hat{z}_{i^*} - \hat{z}_j \ge z_{i^*} - z_j - 2\varepsilon \ge m - 2\varepsilon. \tag{8.10}$$

因此 $m > 2\varepsilon$ 足以保证这个固定前缀上这一步的贪心 token 不变。它不是必要条件；$m \le 2\varepsilon$ 只表示此界不能保证，不能直接判定一定翻转。边界相等还可能出现并列，需要记录 tie 规则。

上述条件也不保证随机采样结果相同：即使最大项不变，概率仍可能改变。自回归生成中，一旦某一步得到不同 token，后续输入前缀也不同，不能再把两条序列的差异都归结为同一输入上的局部量化误差。固定真实前缀的损失评测与自由生成评测，分别观察这条链的不同部分。

8.7 误差穿过多层时，局部校准能说明多少

对同一输入的浮点 block F_ℓ 和量化 block $\widehat{F}_\ell$，可记录局部重构误差

$$\eta_\ell(x) = \left\| \widehat{F}_\ell(x) - F_\ell(x) \right\|. \tag{8.11}$$

若在需要的输入区域内有 $\|F_\ell(u) - F_\ell(v)\| \le K_\ell \|u - v\|$，并以 $E_\ell = \hat{h}_\ell - h_\ell$ 表示先前累计误差，则可分解为

$$\|E_{\ell+1}\| \le \eta_\ell(\hat{h}_\ell) + K_\ell \|E_\ell\|. \tag{8.12}$$

局部重构使用原始 h_ℓ，而真实量化执行接收 $\hat{h}_\ell$；两者的分布差异也会改变 η_ℓ。K_ℓ 又依赖权重、注意力、归一化及输入区域，不能随意设为 1。残差提供了恒等分支，但并不使误差自动消失或保证每层都收缩。

这一分解解释了 block reconstruction 的用途与限度：把 norm、attention、FFN 和残差一同纳入目标，比孤立权重 MSE 更接近实际计算；然而有限校准输入上的重构小，仍不能保证所有长上下文或生成轨迹都稳定。单层扰动、逐层累积和完整模型三个层次的实验，应分别进行。

8.8 局部指标的定义和零分母必须一致

对参考张量 A 与重构 $\widehat{A}$，本书使用第 1 章的 MSE、相对误差与最大绝对误差。MSE 便于汇总每个元素的平均误差，最大误差可以暴露少数极端位置，相对误差衡量误差相对信号的比例。不同层形状不同，不能把未归一化平方和与按元素平均的 MSE 混为同一个指标。

信号量化噪声比（SQNR）通常定义为

$$\mathrm{SQNR}(A, \widehat{A}) = 10 \log_{10} \frac{S}{E}, \qquad S = \|A\|_F^2, \quad E = \left\| A - \widehat{A} \right\|_F^2. \tag{8.13}$$

当 $S, E > 0$ 时，单位为 dB，数值越高表示相对平方误差越小。若 $S > 0, E = 0$，结果是 $+\infty$；若 $S = 0, E > 0$，可记为 $-\infty$；若二者都为零，则 0/0 没有定义，应明确标为“零张量精确重构”，不能伪装成一个有限的优良 SQNR。

在分子分母偷偷加同一个小量会改变这些边界，并可能让小信号层的排序随常数变化。若工程接口必须输出有限数，应报告所用截断或保护规则，并同时保留原始 S, E。余弦相似度在零范数时也没有定义；即使非零，它忽略共同缩放，所以不能代替重构误差。

全张量 SQNR 可能由少数高能通道支配。比较量化配置时，还应观察不同层、头、token 位置的误差分布与输出误差，避免一个平均值掩盖特定子群的退化。

8.9 从损失与 PPL 到任务质量

在相同目标 *token*、相同真实前缀和同一有效位置集合上，设原始与量化模型的平均负对数似然分别为 $\bar{\ell}$、$\hat{\bar{\ell}}$，则

$$\frac{\mathrm{PPL}_{\mathrm{quant}}}{\mathrm{PPL}_{\mathrm{ref}}} = \exp(\hat{\bar{\ell}} - \bar{\ell}). \tag{8.14}$$

损失增大、PPL **升高**才是语言建模变坏；PPL 降低表示该协议下平均对数概率改善。公式算例：若损失增加 0.02 nat，则 PPL 比值为 $e^{0.02} \approx 1.0202$，约增加 2.02%。

局部 logit 界也能连接损失。若一个位置所有 logit 的扰动绝对值至多为 ε，则 log-sum-exp 的变化至多为 ε，真实类别 logit 的变化也至多为 ε，因此该位置交叉熵变化的绝对值至多为 2ε。这个保守界说明误差能够影响概率，却不说明各种方向的扰动具有相同后果。

PPL 比较必须固定 tokenizer、文档拼接/边界、上下文窗口与滑动步幅、聊天模板、有效标签和损失平均方式。更换分词器改变 token 数，按句平均再平均又可能不同于按全部有效 token 加权平均。样本的来源与污染检查同样要记录；用校准集反复选超参数后，再把它当作最终测试集，会高估泛化。

平均 PPL 只观察真实序列上的预测概率，不能替代数学、代码、多语言、指令遵循和长上下文任务。量化模型可能保持大部分普通 token 的概率，却改变少数小间距的关键决策。应按使用场景加入自由生成或可执行判分，并记录采样温度、top-p/top-k、停止条件、生成长度和随机种子。随机生成报告样本数与不确定性，不能只挑一条看起来相似的回答。

8.10 把诊断组织成可复查的比较

发现 A4/A8 或 W/A 联合配置退化时，可以按因果顺序定位。先在同一输入上比较浮点参考与伪量化输出，检查尺度、零点、组边界和重构 dtype；再分别打开权重、激活以及缓存量化，观察哪一条误差路径首先出现变化；随后比较原始输入和已量化前缀输入下的 block 重构。若伪量化正常而部署模型异常，重点检查打包、整数点积修正、累积范围及融合顺序，而不是立即重新搜索校准超参数。

对质量与速度的最终比较，至少需要三个互相衔接的记录：

1. **参考协议**。相同模型检查点、tokenizer 与模板版本，相同有效输入、loss mask、生成规则和测试集；校准与最终评测分离，记录分布及规模。
2. **表示协议**。W/A/KV 各自的位宽、格式、分组轴与大小、尺度和零点类型、非量化层、异常值分支及补偿参数；同时报告有效位宽、真实文件和加载后内存。
3. **执行协议**。硬件、软件与内核版本、批量、输入/输出长度、并发和缓存状态；计时包含哪些量化、解码与数据传输步骤，预热和同步怎样处理。区分单算子微基准、单请求延迟和服务吞吐。

三者中的一项改变，结论就要限定到新的设置。同一 nominal bit 下不同组大小、额外高精度分支或不同内核成熟度，都可能改变排序。量化时间、峰值校准内存和额外训练成本也属于方法代价，不能只保留部署后的有利数字。

质量、常驻内存、延迟和吞吐构成多目标取舍。一个配置若在给定协议下质量不低且内存、延迟均不差，并至少一项更好，才可说在这些指标上支配另一个配置；多数候选只是在不同指标间取舍。画出这种 Pareto 关系比宣称一个脱离负载的“最佳量化方法”更有解释力。

误差评价的四个边界检验

1. 对式 (8.1) 分别令 $\Delta X = 0$、$\Delta W = 0$ 和两者均非零，验证每种消融眞正保留了哪些项。不要把独立测得的误差范数当作可以直接相加的随机变量。
2. 对 $z = [1, 0]$ 构造一个超过 0.5 最大绝对误差却不改变贪心 token 的扰动。说明式 (8.10) 为何只是充分条件。
3. 为 SQNR 编写三类边界输入：非零张量精确重构、零参考但非零重构、双方全零。提示：保存信号能量与误差能量，比把 NaN 一律替换为零更可解释。
4. 某配置 PPL 从 10 变为 10.2，但眞实低位内核未运行。能支持怎样的质量表述？还缺哪些证据才能宣称内存下降或端到端加速？

激活量化使数值近似进入模型的动态计算链。局部指标帮助定位，输入加权与 block 重构帮助校准，固定前缀损失和自由生成检验不同的行为；眞实编码与执行测量最后决定压缩是否带来系统价值。下一章将同样的分析用于跨 decode 步持久存在的 K/V 状态。

第 9 章　KV Cache：可复用状态、量化误差与长上下文

自回归模型每一步都要读取历史，但不必每一步重新计算历史。本章先说明缓存等价性依赖哪些条件，再从误差、布局与长度三个角度分析压缩。第 11 章给出完整内存账本；这里重点讨论缓存表示什么，以及压缩后改变什么。

9.1　缓存递推与全量前向何时等价

固定某一层，采用第 3 章的行向量约定。当前输入块为 $U \in \mathbb{R}^{B\times T\times d}$，投影并重排后

$$Q \in \mathbb{R}^{B\times h_q\times T\times d_h}, \qquad K, V \in \mathbb{R}^{B\times h_{kv}\times T\times d_h}. \tag{9.1}$$

若先前已保存 S_0 个位置，当前块的全局位置为 $S_0, \dots, S_0 + T - 1$。对标准 RoPE，按这些位置旋转 Q/K，通常不旋转 V；记旋转后新 key 为 K_{new}，则

$$K_{\text{cache}}^{\text{next}} = \text{Concat}_{\text{token}}(K_{\text{cache}}, K_{\text{new}}), \tag{9.2}$$

$$V_{\text{cache}}^{\text{next}} = \text{Concat}_{\text{token}}(V_{\text{cache}}, V_{\text{new}}). \tag{9.3}$$

更新后两个缓存的逻辑形状均为 $[B, h_{kv}, S_0 + T, d_h]$。GQA 中第 h 个 query head 读取 $\lfloor h/(h_q/h_{kv})\rfloor$ 对应的 KV head；缓存本身无需真的复制到 h_q 个头。块内第 i 个 query 只能读取全局位置 $j \le S_0 + i$ 的 key，而不是只对当前 $T \times T$ 小块套上三角掩码。

缓存等价性来自因果性：在权重、位置映射和掩码固定的推理模式下，增加未来 token 不改变历史位置已经算出的逐层表示。因此历史 K/V 可复用，旧 query 通常不再需要。整段前向、分块 prefill 和逐 token decode 应在指定浮点容差下得到相同结果。第 16 章用独立标量参考检查这一点。

缓存的物理槽号不等于位置号。滑窗淘汰前缀后，不能把新 token 的 RoPE 位置重置为窗口下标；分页与环形缓冲也要保留逻辑位置映射。若动态位置缩放会随最终长度改变旧位置的旋转规则，旧的已旋转 K 就不再自动有效，需要相应重编码或采用明确的固定策略。改写前缀、切换模型权重或聊天模板、训练模式下重新抽取 dropout，均不能直接沿用旧缓存。跨请求共享还要求实际前缀 token、模型版本与缓存表示一致。

9.2　“没有缓存”的复杂度取决于重算策略

设提示长度为 P，随后产生 G 个输出 token。为便于数量级比较，令第 g 次预测可见长度为 $s_g = P + g$，$g = 0, \dots, G - 1$。记每 token 的全部投影与 FFN 工作为 C_{proj}，完整稠密注意力工作按 $\mathcal{O}(Ls_g^2 d)$ 计。若每一步从输入 ID 重跑整个前缀，则

$$C_{\text{no cache,proj}} = \mathcal{O}\left(C_{\text{proj}} \sum_g s_g\right) = \mathcal{O}\big(C_{\text{proj}}(GP + G^2)\big), \tag{9.4}$$

$$C_{\text{no cache,attn}} = \mathcal{O}\left(Ld \sum_g s_g^2\right) = \mathcal{O}\big(Ld(GP^2 + PG^2 + G^3)\big). \tag{9.5}$$

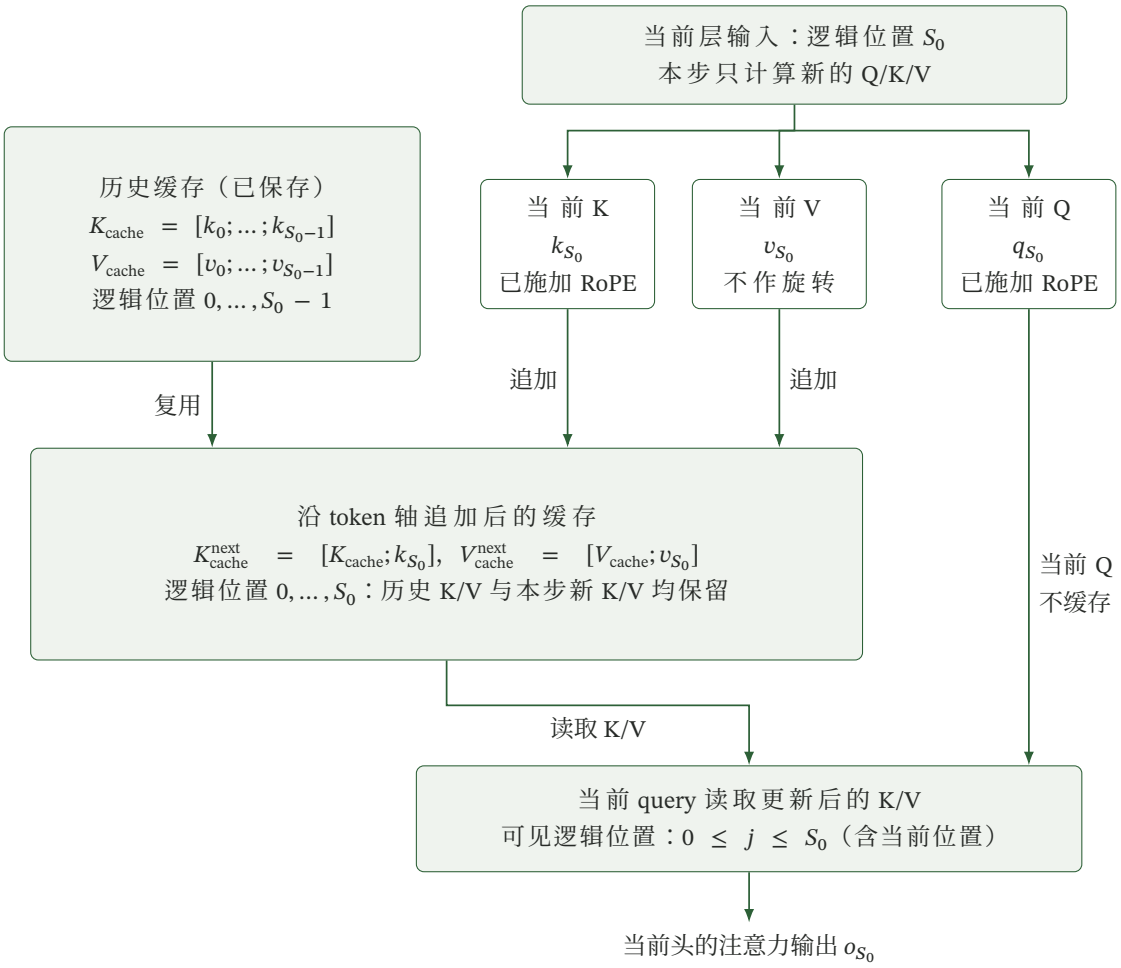


图 9.1 单 token decode 的缓存递推，取 $T = 1$、$S_0 \ge 1$，固定一层、一个请求及已匹配的 query/KV 头。q_j, k_j, v_j 是逻辑位置 j 的行向量投影，其中 Q/K 已按该位置施加 RoPE；分号表示沿 token 行拼接，o_{S_0} 为当前头输出。历史 Q 无需保存，未来位置尚未进入缓存；位置号不等于物理槽号。根据归档原稿自绘图重排，未使用第三方插图。

因果三角、只输出末位置 logits 等可改变常数与部分项，不能把这种重跑策略的所有计算统称为“随生成长度二次增长”。

使用 KV Cache 时，首次 prefill 的投影与注意力分别为 $\mathcal{O}(PC_{\text{proj}})$ 与 $\mathcal{O}(LP^2d)$；后续每步只对新 token 做投影/FFN，而新 query 仍读取历史。因此总量约为

$$C_{\text{cache}} = \mathcal{O}\big((P+G)C_{\text{proj}} + Ld(P^2+PG+G^2)\big). \tag{9.6}$$

缓存消除了历史表示的重复计算，未消除随历史增长的注意力读取。固定提示后，累计 decode 注意力可呈生成长度的二次增长，单步则随当前长度近似线性增长。

还有其他时间 – 空间折中：若逐层保存历史归一化隐状态，每步可从它们重建 K/V，再只算新 query。这样省去部分 K/V 常驻表示，却引入反复投影和另一种历史状态存储；对于 GQA，隐藏宽度 d 还可能大于单个 K/V 的宽度。它不等于从输入 ID 重跑，也不等于无状态推理。比较方法时必须写明保留了什么、重算了什么。

9.3 容量、带宽与物理布局是三个账本

标准 GQA 中，K/V 同维、同精度且 batch 等长时，缓存的有效元素数为 $2LBSh_{kv}d_h$。以 b_{kv} 表示每元素 *bit* 数，其理想字节容量为

$$M_{KV} = 2LBSh_{kv}d_h\frac{b_{kv}}{8}. \tag{9.7}$$

不等长请求把 BS 换成 $\sum_i S_i$；分支共享对实际物理前缀去重；beam 的分支数也不能在已经计入 B 后再乘一次。MLA、滑窗或 recurrent 层须根据实际状态重新计数，不能直接套用此式。

对每层一次 decode，若每个有效 K/V 从高带宽显存各读一次，理想读取底数与该层的缓存字节量同阶。真实流量取决于 GQA 共享是否在 kernel 内复用、分块是否导致重读，以及 scale、页表和解包访问。模型可能同时受到权重读取、KV、计算、通信或调度约束；“decode 一定受 KV 带宽限制”并不成立。

缓存常用 $[B, h_{kv}, S, d_h]$ 或分块布局。PagedAttention 通过页表把逻辑 token 块映射到物理块，减少预留浪费并支持共享 [26]；它不改变每个有效元素的数值表示。量化则减少元素的编码位数，可能引入额外元数据与临时缓冲。把 INT4 缓存每步全部反量化成完整 BF16 张量，虽可能减小持久状态，却会增加临时峰值与延迟。容量、运行流量和峰值占用应分别测量。

