

LLMS FROM SCRATCH • THEORY & PRACTICE

从文本到语言模型

公开源码、数学与工程实践

从文本处理、注意力到完整 GPT
训练、微调与架构扩展

原创中文学习书 · 机制、推导、代码与实验
基于 Sebastian Raschka 的公开项目源码与技术资料

目录

从文本到语言模型：公开源码、数学与工程实践	11
怎样使用这本书	11
来源与许可	12
阅读路线	13
怎样安排实验	13
怎样回查证据	13
第 01 章 先画清语言模型的任务与边界	14
1.1 一次预测到底输入、输出什么	14
1.2 从局部预测到整段文本	14
1.3 预训练、微调与使用阶段改变了什么	15
1.4 “从零”应该让哪些部分可检查	16
1.5 按一条可验证的路线读	16
1.6 练习与检查	16
参考与覆盖边界	17
第 02 章 张量、自动微分与最小训练闭环	18
2.1 张量不仅有数值，还有形状、类型和存储	18
2.2 从一个 sigmoid 手算梯度	18
2.3 计算图不会替你决定何时清空梯度	19
2.4 用 Module 组织三层网络	20
2.5 Dataset 和 DataLoader 分别负责什么	20
2.6 每一步更新都要有明确先后	20
2.7 保存的是哪些状态	21
2.8 从单设备到 DDP，代码多出来的责任	21
2.9 运行、练习与答案	22
参考与实现覆盖	22
第 03 章 文本怎样变成 token：词表、BPE 与特殊符号	23
3.1 先实现一个会失败的分词器	23
3.2 特殊符号是协议，不是普通装饰	23
3.3 字节与字符的区别先于 BPE	24
3.4 训练 BPE：数相邻对，再无重叠替换	24
3.5 编码时为什么看合并排名，不重新训练	25
3.6 上游两种自写 BPE 的用途和实测边界	25
3.7 比较分词器时先比较结果，再比较速度	26
3.8 滑窗负责构造问题和答案	26

3.9 Embedding 是可训练的取行操作	27
3.10 加入位置后才知道“在哪儿”	28
3.11 练习、提示与答案	28
参考与实现覆盖	28
第 04 章 从加权平均推导自注意力与因果掩码	29
4.1 先把加权平均写清楚	29
4.2 为什么拆成查询、键和值	29
4.3 缩放因子来自点积的方差	30
4.4 因果掩码为什么应放在 softmax 前	30
4.5 通过修改未来输入来验收	31
4.6 dropout 并不保持每行权重和恰好为 1	31
4.7 加上批次和多个头	31
4.8 配套实现怎样对照每一行	32
4.9 练习与答案	32
参考与覆盖	33
第 05 章 多头注意力的实现选择与 buffer 生命周期	34
5.1 把三次投影合成一次，不改变线性代数	34
5.2 einsum 让索引关系写在代码里	34
5.3 SDPA 的 mask 与 dropout 约定	35
5.4 完整 MultiheadAttention 类不只是公式包装	35
5.5 FlexAttention 表达规则，执行条件另算	35
5.6 buffer 解决的是模块状态管理	36
5.7 怎样测量而不混淆问题	36
5.8 一张比较表，明确各自代价	37
5.9 练习与答案	37
参考与覆盖	37
第 06 章 把注意力组装成 GPT：归一化、残差与输出头	38
6.1 配置是一组互相约束的尺寸	38
6.2 LayerNorm 沿最后一个特征轴工作	38
6.3 GELU 与逐位置前馈网络	38
6.4 残差提供一条直接传递的路径	39
6.5 一个完整 block 的先后顺序	40
6.6 从 ID 到 logits 的完整路线	40
6.7 用参数数量检查有没有漏层	40
6.8 与上游对齐，误差应该怎样解释	41
6.9 生成循环是外层程序	41
6.10 练习与答案	41
参考与覆盖	42

第 07 章 参数、FLOPs、显存与吞吐怎样估算.....	43
7.1 从一个矩阵乘法开始记账.....	43
7.2 一个普通 Transformer block 的主要运算.....	43
7.3 为何反向常粗估为前向的两倍.....	44
7.4 权重字节数只是内存的第一项.....	44
7.5 thop 的数字要问“数到了哪些算子”.....	44
7.6 二分搜索最大批量能证明什么.....	45
7.7 吞吐和 MFU 的量纲检查.....	45
7.8 用预算决定下一步，而不代替测量.....	45
7.9 练习与答案.....	45
参考与覆盖.....	46
第 08 章 KV cache：先证明等价，再讨论加速.....	47
8.1 为什么可以按层复用.....	47
8.2 prefill 与 decode 的形状不同.....	47
8.3 上游朴素版怎样保存状态.....	48
8.4 本书采用显式输入、输出缓存.....	48
8.5 预分配减少重复分配，却引入容量管理.....	49
8.6 丢弃 K/V 不等于重算截断窗口.....	49
8.7 绝对位置容量仍是硬限制.....	49
8.8 计算量与存储要分开算.....	50
8.9 怎样把等价检查写完整.....	50
8.10 练习与答案.....	50
参考与覆盖.....	51
第 09 章 GQA、MLA 与跨层 KV 共享.....	52
09.1 先把缓存账算在正确的对象上.....	52
09.2 GQA：多个问题，可以共用一套历史记录.....	53
09.3 MLA：先存一个低维向量，需要时再展开.....	53
09.4 跨层共享：消费者仍然提问，生产者负责写入.....	54
09.5 从注意力接回 GPT：哪些步骤没有改变.....	54
09.6 可复现实验、练习与答案.....	55
第 10 章 局部与稀疏注意力：SWA 和 DSA.....	56
10.1 SWA 的窗口到底包含谁.....	56
10.2 持久缓存可以只留 W，当前计算却未必只用 W.....	56
10.3 DSA：先索引，再做注意力.....	57
10.4 稀疏选择不等于稀疏执行，也不等于自动学会选择.....	58
10.5 配套脚本与验证路径.....	58
10.6 补充练习与答案.....	58
第 11 章 Gated DeltaNet：从矩阵状态理解递推.....	60

11.1 一个 key 对应一行读出方向	60
11.2 Delta rule 用预测误差修正记忆	60
11.3 源码每一步的形状与顺序	61
11.4 为什么可以分块，为什么仍不能称为任意 softmax 的等价替代	61
11.5 存储与计算的边界	62
11.6 真实验证与补充练习	62
第 12 章 MoE：路由、专家与负载	63
12.1 路由器决定“谁来计算”，专家决定“计算什么”	63
12.2 为什么先展平，再按专家收集 token	64
12.3 总参数、激活参数与内存是三笔账	64
12.4 负载集中为什么会成为问题	65
12.5 接回主干、验证与练习	65
第 13 章 预训练目标、交叉熵与验证集	67
13.1 交叉熵取的是目标 ID 对应的概率	67
13.2 flatten 不应该打乱位置与标签配对	67
13.3 perplexity 依赖分词与计量单位	68
13.4 验证集要在构造重叠窗口之前分开	68
13.5 loader 平均要明确分母	68
13.6 训练、评价、采样三段的模式切换	68
13.7 AdamW 更新与短实验的目的	69
13.8 如何读上游训练脚本和测试	69
13.9 练习与答案	70
参考与覆盖	70
第 14 章 采样、温度、top-k 与生成边界	71
14.1 贪心选择不需要 softmax	71
14.2 温度改变概率比值	71
14.3 top-k 删除支持集之外的候选	71
14.4 采样频数只能近似概率	72
14.5 一个生成循环应有明确的停止协议	72
14.6 上下文截断与 KV cache 不能混用旧假设	72
14.7 复现采样时应记录的不止 seed	73
14.8 习题与复查	73
参考与覆盖	73
第 15 章 权重格式、参数映射与低内存加载	74
15.1 先区分推理恢复与训练恢复	74
15.2 官方 GPT-2 的名字怎样映射到教学模型	74
15.3 assign 会替换参数对象	75
15.4 三种替代文件入口	75

15.5 下载器的成功条件不能只看文件存在.....	76
15.6 加载峰值为何可能接近两份权重.....	76
15.7 低内存加载不能漏掉 buffer.....	76
15.8 mmap 与 assign 的小型实测.....	77
15.9 如何解读内存监控代码.....	77
15.10 练习与答案.....	78
参考与覆盖.....	78
第 16 章 语料整理与 Gutenberg 训练工程.....	79
16.1 语料清单应先于合并大文件.....	79
16.2 上游清理脚本逐步做了什么.....	79
16.3 分片大小要把分隔符也算进去.....	79
16.4 去重应发生在划分审计中.....	80
16.5 按文件训练怎样管理进度.....	80
16.6 检查点频率代码值得逐字符核对.....	80
16.7 ETA 与日志也有计量假设.....	81
16.8 CLI 与测试入口的边界.....	81
16.9 可运行的小管道与练习.....	81
参考与覆盖.....	82
第 17 章 学习率、梯度裁剪、超参数与 Muon.....	83
17.1 先约定 step 表示什么.....	83
17.2 为什么先升后降.....	83
17.3 梯度裁剪缩放整组向量.....	84
17.4 日程和裁剪的边界要一起测.....	84
17.5 网格搜索的乘法预算.....	85
17.6 搜索脚本的失败路径也属于实验.....	85
17.7 Muon 改变的是矩阵更新方向.....	86
17.8 两个优化器必须不重不漏地分参数.....	86
17.9 练习与答案.....	86
参考与覆盖.....	87
第 18 章 训练性能及 CPU、MPS 数值差异.....	88
18.1 先建立可比较的基线.....	88
18.2 减少中间张量和调用开销.....	88
18.3 TF32、bfloat16 与 autocast 不是同一件事.....	88
18.4 pinned memory 在主机，不是预留显存.....	89
18.5 编译收益需要排除首次成本.....	89
18.6 词表对齐增加的行不是合法文本 token.....	89
18.7 计时边界能让同一个数字有不同含义.....	90
18.8 DDP 同步参数梯度，不会自动汇总所有指标.....	90

18.9 相同 seed 不保证跨设备相同随机数	90
18.10 浮点加法不满足实数里的结合律	91
18.11 练习与答案	91
参考与覆盖	91
第 19 章 从 GPT 到 Llama: RoPE、RMSNorm 与门控前馈	92
19.1 RMSNorm 保留均值, 只调节整体幅度	92
19.2 SwiGLU 的两个分支分别做什么	92
19.3 RoPE 把位置变成二维旋转	93
19.4 Llama 3 的 GQA 与长上下文频率	94
19.5 分词器、对话边界与权重布局是一套协议	94
19.6 主干外的辅助步骤与源码差异	95
19.7 真实检查、练习与答案	95
19.8 上游测试怎样把错误定位到零件	95
第 20 章 Qwen3 与 Qwen3.5: 同一个名字下, 两种记忆接口	97
20.1 Qwen3: Q/K 归一化发生在每个头内部	97
20.2 Qwen3 MoE: 注意力不变, 前馈变为按 token 分派	97
20.3 Qwen3 缓存: 位置计数和 KV 容器缺一不可	98
20.4 Qwen3.5 的全局层: 局部旋转与额外输出门	98
20.5 线性层: 卷积记忆加矩阵记忆	98
20.6 chunk 与 recurrent helper 如何相互校验	99
20.7 混合缓存的一个重要限制	100
20.8 加载、真实结果与练习	100
20.9 上下文拒绝、参考调试器与聊天界面	101
第 21 章 Gemma 3、Gemma 4: 局部层、共享层与数值约定	102
21.1 Gemma 3: 残差分支前后都要归一化	102
21.2 局部/全局层使用不同的位置表和 mask	102
21.3 Gemma 4: 局部头和全局头可以不同宽	103
21.4 proportional RoPE 保留不旋转的子空间	103
21.5 KV 共享必须寻找同一种层的生产者	103
21.6 逐层输入: 每一层都有自己的 token 侧信息	104
21.7 加载器和后半段 notebook 的职责	105
21.8 实测与补充题	105
21.9 一个实际失败案例: 共享消费者读到了裁剪后的 prompt	105
21.10 上游参考对照与逐层调试的范围	106
第 22 章 OLMo 3、Tiny Aya 与可替换模型接口	107
22.1 OLMo 3 把 norm 放在分支输出	107
22.2 YaRN 不是统一放慢所有频率	107
22.3 Tiny Aya: 无 bias 的 LayerNorm 仍然减均值	108

22.4 Tiny Aya 的位置配对与全局层例外	109
22.5 可替换接口：先约定张量，再约定状态	109
22.6 用 Llama/Qwen3 重走第五章训练	109
22.7 真实检查与补充练习	110
22.8 缓存差异与逐层参考测试	110
第 23 章 扩展分词器：新 ID 还需要新的模型行	112
23.1 特殊 token 与普通 BPE 合并项不同	112
23.2 输入 embedding 为什么必须增加行	112
23.3 输出头也必须扩展，但旧概率不会保持不变	113
23.4 Weight tying 必须共享对象，不只是值相同	113
23.5 优化器、数据与 checkpoint 都要一起迁移	114
23.6 真实小实验、源单元与练习	114
23.7 不下载词表的真实 tiktoken 检查	115
第24章 文本分类：一个输出位置，决定模型看见多少内容	116
24.1 从逐词预测改成整段判断	116
24.2 数据集先确定长度，再划清训练与评估	117
24.3 冻结不是切断整条计算图	117
24.4 附加实验怎样构成有意义的对照	118
24.5 IMDb：先问表示是否值得付出成本	119
24.6 保存、恢复与检查	119
第25章 指令微调：先把“该预测什么”排对	121
25.1 模板是模型输入的一部分	121
25.2 从三条变长序列到一个 batch	121
25.3 两种掩码不能互相代替	122
25.4 截断可能把监督全部截没	123
25.5 训练与生成的闭环	123
第26章 指令数据：先防止自己给自己打高分	124
26.1 近似重复是候选关系，不是事实判决	124
26.2 先分组，再切分	124
26.3 被动语态生成与反思修订	125
26.4 模型评分该记录什么	126
26.5 相关不等于校准正确	126
第27章 DPO：比较两个回答，也要说明比较的基准	128
27.1 偏好对不是两个独立分类样本	128
27.2 从受约束奖励到概率比	128
27.3 为什么要用稳定的 logsigmoid	129
27.4 从 token logits 取得回答分数	130

27.5 参考模型、训练与日志	130
第28章 LoRA：冻结大矩阵，学习一个可合并的增量	132
28.1 先统一矩阵朝向	132
28.2 缩放约定必须跟代码走	133
28.3 零初始化其中一边，为什么还能学	133
28.4 替换线性层与参数管理	133
28.5 合并与反向合并	134
第29章 三种界面，同一条推理链路	135
29.1 四个入口对应三类任务	135
29.2 加载成功不是推理准备完成	135
29.3 上游是一次返回，流式是补充接口	136
29.4 上下文预算属于接口，不应留给崩溃处理.....	136
第30章 包接口与测试：怎样知道自己运行的是同一份模型.....	138
30.1 模块名相同，不代表实现细节永远相同	138
30.2 包额外承担的边界.....	138
30.3 测试应该说明它证明了什么	138
30.4 环境检查与持续集成	139
30.5 这份交付怎样复核.....	139
第31章 习题闭环与延伸：知道哪里已经验证，哪里还没有.....	141
31.1 原题、公开解答与本书补充题的区别.....	141
31.2 用一组题检查自己是否真的能修改代码	142
31.3 独立推理仓库的边界	142
31.4 参考文献怎么用，而不只是放在书尾.....	143
31.5 什么叫完成这次复现	143
参考资料索引	144
第 01 章 先画清语言模型的任务与边界	144
第 02 章 张量、自动微分与最小训练闭环	144
第 03 章 文本怎样变成 token：词表、BPE 与特殊符号.....	145
第 04 章 从加权平均推导自注意力与因果掩码	145
第 05 章 多头注意力的实现选择与 buffer 生命周期	145
第 06 章 把注意力组装成 GPT：归一化、残差与输出头	146
第 07 章 参数、FLOPs、显存与吞吐怎样估算.....	146
第 08 章 KV cache：先证明等价，再讨论加速.....	146
第 09 章 GQA、MLA 与跨层 KV 共享	146
第 10 章 局部与稀疏注意力：SWA 和 DSA.....	147
第 11 章 Gated DeltaNet：从矩阵状态理解递推	147
第 12 章 MoE：路由、专家与负载	147

第 13 章 预训练目标、交叉熵与验证集	148
第 14 章 采样、温度、top-k 与生成边界	148
第 15 章 权重格式、参数映射与低内存加载	148
第 16 章 语料整理与 Gutenberg 训练工程	149
第 17 章 学习率、梯度裁剪、超参数与 Muon	149
第 18 章 训练性能及 CPU、MPS 数值差异	149
第 19 章 从 GPT 到 Llama: RoPE、RMSNorm 与门控前馈	150
第 20 章 Qwen3 与 Qwen3.5: 同一个名字下, 两种记忆接口	150
第 21 章 Gemma 3、Gemma 4: 局部层、共享层与数值约定	150
第 22 章 OLMo 3、Tiny Aya 与可替换模型接口	151
第 23 章 扩展分词器: 新 ID 还需要新的模型行	151
第 24 章 文本分类: 一个输出位置, 决定模型看见多少内容	151
第 25 章 指令微调: 先把“该预测什么”排对	152
第 26 章 指令数据: 先防止自己给自己打高分	152
第 27 章 DPO: 比较两个回答, 也要说明比较的基准	152
第 28 章 LoRA: 冻结大矩阵, 学习一个可合并的增量	153
第 29 章 三种界面, 同一条推理链路	153
第 30 章 包接口与测试: 怎样知道自己运行的是同一份模型	153
第 31 章 习题闭环与延伸: 知道哪里已经验证, 哪里还没有	154
完整实验代码	155
examples/architectures/architecture_checks.py	155
examples/architectures/cache_checks.py	158
examples/architectures/draw_figures.py	160
examples/architectures/source_attention_checks.py	161
examples/architectures/source_smoke.py	162
examples/core_training/cache_probes.py	164
examples/core_training/checkpoint_checks.py	165
examples/core_training/core.py	167
examples/core_training/corpus_demo.py	170
examples/core_training/draw_figures.py	171
examples/core_training/loss_sampling_checks.py	173
examples/core_training/optimizer_numerics.py	174
examples/core_training/run_checks.py	175
examples/core_training/source_variant_checks.py	180
examples/core_training/tokenizer_probes.py	181
examples/finetuning/check_upstream_functions.py	183
examples/finetuning/contracts.py	185
examples/finetuning/draw_figures.py	188

从文本到语言模型：公开源码、数学与工程实践

怎样使用这本书

一个能生成句子的模型，距离一个训练过程可解释、评估结果可信的系统，还有数据处理、优化、检查点和部署接口等环节。这本书沿着 `rasbt/LLMs-from-scratch` 的公开代码，把这些环节连接起来。读者可以先做最小闭环，再逐项改变模型结构，观察每个选择带来的结果与成本。

本书是依据公开源码与原始技术资料编写的原创中文学习书，不是 Sebastian Raschka 已出版书籍的翻译或替代版本，不声称原作者审阅或认可。上游许可明确排除了相关书籍和图片；本文不复刻原书图文，图解重新绘制，引用或改编的代码保留原许可与修改说明。

范围固定在提交 `cdbd33e6d57a71e20ea31434841095fac50cecd8`，包含主线七章对应代码、附录 A-E 和本仓库内的补充材料。单独链接的 `reasoning-from-scratch` 是另一项目，本文说明其与本书的接口及延伸方向，不把整套续作算作已经覆盖。

每章讲清数据和张量怎样变化、关键公式为何成立，以及如何用小实验发现错误。CPU 小模型的验证不能证明大模型的能力，数学等价也不能证明某种 GPU 实现更快。涉及未运行的 GPU、多机、预训练权重、真实语料或外部 API，会在相应位置注明。所有版本与输出以本书配套实验记录为准。

先读正文中的局部例子，再用文末完整代码和配套文件复现。不要直接执行下载脚本、在线评测或大规模训练命令；先确认各章的依赖、资源、数据许可和费用边界。

项目入口：[LLMs-from-scratch](#)。

来源与许可

本项目独立成书，不与其他仓库合并。固定源：<https://github.com/rasbt/LLMs-from-scratch/tree/cdbd33e6d57a71e20ea31434841095fac50cecd8>。

仓库采用修改过的 Apache 2.0 文本，Source 定义明确排除相关书籍与图片。不是出版书中文翻译。仅依据公开源码与公开说明原创撰写，所有示意图重新绘制；复制或改写代码保留完整许可、版权、修改标记与适用 NOTICE。禁止把书籍正文或原图默认视为 Apache 授权。

原许可：<https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/LICENSE.txt>，完整副本见 `research/UPSTREAM_LICENSE.txt`。源版本与逐文件哈希见 `research/source_pin.json`、`research/source_inventory.json`。

全部解释、数学展开、练习与新增实验须区分原创、改编和引用，不暗示上游作者认可。修改的上游代码必须在文件头注明修改。未核验第三方权利的图片与截图不直接复用。

阅读路线

这本书按依赖关系组织：先弄清输入、目标和状态，再比较实现选择，最后用小实验核对结论。第一次阅读可以顺序推进，复习时按下面的阶段回查。

阶段	章节	关注的问题
建立最小闭环	1—8	从 token 到注意力、GPT、参数成本与缓存；先把每个张量维度解释清楚。
看懂架构选择	9—12	比较 GQA、MLA、局部/稀疏注意力、线性状态与 MoE 的不同成本。
训练与评估	13—18	掌握损失、采样、权重、数据、优化和性能的边界。
阅读模型变体	19—23	对照 Llama、Qwen、Gemma、OLMo、Tiny Aya 和扩词表实现。
适配任务与交付	24—31	分类、指令、数据质量、DPO、LoRA、界面和可复现实验。

怎样安排实验

各组运行说明见 `examples/core_training`、`examples/architectures`、`examples/finetuning` 及相关章节。先运行不需要预训练权重的 CPU 最小例，再决定是否扩大实验规模。固定上游代码出现的失败案例有单独说明；不要把它们改写成“所有缓存路径等价”的通过结论。

训练损失下降、数学例子相符和某个真实模型表现更好，是三种不同结论。请分别记录输入、版本、随机种子、比较条件和输出；书中未运行的大规模、GPU、API 或在线服务路径不算已验证。

怎样回查证据

正文在关键观点附近标注固定源码、官方文档或原始论文，书末集中列参考资料。完整原创实验代码列于附录，也以单独文件放在 ZIP 中。配套原始依赖保留路径和许可；不要任意移动后再运行。

第 01 章 先画清语言模型的任务与边界

给程序输入“今天的天空很”，它返回“蓝”的分数比“桌子”高，这件事本身还不是一个对话产品。我们需要知道文本怎样变成数字、分数怎样得到、什么数据让这些分数有用，以及程序什么时候应该停止。本书沿着这条计算链构造模型，再讨论如何把模型接到分类、指令和交互任务中。

本书依据 `rasbt/LLMs-from-scratch` 的公开源码撰写，固定提交为 `cdbd33e6d57a71e20ea31434841095fac50cecd8`。上游 `ch01` 明确没有主章节代码，只提供阅读建议和两个视频入口。因此，本章的概率展开、示意图和练习是为本书新写的教学内容，并非出版书第 1 章的翻译，也不声称逐秒核对了视频。来源：[固定版本 ch01/README](#)

1.1 一次预测到底输入、输出什么

设词表大小为 V ，一段文本经过分词后成为整数序列 $x_1, \dots, x_T$ ，每个 x_t 都在 0 到 $V - 1$ 之间。这里的 token 是分词器定义的单位，可能是一个词、词的一部分、标点，也可能只是 UTF-8 字节片段。中文字符与 token 数量没有固定的一一关系。

模型用参数 θ 计算下一个 token 的条件概率：

$$p_{\theta}(x_{t+1} = v \mid x_1, \dots, x_t), \quad v \in \{0, \dots, V - 1\}.$$

一次输入 B 条、每条 T 个 token 的批次，输入整数张量形状是 $[B, T]$ 。模型通常为每个位置都输出 V 个未经归一化的分数，称为 logits，形状为 $[B, T, V]$ 。softmax 把最后一维转成概率。训练时利用所有位置，生成时通常只取最后一个位置的分数。

例如，只有四个 token 的教学词表可以是“天”“蓝”“雨”“结束”。分数为 $[1, 3, 0, -1]$ 时，减去最大值后计算指数，得到 $[e^{-2}, 1, e^{-3}, e^{-4}]$ ，再除以总和。减去同一个数不会改变比例，却能避免对很大正数求指数。分数最大的一项是“蓝”；采样仍可能选中其他项。这是“模型的分布”和“程序的选择策略”的区别，第 14 章会单独处理。

1.2 从局部预测到整段文本

概率的链式法则把联合概率展开为：

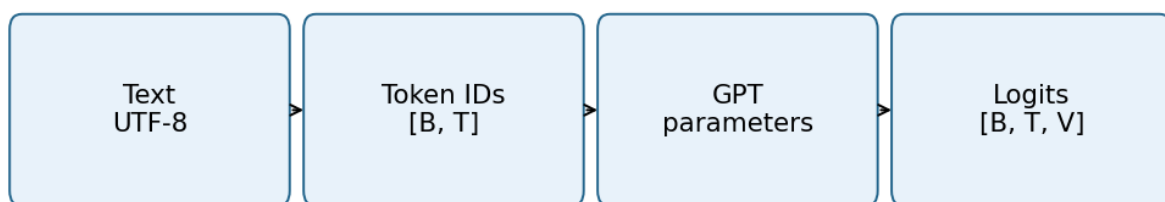
$$p_{\theta}(x_1, \dots, x_T) = \prod_{t=1}^T p_{\theta}(x_t \mid x_{<t}).$$

$x_{<t}$ 表示位置 t 之前的所有 token。第一项需要起始上下文，可以由约定的起始符号或提示词提供。训练时我们观察完整文本，并让位置 t 的计算只能使用此前的信息。这样，每个位置都能同时得到训练信号，而不必真的生成前一个 token 才训练后一个 token。因果掩码就是保证这个限制的机制。

乘许多小概率容易数值下溢。取负对数后，乘积变为求和：

$$-\log p_{\theta}(x_1, \dots, x_T) = -\sum_{t=1}^T \log p_{\theta}(x_t | x_{<t}).$$

这个目标鼓励模型给语料里实际出现的续写更高概率。它不直接测量陈述是否真实、建议是否安全、回答是否符合某个人的意图。训练文本的规律和缺陷都会影响模型。一个很低的训练损失甚至可能只是把短文本记住了。实现落点：上游 `calc_loss_batch` 与训练循环



Training: shifted targets -> loss -> gradients; generation: last logits -> next ID

1.3 预训练、微调与使用阶段改变了什么

预训练主要改变参数，使模型学会语料中的续写关系。文本分类通常保留大部分计算骨架，把输出改成少量类别。指令微调仍可使用下一个 token 目标，但训练样本按照“指令、输入、回答”的约定排布；哪些位置计入损失会改变优化重点。偏好训练则比较回答之间的相对偏好，目标函数与普通续写训练不完全相同。

检索增强是在使用阶段把外部材料加入上下文，不一定更新参数。工具调用还需要外部程序验证模型提出的操作、执行工具并回传结果。模型产生字符串并不自动获得读文件、发邮件或运行代码的能力。把这些层次分开，才能知道问题该在哪一层修：知识过期可涉及检索，标签定义含糊需要改数据，越权执行需要改程序边界，不能全靠扩大模型解决。

本书会建立下表中的机制，但不会把小实验的成功推广为大规模系统的效果。

阶段	直接输入	训练或执行时改变的对象	本书能验证的范围
预训练	token 序列	模型参数	小模型前后向、损失、保存恢复
分类微调	文本及类别	输出头和选定参数	标签、冻结、梯度路径
指令微调	格式化指令与回答	选定模型参数	模板、padding、loss mask

阶段	直接输入	训练或执行时改变的对象	本书能验证的范围
生成	提示词与采样设置	token 序列、可选缓存	形状、概率、停止和缓存一致性
外部工具	模型输出与权限规则	外部系统状态	解释接口，不执行不可信生成代码

1.4 “从零”应该让哪些部分可检查

从零不是拒绝所有库。我们使用 PyTorch 的张量运算和自动微分，但把模型层、数据配对、损失、训练循环和生成逻辑写出来。这样，错误可以落到某个明确的计算上：把标签移错了一位、softmax 归一化了错误的维度、缓存位置从零重算、或者验证数据和训练窗口重叠。

配套的 `examples/core_training/core.py` 使用 16 维隐藏状态、2 层、4 个注意力头和 16 个 token 的小词表。这不是 GPT-2 的预训练权重，而是一个随机初始化、可在 CPU 上运行的模型。它共有 7,264 个可训练参数。第 6 章将推导这个数，实验会把它与上游同结构的缩小配置逐项对齐。

整个实验固定随机种子 2718、CPU 线程数 1，记录 Python 3.12.14、PyTorch 2.6.0+cpu。固定种子帮助在相同条件下重跑，不保证换设备、换库版本后逐位相同。跨 CPU、CUDA、MPS 的区别在第 18 章解释。所有实测输出放在配套 `qa/core_training/results.json`，没有下载模型权重，也没有运行在线评估。

1.5 按一条可验证的路线读

上游阅读建议强调依次阅读、运行代码、做练习，再带着问题尝试自己的项目。本书把这个过程落实为三个检查：先用一句话说清本节输出什么；再手算一个小例子；最后运行断言，故意制造一次错误。只看到 loss 下降还不够，标签错位的模型也可能下降。来源：[上游阅读建议](#)