9.4 K 与 V 的误差沿不同路径进入输出

对一个 *query head* 的一个 *query* 行写清形状。令 $q \in \mathbb{R}^{1\times d_h}$，可见 key/value 为 $K \in \mathbb{R}^{S\times d_h}$、$V \in \mathbb{R}^{S\times d_v}$，

$$s = qK^{\top}/\sqrt{d_h}, \quad p = \text{softmax}(s) \in \mathbb{R}^{1\times S}, \quad o = pV. \tag{9.8}$$

只在可见位置上讨论 softmax；被掩掉的位置不参与此行的归一化。设 $\widehat{K} = K + \Delta K$、$\widehat{V} = V + \Delta V$，并固定 q，则

$$\Delta s = q\Delta K^{\top}/\sqrt{d_h}, \qquad J(p) = \text{diag}(p) - p^{\top}p \in \mathbb{R}^{S\times S}. \tag{9.9}$$

因 p 是行向量，一阶扰动必须按下列顺序相乘：

$$\Delta o \approx \Delta s\, J(p)V + p\Delta V. \tag{9.10}$$

高阶项包含 softmax 曲率以及概率扰动与 ΔV 的乘积。对多行 Q，应逐行使用不同的 Jacobian，不能把一个 J 不加说明地乘整张 score 矩阵。

K 误差通过 logits 改变概率分配，V 误差直接改变取回的内容，但这并不证明 K 总是更敏感。例如所有 value 行都等于同一向量时，任何归一化概率的重新分配都不改变输出，因而这条 K 误差路径为零；反之，被赋予较大权重的 value 若量化失真，可以直接显著改变输出。若所有可见 logits 都加同一个常数，概率也完全不变。实际敏感度取决于 q 的方向、概率分布、value 行之间的差异和下游计算。自回归继续生成时 q 本身也可能已受前面误差影响，式 (9.10) 只是局部诊断，不是完整轨迹误差界。

9.5 KIVI：不同轴、在线追加与高精度尾部

KIVI 在其测试模型上观察到部分 key 特征通道具有持续的幅值异常，并采用 K 按通道、V 按 token 的分组量化，同时保留高精度近期缓存 [27]。这是一种有具体模型与任务范围的设计证据，不是所有架构的统计定律。

把单层单头缓存看作 $[S, d_h]$ 矩阵：K 的“按通道”指固定特征列，在连续若干 *token* 行上统计范围；V 的“按 token”指固定 token 行，在连续若干特征列上统计范围。因此二者的分组轴不同，不能只记录一个含糊的 group size。K 的跨 token 组需要等待新 token 到齐；V 的逐 token 组可在该 token 到达时形成。

KIVI 的在线结构也不应简化成“K 和 V 始终各有恰好 R 个高精度 token”。论文算法令 K 的高精度尾部积累到阈值后成块量化并清空，阈值需与 token 分组兼容；V 则用队列保留近期 token，把移出的旧 token 按特征分组量化。二者实际尾部长度可不同，边界行为还要以代码版本为准。近期高精度区既用于在线成组，也可能改善某些任务的质量；不能仅凭“近期更重要”就推广到所有输入。

元数据格式同样需要核实。第 6 章用整数 zero-point；某些实现保存浮点最小值/偏置，重构为 $\hat{x} = sq + a$。二者都可以表达仿射格点，但保存的参数类型与字节数未必相同。不能因为方法名含“asymmetric”，就假设 zero-point 只占几个逻辑 bit。

设实际量化的 K/V 元素数为 n_K^q, n_V^q，高精度尾部元素数为 n_K^r, n_V^r，编码位数分别为 b_K, b_V, b_{fp}，则更可靠的账本是

$$M_{KV} = \frac{n_K^q b_K + n_V^q b_V + (n_K^r + n_V^r) b_{\text{fp}}}{8} \tag{9.11}$$

$$+ M_{\text{scale/offset}} + M_{\text{padding}} + M_{\text{page}} + M_{\text{aux}}. \tag{9.12}$$

其中辅助对象可含对齐、块索引与实现实际保留的副本。对于长度相同、K/V 配置完全相同的理想化近似，若每 g 个低精度元素存一个 b_s bit scale 和一个 b_z bit offset，可写

$$b_{\text{eff}} = b_q + (b_s + b_z)/g, \qquad \bar{b} \approx (1 - r/S) b_{\text{eff}} + (r/S) b_{\text{fp}}. \tag{9.13}$$

这不是 KIVI 的通用精确大小公式。短序列、尾组和不同 K/V 轴都会使简单压缩比失真；报告 $16/2 = 8$ 倍之前，必须说明这个数仅比较裸编码，还是实际分配量。

在线无校准方案按当前请求统计范围，需付出归约与编码成本；离线校准可固定部分变换或阈值，却要检查领域与长度偏移。二者也可混合，但校准目标必须明示：缓存自身重构、固定前缀下的 block 输出与最终任务质量并非同一目标。混合位宽可按层、头、K/V 或 token 区段分配，局部敏感度仍可能随输入与生成轨迹变化，不能假定各对象误差独立可加。

非均匀标量码本或向量量化还能改变格点集合。向量码本可能利用特征相关性，但索引、码本与高吞吐解码都需要计入；按相同有效位宽比较缓存 MSE 后，仍须验证注意力输出和任务质量。若真实 kernel 先把全部向量展开成高精度历史，编码更短也不保证带宽或延迟获益。

9.6 量化、淘汰与长上下文质量

量化保留位置但降低数值精度；淘汰直接删除部分历史；摘要或 token 合并改变信息表示；latent cache 改变每位置的状态维度。它们可组合，误差也会耦合。一个已删除的证据片段无法靠提高余下缓存位宽恢复，而重要度排序本身也可能被量化扰动。

StreamingLLM 保留初始 attention sink token 与最近窗口，目标是在受限缓存下维持流式运行 [28]。它不承诺窗口外所有语义证据仍可检索。H2O 则利用历史注意力估计重要 token 并与近期信息结合 [29]；过去被关注不保证未来会需要，因此评测应包含延迟出现的检索需求。它们改变可访问集合，与仅降低元素精度有本质区别。

“支持 128K”至少要分清：接口是否接受、位置机制是否有合适定义、系统容量与延迟是否可承受、模型是否在相关任务上真正利用远距证据。RoPE 在数值上可算不等于模型在训练长度外可靠泛化；低精度 KV 能装下更多 token 也不等于增加有效知识。单次 needle retrieval 还不能代表多证据组合、跨文档推理、代码依赖与长对话约束保持。

实际比较应固定模型/tokenizer/位置策略，分别扫描提示长度 P、生成长度 G 和并发 B，同时报告质量、峰值内存、TTFT 与逐步延迟，并记录量化、追加、解码和融合注意力的执行路径。局部 KV MSE、固定前缀输出误差、自由生成质量回答不同问题；先做固定前缀对照，再用多样任务与多条轨迹验证长度边界。第 15 章给出更完整的记录方式。

从缓存元素到实际账本

取 $L = 80, h_{kv} = 8, d_h = 128, B = 1, S = 131072$，K/V 均为 BF16。先按式 (9.7) 计算理想容量；再假设所有元素都用 4 bit 编码、每 64 个值只存一个 FP16 scale，计算理想有效位宽。最后说明若保留高精度尾部、增加 offset 或按页填充，应在哪些项加回开销。提示：前两问为 40 GiB 与 4.25 bit，对应 10.625 GiB；这些数不是某个真实 kernel 的实测。

缓存等价性依赖逻辑位置与模型计算保持一致；缓存压缩则同时改变数值误差和物理存储。下一部分将把这些持久状态放进完整模型账本，并解释为何省容量、少数据移动和降低端到端延迟必须分别验证。

第三部分

从训练目标到推理行为

第 10 章　从训练目标到模型行为

前几章把一个已经给定参数的语言模型拆成了张量运算。本章补上参数的来历：训练如何改变下一个 token 的条件分布，预训练、监督微调与偏好优化又分别改变什么。重点是目标函数与推理行为之间的联系。数据配方、分布式训练工程与完整的强化学习算法可分别转向《现代小模型训练》和《大语言模型后训练》，不需要先掌握那些工程细节才能读懂本章。

10.1　自回归损失是逐位置的概率账本

设一条训练序列为 $x_{1:T}$。在 teacher forcing 下，第 t 个位置的输入只包含 $x_{\le t}$，目标为 x_{t+1}；所有位置可在一次带因果掩码的前向中计算。若有效预测位置集合为 $\mathcal{I}$，每 token 平均负对数似然为

$$\mathcal{L}_{\mathrm{NLL}}(\theta) = -\frac{1}{|\mathcal{I}|}\sum_{t\in\mathcal{I}} \log p_\theta(x_{t+1} \mid x_{\le t}). \tag{10.1}$$

“预测下一个 token”不表示训练时必须逐 token 串行运行。串行依赖发生在需要把模型新生成的 token 反馈为下一步输入时；训练目标已经给出全部正确前缀，因果掩码保证并行计算没有看见未来。

对单个位置，令 logits 为 $z \in \mathbb{R}^V$，正确 token 的 one-hot 向量为 y。由 $p_i = \exp z_i / \sum_j \exp z_j$ 得

$$\ell = -\sum_i y_i \log p_i, \qquad \frac{\partial \ell}{\partial z_i} = p_i - y_i. \tag{10.2}$$

若正确 token 的预测概率为 0.2，该坐标的梯度为 -0.8；其余 token 的梯度为各自的正概率。这个局部梯度经输出头、残差流和各层雅可比反向传播，最终更新所有参与的参数。损失下降并非给权重写入一条显式知识记录，而是重新配置大量上下文中的概率。

交叉熵还给出一个重要限制。令真实数据条件分布为 p_{data}，在理想的总体期望下，交叉熵等于该分布的熵与 $D_{\mathrm{KL}}(p_{\mathrm{data}} \| p_\theta)$ 之和。因此它奖励对训练分布的拟合。训练语料中的过时知识、互相矛盾的文本和错误推理，也可能被预测得很准确。较低损失支持“在这些数据上压缩得更好”，不自动支持“事实更正确”或“会可靠调用工具”。

10.2　数据分布决定优化在强调什么

设训练数据由 K 个来源组成，采样权重为 w_k，$\sum_k w_k = 1$，则总体目标近似为 $\sum_k w_k \mathcal{L}_k$。增大代码来源权重会改变梯度的期望方向；它不是在固定模型上附加一个独立的“代码模块”。若某个小语料反复采样，名义训练 token 数 D 增加了，但独立信息量没有按相同比例增加。报告数据量时，原始 token、去重后的 token 与训练过程中实际见到的 token 应分别记录。

去重、质量过滤和时间切分解决不同问题。精确去重减少逐字重复；近似去重处理格式不同但内容相近的材料；污染检查寻找评测题、答案或近似改写是否出现在训练及校准数据中。一个过滤器可以同时去掉噪声和少数语言、特殊领域或难例，所以不能仅用“过滤后更干净”解释模型收益。更有信息的证据是固定计算预算后的分领域损失、任务结果和保留率。

Tokenizer 也参与这个账本。同一段文本在不同词表下可能有不同的 token 数；每 token 损失和训练 token 预算随之变化。因此，跨 tokenizer 的 perplexity 不能直接按数值高低比较。若要研究文本压缩能力，可补充每字节负对数似然，并固定字符规范化、编码与文本来源。它仍不能取代任务评测，却避免把分词粒度误认为语言能力。

把多条独立文档装入一条训练序列时，还要明确边界处理。若仅插入结束符而不重置注意力，后一文档可见前一文档；若使用分块掩码，则不同文档互不可见。两种目标不同。Padding 位置应从损失中移除，但不能因此把文档结束符一并去掉：结束符本身通常是需要学习的预测目标。

10.3 优化器状态、训练精度与计算预算

Adam 的一阶与二阶矩更新为

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t, \qquad v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2, \tag{10.3}$$

$$\hat{m}_t = m_t/(1-\beta_1^t), \qquad \hat{v}_t = v_t/(1-\beta_2^t). \tag{10.4}$$

AdamW 的一个常见更新约定是

$$\theta_{t+1} = (1-\eta_t\lambda)\theta_t - \eta_t \frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\epsilon}. \tag{10.5}$$

衰减项与自适应梯度项分离。Norm 参数与 bias 是否衰减、矩状态采用何种精度、梯度如何累积，均是配置的一部分，不能从“使用 AdamW”推断出来。大 batch 通过梯度累积实现时，需对累积次数正确缩放损失，并在预定边界更新参数与学习率。

训练内存除了参数，还有梯度、优化器矩和反向需要的激活。若权重与梯度各用 2 bytes，FP32 主权重用 4 bytes，两个 FP32 矩各用 4 bytes，则状态合计为 $16N$ bytes。这是某种实现布局的估算，不是所有混合精度训练的恒等式。BF16 的指数范围较宽，通常无需沿用 FP16 的动态 loss scaling；是否保存 FP32 主权重也取决于优化器和框架。激活检查点以重算换保存量，状态分片则把参数、梯度或矩分散到多卡，二者压缩的是不同对象。

10.4 Scaling law 是带实验范围的经验模型

一个常用的损失拟合写成

$$\mathcal{L}(N,D) = \mathcal{L}_\infty + aN^{-\alpha} + bD^{-\beta}, \qquad a,b,\alpha,\beta > 0. \tag{10.6}$$

这里 N 是参数量，D 是实际训练 token 数，拟合要固定数据分布、架构族、训练目标与评测方式。经典规模化研究及计算最优训练研究说明了这种关系的用途，也展示了结论对实验设置的依赖 [30, 31]。

设计算近似满足 $C = \kappa ND$，代入 $D = C/(\kappa N)$，对 $\log N$ 求导，内部最优点满足

$$\alpha a N^{-\alpha} = \beta b D^{-\beta}. \tag{10.7}$$

因此，在这组假设下有

$$N \propto C^{\beta/(\alpha+\beta)}, \qquad D \propto C^{\alpha/(\alpha+\beta)}. \tag{10.8}$$

参数和数据应一起增长，具体增长比例由拟合指数决定。常见的训练 FLOPs 粗估 $C \approx 6ND$ 还假设稠密网络、前向与反向的典型成本比，并忽略或吸收注意力、输出头等项；长序列和稀疏专家会改变这个近似。

计算最优也不等于部署最优。若一个模型训练一次却被调用很多次，训练更多 token 的较小模型可能降低后续服务成本；若数据难以继续扩展，继续增加重复次数也不保证遵循原有幂律。把某个实验区间内的 token/parameter 比例当成固定自然常数，会丢掉这些决策条件。

离散准确率的突变也需要谨慎解释。假设正确答案相对错误答案的 logit margin 从 −0.01 连续变为 0.01，贪心准确率便从错变对；连续的概率变化可在阈值指标上产生突跃。研究能力随规模的变化时，应同时查看损失、margin、任务准确率和样本分组，不能仅凭一条阶跃曲线推断机制突然出现。

10.5 SFT：从续写语料到条件响应

监督微调（SFT）将提示与答案组织为训练序列。若答案位置集合为 $\mathcal{A}_i$，一种按全部有效答案 token 归一化的批目标为

$$\mathcal{L}_{\mathrm{SFT}} = -\frac{\sum_i \sum_{t\in\mathcal{A}_i} \log p_\theta(x_{i,t} \mid x_{i,<t})}{\sum_i |\mathcal{A}_i|}. \tag{10.9}$$

提示部分可以参与前向和注意力，同时不直接计入损失。若改成先对每条样本平均再对样本平均，长答案与短答案的相对权重就改变了。多轮对话还必须写清楚哪些 assistant 回合受到监督、结束标记是否纳入目标，以及 chat template 的版本。

SFT 既能改变格式，也能改变任务策略和知识，但其效果由样本覆盖与优化决定。“SFT 只教格式”与“SFT 会全面增强能力”都过强。少量正确解题轨迹可以提高目标任务的条件概率，也可能使模型学会长度、措辞或模板上的捷径；应通过新题与模板变体区分两者。

LoRA 在冻结基底矩阵 $W \in \mathbb{R}^{d_{\mathrm{in}}\times d_{\mathrm{out}}}$ 时，采用行向量约定

$$W' = W + sAB, \qquad A \in \mathbb{R}^{d_{\mathrm{in}}\times r}, \quad B \in \mathbb{R}^{r\times d_{\mathrm{out}}}, \tag{10.10}$$

常见缩放为 $s = \alpha/r$，具体实现也有其他约定。可训练参数由 $d_{\mathrm{in}}d_{\mathrm{out}}$ 变为 $r(d_{\mathrm{in}} + d_{\mathrm{out}})$，但梯度仍需穿过冻结基底和后续网络，激活与计算并未同等消失 [32]。QLoRA 将冻结基底以低比特格式保存，训练低秩适配器；“基底以 4 bit 保存”不意味着全部前向、反向和累积都用 4 bit[21]。

10.6 偏好优化：从 KL 约束到 DPO

对给定提示 x，令 π_{ref} 为固定参考策略，$r(x, y)$ 为序列奖励。一种理想化目标为

$$\max_{\pi} \sum_{y} \pi(y \mid x) r(x, y) - \beta D_{\mathrm{KL}}\big(\pi(\cdot \mid x) \| \pi_{\mathrm{ref}}(\cdot \mid x)\big), \quad \beta > 0. \tag{10.11}$$

假设参考策略在候选支持集上为正，奖励与归一化常数有限，在概率总和为一的约束下求驻点可得

$$\pi^*(y \mid x) = \frac{\pi_{\mathrm{ref}}(y \mid x) \exp(r(x, y)/\beta)}{Z(x)}. \tag{10.12}$$