第 2 章补足 PyTorch 的最小训练闭环；第 3—6 章沿文本、注意力、Transformer 到 logits 逐步搭建；第 7—12 章处理成本、缓存及架构变化；第 13 章以后进入训练、加载和各种适配任务。后面的优化必须保留前面的计算含义。一个快了两倍却悄悄看到了未来标签的实现，没有通过验收。

1.6 练习与检查

以下是本书补充题。

1. 词表有 10 项，输入为 3 条、每条 7 个 token。输入、logits、生成时最后位置 logits 的形状分别是什么？
2. 若训练文本中的某个回答出现频率很高，是否足以说明模型生成该回答时事实正确？写出一个反例。
3. 为什么训练时能并行计算多个位置，而自回归生成仍要逐步进行？

提示：区分“已知完整训练序列”和“尚未选择出的未来 token”。

答案：形状依次是 $[3, 7]$ 、 $[3, 7, 10]$ 、 $[3, 10]$ 。大量重复的错误日期可以得到高预测概率，频率不能代替事实核验。训练目标已由数据给出，因果掩码限制每个位置的可见范围；生成时下一步输入依赖刚选出的 token，存在真实的数据依赖。

参考与覆盖边界

- 固定版本 ch01：无主代码；阅读建议和外链入口已区分。
- 固定版本主训练脚本：本章任务定义的后续可执行落点。
- 上游视频、出版书正文及图片不计为本章已逐项复现的材料。图为新绘制，具体算法的代码证据在后续各章给出。

第 02 章 张量、自动微分与最小训练闭环

训练语言模型之前，先让一个小分类器完整地训练、检查和保存。这个例子没有文本，却包含后面几乎所有训练程序的基本动作。读完本章，应能追踪一个批次的形状、说明每一步梯度从哪里来，并区分数据、参数、计算图和设备。

上游对应 [appendix-A](#) 的两个主 notebook、两种 DDP 启动脚本及练习。本章逐步解释这些实现；CPU 实验另用四个自造样本，避免把上游演示输出冒充本书实测。[固定版本 Appendix A](#)

2.1 张量不仅有数值，还有形状、类型和存储

标量的形状是 `[]`，向量是 `[D]`，矩阵是 `[B,D]`。三维张量 `[B,T,D]` 在本书中通常表示批次、序列位置和特征维。轴的含义是程序约定，不是张量自己知道的事情。一个维度相同的转置错误可能不会报错，却会改变模型。

整数 ID 通常用 `torch.long`，模型权重通常用浮点类型。整数不适合做需要连续梯度的参数。`.to(torch.float32)` 改变类型，`.to(device)` 改变存放设备；两种变化都应有明确理由。

`torch.tensor(numpy_array)` 复制数据，`torch.from_numpy(numpy_array)` 在支持的条件下共享底层内存。后者节省复制，但修改原数组也会改变张量，不能把它误当成快照。

矩阵 $A \in \mathbb{R}^{2 \times 3}$ 和转置 $A^T \in \mathbb{R}^{3 \times 2}$ 相乘，得到 $AA^T \in \mathbb{R}^{2 \times 2}$ 。第 (i,j) 项是两行向量的点积。`A * A` 是逐元素乘法，`A @ A.T` 才是这里的矩阵乘法。

`reshape` 不等于转置。把 `[2,3]` 改成 `[3,2]` 只是按存储访问顺序重排形状；`.T` 则交换轴。张量转置后通常不连续，`view` 要求新形状与原有步幅兼容，`reshape` 必要时会复制。因此注意力里常出现 `.transpose(...).contiguous().view(...)`。我们的反例把 `arange(6).reshape(2,3)` 转置，再 `view(6)`，确实触发错误；`reshape(6)` 返回 `[0,3,1,4,2,5]`，并没有回到原始行顺序。

2.2 从一个 sigmoid 手算梯度

设输入 $x = 1.3$ ，权重 $w = 0.7$ ，偏置 $b = -0.2$ ，真实标签 $y = 1$ 。先算 $\text{logit } z = wx + b = 0.71$ ，再算概率 $a = \sigma(z) = 1/(1 + e^{-z})$ 。二分类交叉熵为

$$L = -y \log a - (1 - y) \log(1 - a).$$

逐步应用链式法则：

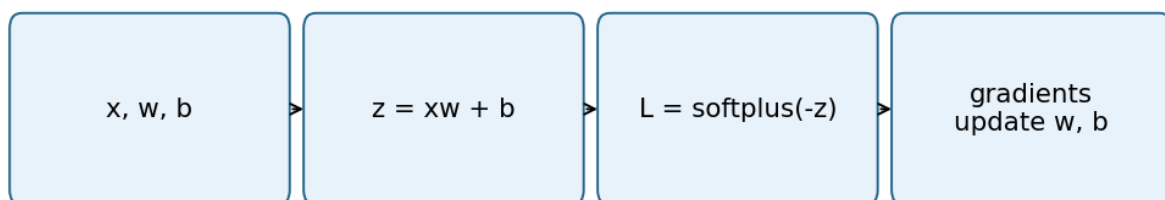
$$\frac{\partial L}{\partial a} = -\frac{y}{a} + \frac{1-y}{1-a}, \quad \frac{\partial a}{\partial z} = a(1-a), \quad \frac{\partial L}{\partial z} = a - y.$$

最后得到 $\partial L / \partial w = (a - y)x$ 、 $\partial L / \partial b = a - y$ 。本书用 float64 实测 $L = 0.3998789998$ ， $\partial L / \partial w = -0.4284784922$ ， $\partial L / \partial b = -0.3295988402$ 。中心差分

$$\frac{L(w + \varepsilon) - L(w - \varepsilon)}{2\varepsilon}, \quad \varepsilon = 10^{-6}$$

与自动微分的绝对误差约为 2.19×10^{-11} 。差分只用于检查小例子；给每个大模型参数都做两次前向计算不可行。

上游先使用 sigmoid 再调用 `binary_cross_entropy`，让概率与损失的关系直观可见。实际实现可把两步合成 `binary_cross_entropy_with_logits`，避免显式计算接近 0 或 1 的概率后取对数。传给它的是 z ，不能再预先 sigmoid。



For $y = 1$: $dL/dz = \text{sigmoid}(z) - 1$; $dL/dw = x * dL/dz$

2.3 计算图不会替你决定何时清空梯度

`requires_grad=True` 让浮点张量参与梯度追踪。前向运算构建本次计算图，`torch.autograd.grad(loss, parameters)` 返回所请求的导数；`loss.backward()` 则把导数累加到叶子参数的 `.grad`。同一张图的中间结果通常在反向后释放，再用它反向需要保留图或重新前向。

这里有两个常见混淆。第一，`retain_graph=True` 是保留计算图，不是“让训练记住历史”。第二，`optimizer.zero_grad()` 清空梯度，不会清空模型权重。省略清零会把多个批次的导数加起来；这可以有意的梯度累积，也可能是错误。真正的梯度累积还要统一损失的分母，不能无条件把不同大小的批次平均值相加。

推理时 `torch.no_grad()` 停止记录反向图，节省内存；`model.eval()` 切换 dropout、BatchNorm 等层的行为。两者做不同的事。一个没有 dropout 的小网络可能让错误写法暂时看不出差别，到了 GPT 就会出现不稳定生成。

2.4 用 Module 组织三层网络

上游 `NeuralNetwork` 使用 `Linear → ReLU → Linear → ReLU → Linear`，隐藏宽度依次为 30、20。输入 `[B, 50]`、输出 3 类时，权重形状分别为 `[30, 50]`、`[20, 30]`、`[3, 20]`，偏置分别为 `[30]`、`[20]`、`[3]`。PyTorch 的 `Linear` 保存的是 `[out_features, in_features]`，前向相当于 $XW^T + b$ 。

参数数量要把偏置算进去：

$$(50 \times 30 + 30) + (30 \times 20 + 20) + (20 \times 3 + 3) = 2213.$$

上游练习 A.3 改成 2 维输入、2 类输出时，则为

$(2 \times 30 + 30) + (30 \times 20 + 20) + (20 \times 2 + 2) = 752$ 。用 `sum(p.numel() for p in model.parameters())` 应得到相同数字。只有 `requires_grad` 的参数才应计入“可训练参数”，但被冻结的参数仍占存储。

`super().__init__()` 建立模块注册机制。把层放在 `nn.Sequential` 里，PyTorch 才能递归找到参数、切换模式、移动设备和保存状态。普通 Python 列表不会像 `ModuleList` 那样注册子模块，这是自行组装模型时很隐蔽的错误。

输出是 logits。多分类时，`cross_entropy(logits, labels)` 内部进行稳定的 log-softmax。求预测类别可以直接 `argmax(logits, dim=1)`；softmax 保持大小顺序，因此先转概率再 `argmax` 只增加计算。

2.5 Dataset 和 DataLoader 分别负责什么

上游 `ToyDataset.__len__` 返回样本数，`__getitem__` 返回指定行的特征和标签；`DataLoader` 才决定如何打乱、分批、并行读取。五个样本、批大小 2，会形成 2、2、1 三个批次。`drop_last=True` 删除最后一批，统一大小，但每轮少看一个样本；验证集通常不应默默丢弃尾部。

`shuffle=True` 打乱的是样本访问顺序，并没有修改特征与标签的配对。数据很少时，`num_workers=0` 更容易排错；增加进程可能反而被启动和通信成本拖慢。GPU 数据加载可使用 `pin_memory` 改善特定传输路径，但在 CPU 小实验里不是速度保证。

准确率应累计正确样本数，再除以实际处理的总样本数。对两个大小分别为 2 和 1 的批次，简单平均两批准准确率会给单个样本过高权重。空加载器还会导致零分母，应在训练前检查数据和 `drop_last` 是否把所有批次删掉。

2.6 每一步更新都要有明确先后

完整实现见 `examples/core_training/run_checks.py` 的分类器段。核心顺序如下：

```
optimizer.zero_grad(set_to_none=True)
logits = model(features)
loss = torch.nn.functional.cross_entropy(logits, labels)
loss.backward()
optimizer.step()
```

`backward` 计算当前权重下的梯度，`step` 按优化器规则更新参数。对于 SGD， $\theta_{s+1} = \theta_s - \eta \nabla_{\theta} L_s$ ， η 是学习率。不要在计算 `loss` 后、反向前原地修改参与计算的权重。

本书的小网络为 2→8→2，四个样本分成两个明显分离的类别，学习率 0.2，训练 40 步。实测交叉熵从 0.5594008565 降到 0.0106433481，四个训练样本全部分类正确。这只验证训练闭环，样本太少且太容易，不能用来宣称泛化性能。上游的五训练样本、两测试样本演示也应按同样标准理解。

2.7 保存的是哪些状态

`state_dict()` 保存参数和持久 buffer 的名字及张量。恢复时先建立相同结构，再 `load_state_dict`。只保存权重足以恢复相同前向计算，却不一定恢复训练轨迹；完整续训还需要优化器状态、学习率阶段、已消费的数据位置和相关随机数状态。第 15 章详细展开。

上游使用 `torch.load(..., weights_only=True)` 加载权重。本书也只加载自己刚保存的临时检查点，并把张量映射到 CPU。这个选项缩小反序列化的执行面，不意味着可以不核验任意陌生文件。本书实测保存、恢复后 logits 最大差为 0。

2.8 从单设备到 DDP，代码多出来的责任

第二个 notebook 把模型、特征和标签移到同一个设备。两个待相加张量分别留在 CPU 和 CUDA 会报设备不一致；同一模型的参数和输入也必须相容。`torch.cuda.is_available()` 是运行前的检查，本书环境没有 CUDA 实测，不能把选择 CPU 的分支称作 GPU 测试。

两份 DDP 脚本的主体相同：初始化进程组，每张 GPU 一个进程，给每个进程设置设备，用 `DistributedSampler` 划分样本，把模型包装为 `DistributedDataParallel`，反向时同步梯度，结束后销毁进程组。DDP-script.py 用 `mp.spawn` 启动；DDP-script-torchrun.py 从 `WORLD_SIZE`、`LOCAL_RANK` 或 `RANK` 读取启动器配置，并且不覆盖启动器已经设置的主地址和端口。单机时 rank 与 GPU 编号容易相同，多机时全局 rank 和本地 rank 不能混用。

`sampler.set_epoch(epoch)` 让每个 epoch 使用不同的、跨进程一致的打乱规则。`DataLoader` 此时不能再自己启用 `shuffle`。默认分布式采样可能为凑齐各进程样本数而补重复样本；再加上 `drop_last`，实际训练和评估的样本集合需要核对。上游极小数据对多于两张 GPU 的情况专门给出错误提示，说明“GPU 更多”不自动意味着有更多有效批次。

若各进程批量相同，平均各进程平均梯度等于整体平均梯度。若样本数不同， $\frac{1}{R} \sum_r g_r$ 不等于按样本数加权的全局平均值。准确率同样需要跨进程累加正确数和总数；只打印每个 rank 的训练准确率不是全局指标。本书没有实际启动 DDP，相关脚本属于已解释、未运行的硬件路径。

2.9 运行、练习与答案

在配套代码根目录运行：

```
python examples/core_training/run_checks.py --output qa/core_training/results.json
```

环境需安装匹配的 PyTorch；本书记录的版本和运行日志在配套 QA 中。该命令不下载数据。若已拥有固定上游源码，可以加 `--upstream runtime_sources/upstream` 运行第 6 章的对照。

上游 A.1、A.2 的环境题分别指向 Python 设置和依赖检查；本书第 30 章集中说明。A.3 参数计数答案已在 2.4 推导。A.4 比较 `[100,200]@[200,300]` 的 CPU/GPU 时间，正确测量应预热并同步 GPU，不能直接把异步提交时间当完成时间；本书未执行这项 GPU 比较。

补充题：将训练循环的清零移到循环外，解释为什么第二次梯度不再等于第二个批次的导数；再让 `drop_last=True` 的批大小大于数据量，检查加载器长度。提示：先打印每步更新前的 `.grad` 和 `len(loader)`。答案：梯度会跨步累加，第二次是旧梯度与新梯度之和；大批大小会使加载器没有完整批次。应显式失败而非报告一个没有更新的“训练完成”。

参考与实现覆盖

- [Appendix A Part 1](#)：张量、共享存储、类型、自动微分、网络、数据、训练、保存。
- [Part 2](#)：设备迁移与单 GPU 变体，正文明确未运行边界。
- [DDP 启动脚本](#)：进程组、采样器、模式、准确率和清理。
- [习题代码](#)。上游图片不复用；本书图为原创。

第 03 章 文本怎样变成 token：词表、BPE 与特殊符号

分词器最重要的约定不是“把一句话分成几个词”，而是同一段字节怎样稳定地映射成同一组整数。模型学到的是这些整数对应的规律。训练后随意调换两个 ID，就相当于把模型词典中的两个词交换了位置，即使词表大小没有变，含义也已经错位。

本章覆盖上游 ch02 的主 notebook、压缩版 dataloader、习题，以及六组目录中的 BPE 对照、embedding 对照、滑窗直觉、自写 BPE 和保留空白的 V3。教学实现之间的差异会单独说明，不能把“示例能往返”误写成“兼容所有 Unicode 或 GPT-2”。[固定版本 ch02](#)

3.1 先实现一个会失败的分词器

从 `re.split` 开始很容易观察分词过程。正则表达式中的捕获括号会保留分隔符，因此按空格和逗号切开“青山, 青山。”时，可以同时得到文字、标点、空白和空字符串。去除空字符串合理，但把每个片段 `.strip()` 后再丢掉空白，就永久失去了连续空格、缩进和换行信息。

上游 `SimpleTokenizerV1` 把切好的片段去重、排序，再用 `enumerate` 生成 `str_to_int`；逆向字典 `int_to_str` 用来解码。排序使同一词集合的 ID 分配稳定，但不能保证换语料后旧 ID 不变。`encode` 对每个片段查字典；遇到未见词会抛出 `KeyError`。`decode` 用空格拼接，再用正则删除部分标点前的空格。这是一套排版重建规则，不能恢复任意原文。

V2 添加 `<|unk|>` 和 `<|endoftext|>`。前者让未见词不再报错，却把所有未见词压成同一个 ID；后者标记文档边界。若“青山”和“远海”都不在词表中，它们都会变成未知词，解码时不可能猜回原词。一个没有报错的分词器仍可能严重丢信息。

V3 的改动很小但性质不同：保留每个非空字符串片段，包括空格和换行；解码用空字符串连接。只要所有片段已在词表中，就可精确往返，包括两个连续空格。它仍依赖封闭词表，未知词仍丢失，因此不是字节级通用分词器。[V3 源码](#)

3.2 特殊符号是协议，不是普通装饰

BOS、EOS、PAD、UNK 分别常用于序列开始、结束、补齐和未知片段，但并非每个模型都有这四种独立符号。GPT-2 使用 50,257 项词表，其中 `<|endoftext|>` 的 ID 为 50,256。PAD 与 EOS 是否可以共享 ID，取决于数据和损失掩码怎样处理，不能因为 ID 相同就把真实结束位置一起忽略。

如果一段普通用户文字恰好包含特殊符号的字面字符串，程序需要决定：把它作为普通字符编码，还是解释为控制 token。上游 tiktoken 示例通过 `allowed_special={"<|endoftext|>"}` 显式允许这一项。把任意外部文本都当作允许的控制符号可能改变文档边界或模板含义。相反，不允许时出现错误也不说明文本本身无法编码，而是当前接口拒绝把它解释为特殊符号。

边界 token 告诉模型“这里出现过边界”，不自动阻止注意力看见上一份文档。若任务要求严格文档隔离，还要构造分块注意力掩码。这个区别会影响预训练样本拼接，也会影响后面的指令模板。

3.3 字节与字符的区别先于 BPE

UTF-8 把文本编码成字节序列，每个字节在 0—255 之间。“山”是一个 Unicode 字符，却对应三个 UTF-8 字节；某些表情需要四个字节。因此一个含全部 256 种单字节的词表能表示任意有效 UTF-8 文本，但一个只含 `chr(0)` 到 `chr(255)` 的字符词表并不等价。

本书 ByteBPE 直接保存 bytes 片段，从 256 个单字节开始。解码时先把所有 token 对应的字节拼起来，最后才执行 UTF-8 解码。不要把每个 token 单独解码再拼接，因为一个 token 可能只含某个字符的一部分，单独解码会失败或产生替换字符。

GPT-2 的公开编码器使用 `bytes_to_unicode` 把所有字节可逆地映射到一组可打印字符，避免空白和控制字符干扰字符串处理。这个中间字符不是原文字符。例如显示出来的 `Ġ` 常用于代表空格字节；它只是编码表示，不是要求在用户文本里插入一个拉丁字母。[上游所附 GPT-2 编码器](#)

3.4 训练 BPE：数相邻对，再无重叠替换

给定序列 `[a,b,a,b,a]`，相邻对 `(a,b)` 出现两次，`(b,a)` 也出现两次。需要规定并列时怎样选择，否则不同实现可能学到不同 ID。假设选择 `(a,b)`，分配新 ID `c`，从左向右无重叠替换，得到 `[c,c,a]`。再数相邻对，重复直到达到词表预算或停止条件。

这里的“无重叠”很重要。序列 `[a,a,a]` 有两个相邻 `(a,a)` 位置，但一次替换最多得到 `[c,a]`，不能同时把中间的 `a` 用两遍。计数反映候选频率，替换反映不重叠的实际编码。

本书实现的训练规则是：选择频率最高的相邻对；频率并列时按两个整数的字典序；只合并至少出现两次的对。这样目标词表大小是上限，不保证用满。用 `低山 low low lower low\n低山` 训练，预算 274，实测得到 265 项词表、9 次合并。上游自写 BPE 的停止规则不同，可以继续合并只出现一次的对，不能拿本书的词表大小直接对照其 1,000 项示例。

代码中 `Counter(zip(ids, ids[1:]))` 建立计数，`merge` 同时移动指针一格或两格，新 token 的字节内容是两个旧片段的拼接。完整实现位于 `examples/core_training/core.py`，这些动作都没有依赖预训练词表。

3.5 编码时为什么看合并排名，不重新训练

训练已经决定了哪些相邻对优先合并。给一段新文本编码时，应使用已保存的规则，而不是按这段文本的频率重新学习。否则同一个 token ID 在不同请求里可能表示不同内容。

设已有合并 $(b,c) \rightarrow u$ ，排名 0； $(a,b) \rightarrow v$ ，排名 1。新输入 $[a,b,c]$ 应先合并排名更小的 (b,c) ，得到 $[a,u]$ 。一个从左到右看见可合并对就立刻合并的算法会得到 $[v,c]$ 。两者都可能解码回原文，但 token 序列不同；使用同一预训练模型时，这种差异不能忽略。

本书的 `encode` 每轮取当前相邻对与规则集合的交集，选择最早学到的规则，再合并该对的所有不重叠出现位置。GPT-2 `Encoder.bpe` 也以最低 rank 选择合并，并用缓存避免反复处理相同预分词片段。缓存只提高速度，不应改变编码结果。

GPT-2 在 BPE 前还有正则预分词：分别处理缩写、字母、数字、标点和空白片段，BPE 不跨这些片段合并。因此，仅拥有同一词表和合并规则还不足以保证兼容；预分词、字节映射、特殊符号和解码规则也必须相同。

3.6 上游两种自写 BPE 的用途和实测边界

`bpe-from-scratch-simple.ipynb` 优先展示数据结构：`vocab`、逆词表、`bpe_merges`，训练时用 `find_freq_pair` 和 `replace_pair`。`find_freq_pair` 还接受 `most` 和 `least` 两种模式，非法模式报错；空序列或单项序列没有相邻对，返回 `None`。其编码路径通过 `.split()` 分词，连续空白和换行因此丢失。它从字符而非完整 UTF-8 字节集合构建实际输入表示，未见 Unicode 字符可能报错。

较复杂的 `bpe-from-scratch.ipynb` 增加 `pretokenize_text`，明确保存空格、LF、CR、CRLF；训练在每个预分词片段内计数与合并，避免跨词边界产生难以解释的空格标记。

`load_vocab_and_merges_from_openai` 读取 `encoder.json` 和 `vocab.bpe`，保留官方合并 rank，并验证换行显示字符和结束符号的指定 ID。`encode` 先检查禁用特殊符号，再按允许的符号切分，普通部分递归进入常规编码。按长度递减排列特殊符号可以避免短符号抢先匹配长符号的前缀。

`save_vocab_and_merges` 把词表和合并列表写为 JSON；加载时 JSON 对象键是字符串，必须转回整数。`get_special_token_id` 用缓存查 ID。若同一个对象的词表重新训练或重新加载，缓存及旧合并规则也需要保持一致；最容易审计的做法是创建新对象再加载，而不是假定所有可变状态都会自动重置。

复杂版也不应被当成所有输入上的正式 GPT-2 替代品。本书只抽取已检查的类定义，离线训练一个小词表，运行以下探针，结果见 `qa/core_training/tokenizer_probes.json`：

输入	简单版解码	复杂版解码	说明
<code>a%b\n</code>	<code>a%b</code>	原样恢复	复杂版改善空格和换行
<code>a\tb</code>	<code>a%b</code>	<code>ab</code>	<code>tab</code> 不满足任意空白可逆的承诺

输入	简单版解码	复杂版解码	说明
␣	␣	换行	自训字符 ID 198 与官方换行 ID 的专门处理冲突
未训练字符“山”	报错	报错	字符初始化不等于全字节覆盖

表中的 ␣ 仅用于显示空格。␣ 反例来自复杂版 decode 无条件把 ID 198 当换行，而它从零训练时 `chr(198)` 正好是 ␣。这并不否定演示中英文样例的结果，只说明加载官方词表的特殊约定不能无条件用于自训词表。本书字节版保留了中文、表情、CRLF、连续空格和普通特殊符号字面量的精确往返，但没有实现 GPT-2 正则或特殊控制符号，明确不声称 token ID 兼容。

上游 `tests.py` 检查固定训练词表、预期 token IDs、标点组合、换行、空格和结束符号。其 notebook 导入器会执行全部代码单元，可能触发下载；本书没有直接运行该测试入口。我们仅执行定义和独立探针，下载路径与完整 tiktoken 测试套件属于未运行范围。固定源树已经包含 `encoder.json` 与 `vocab.bpe`，因此本书另用原始 GPT-2 Encoder 离线读取它们，逐项比较六条输入。四条英文、空格与换行样例 ID 相同；中文加表情在原始编码器可往返，教学复杂版报未见字符；`tab` 在教学版丢失。结果见 `source_variant_checks.json`，属于有限探针，不是全量兼容性证明。

3.7 比较分词器时先比较结果，再比较速度

上游对照 notebook 比较原始 Python GPT-2 编码器、tiktoken、Hugging Face 普通与 fast 分词器，以及自写版本。先编码同一短文本、核对 ID 和解码，再用同一篇文本计时。这是合理顺序；输出不同的两个程序不能直接当作同一任务比较速度。

计时前需要确认首次加载、文件下载、缓存预热是否包含在时间内。Hugging Face 的截断选项还会改变处理范围，必须确认返回的 token 数相同。`%timeit` 是 notebook 魔法命令，不能直接放进普通 Python 脚本。上游曾展示某个环境下的速度差，本书没有复测这些依赖和下载路径，不把那个倍数写成普遍结论。

原始编码器的 `get_encoder` 只读取词表与合并文件，`download_vocab` 负责下载；两件事应分开。主 notebook 下载短文时检查本地文件，网络请求设置超时并检查状态。证书失败应修复环境的证书链，不能关闭验证。配套实验完全离线，不需要下载这篇短文或官方词表。

3.8 滑窗负责构造问题和答案

令完整 ID 序列长度为 N ，上下文长度为 T ，步长为 s 。窗口起点为 $i = 0, s, 2s, \dots$ ，满足 $i < N - T$ 。每个样本取

$$X_i = (x_i, \dots, x_{i+T-1}), \quad Y_i = (x_{i+1}, \dots, x_{i+T}).$$

目标比输入的索引大一位。不要只凭“右移”或“左移”的口头说法判断方向，写出索引最清楚。

$N \geq T + 1$ 时，窗口数为 $\lfloor (N - T - 1)/s \rfloor + 1$ 。若只剩 T 个 token，没有最后一个监督标签，不能凑成完整样本。

本书用整数 0—10、 $T = 4$ 、 $s = 3$ ，得到三条输入 [0,1,2,3]、[3,4,5,6]、[6,7,8,9]，对应目标 [1,2,3,4]、[4,5,6,7]、[7,8,9,10]。断言 $X[:,1:] == Y[:, :-1]$ 能检查内部移位，但还应核对每条的最后标签。

Context length 4, stride 3: target index = input index + 1

input i=0 -	0	1	2	3
target i=0 -	1	2	3	4
input i=3 -	3	4	5	6
target i=3 -	4	5	6	7

上游 dataloader 直觉 notebook 把文本换成 0—1000 的数字，暂时绕开分词器，依次观察 stride 1、stride 4、批量与 shuffle。这种隔离变量的方法很有效：数据窗口不对时，先看清楚整数，不必同时排查 BPE。压缩版 dataloader.ipynb 与主版本保持相同 Dataset/Loader 接口；练习版本默认不打乱、不丢最后一批，与主版本默认值不同，比较“第一批”时必须统一设置。

窗口重叠增加相同 token 被训练的次数。它可能有用途，但训练验证划分必须先作用于文档或不重叠文本区间，再在各自内部建窗口。先造所有重叠窗口再随机分到训练、验证，几乎相同的上下文可能落到两边，验证损失会误导。

3.9 Embedding 是可训练的取行操作

词表 V 、嵌入宽度 D 对应矩阵 $E \in \mathbb{R}^{V \times D}$ 。ID i 的向量是第 i 行 E_i 。若用 one-hot 行向量 $e_i \in \mathbb{R}^{1 \times V}$ ，则 $e_i E = E_i$ 。这证明查表与无偏置线性映射在数学上等价；实际查表不需要构造大量零元素。

上游 embedding 对照把 Linear.weight 设为 embedding.weight.T，因为 Linear 内部使用权重转置。仅给两层相同随机种子不够：初始化规则、形状及随机数消费顺序可以不同。比较时应显式复制权重。

反向也等价。若 ID 2 出现两次，损失对 E_2 的梯度是两个位置贡献的和；未被访问的行在这个独立 embedding 例子中梯度为零。若模型把输出头与输入 embedding 共享权重，未作为输入出现的行仍可能从输出 softmax 获得梯度，不能把独立查表结论直接推广。我们的 float64 检查使用 [2,2,6]，查表与 one-hot 前向、梯度误差均为 0。

3.10 加入位置后才知道“在哪儿”

相同 ID 查到相同向量，单靠 token embedding 不区分两个相同词的位置。GPT-2 使用可训练的绝对位置矩阵 $P \in \mathbb{R}^{C \times D}$ ， C 为上下文容量。输入为

$$H_{b,t,:} = E_{X_{b,t,:}} + P_{t,:}.$$

token embedding 形状是 $[B, T, D]$ ，位置向量是 $[T, D]$ ，相加时沿批次广播。位置 ID 应是 `arange(T)`，而不是 token ID。训练时见过的位置有限，直接把位置索引设为 C 会越界；延长上下文需要更改模型或位置机制，不是给 DataLoader 一个更大的整数就行。

3.11 练习、提示与答案

上游练习 2.1 用一个罕见字符串逐项检查分词，说明一个未见词可以由已有子词组成。不要把某次分词输出当成所有分词器的答案。练习 2.2 改 `max_length` 和 `stride`，观察输入窗口。其关键结论可由 3.8 的索引直接验证。

补充题一：构造 $[a, b, c]$ 的两条合并规则，让从左到右合并与按 rank 合并不同。提示：让右侧的合并排名更早。答案就是 3.5 的 (b, c) 与 (a, b) 。

补充题二：为什么单 token 的 UTF-8 解码可能失败，而整个序列能解码？答案：token 边界可以切在多字节字符内部，正确顺序是先拼字节再解码。

补充题三： $N = 11, T = 4, s = 4$ 有几个窗口？答案为 2，起点 0 和 4；起点 8 没有五个剩余 token。

补充题四：给 V3 一个未在训练词表出现过的 tab，能否保证精确往返？不能；保留分隔符还要求词表覆盖该片段。

运行主断言用 `python examples/core_training/run_checks.py`。若已拥有固定源码，运行 `python examples/core_training/tokenizer_probes.py runtime_sources/upstream qa/core_training/tokenizer_probes.json` 可复查上述上游反例；此脚本不执行下载单元。

参考与实现覆盖

- 主 notebook 与习题：正则、词表、特殊符号、滑窗、嵌入、位置。
- BPE 对照和 GPT-2 编码器：字节映射、rank、缓存、文件加载和计时边界。
- embedding 对照、数字滑窗。
- 两版自写 BPE 与测试：已区分自训和官方加载路径，未声明任意 Unicode 兼容。

第 04 章 从加权平均推导自注意力与因果掩码

一个 token 的输入向量只说明它自身的 ID 和位置。要预测续写，还需要根据当前上下文重新组合信息。注意力的基本动作是：当前位置提出一个查询，给各个可见位置打分，把分数转成权重，再加权汇总那些位置的值。

上游 ch03 从没有可训练投影的点积示例，逐步加入 Q、K、V、因果限制、dropout、批次和多头。本章保留这条推导顺序，第 5 章再讨论不同实现怎样保持同一计算。[主 notebook](#)

4.1 先把加权平均写清楚

设三个输入向量为 $x_0 = (1, 0)$ 、 $x_1 = (0, 1)$ 、 $x_2 = (1, 1)$ 。暂时选 x_1 作为查询，点积分数为 $[0, 1, 1]$ 。直接除以分数之和虽然在此例得到和为 1 的数，但遇到负分数或总和为零就失去概率含义。

softmax 对分数 s_j 定义

$$\alpha_j = \frac{e^{s_j}}{\sum_k e^{s_k}}.$$

每项非负、总和为 1。此例得到 $[1, e, e]/(1 + 2e)$ ，约为 $[0.1554, 0.4223, 0.4223]$ 。加权结果是

$$z_1 = \sum_j \alpha_j x_j = (0.5777, 0.8446).$$

这是用上下文重新表示第二个位置，而不是从三个向量中只选一个。代码里的两层循环可被 `scores = X @ X.T` 取代，全部上下文向量则是 `softmax(scores, dim=-1) @ X`。最后一维是被查询的位置，所以沿该维归一化。

若 $X \in \mathbb{R}^{T \times D}$ ，分数形状为 $[T, T]$ ：行是提出查询的位置，列是提供信息的位置。把 softmax 改到列方向不会必然报形状错，但它改变了“每个查询分配总权重为 1”的定义。

4.2 为什么拆成查询、键和值

用同一输入做查询、键和值，限制了“怎样匹配”和“传递什么内容”的表示空间。引入三个可训练矩阵：

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V.$$

这里 X 为 $[T, D_{in}]$ ， W_Q, W_K 为 $[D_{in}, d_k]$ ， W_V 为 $[D_{in}, d_v]$ 。 $QK^\top$ 是 $[T, T]$ ，最终 $Z = AV$ 是 $[T, d_v]$ 。查询与键的宽度必须相同才能点积，值的宽度在一般定义中可以不同；本书多头实现让它们相同，简化布局。

行 q_i 可以理解为当前位置怎样查找相关信息； k_j 决定位置 j 在什么查询下得到高分； v_j 承载要汇总的特征。它们由同一个输入投影而来，因此称作自注意力。交叉注意力则允许查询与键值来自不同序列。名称是帮助追踪计算的，不表示每个维度都能解释成一个人工定义的语义。

上游 SelfAttention_v1 直接保存 [D_in,D_out] 的 Parameter，用 $x @ W$ ；v2 使用 nn.Linear，保存的是转置布局。两版初始化不同，相同 seed 不保证结果相同。上游习题把 v2 的三份权重转置后交给 v1，验证的正是布局转换，而不是两个不同随机模型是否碰巧输出一致。

4.3 缩放因子来自点积的方差

假设查询和键的每个分量近似独立、均值为零、方差为 1。点积 $s = \sum_{r=1}^{d_k} q_r k_r$ 的均值为 0；在这些假设下，每个乘积的方差为 1，和的方差约为 d_k 。宽度越大，分数越容易出现很大的正负差，softmax 越接近极端分布，部分梯度越小。

除以 $\sqrt{d_k}$ 把方差量级拉回 1：

$$A = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right), \quad Z = AV.$$

这不是声称训练中的分量始终独立，而是一个控制初始化尺度的动机。多头注意力要用每个头的宽度 d_k ，不是合并所有头后的总宽度 D 。[上游缩放实现](#)

4.4 因果掩码为什么应放在 softmax 前

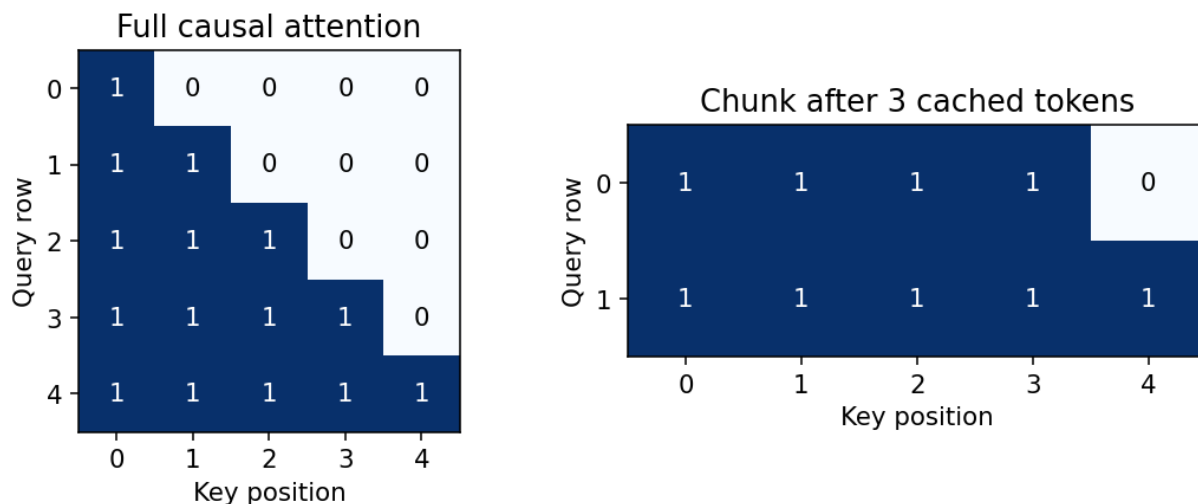
预测位置 i 的下一个 token 时，只允许访问 $j \leq i$ 的位置。定义加性掩码

$$M_{ij} = \begin{cases} 0 & j \leq i, \\ -\infty & j > i. \end{cases}$$

于是 $A = \text{softmax}(QK^\top/\sqrt{d_k} + M)$ 。由于 $e^{-\infty} = 0$ ，被遮住的位置得到零概率，剩余位置自动重新归一化。代码可用 `torch.triu(..., diagonal=1)` 标记严格上三角，再 `masked_fill(..., -torch.inf)`。

如果先 softmax 再把未来位置置零，剩下的权重通常不再和为 1。上游先展示这种中间状态，再除以每行剩余和，最后换成 softmax 前的掩码。精确算术下，后掩码并重新归一化可得到相同结果；实际程序应避免先计算无用的未来概率，且需要处理数值稳定性。

对角线不能一起遮住：位置 0 至少要能看见自己。如果整行全是负无穷，普通 softmax 会出现 NaN。padding mask 与因果 mask 叠加后也可能产生空行，尤其是左 padding。不能只检查掩码形状而忽略每行是否有允许的位置。



图中 1 表示允许参与注意力。左图每一行只能看见对角线及其左边；右图提前展示第 8 章的缓存场景：已有三个位置时，新块第一行可看见四个位置，而不是从第一列重新开始画三角形。

4.5 通过修改未来输入来验收

输出形状正确不代表没有泄漏。更有力的检查是：保持前四个位置不变，把后两个位置加上很大的扰动；前四个输出必须不变。配套实验用 $[B, T, D] = [2, 6, 12]$ ，3 个头，在 float64、评估模式下执行此检查，最大前缀差为 0。

这个测试还可以检查输入梯度：早期位置的损失不应依赖未来输入。需要先关闭 dropout，或者控制同一随机掩码，否则两次前向的随机差异会干扰因果测试。仅把注意力矩阵打印出来“看着像三角形”不如这种行为检查可靠。

4.6 dropout 并不保持每行权重和恰好为 1

上游在 softmax 后加入注意力 dropout。以概率 p 丢弃某项，保留项乘以 $1/(1-p)$ 。每项的期望保持不变，但一次具体采样中的行和可以大于、小于或等于 1。因此，softmax 后、dropout 前检查行和；不要拿训练模式下 dropout 后的权重行和来判定 softmax 有错。

推理时 dropout 应关闭。nn.Dropout 会服从 `model.eval()`；某些函数式接口需要调用者显式把概率设为 0，第 5 章将说明。dropout 不是因果遮挡，两种掩码的目的和生命周期不同。

4.7 加上批次和多个头

批次输入 $X \in \mathbb{R}^{B \times T \times D}$ 。总输出宽度为 D_o ，头数为 H ，要求 D_o 能被 H 整除，每头宽度 $d = D_o/H$ 。Q、K、V 先投影成 $[B, T, D_o]$ ，再改形状为 $[B, T, H, d]$ ，交换中间两轴得到 $[B, H, T, d]$ 。

此时矩阵乘法自动对每个批次、每个头独立进行：

$$S_{b,h,i,j} = \frac{1}{\sqrt{d}} \sum_r Q_{b,h,i,r} K_{b,h,j,r}, \quad Z_{b,h,i,r} = \sum_j A_{b,h,i,j} V_{b,h,j,r}.$$

分数形状是 $[B,H,T,T]$ ，上下文是 $[B,H,T,d]$ 。交换回 $[B,T,H,d]$ ，把最后两维合并为 $[B,T,D_o]$ ，再用输出投影混合不同头的信息。

上游最早的 `MultiHeadAttentionWrapper` 实例化多个独立 `CausalAttention` 并拼接，它的 `d_out` 表示每个头宽度；后来的 `MultiHeadAttention` 中 `d_out` 表示所有头的总宽度。名字相同，含义不同。若照搬数字而不读形状，会不小心把输出宽度扩大 H 倍。高效版同时投影后拆头，与独立头在正确对齐权重时可表达相同计算。

4.8 配套实现怎样对照每一行

`core.py` 的 `Attention` 把三份投影合成一个 `Linear(D,3D,bias=False)`。`chunk(3,-1)` 分成 Q、K、V，再 `reshape` 和 `transpose` 分头。`scores` 用最后两轴乘法，布尔 `future buffer` 截出需要的行列，`softmax` 后乘 V，最后合并头并输出投影。第 8 章解释它额外接受的 `cache` 参数；不传 `cache` 就是本章的普通因果注意力。

压缩版 `multihead-attention.ipynb` 的 Variant A 把单头类命名为 `CausalSelfAttention`，并在 `wrapper` 拼接后额外加入输出投影；主 notebook 最早的 `wrapper` 没有这一层。Variant B 则使用一次投影后拆头的 `MultiHeadAttention`。压缩版还串接第 3 章的数据、token 与位置 embedding，因而能检查完整输入接口。

主检查脚本还直接比较手写注意力和 PyTorch SDPA，使用 Q、K、V 形状 $[2,3,5,4]$ 。实测 float64 最大前向差约 2.78×10^{-16} ，对 Q、K、V 的梯度最大差约 1.33×10^{-15} 。这个结果验证了小 CPU 张量的数学一致性，没有涉及 GPU 融合内核性能。

4.9 练习与答案

上游习题 3.1 是 $v1/v2$ 权重转置对齐，见 4.2；3.2 要让两个独立头拼接后总宽度为 2，所以每头 `d_out` 应为 1；3.3 用 $D=768, H=12$ 构造 GPT-2 尺寸注意力，QKV 无偏置时参数为 $4D^2 + D = 2,360,064$ 。这是单个注意力模块的数量，不是完整模型。下面是补充题。

1. 三个分数全为 0，因果掩码后的三行权重分别是什么？
2. $D_o = 12, H = 3, T = 5, B = 2$ ，列出 Q、分数、合并头输出的形状。
3. 为什么 `softmax(scores, dim=-2)` 可能不报错却不正确？
4. 试把因果掩码的 `diagonal=1` 改成 0。预测第一行会发生什么，再运行一个小例子。

答案：三行分别为 $[1,0,0]$ 、 $[1/2,1/2,0]$ 、 $[1/3,1/3,1/3]$ 。Q 分头后 $[2,3,5,4]$ ，分数 $[2,3,5,5]$ ，合并后 $[2,5,12]$ 。错误 `softmax` 沿查询行归一化，不能保证每个查询对键的权重和为 1。遮住对角线后第一行没有合法键，普通 `softmax` 出现 `NaN`。

参考与覆盖

- 主 [notebook](#): 点积循环、矩阵化、两版投影、掩码的两种推导、dropout、批次、wrapper 和拆头。
- 压缩版实现: 同一 MHA 的集中实现, 映射到 4.7—4.8。
- 习题: 权重布局对照。

第 05 章 多头注意力的实现选择与 buffer 生命周期

注意力公式很短，实现却可能很不一样：逐头 Python 循环、一次性矩阵乘法、einsum、SDPA、完整的 MultiheadAttention 类，以及 FlexAttention。选择实现前，先分清哪些只是布局变化，哪些改变了参数或功能，再谈速度。

本章依据上游高效注意力 notebook 和 buffer notebook。上游大尺寸计时在本书没有运行，所有实测对照均为小型 CPU 张量。[实现集合](#)

5.1 把三次投影合成一次，不改变线性代数

若 Q、K、V 都是 D 维，把三个 PyTorch 权重按输出行拼接：

$$W_{QKV} = \begin{bmatrix} W_Q \\ W_K \\ W_V \end{bmatrix} \in \mathbb{R}^{3D \times D}.$$

Linear($D, 3D$) 输出 $[B, T, 3D]$ ，然后拆成三个 $[B, T, D]$ 。如果偏置存在，也必须按同样顺序拼接。这样减少了分开的调用和部分读写开销，但参数数量没有减少，主要矩阵乘法量也没有消失。

上游 combined-QKV 版本先 `view(B, T, 3, H, d)`，再 `permute(2, 0, 3, 1, 4)`，得到 $[3, B, H, T, d]$ ，最后 `unbind(0)`。本书先 `chunk` 再分头，结果布局相同。错误的 `reshape` 可以让 Q 的某些通道流入 K，即使总元素数量完全匹配。

该上游类最终用输入的 `embed_dim` 合并输出，示例中 `d_in=d_out` 因而成立。如果尝试让输入宽度与输出宽度不同，应该合并成 `d_out`，不能继续假定二者相等。这是从“示例可用”推广为“通用模块”时需要补的检查。

5.2 einsum 让索引关系写在代码里

上游 MHAEinsum 用 `bnd, do->bno` 表示投影：批次 b 、位置 n 保留，输入维 d 求和，输出维 o 保留。分数用 `bhnd, bhmd->bhnm`，对每头特征 d 求和；加权汇总用 `bhnm, bhmd->bhnd`，对键位置 m 求和。

这几行清楚表达了与第 4 章同一计算，但 einsum 不自动保证更快。底层如何安排乘法与内存访问依赖形状和实现。上游用 Parameter 保存 `[d_in, d_out]` 权重，再自定义初始化；而 Linear 的权重布局相反，`fan-in` 的解释也与轴布局有关。因此比较时必须显式复制参数，不能只固定 `seed`。

如果启用 QKV 偏置，einsum 版在投影之后加 `[d_out]` 向量；禁用时用 `register_parameter(name, None)`。这保留了统一接口，同时不把不存在的偏置计入参数数量。

5.3 SDPA 的 mask 与 dropout 约定

`scaled_dot_product_attention` 接受已经分头的 Q、K、V，内部完成缩放、掩码、softmax 和加权汇总。使用 `is_causal=True` 可以表达普通方形因果注意力。本书对照测试的数学式与该接口在 CPU 上一致。

最危险的差异是布尔 mask 的含义。手写 `masked_fill(mask, -inf)` 把 True 当“禁止”；SDPA 的布尔 `attn_mask` 把 True 当“允许”。上游显式 mask 版本因此传入 `~attn_mask`。`nn.MultiheadAttention` 的布尔 mask 又使用 True 表示禁止。把同一个变量原样传给不同接口，很可能把可见范围颠倒。

函数式 SDPA 的 `dropout_p` 不会自动根据外层模块的 `eval()` 猜测意图，上游明确写 `0. if not self.training else self.dropout`。不这样做，评估模式仍可能采样 dropout，破坏保存恢复和 cache 等价检查。

上游把一个显式 mask 变体命名为 `WithoutFlash`。这个类名只能说明作者比较的路径，不能作为所有 PyTorch 版本、所有设备上禁用某个内核的保证。实际后端选择还依赖数据类型、形状、mask 支持及运行库；要报告用了哪个内核，需要后端控制或 profiler 证据。本书只报告 CPU 数值等价，不把接口名称当作 `FlashAttention` 运行证据。

5.4 完整 MultiheadAttention 类不只是公式包装

上游 `MHAPyTorchClass` 内部使用 `nn.MultiheadAttention(batch_first=True)`，输入保持 `[B,T,D]`，把同一 `x` 作为 query、key、value。设置 `need_weights=False` 可以避免请求返回注意力权重，也可能开启更合适的执行路径；是否加速要测量。

还要留意功能差异：这个包装类在 `nn.MultiheadAttention` 自带输出投影之后，又加了一层 `proj`。它不自动与只有一层输出投影的手写版本具有相同参数量。若 `qkv_bias=True`，代码还打开 `add_bias_kv`，这会引入额外键值偏置项，不能当成普通 QKV 线性层偏置的完全等价开关。比较“相同公式的速度”时，应先消除这些额外功能，或者明确承认比较的是不同模块。

5.5 FlexAttention 表达规则，执行条件另算

上游 `causal(b,h,q_idx,kv_idx)` 返回 `q_idx >= kv_idx`。`create_block_mask` 把这个位置规则组织成 block mask，再交给 `flex_attention`。与显式存一个完整三角矩阵相比，这种接口更方便表达复杂可见范围；但它的可用设备、版本和编译支持需要独立核验。

固定版本 notebook 对 PyTorch 版本做标准化，去掉构建后缀后比较最低版本，并用 CUDA 可用性保护导入及实例化。某些后面的独立 `%timeit` 单元没有同样的保护，在 CPU 环境逐个执行全部单元仍可能引用未定义对象。正文中的兼容注释也不应替代实际安装版本的检查。

该实现接收 dropout 参数，但当前 forward 没有把它应用到 FlexAttention 路径。示例用 0 因而没有这项差异；若把它改成非零，不能认为自动获得了与其他版本相同的训练正则化。新版分块掩码接口变化时，上游选择按当前序列长度重新构造 mask，而非切片旧对象。本书没有执行 FlexAttention 或 GPU 编译测试。

5.6 buffer 解决的是模块状态管理

因果 mask 不是需要优化器更新的参数，却是模型计算所需的张量。如果只是 `self.mask = tensor`，它是普通属性，`model.to(device)` 不一定移动它，`state_dict()` 也不会保存它。结果可能是权重已在 GPU，而 mask 留在 CPU，直到前向才报错。

`register_buffer("mask", tensor)` 把它注册为 buffer。持久 buffer 会跟随设备迁移，进入 `state_dict`，但不会出现在 `named_parameters()` 里。`persistent=False` 可用于可重建而不希望保存的状态；它仍参与设备迁移。是否持久和是否训练是两个独立问题。

上游 buffer notebook 分别构造无 buffer 和有 buffer 的 CausalAttention，先观察设备，再手动迁移普通属性，最后把 mask 中的 1 改成 2 并保存恢复。由于前向会把 mask 转布尔，1 和 2 都表示禁止，修改是为了检查状态是否被保存，而不是改变因果含义。无 buffer 的新对象会重新得到初始 mask；有 buffer 的对象会加载修改后的值。[buffer 实验](#)

本书直接保存布尔 mask，避免每次 `.bool()` 转换和浮点存储。实验确认 future 在 `state_dict` 中、不在参数字典中，dtype 为 bool。缓存也可能注册为非持久 buffer，但它与静态 mask 不同：缓存属于某次请求，必须在新请求开始时清空，并正确处理 batch、设备和 dtype 变化。

5.7 怎样测量而不混淆问题

上游依次比较前向、前后向以及 `torch.compile` 后的前后向。CUDA 计时函数先预热，再记录 CUDA Event，结束后同步，收集均值和标准差；绘图函数把各模块结果画成带误差条的柱状图。相比直接用 Python 时钟围住异步提交，这更接近设备执行时间。

不过，前向是否记录计算图、输出是否保留、反向是否清空参数梯度，都应保持一致。上游 `forward_backward` 清理输入的梯度，但没有在每次重复前清理各模块参数的梯度；参数梯度会累加。它仍能做某种重复 backward 的计时，却不是完整优化器步骤，也不能把结果写成端到端训练吞吐。`torch.compile` 的首次编译时间需要与稳定态时间分开；设置 `suppress_errors` 允许回退，更不能仅凭“不报错”断定整段图已经编译。

`set_float32_matmul_precision("high")` 是精度/性能选择，不代表任何 CPU 都拥有或启用了 Tensor Core。应记录设备、dtype、批量、长度、宽度、头数、模式、是否反向、是否同步，以及编译状态。内核优化还可能改变浮点求和顺序，所以性能检查应伴随误差和梯度检查。

5.8 一张比较表，明确各自代价

实现	主要变化	容易遗漏的条件	本书验证
多头 wrapper	每头独立模块，拼接	d_out 是每头宽度	解释与形状推导
拆头 MHA	一次大投影后 reshape	总宽度能被头数整除	微型 GPT 对照
合并 QKV	三投影合成一投影	QKV 顺序、偏置、输入输出宽度	参数映射对照
einsum	显式写求和索引	初始化与权重布局	三组缩小尺寸对照
SDPA	库实现核心注意力	mask 极性、dropout、后端	CPU 前向和梯度
MultiheadAttention	更完整的层接口	额外投影、返回权重、bias-kv	静态解释
FlexAttention	用规则构造块掩码	版本、设备、未应用的 dropout	未运行硬件路径

5.9 练习与答案

以下为补充题。给 masked_fill 的禁止 mask 是严格上三角，改用 SDPA 时应传什么？提示：看 True 的含义。答案是取反后的允许 mask，并且不要同时误用不相容的 causal 参数。

第二题：模型 .to(float64) 后，一个普通浮点属性和一个浮点 buffer 会怎样？答案：注册的参数和浮点 buffer 跟随转换，普通属性不会自动受管理；布尔 buffer 仍应保持布尔类型。

第三题：为什么两版 MHA 都输出 [2,5,12]，仍可能不能公平比速度？答案：输出形状没有说明参数量、额外输出投影、dropout、返回权重和反向功能是否相同。先对齐这些条件，再复制权重和比较数值。

上游 tests/test_mha_implementations.py 明确复制并转置三份权重，比较相等、变宽、变窄三组输入输出尺寸，还检查显式 SDPA mask 的未来不泄漏。本书用更小的三组尺寸复查：12→12、8→12、12→8，float64 最大误差均小于 1.4×10^{-16} ，显式 SDPA 前缀扰动误差为 0。结果在 source_variant_checks.json；没有执行其较大默认尺寸。

主实测命令仍是 python examples/core_training/run_checks.py。其中 SDPA 对照最大前向误差 2.78×10^{-16} ，梯度误差 1.33×10^{-15} 。这里没有 GPU 加速倍数。

参考与覆盖

- 高效 MHA 完整 notebook：九种比较路径、版本检查、计时、绘图、编译包装。
- buffer 生命周期 notebook：普通属性、注册状态、设备迁移与保存恢复。
- 本章没有复用上游图、没有运行其大张量 CUDA 计时，性能断言与数学一致性检查已分开。

第 06 章 把注意力组装成 GPT：归一化、残差与输出头

到这里，注意力已经能把上下文混合成新的向量。但仅有注意力还不是 GPT。我们还需要稳定特征尺度、提供逐位置的非线性变换、让多层信息能够传递，并把最后的向量转换成整个词表上的分数。

上游先搭 `DummyGPTModel`，用不做任何计算的占位 block 和归一化检查接口，再逐项替换。这种顺序能提早发现维度问题，但占位模型只验证外壳，不具备真正 Transformer 的计算能力。[固定版本 ch04 主代码](#)

6.1 配置是一组互相约束的尺寸

定义词表 V 、上下文容量 C 、隐藏宽度 D 、头数 H 、层数 L 。要求 $D \bmod H = 0$ ；输入 ID 小于 V ；位置小于 C 。上游小型 GPT-2 配置使用 $V = 50257, C = 1024, D = 768, H = 12, L = 12$ ，但这里的名称不意味着已经载入 GPT-2 权重。

本书从小配置开始： $V = 16, C = 16, D = 16, H = 4, L = 2$ ，dropout 为零、QKV 无偏置、输出头不共享权重。输入 $[2, 6]$ 应得到 $[2, 6, 16]$ 的 logits。先固定这些约定，参数对照才有意义。

6.2 LayerNorm 沿最后一个特征轴工作

对一个位置的向量 $x \in \mathbb{R}^D$ ，定义

$$\mu = \frac{1}{D} \sum_{j=1}^D x_j, \quad \sigma^2 = \frac{1}{D} \sum_{j=1}^D (x_j - \mu)^2, \quad \text{LN}(x)_j = \gamma_j \frac{x_j - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta_j.$$

$\gamma, \beta \in \mathbb{R}^D$ 是可训练缩放和偏移，初始为 1 和 0。 ϵ 防止方差为零时除零。对 $[B, T, D]$ ，均值和方差形状为 $[B, T, 1]$ ，不同批次和不同位置各自归一化，不跨 token 混合。

例如 $x = [1, 3]$ ，均值为 2，总体方差为 1；不考虑 ϵ 时归一化结果是 $[-1, 1]$ 。若用无偏样本方差，分母是 $D - 1$ ，此例方差变成 2，结果不同。上游最初的探索单元使用 `var` 默认设置观察统计量，正式 `LayerNorm` 则明确 `unbiased=False`。检查正式层必须使用同一约定。

加入 ϵ 后，归一化方差通常只是接近 1；再经过可训练 γ, β 后，输出也不必均值 0、方差 1。不要在训练后强行断言最终输出满足初始化时的统计量。

6.3 GELU 与逐位置前馈网络

GELU 的常见定义为 $x\Phi(x)$ ，其中 Φ 是标准正态分布的累积分布函数。上游使用 `tanh` 近似：

$$\text{GELU}(x) \approx \frac{x}{2} \left[1 + \tanh \left(\sqrt{\frac{2}{\pi}} (x + 0.044715x^3) \right) \right].$$

ReLU 把所有负数截成零；GELU 允许部分负输入产生小的负输出，过渡更平滑。上游绘图比较两者的曲线是帮助理解激活函数，不是训练效果对比。模型中选择哪种激活还涉及预训练权重兼容，不能随意替换后继续声称同一模型。

前馈网络把每个位置从 D 扩展到 $4D$ ，经过 GELU，再投回 D ：

$$\text{FFN}(x) = \text{GELU}(xW_1 + b_1)W_2 + b_2.$$

数学布局下 W_1 为 $[D, 4D]$ 、 W_2 为 $[4D, D]$ 。它对每个 token 独立处理，相同参数沿批次和位置共享。跨 token 交互发生在注意力层，FFN 则变换每个位置已经汇总的特征。

上游 GELU 在前向内创建 `torch.tensor(2.0 / torch.pi)`，默认浮点类型会影响常数精度。本书使用 `nn.GELU(approximate="tanh")`。两者公式一致，但 float64 对照不必逐位相同；我们的误差阈值明确考虑了这一点。

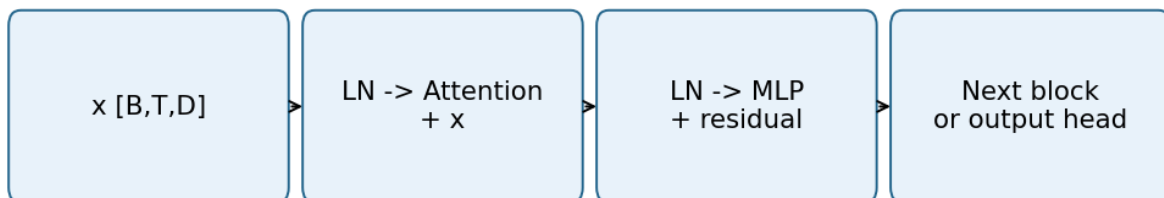
6.4 残差提供一条直接传递的路径

若层输出为 $y = x + f(x)$ ，其雅可比为

$$\frac{\partial y}{\partial x} = I + \frac{\partial f}{\partial x}.$$

反向传播多了一条恒等路径，有助于避免所有信号都必须穿过一串变换。这不是“梯度永远不会消失或爆炸”的保证，因为 f 的导数、尺度和多层乘积仍会影响结果。

上游 `ExampleDeepNeuralNetwork` 在输入输出形状相同的层启用 `shortcut`，再用 `print_gradients` 比较各层权重的平均绝对梯度。最后从宽度 3 变成 1 的层不直接相加；若想跨维度残差，需要额外投影。比较时使用相同 `seed` 和架构，尽量只改变是否加残差。重复调用梯度打印函数前还应清零，以免把旧梯度混入观察。