于是 $r(x, y) = \beta \log[\pi^*(y \mid x)/\pi_{\mathrm{ref}}(y \mid x)] + \beta \log Z(x)$。若偏好遵循 Bradley–Terry 模型 $P(y_w \succ y_l \mid x) = \sigma(r(x, y_w) - r(x, y_l))$，两条回答的奖励相减时 $\log Z(x)$ 抵消。把未知最优策略以 π_θ 参数化，就得到 DPO 的成对负对数似然 [33]：

$$\Delta_\theta = \log \frac{\pi_\theta(y_w \mid x)}{\pi_{\mathrm{ref}}(y_w \mid x)} - \log \frac{\pi_\theta(y_l \mid x)}{\pi_{\mathrm{ref}}(y_l \mid x)}, \tag{10.13}$$

$$\mathcal{L}_{\mathrm{DPO}} = -\log \sigma(\beta \Delta_\theta). \tag{10.14}$$

偏好噪声温度可吸收到奖励或系数约定中。这里的序列 log probability 是答案 token 的条件 log probability 之和；擅自除以长度，会得到另一目标。

DPO 避免了经典流程中显式奖励模型与在线策略优化的组合，却没有消除偏好数据的覆盖问题。其推导也不证明有限离线样本上的神经网络优化等价于求解任意奖励下的全局最优策略。参考策略、长度分布、标注偏差与分布外回答都影响最终行为。经典 RLHF 则可采样当前策略的输出并通过奖励模型更新，代价是额外的探索、估计与训练稳定性问题 [34]。

10.7 可验证奖励与测试时计算

答案精确匹配、代码测试和形式验证器可以提供比开放式偏好更明确的部分反馈，但仍有验证覆盖范围。通过测试的程序可能只适配有限样例；正确最终答案也不保证每一步解释正确。奖励定义应区分结果正确、过程合规与输出格式。

模型生成更长轨迹、采样更多候选、调用外部验证器，与参数更新带来的能力改变是不同干预。比较“推理增强”时，应固定或同时报告输出 token 预算、候选数、选择规则、工具调用数与总时间。如果只允许新方法多采样十次，成功率提高不能全部归因于训练方法。相反，在相同预算下提高首个候选成功率，才直接支持策略分布发生了有益变化。

量化会进一步移动概率和序列排名。某些任务上短答案损失几乎不变，长生成成功率却下降；这并不要求假设每一步独立同分布地积累噪声，而可能来自少数关键 token 的 margin 翻转和轨迹分叉。第 15 章会把概率误差、解码设置与任务结果分开记录。

10.8 MoE：总容量与每 token 工作量分离

稀疏专家混合（MoE）常以多个 FFN 专家替换部分稠密 FFN。对于行向量 $x \in \mathbb{R}^d$，router 产生 E 个分数，选出集合 $\mathcal{S}_k(x)$，输出

$$y = \sum_{e \in \mathcal{S}_k(x)} g_e(x)\ \mathrm{FFN}_e(x). \tag{10.15}$$

若使用选中专家内归一化，$g_e = \exp r_e / \sum_{j \in \mathcal{S}_k} \exp r_j$；若先对全部专家 softmax 再截取，选中权重一般不再和为一。这两种约定应从实现确认，不能只写一个含糊的 top-k。

设每个专家有 N_e 个参数，始终激活的部分有 N_s 个参数。总量约为 $N_s + EN_e$，每 token 使用的参数约为 $N_s + kN_e$。后者只是粗略的计算代理：路由、共享层、注意力和调度另有代价，而且一个批次可能覆盖全部专家。全部专家仍需常驻、分片或按需加载。稀疏激活不等于只需存储被单个 token 选中的那几个专家。

在一批 U 个 token、每 token 选择 k 个专家时，总分配数为 kU。平均每专家接收 kU/E，设容量因子为 c，一种容量上限可取 $\lceil ckU/E \rceil$。超过容量时，可以丢弃、转发或延后 token；采用哪种方案会改变训练目标或服务行为。负载均衡损失鼓励利用多个专家，但不保证每个设备的计算、通信和实际时延相等 [35]。

专家分布在多设备上时，token dispatch 与结果回收形成通信。热点专家、批量大小和设备拓扑可能使“每 token FLOPs 更少”的模型反而更难服务。量化能缩小专家权重，router 的数值误差却可能改变离散选择；应比较路由分布、专家命中率与端到端质量，而不是假定所有部分使用相同位宽就最优。

把目标函数变成可检查的预测

设两条答案分别有 10 与 100 个有效 token。比较按 token 平均与按样本平均的 SFT 目标，写出每个 token 的系数。再固定一个提示、两条回答及参考策略，检查 DPO 中把序列 log probability 改成平均 log probability 后，长度如何进入偏好 margin。提示：先把两种损失完全展开，不要只比较最后的标量。

本章的共同线索是概率分布的重分配。预训练定义希望拟合的数据分布，后训练改变提示下答案的相对概率，MoE 改变实现这些概率所用的参数路径。下一章回到固定参数的推理阶段，把这些结构换算成可复算的字节与运算量。

第 11 章　推理的内存账本与计算账本

模型文件大小、设备已分配显存、一次前向的 FLOPs 和用户等待时间是四个不同的量。本章先把张量的数量、精度和生命周期写清楚，再用 roofline 解释可能的瓶颈。所得数字是教学配置的解析估算，不是某个产品或某块设备的实测性能；真实执行还要接受第 15 章的测量检验。

11.1　先固定单位与符号

本书用 N 表示参数个数，D 表示训练 token 数；B 为同时活跃的序列数，P 为提示长度，G 为生成长度，S 为某一步实际可见的 KV 长度。T 表示一段当前参与矩阵计算的序列长度。位宽 b_w, b_{kv} 的单位均为 bit，换成 bytes 时必须除以 8。$1\,\mathrm{GB} = 10^9$ bytes，$1\,\mathrm{GiB} = 2^{30}$ bytes；$1\,\mathrm{KiB} = 2^{10}$ bytes。容量与带宽相除前，应把单位化成同一体系。

一次运行的峰值显存可以按不重复计数的对象分解为

$$\begin{aligned} M_{\text{peak}} = M_W + M_{KV} + M_{\text{act}} + M_{\text{work}} \\ + M_{\text{runtime}} + M_{\text{reserved slack}}. \end{aligned} \tag{11.1}$$

其中权重与 KV 是跨步持久状态；激活是当前计算的中间值；工作区是算子、图执行或通信需要的临时缓冲；运行时包括其他持久对象。最后一项包含池中暂时未用和分配粒度造成的余量。框架的 allocated 与 reserved、驱动看到的使用量可能覆盖不同集合，所以不能把几个监控数字直接相加。

峰值是时间上的最大同时存活量，不是逐项各自峰值的必然总和。一个算子的工作区可以复用前一个算子释放的内存；反之，异步执行、流水并行或为图回放固定地址，可能让本以为已释放的对象继续存活。解析账本用于发现必需的底数，分配器快照与执行轨迹用于确认实际重叠。

11.2　参数量从矩阵形状相加

考虑 L 层 decoder，隐藏维度 $d = h_q d_h$，KV 头数 h_{kv}，SwiGLU 中间维度 d_{ff}。忽略 bias 与 norm 参数时，每层注意力投影参数为

$$N_Q + N_O = 2d^2, \tag{11.2}$$

$$N_K + N_V = 2d\,h_{kv} d_h, \tag{11.3}$$

SwiGLU 的 gate、up、down 三个矩阵共有 $3dd_{\text{ff}}$ 个参数。因此，输入 embedding 与输出头共享时

$$N \approx L\big(2d^2 + 2dh_{kv}d_h + 3dd_{\text{ff}}\big) + Vd. \tag{11.4}$$

未共享时再加 Vd。即使共享权重，推理时的词表投影仍要计算；参数存储共享不等于免去一次矩阵乘法。MHA 取 $h_{kv} = h_q$ 时，注意力项回到 $4d^2$。GQA 改变的是 K/V 投影项，Q/O 与 FFN 不会同比缩小。

若每个参数均用同一精度存储，$M_W = Nb_w/8$；混合配置应分组求和。量化后的存储还要包含 scale、zero-point、codebook、异常值支路、未量化层和对齐。对称分组量化中，每 g 个权重使用一个 b_s bit scale，忽略边界填充时

$$b_{w,\text{eff}} = b_w + \frac{b_s}{g}, \qquad M_W \approx \frac{N b_{w,\text{eff}}}{8}. \tag{11.5}$$

若还保存同组 zero-point，应再加其平均位宽。真实矩阵各组末尾可能补齐，组数应按每个张量的实际切分取整，而不是对全模型参数总量只取一次整。

11.3 KV 是逐层、逐请求增长的持久状态

标准 GQA 每层缓存 K 与 V，单个缓存的逻辑形状均为 $[B, h_{kv}, S, d_h]$。在等长批次、K/V 同精度且没有额外状态的假设下

$$M_{KV} = 2LBSh_{kv}d_h\frac{b_{kv}}{8}. \tag{11.6}$$

对请求长度不同的服务批次，应把 BS 替换为 $\sum_{i=1}^{B} S_i$。若 K、V 的精度或维度不同，则分开求和。MLA、滑窗、跨层共享或 recurrent 层也不能直接套用标准 GQA 式，必须从实际持久张量重新记账。

分页管理使物理容量按页分配。设每页容纳 r 个 token，仅考虑每请求末页的内部浪费，分配的 token 槽位为

$$A_{\text{slots}} = r\sum_i \left\lceil \frac{S_i}{r} \right\rceil, \qquad 0 \le A_{\text{slots}} - \sum_i S_i < Br. \tag{11.7}$$

共享前缀时，多个逻辑页还可指向同一物理页，因此实际存储要对物理页去重。分页减少浪费，却不会把每个有效 K/V 元素的数值表示自动压缩。量化减少每元素平均位数，而 residual window 又使近期 token 保留较高精度；这三项应分别入账。

11.4 一个贯穿全章的虚构模型

取教学配置

$$\begin{gathered} L = 32, \quad d = 4096, \quad d_{\text{ff}} = 11008, \quad V = 32000, \\ h_q = 32, \quad h_{kv} = 8, \quad d_h = 128, \end{gathered} \tag{11.8}$$

绑定 embedding 与输出头，并继续忽略 norm/bias。由式 (11.4) 得到

$$N = 5\,801\,771\,008. \tag{11.9}$$

这是一组用来复算的结构参数，不对应某个声称达到特定质量的模型。表 11.1 将同一配置下的存储列在一起。

表 11.1 解析内存示例；不含激活、工作区、分配余量与运行时

对象与条件	bytes	GiB
BF16 权重，16 bit/参数	11 603 542 016	10.8066
理想分组 W4，FP16 scale/64 权重	3 082 190 848	2.8705
BF16 KV，$B = 1, S = 4096$	536 870 912	0.5000
BF16 KV，$B = 8, S = 16384$	17 179 869 184	16.0000
BF16 KV，$B = 1, S = 131072$	17 179 869 184	16.0000
理想 KV4.25，$B = 8, S = 16384$	4 563 402 752	4.2500

每条序列每多缓存一个 token，BF16 KV 增量为 $2 \times 32 \times 8 \times 128 \times 2 = 131072$ bytes，即 128 KiB。把 GQA 换成同样 $h_q = 32$ 的 MHA，KV 增加为四倍。表中的 KV4.25 只表示每元素有效 4.25 bit 的理想账本，未承诺某种量化配置达到相同质量，也不包含近期高精度窗口。

在 $B = 8, S = 16384$ 时，仅 BF16 权重与 KV 就需要约 26.8066 GiB，已经超过 24 GiB 的可用预算。若使用表中两种理想量化，二者降为约 7.1205 GiB；这个数说明持久状态变小，却还不足以证明任意实现都能运行。仍需添加真实未量化层、量化元数据、算子临时峰值和运行时开销。

11.5 激活与工作区：为什么不能一律乘层数

训练为了反向传播，可能保存多层激活；普通无梯度推理可以逐层释放已用中间值，所以推理激活通常不能直接按 L 倍的单层峰值估计。对当前 $n = BT$ 个 token，一份残差激活含 nd 个元素；SwiGLU 的 gate/up 中间结果各含 nd_{ff} 个元素。是否同时物化两个分支、是否融合非线性与乘法，以及输入是否需要保留给残差相加，决定峰值。

朴素注意力若物化 score，张量大小为 Bh_qT^2；若还物化概率，会再多一个同阶对象。因果掩码使一半逻辑位置无效，不意味着使用完整稠密张量的实现就自动只占一半内存。FlashAttention 的价值之一是分块处理并避免把完整 score/probability 写入高带宽显存；其最终输出仍有 BTd 个元素，必要的 Q/K/V 和运行状态也仍存在。

输出 logits 也是容易漏掉的一项。全序列似然评估若同时物化 $[B, T, V]$，一个 $B = 1, T = 4096, V = 32000$ 的 FP32 张量就占约 0.4883 GiB。只为生成首个 token 的 prefill 通常只需最后有效位置的 logits，不必保留每个位置的完整词表向量。评测与服务采用不同路径，测到的显存不可无条件互换。

工作区与算法选择有关。某个矩阵乘法算法可能用额外缓冲换取吞吐；CUDA graph 捕获可能固定多种形状的地址；张量并行还有通信缓冲。应记录显存峰值出现在哪个阶段，并同时报告模型加载后基线、prefill 峰值与稳定 decode 峰值。一次内存不足不必然说明持久权重太大，也可能是某个长 prompt 的临时张量过大。

11.6 用 FLOPs 解释 prefill 与 decode

约定一次乘加计为 2 FLOPs。忽略 bias、norm、激活函数、softmax 和旋转的较小项，令

$$N_{\text{blocks}} = L\left(2d^2 + 2dh_{kv}d_h + 3dd_{\text{ff}}\right). \tag{11.10}$$

处理 BT 个新 token 的线性投影约需 $2BTN_{\text{blocks}}$ FLOPs。输入 embedding 是索引读取，不能像普通稠密投影那样再记一次 $2BTVd$。

在无历史缓存的因果 prefill 中，每头有效 query–key 对数为 $T(T+1)/2$。每对包含 QK 点积与概率乘 V，两部分合计约 $4d_h$ FLOPs，因此

$$F_{\text{attn,prefill}} \approx 2BLh_qd_hT(T+1). \tag{11.11}$$

这是按因果有效元素计的算法量；执行完整方阵再掩码的实现会做更多工作。若在 S_0 个历史 token 后一次加入 T 个新 token，每头有效对数改为 $TS_0 + T(T+1)/2$。

在 decode 中，每条请求只加入一个 token，若本步可见缓存长度为 S，则

$$F_{\text{decode}} \approx 2BN_{\text{blocks}} + 4BLh_qd_hS + 2BdV. \tag{11.12}$$

最后一项是每请求当前 token 的输出头。GQA 降低缓存头数，但逻辑上每个 query head 仍要计算注意力，所以注意力 FLOPs 中使用 h_q，不是 h_{kv}。

在教学配置 $B = 1, T = 4096$ 下，各层线性投影约为 46.4544 TFLOPs，因果注意力约为 4.3991 TFLOPs。若计算全部位置的词表 logits，再加约 1.0737 TFLOPs。这些是解析运算计数；这里

的 TFLOPs 表示 10^{12} 次运算，不是每秒性能。实际 kernel 可能融合、重算、跳过 padding 或执行额外的对齐计算。

生成总成本也不能只写一个固定的“每 token 成本”。提示为 P、生成 G 个 token 时，各步历史长度逐渐增长，decode 注意力的长度和含有

$$\sum_{j=1}^{G}(P+j) = GP + \frac{G(G+1)}{2} \tag{11.13}$$

这样的项，具体首步索引由 prefill 是否已经产出首个采样分布决定。长生成的末段可能比开头慢，平均每 token 时间会掩盖这个趋势。

11.7 Roofline 的适用对象是具体数据移动层级

算术强度定义为 $I = F/Q$，其中 Q 是选定存储层级实际移动的 bytes。若讨论设备显存带宽，则 Q 应是 HBM 读写量；若讨论寄存器或片上 SRAM，不能沿用同一个 Q。经典 roofline 上界为

$$\text{Perf} \le \min\bigl(\text{Perf}_{\text{peak}},\, IBW\bigr). \tag{11.14}$$

等价的理想时间下界是

$$t \ge \max\left(\frac{F}{\text{Perf}_{\text{peak}}}, \frac{Q}{BW}\right). \tag{11.15}$$

它不是时间预测器：小矩阵、访存不合并、同步、启动与通信会使实际时间更长。混合精度或不受硬件直接支持的低比特乘法也没有一个可随意套用的“峰值”。

考虑 $Y = XW$，$X \in \mathbb{R}^{m\times k}$、$W \in \mathbb{R}^{k\times n}$。若权重是主要流量且每次从 HBM 读一遍，$F \approx 2mkn$，$Q \approx knb_w/8$，于是

$$I \approx \frac{16m}{b_w}\ \text{FLOPs/byte}. \tag{11.16}$$

批内 token 行数 m 越多，单次权重读取可复用得越充分。单请求 decode 的 $m = 1$，BF16 权重下约为 1 FLOP/byte；大 prefill 则有更多行。这个估算解释倾向，不证明所有 decode 都受带宽限制：足够大的 batch、长 KV、通信与特定矩阵形状都会改变主导项。

假设仅作下界演示，权重必须读出 11.6035 GB，可用 HBM 带宽被设为理想的 1 TB/s，则仅权重读取就需至少约 11.60 ms。这是人为指定带宽下的算术结果，没有对应设备实测；KV、写回、解包、未达带宽和同步只会增加总时间。不能把它写成该模型的生成速度。

11.8 量化与批处理改变哪个瓶颈

减小权重 bytes 会提高式 (11.16) 中的算术强度，可能降低带宽时间；但 scale 读取、解包与反量化也增加流量和运算。若原本已经计算受限，单纯压缩文件大小不保证更快。长上下文时，即使权重很小，持续读取 KV 也可能成为主要开销。理想 GQA decode 中，若每个 K/V 只从 HBM 读一次，KV 读取底数与式 (11.6) 同阶；query 头共享是否得到实际复用，还取决于 kernel。

增大 B 可让多个请求共享一次权重读取，提高总 token 吞吐，却同时扩大 KV 与排队风险。单请求速度、总吞吐和满足延迟要求的有效吞吐是不同目标。若原耗时中比例 p 的部分加速 s 倍，其他部分不变，则总加速为

$$S_{\text{Amdahl}} = \frac{1}{(1-p) + p/s}. \tag{11.17}$$

例如 $p = 0.6, s = 4$ 时总加速约 1.82 倍；若额外增加调度成本，收益还会下降。优化前应先定位时间和流量的主项；账本练习及提示见附录 B.6 节。