Pre-normalization residual block; each addition requires matching shapes

6.5 一个完整 block 的先后顺序

上游使用 pre-norm:

$$u = x + \text{Dropout}(\text{MHA}(\text{LN}_1(x))), \quad y = u + \text{Dropout}(\text{FFN}(\text{LN}_2(u))).$$

两个归一化层各有自己的参数。注意力和 FFN 输出都回到 $[B, T, D]$ ，才能与残差相加。不能把同一份旧 shortcut 一直加到两个子层上；FFN 的残差应该是更新后的 u 。

主配置用同一 `drop_rate` 控制 embedding、注意力权重和残差分支上的 dropout。上游练习把它们拆成 `drop_rate_emb`、`drop_rate_attn`、`drop_rate_shortcut`，增加调参独立性。拆配置键时，构造器和调用处必须一起改，否则某一条路径可能继续使用旧值。

6.6 从 ID 到 logits 的完整路线

GPTModel 先查 token embedding，加位置 embedding，经过 embedding dropout，再串联 L 个 block。最后 LayerNorm 后，用无偏置线性层从 D 投影到 V 。隐藏状态保持 $[B, T, D]$ ，只有输出头把最后一维变成词表大小。

本书 TinyGPT 用 ModuleList 是为了第 8 章逐层传递缓存，普通上游版本用 Sequential，因为每个 block 只接受一个 x 。两者都能注册子模块。最终 logits 不需要在模型内 softmax：训练损失自己完成稳定归一化，贪心生成只需要 argmax。

输入 embedding 与输出头权重都有 $[V, D]$ 形状，可以共享同一个 Parameter。但“从参数总数里减去输出头数量”只是估算共享后的计数，不会真正建立共享关系。真实共享需要两个属性指向同一参数；如果只是复制数值，它们仍会在训练后各自变化。上游主代码默认独立，打印了考虑共享时的数量供比较。

6.7 用参数数量检查有没有漏层

在 QKV 无偏置、注意力输出有偏置、FFN 有偏置的情况下，每个 block 有：

- QKV 与输出投影： $3D^2 + (D^2 + D) = 4D^2 + D$ ；
- FFN： $D(4D) + 4D + (4D)D + D = 8D^2 + 5D$ ；
- 两个 LayerNorm： $4D$ 。

合计 $12D^2 + 10D$ 。再加 token embedding VD 、位置 embedding CD 、最后 LayerNorm $2D$ 、独立输出头 VD ，得到

$$N = 2VD + CD + L(12D^2 + 10D) + 2D.$$

代入小配置： $2 \times 16 \times 16 + 16 \times 16 + 2(12 \times 16^2 + 10 \times 16) + 2 \times 16 = 7264$ ，与程序计数一致。如果 QKV 也带偏置，每层再加 $3D$ ；如果共享输出头，减去一份 VD 。因果 mask 是 buffer，不计入可训练参数，但占用内存。

上游练习还要求比较注意力与 FFN 参数、配置 small/medium/large/xl。其 `get_config` 复制基础字典后改宽度、层数和头数；未知名字抛错。大配置可以用上述公式估算，无需为了数参数就实际分配所有张量。

6.8 与上游对齐，误差应该怎样解释

配套 `run_checks.py --upstream runtime_sources/upstream` 读取固定 `gpt.py`，只执行六个经过检查的类定义，不运行下载、分词或大模型 `main`。然后把本书 token/position embedding、LayerNorm、FFN、输出头逐项复制到上游模型，把合并 QKV 权重沿输出轴拆成三份。

在相同小配置、float64、dropout 0、输入 [2,6] 下，最大 logits 差为 2.3896×10^{-9} ，通过 10^{-7} 阈值。主要已知差异是前述 GELU 常数精度。这个检查比比较随机生成的字符串更强：两个 logits 不同的模型仍可能碰巧选中同一个最大项。

上游 `tests.py` 对大默认配置的完整终端输出做字符串回归，先规范换行和行尾空白。它能发现输出格式或随机序列变化，但高度依赖版本、初始化和生成设置。本书不直接运行该大配置测试，也不复制其随机乱码作为本书结果。

6.9 生成循环是外层程序

`generate_text_simple` 每次截取最后 `context_size` 个 ID，前向后取最后位置 logits，`argmax` 得到 [B,1]，拼回序列。主 notebook 先 `softmax` 再 `argmax`，集中脚本省去 `softmax`，二者在有限、非 NaN logits 下的最大位置一致。

随机初始化模型也能走完整个循环，但生成的文本没有语言能力证据。调用前应 `model.eval()` 并用 `no_grad`。上下文截断改变输入窗口，且绝对位置每次从零编号；这与“无限延续的缓存”不是同一个计算，第 8 章必须单独处理。

6.10 练习与答案

补充题： $D = 8, L = 1, V = 20, C = 10$ 、其余约定同本章，参数多少？答案为 $2 \times 20 \times 8 + 10 \times 8 + (12 \times 64 + 80) + 16 = 1264$ 。若共享输出头，减去 160，得到 1104。

第二题：为何位置 embedding 不能与 token embedding 混成同一个索引表？提示：同一 token 可出现在多个位置。答案：两个表分别表示内容与位置，输入需要同时包含两项；混用一个 ID 无法表达两个独立索引。

第三题：把 LayerNorm 方差从除以 D 改为除以 $D - 1$ 后，输出仍是同一形状，是否可以直接载入原权重？答案：载入可能不报错，但计算已变，不能声称原模型等价。应检查实际 logits 和任务效果。

参考与覆盖

- [ch04 主 notebook](#)：占位骨架、LayerNorm、GELU 图、FFN、shortcut、完整 GPT、计数与生成。
- [集中脚本 gpt.py](#)：本书小配置对照目标；其中复用的数据和注意力代码见第 3—4 章。
- [习题与变体](#)：模块计数、四档尺寸、三种 dropout。

第 07 章 参数、FLOPs、显存与吞吐怎样估算

“模型多大”至少可能指四件不同的事：参数数量、权重文件字节数、一次计算的浮点运算量，以及运行时峰值内存。它们有关联，却不能互相替代。一个参数不变的模型，序列长度翻倍后，注意力中间矩阵可能变成四倍。

上游性能 notebook 使用 thop 做运算计数、搜索可容纳的批大小，再估算 GPU 利用率。本章先推导可核验的预算，再解释这些工具结果的适用范围。没有运行其大型模型、OOM 搜索或 GPU 性能表。[固定版本性能分析](#)

7.1 从一个矩阵乘法开始记账

矩阵 $A \in \mathbb{R}^{m \times k}$ 乘 $B \in \mathbb{R}^{k \times n}$ ，输出 mn 个元素，每个元素做 k 次乘法和约 k 次加法。常用近似是 $2mkn$ FLOPs。若工具报告 MACs，即乘加对数，按这一约定乘 2 才得到 FLOPs。精确标量加法数是 $mn(k-1)$ ，但大矩阵估算通常忽略这个小差异。

FLOPs 是运算总数；FLOP/s 是每秒运算率。两者混写会让公式失去量纲。一次批量前向需要多少 FLOPs，与硬件每秒能完成多少 FLOPs 是不同问题。

7.2 一个普通 Transformer block 的主要运算

沿用 B 批量、 T 序列长度、 D 宽度、 H 头数，每头 $d = D/H$ 。忽略偏置、归一化、激活和 softmax 的较小项，主要矩阵乘法为：

部分	运算量近似
Q、K、V 投影	$6BTD^2$
注意力输出投影	$2BTD^2$
两个 FFN 投影，扩展到 $4D$	$16BTD^2$
$QK^\top$	$2BHT^2d = 2BT^2D$
AV	$2BT^2D$

每层约为 $24BTD^2 + 4BT^2D$ 。 L 层再乘 L ；输出头若为每个位置计算全词表 logits，还需要约 $2BTDV$ 。embedding 查表主要是访存，不适合当作密集矩阵乘法计数。

因果掩码只允许一半左右的位置，但朴素 dense 矩阵乘法仍可能计算整个方阵。只有具体内核确实跳过相应块时，才可以把实际计算按三角结构折减。不要因为 mask 是三角形就自动把上式除以二。

对本书 $B = 2, T = 6, D = 16, L = 2, V = 16$ ，主矩阵前向预算为

$$2(24 \times 2 \times 6 \times 16^2 + 4 \times 2 \times 6^2 \times 16) + 2 \times 2 \times 6 \times 16 \times 16 = 162816.$$

这是本章公式计算的估算，不是 profiler 实测。它排除了 softmax、LayerNorm、激活和内存搬运，也不能直接换算为某台 CPU 的耗时。

7.3 为何反向常粗估为前向的两倍

对 $Y = XW$ ，反向需要 $dX = dYW^\top$ 与 $dW = X^\top dY$ ，两次与前向同量级的矩阵乘法。因此在主要成本来自这些乘法时，反向常近似为前向的两倍，训练前后向合计约三倍。

这不是每个算子的精确规律。冻结参数时可能不需要部分权重梯度；激活重计算会增加额外前向；优化器更新、通信、数据加载也不在这个三倍因子内。常见的每 token 约 $6N$ 训练 FLOPs 估算，把主要线性层成本与参数数目联系起来，长上下文注意力项和大输出头需要另行检查。

7.4 权重字节数只是内存的第一项

若有 N 个参数，每个元素 s 字节，权重占用约 Ns 。float32 是 4 字节，float16/bfloat16 通常是 2 字节。本书模型 $N = 7264$ ，纯 float32 参数为 29,056 字节。除以 1024^2 得到 MiB；除以 10^6 得到 MB，不应把两个单位混在一起。

训练还需要梯度、优化器状态、保存给反向的激活、临时工作区和分配器保留空间。以全 float32 AdamW 的简化账本为例：权重、梯度、两个动量各一份，共约 $16N$ 字节，尚未计激活。混合精度可能另外保留主权重，具体要看实现，不能只用“半精度权重两字节”推断总训练内存减半。

显式注意力分数是 $[B, H, T, T]$ ，元素数 BHT^2 。若 $B = 2, H = 12, T = 1024$ ，一份 float32 分数张量就是 $2 \times 12 \times 1024^2 \times 4$ 字节，约 96 MiB；训练可能还保存概率或 dropout 相关状态。融合注意力通过改变存储和计算安排，可能减少这类中间张量，但不能在未运行相应后端时报告具体节省量。

7.5 thop 的数字要问“数到了哪些算子”

上游 `profile(model, inputs=...)` 返回 MACs 和参数数，再用 $2 \times \text{macs}$ 换算。基于模块 hook 的工具对已注册的 Linear 等模块通常容易计数，但模型里直接写的矩阵乘法、自定义模块、某些函数式算子可能缺少对应处理。模型返回一个数字，不等于所有算子都已计入。

验证方法是先用一个小模块手算，再比较工具结果；特别核对 $QK^\top$ 与 AV 的 T^2 项是否随长度增长出现。若只捕捉线性层，结果会过分接近随 T 线性增长。上游实验还把模型转为 bfloat16；运算计数不因每个数更窄就减半，变化的是数据类型和可能的硬件吞吐。

7.6 二分搜索最大批量能证明什么

上游从可选批量区间中取中点，尝试创建输入、模型并前向计数；成功就扩大下界，捕获 out-of-memory 则缩小上界，同时释放引用和清理缓存。它搜索的是当时那个执行路径、那个设备状态下能否完成，不是所有训练场景的最大批量。

只做前向的最大批量不等于加上 backward、优化器状态、验证和生成后的最大批量。`empty_cache()` 释放的是分配器未被张量引用的缓存，不能删除仍活着的计算图。若异常不是 OOM，应继续抛出，不能把 dtype 不支持或形状错误也当内存不足。

在共享或服务环境反复逼近 OOM 还会影响其他任务。本书不执行这条路径，读者应先用保守估算和小批量检查，再在自己控制的设备上测量，并为运行波动留余量。

7.7 吞吐和 MFU 的量纲检查

若一步处理 BT 个 token，耗时 Δt 秒，吞吐是 $BT/\Delta t$ token/s。若每 token 估算需要 f FLOPs，设备针对相同精度和算子类型的峰值为 P FLOP/s，则理想吞吐为 P/f ，模型 FLOPs 利用率为

$$\text{MFU} = \frac{BT/\Delta t}{P/f} = \frac{BTf}{\Delta t P}.$$

分子必须与计时范围一致。若计时包括前后向，却只给分子填前向 FLOPs，MFU 会被低估；若峰值来自另一种精度、稀疏模式或不同 GPU 型号，比例也没有可比意义。

上游 `get_gpu_model` 用设备名称子串查硬编码表，再计算这几个量。表格是实验设置，不是当前硬件规格的权威保证；尤其不能因为字典里给某种 dtype 填了一个值，就推断对应硬件支持它。正式报告应查厂商对应型号、精度、稀疏/稠密条件的规格，并记录计数器覆盖。本书没有据此发布 GPU MFU 数字。

7.8 用预算决定下一步，而不代替测量

当 T 较小、 D 较大，线性层的 TD^2 项可能占主导；当 T 很长，注意力的 T^2D 项和中间存储会越来越重要。生成时每次只计算一个新位置，成本结构又会改变，下一章的 KV cache 正是利用这一点。

预算最有用的用途是排除明显不可能的方案、解释性能随尺寸变化的方向，并设计测量范围。它不能替代真实吞吐、延迟分布和峰值内存。单用户首 token 延迟、逐 token 延迟和多请求总吞吐也应分别报告。

7.9 练习与答案

补充题一：保持其他量不变， T 翻倍时，线性投影和显式注意力分数分别增长多少？答案：前者两倍，后者四倍。

补充题二：一个工具报告 10 GMAC，一次前向用了 0.1 秒。按本章约定，前向运算率是多少？答案：20 GFLOPs / 0.1 秒 = 200 GFLOP/s；不是 200 GFLOPs。

补充题三：纯权重占 2 GiB，为什么 4 GiB 显存可能仍无法训练？答案：梯度、优化器状态、激活、临时空间和框架开销还未计入。必须给完整执行路径做账。

参考与覆盖

- [FLOPs notebook](#)：四档配置、thop、二分批量、异常清理、设备名称识别、同步计时、每 token FLOPs 与 MFU。
- [依赖清单](#)：性能工具依赖，未在本书共享环境新增安装。
- 参数公式已与第 6 章实际模型计数核对；本章 FLOPs 为显式近似推导，GPU 表格和性能测试未运行。

第 08 章 KV cache：先证明等价，再讨论加速

假设模型已经读过六个 token，现在只新增第七个。没有缓存的生成循环会重新计算前六个位置的所有层。因果模型中，前六个位置看不到第七个，它们原先的表示不会因为追加未来 token 而改变。KV cache 保存的正是这些可以复用的中间结果。

这个结论有条件：权重不变、位置编号一致、可见上下文不变、关闭训练随机性。缓存是推理优化，不应在每次更新权重后的普通训练批次之间继续复用旧 K/V。上游 KV cache 目录

8.1 为什么可以按层复用

第一层某个旧位置的输入只由该 token 与其位置决定，追加 token 不改变它。因果注意力只汇总它此前的位置，所以第一层旧输出不变。假设第 ℓ 层所有旧位置的输入不变，它们的 Q、K、V 和输出也不变；由归纳可知，各层旧位置都无需重新算。

生成新位置时，需要它自己的查询 q_t ，以及该层所有可见位置的键和值：

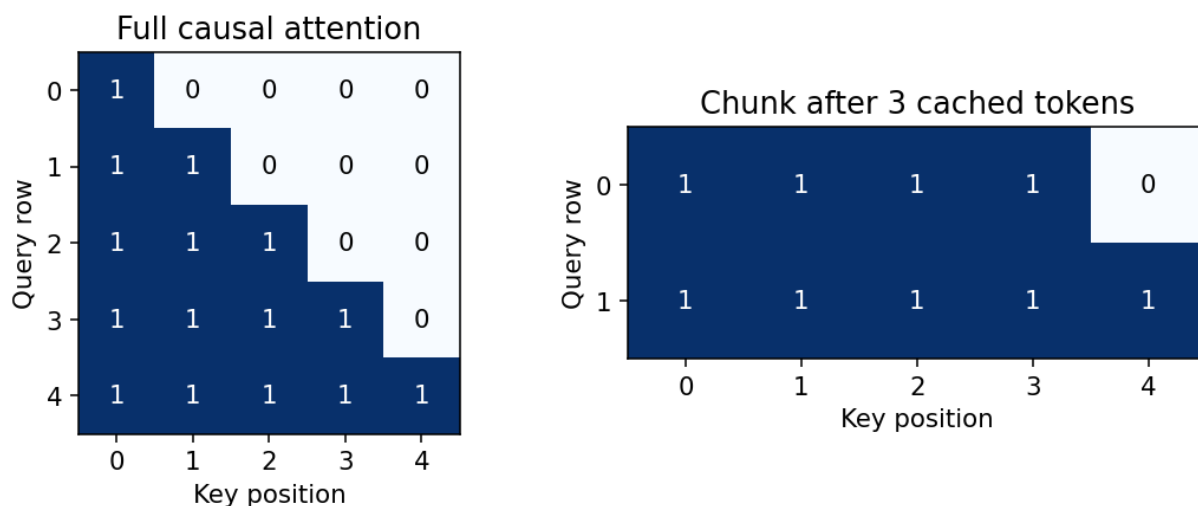
$$a_t = \text{softmax}(q_t K_{\leq t}^\top / \sqrt{d}), \quad z_t = a_t V_{\leq t}.$$

旧查询不参与新输出，因此通常缓存 K 和 V，而不缓存 Q。每层必须有自己的 K/V，不能拿第一层缓存供后面所有层使用，除非架构本身专门设计了共享机制。

8.2 prefill 与 decode 的形状不同

prefill 一次处理提示词 P 个位置，Q、K、V 都可为 $[B, H, P, d]$ 。随后单 token decode 中，Q 为 $[B, H, 1, d]$ ，K/V 则为 $[B, H, P+1, d]$ 。分数矩阵从方形变成 $[B, H, 1, P+1]$ ，只计算新增查询这一行。

若一次新增一个长度为 C_n 的块，已有缓存 P ，查询形状是 $[B, H, C_n, d]$ ，键值长度是 $P + C_n$ 。块内第 i 行允许看见键 $j \leq P + i$ 。直接对 $[C_n, P+C_n]$ 用从左上角开始的普通三角 mask，会错误地遮住大部分历史上下文。



图右已有三个 token，新增两行：第一行看见四列，第二行看见五列。这个偏移是最值得单独写断言的地方。

8.3 上游朴素版怎样保存状态

`gpt_with_kv_cache.py` 在注意力层注册 `cache_k/cache_v` 为非持久 buffer，初始为 `None`。每次前向只对新输入做投影；第一次直接赋值，后面沿 token 轴 cat。其缓存先保存 $[B, T, H, d]$ 布局，再转置成计算所需 $[B, H, T, d]$ 。

每层 `ptr_current_pos` 用来选择原三角 mask 的对应行；模型级 `current_pos` 则用来选择位置 embedding。两个指针含义相关但责任不同：一个决定可见关系，一个决定输入位置特征。只更新其中一个会让缓存结果偏离完整前向。

`TransformerBlock.forward` 增加 `use_cache` 参数并向注意力传递；模型不能再直接用一个只接受 `x` 的 `Sequential` 调用，而要显式遍历 block。`reset_kv_cache` 遍历各层清除 K/V 和指针，并清空模型位置计数。新请求必须调用它；同一权重对象服务两个请求时，不能让它们共享未隔离的状态。

8.4 本书采用显式输入、输出缓存

`core.py` 不把请求状态藏在模型对象里，而让 `forward(ids, caches, return_cache=True)` 返回新缓存。每层缓存是一对 $[B, H, T, d]$ 张量，模型通过已有缓存长度确定位置偏移。这样调用者能清楚看到哪个请求持有哪份状态。

核心片段的含义是：

```
past = cache[0].shape[2]
k = torch.cat((cache[0], k_new), dim=2)
v = torch.cat((cache[1], v_new), dim=2)
mask = future[past:past + new_length, :past + new_length]
```

这仍是便于理解的拼接实现，不是内存分配最优的服务实现。调用者应在 no-grad 下推理，避免把历史反向图也保留下来。本书没有支持异步并发请求、不同长度 batch 的独立指针或自动分页缓存；这些不能仅靠当前接口名称推断出来。

8.5 预分配减少重复分配，却引入容量管理

上游优化版预分配 $[B, H, W, d]$ 的 K/V， W 为缓存窗口；新张量写入指定切片，`ptr_cur` 指向下一空位。满了时把旧内容左移、丢掉最早部分，再写新内容。为了避免重叠赋值，源切片先 clone；这仍会产生复制和临时内存，不能把左移称为零成本视图操作。

优化版在 batch 大小改变时重新分配，输入块不能大于窗口。生成函数对长提示按窗口大小分块 prefill。分块能让程序处理长度超过单次缓存容量的提示，但丢掉历史后，它不再等价于完整上下文模型。

还有一个实际类型边界：固定源码预分配 `torch.zeros(..., device=x.device)` 时没有显式传 `dtype=x.dtype`。默认 float32 路径正常，但把模型改成 float64 或半精度时，缓存 dtype 未必跟随新投影，可能触发不相容或精度变化。通用实现应由 `keys_new.new_zeros(...)` 创建，或者明确传 dtype。本书对上游优化版的执行限定在 float32。

8.6 丢弃 K/V 不等于重算截断窗口

若窗口只保留最近四个位置，有两种不同算法。第一种保留旧位置已经算好的高层 K/V，它们可能包含更早上下文的信息；第二种截取最近四个原始 ID，从第一层重新计算，且可能把位置重新编号为 0—3。两者的输入表示和历史依赖都不同，通常不会给出相同 logits。

因此，缓存等价应先在“不发生丢弃、没有位置溢出”的条件下证明。对上游三个模型使用相同小权重、长度 6、窗口 12，朴素和优化缓存与无缓存基线的最大 float32 误差均为 2.38×10^{-7} 。把优化窗口减为 4 后，同一比较最大差变为 0.13734。这是预期的语义变化，不是把容差调大就该忽略的误差。证据在 `qa/core_training/cache_probes.json`。

8.7 绝对位置容量仍是硬限制

缓存窗口变小，只限制保存多少 K/V，不会自动扩大位置 embedding 表。优化版显式检查累计位置不超过 `context_length`，生成时还把请求的新 token 数限制为 `context_size - prompt_length`。本书小测试提示长 6、上下文容量 10、请求生成 20，实际只增加 4 个 token，保留输入前缀。

从计算依赖上说，最后一个预测出的 token 不一定需要再被模型读入，所以上游循环跳过了最后 token 之后的多余前向。它的容量限制是一个保守生成策略，应按实际代码说明，不能描述为自动无限滚动生成。若希望继续生成，需要重新设计位置与缓存策略，并重新验证语义。

8.8 计算量与存储要分开算

在不截断的长度 n 生成过程中, 没有缓存、每步重算完整前缀时, 注意力方阵累计成本包含 $\sum_{t=1}^n O(t^2 D) = O(n^3 D)$, 线性层累计约 $O(n^2 D^2)$ 。缓存后, 每步只计算一行注意力, 累计为 $\sum_t O(tD) = O(n^2 D)$, 新增位置的线性层累计为 $O(nD^2)$ 。这讨论的是朴素完整重算与增量推理的比较, 不是把注意力本身变成与上下文无关的常数成本。

普通 MHA 的缓存元素数为 $2LBTHd = 2LBTD$, 乘每个元素字节数即存储量。本书 $L = 2, B = 2, T = 6, D = 16$, 共 768 个元素; float64 是 6,144 字节。长上下文和大批量会线性放大缓存, 这就是后续 GQA、MLA 等方法关心的成本。

实际加速还受矩阵大小、分配、同步、设备带宽和内核调用影响。小模型可能被 Python 和分配开销主导。上游 README 的特定 Mac 性能结果属于上游实验, 不是本书复测; 我们不把某个倍数写成所有 CPU/GPU 的承诺。

8.9 怎样把等价检查写完整

配套有两组检查。run_checks.py 比较本书模型完整前向、逐 token 和先三 token 后三 token 的块缓存, float64 最大差分别约 4.44×10^{-16} 和 3.33×10^{-16} 。cache_probes.py 则抽取上游三版定义, 用缩小配置测试单 token、分块、重置、容量限制和窗口丢弃反例。

上游 tests.py 比较三版生成 ID, 另外检查位置溢出和提示分块。生成 ID 相同是必要但不充分的检查, logits 的最大误差也应记录; 重置测试能发现跨请求污染; 容量测试应检查实际返回长度, 而不只是“不崩溃”。小配置测试不覆盖多请求并发或真实 GPU 性能。

运行:

```
python examples/core_training/cache_probes.py runtime_sources/upstream qa/core_training/cache_probes.json
```

该脚本要求已准备好的固定源码, 不会下载词表或权重。

8.10 练习与答案

补充题一: 已有 5 个缓存 token, 一次新增 3 个, 允许 mask 的三行各有几个 True? 答案: 6、7、8。

补充题二: 为何普通训练中更新参数后不能继续复用旧缓存? 答案: 旧 K/V 是旧参数和旧中间表示的函数, 新参数改变了这些值, 复用会跳过应有的重新计算和梯度路径。

补充题三: 缓存只保留最近 100 个位置, 是否说明模型可以使用绝对位置表生成第 10,000 个位置? 答案: 不能, 缓存容量和位置表容量是独立限制。

参考与覆盖

- 无缓存、朴素缓存、优化缓存脚本：复用的 LayerNorm/GELU/FFN 见第 6 章，新增状态与生成路径在本章逐项解释。
- 上游测试：等价、位置容量、分块 prefill；本书以小配置新增 logits 与反例检查。
- 上游图片与硬件计时没有复用；本章图为原创，所有数值注明本书实际配置。

第 09 章 GQA、MLA 与跨层 KV 共享

生成一句话时，模型每次只增加一个 token，却要反复读取历史的 key 和 value。第 08 章用缓存避免了重复投影；本章进一步问：缓存里的东西，能不能少存一些？三种方法压缩的是三个不同方向：GQA 减少 KV 头，MLA 减少每个 token 的存储维度，跨层共享减少独立保存 KV 的层数。它们改变了模型的约束，不能在任意已训练 MHA 模型上直接打开而期待输出不变。

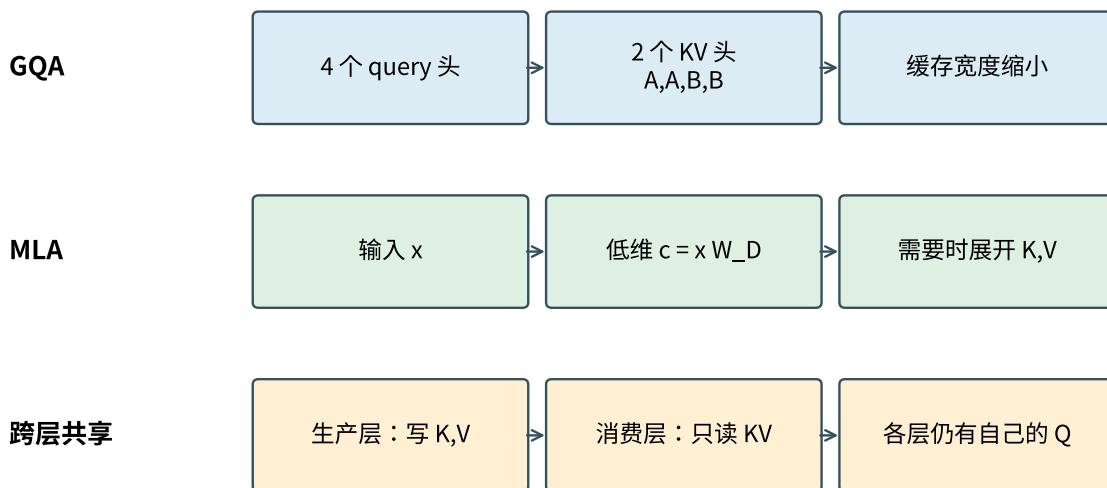
本章源码基准为固定提交中的 GQA、MLA 和 KV sharing。以下推导和小实验为本书原创；上游的随机模型生成和 GPU 时间不是本书实测。

09.1 先把缓存账算在正确的对象上

设批量为 B ，历史长度为 T ，层数为 L ，query 头数为 H ，每头维度为 d ，每个元素占 s 字节。若 key 和 value 维度相同，普通 MHA 缓存占 $2BTLHds$ 字节。这里没有参数、临时注意力分数、框架内存池和激活；缓存少四倍，不代表进程显存少四倍。

例如 $B = 1, T = 1024, L = 8, H = 8, d = 64, s = 2$ ，得到 16,777,216 字节，即 16 MiB。GB 使用十进制，GiB 使用二进制。对估算器结果，除了参数，还应核对单位。GQA 等估算脚本采用向上取整头宽以容许不整除的输入，实际模型则通常断言整除；因此估算器能打印结果不等于模型能实例化。

KV 可以沿不同方向共享



09.2 GQA: 多个问题，可以共用一套历史记录

令 KV 头数为 G ，要求 H 能被 G 整除，每组含 $r = H/G$ 个 query 头。输入 X 的形状为 $[B, T, D]$ 。三个投影分别输出 $Q : [B, T, Hd]$ 、 $K, V : [B, T, Gd]$ ；拆头以后，query 第 h 个头读取 KV 第 $g(h) = \lfloor h/r \rfloor$ 个头：

$$A_h = \text{softmax} \left(\frac{Q_h K_{g(h)}^\top}{\sqrt{d}} + M \right) V_{g(h)}.$$

$M_{t,j}$ 在 $j > t$ 时为负无穷，否则为零。每个 query 头仍有自己的投影，因而共享 KV 不等于共享注意力权重。缓存缩为 $2BTLGds$ ，与 MHA 之比为 G/H 。当 $G = H$ 时退化为 MHA； $G = 1$ 时是 MQA。GQA 原论文研究了从多头模型继续训练的路径，论文中的质量结论依赖训练方案，不能由本章的等价性测试推出。

源码 `GroupedQueryAttention.__init__` 用 `num_kv_groups` 表示 KV 头数，建立输出宽度为 `num_kv_groups * head_dim` 的 `W_key` 和 `W_value`。`forward` 先缓存未展开的 KV，再用 `repeat_interleave(group_size, dim=1)` 对齐 query 头。若两个 KV 头记作 A、B，四个 query 头应读取 A、A、B、B。误用 `repeat` 会得到 A、B、A、B；张量形状完全一样，但模型函数已经变了。

一个容易混淆的细节来自同目录的估算器：其 CLI `--n_kv_groups` 用作 **每个 KV 头分摊的 query 头数**，以 `n_heads // n_kv_groups` 推出 KV 头数。不要把命令行参数原封不动搬入模型字典。最稳妥的笔记方式是始终明确写 H, G, r ，再映射到具体函数参数。

缓存推理时， Q 长度可能只有 1，而 K 长度为全部历史。`ptr_current_pos` 决定 query 的绝对位置；不能对一个 $1 \times T$ 矩阵套用从零开始的上三角模板，否则当前 query 只会看见第一个 key。`reset_cache` 同时清空 K、V 和位置计数；新请求如果只重置其中一个，会产生跨请求污染或位置错位。

09.3 MLA: 先存一个低维向量，需要时再展开

教学版 MLA 令潜在宽度为 c ，用三个矩阵建立分解：

$$C = XW_D, \quad K = CW_{UK}, \quad V = CW_{UV},$$

其中 $W_D : [D, c]$ ，两个上投影矩阵为 $[c, Hd]$ ，缓存 $C : [B, T, c]$ 。缓存大小变为 $BTLcs$ 。例如 $Hd = 512, c = 128$ ，每 token 的 KV 存储从 1024 个数缩到 128 个数。但投影 $W_D W_{UK}$ 的秩至多为 c ；这是一种建模约束，并不保证任意旧 key 都能无损压缩。

源码 `MultiHeadLatentAttention.forward` 的顺序很重要：先对新输入产生 query 与 `latent_new`，把 `latent` 追加到 `cache_c_kv`，再对 **全部历史 latent** 执行 `w_UK`、`w_UV`。因此它演示了持久缓存压缩，却仍可能形成完整 KV 临时张量，并在每个解码步重做上投影。不得将 `cache_c_kv.numel()` 当作整个注意力峰值内存，也不得从字节估算推断速度。

对一个头，暂忽略偏置与位置旋转，有

$$Q(CU_K)^\top = (QU_K^\top)C^\top, \quad P(CU_V) = (PC)U_V.$$

先把 query 投到潜在空间，再与 C 点乘，可不显式重建 K ；value 的上投影也能后移。这是矩阵乘法结合律，不是近似。但若对展开的 key 施加依赖位置的旋转，旋转与上投影一般不能随意交换。

DeepSeek-V2 的完整 MLA 使用解耦的旋转位置部分；本目录保留 GPT 的绝对位置 embedding，直接从输入生成 query，没有完整复现论文的 query 低秩支路和解耦 RoPE。完整模型的缓存公式还应计入位置相关 key，而不能只剩 c 。

本书 `latent_and_sharing()` 同时检查 $X(W_D W_{UK}) = (XW_D)W_{UK}$ 和上述投影吸收。最大误差均为 1.78×10^{-15} 。测试只证明给定低秩参数的两种算法等价；没有证明低秩模型能替代任何密集模型。

09.4 跨层共享：消费者仍然提问，生产者负责写入

设只有 P 层产生 KV，其余层引用生产者缓存。每层仍用自己的当前隐藏状态形成 query。缓存大小为 $2BTPGds$ 。这种方法减少层方向的独立历史表示；理论背景可见 [Cross-Layer Attention](#)。

教学脚本 `MultiHeadAttentionWithKVSharing.forward` 接收 `shared_kv`。它为 `None` 时执行 K/V 投影和追加，并返回 `new_shared_kv`；非空时只读传入张量，返回的新共享值为 `None`。`GPTModel.forward` 顺序遍历 `ModuleList`：前 P 层独立生产，后面的层复用最近的生产者。注意，这不是任意相邻层两两共享；也不是 Gemma 4 按局部/全局类型寻找生产者的规则，后者见第 21 章。

消费者不可把已经包含当前 token 的共享 KV 再追加一遍，否则每步历史长度会增长两次。消费者也不能就地旋转或修改共享 key，因为其他层正在读取同一对象。本章小例检验三位消费者的 `data_ptr` 相同，再改变 query，确认共享缓存仍能产生不同输出。

教学实现保留消费者的 `W_key/W_value` 模块。它们在共享分支中不参与计算，但仍可能占参数内存。因此“没有计算该投影”“没有缓存该层 KV”“根本没有该参数”是三个不同结论。上游测试分别验证所有层生产时与 MHA 一致、只有生产层留下缓存、估算器用生产层数计费；这比只检查输出形状更有针对性。

09.5 从注意力接回 GPT：哪些步骤没有改变

三个目录中的 `LayerNorm` 仍沿特征轴计算均值与总体方差，`GELU` 使用 `tanh` 近似，`FeedForward` 保留扩展再投回的两层结构。`TransformerBlock.forward` 依次执行归一化、注意力、残差，再归一化、前馈、残差。改变 KV 机制没有自动改变这些步骤。

`GPTModel` 的 token/position embedding 相加，最后归一化并经输出头得到 $[B, T, V]$ logits。缓存版用 `ModuleList` 传递 `use_cache`，记录模型级绝对位置，并通过 `reset_kv_cache` 重置全部注意力层。`generate_text_simple_cached` 先处理整个 prompt，再仅处理新增 token；到上下文边界时要重建裁

剪后的上下文，不能让绝对位置 embedding 越界。CLI 的 `main`、`parse_args`、计时和设备峰值内存报告属于运行入口，不是模型质量评估。目录里的 MHA 对照文件保持相同主干，差别集中在投影、缓存与掩码。

`memory_estimator_*` 的字节函数、参数校验、单位换算和 `main` 把 09.1—09.4 的式子变成命令行输出；`plot_memory_estimates_*` 在不同 T 上重复计算这些式子并画线。它们画的是解析估算，而不是运行模型采集的测量。本书重新绘图，不复用上游 webp。

09.6 可复现实验、练习与答案

完整可运行程序是 `examples/architectures/architecture_checks.py`，运行 `python examples/architectures/architecture_checks.py`。实际环境为 Python 3.12.14、PyTorch 2.6.0+cpu，seed 1949、单线程、float64。日志 `qa/architectures/algebra_checks.json` 记录：GQA 分组与展开误差为 0，错误 repeat 被检出；MLA 两项最大误差为 1.78×10^{-15} ；共享相同查询的误差为 0。该程序不导入上游模型，不下载权重。

补充练习。 ① $H = 12, G = 3$ 时，query 头 7 应读取哪个 KV 头？② 为什么 MLA 的低维缓存与标准 MHA 输出不应直接比较？③ 所有层都设为生产者时，跨层共享模型应满足什么回归测试？④ 为什么显式复制 KV 可能抵消部分 GQA 工作内存收益？

提示与答案。 ① 零基编号， $r = 4$ ， $\lceil 7/4 \rceil = 1$ 。② 等价性需要两边用相同的因子化投影，任意 MHA 参数通常不满足秩约束。③ 同权重、关闭 dropout、相同位置和 mask 时，完整前向及缓存生成均应与 MHA 一致。④ 缓存可以保留 G 个头，但 `repeat_interleave` 的临时张量恢复了 H 个头；专用内核可避免该展开，教学实现未承诺做到。

进一步的真实源码检查 `source_attention_checks.py` 导入已审查的 GQA/MLA/KV 共享脚本（不运行 CLI）。GQA 与 MLA 完整/分块误差均为 0；KV 共享三层模型误差为 1.49×10^{-7} ，且消费者没有独立缓存。准确结果见 `qa/architectures/source_attention_checks.json`。

参考资料与边界

- 固定源码：前述三个目录，逐函数映射见配套覆盖台账。
- Ainslie 等，GQA (2023)：https://arxiv.org/abs/2305.13245。
- DeepSeek-AI, DeepSeek-V2 (2024)：https://arxiv.org/abs/2405.04434。
- Brandon 等，Cross-Layer Attention (2024)：https://arxiv.org/abs/2405.12981。
- 未运行：大模型生成、GPU 内存/吞吐比较、从 MHA checkpoint 继续训练。

第 10 章 局部与稀疏注意力：SWA 和 DSA

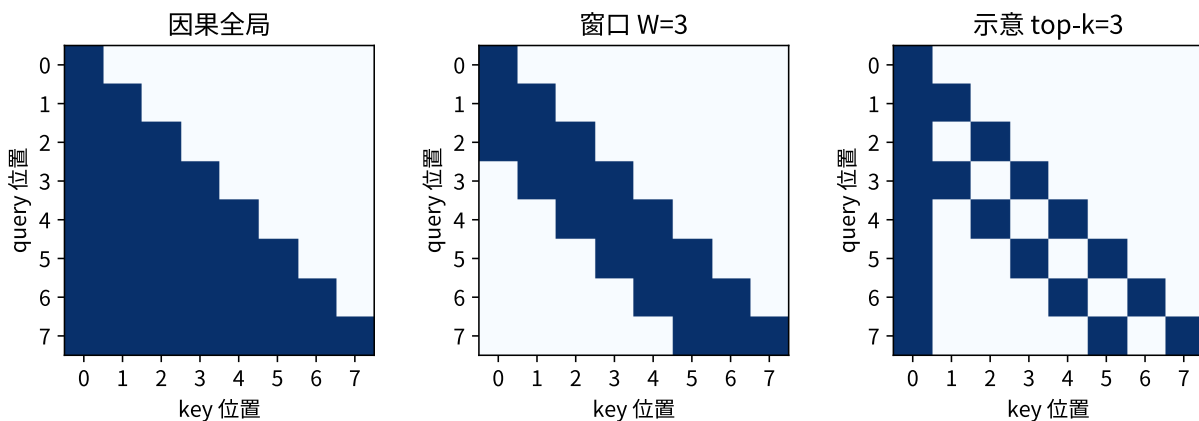
第 09 章主要减少历史的存储方式。本章减少一次 query 实际读取的历史位置：SWA 按距离选择，DSA 按学习到的分数选择。必须同时检查两件事：输出是否只使用允许的位置，以及程序是否真的少做了计算。给稠密矩阵涂上一层 mask，只完成前一件事。

10.1 SWA 的窗口到底包含谁

零基位置 t 的因果滑动窗口宽度为 W ，本章约定包含当前 token。允许的位置满足

$$\max(0, t - W + 1) \leq j \leq t.$$

例如 $W = 3, t = 4$ ，可见位置是 2、3、4，数量为 3。源码 `MultiHeadAttentionWithSWA` 先形成差值 $\Delta_{t,j} = t - j$ ，再以 $(\text{diff} < 0) \mid (\text{diff} \geq W)$ 标记禁止位置。第一项挡未来，第二项挡过远历史。只写因果条件并没有滑窗；写成 $\text{diff} > W$ 则多放进一个 token。见 [固定 SWA 源码](#)。



窗口缩短了单层的信息通路，但多层局部注意力可以逐步传播信息。若每层窗口为 W ，不考虑边界，经过 L 层后，一个位置可能间接受到最多 $1 + L(W - 1)$ 个位置影响。这不等于拥有直接读取任意旧 token 的能力：信息必须穿过中间表示。混合局部与全局层，让部分层维持远距离直接连接。

[Longformer](#) 和 [Gemma 3 技术报告](#) 展示了不同任务与架构中的局部/全局组合，不能把某一个比率当成通用最优值。

10.2 持久缓存可以只留 W ，当前计算却未必只用 W

逐 token 解码时只需保留最后 W 个 KV，持久缓存上界是 $2BWGds$ 每层。但分块预填充容易出错。假设旧位置处理到 4，本次同时输入 5、6、7， $W = 3$ 。位置 5 需要 3、4、5，位置 7 需要 5、6、7。当前计算的并集是 3—7，共 $W + T_q - 1 = 5$ 个位置，不能一上来裁成 5、6、7。

源码先保存 `old_cache` 与 `old_len`，拼出 `combined_k/v`。注意力所需部分保留最多 $W + \text{num_tokens} - 1$ ；下一步的持久 cache 只留最后 W 。两个切片用途不同。丢弃开头若干项以后，用 `k_start_pos_abs = ptr_current_pos - old_len + dropped` 找回 key 的绝对起点。query 从 `ptr_current_pos` 开始。这样缓存即使只有三个槽，槽里的 token 也不会被错误当成位置 0、1、2。

若有 L_s 个局部层、 L_f 个全局层，缓存估算为

$$2BGds [L_s \min(T, W) + L_f T].$$

`memory_estimator_swa.py` 解析比率、分配整数量的局部/全局层，再求和；不能把 L_s/L_f 的小数比率直接当作实际层数。模型里 `sliding_window_stride` 决定具体层的位置，两者必须核对。绘图脚本只把这个估算在不同长度上展开。训练时教学实现仍构造 $T \times T$ 注意力分数，因而不具备专用滑窗内核的线性工作量。

10.3 DSA：先索引，再做注意力

固定窗口不会问旧 token 是否重要。DSA 的索引器为候选历史生成相关性分数，再选择 top- k 。固定提交的 [DSA 源码](#) 使用输入隐藏状态，而完整 DeepSeek 模型将索引器与 MLA 组合。这个教学差异影响了内存和投影成本。

设索引头数 H_I 、索引宽度 d_I ，当前 query 输入形状为 $[B, T_q, D]$ ，候选历史为 $[B, T_k, D]$ 。w_q_index 输出 $[B, T_q, H_I, d_I]$ ，w_k_index 输出共享的 $[B, T_k, d_I]$ ，w_weights 输出 $[B, T_q, H_I]$ 。索引分数为

$$I_{t,j} = \sum_{h=1}^{H_I} \frac{w_{t,h}}{\sqrt{H_I}} \text{ReLU} \left(\frac{q_{t,h}^\top k_j^I}{\sqrt{d_I}} \right).$$

w 是原始线性输出，不经过 softmax，可以为负。ReLU 把每头负点积清零，却不保证最终每个分数非负，也不产生“每行只剩 k 项”的稀疏性。真正的离散选择来自 `topk(...).indices`。[DeepSeek-V3.2 官方报告](#) 是完整系统的来源，本文只对已列教学实现作代码级解释。

索引器必须在 top- k 之前加因果 mask，否则未来位置可能占满名额。之后主注意力仍应与因果 mask 合并：短序列开头可见位置少于 k ，topk 为凑满固定输出宽度可能返回分数为负无穷的未来索引。最终取“被选中且不在未来”的交集，才能安全地 softmax。

主注意力根据被选位置构造 bool mask，将其余 logit 置为负无穷，缩放再 softmax，再对 V 加权。若全部位置都被挡住，负无穷减负无穷会给出 NaN；不能用有限最小值来掩盖这个逻辑错误，因为整行同为有限最小值时 softmax 可能给出均匀分布，反而泄漏信息。有效 query 至少应有一个合法 key。

10.4 稀疏选择不等于稀疏执行，也不等于自动学会选择

教学 `MultiHeadAttentionWithDSA` 首先计算完整的 $QK^\top$ ，再应用稀疏 mask。它还缓存 `cache_x`，让索引器重投影历史输入。相较普通 MHA，持久缓存多了隐藏状态，不能宣称 DSA 在这个文件中减少了 KV 存储。原理上稀疏 kernel 可只对选中键执行主注意力点积，约为 $O(Tkd)$ ；索引器仍有自己的扫描成本，完整复杂度要分别计账。

另一个边界是训练。`topk` 的索引是离散的，而该实现只把索引用于布尔 mask，主任务 loss 无法沿索引回传到索引器分数。没有额外可微训练目标，仅把随机索引器接进 GPT 并优化语言模型 loss，不会自动学到预期检索策略。部署报告的收益建立在已训练的索引器和专用实现之上；上游这里是可检查的选点演示。

10.5 配套脚本与验证路径

SWA 的 `TransformerBlock` 根据层编号选择局部或全局模块，残差、LayerNorm、GELU 前馈没有改变。模型级位置 embedding、`reset_kv_cache`、缓存生成以及参数 CLI 与第 09 章同一主干。DSA 多出三个索引参数：头数、头宽和 `top-k`；它们影响选择器形状，不改变输出词表大小。两个目录的 README 给动机、命令与上游时间示例；本书未执行那些长序列性能实验。

`tests.py` 和 `test_dsa.py` 的核心检查包括完整输入与缓存预填充的一致性、跨 chunk 的窗口边界、选择不会越过因果边界、`top-k` 足够大时退回密集行为。相同的 dropout 关闭、相同参数、相同绝对位置，是比较成立的前提；不同模型的随机初始化输出不应逐项相等。

本书完整程序 `examples/architectures/architecture_checks.py` 中的 `masks_and_sparse()` 用 7 个 token、宽度 3、分块 `[0:2], [2:5], [5:7]` 验证 SWA。完整与分块输出误差为 0；修改未来 value 后过去输出误差为 0。DSA 的稠密 mask 与逐 query 真正 gather 的结果最大误差为 4.44×10^{-16} 。程序还实际验证整行负无穷 softmax 产生 NaN。环境、seed 与完整日志见 `qa/architectures/algebra_checks.json`。这里没有速度测试。

10.6 补充练习与答案

题目。 ① 宽度 4、query 位置 9，可见哪些位置？② 分块长度 5 时，为什么当前注意力可能要用 8 个历史/当前位置？③ 如果先 `top-k` 再删除未来项，为什么结果可能缺少本来合法的重要历史？④ 只返回 `top-k` 索引的索引器，靠主注意力 loss 是否能获得常规梯度？

答案。 ① 6、7、8、9。② $W + T_q - 1 = 4 + 5 - 1 = 8$ ，早期 query 和后期 query 的合法窗口并集更宽。③ 未来项已经挤占选择名额，删除它们不会自动把被挤掉的历史补回来。④ 不行，离散索引切断该路径；需单独设计训练目标或其他可微方案。

本书还运行 `source_attention_checks.py` 中的真实上游SWA/DSA类，完整前向与三块缓存计算最大误差均为0。对DSA主输出平方和反传后，所有indexer参数的grad确实为None，实证支持10.4的训练边界。日志见 `qa/architectures/source_attention_checks.json`。

参考资料与未验证边界

- 固定源码 `ch04/06_swa`、`ch04/09_dsa`，模型、估算、绘图和测试均纳入覆盖台账。
- Longformer: <https://arxiv.org/abs/2004.05150>。
- Gemma 3: <https://arxiv.org/abs/2503.19786>。
- DeepSeek-V3.2 官方报告：上文链接。
- 未验证：长上下文语言质量、稀疏 CUDA kernel、索引器训练、报告中的部署成本。

第 11 章 Gated DeltaNet: 从矩阵状态理解递推

如果不保存每个历史 token，而只保留一个固定大小的矩阵，生成时的状态就不会随着句子长度一直增长。问题是：新信息怎样写进矩阵，旧信息怎样被读取或修正？Gated DeltaNet 给出的不是 softmax 注意力的另一种写法，而是另一种记忆更新规则。

本章沿固定提交的 [Gated DeltaNet README](#) 与估算脚本展开。这个目录把核心模块放在 README 的代码块中，独立 Python 文件是内存估算绘图；不要把文件数少误解为实现内容少。完整含卷积的 Qwen3.5 模块在第 20 章继续解释。[Gated Delta Networks 原论文](#) 是递推规则的研究来源。

11.1 一个 key 对应一行读出方向

对一个 batch、一个头，令 key/query 宽度为 d_k ，value 宽度为 d_v 。状态 $S_t \in \mathbb{R}^{d_k \times d_v}$ 可以看作一个不断改变的线性映射。给定列向量 $k_t \in \mathbb{R}^{d_k}$ ，预测 value 为 $S_t^\top k_t$ ；按 PyTorch 行向量写法就是 `k_t @ S`。读出 $y_t = q_t^\top S_t$ ，得到长度 d_v 的行向量。

先考虑简单外积累加 $S_t = S_{t-1} + k_t v_t^\top$ 。展开后， $q_t^\top S_t = \sum_{i \leq t} (q_t^\top k_i) v_i^\top$ 。这很像未归一化的线性注意力，但同一个 key 多次出现时，对应记录一直累加。若第二次出现是纠正第一次的信息，直接累加可能不是合适的更新。

11.2 Delta rule 用预测误差修正记忆

先对旧状态乘遗忘系数 α_t ，再读出旧状态对当前 key 的预测：

$$\tilde{S}_t = \alpha_t S_{t-1}, \quad \hat{v}_t = \tilde{S}_t^\top k_t.$$

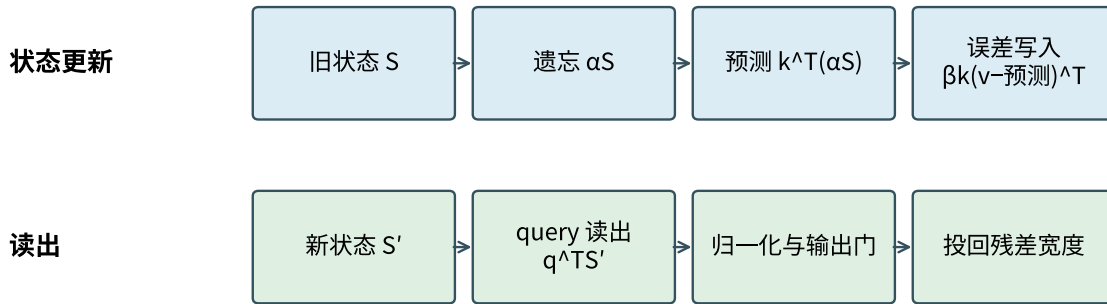
用真实 value 减去预测值得到误差，乘写入强度 β_t ，沿当前 key 的方向更新：

$$S_t = \tilde{S}_t + \beta_t k_t (v_t - \hat{v}_t)^\top, \quad y_t = S_t^\top q_t.$$

本章示例把 α, β 约束到 $(0, 1)$ ；它们是从输入学习到的门，不是人为固定的学习率。 α 决定整体保留多少旧记忆， β 决定当前误差修正多强。两者作用不同：把 β 设为零只停止写入，状态仍会按 α 衰减。

这条规则还可以从瞬时平方误差理解。把 k_t 当常量，目标为 $\frac{1}{2} \|S_t^\top k_t - v_t\|^2$ ，对 S 的梯度是 $k_t (S_t^\top k_t - v_t)^\top$ 。一次梯度步正好给出 delta 更新。这里是在前向过程更新状态，并非调用优化器修改模型参数；训练模型参数仍通过反向传播完成。

每一步先修正记忆，再读取



取 $k = [1, 0]^T$, $v = [3, 4]^T$, $\alpha = 1$, $\beta = 1$ 。无论旧状态第一行是什么，更新后第一行恰好是 $[3, 4]$ ，第二行不变。因为 $k^T k = 1$ ，再次沿同一个 key 读出，就精确得到新 value。这解释了源码为何对 key 作 L2 归一化。若把 key 放大 10 倍，更新中的新 value 写入项 $\beta k v^T$ 放大 10 倍，而旧记忆纠正项 $-\beta k k^T \tilde{S}$ 放大 100 倍。因此整个增量并非统一放大某个倍数，也不能再把 $\beta = 1$ 理解为一次覆盖。

11.3 源码每一步的形状与顺序

README 的示例从 $X : [B, T, D]$ 生成 Q、K、V、输出 gate 和更新门。拆头后 Q/K/V 为 $[B, H, T, d]$ ，这里为了易读令 $d_k = d_v = d_o$ 。Q 作 L2 归一化后再除以 $\sqrt{d}$ ，K 只作 L2 归一化。 $s = x.\text{new_zeros}(B, H, d, d)$ 与输入保持设备和 dtype。

时间循环里， $S = S * a_t$ 必须早于 kv_mem 的计算，因此误差比较的是已经衰减的状态。 $(S * k_t.\text{unsqueeze}(-1)).\text{sum}(\text{dim}=-2)$ 沿 key 维求和，得到 $[B, H, d_v]$ 预测；更新外积需要 $k_t.\text{unsqueeze}(-1) * \text{delta}.\text{unsqueeze}(-2)$ ，产生 $[B, H, d_k, d_v]$ 。若两个 `unsqueeze` 颠倒，方阵情况下可能仍可运行，却把记忆方向转置了。

输出沿 query 方向读取，所有时间结果堆叠后转成 $[B, T, H, d_v]$ 。模块随后做 RMSNorm 和 SiLU 输出门，再合并头、dropout、输出投影。输出 gate 调节已经读出的表示，与 α 、 β 两种状态门不能互换。普通“gated attention”只是给 softmax 注意力输出加门，仍保留 $T \times T$ 点积矩阵；这里没有这个矩阵。

README 为聚焦递推省略了局部卷积。它也没有在这个简化接口里跨 forward 调用保存状态；如果每次调用都重新建零矩阵，只喂最后一个 token 会丢失全部历史。真正流式实现应把最终状态传出，并在下次作为初始状态传入，同时重置不同请求之间的状态。Qwen3.5 的 cache 正是为此增加另一套协议。

11.4 为什么可以分块，为什么仍不能称为任意 softmax 的等价替代

将递推改写为

$$S_t = A_t S_{t-1} + U_t, \quad A_t = \alpha_t (I - \beta_t k_t k_t^T), \quad U_t = \beta_t k_t v_t^T.$$

从零状态展开有

$$S_t = \sum_{i=1}^t \left(\prod_{j=t}^{i+1} A_j \right) U_i.$$

乘积顺序从右到左按时间作用，不能交换。这个表达式为分块、并行前缀组合提供了入口，因为仿射映射可以结合；它没有让矩阵乘法变成可交换运算。读者可以先以慢速展开检查递推，再研究 chunk kernel 如何在块内使用三角结构。

固定状态会把不同历史映射到相同矩阵。它无法像存下所有 key/value 那样，之后随 query 任意重新分配每个 token 的 softmax 权重。这是压缩带来的表达约束，不是实现 bug。混合模型保留部分全局注意力层，用两种记忆方式承担不同职责，具体效果仍需训练与质量评估。

11.5 存储与计算的边界

每层推理状态占 $BHd_k d_v s$ 字节；等宽时为 $BHd^2 s$ 。普通 MHA 为 $2BTHds$ ，两式相等时 $T = d/2$ 。因此非常短的序列上，固定矩阵状态未必更小。简化递推计算量随 T 线性增长，约 $O(BTHd_k d_v)$ ；“线性”针对序列长度，不代表对头宽也线性。

plot_memory_estimates_gated_deltanet.py 用配置计算 MHA 和固定状态、再按混合比例相加。曲线没有计入卷积缓存、权重、训练时反向保存的中间状态，也不是 GPU 显存实测。训练可能需要保存、重算或分块组织中间量，不能直接沿用推理常量内存结论。

11.6 真实验证与补充练习

完整程序 examples/architectures/architecture_checks.py 的 delta() 用 6 个 token、3 维 key/value，对照逐步递推与逐项展开。float64 最大误差 4.44×10^{-16} ；单位 key、 $\beta = 1$ 的覆盖测试误差为 0。环境为 Python 3.12.14、torch 2.6.0+cpu、seed 1949、单线程。准确命令和全部数值在 qa/architectures/algebra_checks.json，没有运行论文训练实验。

题目。① $\alpha = 0$ 时状态还包含什么？② $\beta = 0$ 时是不是输出也必为零？③ 将公式中的 v - prediction 改成 v 会发生什么？④ 为什么 Q/K 不等宽于 V 时更容易检查转置错误？

答案。① 只剩当前写入 $\beta k v^T$ 。② 不是，仍可从衰减后的旧状态读取。③ 变成带遗忘的外积累加，失去针对当前 key 的误差纠正。④ 正确状态应为 $d_k \times d_v$ ；一旦误转置，矩阵形状通常立即报错，方阵反而容易掩盖错误。

参考资料

- 固定提交 ch04/08_deltanet/README.md 中的模块、比较片段与绘图说明。
- Yang 等，Gated Delta Networks: <https://arxiv.org/abs/2412.06464>。
- 完整 Qwen3.5 helper 的 chunk/recurrent 实现与卷积状态见第 20 章。

第 12 章 MoE：路由、专家与负载

一个前馈层的参数很多，但每个 token 是否都需要用到全部参数？混合专家模型（Mixture of Experts, MoE）把一层前馈换成多个专家，由路由器选择少数专家参与当前 token 的计算。它通常增加总参数，而不是把整个模型变小。收益来自每个 token 激活的专家较少，代价来自权重常驻、分派开销和负载不均。

本章解释固定提交 [ch04/07_moe](#) 的代码。这里的注意力仍是带 KV cache 的普通 MHA，MoE 换掉的是前馈模块。

12.1 路由器决定“谁来计算”，专家决定“计算什么”