第 12 章　从注意力 IO 到服务执行

第 11 章给出了模型所需的有效计算与状态底数，服务系统还要决定这些张量何时计算、放在哪里、由哪些请求共同使用。FlashAttention、PagedAttention 和连续批处理经常同时出现，但分别作用于算子 IO、物理内存布局与请求调度。把三个层次分开，才能解释某项优化为什么在一种负载下有效、换一个负载后却没有收益。

12.1　FlashAttention：不物化整个注意力矩阵

普通注意力写成 $O=\mathrm{softmax}(QK^{\top}/\sqrt{d_h}+M)V$。若先将全部 score 写入 HBM，再读回做 softmax，再保存概率并读回乘 V，T^2 级中间张量会产生大量 IO。FlashAttention 将 Q、K、V 分块载入片上存储，计算一个局部 score 块后立即更新输出，不把完整 score 与概率写回 HBM[36]。

它依赖在线 softmax。对固定 query 行，设已处理键集合上的最大 score 为 m，归一化分母为 ℓ，未归一化加权和为 $u\in\mathbb{R}^{d_h}$：

$$\ell=\sum_{j\in\mathcal{J}}\exp(s_j-m),\qquad u=\sum_{j\in\mathcal{J}}\exp(s_j-m)v_j. \tag{12.1}$$

加入新块 $\mathcal{B}$，令

$$m'=\max\bigl(m,\max_{j\in\mathcal{B}}s_j\bigr),\quad a=\exp(m-m'),\quad e_j=\exp(s_j-m'), \tag{12.2}$$

即可更新

$$\ell'=a\ell+\sum_{j\in\mathcal{B}}e_j,\qquad u'=au+\sum_{j\in\mathcal{B}}e_jv_j. \tag{12.3}$$

最终 $o=u/\ell$。旧贡献乘 a 是因为参考最大值从 m 改成了 m'。若只更新分母而不相应缩放 u，就不再是原来的 softmax。

初始空集合可视为 $\ell=0,u=0,m=-\infty$，但实现需避免直接计算 $\exp(-\infty-(-\infty))$。全被掩码的块应跳过；若某个 query 的全部键都无效，应由接口定义处理，而不能让除零或 NaN 悄悄传播。普通 decoder 至少允许关注自身，所以有效行通常有合法键。

在实数算术中，式 (12.3) 只是改变求和次序，没有删掉合法的 query–key 对；这就是文献中 exact attention 的含义。浮点求和顺序、累积精度和融合算子仍可使结果与朴素实现有微小差异。因此验证应比较给定绝对/相对容差下的输出及梯度，而不是要求跨 kernel 逐 bit 相等。

这种 IO 改写没有把全注意力的算法量从二次变为线性。它减少 HBM 中间量，并通过分块提高复用；实际收益依赖序列长度、head 维度、掩码、硬件与实现。Decode 时 query 只有一行或很少几行，主要问题可能是读取长 KV，kernel 的并行策略也与大 prefill 不同。不能仅凭 prefill benchmark 推断 decode 同样快多少。

12.2　PagedAttention：逻辑序列不要求物理连续

KV 随请求生成而增长。如果每条请求预留一个等于最大长度的连续区域，短请求会浪费空间；动态扩展又可能造成搬迁和碎片。分页方法将序列切成固定 token 页，使用页表把逻辑页映射到物理块。Attention kernel 按页表读到所需 K/V，而不要求整条序列在物理地址上连续 [26]。

例如页大小为 4 token，长度为 10 的请求有逻辑页 0、1、2。其物理块可以分别位于 7、2、11；最后一页只有两个有效 token。页表改变寻址方式，因果位置仍按逻辑 token 编号判断。把物理块号误当成位置，会同时破坏掩码与位置编码。

分页的收益来自按需分配、较小的末页浪费和共享页面。多个请求可以引用同一段不可变前缀，后续修改共享末页时再复制，以维护各自的逻辑状态。它不是自动把 KV 从 GPU 换出到磁盘的完整方案；offload、抢占与恢复需要额外机制。它也不会减少有效 token 数、head 数或每元素位宽，这些属于注意力结构或量化的选择。

页大小是折中。小页减少末页浪费，却扩大页表和间接寻址成本；大页有利于连续访存，但短请求或频繁分支可能浪费更多槽位。物理碎片、页表开销与 kernel 效率应在真实长度分布下评测。一个仅比较“理论可容纳多少 token”的表，无法证明实际服务吞吐同比增加。

表 12.1 三类优化的作用位置

方法	主要改变	仍需承担
FlashAttention	分块执行与在线归一化，减少中间 IO	合法注意力对的计算与持久 KV
PagedAttention	逻辑页到物理 KV 页的映射	有效缓存元素与页表寻址
连续批处理	每轮参与执行的请求集合	单请求的自回归依赖与服务排队

12.3 连续批处理：在迭代边界更换活跃集合

一个朴素固定批次预先选好若干请求，执行直到最长请求结束。较短请求结束后，若实现继续保留其槽位并做 padding，便浪费计算。固定批次也可通过 mask 或 compaction 缓解浪费，所以“静态批处理必然等待全部完成”是对特定策略的描述，不是数学限制。

连续批处理进一步允许在生成迭代边界移除完成请求，并加入已经准备好的新请求。单个批次不再是从头到尾固定的集合，而是每轮调度器给执行器的一组工作 [37]。当不同请求的 prompt 长度、已缓存长度和停止时间不同，执行器需携带各自的页表、位置、采样状态与输出缓冲。

共享一次矩阵计算可以提高权重复用，但不能打破同一请求的 token 依赖：在精确自回归生成中，下一 token 的条件分布仍依赖当前采样结果。请求间可以并行，单请求内部只有借助第 13 章的验证机制才能安全减少串行大模型调用。

调度还要处理接纳控制。若只按当前剩余显存接纳大量短 prompt，却忽略后续生成的 KV 增量，系统可能在生成中途耗尽页。可以预留生成预算、限制活跃 token 数、在压力下抢占，或采用可恢复的换出策略。不同方案分别付出保守利用率、等待时间或传输成本；不能只看稳定运行中某一瞬间的最大批大小。

12.4 分块 prefill：保持因果语义，改变执行粒度

一个长 prompt 的完整 prefill 可能占用较长执行区间，使正在生成的其他请求迟迟得不到下一步。分块 prefill 把新 prompt 切成若干块，与 decode 工作交错安排。第 j 块的 Q 既关注前面已缓存的块，也关注本块内合法的因果位置。

若每块都重置位置、只在块内 attention，或不保留此前 K/V，得到的就不是同一全上下文模型。正确分块应保持全局位置、掩码与缓存一致；其输出在给定浮点容差下与一次完整 prefill 对应。这里讨论的是执行分块，不是滑窗或块稀疏注意力的建模近似。

设一轮允许的新增 token 预算为 U，调度器可以先分配给活跃 decode 请求，再用剩余预算处理 prefill 块。长 prompt 于是不会一次占据所有轮次，但过小的块使矩阵更瘦、启动更频繁，还可能反复读此前 KV。预算过大则仍可能产生较长 ITL。最佳块大小依赖请求分布与延迟目标，不能从一个孤立的矩阵乘法峰值推导。

一个有用的实验是固定全部请求及到达时间，仅改变分块 token 预算，画出 TTFT、ITL 尾分位与总输出吞吐三条曲线。若 TTFT 改善而 ITL 恶化，应检查调度优先级；若两者同时改善但吞吐下降，则可能是通过留出余量换取延迟。应把这个折中写成观察，而非把所有指标都称为“加速”。

12.5 前缀复用的条件比字符串相同更严格

两个请求若具有相同的有效前缀计算，可以复用该前缀各层的 K/V，省去重复 prefill。所需身份至少包括模型 revision、适配器、token ID 序列、位置与掩码语义。对多模态模型，视觉输入或其 embedding 也属于计算身份；相同占位 token 不代表相同图像。

RoPE 缩放配置、缓存精度、量化布局和模型推理模式也会影响状态。如果仅以可见文本为 key，模板更新或 tokenizer 改动可能命中错误缓存。缓存通常要求 eval 模式下前缀计算可复用；若启用随机 dropout 或依赖外部动态状态，需重新定义一致性条件。

共享公共 system prompt 与共享私人文档前缀具有不同的隔离要求。这里的技术原则是：缓存的命中与可见性不能超出请求本来具有的数据访问范围。它与是否按物理页共享不是同一个问题，应由缓存命名空间和服务接口明确表达。

命中率也不等于省时比例。命中长前缀有望省去更多计算，但当前请求仍须执行未缓存部分、读取缓存并等待调度。占用大量显存的缓存还可能挤掉活跃请求。报告收益时，应同时给出命中 token 数、命中请求比例、占用 bytes 与具体负载下的延迟，而不是只报一个百分比。

12.6 模型并行从矩阵分解理解

张量并行把单层运算分散到多设备。以两个线性层之间的逐元素非线性为例，令

$$W_1 = [W_1^{(1)} \ \cdots \ W_1^{(K)}], \qquad W_2 = \begin{bmatrix} W_2^{(1)} \\ \vdots \\ W_2^{(K)} \end{bmatrix}. \tag{12.4}$$

每设备可计算 $z_i = \phi(XW_1^{(i)})W_2^{(i)}$，最终 $Y = \sum_i z_i$。第一层按输出特征切，第二层按输入特征切，中间非线性在本地作用，末端通过规约合并。这解释了为什么张量并行常有层内通信，而不是把模型文件均分后各算各的。

注意力可按 head 划分，但 GQA 的 KV 头数可能小于并行度，某些实现于是复制 KV 头或采用其他布局。单卡 KV 不一定简单等于总 KV 除以设备数。稀疏专家的 expert parallel 则把不同专家放在不同设备上，按 router 结果进行 token dispatch；通信量由分配的 token、隐藏维度、精度和放置决定。

流水并行按层切分。一个请求的一步仍需依次经过各阶段，多条请求或多个 microbatch 才能填充流水线。阶段划分不均会造成等待，单请求延迟也包含跨阶段传输。多副本把整个模型复制到多组设备并分配独立请求，它提高服务容量，但不减少单个请求需要经过的层数。三种策略可以组合，其内存与通信账本必须跟随实际布局重算。

训练中的 ZeRO/FSDP 主要切分参数、梯度和优化器状态，并按需收集参数；推理中的张量并行、

流水与副本调度解决不同负载。仅因为某训练配置能放进多卡，就推断相同布局具有合适的 decode 延迟，缺少执行层面的依据。

12.7 分离 prefill 与 decode 也要支付状态传输

Prefill 具有较大矩阵计算，decode 具有小批逐步计算与持续 KV 读取，因而可以考虑将二者放在不同设备池上。这样能分别调度，但首 token 前后需要转交请求状态，尤其是全部层的 KV。若需传输的缓存为 M_{KV} bytes，链路有效带宽为 BW_{link}，即使完全忽略同步和格式转换，传输时间也至少为

$$t_{\text{transfer}} \geq M_{KV}/BW_{\text{link}}. \tag{12.5}$$

压缩、流水传输与复用可改变这笔代价；格式、位置与量化配置不兼容又可能要求转换。长 prompt、省下的排队时间、链路带宽和服务负载共同决定是否划算，不能仅凭 prefill 与 decode 瓶颈不同就得出必然收益。

从现象定位到执行层次

若优化后可容纳更多并发，却没有提高单请求 tokens/s，先检查它是否主要减少了分页浪费。若 prefill 峰值显存大幅下降而长 decode 变化不大，检查是否只是避免了完整 score 张量。若总吞吐提高、尾延迟变差，检查调度与接纳策略。每个现象先对应一个张量生命周期或调度决策，再讨论性能原因。

这些机制在数学模型不变的情况下重排工作、减少 IO 或改善利用率。下一章研究另一类优化：允许便宜的模型先猜测未来，但通过精确验证仍保持目标模型的采样分布。

第 13 章　推测解码：用可回滚的猜测保持目标分布

普通自回归生成每得到一个 token，都要根据它计算下一步条件分布。跨请求批处理可以提高利用率，却不能直接解除同一请求的这条依赖链。推测解码让一个便宜的 draft 先提出若干候选，再让 target 沿这条候选路径并行计算条件概率；只有通过验证的前缀可以提交。关键在于验证规则：若要求保持随机采样分布，不能只检查两个模型的最大概率 token 是否相同 [38]。

13.1　先定义需要保持的分布

令 h 表示已经提交的 token 前缀。$p(\cdot \mid h)$ 是基线 target 在本步实际使用的归一化采样分布，$q(\cdot \mid h)$ 是 draft 实际提出候选时使用的分布。温度、top-k、top-p、重复惩罚和约束规则，都应在定义分布时考虑。这里要求的是最终输出遵循 p；q 可以采用不同的提议规则，只要验证器准确知道它抽样时的概率。

最直接的实现要求 draft 与 target 共享词表和 token ID 语义。若 tokenizer 不同，必须另外处理可变长度映射和路径概率，不能直接把两个词表向量逐元素相除。target 也必须在和基线相同的权重、位置、掩码及有效前缀上计算。若 target 本来就是量化模型，证明保持的是这个 target 的分布；不能用量化 target 验证后声称仍精确保持另一个高精度模型。

“保持分布”指任意输出序列的概率与基线相同，不表示给定随机种子后逐字相同。两个程序可以消费不同数量、不同顺序的随机数。以下先在精确概率算术下证明，再讨论浮点与缓存条件。

13.2　一个 token 的精确接受与拒绝

固定前缀，省略条件 h。先抽取 $y \sim q$，再抽取独立的 $u \sim \mathcal{U}(0,1)$。由于被抽中的候选满足 $q(y) > 0$，接受概率可以定义为

$$a(y) = \min\left(1, \frac{p(y)}{q(y)}\right). \tag{13.1}$$

若 $u \le a(y)$，提交 y；否则从残差分布重新抽取一个 token z：

$$r(x) = \frac{[p(x) - q(x)]_+}{\rho}, \qquad [v]_+ = \max(v, 0), \qquad \rho = \sum_x [p(x) - q(x)]_+. \tag{13.2}$$

当 $p = q$ 时 $\rho = 0$，所有候选都接受，残差分支不会到达，不应仍做一次除零运算。即使某个 token 的 $q(x) = 0, p(x) > 0$，也没有缺失支持的问题：它不能由 draft 直接提出，却可以在拒绝后从残差分布产生。

13.2.1　为什么不能拒绝后直接从 target 重采样

令总体接受概率为

$$\alpha = \sum_x q(x)a(x) = \sum_x \min(p(x), q(x)). \tag{13.3}$$

直接接受并输出 x 的概率质量已经是 $\min(p(x), q(x))$。若拒绝后再从原始 p 抽样，这部分会额外得到 $(1-\alpha)p(x)$，一般不会加成 $p(x)$。残差分布恰好补齐尚未被直接接受路径覆盖的质量，所以必须

使用式 (13.2)。

更明确地，$\rho = 1 - \alpha$，因为 $[p(x) - q(x)]_+ = p(x) - \min(p(x), q(x))$。最终输出 x 的概率为

$$\begin{aligned} \Pr(\text{输出 } x) &= q(x)a(x) + (1-\alpha)r(x) && (13.4)\\ &= \min(p(x), q(x)) + [p(x) - q(x)]_+ && (13.5)\\ &= p(x). && (13.6) \end{aligned}$$

这是接受/拒绝规则的完整局部证明。它不依赖 draft 一定很强，也不要求两个模型的 argmax 相同；draft 的质量影响效率，target 与校正规则决定分布。

三 token 词表上的质量补齐

设词表为 $\{a, b, c\}$，$p = [0.5, 0.3, 0.2]$，$q = [0.7, 0.2, 0.1]$。候选 a 的接受率为 5/7，b, c 的接受率均为 1，故直接接受的质量为 $[0.5, 0.2, 0.1]$，总计 0.8。拒绝概率为 0.2，残差为 $r = [0, 0.5, 0.5]$，补上 $[0, 0.1, 0.1]$，最终正好是 p。
若拒绝后直接从 p 抽样，得到的却是 $[0.6, 0.26, 0.14]$。这是一个无需运行大模型就能发现错误验证规则的反例。

式 (13.3) 还给出

$$\alpha = 1 - \frac{1}{2}\sum_x |p(x) - q(x)|. \tag{13.7}$$

所以本前缀的接受率由两个分布的总变差距离决定。PPL、KL、top-1 一致率分别衡量其他性质，都不能直接替代这个重叠量。即使两个 argmax 一样，其余概率质量仍可能差异很大；反过来，两个相近候选交换 argmax，也不一定使重叠量很低。

13.3 一轮验证多个候选：只提交连续前缀

设 draft 沿已提交前缀 h 自回归地产生 k 个候选：

$$q_i(\cdot) = q(\cdot \mid h, y_1, \dots, y_{i-1}), \qquad y_i \sim q_i, \quad i = 1, \dots, k. \tag{13.8}$$

然后 target 用一个遵守因果关系的分块前向，沿同一条候选路径计算

$$p_i(\cdot) = p(\cdot \mid h, y_1, \dots, y_{i-1}), \quad i = 1, \dots, k+1. \tag{13.9}$$

各位置能在同一次矩阵计算中得到 logits，是因为验证输入已经由 draft 给定，等同于对这条路径做 teacher forcing；不是 target 已经摆脱了自回归条件。p_i 不能看见 y_i 或未来候选，否则不再是验证所需的分布。

按 $i = 1, 2, \dots$ 顺序使用独立均匀随机数验证式 (13.1)。设第一个拒绝发生在 j：

1. 提交 $y_1, \dots, y_{j-1}$。
2. 由 $r_j = \text{normalize}([p_j - q_j]_+)$ 抽取并提交纠正 token z。
3. 丢弃 $y_j, \dots, y_k$ 以及依赖这些未提交 token 的状态，下一轮从 $h, y_1, \dots, y_{j-1}, z$ 继续。

若 k 个候选全部接受，则额外从 p_{k+1} 抽取一个 token 并提交。在未触发停止或长度上限时，一轮因此输出至少 1 个、至多 $k+1$ 个 token。额外 token 所需分布已经由最后一个候选位置的 target 隐藏状态给出；它的 K/V 是否已建立，是另外一件事。