设输入为 $X : [B, T, D]$ ，专家数为 E ，每个 token 选 k 位专家。路由矩阵 $W_r : [D, E]$ 产生 $R = XW_r$ 。对每个 token 取最大的 k 个分数，记对应集合为 $\mathcal{K}_t$ ，再只在选中集合归一化：

$$p_{t,e} = \frac{\exp R_{t,e}}{\sum_{j \in \mathcal{K}_t} \exp R_{t,j}}, \quad y_t = \sum_{e \in \mathcal{K}_t} p_{t,e} f_e(x_t).$$

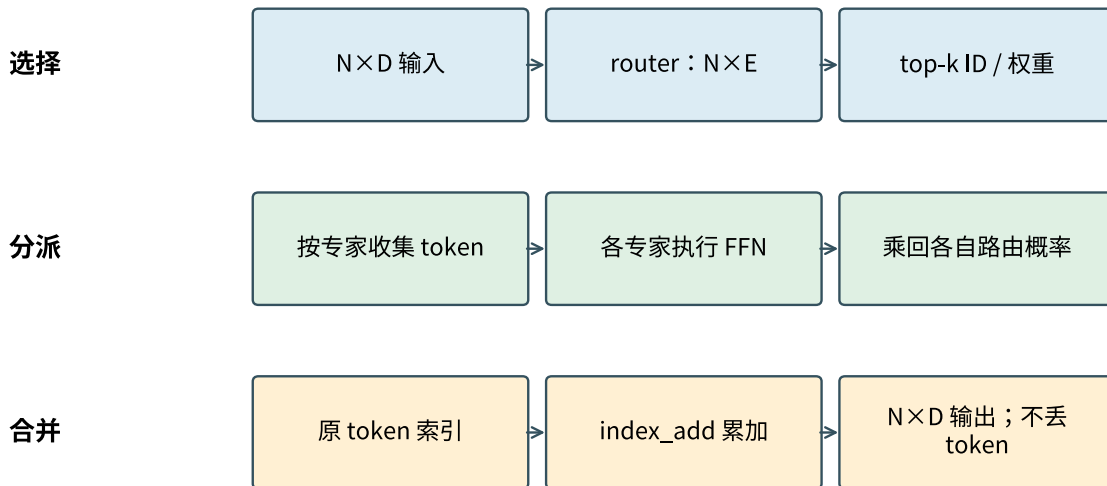
这与先对所有专家 softmax 再取 top- k 且不重归一化不同。后一种方式的权重和小于 1，会改变输出尺度。源码 `topk_scores.softmax(-1)` 明确采用前一种规则。

每个专家使用门控前馈。令中间宽度为 F ，无偏置时

$$f_e(x) = [\text{SiLU}(xW_{1,e}) \odot (xW_{2,e})] W_{3,e},$$

其中 $W_1, W_2 : [D, F]$ ， $W_3 : [F, D]$ ，参数约 $3DF$ 。三个矩阵分别保存于 `fc1/fc2/fc3` 的 `ModuleList`，确保优化器能发现参数。普通 Python list 不会注册为子模块，模型能前向但训练、存盘和设备迁移会出问题。

5 个 token、top-2：应有 10 次专家分派



12.2 为什么先展平，再按专家收集 token

若逐 token 运行每个选中专家，Python 循环和小矩阵乘法太碎。源码把 $[B, T, D]$ 展成 $[N, D]$ ，其中 $N = BT$ ；路由索引与概率相应成为 $[N, k]$ 。随后遍历本批真正被选中的 `unique_experts`。

对专家 e ，`topk_indices_flat == e` 得到 $[N, k]$ 布尔矩阵。沿 slot 维取 `any`，找到哪些 token 交给它；`nonzero` 得到 `selected_idx`，`index_select` 收集专家输入。计算结束后，仍须为每个 token 找回它在 `top-k` 中的 slot，`gather` 取出正确概率。只取第一列概率，会让同一专家在不同 token 上使用错误权重。

最后 `out_flat.index_add_(0, selected_idx, expert_out * selected_probs[:,None])` 将结果加回原 token。必须累加，因为一个 token 有 k 份结果。用覆盖赋值则只留下最后一个专家。上游通过选中 `mask` 的 `argmax` 找 slot，这依赖 `topk` 给每个 token 返回互不重复的专家编号。

MoE 不改变 token 的顺序或数量。输入 N 个 token、每 token k 次分派，应有 Nk 个分派记录；输出仍为 N 个 token。输出形状相同并不足以证明没漏算，因此本书测试同时核对分派总数和逐 token 基准。

12.3 总参数、激活参数与内存是三笔账

忽略 bias，MoE 层总参数约 $3EDF + DE$ ；每 token 激活专家参数约 $3kDF$ ，另有 router。

$E = 8, k = 2$ 时，专家总参数是一个专家的八倍，每个 token 用其中两份。训练时优化器状态可能覆盖全部专家，推理若全部常驻，也必须存放所有专家权重。

稀疏计算还不等于同样比例的实际加速。许多小专家的矩阵乘法可能利用率低；同一 batch 的不同 token 会共同激活许多甚至所有专家；多设备专家并行可能需要通信。源码通过 Python 循环、`.item()`、`index_select` 和 `index_add` 清楚展示分派，但没有融合 kernel 或跨设备调度。

上游 `memory_estimator_moe.py` 与 `plot_memory_estimates_moe.py` 分开报告/绘制总量和每 token 活跃量。`get_num_param_matrices` 区分两矩阵 GELU 与三矩阵 SwiGLU；`calc_router_params` 计入 DE ；`estimate_params_and_hidden` 在 `match_dense` 时先从密集预算扣除 router，再除以专家数和每单位宽度的参数量，四舍五入得到专家宽度。若 router 已用尽密集预算则报错。它匹配的是近似总参数，不是每 token 激活参数。因此应先核比较的 FFN 宽度及是否是两矩阵 GELU FFN、三矩阵门控 FFN。若只改变专家个数却同时改变中间宽度，不能把所有差异归于路由。`gpt_with_kv_ffn.py` 为密集对照，`gpt_with_kv_moe.py` 根据 `num_experts > 0` 选择分支；计时列表和设备内存统计是测量辅助，不参与数学定义。

12.4 负载集中为什么会成为问题

假设 100 个 token 都选同一位专家，其他专家闲置；总激活次数仍正确，但计算负载极不均衡。若引入容量上限 C ，超出部分必须明确处理：丢弃、改路由、排队或额外计算。源码这个教学版本没有容量截断，所有选中 token 都会被处理。不能把“没有容量机制”解释成“已解决负载均衡”。

Switch Transformer 是理解 top-1 路由、容量及辅助均衡目标的原始参考。一个常见辅助目标形式是 $E \sum_e f_e \bar{p}_e$ ，其中 f_e 是实际分派比例， $\bar{p}_e$ 是平均 router 概率；实现必须明确多选路由的分母，以及是否对离散统计停止梯度。本章没有声称上游教学脚本实现了该目标，也没有把它当成所有 MoE 的唯一方案。

硬 top- k 的选择边界不可导，但选中 logit 的 softmax 权重通常仍可导。一个特殊反例是这里 $k = 1$ 且只在选中项 softmax：单元 softmax 恒等于 1，主任务 loss 对 router 分数的这条权重路径梯度为零。不能仅靠“模型能反向传播”就认定 router 获得了有效训练信号。设计训练方案时需检查 router 梯度、分派分布和辅助目标。

12.5 接回主干、验证与练习

MultiHeadAttention 的 query/key/value 投影、绝对位置因果 mask 与 KV 缓存没有因 MoE 改变。LayerNorm/GELU/FeedForward 提供密集基线；TransformerBlock 在第二个残差分支选择密集或专家前馈；GPTModel、缓存重置和生成入口延续第 09 章的调用协议。README 的大配置与 GPU 时间是来源示例，本书未复跑。

完整程序 `examples/architectures/architecture_checks.py` 的 `experts()` 用 5 个 token、3 位专家、top-2，对比按专家聚集与逐 token 求和。为隔离分派逻辑，专家缩成线性映射，不能视为完整 SwiGLU 模型复现。输出误差为 0，10 个分派全部计入；人为设每位专家容量为 1，会产生 7 个超额分派。

Python 3.12.14、torch 2.6.0+cpu、seed 1949、float64、单线程，准确日志见 `qa/architectures/algebra_checks.json`。

补充练习。 ① top-2 输出合并时为什么要 `index_add_`? ② 专家总数翻倍、每 token 仍选两位，哪些成本增加，哪些近似不变? ③ $k = 1$ 时本实现 router 的主任务梯度为什么可能为零? ④ 如果把空专家 batch 也送入矩阵乘法，会不会改变数学输出?

答案。 ① 两份贡献必须相加，覆盖会漏掉一份。② 总专家参数及其存储增加，每 token 的专家算术量近似不变，router 输出与调度成本可能增加。③ 选中集合仅一个元素，softmax 值恒为 1；离散索引本身无常规梯度。④ 数学上空 batch 没有贡献，工程上会增加无用调度，某些模块还可能不支持空输入；源码因此跳过。

另以 `source_attention_checks.py` 运行上游真正的三矩阵 MoE 专家，设 top-1 并对输出平方和反传，router 权重最大梯度实测为 0。这是 12.4 所述训练失败条件的实际反例，不表示 top-2 等配置也必为零。

参考资料

- 固定源码 `ch04/07_moe` 全目录。
- Fedus 等, Switch Transformers: <https://arxiv.org/abs/2101.03961>。
- 门控前馈的原始研究: <https://arxiv.org/abs/2002.05202>。
- 本章未运行专家并行、容量溢出训练、负载均衡训练或吞吐基准。

第 13 章 预训练目标、交叉熵与验证集

模型已经输出 $[B, T, V]$ 的 logits，接下来要把“正确续写的分数不够高”变成可优化的标量。训练循环本身只有几步，真正容易出错的是：哪个位置预测哪个 token、损失平均了哪些项目、验证数据是否泄漏，以及图上的下降究竟说明什么。

本章对应上游 ch05 主 notebook 的训练部分、gpt_train.py 及相关练习。生成策略放在第 14 章，保存恢复与官方权重映射放在第 15 章，避免把三种不同责任混在一个长脚本里。固定训练脚本

13.1 交叉熵取的是目标 ID 对应的概率

设位置 (b, t) 的 logits 为 $z_{b,t,v}$ ，真实下一个 token 为 $y_{b,t}$ 。概率为

$$p_{b,t,v} = \frac{e^{z_{b,t,v}}}{\sum_{u=0}^{V-1} e^{z_{b,t,u}}}, \quad L = -\frac{1}{BT} \sum_{b,t} \log p_{b,t,y_{b,t}}.$$

目标张量形状为 $[B, T]$ ，每项是一个整数 ID，不需要先转 one-hot。上游先显式索引目标概率、取 log、求平均，再与 cross_entropy 比较。这能解释 API 究竟在计算什么，但生产实现应直接用 logits，避免概率先下溢为零。

对单个位置，可以改写为

$$L = \log \sum_v e^{z_v} - z_y.$$

设 $m = \max_v z_v$ ，则 $\log \sum e^{z_v} = m + \log \sum e^{z_v - m}$ 。这是稳定 log-sum-exp 的来源。对 z_j 求导，得到 $\partial L / \partial z_j = p_j - \mathbf{1}[j = y]$ ：真实项概率不足时梯度为负，梯度下降会提高它的 logit；其余项梯度为正。

13.2 flatten 不应该打乱位置与标签配对

logits.flatten(0,1) 把 $[B, T, V]$ 改成 $[BT, V]$ ，targets.flatten() 把 $[B, T]$ 改成 $[BT]$ 。二者按相同批次优先、位置随后顺序展平，所以第 $bT + t$ 行仍对应目标 $y_{b,t}$ 。如果一边先转置再 flatten，形状可能一样，配对却错了。

cross_entropy 需要未经 softmax 的分数。配套反例对 $[4, 0, -2]$ 、真实类别 0，正确 loss 为 0.0205811；先 softmax 再交给 cross_entropy 得到 0.5645241。后者不会报错，因为概率也是浮点数，但 API 又把它当 logits 做了一次归一化。

均匀分布给每个 token 概率 $1/V$ ，平均损失为 $\log V$ 。本书 $V = 8$ ，实测零 logits 的损失为 2.0794413，与 $\log 8$ 一致。这是检查词表维度、损失方向和数量级的便宜基线。

13.3 perplexity 依赖分词与计量单位

困惑度定义为 $\exp(L)$ ，这里 L 使用自然对数，是平均 token 负对数似然。均匀八项分布的困惑度为 8。它可解释为几何平均意义下的不确定度，但不是模型每一步真的只在八个候选里均匀选择。

不同分词器把同一文字切成不同数量的 token，平均每 token 损失的单位就不同。因此不能直接拿两种分词器的 perplexity 排语言能力。即使分词器相同，数据域、上下文长度、文档边界和 loss mask 不同，也会改变指标。

13.4 验证集要在造重叠窗口之前分开

上游把短文按字符位置切成前 90% 训练、后 10% 验证，再各自分词、建窗口。它避免了先造重叠窗口再随机划分的直接泄漏，但同一作品前后段仍高度相关，不能代表广泛领域的泛化。对于多文档语料，更稳妥的是以文档、来源或时间等有意义单位划分，并去除跨划分重复。

字符比例不等于 token 比例。用 `total_tokens*ratio` 做容量提醒只是粗检查；真正应分别编码两段，核对各自是否至少有 `context_length+1` 个 token，以及 `drop_last` 后是否还有完整批次。训练 loader 打乱并丢尾，验证 loader 不打乱且保留尾部，这些设置会影响实际参与统计的 token 数。

13.5 loader 平均要明确分母

上游 `calc_loss_batch` 返回一个批次的 token 平均损失；`calc_loss_loader` 把若干批平均再平均。批次 token 数相同时，这等于整体平均。如果最后一批较小，或者未来引入 padding/loss mask，简单的平均之平均会失真。

假设第一批有三个 token，损失都是 1；第二批只有一个 token，损失为 4。正确 token 平均是 $(1 + 1 + 1 + 4)/4 = 1.75$ ，批均值平均却是 $(1 + 4)/2 = 2.5$ 。配套检查执行了这个反例。通用评价应累计负对数似然之和与有效 token 数，最后相除。

上游空 loader 返回 NaN，提醒没有可评估数据；若显式 `num_batches=0`，还需避免零分母。限制 `eval_iter` 可以节省时间，但只评估前几个 batch 的结果是局部估计，不能写成完整验证集指标。训练 loader 的 shuffle 会使被选中的评估样本随时改变。

13.6 训练、评价、采样三段的模式切换

`train_model_simple` 每个批次先清梯度、前向算 loss、反向、优化器更新，然后增加 `tokens_seen` 和 `global_step`。`tokens_seen += input_batch.numel()` 计的是处理次数，重叠窗口和多轮训练会重复计数，不等于独立语料大小。

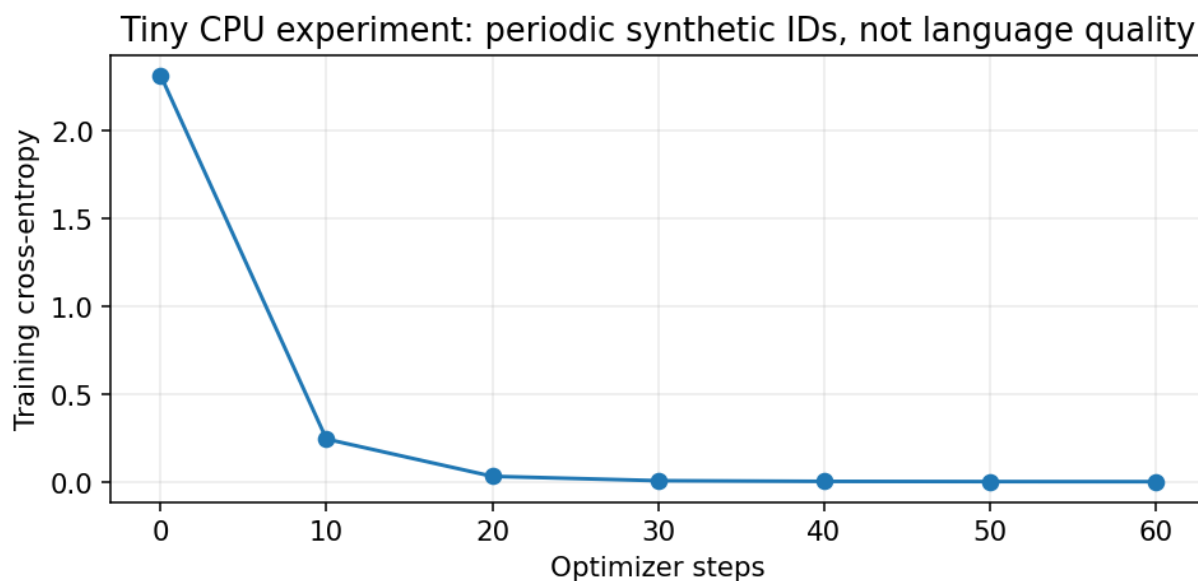
评价函数切到 eval，并在 no-grad 下计算训练与验证损失，然后恢复 train。样例生成也暂时切 eval，结束后恢复 train。若把这些函数放进更通用的工具，应记住调用前的模式再恢复，而不是无条件设 train；本书生成函数采用这种方式。

记录的训练损失还要说明是在参数更新前、更新后，还是 eval 模式重测。dropout 开启时，一个训练批次的即时 loss 与 eval 模式下相同样本的 loss 不必相等。比较曲线时要统一定义。

13.7 AdamW 更新与短实验的目的

AdamW 保存梯度一阶、二阶移动平均，并对参数做解耦权重衰减。它需要额外状态，因此内存预算不同于 SGD。学习率和 weight_decay 都改变训练轨迹，固定 seed 不能抵消这些设置差异。

本书不训练 124M 模型。使用 $V = 8, C = 12, D = 16, H = 4, L = 1$ 的 TinyGPT，在循环 ID $0, 1, \dots, 7$ 上造长度 8、stride 8 的窗口，AdamW 学习率 0.015、weight_decay 0.01，梯度范数上限 1，更新 60 次。第 0、10、20、30、40、50、60 步重测损失依次约为 2.315901、0.244195、0.032200、0.007208、0.003374、0.002243、0.001781。



这个任务刻意简单：每个 ID 的后继几乎确定，所以 loss 应快速下降。它验证梯度、更新、标签对齐和模型容量是否工作，不验证自然语言、事实性或泛化。没有把同一周期数据切两半就称作独立语言验证集。

13.8 如何读上游训练脚本和测试

text_to_token_ids 增加批次维，token_ids_to_text 去掉单条批次维并解码；它们默认面向 batch size 1 的显示。主 notebook 允许结束符号，集中训练脚本的简单包装使用默认编码策略，这个差异在处理含特殊字面量的文本时要留意。

`generate_and_print_sample` 每轮生成短文本，并把换行替换为空格，仅影响终端显示，不改变模型训练数据。`plot_losses` 用 `epoch` 和 `tokens_seen` 两个横轴展示训练进度；若评价点间隔不均匀，用 `linspace` 映射 `epoch` 只是近似，最好记录真实更新位置。

`main` 负责本地优先读取短文、必要时下载、建模型、建 loader 和启动循环。执行文件末尾还保存权重及损失图。本书不运行其默认大配置；同目录 `tests.py` 用宽度32、2层、上下文12的缩小模型检查三条日志列表各有39项；另用 HEAD 检查两处镜像、两个尺寸的文件长度。前者依赖固定数据和评价频率，后者只验证可访问性和长度，均不等于权重内容校验。本书未调用网络测试；自己的缩小训练直接检查损失与保存恢复。

13.9 练习与答案

补充题一：词表 16 项，完全均匀预测的 loss 和 perplexity 是多少？答案为 $\log 16 \approx 2.772589$ 和 16。

补充题二：把所有 logits 加 100，交叉熵是否改变？提示：把常数代入 `log-sum-exp`。答案：精确算术下不变，分子分母的公共因子抵消；稳定实现应在正常有限范围内保持这一性质。

补充题三：训练 loss 继续降，独立验证 loss 开始升，应先检查什么？答案包括数据泄漏、评价模式与分母、训练验证分布及过拟合；不能只凭一条曲线决定扩大模型。上游续训练习在第 15 章讨论恢复状态，不应把重新加载权重当作完整续训。

执行 `python examples/core_training/run_checks.py` 可重跑训练闭环，执行 `python examples/core_training/loss_sampling_checks.py qa/core_training/loss_sampling.json` 可检查交叉熵梯度与平均分母反例。

参考与覆盖

- [ch05 主 notebook](#)：目标概率、负对数平均、flatten、perplexity、划分、loader、训练和曲线。
- `gpt_train.py`：转换包装、批损失、loader 评价、模式切换、采样显示和 `main`。
- 同目录生成、checkpoint 与预训练参数单元分别在第 14—15 章解释，避免遗漏与重复计算覆盖。

第 14 章 采样、温度、top-k 与生成边界

模型给的是候选 token 的分布，生成程序才决定选哪个、是否继续以及怎样截取上下文。降低温度不会更新权重，top-k 也不会让模型学到新知识。它们改变的是同一组 logits 被使用的方式。

本章对应上游 ch05 的温度、频数、top-k、完整 generate 函数和生成练习；真实小实验只使用自造 logits，不把随机小模型输出写成自然语言质量证据。[上游生成脚本](#)

14.1 贪心选择不需要 softmax

若 logits 为 z_v ，贪心选择 $\arg \max_v z_v$ 。softmax 是严格单调变换后再除公共分母，因此有限分数的最大项不会改变。上游早期生成先 softmax 再 argmax，后面直接 argmax，数学上是同一选择。

贪心每一步选当前概率最大的 token，不等于整段序列联合概率最大。第一步稍低的候选可能带来后面更高概率的序列。它也容易重复某些高概率模式，但“随机一点”不保证事实更正确。

14.2 温度改变概率比值

温度 $\tau > 0$ 时，采样分布为

$$p_v(\tau) = \frac{e^{z_v/\tau}}{\sum_u e^{z_u/\tau}}.$$

两个候选的比值为 $p_i/p_j = e^{(z_i-z_j)/\tau}$ 。温度小，分数差被放大，分布更集中；温度大，差距被压缩，分布更平坦。只要温度为正，候选排序不变。 $\tau \rightarrow 0^+$ 时趋向最大项集合， $\tau \rightarrow \infty$ 时有限候选趋向均匀。

程序通常把 temperature=0 定义为贪心分支，不真的除以零。负温度会颠倒排序，通常不属于这个接口的有效输入，本书实现直接拒绝。NaN、全负无穷 logits 同样需要显式排错，不能交给 multinomial 后才猜原因。

本书 logits [0,1,2,3]，温度 0.8 的完整分布约为 [0.016894,0.058965,0.205807,0.718335]。这四个数由同一公式计算，和接近 1。随后若使用 top-k，还会再次删除候选并归一化，不能把删除前的概率当采样后的概率。

14.3 top-k 删除支持集之外的候选

top-k 找分数最大的 k 项，其余设为负无穷，再 softmax。与“把其余概率置零但不重新归一化”相比，这样自然满足采样分布总和为 1。应检查 $1 \leq k \leq V$ ， $k = 1$ 与贪心在无并列时一致， $k = V$ 不过滤任何项。

上游实现取第 k 大值作为阈值，只删除严格小于阈值的项。若出现并列，保留项可以超过 k 。例如 `[3,2,2,0]`、 $k=2$ ，会留下前三项。本书 `sample` 使用 `topk` 返回的具体索引再 `scatter`，严格留下两个位置。两种都是可定义的策略，但输出支持集不同；并列时具体索引顺序也不应假定跨后端永远一致。

正温度缩放不改变排序，所以无并列、有限值时，先 `top-k` 再除温度与先除温度再 `top-k` 的保留集合相同。数值精度、NaN 或额外过滤规则会让实际边界更复杂。

14.4 采样频数只能近似概率

`torch.multinomial(probs,1)` 从分布抽样。重复 1,000 次后，频数会围绕概率波动，不需要恰好等于概率乘 1,000。固定种子只是固定某次伪随机序列，并不使频数成为理论概率。

本书对 `[0,1,2,3]`、温度 0.8、`top-k=2`，seed 2718 的 1,000 次抽样只出现 ID 2 和 3，次数分别为 230 和 770。过滤后理论概率约为 $1/(1 + e^{1/0.8}) \approx 0.2227$ 与 0.7773。这个结果与分布相容，测试真正严格保证的是没有抽到被过滤项。

上游 `print_sampled_tokens` 重置 seed、循环抽样并用 `bincount` 统计，温度图比较同一 logits 的几种温度。`bincount(minlength=V)` 能显示一次也没抽到的末尾候选；不指定 `minlength` 时，输出长度只到实际最大 ID。不要把没有打印的一行误解成不存在的词表项。

14.5 一个生成循环应有明确的停止协议

完整流程是：截取上下文，模型前向，取最后位置 logits，过滤与温度处理，选择新 ID，检查 EOS，再决定是否拼接。上游生成脚本在检查 EOS 后才拼接，因此返回序列不包含新采到的 EOS。其他库可能包含 EOS，两者都可以，但调用方必须知道。

固定版脚本在指定 EOS 时只支持 batch size 1，并显式拒绝其他批大小；notebook 中较早的简单条件没有完整的批处理停止状态。对多个请求，不能直接用 `if tensor == eos_id`，因为多元素布尔张量没有唯一真值。通用批量生成需要每行的 `finished` 标记、已经结束行的填充策略，以及何时整个 batch 停止。

本书配套 `generate` 为降低隐藏状态复杂度，明确只接受一条非空提示词，返回结果不含 EOS，限制最大新 token 数，保存并恢复模型原来的训练模式。测试用一个总偏向 ID 3 的固定模型：EOS=3 时原提示 `[1,2]` 原样返回；不设 EOS、最多生成 3 个时得到 `[1,2,3,3,3]`。这是停止协议测试，不是文本生成展示。

14.6 上下文截断与 KV cache 不能混用旧假设

无缓存循环每轮取最后 C 个 token，位置从 0 开始。缓存循环只读新 token，并保留累计位置。超过上下文后，不能既截掉输入前缀，又继续使用没有同步裁剪或重算的旧缓存，否则程序实际看到的历史与读者以为的不一致。

即使设置 `max_new_tokens`，还应考虑 EOS、位置容量和服务端响应预算。提示词为空时没有“最后位置 logits”，需要规定 BOS 或其他初始化策略。生成函数不应悄悄接受空输入直到模型深处才报错。

14.7 复现采样时应记录的不止 seed

模型权重、tokenizer 版本和词表、提示词字节、模板、上下文裁剪、dtype、设备、温度、top-k、EOS 规则以及随机数消费顺序，都可能改变输出。前面多做一次随机采样，会改变后面的序列。GPU/MPS 与 CPU 即使 logits 极接近，也可能在候选分界处选择不同 token，后续自回归差异会迅速扩大。

上游集中生成脚本在温度分支 softmax 前减去行最大值，改善数值稳定；这不等于保证所有后端抽到同一序列。第 18 章区分随机数差异与计算误差。比较实现正确性时，优先比较 logits 和分布，再比较确定性贪心，最后才讨论随机文本。集中脚本的 CLI 接受 `--prompt` 和 `--device`，内部直接 `torch.device(args.device)`；虽然帮助文字举了 `auto`，固定实现没有解析 `auto` 的分支，实际应传 `cpu`、`cuda` 或 `mps` 等有效设备并验证可用性。模型尺寸由配置字典和名称提取的尺寸字符串共同决定，不能只改下载名字。

14.8 习题与复查

上游温度习题计算低概率候选在高温时的概率并观察抽样；贪心等价题比较 `generate_text_simple` 与 `generate(temperature=0, top_k=None)`；另有已训练与更大预训练模型的生成演示。前两项机制已在本章推导，权重加载见第 15 章；本书没有运行大权重质量比较。

补充题一：对 logits `[1, 1, 0]`，令温度趋近零，会得到什么？答案：概率集中到前两个并列最大项，各约一半；程序的 `argmax` 分支则按具体索引规则只取一个，不能把二者完全混同。

补充题二：为何 `top-k=1` 时正温度通常不改变结果？答案：支持集中只剩一个项，归一化后概率为 1。

补充题三：如果采样到 EOS 后不把它拼接，返回长度是否包含结束符？答案：不包含，调用者若需要显式记录结束原因，应另返回停止状态。

运行 `python examples/core_training/loss_sampling_checks.py qa/core_training/loss_sampling.json` 检查并列支持集、EOS 与模式恢复；主检查脚本记录 1,000 次采样频数。

参考与覆盖

- 主 notebook：温度曲线、频数、top-k 和生成循环。
- 生成脚本：EOS batch 限制、稳定 softmax、CLI、官方权重路径；加载部分见第 15 章。
- 习题：温度、贪心、续训、预训练模型评价与生成已分章映射。

第 15 章 权重格式、参数映射与低内存加载

下载一个权重文件和得到正确模型之间，还隔着结构、名字、形状、布局、dtype、设备及共享关系。只检查 `load_state_dict` 没有报错，不能证明 tokenizer 与模型兼容；只检查每层形状相同，也不能发现 Q、K 被交换或方阵忘记转置。

本章覆盖 ch05 主代码的保存恢复、TensorFlow GPT-2 导入、三种替代加载 notebook，以及低内存 `state_dict` notebook。所有实际运行只读写本书自己创建的小检查点，不下载预训练权重。[替代加载](#)

15.1 先区分推理恢复与训练恢复

推理需要结构配置、参数与必要 buffer、tokenizer 及模板约定。恢复训练还需要优化器状态、已完成更新步数、学习率调度状态、随机数状态，以及数据进度。AdamW 的一阶、二阶动量丢失后，下一步更新通常不同，即使权重完全一样。