拒绝后的后缀不能保留

若 y_j 被替换为 z，预先计算的 p_{j+1} 条件是 $h, y_1, \dots, y_j$，不包含 z。即使后续某个候选恰好等于此位置的 target argmax，也不能跳过拒绝位置继续提交。这些条件分布与 K/V 都绑定于错误的前缀。普通路径式推测解码接受的是连续前缀，不是若干通过独立检查的离散位置。

正确性可以由局部证明递推得到。给定已到达的合法前缀，候选分布是对应的 q_i，target 为对应的 p_i，独立验证随机数不引入额外条件依赖；式 (13.6) 使这一位置的最终输出遵循 p_i。接受后继续同一路径，拒绝后从纠正前缀重新开始；全接受时额外 token 直接来自正确的 p_{k+1}。沿输出顺序使用概率链式法则，最终序列就具有 target 的自回归分布。

13.4 把 logits 偏移与 KV 回滚写成状态事务

并行验证最常见的错误是偏移一位。约定本轮开始时，两个模型都已处理全部 n 个已提交 token，缓存长度为 n；同时保留最后位置预测下一 token 的 logits。此时保存的 target 分布就是 p_1。向 target 输入候选 $[y_1, \dots, y_k]$ 后，本次前向的第 i 行隐藏状态已经读入 y_i，所以对应的是 p_{i+1}。验证 y_i 需要 p_i，不能把本次前向同一行的 logits 直接与 y_i 对齐。

在这个约定下，target 一次因果前向可以把缓存暂时扩到 $n+k$，并给出 $p_2, \dots, p_{k+1}$；加上保存的 p_1，验证所需分布就齐了。draft 为简化状态管理也可在提出每个候选后将它送入自己的模型，使暂存缓存达到同样的逻辑长度。其他实现可能让最后一个 token 处于“已采样、未计算 K/V”的 pending 状态，这也可行，但必须记录有效长度与 logits 所属的前缀。

若第 j 项拒绝，应把两个模型的暂存缓存截回 $n + j - 1$，只保留原前缀和接受候选。纠正 token z 尚没有对应缓存，需要将它送入两个模型以建立新状态，或明确把它留作下一轮的第一个 pending 输入。若全部接受，额外 token $z \sim p_{k+1}$ 同样尚未被该次验证前向处理。把“已输出一个 token”误当成“其各层 K/V 已存在”，会在下一轮漏掉输入或重复消费 token。

分页缓存可以通过恢复逻辑长度和回收暂存页面完成回滚；若末页有共享写入，还要遵守写时复制。回滚范围不限于 K/V：位置计数、重复惩罚计数、语法约束自动机、停止字符串状态和输出流，都只能提交合法前缀。draft 的某些状态不可按长度直接切片时，需要快照或重算。任何已经发送给用户的文本都不宜再撤回，因此流式输出应以验证提交边界为准。

一轮结束后必须成立的不变量

两模型下一次使用的缓存与位置，必须对应同一条已提交 *token* 前缀；若有 pending token，需记录其尚未进入缓存。保存的 logits 必须预测该前缀之后的位置。所有未提交候选的 K/V、约束计数与可见输出都已回滚。满足这些条件后，才可以讨论连续轮次的分布保持。

13.5 EOS、约束与有限精度的边界

EOS 是分布中的一个 token，也要经过接受/拒绝。若 draft 提出 EOS，可以将候选路径截到它；若 EOS 接受，立即结束，不再抽取额外 token；若 EOS 被拒绝，则按同一残差规则产生纠正结果，不能因为 draft 提前结束就让 target 无条件结束。target 的纠正或额外 token 若为 EOS，同样停止。最大输出长度是外部截断：剩余额度不足时，只提交允许的 token，不能为了获得额外 token 超出限制。停止字符串则要在已提交解码文本上检查，不能由未验证的候选提前触发。

若约束或惩罚依赖历史，应对每个候选前缀分别构造 p_i, q_i；不能把第一位置的 top-p 集合或语法合法集复用到整段。draft 可以提出在 target 下概率为零的 token，它会被拒绝；若验证器把所有零

概率候选简单忽略，却继续使用其后缀，仍然破坏条件链。

在浮点实现中，可用

$$\log u \le \min\big(0, \log p(y) - \log q(y)\big) \tag{13.10}$$

减少比值的上溢风险。残差归一化仍需处理 $p \approx q$ 时的相消、负数舍入和归一化误差。任意加入接受阈值、把微小残差直接丢弃，或在拒绝时采用启发式 fallback，都不能再直接引用精确证明。应清楚区分数学算法与实现近似。

同样，分块验证与单 token 基线的浮点归约路径可能略有差别。检验时先比较同一前缀上的 logits 与概率，记录 dtype、容差和 kernel；再检查采样器是否实现了目标规则。分布保持不要求两条运行轨迹使用同一随机流，也不能用少量生成文本恰好相同来证明。

13.6 为什么少调用 target 仍可能没有加速

令 R 为一轮提交的 token 数，暂不考虑 EOS 与长度截断。一般有

$$\mathbb{E}[R] = 1 + \sum_{j=1}^{k} \Pr(\text{前 } j \text{ 个候选均接受}). \tag{13.11}$$

各位置的接受率随前缀变化，通常不独立。只有在额外采用每步接受事件独立、接受率恒为 α 的教学近似下，才得到

$$\mathbb{E}[R] = \sum_{j=0}^{k} \alpha^j = \frac{1-\alpha^{k+1}}{1-\alpha}, \quad \alpha \neq 1; \qquad \alpha = 1 \text{ 时为 } k+1. \tag{13.12}$$

例如人为设定 $\alpha = 0.8, k = 4$，结果为 3.3616；这只是该独立假设下的算术值，不能用全局平均接受率推断任意真实请求的相同期望。实测应保存接受长度分布，以及请求、长度和领域的差异。

每轮实际时间还包含 draft 的多步生成、target 的分块验证、概率处理和状态管理。令其分别为 $t_D(k)$、$t_V(k)$、$t_S(k)$，其中 t_S 包括必要的同步、回滚和 pending 输入处理。在负载近似平稳时，可以用

$$\text{每输出 token 平均耗时} \approx \frac{\mathbb{E}[t_D(k) + t_V(k) + t_S(k)]}{\mathbb{E}[R]} \tag{13.13}$$

与同一 target、同一请求条件下的普通 decode 比较。$t_V(k)$ 不应默认等于一次普通 decode；增加候选长度会改变矩阵形状、attention 计算、logits 大小和工作区。更大的 k 还增加 draft 延迟与低接受后缀的浪费。即使 target 调用数下降，总 FLOPs、峰值显存或端到端延迟也可能上升。

当小 batch decode 受权重读取限制、设备还有可利用的计算能力时，多个验证位置有机会复用权重读取。但大 batch、长 KV、已接近计算上限、便宜 target 或昂贵 draft，会改变收益。双模型常驻权重、各自 KV 和验证暂存区也要纳入显存账本。性能报告至少给出 draft/target 版本、采样配置、候选策略、接受长度、target 调用数、TTFT、每 token 延迟与总吞吐；只报“接受率很高”不足以证明端到端更快。

13.7 自推测、贪心与候选树的适用条件

Draft 可以来自小模型、提前退出分支或较低精度版本。只要它定义了可知的 q，精确校正不要求它与 target 共享全部结构；但共享权重不意味着中间 K/V 也可任意共享。若量化误差进入 target 的缓存或验证 logits，变化的已经是 p，不能再把它全部归为“只影响接受率”。模型、缓存和状态的边界必须画清楚。

若基线是贪心解码，可以把 p 看成在 target argmax 上的点质量，并固定并列值的规则；此时接受与否变成对确定性 token 的一致检查，拒绝后输出 target 的贪心结果。这个特例不能反过来证明 softmax 随机采样中“argmax 相同就接受”正确。两种基线应分别评测与实现。

候选树可以在一个 target 调用里验证多个分支，但需要使每个节点只看见祖先和合法历史，并保存分支对应的 K/V。若先从多个候选中按 target 分数择优，再套用单提议的 p/q，实际提议分布已经经过选择，证明通常不成立。树形、无放回候选或多提议算法要分别说明候选生成、分支选择与校正规则；本章的证明只覆盖上述单路径规则。树扩得更宽还会增加无效节点、掩码复杂度和暂存状态。

把分布证明与工程状态一起验证

1. 对三 token 算例独立列出“提出并接受”与“拒绝后重采样”的六条质量贡献，确认每个 token 的总和为 p。把 q 改为 $[1,0,0]$，解释为何仍能恢复同一个 p。
2. $k=3$，第二个候选首次被拒绝。本轮输出几个 token？缓存哪些位置可复用？提示：输出 y_1, z；原候选缓存只保留到 y_1，z 需另行计算或作为 pending 输入。
3. 证明式 (13.11)：用尾概率和表示一个介于 1 与 $k+1$ 的整数变量期望；再说明相关的接受事件为什么不能简单替换为平均率的幂。
4. 在一个极小词表上构造非均匀 p, q，用概率枚举核对算法，再以多次抽样检查频率误差。枚举检验规则正确性；统计频率只作为有限样本的实现诊断，须报告样本数和种子。
5. 给缓存附上其前缀 ID 的独立校验值，在拒绝、全接受、EOS 和达到长度上限四种路径上检查不变量。不要仅靠正常生成的一条路径覆盖回滚机制。

推测解码把“先计算”与“可提交”分开：先付出一批便宜猜测和并行验证，再用分布校正与状态回滚维护目标语义。它能减少昂贵模型的串行调用，收益仍取决于接受长度、验证负载和实际状态成本。下一章继续比较改变模型状态表示与外部交互方式的其他架构。

第 14 章　架构与外部世界：状态、证据与动作

到这里，我们一直以显式 K/V 历史解释模型的记忆。但模型也可以把历史压进递归状态，把图像变成可读取的证据，或在文本生成之外发起工具动作。本章只保留理解这些接口所需的机制：它们分别改变什么状态、增加什么信息，以及误差如何传播。详细训练配方与完整 Agent 工程留给相应主题分册。

14.1　从显式历史到有限维状态

Attention 保存逐 token 的 key/value，新的 query 按内容读取历史；长度增加时，这份显式状态通常随之增加。递归或状态空间模型（state-space model，SSM）则把历史逐步写入固定维状态。固定状态有利于控制推理的历史存储，却必须在有限表示中保存未来任务需要的信息，不能仅由状态大小推断检索质量。

沿用本书的行向量约定，设输入 $x(t) \in \mathbb{R}^{1\times d_x}$、状态 $h(t) \in \mathbb{R}^{1\times n}$、输出 $y(t) \in \mathbb{R}^{1\times d_y}$。连续线性系统可写为

$$\dot{h}(t) = h(t)A + x(t)B, \qquad y(t) = h(t)C + x(t)D, \tag{14.1}$$

其中 $A \in \mathbb{R}^{n\times n}$、$B \in \mathbb{R}^{d_x\times n}$、$C \in \mathbb{R}^{n\times d_y}$、$D \in \mathbb{R}^{d_x\times d_y}$。离散化与离散步长有关；用 $\overline{A}, \overline{B}$ 表示离散后的矩阵，得到

$$h_t = h_{t-1}\overline{A} + x_t\overline{B}, \qquad y_t = h_tC + x_tD. \tag{14.2}$$

这里先把状态展平成一行，实际架构常有逐通道状态、门控、卷积与更多投影，不能把这个抽象直接当成完整模型配置。

当矩阵与时间无关时，展开递推可得

$$h_t = h_0\overline{A}^t + \sum_{i=1}^{t} x_i\overline{BA}^{t-i}. \tag{14.3}$$

忽略初始状态项后，输出可写成因果卷积；零时滞核为 $K_0 = \overline{B}C + D$，其余为 $K_j = \overline{BA}^jC$（$j \geq 1$），其中 D 保留了输入的直通分支。因此训练时存在并行计算途径。Mamba 等选择性 SSM 让部分离散化/写入/读取参数依赖输入，使状态更新具有选择性 [39]。此时系数随 token 改变，不再是一份固定卷积核；并行 scan 等实现依靠递推结构，而非假装系统仍然时不变。

单步 recurrent 推理在固定层数与状态维度下，历史状态容量可不随已见长度增加；局部卷积缓冲、输出日志及外部记忆仍须另外记账。混合架构中的 attention 层也可能保留增长的 KV，不能因模型含 SSM 就宣称整个系统具有恒定内存。如何在精确内容检索与有限状态压缩之间取舍，取决于架构、训练数据和任务证据。

Mamba-2 的结构化状态空间对偶（SSD）进一步连接了特定 SSM 与结构化掩码注意力 [40]。其核心选择把状态转移的结构简化为标量乘单位矩阵，并利用半可分矩阵的分块分解组织运算。递推形式与二次形式可描述同一个受约束的层，分块算法再在两种计算之间取舍；这没有证明任意 softmax Transformer 都能直接替换为相同质量的 SSM。论文中的算法速度也有指定实现与对照，本书不把它当作所有任务的端到端加速保证。

14.2 递归误差：渐近稳定不等于短时不放大

先在固定的线性递推中，只考虑每步额外注入的状态误差 e_t。令 $A_d = \overline{A}$，有

$$\delta h_t = \delta h_{t-1} A_d + e_t = \delta h_0 A_d^t + \sum_{i=1}^{t} e_i A_d^{t-i}. \tag{14.4}$$

量化矩阵、输入与门控带来的变化也会进入误差项；它们未必独立、零均值或与状态无关。此式说明过去误差会参与后续状态更新，与一次写入后重复读取的固定 KV 具有不同传播路径。

对有限维常系数系统，谱半径 $\rho(A_d) < 1$ 保证 A_d^t 渐近趋于零，但不保证每一步范数都减小。考虑非正规矩阵

$$A_d = \begin{bmatrix} 1/2 & a \\ 0 & 1/2 \end{bmatrix}, \qquad A_d^t = \begin{bmatrix} 2^{-t} & t a\, 2^{-(t-1)} \\ 0 & 2^{-t} \end{bmatrix}. \tag{14.5}$$

两个特征值都为 1/2。若 $a = 10$、$\delta h_0 = [1, 0]$，一步后变为 $[1/2, 10]$，欧氏范数已明显增大。随后会衰减，但短期放大足以影响输出。只有例如一致的算子范数约束 $\|A_d\| \le \alpha < 1$，才可直接得到几何型上界

$$\|\delta h_t\| \le \alpha^t \|\delta h_0\| + \sum_{i=1}^{t} \alpha^{t-i} \|e_i\|. \tag{14.6}$$

即使这个上界成立，持续误差也一般留下非零误差底数。

选择性系统需用随时间变化的 A_t。在两条路径使用同一组系数的局部比较中，

$$\delta h_t = \delta h_0 A_1 A_2 \cdots A_t + \sum_{i=1}^{t} e_i A_{i+1} A_{i+2} \cdots A_t, \tag{14.7}$$

空乘积为单位矩阵。顺序不能交换，单个 A_t 的谱半径也不足以控制任意有序积；若扰动改变了选择门控，系数变化还要纳入模型比较。因此状态量化应检查长 horizon 的漂移、短时最大放大与真实输出质量，不能仅凭静态权重 MSE 判断。

14.3 视觉证据怎样进入语言计算

典型视觉语言模型先由视觉编码器把图像变成特征，再由投影器或 resampler 映射到语言模型可读取的表示。设图像为 $H \times W$、方形 patch 边长为 p_v，当两边均整除时，基础 patch 数为

$$T_v = (H/p_v)(W/p_v). \tag{14.8}$$

分类 token、任意分辨率切块、边界填充与视频多帧都会改变实际数量；压缩后的 latent 数也未必等于 T_v。

若视觉表示与文本直接拼接进入 self-attention，它们共同影响 prefill 长度与逐层 KV；若文本通过 cross-attention 查询独立视觉 memory，则需要分别记录 self-attention KV 与视觉缓存。Resampler 把多个 patch 特征映射成较少 latent，改变的是输入证据的表示与数量。三种接口不能只用“图像变成 token”一句话推导相同的成本公式。

代表性连接方式包括 BLIP-2 的 Querying Transformer：它在冻结视觉编码器与冻结语言模型之间学习接口；LLaVA 则以视觉指令数据连接视觉编码器与语言模型 [41, 42]。它们说明接口和训练数据都是设计对象，不是仅把像素改成另一个 ID。具体模块、冻结阶段和输入组织应按所选版本核对，不能用某一工作的一组评测分数概括所有视觉语言系统。

视觉 token 筛选/合并发生在输入表示上，KV 量化发生在层内状态上，权重量化又是第三个位置。误删关键字符 patch 后，即使剩余 KV 完全精确也无法恢复那份证据。因而压缩评测除通用问答

外，应覆盖 OCR、计数、空间关系等依赖细节的任务，并做纯文本回归。具体模块与校准数据必须记录，不能把“只压缩语言权重”概括为压缩了全部多模态系统。

视频可把帧数与每帧 patch 数相乘作最初账本，但帧选择、时空编码与跨帧压缩会改变它。需要研究显式三维场景表示的读者可转向本系列《NeRF 与 3D 高斯泼溅》；本书只解释这些证据进入语言模型后的张量与状态开销，不预设特定三维接口已经训练成功。

14.4 检索增强：潜在文档与拼接证据是不同建模方式

检索增强生成（RAG）把外部文档作为生成条件。原始 RAG 的一种序列级抽象把一个检索文档 d 视为潜变量：

$$p(y \mid x) = \sum_{d} p_\eta(d \mid x) \prod_{t} p_\theta(y_t \mid x, d, y_{<t}). \tag{14.9}$$

它对文档条件下的序列概率求混合；实际检索往往只保留有限候选，是否对候选权重重新归一化要写清楚。原论文也区分 token 级边缘化等方案 [43]，不能将上述形式当成所有 RAG 实现的定义。

常见工程管线则先得到 top-k 集合 $\mathcal{D}_k(x)$，按固定策略拼成证据串 $c = \mathrm{Pack}(\mathcal{D}_k(x))$，再计算 $p_\theta(y \mid x, c)$。这里的生成器同时看到若干片段，并未自动计算式 (14.9) 的潜文档混合。检索分数也不能直接解释成答案正确概率。