上游示例分别保存 `model.state_dict()` 和包含 `model_state_dict/optimizer_state_dict` 的字典。恢复顺序是建立模型、载入模型状态、用新模型参数建立优化器、再载入优化器状态。不能把旧优化器继续指向一个已经被替换掉的模型对象。

本书 `checkpoint_checks.py` 在小模型训练 3 步后保存模型、优化器、CPU RNG、配置和 step，继续第 4 步作为基线；另一条路径加载后做相同第 4 步，全部浮点参数最大差为 0。这里数据顺序固定、dropout 为零，没有验证多进程 DataLoader 的精确续跑。记录哪些状态尚未覆盖，比笼统宣称“完全可复现”更有用。

15.2 官方 GPT-2 的名字怎样映射到教学模型

上游 TensorFlow checkpoint 读取器列举变量名、读取数组、去掉多余单维，把 `model/` 前缀后的路径拆开。形如 `h0/...` 的变量放进第 0 个 block，其他项进入全局参数字典。`setdefault` 逐级创建嵌套字典，最后存实际数组。[读取与映射代码](#)

对应关系如下：

源参数	教学模型目标	必要变换
wte	token embedding、输出头初值	[V,D] 直接复制
wpe	position embedding	[C,D] 直接复制
attn/c_attn/w	Q、K、V 权重	最后一轴拆三份，再各自转置
attn/c_attn/b	Q、K、V 偏置	最后一轴拆三份

源参数	教学模型目标	必要变换
attn/c_proj/w	注意力输出投影	转置
mlp/c_fc/w	FFN 第一层	转置
mlp/c_proj/w	FFN 第二层	转置
ln_1/ln_2 的 g,b	两层 LayerNorm scale、shift	直接复制
全局 g,b	final LayerNorm	直接复制

源合并 QKV 权重为 $[D, 3D]$ ，拆分后每份 $[D, D]$ 。因为方阵转置后形状不变，形状检查无法发现漏转置。FFN $[D, 4D]$ 的转置更容易被形状检查抓住。应使用有不同元素的测试矩阵，而不是全零或单位矩阵来验证布局。

载入 GPT-2 参数前，上游把上下文容量恢复为 1024，并启用 QKV 偏置。训练示例的 256 上下文和无 QKV 偏置配置不能直接接收全部原权重。

15.3 assign 会替换参数对象

上游 `assign(left, right)` 先检查 `shape`，再构造新的 `nn.Parameter` 返回给模块属性。TensorFlow 数组通过 `torch.tensor` 转成张量；Hugging Face 变体使用 `clone().detach()`；safetensors 变体使用 `detach()`，可能保留底层存储共享。这些选择影响复制量与别名关系。

替换 `Parameter` 后，之前创建的优化器可能仍持有旧对象。配套测试明确演示了这一点：把 `token embedding` 属性换成同数值的新 `Parameter`，旧优化器仍引用旧权重，且不包含新权重。安全顺序是完成所有替换后再建优化器，或在已有参数对象上用 `no-grad` 的 `copy_` 保持身份。

输出头与 `token embedding` 都从 `wte` 加载，不一定意味着同一 `Parameter`。复制数值、共享存储、共享参数对象是三个层次。训练中的真正权重绑定应显式核对对象关系和梯度路径，而非只比较初始化数值。

15.4 三种替代文件入口

`weight-loading-pytorch.ipynb` 使用已经转成教学模型命名的 `.pth`，因此可以直接 `load_state_dict`；同一 notebook 也演示同命名的 `safetensors` 文件。它省去了外部参数名转换，但仍需要正确模型配置。

`weight-loading-hf-transformers.ipynb` 通过 `GPT2Model.from_pretrained` 获得 Hugging Face 模型，再从其 `state_dict` 按 `h.{b}.attn.c_attn.weight` 等名字映射。源模型没有独立语言建模头时，目标输出头使用 `token embedding` 权重。该版本在 `block` 循环内重复设置 `final norm` 和输出头，最终结果可以相同，但重复赋值和复制不是必要工作，搬到循环外更清楚。

`weight-loading-hf-safetensors.ipynb` 直接读取 Hugging Face `model.safetensors`，不先实例化完整源模型，随后进行相同语义映射。文件格式避免了通用 Python 对象反序列化，但不能保证权重来源可靠、架构正确或模型行为安全。固定 notebook 对其中一个尺寸附有生成问题注释，本书没有下载核验，因此不把所有尺寸写成已通过。

TensorFlow 路径还依赖 checkpoint 元数据和相关库；直接 PyTorch/safetensors 路径减少了格式桥接步骤。它们解决的是不同输入文件的适配，不是三种不同 GPT 结构。

15.5 下载器的成功条件不能只看文件存在

上游下载器验证允许的模型尺寸，创建目录，下载 checkpoint、词表、超参数和数据文件。流式读取、超时和 HTTP 状态检查能避免部分网络错误。某些版本以 Content-Length 与本地文件大小相同作为跳过条件；同大小不证明内容相同，正式归档应记录来源 revision 与内容哈希。

独立 `gpt_download.py` 还提供备用镜像：内部 `_attempt_download` 先试主地址，网络异常时再试备用地址；全部失败时打印错误，但调用方仍应显式检查文件齐全，不能继续把缺失文件交给 TensorFlow。`gpt_generate.py` 内嵌的下载版本没有同样的备用参数，这是两个重复实现的实际差异。

直接覆盖目标文件可能在中断后留下不完整文件。更稳健的流程是写临时文件、校验、再原子替换，并核对所有分片齐全。本书未实际下载任何大文件，相关函数只做源代码解释。不能把远端 `main` 文件路径当成永远不变的版本标识。

15.6 加载峰值为何可能接近两份权重

常见路径先建立随机模型并移到 GPU，再把整个 `state_dict` 加载到 GPU，最后逐项复制。加载期间，目标参数与源字典可能同时存在，相当于额外一整份权重；初始化还消耗时间和随机数。

上游内存 notebook 比较了多种策略：先在 CPU 建模和加载再整体迁移；GPU 模型配 CPU `state_dict` 逐张量复制；`meta` 构建后 `to_empty`；`meta` 加 `mmap` 与 `assign=True`；以及每个参数独立文件、逐个读取。不同方法把成本转移到 CPU、GPU、文件映射或小张量临时空间，不应只看最终常驻内存。

`meta` 张量只有形状、`dtype` 等元信息，没有可用数值；`to_empty` 分配未初始化存储，不会自动重建原本的三角 `mask`。不能对未初始化模型直接推理。

15.7 低内存加载不能漏掉 buffer

上游部分“逐参数复制”示例只遍历 `named_parameters()`，而保存时 `state_dict` 包含持久 `buffer`。若目标模型正常初始化，因果 `mask` 可能恰好已正确生成，遗漏暂时不暴露；若模型在 `meta` 上构造再 `to_empty`，`mask` 也变成未初始化存储，只复制参数就不够了。

本书测试用确定性的全 False mask 代表“没有正确恢复的 mask”，避免读取随机未初始化内存。参数全部正确时，这个错误 mask 仍使 logits 最大偏差达到 0.433731。它证明前向所需的 buffer 必须一起恢复，不意味着未初始化内存每次恰好产生这个数字。

完整 load_state_dict 能统一处理参数和持久 buffer，并报告缺失或多余键。若确实逐个文件加载，应遍历完整状态、核对每项 dtype/shape/device，并恢复可重建的非持久状态。仅打印 missing warning 后继续使用模型不适合作为严格验收。

15.8 mmap 与 assign 的小型实测

本书在 CPU 上运行：

```
with torch.device("meta"):
    model = TinyGPT(**config)
    state = torch.load(path, map_location="cpu", weights_only=True, mmap=True)
    model.load_state_dict(state, assign=True)
```

mmap=True 让支持格式的本地存储可由文件映射，assign=True 让模块采用 state_dict 的张量属性，避免先创建一套随机参数再逐项复制。所有持久 buffer 同样加载。测试确认没有参数留在 meta，模型输出与原模型逐位相同。

采用 assign 后应在加载完成之后创建优化器，并核对共享权重关系。文件映射也不意味着数据永远不占物理内存；访问页面时操作系统仍需提供实际数据。本书没有报告峰值 RSS 或 GPU 内存节省量，只验证该 API 路径在 PyTorch 2.6.0+cpu 的正确性。

15.9 如何解读内存监控代码

上游 start_memory_tracking/print_memory_usage/cleanup 使用 CUDA 峰值统计、垃圾回收和缓存清理；没有 CUDA 时，部分后续调用仍不适合直接执行。max_memory_allocated 统计的是张量分配峰值，不等于整个进程占据的所有显存。

CPU 监控函数每 0.1 秒读取 RSS，用单独线程收集样本后减去基线。它可能漏掉更短的峰值；若被测函数抛错，还应在 finally 中停止监控线程；采样列表为空也要处理。不同实验在同一进程里的缓存和映射状态会影响基线，不能把一次数字当作普适结论。

逐权重文件方案降低单次源张量大小，却增加文件数量、目录开销和 I/O 管理。还需要清单、哈希、全部状态完整性以及中断恢复。选择方案时应先明确瓶颈是 CPU 内存、GPU 峰值还是读取时间。

15.10 练习与答案

上游续训练加载模型和优化器再训练一轮；预训练模型评价练习比较不同 GPT-2 尺寸；生成练习继续使用载入的模型。机制见本章和第 13—14 章，真实大模型比较未运行。

补充题一：QKV 权重拆分后都是方阵，为什么 shape 检查还不够？答案：Q/K/V 顺序和转置错误都可能保持 shape，必须用数值与前向对照。

补充题二：只恢复权重、没有恢复 AdamW 动量，下一步一定相同吗？不能保证相同，通常会不同，因为优化器历史影响更新。补充题三：meta 后逐参数复制为何仍可能看见未来？答案：因果 mask 是 buffer，没被参数遍历恢复。

执行 `python examples/core_training/checkpoint_checks.py qa/core_training/checkpoint_checks.json` 可复查续训、meta/mmap/assign、buffer 反例和优化器旧引用。

参考与覆盖

- 主加载与生成脚本、下载器。
- 三种替代加载 notebook。
- 低内存 `state_dict` notebook：各加载路径与 GPU/CPU 监控均已解释，未运行大模型内存基准。

第 16 章 语料整理与 Gutenberg 训练工程

把训练文本从一篇变成一千篇，困难不只是循环多跑几次。文件编码、重复文档、边界标记、分片大小、顺序、验证集和中断恢复都会影响训练。数据管道的错误往往仍会输出一个下降的 loss，因此要先检查数据产物，再启动训练。

上游 Gutenberg bonus 提供整理脚本和按文件训练脚本。本书只分析公开代码，并用六条自造记录验证小型离线管道；没有下载整套 Gutenberg 语料，也没有据此训练模型。[固定版本目录](#)

16.1 语料清单应先于合并大文件

每份文档至少需要稳定标识、原始来源、原始字节哈希、解码方式、清理版本、保留/排除原因和训练划分。公开可下载不等于所有使用方式都已获得许可；实际项目还需核验具体材料与使用要求。本书小管道仅使用新写的短句，不依赖第三方正文授权判断。

文件顺序会影响分片与训练顺序。`os.walk` 的返回顺序不应被当作跨系统稳定顺序，整理前先排序，并在清单记录最终顺序。同一语料内容若因换行风格不同而被当成两份，也会污染统计，因此要区分原始字节哈希和按明确规则规范化后的内容哈希。

16.2 上游清理脚本逐步做了什么

`combine_files` 按路径读取，先尝试 UTF-8，失败后回退 latin1；用 `is_english` 的 ASCII 字符占比过滤，再调用外部 Gutenberg cleanup 的 `strip_headers`，压缩多余空行，估算 UTF-8 字节数，最后用 `<|endoftext|>` 拼接进大文件。[prepare_dataset.py](#)

每一步都有边界。latin1 可以把任意字节解释成字符，因此“成功解码”不证明文本本来就是 latin1，可能产生乱码。ASCII 占比高只能说明字符集合偏向 ASCII，不能证明语言是英语；其他拉丁文字、数字表格或代码也可能通过。空字符串会让占比分母为零。该函数名是教学启发式，不应当作经过验证的语言识别器。

删除头尾元数据可能改善语言建模，但原始出处与处理记录应保留在旁边的清单，不能随训练文本一起丢掉。空行压缩也会改变诗歌、代码或格式化文本结构，是否适用取决于目标。

16.3 分片大小要把分隔符也算进去

上游累加每篇清理后内容的 UTF-8 字节数，却没有把文档间 separator 的字节算入 `current_size`。因此最终文件可能略超预算。如果第一篇就超过预算，代码会先写当前空列表，产生空分片；超大单文档本身也不能靠“换新文件”缩小。

本书 `pack` 明确规定：预算包含分隔符；只有当前分片非空时才写出；单篇超大文档独占一个分片，并由调用者标记其超预算状态。这个策略不等于绝对硬限制，若系统要求每片都不超限，还需在合法边界拆文档并记录新边界。

字符数和字节数不同。“山与海”有三个字符，但 UTF-8 为 9 字节。分词后的 token 数又是第三种单位。磁盘分片预算用字节，训练上下文预算用 token，不能互换。

16.4 去重应发生在划分审计中

本书小例子有一份 CRLF 文档和一份 LF 版本，规范化换行后哈希相同。按稳定名字顺序保留第一份，剔除第二份，记录“exact duplicate”。这只做精确去重，不识别近似改写、长段重复或同书不同版本。

六条输入保留四条、剔除一条重复和一条空文档。训练与验证哈希集合不相交。训练三条内容按 64 字节预算打包成两片，实际 42 和 47 字节。分隔符 `\n<|endoftext|>\n` 本身是 15 字节，测试专门检查不能漏计。

这些小断言能暴露数据管道常见错误，但不能保证真实大语料无重复。更完整的去重需要明确比较单位、相似度规则与误删风险，并对训练验证边界单独审计。

16.5 按文件训练怎样管理进度

上游 `read_text_file` 读取一个合并文件，再附加结束符。`create_dataloaders` 在该文件内部按字符比例划分训练和验证，沿用固定长度、stride 等于上下文的窗口。每个文件重新创建 loader，模型和优化器持续保留，所以训练状态跨文件延续。

这种方式降低一次持有全部原文的需求，但仍会把当前大文件完整读入、分词并存窗口。上游 Dataset 还物化了输入与目标张量，可能重复保存大量 ID。真正的大语料实现通常要考虑预分词、内存映射 token 数组、按索引切片或流式读取；本书不把当前逐文件实现描述为常数内存流式训练。

验证 loss 每次来自当前文件后段，会随文件主题变化。若要比跨训练过程的模型进步，应另保留固定验证集合。否则某个时间点 loss 升高可能只是切到更难的文件，而不一定是训练退化。

16.6 检查点频率代码值得逐字符核对

固定版本训练脚本在文件结束后写：`if global_step % save_ckpt_freq:`。Python 把非零余数当 True，因此它不是“每逢整除时保存”，而是大多数非整除时保存；位置又在文件循环末尾，无法保证精确每 N 个优化器步骤触发。

若意图按正整数步数定期保存，应在更新循环内检查 `(step + 1) % frequency == 0`，或先定义清楚 `step` 从 0 还是 1 开始。本书小测试 `frequency=3` 时得到步骤 3、6、9。还应在正常完成和中断时保存，并防止异常中尚未定义有效 `step`。

上游捕获 `KeyboardInterrupt` 后保存模型权重，这能留住已训练参数，但不等于精确断点续训。需要再保存优化器、`epoch`、文件索引、文件内批次位置和随机数状态，第 15 章已有最小续训对照。输出目录也应独立命名，避免不同实验覆盖同名检查点。

16.7 ETA 与日志也有计量假设

`convert_time` 用 `divmod` 把秒转成时、分、秒。`print_eta` 用已处理文件平均时间乘剩余文件数，因此假定文件成本近似。若分片大小差很多、首次分词慢或后面发生验证与保存，ETA 只能是粗估。

脚本在多 `epoch` 情况下用从训练开始的累计时间、当前 `epoch` 内的文件 `index` 估算，时间与计数的范围可能不一致。更可靠的是用已处理 `token` 或总批次数，并明确包含哪些 I/O、验证和生成时间。吞吐统计也应区分训练 `token` 与评估 `token`。

16.8 CLI 与测试入口的边界

上游提供 `data_dir`、`output_dir`、`epochs`、采样间隔、评价间隔、保存频率、学习率、`batch_size` 和 `debug` 配置。`type=bool` 解析命令行字符串容易误导：非空字符串 `"false"` 在 Python 中也是真。更合适的是 `store_true` 或明确布尔解析。

`debug` 模型缩小宽度、层数和上下文，但仍保留 GPT-2 的 50,257 词表，且依赖分词器及包接口。上游测试写入重复字符串，调用 `debug` 脚本，只检查 `stdout` 中是否出现最终内存提示；它是 `smoke test`，不是 `loss`、标签、数据划分或权重恢复的严格验证。重复字符串直接相乘还会使片段末尾与下一段开头贴在一起，若需要边界，必须显式加分隔符。

没有文件时脚本退出；频率参数为零会导致取模错误；过短文件可能造不出窗口。正式入口应在开始前检查这些条件，并把跳过文件和原因记录下来，而不是到训练一半才失败。

16.9 可运行的小管道与练习

运行：

```
python examples/core_training/corpus_demo.py qa/core_training/corpus_demo.json
```

程序只处理代码中自造的六条文本，不写外部语料，也不启动模型训练。它验证空输入、空文档、换行规范化、精确重复、划分隔离、UTF-8 字节预算、超大文档策略和保存步数。主训练链路已在第 13 章独立验证，把两者分开有助于定位错误。

补充题一：两个各 10 字节文档、15 字节分隔符，能装进 30 字节分片吗？答案：不能，共 35 字节。补充题二：ASCII 占比 100% 的文档一定是英文吗？答案：不是，例如数字和源代码。补充题三：为什么把所有窗口随机分训练验证可能比按文档分更乐观？答案：重叠上下文和重复内容可能跨集合出现。

参考与覆盖

- `prepare_dataset.py`: 编码回退、ASCII 启发式、清理、空行、分片和 CLI。
- `pretraining_simple.py`: 逐文件 loader、训练、评价、生成、保存、ETA、中断、debug。
- 测试入口：仅说明 smoke test 的覆盖程度，未运行完整 Gutenberg 下载或训练。

第 17 章 学习率、梯度裁剪、超参数与 Muon

一个训练循环能下降，不代表它已经适合更长训练或更大模型。学习率决定每步更新幅度，裁剪限制异常梯度，超参数搜索决定怎样比较实验，优化器则规定如何利用梯度历史。这些机制最好逐项引入，每加一项都先检查边界。

上游 ch05 的 scheduler 目录指向 Appendix D，另有网格搜索脚本和 Muon 双优化器示例。本章覆盖这些路径，不把短文本上的一次结果当作优化器排名。[Appendix D](#)

17.1 先约定 step 表示什么

设 s 是从 0 开始的更新索引，warmup 长度为 W ，余弦终点为 S ，要求 $0 \leq W < S$ 。初始、峰值和结束学习率分别为 η_0, η_p, η_f 。本书定义

$$\eta(s) = \begin{cases} \eta_0 + (\eta_p - \eta_0)s/W, & 0 \leq s < W, \\ \eta_f + \frac{\eta_p - \eta_f}{2} \left[1 + \cos\left(\pi \frac{s-W}{S-W}\right) \right], & W \leq s \leq S, \\ \eta_f, & s > S. \end{cases}$$

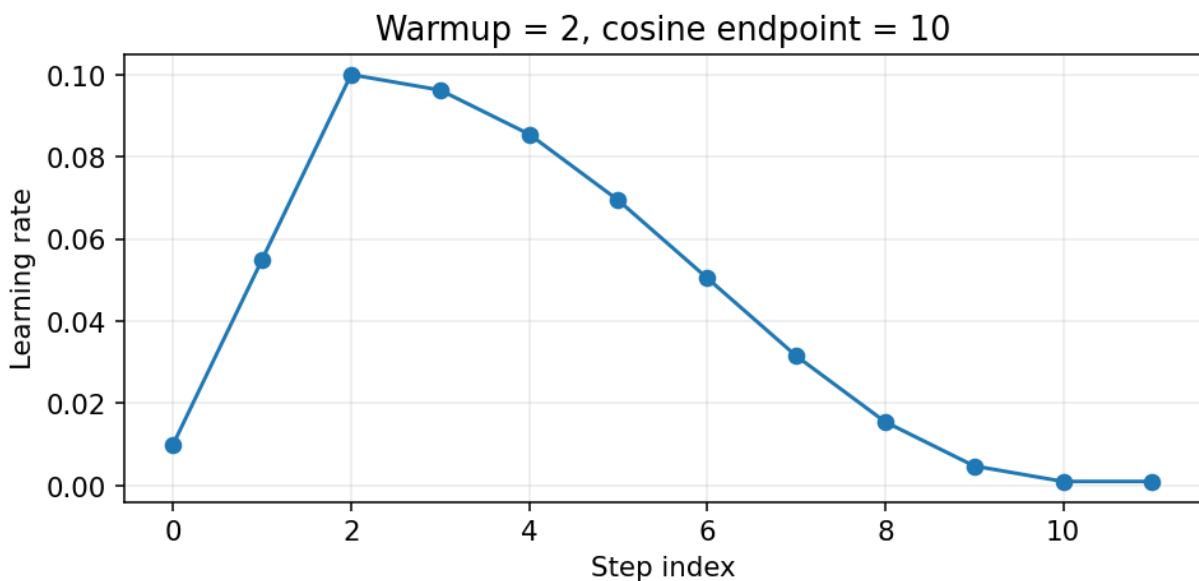
当 $W = 0$ ，直接跳过 warmup，不能计算除以零的增量。 $s = W$ 正好到峰值， $s = S$ 正好到结束值。若只实际更新 S 步、索引 0 到 $S - 1$ ，最后一步尚未取到终点值；这不是公式错，而是“终点”与“更新次数”的约定需要一致。

上游 Appendix D 用 global_step 从 -1 开始、更新前加一；网格搜索从 0 开始、每次先加一，且 warmup 条件使用 $\leq$ 。两者都能定义合理日程，但第一步学习率不同，不能只比较相同 warmup 数字就说完全一样。

17.2 为什么先升后降

初始参数和优化器统计尚未适应数据时，较小的学习率可以限制早期更新；warmup 逐步达到目标步幅。后期降低学习率则缩小每次移动，帮助在当前区域细调。这是机制解释，不是保证每个任务都需要相同 warmup 比例。

本书示例 $W = 2, S = 10, \eta_0 = 0.01, \eta_p = 0.1, \eta_f = 0.001$ 。实测索引 0、1、2 的值为 0.01、0.055、0.1；索引 6 为 0.0505；索引 10 和 11 都为 0.001。日程超出终点后钳在结束值，避免未经限制的 cosine 再次上升。



上游前两个日程展示循环只记录 learning rate，计算 loss 和更新权重处是注释。这是可视化日程，不是已经训练了模型。最终 `train_model` 才把日程、反向、裁剪、更新和评价放在一起。

17.3 梯度裁剪缩放整组向量

把所有参数梯度拼成一个向量 g ，范数裁剪阈值为 c ，更新前改成

$$\tilde{g} = g \min \left(1, \frac{c}{\|g\|_2 + \epsilon} \right).$$

若 $g = (3, 4)$ ，范数为 5，阈值 1，结果近似 $(0.6, 0.8)$ 。方向保持，长度被限制。逐元素 clamp 到 $[-1, 1]$ 则得到 $(1, 1)$ ，方向已经改变，二者不能混称同一种裁剪。

`clip_grad_norm_` 在 `backward` 后、`optimizer.step` 前执行，并返回裁剪前的总范数。配套实测为 5，裁剪后因实现 `epsilon` 得到约 $[0.5999999, 0.7999998]$ 。检查时用容差，不要求十进制打印恰好等于 0.6、0.8。

上游 `find_highest_gradient` 找的是各梯度中的最大有符号元素，不是最大绝对值，也不是总范数。如果梯度全为负，打印出的“最高”值尤其不能代表最大幅度。可用它观察某种变化，但不能据此判断全局裁剪是否正确。

17.4 日程和裁剪的边界要一起测

固定 Appendix D 保留 `ORIG_BOOK_VERSION` 分支，说明 `>` 与 `>=` 的一位差别会跳过 warmup 结束后的某次裁剪。当前默认分支在 `global_step >= warmup_steps` 时裁剪。是否应该 warmup 时也裁剪属于训练设计，不是数学必需；如果目标是始终限制梯度，可以每次 `backward` 后都裁剪。

空 loader、总步数不大于 warmup、零 warmup 都需要处理。把 warmup 占总步数 20% 后取整，小实验可能得到零。梯度非有限时还应先排查 loss 与数据，而不是假定裁剪能把 NaN 修好。

混合精度若使用梯度缩放，应先 unscale 再按真实梯度阈值裁剪；梯度累积则在所有 microbatch 累积完成后裁剪一次，才对应整批梯度。否则每个 microbatch 各自改变方向和比例，结果不再等同于整体裁剪。

17.5 网格搜索的乘法预算

上游网格有批量 4 档、dropout 3 档、warmup 3 档、weight_decay 3 档、peak_lr 4 档、initial_lr 2 档、min_lr 3 档、epoch 5 档，总计

$$4 \times 3 \times 3 \times 3 \times 4 \times 2 \times 3 \times 5 = 12960$$

组。每组都重新建模型、数据 loader 和优化器，训练后按验证 loss 更新 best 配置。这个数量说明：随意增加一个维度的档位会乘法放大成本，不能把一个教学网格直接当成可接受的默认任务。

比较时还要统一预算。不同 batch_size 在相同 epoch 下更新次数不同，warmup_iters 占比也不同；不同 n_epochs 又改变总 token 次数。若目标是比较计算效率，应按同 token 或同计算预算评价，而不只看最终最低 loss。

上游把每组 seed 重设为 123，有助于控制初始化；但数据顺序、批量、dropout 的随机数消费仍不同。eval_iter=1 只看一个验证 batch，噪声与偏差都可能很大。调参反复看同一验证集后，最终报告还需要未参与选择的测试集合。

17.6 搜索脚本的失败路径也属于实验

warmup 大于总步数时，训练可能始终在 warmup；warmup 等于总步数时余弦分母为零的边界要避免；initial_lr 高于 peak_lr 时所谓 warmup 实际在下降。应在生成配置时先过滤不合意组合，并记录被过滤原因。

上游打印最优配置时，有的训练 loss 变量来自最后一次运行，不一定是最佳验证配置对应的训练 loss。中断也可能发生在第一次成功结果之前。稳健记录应把配置、训练/验证指标、步数、seed 和状态保存在同一条结果里，而非依靠多个全局变量碰巧同步。

本书不执行 12,960 组搜索。它保留搜索空间计算与日程边界断言，读者可以先用小模型、少数组合验证整个记录流程，再决定实际预算。[hparam_search.py](#)

17.7 Muon 改变的是矩阵更新方向

Muon 针对二维矩阵参数，使用动量并对矩阵更新方向做正交化处理。上游引用的研究强调权重衰减和更新尺度对大模型训练的重要性；其大规模结果属于论文实验，本书没有复现。[Muon is Scalable for LLM Training](#)

用一个独立数学例子理解“正交化”：以满秩矩阵的薄 SVD $G = U\Sigma V^T$ 为例，把奇异值的幅度替换为 1，得到方向 UV^T 。对满秩方阵 $G = \text{diag}(4, 1)$ ，这个方向是单位矩阵；原梯度在两个主方向的幅度比是 4:1，正交化后变为 1:1。因此它不是简单把整个矩阵除以一个全局范数。

本书用 float64 SVD 验证这个小例子，并检查方向转置乘自身为单位矩阵。这个程序只是解释矩阵方向，不是完整 Muon：实际优化器还包含动量、可选 Nesterov、近似正交化迭代、尺度规则和权重衰减。PyTorch 的 Muon 接口使用 Newton-Schulz 相关设置及学习率调整选项；具体参数以所用版本为准。[PyTorch Muon 文档](#)

17.8 两个优化器必须不重不漏地分参数

上游 `create_muon_optimizers` 先遍历 Embedding 模块收集其参数名，再遍历所有可训练参数：二维、且不属于 embedding 的参数交给 Muon；其他交给 AdamW。于是注意力与 FFN 矩阵进入 Muon，embedding、偏置和归一化向量进入 AdamW。独立输出头也是二维非 embedding，因此在这个具体示例中进入 Muon，这不是所有 Muon 配方统一的选择。[双优化器源码](#)

本书按参数对象身份做同样的分组审计，小 GPT 的 Muon 候选共 6,400 个元素，AdamW 候选 864 个，总和 7,264，交集为空。权重共享时更应按身份去重，防止同一参数被两个优化器各更新一次。冻结参数不应进入任一组。

训练每步对两个优化器都清零，只 backward 一次，再依次 step；保存续训时必须保存两份优化器状态。每组可以有不同学习率，调度器也要分别管理，不能用一个 lr 无意覆盖两组比例。`max_steps` 在更新后检查，若允许 0，需要定义是否真的零步，否则简单后检查会先更新一次。

本书环境 PyTorch 2.6.0+cpu 实测没有 `torch.optim.Muon`，上游函数会明确抛出 `RuntimeError`。我们没有升级共享环境，没有运行 Muon 训练，也没有用自写 SVD 例子冒充其实现。分组与数学方向检查是已完成部分，正式 Muon 优化器路径保留为未运行。