无论采用哪种形式，错误都可能来自语料缺失/过期、漏召回、错误排序、打包截断、模型误读或引用与答案不对应。诊断时要分别观察证据是否存在、是否被送入模型、以及是否支持答案；只换更大模型无法定位这些来源。检索也不等于把远距原文完整保存在 KV：前者决定未来取回什么，后者是当前计算已建立的内部状态。

14.5 工具与 Agent：从概率输出到环境闭环

工具调用通常先由模型产生结构化名称与参数，再由运行时校验、执行，并把观察结果返回模型。函数名 token 的高概率并不能保证参数有效、工具成功或返回内容可信。超时、幂等与重试属于运行时语义；外部文档和工具返回内容是数据，不能仅因进入上下文就获得系统指令的权限。

Agent 在多步循环中组合观察、决策、动作、反馈与记忆。其成功率因此是模型、工具集合、环境、终止规则和恢复策略共同作用的结果。量化若改变一次接近决策边界的工具选择，后续轨迹可能分叉；但语言表面差异也未必改变实际动作。因此比较应固定环境版本与种子，保存结构化动作/观察日志，统计多条轨迹的任务结果、工具错误、步骤与成本，不能从一次生成或 token PPL 推断闭环可靠性。

ReAct 把文本推理轨迹与任务动作交错生成，并将外部观察带回后续条件中 [44]。这是组织交互的一种方法；显示出的推理文本并不自动构成模型内部计算的完整因果解释，动作成功也不能只由语言连贯性判定。

本系列《LLM 智能体系统》进一步讨论工具、检索、记忆和多智能体组织；《智能体模型量化压缩与部署》关注轨迹与状态压缩后的部署验证；训练配方可衔接《现代小模型训练》，集群服务成本则对应《大模型推理加速与 AI 基础设施》。这些阅读方向扩展应用深度。三种误差传播的比较练习见附录 B.7 节；下一章再把局部数值变化和最终任务结果连接起来。

第 15 章　评测与可复现研究

前面把一个推理结果拆成参数、计算与状态。本章再问：改动是否保持了所需能力，节省的资源是否转化为真实收益，别人能否在相同条件下得到同一结论？这三个问题分别需要质量协议、系统测量和完整记录。一个局部 MSE、一张排行榜或一次最快计时，都不能单独回答它们。

15.1　先固定任务与被比较的概率

15.1.1　困惑度的分母也是协议的一部分

对有效预测位置集合 $\mathcal{I}$，令

$$\ell = -\frac{1}{|\mathcal{I}|}\sum_{t\in\mathcal{I}} \log p_\theta(x_t \mid x_{<t}), \qquad \mathrm{PPL} = \exp(\ell). \tag{15.1}$$

这里用模型原始条件分布和自然对数；温度、top-k/top-p 截断后的采样分布通常不用于计算基准 PPL。若 $|\mathcal{I}| = 0$，结果未定义，程序应报错或跳过并记录，不能把分母改为一后报告“完美困惑度”。

比较 PPL 必须固定文本、tokenizer、特殊 token、上下文长度、窗口步长、文档边界和 loss mask。不同分词器的每 token PPL 不直接可比：同一句话被切成更多 token 后，平均项已经改变。即使分词器相同，每段是否重置上下文、滑窗是否重复计入目标 token，也会改变结果。报告有效 token 总数，并先求总 NLL 再按总 token 数归一化，避免把长短不同文档的平均 loss 无意赋予相同权重。

同协议下 PPL 上升表示平均 NLL 变差；它不保证每一种问答、代码或安全行为都变差。反之，PPL 几乎不变也不保证某个低 margin 的离散决策保持不变。第 8 章的 logit 扰动条件正说明了局部均值与具体翻转之间的区别。

15.1.2　任务分数需要可执行的判分规则

多项选择题要记录答案 token 的打分方式、是否按答案长度归一化、选项顺序与提示模板。生成题要固定抽样参数、最大输出长度、停止词、答案提取器和失败处理。代码题还需记录运行环境、测试集、超时与资源限制。格式错误、空回答和超时应进入预先定义的统计口径；只保留成功输出会产生选择偏差。

HELM 把准确性、鲁棒性、校准和效率等维度放进统一评测框架 [45]。本书采用这个原则，但不把某一组任务的综合分数解释为所有应用的通用能力。实际部署应先定义工作负载：文档问答关注证据与答案，工具调用关注参数合法性和任务完成率，代码生成关注可执行正确性。

15.2　系统指标：从一条请求的时间线开始

设请求在边界处被接收的时刻为 a，第 j 个输出 *token* 到达相同测量边界的时刻为 t_j，共输出 n 个 token。则

$$\mathrm{TTFT} = t_1 - a, \tag{15.2}$$

$$\mathrm{ITL}_j = t_j - t_{j-1}, \quad j = 2, \dots, n, \tag{15.3}$$

$$\mathrm{TPOT} = \frac{t_n - t_1}{n-1}, \quad n > 1. \tag{15.4}$$

TTFT 是首 token 延迟，ITL 是相邻输出 token 间隔，TPOT 是单请求首 token 之后的平均每 token 时间。只有一个 token 时 TPOT 未定义。到最后输出 token 的延迟满足 $t_n - a = \mathrm{TTFT} + (n-1)\mathrm{TPOT}$；连接关闭、后处理或 EOS 的额外时间要另作定义。若一个流式消息包含多个 token，只记录消息到达时间便不能恢复真实的逐 token ITL，应报告消息粒度而不制造 token 时间戳。

测量边界决定包含哪些成本。客户端 TTFT 可以包含网络、排队、分词和 prefill；服务端核心计时可以从调度开始。两者都可用，但必须命名清楚。只测算子时间无法发现排队或网络造成的变化。

以下以有限的请求到达批次为统计集合：Δ 从首条请求被接收开始，到该批所有请求完成或超时为止，包含排空时间；n_i 是第 i 条请求实际产生的输出 token 数，也包括失败前的部分输出。输出吞吐可定义为 $\mathrm{TPS}_{\mathrm{out}} = \sum_i n_i/\Delta$。若采用稳态截窗，则应只计窗口内实际到达的输出 token，并另报跨窗请求的处理方式。输入 token/s、输出 token/s 和完成请求/s 是不同量。若预先给定每请求的延迟与正确性约束 C_i，一种服务有效吞吐（goodput）定义为

$$\mathrm{Goodput}_{\mathrm{out}} = \frac{\sum_i n_i \mathbf{1}\{C_i\}}{\Delta}. \tag{15.5}$$

C_i 可以是 TTFT 与 TPOT 均低于阈值且结果有效，也可以采用更严格的尾部 ITL 限制；必须在实验前固定。对 $n_i = 1$ 的有效请求，本例只检查 TTFT，将 TPOT 约束标为不适用；空输出或失败请求不计有效 token。失败和未完成请求仍需报告数量、等待时间与超时，不能从到达数中悄悄删除。终止观察时尚未完成的请求是被删失数据，不能当作零延迟。观察区间的起止与排空策略也应固定。

表 15.1 系统比较必须同时固定的条件

条件	最小记录内容
工作负载	输入/输出长度分布、并发或到达率、前缀重复率、任务与采样参数
软件路径	模型/tokenizer revision、引擎与 kernel 版本、dtype、量化格式、图捕获和分页设置
硬件路径	加速卡型号/数量、显存、功率或频率设置、互联和主机环境
测量边界	客户端或服务端、排队/网络是否计入、预热与观察区间、同步方法
结果与失败	分位延迟、吞吐、峰值显存、拒绝/超时/OOM、正确性和重复轮次

15.2.1 负载与计时方法会改变结论

闭环负载固定并发：一条请求完成后再补下一条。系统变慢时，到达率也自动下降。开环负载按外部时间表发请求，能暴露队列积累和过载，但需记录实际发送时间，防止压测客户端自身成为瓶颈。因此“并发 32”与“每秒到达 32 条”不是同一种条件。报告 p50/p95/p99 时，还要说明统计对象是请求 TTFT、请求平均 TPOT，还是汇总的所有 token 间隔；不同加权方式不能混用。

首次运行可能包含模型加载、编译、自动调优和图捕获。冷启动与稳态应分别测量。GPU 操作通常异步提交：只在 CPU 上包围一次调用，可能只测到了提交时间。单算子可用设备事件或受控同步计时，端到端则按完整响应边界计时；不要在每个算子后插入同步再声称测得自然流水线性能。预热后运行多个独立轮次，记录波动；真实服务测试还需保留队列与失败，不挑选最快一轮。

15.3 污染、不确定性与生成判分

公开 benchmark 可能进入训练数据，也可能因反复调参而成为事实上的开发集。时间切分、独立 held-out、程序生成变体和真实业务题能提供互补证据，但简单的字符串去重只能发现部分污染；不能据此证明语义近似题或答案从未出现。应记录数据版本、切分、污染检查方法及其局限，并把

校准集、开发集和最终评测集分开。

对同一组 m 个问题，比较两个方法应尽量保留配对关系。令每题得分差为 d_i，报告 $\bar{d} = m^{-1}\sum_i d_i$，可按问题重采样计算配对 bootstrap 区间。若同一问题有多次生成或多个相关子题，应按问题或数据来源分组重采样，不能把相关输出当成更多独立样本。区间描述所选样本与抽样过程的不确定性，不替代跨数据域、模板或模型版本的验证。

开放生成没有唯一正确字符串时，可用人工偏好、参考指标、模型评审或可执行验证器。模型评审（LLM-as-a-judge）的系统性偏差已有专门研究 [46]。至少要交换回答顺序、隐藏方法身份、固定 judge 模型与提示、抽样人工复核，并报告平局及无法判断的处理。更长的回答可能更容易获得高分；控制长度或单列长度结果有助于判断收益来自内容还是表达。验证器也有覆盖边界：通过有限单元测试不等于程序对所有输入正确。

15.4 校准与解释：明确被预测的事件

对有正确标签的分类任务，设第 i 题预测置信度为 c_i，正确性为 $a_i \in \{0,1\}$，分桶集合为 $B_1, \ldots, B_M$。常见的期望校准误差为

$$\mathrm{ECE} = \sum_{b=1}^{M} \frac{|B_b|}{m} \left| \frac{\sum_{i\in B_b} a_i}{|B_b|} - \frac{\sum_{i\in B_b} c_i}{|B_b|} \right|. \tag{15.6}$$

空桶不参与求和。手算例：设五题平均置信度 0.9、答对三题，另五题平均置信度 0.6、答对四题，按这两个桶算得 ECE 为 $\frac{1}{2}|0.6-0.9| + \frac{1}{2}|0.8-0.6| = 0.25$。

ECE 依赖分桶方式与样本量，也会掩盖不同子群内方向相反的偏差。生成任务首先要定义“正确”事件：下一个 token、最终事实陈述、整段程序还是工具任务完成。低 token entropy 表示分布集中，不能自动作为答案正确概率；多次采样一致也可能一致地错。校准分析应同时给出事件、置信度构造、可靠性曲线和适当评分规则，如 NLL；不能仅凭单个 ECE 宣称模型可信。

可解释性也有不同证据层级。线性 probe 说明信息可被读出，注意力图或 attribution 提供相关线索；activation patching 则通过替换激活检验某个计算路径对所选任务的作用。干预本身可能造成分布外激活，因此需要干净/受损配对样本、随机或无关位置控制、不同替换方式以及任务级复核。“存在一个可读方向”不等于模型实际依赖它，局部干预效果也不自动推广到所有输入。

量化可作为结构化干预来寻找敏感层或头，但它会同时改变多个方向、尺度及舍入边界。若要主张某特征导致行为变化，至少要与匹配输出 MSE 的随机扰动、相同预算的消融和定向 patching 对照。第 8 章的 margin 不等式只是给定 logit 扰动下的数学充分条件，并非“安全行为总处于狭窄 margin”的实证证据。

15.5 安全行为需要在最终系统上重新验证

拒答、事实正确性、工具权限与资源隔离属于不同层次。RAG 文档和工具返回内容是外部输入，不能靠“内部来源”这个标签获得更高指令权限。完整任务要检查数据治理、提示注入边界、工具参数验证、沙箱、秘密隔离与运行监测。

压缩、微调、采样或提示模板改变后，应在最终配置上重新测试所需安全行为。可能改变拒答或工具选择，是需要检验的假设；没有测量就不能声称量化必然削弱或增强安全。评测要固定攻击集合、普通请求集合、攻击者预算和成功判据，同时报告越界成功率与正常任务误拒率。多轮或自适应攻击还需把整条会话视为样本，避免把同一条轨迹中的尝试当作独立证据。

15.6 把论文主张变成可证伪实验

读论文时，先把方法还原为一个带条件的句子：针对什么对象、在哪个阶段、改变了哪个张量或状态、依赖什么统计结构、额外花了多少校准和运行成本、用什么对照检验。随后逐张量写 shape、dtype、分组轴和生命周期，将离线变换、校准、打包与在线 kernel 分开。只读配置名，往往看不见实际执行路径。

以 KV 向量量化为例，“VQ 会更好”没有规定比较方式。一个可检验假设是：在相同实际缓存字节、相同上下文/任务和相近实现预算下，利用 V 向量相关性的码本能降低输出误差；若相关性被打散，收益应下降。这里输出误差、任务分数、码本开销和解码时间必须共同记录。即使质量假设成立，真实 kernel 仍可能因为索引查表或不规则访问而更慢。

1. **先过机制检查**。用合成矩阵检验格点、pack/unpack、零值、尾组与独立参考；再到单层和完整 block，观察残差及非线性的耦合。
2. **再过模型检查**。固定校准数据与超参预算，比较 PPL、任务分数、长度和领域，保留失败结果，不根据最终测试反复选阈值。
3. **最后过系统检查**。在真实表示与 kernel 上测字节、峰值显存、时间及服务指标，明确预热、负载、硬件和软件版本。

若组合 VQ、旋转与低秩补偿，至少要有逐模块与关键组合的消融，并把码本、scale、索引、低秩分支和未量化层全部计入预算。比较“相同名义 bit”却多放一条高精度支路，会把容量收益误当成算法收益。校准 token 数、搜索轮次与离线量化时间同样属于成本。

误差诊断可沿层级记录：权重重构 MSE/SQNR、层或 block 输出误差、注意力分布变化、logit KL 与 margin、最终任务分数；系统侧再对照数据移动量和时间。相关趋势只是寻找机制的线索，应主动测试更长序列、不同数据域、GQA/MLA 及小 batch 等失败边界。条件刻画清楚的负面结果，也能帮助判断方法的适用范围。

一次可复现比较应保存什么

保存代码 revision、环境、模型与 tokenizer revision、chat template、数据切分及校准索引、全部种子、量化轴/组长/舍入/零点、scale 或码本哈希、命令与原始结果。保存失败请求和实际到达记录；若隐私或许可限制不能发布原数据，应说明限制并提供可公开的最小复现数据。报告未测试的硬件与场景，不把它们写成已验证结论。

15.7 可选的十二周实践路线

这条路线服务于学习，不要求每位读者训练大模型。第 16 章的 CPU 实验提供机制层的起点；是否进入 GPU 内核或真实模型阶段，应按可用资源选择。

表 15.2 围绕可验证产物推进的实践路线

周	核心问题	实践内容	可审阅产物
1–2	张量与条件分布	手算 linear、softmax、RMSNorm；实现 MHA/GQA、mask 与 RoPE	独立参考和形状记录
3–4	block 与量化格点	核验残差/SwiGLU，比较逐张量/分组对称与仿射量化	block 输出及误差表

周	核心问题	实践内容	可审阅产物
5–6	输入敏感的压缩	重建 GPTQ/AWQ/SmoothQuant 的目标与等价式，固定校准预算	单层/单块消融
7–8	状态与长上下文	实现多层 cache，估算 KV；改变长度、头数和残余窗口	等价检查及内存账本
9–10	从字节到系统时间	按资源选眞实 kernel 或可执行测量方案，区分 prefill/decode	带条件的性能报告
11–12	可证伪的小课题	在独立任务或架构上测试一个机制假设，记录失败并整理材料	方法、证据与局限说明

为一张结果表补齐证据

假设某方法只报告“W4、吞吐提升两倍、PPL 基本不变”。写出至少六项仍需确认的条件，并指出每项影响质量、公平预算或测量中的哪一环。再设计一个同时包含质量约束和 TTFT/TPOT 约束的 goodput 定义，说明失败请求怎样处理。

本章联系

质量结论依赖数据与概率协议，速度结论依赖负载与计时边界，研究结论依赖对照与可复现记录。先把这些条件固定，再讨论一个方法是否有效，以及有效到什么范围。

第四部分

让机制接受独立检验

第 16 章　CPU 实验工作坊：把形状、缓存与量化跑起来

本章给出不依赖模型下载或 GPU 的可执行实验。代码文件是 `manuscript/experiments/run_cpu.py`，固定手算输入在同目录的 `hand_block_case.json`。实验使用合成数据与 float64，检验形状、位置、状态递推和误差定义。它不测量实模型的任务质量，也不提供低比特 kernel 的速度结论。

16.1　运行、配置与结果追踪

在本主题根目录运行：

```
python3 manuscript/experiments/run_cpu.py \
  --output output/experiments > output/experiments/run.log 2>&1
```

需要可用的 Python 3 与 NumPy。本次实际解释器为 `/opt/anaconda3/bin/python`，Python 3.12.7；完整 NumPy 与平台版本保存在结果文件。命令中的重定向目录须已存在，首次运行可先执行 `mkdir -p output/experiments`。脚本也可不重定向，以便直接观察失败。

主随机种子为 20261021，新增多层实验使用相同种子的独立随机流，避免改变既有算例。脚本把真实解释器与参数、工作目录、上海时区时间、版本、配置、代码与手算 fixture 的 SHA-256、逐项误差与检查状态保存到 `output/experiments/results.json`，并生成正文使用的 `manuscript/experiments/results.tex`；输出目录保留一份相同表格。固定 fixture 的输出容差从 JSON 配置读取，为 10^{-10}，用于容纳手算十进制记录；向量化与独立标量参考、缓存等价及 RoPE 恒等式采用绝对容差 10^{-12}。

表 16.1　本次 CPU 实验的三种前向配置

实验	明确配置
注意力与位置	$B=2, T=7, h_q=4, h_{kv}=2, d_h=6$；分块 $[3,2,1,1]$，RoPE base 10000，无 dropout
手算完整 block	$B=1, T=2, d=4, h_q=h_{kv}=2$；固定小矩阵，norm 的 $\epsilon=0$、非零输入，关闭 RoPE，与第 3 章完全对应
三层完整 decoder	$B=2, T=9, L=3, d=16, d_{\mathrm{ff}}=24, h_q=4, h_{kv}=2, d_h=4$；分块 $[4,2,1,1,1]$，$\epsilon=10^{-6}$，非单位 norm 参数，RoPE 开启

16.2　先验证注意力、头映射与位置

Q 为 $[B, h_q, T_q, d_h]$，K/V 为 $[B, h_{kv}, S, d_h]$。教学实现通过重复 K/V 头形成显式参考；这有利于看清映射，但不是推荐的高效 GQA 内核。独立标量实现则用 $\lfloor h/(h_q/h_{kv}) \rfloor$ 直接选择 KV head，并逐 query–key 点积、求和，避免两个实现共享同一头广播错误。

以下是随书脚本实际使用的 RoPE 函数。相邻偶奇特征组成一个平面，`positions` 是全局逻辑位置；形状扩展把相同位置角度广播到 batch 与 head：

```
def rope(x, positions, base=10000.0):
    # x: [B,H,T,D]; adjacent even/odd channels form each rotation plane.
    if x.shape[-1] % 2:
        raise ValueError('RoPE head dimension must be even in this example')
    freq = base ** (-np.arange(0, x.shape[-1], 2)/x.shape[-1])
    angles = np.asarray(positions)[:, None]*freq[None, :]
    c, s = np.cos(angles)[None,None,:,:], np.sin(angles)[None,None,:,:]
    out = np.empty_like(x)
    out[...,0::2] = x[...,0::2]*c-x[...,1::2]*s
    out[...,1::2] = x[...,0::2]*s+x[...,1::2]*c
    return out
```

本例只实现相邻配对布局；有些模型采用前后半区配对，转换权重或 cache 时必须连同布局核对。检查包括：相对位置点积恒等式、旋转保持范数、因果概率和为一、未来位置概率严格为零，以及修改未来 V 不改变过去输出。故意把新 query 的 RoPE 位置重置为零会产生可检测误差；这是一项反例检查，证明测试能够发现位置错误。头数不能整除与全 key 被 mask 的 query 都显式报错，避免把 NaN 隐藏为一个正常输出。

16.3 从预生成 QKV 到真正的多层缓存

第一组 cache 实验预先生成 Q/K/V，比较整段因果计算与分块 prefill 加逐 token decode。这验证了状态拼接与 mask，但没有覆盖投影、norm、FFN 和层间耦合。因此第二组实验从原始输入开始，经过三层 RMSNorm、GQA、RoPE、输出投影、SwiGLU 与两条残差；每一层仅保存自己的旋转后 K 与未旋转 V。

缓存路径每块都从自己的隐状态重新计算当前层投影和 FFN，不能读取全量路径的中间结果。全量路径、分块路径和标量注意力路径分别运行，再比较最终输出、每层输出及每层 K/V。每层最终 cache 都是 $[2, 2, 9, 4]$，总长度由全局位置决定。另将原始输入第 5 个 token 起全部改变，验证经过三层后的前 4 个位置输出仍在容差内不变，本次实测误差为零，覆盖因果性在完整网络中的传播。

第 3 章的手算 fixture 则独立保存固定权重和期望输出，核对双头拼接、W_O、第一次残差、第二次 norm 和 FFN。它与随机三层实验互补：前者能定位漏掉一个分支的具体数值，后者能发现分块状态和跨层传递错误。数值对照见表 16.2。

表 16.2 CPU float64 独立参考对照（2026-10-22；绝对误差）

检查	最大绝对误差
GQA 与标量参考	4.441e-16
预生成 QKV 的完整与增量缓存	4.441e-16
三层完整 decoder 的完整与增量缓存	1.554e-15
三层完整 decoder 与标量注意力参考	2.665e-15
RoPE 相对位置恒等式	8.882e-16
完整 block 与手算	0.000e+00

表 16.3 同一组权重的 4-bit 伪量化实测；分组沿输入轴，组长为 3

配置	权重 MSE	相对权重误差	相对输出误差
对称逐张量	0.048355	0.123491	0.154699
对称分组	0.000075	0.004854	0.005591
仿射逐张量	0.038067	0.109570	0.133401
仿射分组	0.000024	0.002763	0.002973

16.4 同一权重上的四种伪量化

权重矩阵固定为 $[7, 4]$，包含全零列、常量列和一个值为 9 的异常值；输入矩阵为 $[11, 7]$。四种配置均使用四位编码：逐张量/输入轴分组，各自配对称/仿射量化。组长为 3，最后一组只有一行，因此也检验尾组处理。对称码域为 $[-7, 7]$，仿射码域为 $[0, 15]$，舍入采用 ties-to-even。

脚本中的单组核心为：

```
def quantize_block(x,bits,mode):
    if bits < 2: raise ValueError('This teaching implementation requires bits >= 2')
    if mode=='symmetric':
        qmax=2**(bits-1)-1
        a=float(np.max(np.abs(x)));scale=a/qmax if a else 1.0;zero=0
        q=np.clip(np.rint(x/scale),-qmax,qmax)
    elif mode=='affine':
        qmax=2**bits-1
        lo=min(float(np.min(x)),0.0);hi=max(float(np.max(x)),0.0)
        scale=(hi-lo)/qmax if hi>lo else 1.0
        zero=int(np.clip(np.rint(-lo/scale),0,qmax))
        q=np.clip(np.rint(x/scale)+zero,0,qmax)
    else: raise ValueError(mode)
    return (q-zero)*scale,dict(scale=scale,zero_point=zero)
```

四种配置共享同一组 W 与 X；表 16.3 的相对输出误差是 $\|X(W-\widehat{W})\|_F/\|XW\|_F$，相对权重误差是 $\|W-\widehat{W}\|_F/\|W\|_F$。本例分组后误差更小，原因可从异常值不再控制所有组的步长看出；它不构成所有权重或任务上的单调保证。更细分组的尺度数量增加，本章比较的是数值误差，不能把它直接当作同存储预算的最优配置。

独立格点检查包括窄对称两位的 $[-1, 0, 1]$，以及仿射两位、$s = 1, z = 1$ 的 $[-1, 0, 1, 2]$；后者的期望整数编码直接写为 $[0, 1, 2, 3]$，不是再调用同一个量化函数生成参考。另测全零、正负常量、半格 ties 与非整组。浮点伪量化始终在 float64 上完成后续运算，没有 bit packing、低比特乘法或反量化内核测速。

16.5 参数、KV 与概率账本复算

脚本把第 11 章的虚构 GQA 模型逐矩阵列成清单，独立求和并与解析式比较：

$$N = 5\,801\,771\,008, \qquad M_W^{\mathrm{BF16}} = 11\,603\,542\,016 \text{ bytes}. \tag{16.1}$$

理想 W4、每 64 个权重一个 FP16 scale 时为 3 082 190 848 bytes，约 2.870514 GiB。每序列每 token 的 BF16 KV 是 128 KiB；$B = 8, S = 16384$ 时恰为 16 GiB。这些是实际执行脚本得到的解析整数复算，

不含设备分配器、页表、激活、工作区或峰值实测。不要把实验小矩阵的组长 3 与此处资源账本的组长 64 混为一项配置。

第 13 章的推测采样还提供一个分布检查：$p = [0.1, 0.6, 0.3]$、$q = [0.4, 0.2, 0.4]$，直接枚举接受质量与拒绝后的残差质量，重构 target 分布；再采样 50000 次，要求各频率差低于预设六倍标准差阈值。额外覆盖 $p = q$ 的全接受分支和 draft 零支持分支。频率通过是本次有限样本结果，数学上的精确性仍依赖第 13 章的证明。

16.6 哪些实验还没有运行

本次共通过 52 项检查。若要进一步回答“在真实模型上是否值得”，可执行以下方案；它们未在本次运行：

1. 固定公开小模型及 tokenizer revision、数据版本、校准切分和种子；先核对高精度基线的 NLL，再比较伪量化/真实量化的 PPL 与任务分数，保留所有失败与有效 token 分母。
2. 在支持目标格式的 GPU 上固定引擎、kernel、组长及未量化层，分别测加载后、prefill 和 decode 峰值；保存真实 packed 权重大小与 scale/offset，不能只用名义 bit 估算。
3. 使用预先声明的输入/输出长度分布与到达模式，预热并重复测 TTFT、ITL/TPOT、吞吐和 goodput，报告排队、同步边界及超时；按照第 15 章的协议检验质量与延迟约束是否同时成立。

无需等待这些扩展实验才能验证本书的 CPU 恒等式与缓存机制；也不能在尚未运行时把预期质量或吞吐填入结果表。

修改一个假设再运行

在账本配置中把 h_{kv} 从 8 减为 4，检查 KV 字节减半而参数只减少 K/V 投影；另把序列长度加倍，分别记录最终容量和累计 decode attention 工作量。再尝试只重置一层 cache 的逻辑位置，说明哪一项输出对照会失败。保留配置、命令与观察，不放宽容差掩盖错误。

本章联系

小实验把公式变成可失败的断言：先验证独立参考、边界和状态，再解释误差与资源。数值正确、真实表示与端到端收益是相互连接而各需证据的三个层次。

附录 A　符号、形状与公式速查

本附录沿用正文的行向量约定：线性层右乘权重，XW 的最后一轴为输出特征。张量的逻辑形状与框架实际存储顺序可以不同；对照代码时须同时核对转置、分组轴与 dtype。

A.1　一套符号，明确的单位

表 A.1　正文符号及其作用范围

符号	含义	约定与边界
N, D	参数个数、训练 token 数	两者都是计数，不是存储字节。SSM 方程中的矩阵 D 由所在局部定义。
B	同时活跃序列数	含已计入的分支；不再重复乘相同 beam 数。
T	当前计算块长度	可为完整 prefill，也可为一个 chunk 或单 token。
P, G, S	提示、生成、可见 KV 长度	生成计步边界按正文约定；S 不必等于当前块长 T。
V	词表大小	在注意力公式中 V 也表示 value 张量，依上下文区分。
L, d, d_{ff}	层数、残差宽度、FFN 中间宽度	SwiGLU 的 gate/up 均映射到 d_{ff}。
h_q, h_{kv}, d_h	Q 头数、KV 头数、每头宽度	本书标准配置 $d = h_q d_h$；GQA 要求 h_q/h_{kv} 为整数。
X, H	激活、隐状态	通常为 $[B, T, d]$，归一化沿最后一轴。
Q, K, V	queries、keys、values	当前 Q 为 $[B, h_q, T, d_h]$；完整 K/V cache 为 $[B, h_{kv}, S, d_h]$。
s, z	scale、整数 zero-point	若实现保存浮点 offset，应另说明参数与重构公式。
b_w, b_a, b_{kv}	W、A、KV 编码位数	单位均为 bit/element；换成 bytes 必须除以 8。
M_W, M_{KV}	权重、KV 字节容量	实际量化存储还含 scale、offset、对齐与其他状态。

1 GB $= 10^9$ bytes，1 GiB $= 2^{30}$ bytes，1 KiB $= 2^{10}$ bytes。GB/s 与 GiB/s 也要按同一体系换算。正文以 b_w, b_a, b_{kv} 专指编码位数，字节量以 bytes 明写。

A.2　行向量公式卡片

自回归模型分解与逐 token 损失见第 2 章：

$$p(x_{1:T}) = \prod_{t=1}^{T} p(x_t \mid x_{<t}), \qquad \mathcal{L} = -\frac{1}{n}\sum_t m_t \log p(x_t \mid x_{<t}), \quad n = \sum_t m_t. \tag{A.1}$$

注意 BOS、目标偏移及 $n = 0$ 的接口处理；损失 mask 与 attention mask 是两个对象。

对单个 head 的可见位置，$Q \in \mathbb{R}^{T\times d_h}$、$K \in \mathbb{R}^{S\times d_h}$、$V \in \mathbb{R}^{S\times d_v}$，

$$O = \mathrm{softmax}(QK^{\top}/\sqrt{d_h} + M)V \in \mathbb{R}^{T\times d_v}. \tag{A.2}$$

softmax 沿 key 轴；每个有效 query 至少有一个可见 key。GQA 在执行此式之前，先确定每个 query head 对应哪个 KV head，不是直接让不匹配的头轴相乘。

对 $x \in \mathbb{R}^{1\times d}$ 与 $U \in \mathbb{R}^{T\times d}$，

$$\mathrm{RMSNorm}(x) = \frac{x}{\sqrt{d^{-1}\sum_j x_j^2 + \epsilon}} \odot \gamma, \tag{A.3}$$

$$\mathrm{SwiGLU}(U) = [\mathrm{SiLU}(UW_g) \odot (UW_u)]W_d. \tag{A.4}$$

$W_g, W_u \in \mathbb{R}^{d\times d_{\mathrm{ff}}}$、$W_d \in \mathbb{R}^{d_{\mathrm{ff}}\times d}$；不得把列向量写法 $W_g x$ 原封不动放进这一约定。

仿射量化明确编码域后为

$$q(x) = \mathrm{clip}(\mathrm{round}(x/s) + z, q_{\min}, q_{\max}), \qquad \hat{x} = s(q(x) - z). \tag{A.5}$$

q 是离散码，$\hat{x}$ 是重构实数；舍入规则、共享轴与范围选择见第 6 章。

LoRA 在 $W \in \mathbb{R}^{d_{\mathrm{in}}\times d_{\mathrm{out}}}$ 上的行向量写法为

$$W' = W + sAB, \quad A \in \mathbb{R}^{d_{\mathrm{in}}\times r}, \quad B \in \mathbb{R}^{r\times d_{\mathrm{out}}}, \qquad XW' = XW + s(XA)B. \tag{A.6}$$

s 是适配器缩放（此处并非量化 scale），更新秩至多为 r。使用不同权重朝向的库可能写作 BA，但必须连同两因子的形状一起转换。冻结基底不等于不存训练激活；低比特基底也不定义所有乘法与累积的精度。

A.3 一眼判断精度标签还缺哪些信息

看到 W4A8KV4 时，应查明 W/A/K/V 分别使用整数、浮点格式还是码本索引；每个对象的分组轴、组长、scale/offset dtype 与未量化层；乘法输入、内部转换及累积精度；是否真正打包、使用哪个 kernel、是否存在高精度临时副本。伪量化只能验证数值行为，不能据此报告低比特存储量或端到端速度。

附录 B　账本使用与练习提示

B.1　资源估算以同一份矩阵清单为准

参数与 KV 的完整账本、虚构 GQA 模型配置、GB/GiB 换算和峰值边界统一见第 11 章；可执行复算见第 16 章。把同一份配置同时用于公式、代码与误差比较，才能确认差异来自方法，而非模型形状或精度设置发生变化。

使用账本时，先按实际矩阵形状数参数并确认 embedding/head 是否共享；再按物理缓存数量、每请求可见长度与表示精度求和；最后另加激活、工作区、运行时与分配余量。各自峰值若不同时发生，不能机械相加成精确峰值。带宽下界只对指定的数据量与带宽假设成立，也不能当作一次生成测得的延迟。

B.2　第 1 章：形状、损失与误差

$[2,3,5]$ 输入右乘 $[5,7]$ 权重得到 $[2,3,7]$。按一次乘加算两 FLOPs，工作量约为 $2\times2\times3\times5\times7=420$ FLOPs；$[2,3,1]$ 的量沿最后特征轴广播。对 logits $[1002,1000,999]$，减最大值后为 $[0,-2,-3]$，第三类的负对数概率为 $3+\log(1+e^{-2}+e^{-3})$。

两条序列含 2 与 8 个有效目标时，先求各自平均再平均赋予两条序列相同权重；汇总全部有效目标则按 2:8 加权。比较训练与评测损失必须记录这个分母。$X=[1,1]$ 对误差方向 $[0.1,-0.1]^\top$ 与 $[0.1,0.1]^\top$ 分别产生 0 与 0.2 的输出变化，说明权重 MSE 不定义输入相关误差。

B.3　第 2 章：输入输出协议

输入 $[\mathrm{BOS},a,b]$、目标 $[a,b,\mathrm{EOS}]$ 只监督后两项时，损失 mask 为 $[0,1,1]$、有效数 $n=2$；这不要求禁止模型看到 a。$V=32000, d=4096$ 时，单个 embedding 矩阵含 131072000 个参数，共享输出头不再增加第二份参数；不共享则两份共 262144000 个。

对 logits $[0,\log 3]$，第二类概率在温度 1 与 2 时分别为 3/4 与 $\sqrt{3}/(1+\sqrt{3})$。温度为正时不改变最大值位置，却改变采样分布。top-p 刚好到达阈值时要检查累计与保留边界，不能仅以理想十进制心算预测每个库的行为。

B.4　第 3、4 章：头、位置与完整前向

若 $B=1, T=2048, h_q=32, d_h=128$，Q 为 $[1,32,2048,128]$。GQA 取 $h_{kv}=8$ 时，每个 KV head 服务 4 个 query heads；逻辑 score 仍有 32 个 query heads，而持久 K/V 不需要复制为 32 头。对完整 block，漏掉输出投影或把归一化后的 U 代替第一条残差 H' 加回去，都会改变最终函数。

手算 block 中若仅把 W_O 从 $I_4/2$ 改为 I_4，Q/K/V、分数与注意力概率不变；拼接后投影、第一次残差、第二次 norm、FFN 与最终输出均需重算。位置方面，缓存前缀长度为 S_0 的新块必须使用 S_0+i，不能用块内 i 代替全局位置。

B.5 第 6 章：窄对称格点与尾组

对 $[-0.9, -0.2, 0.4, 2]$ 采用三位窄对称区间 $[-3, 3]$，$s = 2/3$，最近邻整数编码为 $[-1, 0, 1, 3]$，重构为 $[-2/3, 0, 2/3, 2]$。更改负端点或舍入规则后，应重新按明示的量化器计算，而不是保留同一答案。

$W \in \mathbb{R}^{130\times16}$ 沿输入轴每 64 行分组时，每列有 3 个组，共 48 个尺度，尾组长度为 2；若按完整组填充，编码元素为 $192 \times 16 = 3072$。改为组长 32 后尺度更多，但模型质量仍取决于范围选择、元数据、校准目标与真实下游输入，不由尺度个数单独决定。

B.6 第 9、11 章：长度与物理容量

练习：沿用第 11 章配置，将 h_{kv} 从 8 改为 4，保持 h_q, d, d_{ff} 不变，计算参数量与 KV 各下降多少。再将生成长度翻倍，比较权重、最终 KV 容量与累计注意力 FLOPs 的变化。

$L = 80, h_{kv} = 8, d_h = 128, B = 1, S = 131072$，BF16 的 K/V 理想容量为 40 GiB。全部改为四位、每 64 个值仅存一个 FP16 scale，则有效位宽为 4.25 bit，理想容量为 10.625 GiB；这未含 offset、高精度尾部、对齐与页表。

在第 11 章的固定模型中，h_{kv} 从 8 减到 4 且其他维度不变，KV 裸元素数减半。参数只减少每层 K/V 两项：共减少 $2Ld(8-4)d_h$，Q/O、FFN 与词嵌入不变，所以总参数并不会减半。固定提示 P 后，最终缓存随 $P + G$ 线性增长，而累计 decode 注意力包含 $PG + G^2$ 量级项；两者要分别计算。

B.7 第 14 章：递归状态与动作

练习：比较一个 KV value 的固定量化误差、非正规递归矩阵中的状态误差与工具选择附近的 logit 扰动。为何它们不能用同一条“误差逐层相加”规则概括？

固定 KV 的一份误差会被未来 query 以不同权重读取；递归状态误差则通过有序状态矩阵乘积参与以后更新；离散工具选择若变化，环境与下一步输入也可能变化。三者共享“局部扰动可能影响未来”的直觉，却没有通用的同一条误差相加公式。尤其 $\rho(A) < 1$ 只说明常系数系统渐近衰减，非正规矩阵仍能短时放大。

B.8 第 15 章：一张速度表还缺什么

至少补查模型/tokenizer/template 与数据版本、PPL 的有效 token 分母；W4 的真实表示、分组轴/组长、scale 和补偿预算；硬件与 kernel 版本；输入/输出长度分布和到达模式；计时边界、预热与同步；失败、超时和重复轮次。例如令 C_i 表示输出有效且 TTFT 不超过阈值，$n_i > 1$ 时再要求 TPOT 不超过阈值；单 token 请求的 TPOT 标为不适用。按完整到达批次与排空时长计 goodput，失败不贡献有效 token，但仍保留在请求总数和失败率中。

附录 C　文献地图与术语

C.1　沿问题阅读原始工作

先读 Transformer[1]，逐项对应第 3 章的 Q/K/V、掩码与多头投影；再读 RoPE[8] 与 GQA[10]，区分位置几何与 KV 头共享。规模化部分可结合 GPT-3[2]、Scaling Laws[30] 与 Chinchilla[31]，追踪数据、目标与计算预算，不把某次拟合比例当成普遍常数。

系统层面，FlashAttention[36] 研究算子数据移动，PagedAttention[26] 研究缓存分配与共享；两者不能互相代替。量化主线可由 LLM.int8()[25] 认识异常值，再用 GPTQ[15]、AWQ[16] 与 SmoothQuant[17] 比较输入感知与等价尺度变换，最后读 QuaRot[19] 与 QLoRA[21]，分清旋转、表示格式与适配器训练。

缓存与替代状态可对照 KIVI[27]、StreamingLLM[28]、H2O[29] 与 Mamba[39]。它们分别改变数值精度、可见历史或状态表示，不能只按一个“压缩率”排名。每读一篇，记录对象、阶段、形状、核心等式、元数据、实际 kernel、评测条件和失败边界，便能把论文结果与本书可复算机制对应起来。

C.2　常用术语的作用边界

表 C.1　术语与使用条件

术语	含义
Activation	由输入与参数产生的中间张量；也可指非线性激活函数，依上下文区分。
Calibration	用代表性输入估计范围、敏感度或优化量化参数；校准集与最终测试集应区分。
Causal mask	按逻辑位置约束可见性，禁止有效 query 读取未来位置。
Channel	指定张量轴上的特征坐标；逐通道必须说明沿哪一轴归约。
Codebook	用于标量或向量重构的有限候选集合，其存储与查表成本也属账本。
Decode	已有前缀状态后产生后续位置的计算；缓存实现通常只计算新位置。
Effective bits	明示计入范围后，把元数据等摊入每真实元素的平均 bit 数。
Fake quantization	浮点模拟编码/重构误差；不包含真实打包、低比特 kernel 的速度证据。
GQA	若干 query heads 共享一个 KV head；逻辑注意力头数仍为 h_q。
KV Cache	各层历史 key/value 的持久推理状态，具有位置与模型版本约束。
Mixed precision	不同对象或存储、乘法、累积阶段使用不同精度；须说明各自 dtype。
Outlier	相对指定统计总体的异常幅值；阈值、轴与时间范围须明确。
Per-group	一组明确定义的元素共享量化参数；组长未必整除实际轴长。

术语	含义
PTQ	训练后量化；可包括校准与参数优化，并非必须完全无数据。
Prefill	处理一段已知输入并建立内部状态；可以整段或分块执行。
QAT	在训练中引入量化行为或其近似，使参数适应目标表示。
Residual buffer	在线 KV 量化中近期的高精度尾部；实际长度随算法/阶段变化。
Scale / zero-point	scale 决定格点尺度；整数 zero-point 是实数零对应的编码。浮点 offset 是另一种参数化。
TTFT / ITL	请求至首 token 的延迟、相邻输出 token 的间隔；计时边界和是否含排队须记录。
Vector quantization	用码本向量近似一组联合元素，需明确向量轴、索引位数与码本成本。

参考文献

[1] A. Vaswani, N. Shazeer, N. Parmar, et al. *Attention Is All You Need*. arXiv v7. June 12, 2017. arXiv: 1706.03762. URL: https://arxiv.org/abs/1706.03762v7 (visited on 10/22/2026).

[2] T. B. Brown, B. Mann, N. Ryder, et al. *Language Models are Few-Shot Learners*. arXiv v4. May 28, 2020. arXiv: 2005.14165. URL: https://arxiv.org/abs/2005.14165v4 (visited on 10/22/2026).

[3] R. Sennrich, B. Haddow, and A. Birch. *Neural Machine Translation of Rare Words with Subword Units*. arXiv v5. Aug. 31, 2015. arXiv: 1508.07909. URL: https://arxiv.org/abs/1508.07909v5 (visited on 10/22/2026).

[4] T. Kudo and J. Richardson. *SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing*. arXiv v1. Aug. 19, 2018. arXiv: 1808.06226. URL: https://arxiv.org/abs/1808.06226v1 (visited on 10/22/2026).

[5] O. Press and L. Wolf. *Using the Output Embedding to Improve Language Models*. arXiv v3. Aug. 20, 2016. arXiv: 1608.05859. URL: https://arxiv.org/abs/1608.05859v3 (visited on 10/22/2026).

[6] B. Zhang and R. Sennrich. *Root Mean Square Layer Normalization*. arXiv v1. Oct. 16, 2019. arXiv: 1910.07467. URL: https://arxiv.org/abs/1910.07467v1 (visited on 10/22/2026).

[7] N. Shazeer. *GLU Variants Improve Transformer*. arXiv v1. Feb. 12, 2020. arXiv: 2002.05202. URL: https://arxiv.org/abs/2002.05202v1 (visited on 10/22/2026).

[8] J. Su, Y. Lu, S. Pan, A. Murtadha, B. Wen, and Y. Liu. *RoFormer: Enhanced Transformer with Rotary Position Embedding*. arXiv v5. Apr. 20, 2021. arXiv: 2104.09864. URL: https://arxiv.org/abs/2104.09864v5 (visited on 10/22/2026).

[9] N. Shazeer. *Fast Transformer Decoding: One Write-Head is All You Need*. arXiv v1. Nov. 6, 2019. arXiv: 1911.02150. URL: https://arxiv.org/abs/1911.02150v1 (visited on 10/22/2026).

[10] J. Ainslie, J. Lee-Thorp, M. de Jong, Y. Zemlyanskiy, F. Lebrón, and S. Sanghai. *GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints*. arXiv v3. May 22, 2023. arXiv: 2305.13245. URL: https://arxiv.org/abs/2305.13245v3 (visited on 10/22/2026).

[11] A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret. *Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention*. arXiv v3. June 29, 2020. arXiv: 2006.16236. URL: https://arxiv.org/abs/2006.16236v3 (visited on 10/22/2026).

[12] DeepSeek-AI, A. Liu, B. Feng, et al. *DeepSeek-V2: A Strong, Economical, and Efficient Mixture-of-Experts Language Model*. arXiv v5. May 7, 2024. arXiv: 2405.04434. URL: https://arxiv.org/abs/2405.04434v5 (visited on 10/22/2026).

[13] P. Micikevicius, D. Stosic, N. Burgess, et al. *FP8 Formats for Deep Learning*. arXiv v2. Sept. 12, 2022. arXiv: 2209.05433. URL: https://arxiv.org/abs/2209.05433v2 (visited on 10/22/2026).

[14] B. Jacob, S. Kligys, B. Chen, et al. *Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference*. arXiv v1. Dec. 15, 2017. arXiv: 1712.05877. URL: https://arxiv.org/abs/1712.05877v1 (visited on 10/22/2026).

[15] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh. *GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers*. arXiv v2. Oct. 31, 2022. arXiv: 2210.17323. URL: https://arxiv.org/abs/2210.17323v2 (visited on 10/22/2026).

[16] J. Lin, J. Tang, H. Tang, et al. *AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration*. arXiv v6. June 1, 2023. arXiv: 2306.00978. URL: https://arxiv.org/abs/2306.00978v6 (visited on 10/22/2026).

[17] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han. *SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models*. arXiv v7. Nov. 18, 2022. arXiv: 2211.10438. URL: https://arxiv.org/abs/2211.10438v7 (visited on 10/22/2026).

[18] W. Shao, M. Chen, Z. Zhang, et al. *OmniQuant: Omnidirectionally Calibrated Quantization for Large Language Models*. arXiv v3. Aug. 25, 2023. arXiv: 2308.13137. URL: https://arxiv.org/abs/2308.13137v3 (visited on 10/22/2026).

[19] S. Ashkboos, A. Mohtashami, M. L. Croci, et al. *QuaRot: Outlier-Free 4-Bit Inference in Rotated LLMs*. arXiv v2. Mar. 30, 2024. arXiv: 2404.00456. URL: https://arxiv.org/abs/2404.00456v2 (visited on 10/22/2026).

[20] Z. Liu, C. Zhao, I. Fedorov, et al. *SpinQuant: LLM quantization with learned rotations*. arXiv v4. May 26, 2024. arXiv: 2405.16406. URL: https://arxiv.org/abs/2405.16406v4 (visited on 10/22/2026).

[21] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer. *QLoRA: Efficient Finetuning of Quantized LLMs*. arXiv v1. May 23, 2023. arXiv: 2305.14314. URL: https://arxiv.org/abs/2305.14314v1 (visited on 10/22/2026).

[22] V. Egiazarian, A. Panferov, D. Kuznedelev, E. Frantar, A. Babenko, and D. Alistarh. *Extreme Compression of Large Language Models via Additive Quantization*. arXiv v4. Jan. 11, 2024. arXiv: 2401.06118. URL: https://arxiv.org/abs/2401.06118v4 (visited on 10/22/2026).

[23] A. Tseng, J. Chee, Q. Sun, V. Kuleshov, and C. De Sa. *QuIP#: Even Better LLM Quantization with Hadamard Incoherence and Lattice Codebooks*. arXiv v2. Feb. 6, 2024. arXiv: 2402.04396. URL: https://arxiv.org/abs/2402.04396v2 (visited on 10/22/2026).

[24] S. Ma, H. Wang, L. Ma, et al. *The Era of 1-bit LLMs: All Large Language Models are in 1.58 Bits*. arXiv v1. Feb. 27, 2024. arXiv: 2402.17764. URL: https://arxiv.org/abs/2402.17764v1 (visited on 10/22/2026).

[25] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer. *LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale*. arXiv v2. Aug. 15, 2022. arXiv: 2208.07339. URL: https://arxiv.org/abs/2208.07339v2 (visited on 10/22/2026).

[26] W. Kwon, Z. Li, S. Zhuang, et al. *Efficient Memory Management for Large Language Model Serving with PagedAttention*. arXiv v1. Sept. 12, 2023. arXiv: 2309.06180. URL: https://arxiv.org/abs/2309.06180v1 (visited on 10/22/2026).

[27] Z. Liu, J. Yuan, H. Jin, et al. *KIVI: A Tuning-Free Asymmetric 2bit Quantization for KV Cache*. arXiv v2. Feb. 5, 2024. arXiv: 2402.02750. URL: https://arxiv.org/abs/2402.02750v2 (visited on 10/22/2026).

[28] G. Xiao, Y. Tian, B. Chen, S. Han, and M. Lewis. *Efficient Streaming Language Models with Attention Sinks*. arXiv v4. Sept. 29, 2023. arXiv: 2309.17453. URL: https://arxiv.org/abs/2309.17453v4 (visited on 10/22/2026).

[29] Z. Zhang, Y. Sheng, T. Zhou, et al. *H_2O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models*. arXiv v3. June 24, 2023. arXiv: 2306.14048. URL: https://arxiv.org/abs/2306.14048v3 (visited on 10/22/2026).

[30] J. Kaplan, S. McCandlish, T. Henighan, et al. *Scaling Laws for Neural Language Models*. arXiv v1. Jan. 23, 2020. arXiv: 2001.08361. URL: https://arxiv.org/abs/2001.08361v1 (visited on 10/22/2026).

[31] J. Hoffmann, S. Borgeaud, A. Mensch, et al. *Training Compute-Optimal Large Language Models*. arXiv v1. Mar. 29, 2022. arXiv: 2203.15556. URL: https://arxiv.org/abs/2203.15556v1 (visited on 10/22/2026).

[32] E. J. Hu, Y. Shen, P. Wallis, et al. *LoRA: Low-Rank Adaptation of Large Language Models*. arXiv v2. June 17, 2021. arXiv: 2106.09685. URL: https://arxiv.org/abs/2106.09685v2 (visited on 10/22/2026).

[33] R. Rafailov, A. Sharma, E. Mitchell, S. Ermon, C. D. Manning, and C. Finn. *Direct Preference Optimization: Your Language Model is Secretly a Reward Model*. arXiv v3. May 29, 2023. arXiv: 2305.18290. URL: https://arxiv.org/abs/2305.18290v3 (visited on 10/22/2026).

[34] L. Ouyang, J. Wu, X. Jiang, et al. *Training language models to follow instructions with human feedback*. arXiv v1. Mar. 4, 2022. arXiv: 2203.02155. URL: https://arxiv.org/abs/2203.02155v1 (visited on 10/22/2026).

[35] W. Fedus, B. Zoph, and N. Shazeer. *Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity*. arXiv v3. Jan. 11, 2021. arXiv: 2101.03961. URL: https://arxiv.org/abs/2101.03961v3 (visited on 10/22/2026).

[36] T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. Ré. *FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness*. arXiv v2. May 27, 2022. arXiv: 2205.14135. URL: https://arxiv.org/abs/2205.14135v2 (visited on 10/22/2026).

[37] G.-I. Yu, J. S. Jeong, G.-W. Kim, S. Kim, and B.-G. Chun. "Orca: A Distributed Serving System for Transformer-Based Generative Models". In: *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. July 2022, pp. 521–538. URL: https://www.usenix.org/conference/osdi22/presentation/yu (visited on 10/22/2026).

[38] Y. Leviathan, M. Kalman, and Y. Matias. *Fast Inference from Transformers via Speculative Decoding*. arXiv v2. Nov. 30, 2022. arXiv: 2211.17192. URL: https://arxiv.org/abs/2211.17192v2 (visited on 10/22/2026).

[39] A. Gu and T. Dao. *Mamba: Linear-Time Sequence Modeling with Selective State Spaces*. arXiv v2. Dec. 1, 2023. arXiv: 2312.00752. URL: https://arxiv.org/abs/2312.00752v2 (visited on 10/22/2026).

[40] T. Dao and A. Gu. *Transformers are SSMs: Generalized Models and Efficient Algorithms Through Structured State Space Duality*. arXiv v1. May 31, 2024. arXiv: 2405.21060. URL: https://arxiv.org/abs/2405.21060v1 (visited on 10/22/2026).

[41] J. Li, D. Li, S. Savarese, and S. Hoi. *BLIP-2: Bootstrapping Language-Image Pre-training with Frozen Image Encoders and Large Language Models*. arXiv v3. Jan. 30, 2023. arXiv: 2301.12597. URL: https://arxiv.org/abs/2301.12597v3 (visited on 10/22/2026).

[42] H. Liu, C. Li, Q. Wu, and Y. J. Lee. *Visual Instruction Tuning*. arXiv v2. Apr. 17, 2023. arXiv: 2304.08485. URL: https://arxiv.org/abs/2304.08485v2 (visited on 10/22/2026).

[43] P. Lewis, E. Perez, A. Piktus, et al. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. arXiv v4. May 22, 2020. arXiv: 2005.11401. URL: https://arxiv.org/abs/2005.11401v4 (visited on 10/22/2026).

[44] S. Yao, J. Zhao, D. Yu, et al. *ReAct: Synergizing Reasoning and Acting in Language Models*. arXiv v3. Oct. 6, 2022. arXiv: 2210.03629. URL: https://arxiv.org/abs/2210.03629v3 (visited on 10/22/2026).

[45] P. Liang, R. Bommasani, T. Lee, et al. *Holistic Evaluation of Language Models*. arXiv v2. Nov. 16, 2022. arXiv: 2211.09110. URL: https://arxiv.org/abs/2211.09110v2 (visited on 10/22/2026).

[46] L. Zheng, W.-L. Chiang, Y. Sheng, et al. *Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena*. arXiv v4. June 9, 2023. arXiv: 2306.05685. URL: https://arxiv.org/abs/2306.05685v4 (visited on 10/22/2026).