17.9 练习与答案

补充题一： $W = 0$ 时应如何计算学习率？答案：跳过线性段，从余弦峰值开始，不能先算除以 W 的增量。

补充题二：把两个 microbatch 的梯度各自裁剪再相加，是否总等于相加后裁剪？答案：不等，裁剪是非线性操作。例如两个梯度方向不同、范数不同，各自缩放比例会改变合成方向。

补充题三：embedding 是二维矩阵，为何没有自动分给 Muon？答案：分组是优化配方的选择，不能只看维数；上游明确排除 embedding，并为它使用 AdamW。补充题四：只保存主模型权重后恢复双优化器训练会缺什么？答案：两组动量/统计、各自学习率与调度阶段，以及数据和 RNG 进度。

执行 `python examples/core_training/optimizer_numerics.py qa/core_training/optimizer_numerics.json` 可复查分组、SVD 示例和日程非法边界；主检查脚本记录完整日程与梯度裁剪值。

参考与覆盖

- [Appendix D](#)：日程展示、梯度检查、修正分支、综合训练和绘图。
- [scheduler](#) 入口、网格搜索。
- [Muon baseline](#) 与变体：复用训练、评价、图和生成见第 13—14 章，新增分组与双优化器在本章。

第 18 章 训练性能及 CPU、MPS 数值差异

优化一个实现时，应该同时回答两个问题：它少花了哪些时间或内存，仍然在计算什么。把模型转为 bfloat16、替换注意力内核、增加批量或扩大词表，都可能提高吞吐，却也可能改变数值、目标函数或训练预算。

本章覆盖上游 00_orig.py、01_opt_single_gpu.py、02_opt_multi_gpu_ddp.py，以及 CPU/MPS 随机采样 notebook。只运行了 CPU 数学和随机数检查，没有执行 CUDA、MPS、torch.compile 性能或真实 DDP。[性能脚本目录](#)

18.1 先建立可比较的基线

上游 baseline 基于主训练代码，但增大上下文和批量，并换成更长文本，新增区间 token/s 和内存统计。它与第 13 章短文训练的资源占用不同，不能把两者的差异全部归因于某个优化。

优化版集中做了合并 QKV、SDPA、原生 LayerNorm/GELU、bfloat16、fused AdamW、pinned memory、编译、词表对齐和更大批量。要知道单项贡献，最好每次只改变一项，并保留数值与训练轨迹检查。累积加入多项的表格描述的是那条实验路线，不保证各项独立加速比可以相乘。

上游 README 的 A100 吞吐数字是作者在其硬件与软件组合上的结果，本书没有复测，不把它们列为本地实验成绩。读者应优先复现固定输入、固定步数和计时范围，而不是追逐一个没有环境说明的目标数字。

18.2 减少中间张量和调用开销

原生 LayerNorm 和 GELU 可以调用优化过的实现，避免教学版的若干逐步运算。替换时仍要一致：LayerNorm epsilon、总体方差定义、GELU 是否采用 tanh 近似都影响结果。本书第 6 章已经展示即使公式相同，常数精度也会造成微小差异。

SDPA 把注意力核心交给库，可能使用适合设备的融合实现，减少显式 [B,H,T,T] 中间存储。合并 QKV 减少三次独立线性投影调用。普通因果模式通过 is_causal=True 表达，不再让每层永久保存完整 float mask。这些机制的正确性可用第 4—5 章的前向与梯度对照检查。

18.3 TF32、bfloat16 与 autocast 不是同一件事

TF32 相关设置影响某些支持设备上的 float32 矩阵乘法路径；bfloat16 改变张量数据类型；autocast 则在运算级选择类型，通常不等于把整个模型参数永久改成 bfloat16。上游优化脚本明确执行 model.to(device).to(torch.bfloat16)，并非一个保留全部 float32 参数、只在前向里 autocast 的示例。

bfloat16 指数范围较宽，但有效尾数位少，较小更新可能因舍入难以反映在低精度参数里。优化器状态的 dtype 与实现同样要核对。低精度方案应检查 loss 是否有限、梯度是否合理，以及与高精度参考的误差，而非仅看速度。

上游按 CUDA capability 条件设置 float32 matmul precision，并注明这不影响已使用 bfloat16 的矩阵乘法。不能把“开启 TF32”与“bfloat16 自动使用合适内核”重复当成同一运算的两份收益。

18.4 pinned memory 在主机，不是预留显存

DataLoader 的 `pin_memory=True` 使用页锁定的主机内存，可能改善 CPU 到 GPU 的传输。它不是预分配 GPU 权重或重复使用显存。与合适的非阻塞传输、流水线配合时，才可能更好地重叠读取和计算。

上游数据传输仍主要写 `.to(device)`，因此不能仅凭 `pin_memory` 开关就声称所有数据搬运都异步重叠。增加 `num_workers` 也可能增加进程、内存和序列化成本。短数据、内存中已构造好的 `tensor` 数据集，未必因更多 worker 更快。

fused AdamW 把部分更新操作融合，减少调用与中间数据开销。它受设备、dtype 和版本支持限制。CPU fallback 的存在不意味着脚本里所有优化选项都在 CPU 可用，应分别检查 `compile`、`fused optimizer` 和 bfloat16 算子支持。

18.5 编译收益需要排除首次成本

`torch.compile` 可能合并操作、优化图和生成更合适的代码。第一次调用含编译成本，输入形状或控制流变化可能触发新图或回退。因此要同时报告冷启动和稳定态，而不是丢掉慢的几步后只说“总共加速多少”。

编译后的模块保存名可能带包装前缀，保存时应理解是否需要访问原始模型；否则另一个未编译模型加载会出现键名不匹配。上游文件末尾提供了相关保存提示，但没有构成完整跨版本检查点协议。

编译工具链失败可以先跳过编译继续验证数学正确性，不必为了运行教学实验而安装不明来源编译组件。本书共享 CPU 环境没有执行编译性能路径。

18.6 词表对齐增加的行不是合法文本 token

上游把词表从 50,257 扩为 50,304，使矩阵维度更适合某些硬件路径。额外 47 行不自动对应 tokenizer 的有效 ID。训练标签仍只含原词表 ID，但输出 softmax 若包含新增行，分母和优化目标会改变；生成时若选到新增 ID，解码器可能无法处理。

本书小反例把一个无效补齐类的 logit 设为 100，贪心立刻选到它。明确屏蔽 `logits[... , valid_vocab:] = -inf` 后才回到合法候选。是否在训练也屏蔽，应作为目标定义的一部分记录。矩阵对齐是工程选择，不应悄悄改变 tokenizer 协议。

18.7 计时边界能让同一个数字有不同含义

上游 `train_model_simple_with_timing` 在评价间隔计算新增 token 数与经过时间，CUDA 路径使用 `event` 和 `synchronize`，CPU 用墙钟。第一次区间很短且可能含初始化开销，单 GPU 平均值跳过第一次；DDP 变体的累计规则不同，比较时必须看实际实现。

定时器在评价前重启，因此下一段时间可能包含前一次评价、打印和 epoch 末尾生成的开销；分子却只增加训练 token。这可以作为某种包含附带工作的有效吞吐，但不能称作纯优化器步骤吞吐。若想只测训练，应在评价完成后重新开始计时；若想端到端，应清楚包含数据加载、评价和保存。

`allocated memory` 是当前被张量使用的分配量，`reserved memory` 是分配器保留量，二者都不等于峰值。某些脚本把 `device` 局部变量重新赋成当前 CUDA 整数编号，后续传递时应保持类型约定清楚。测量代码本身也需要审查。

18.8 DDP 同步参数梯度，不会自动汇总所有指标

DDP 版由 `torchrun` 提供进程环境，设置设备和进程组，使用 `DistributedSampler`，编译后包装模型。数据下载由指定进程协调，并通过 `barrier` 等待其他进程看到文件；这隐含文件路径可访问的条件。训练每个 epoch 调 `set_epoch`，各进程执行相同数量的同步步骤。

梯度平均在等量局部样本时对应整体平均。配套 CPU 数学模拟把四个样本分成两半，局部梯度平均与整体梯度最大差约 5.55×10^{-17} 。这只是代数检查，没有启动进程或通信后端。

上游吞吐把各 rank 的 token 数 all-reduce 求和，再除以本地 `elapsed`；更严格的全局端到端测量应考虑最慢 rank 和同步边界。验证 loss 没有同样自动加权 all-reduce，rank 0 打印自己的值不能被称为全局验证 loss。`DistributedSampler` 默认补齐样本也可能造成评估重复，需要记录有效样本权重。

只有 rank 0 生成样例时，还要考虑 DDP forward 内部可能发生的同步或 buffer 广播。其他 rank 如果进入下一轮训练，控制流不一致可能等待或死锁。独立生成通常应使用未包装模块，并在各 rank 明确同步阶段；不能因为只想“少打印几行”就忽略分布式调用顺序。上游脚本是单机多 GPU 教学路径，不能直接等同已验证的多机部署。

18.9 相同 seed 不保证跨设备相同随机数

上游 CPU/MPS notebook 先创建概率，再分别在 CPU、MPS、CUDA 调用 `multinomial`，每次设置相同 seed；第二个函数把概率搬回 CPU 抽样，再把结果搬到原设备。它隔离的是随机采样位置，不是在证明所有设备上的模型 logits 一样。[随机采样 notebook](#)

不同设备可能使用不同随机数生成器与实现，相同整数种子没有跨算法逐位一致保证。本书只验证同一 CPU、同一 seed、同一调用顺序会重现六个随机数；在前面额外抽一个数，后续序列就改变。把采样统一放 CPU 有助于控制一个变量，但引入设备传输，也不能消除前向数值差异。

18.10 浮点加法不满足实数里的结合律

float32 小例子中, $(10^8 - 10^8) + 1$ 得到 1, 而 $10^8 + (-10^8 + 1)$ 得到 0。因为第二种括号内的 1 在有限精度中被舍入掉。并行归约、融合内核、不同矩阵乘法路径都会改变运算次序, 即使没有任何随机数, 也可能出现小差异。

排查跨设备差异应分层: 先固定输入和参数, 在 eval/no-grad 下比较 logits; 再比较 softmax; 最后把同一概率交给统一采样器。还应记录 dtype、库版本、后端和误差容差。若直接比较两段长随机文本, 一次微小分叉会让后续全部不同, 几乎无法定位原因。

上游其他 notebook 对部分 MPS 版本采用 CPU 回退, 这属于固定代码的兼容策略。本书运行环境没有 MPS, 没有验证某个新版是否已经消除特定问题, 也没有把 CPU 检查写成跨设备测试。

18.11 练习与答案

补充题一: 提高 batch_size 后 tokens/s 增加, 是否说明同样的训练步骤数具有相同预算? 答案: 不是, 每步 token 数也增加了, 应同时记录总 token 和更新次数。

补充题二: DDP 已 all-reduce 梯度, 验证指标为何还需聚合? 答案: 梯度同步与用户计算的统计量是两条路径, 准确率、损失分子分母不会自动变成全局值。

补充题三: 把所有采样放 CPU 后, 跨设备生成一定一致吗? 答案: 不一定, 模型前向分布可能不同, 调用顺序和传输类型也可能不同。补充题四: pinned memory 是哪一侧的内存? 答案: 页锁定主机内存, 不是 GPU 显存。

执行 `python examples/core_training/optimizer_numerics.py qa/core_training/optimizer_numerics.json` 可复查 CPU 随机数、结合律、补齐词表和等量梯度平均反例。GPU、MPS、编译和通信性能明确未运行。

参考与覆盖

- [baseline](#)、单 GPU、DDP 三脚本及 [README](#): 复用数据/模型/损失见第 3—6、13 章, 新增计时、精度、编译、词表与同步路径在本章。
- [CPU/MPS 随机采样](#): 同种子设备分支、CPU 采样桥接与边界。

第 19 章 从 GPT 到 Llama: RoPE、RMSNorm 与门控前馈

“把 GPT 改成 Llama” 容易被误解成改几个配置数值。真正需要改变的是模型函数：怎样归一化、怎样表达位置、怎样组合前馈支路，以及如何把 checkpoint 的参数放到正确位置。形状对得上只是第一步，数值约定对得上才可能加载成功。

固定源码 [ch05/07_gpt_to_llama](#) 包括 GPT→Llama 2、Llama 2→Llama 3/3.1/3.2 和独立 Llama 3.2 三份 notebook。转换指的是逐步修改实现，不是把 GPT 权重无损变成 Llama 权重。本章不下载任何 pretrained checkpoint。

19.1 RMSNorm 保留均值，只调节整体幅度

LayerNorm 对长度为 D 的向量先减均值，再除以标准差。RMSNorm 使用均方根：

$$\text{RMSNorm}(x)_i = \gamma_i \frac{x_i}{\sqrt{D^{-1} \sum_{j=1}^D x_j^2 + \epsilon}}.$$

γ 是可学习缩放向量， ϵ 防止接近零时除法不稳定。它不减均值，也没有必须存在的平移参数。对于 [2,2,2]，LayerNorm 在无 affine 情况下输出全零；RMSNorm 输出接近 [1,1,1]。两者不可只按类名替换，然后沿用旧 checkpoint 期待相同输出。[RMSNorm 原论文](#) 给出这一归一化设计的动机与实验。

GPT→Llama 2 notebook 先定义 RMSNorm，与 `torch.nn.RMSNorm` 比较，再将其放入 block 的两个预归一化位置和模型末尾。独立 Llama 3.2 使用 PyTorch 模块；dtype、eps、参数初始化必须与来源一致。低精度时平方和常提升到 float32，不意味着所有模型都可把归一化任意搬动。第 21、22 章会看到零中心权重、头内归一化和投影整体归一化的不同约定。

19.2 SwiGLU 的两个分支分别做什么

令 $x : [B, T, D]$ ，中间宽度 F ，门控前馈为

$$u = xW_u, \quad g = \text{SiLU}(xW_g), \quad y = (u \odot g)W_d,$$

$W_u, W_g : [D, F]$ ， $W_d : [F, D]$ ， $\text{SiLU}(z) = z\sigma(z)$ 。一个分支生成候选特征，另一个连续调制它；SiLU 的输出可以为负，所以它不是概率门。源码 `fc1/fc2/fc3` 的次序要和 checkpoint 的 `gate_proj/up_proj/down_proj` 对应，不能因为前两个矩阵形状相同就随意交换。

不计 bias, 参数为 $3DF$ 。普通扩展到 $4D$ 的两矩阵 FFN 参数约 $8D^2$; 若想保持相近预算, 门控宽度可取约 $8D/3$, 实际模型还会作硬件友好的整除取整。三矩阵不必意味着参数一定更多, 关键是中间宽度。GLU Variants 是这些门控前馈变体的原始参考。

19.3 RoPE 把位置变成二维旋转

绝对位置 embedding 把一个位置向量直接加到 token embedding。RoPE 则旋转 Q 和 K。先取一对坐标 (a, b) , 位置 m 对应角度 $m\omega$:

$$R(m\omega) \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} a \cos(m\omega) - b \sin(m\omega) \\ a \sin(m\omega) + b \cos(m\omega) \end{bmatrix}.$$

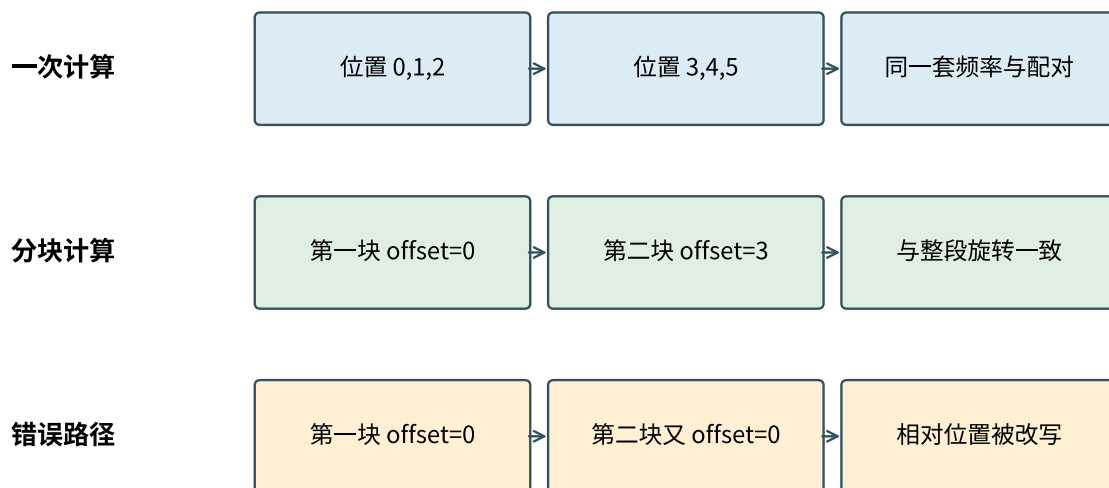
头宽 d 必须为偶数, 频率为 $\omega_i = \theta^{-2i/d}$, $i = 0, \dots, d/2 - 1$ 。每对坐标使用一个频率。源码采用前半/后半配对: 第 i 个坐标与第 $i + d/2$ 个坐标成对; `cat([-x2, x1])` 实现旋转的垂直方向, `cos/sin` 表在两个半区重复。另一些模型使用偶数/奇数配对, 公式相同, 但需要相应置换投影权重。

关键性质来自旋转矩阵:

$$(R(m)q)^\top (R(n)k) = q^\top R(n-m)k.$$

所以点积通过位置差进入。旋转保范数, 但这不等于注意力完全不含绝对位置信息, 因为输入表示、边界和可见历史仍有区别。RoFormer 给出了该位置编码的定义。通常不旋转 V, 因为输出被聚合的是内容表示。

缓存槽号不等于 token 的绝对位置



`precompute_rope_params/compute_rope_params` 产生 $[T_{max}, d]$ `cos/sin` 表, `compute_rope/apply_rope` 将它们广播到 $[B, H, T, d]$ 。模型去掉原来的 position embedding, 将因果 mask 和 RoPE 表在层间复用。buffer 随设备移动, 但不必作为可训练参数, 也不必重复保存在每个 checkpoint 中。

19.4 Llama 3 的 GQA 与长上下文频率

Llama 3 的注意力改用 GQA, query 仍有 H 个头, KV 只有 G 个头, 展开规则见第 09 章。Llama 2→3 notebook 的 `GroupedQueryAttention` 把 mask 与 RoPE 参数作为共同输入; `TransformerBlock` 顺序执行预归一化注意力与预归一化门控 FFN; `Llama3Model` 管理 token embedding、共享位置表、因果 mask、最后归一化与词表输出。

调整 RoPE base 影响各维频率, 不能简单理解为“把最大长度改大”。Llama 3.1/3.2 的 `freq_config` 进一步分段缩放。设原上下文长度为 C_0 , 波长 $\lambda_i = 2\pi/\omega_i$, 低/高频阈值由 `low_freq_factor/` `high_freq_factor` 给出: 长波长的低频降频, 短波长高频保留, 中间频率平滑插值。源码先构造缩放频率, 再用 `is_medium_freq` 对中间带替换。这个频率修改改变所有后续位置角度, 不能先缓存旧 key 再中途换规则。

缓存扩展时, 新 token 的 RoPE 位置应从已处理长度开始。独立 Llama 3.2 这份 notebook 本身是无 KV-cache 实现, 不能直接只输入最后一个 token 来“加速”。本书原创 `rope_cache_offset` 测试演示正确 offset 与错误从零重启的区别; 缓存版具体接口在第 20 章对照。

19.5 分词器、对话边界与权重布局是一整套协议

Llama 2 notebook 的 `LlamaTokenizer` 包装 `SentencePiece`; Llama 3 的 `Tokenizer` 用 `tiktoken` 加载指定 mergeable ranks, 并加入 BOS/EOS、header、回合结束等特殊 ID。encode 的 BOS/EOS 开关和 `ChatFormat` 的角色 header 决定输入协议。不要把“都是 BPE”当成可共用词表; 同一个 ID 在不同词表里可能代表不同字节。`clean_text` 只清理显示输出, 不能代替正确的停止 token 处理。

`assign` 在 no-grad 下核对源/目标 shape 并复制, 避免静默广播; Llama 2 的 `permute` 处理 RoPE 配对布局与 checkpoint 的投影约定。这一 Llama 2 加载器还有一个配置边界: 函数内部把 `cfg` 固定为全局 `LLAMA2_CONFIG_7B`, Q/K 的 `permute` 使用其中的头数和宽度, 而层循环读取传入的 `param_config`。因此传入小模型或其他尺寸配置, 并不会同步改变所有重排尺寸; 移植时应统一从实际配置取值, 并加入非默认尺寸的布局检查。

转换时除 Q/K/V/O 外, 还须映射三份 FFN 权重、两个 block norm、final norm、embedding 与 output head。错误交换同形状矩阵不会触发 shape 错误, 最好加入一层输出的来源对照; 本书没有下载权重, 因此没有做 checkpoint 级一致性声明。

Llama 3.2 的词嵌入和输出头可以共享参数。`count_unique_parameters` 按存储身份去重, 区别于把两个名称下的元素数简单相加。共享要求 `out_head.weight` 与 `tok_emb.weight` 指向同一个 `Parameter`, 复制数值不等于绑定。`calc_model_memory_size` 统计参数、梯度、buffer 的字节, 只是特定假设下的预算, 没有包括每种优化器状态和完整训练峰值。

目录的 `config.json` 是认证配置模板, 不是模型权重配置; 本书不要求把令牌写进文档或源码包。登录、许可确认、Hub 下载以及 `safetensors` 合并单元均只做静态讲解。源码公开许可不等于模型权重许可, 访问权重仍需遵守发布方条款。

19.6 主干外的辅助步骤与源码差异

`previous_chapters.py` 提供 GPTDataset、滑窗 dataloader、GPT 基线、采样、交叉熵、验证、训练与绘图等复用函数。它们使 notebook 可以逐步比较旧主干和新主干；数据集将输入与目标错开一个 token，loss 展平为 $[BT, V]$ 与 $[BT]$ ，训练循环清梯度、前向、反向、更新，评估关闭梯度与 dropout。这里改变模型后仍保留相同 logits 协议，详细训练方法见第 13—18 章。

三份 notebook 有意重复类定义，独立版删去转换演示与旧配置，保留一套完整 Llama 3.2 前向、分词、权重加载和生成。`import_from_notebook` 从前一 notebook 读定义用于递进演示，本书不运行这个入口，因为全本执行可能触发下载和大模型分配；测试只白名单执行已审查的架构定义单元。

19.7 真实检查、练习与答案

`examples/architectures/architecture_checks.py` 的 RoPE 测试证明整段/分块位置一致、范数保持和相对位移恒等式；误差不超过 1.78×10^{-15} 。`examples/architectures/source_smoke.py` 另外执行独立 notebook 单元 6—10 的真实类定义，在词表 31、宽度 16、两层、上下文 16 的缩小配置上产生 $[1, 6, 31]$ 有限 logits；修改第 4 个位置以后的 token，前四个位置输出不变。完整日志见 `qa/architectures/source_smoke.json`。此处是随机参数、CPU 结构检查，不是 Llama 语言能力测试。

补充题。 ① $[2, 2, 2]$ 能区分 RMSNorm 和 LayerNorm 吗？② 前半/后半与偶奇配对为什么不能只改旋转函数？③ 扩词表后怎样保持 weight tying？④ 把 RoPE 表长度扩大十倍是否证明长文效果可靠？

答案。 ① 能，前者保留非零均值。② Q/K checkpoint 的坐标布局必须同步置换。③ 重新绑定同一个 Parameter，并检查优化器引用，见第 23 章。④ 只能避免索引越界；质量还依赖训练、频率方案与验证。

19.8 上游测试怎样把错误定位到零件

`tests/test_llama32_nb.py` 用 `import_notebook_defs` 只导入定义，`dummy_cfg_base/dummy_input` 建立小配置，再做形状检查；可选 Transformers 对照先建立随机参考模型，再把它的 `state_dict` 交给本实现加载器，比较 logits。两边必须使用同一份权重；各自随机初始化后比较没有意义。

`tests_rope_and_parts.py` 另保留 LitGPT 的 `litgpt_build_rope_cache/litgpt_apply_rope` 参考函数及其 Apache 来源声明，比较 Llama 2、3、3.1/3.2 的旋转表与应用结果。局部 `RoPEConfig.standardize_rope_params` 适配 Transformers 不同版本的配置接口；`set_seed` 固定随机性；`test_silu/test_rmsnorm` 对比 PyTorch 基础算子。测试中的 notebook 导入 helper 只提取所需定义，不能据此许可执行后面的登录/下载单元。当前环境没有 Transformers 或 pytest，本书未运行这些上游参考对照，仅运行前述自包含代数与真实类 smoke 检查。

参考资料

- 固定源码 `ch05/07_gpt_to_llama`, 所有 notebook 单元及 Python 定义见覆盖台账。
- RMSNorm: <https://arxiv.org/abs/1910.07467>; GLU variants: <https://arxiv.org/abs/2002.05202>。
- RoFormer: <https://arxiv.org/abs/2104.09864>。
- Llama 2: <https://arxiv.org/abs/2307.09288>; Llama 3: <https://arxiv.org/abs/2407.21783>。
- 未运行: 认证/权重下载、全尺寸分配、预训练、checkpoint 输出对齐和长上下文质量评测。

第 20 章 Qwen3 与 Qwen3.5: 同一个名字下, 两种记忆接口

Qwen3 的密集版和 MoE 版主要在前馈模块上分岔; Qwen3.5 的教学实现又引入全局注意力与 Gated DeltaNet 的混合。因此“换 checkpoint 名称”不足以完成模型替换。投影宽度、归一化参数含义、位置旋转范围、缓存种类都必须一起核对。

来源为固定提交 [ch05/11_qwen3](#) 的四份 notebook, 以及 [ch05/16_qwen3.5](#) 的两份 notebook 与 helper。这里限定为源码中的文本模型, 不涵盖整个系列的多模态组件。

20.1 Qwen3: Q/K 归一化发生在每个头内部

Qwen3 的主干仍是 token embedding、重复的预归一化残差块、final RMSNorm、输出头。FFN 是 SiLU 门控前馈。与 Llama 3.2 的实现相比, 一个重要差别是 attention 的 `qk_norm`: Q/K 拆成 $[B, H, T, d]$ 后, 沿最后的头宽 d 做 RMSNorm, 再施加 RoPE, 之后展开 GQA 的 KV 头。

这不会把 Q/K 变成概率, 也不是 L2 范数恒为 1: RMSNorm 除以均方根, 若无缩放参数, 其向量二范数约为 $\sqrt{d}$ 。最终点积仍除以 $\sqrt{d}$ 。源码 RMSNorm 默认 `qwen3_compatible=True` 时先转 float32 计算平方均值, 再转回输入 dtype; scale 初始化为 1。归一化轴若错放到合并后的全部头, 模型函数会改变, 第 22 章 OLMo 正好展示了另一种合法但不同的选择。

`head_dim` 可以显式指定, 不必等于 `emb_dim // n_heads`。例如源码 0.6B 配置的 embedding 宽度为 1024, 16 个 query 头、每头 128, 注意力内部总宽为 2048, 最后 `out_proj` 再映射回 1024。实现必须用 Hd reshape, 不能沿用 GPT 的“所有投影都是 D 宽”假设。

这些具体实现点可在 `GroupedQueryAttention.__init__/forward` 逐项核对; 更广泛的训练与评估背景见 [Qwen3 技术报告](#), 本书不复述其大规模指标。

20.2 Qwen3 MoE: 注意力不变, 前馈变为按 token 分派

MoE notebook 添加 `MoEFeedForward`, 用 `num_experts` 个门控专家和 `num_experts_per_tok` 的 top- k 路由代替密集 FFN。专家中间宽度独立叫 `moe_hidden_dim`, 不能直接沿用密集模型的 `hidden_dim`。源码仍在选中分数上 softmax, 收集输入、取各 token 的路由概率、`index_add` 合并, 完整过程见第 12 章。

加载器逐层映射 router 的 gate, 以及每个 expert 的 gate/up/down 三个投影。所有专家权重都要存在, 虽然一个 token 只激活其中几个。`calc_model_memory_size` 的全参数结果不会因为 top- k 小就自动减少。Qwen3Model 的外部输入输出仍为 $[B, T] \rightarrow [B, T, V]$, 训练 loss 可保持, 但性能与优化器内存不能照抄密集版结论。

目录四份 notebook 形成两条独立对照轴: 密集/MoE、无 cache/有 cache。缓存版只改变 attention 与模型的状态协议, 不会把 MoE 改成另一种路由算法。覆盖台账分别保留四份单元位置, 避免把重复实现当成缺失。

20.3 Qwen3 缓存: 位置计数和 KV 容器缺一不可

缓存版 KVCache 是逐层列表, `get/update/get_all/reset` 管理每层的 K/V 元组。attention 将新 K 先按 `start_pos` 旋转, 再与旧 K 拼接; 旧 K 已经带位置, 不能再旋转。模型保存 `current_pos`, 构造从 `pos_start` 开始的矩形 causal mask, 并把更新后的 cache 交还容器。

`model.reset_kv_cache()` 在这些 notebook 中主要重置模型位置; `cache.reset()` 才清空外部容器。新请求必须两者一致。生成器先用 prompt 预填充, 随后只传新增 token; 有上下文裁剪参数的版本到边界时清缓存重建。EOS 判断应使用当前 tokenizer 的 ID, 不要把另一个模型的终止编号带进来。

`generate_text_basic_stream` 逐个 yield token ID, 流式显示不改变因果模型的数学。tokenizer 从发布文件加载, Base/Reasoning/Instruct 的选择影响 checkpoint 与模板。推理格式的开关不意味着随机架构自己学会推理; 这来自具体训练过的参数与输入协议。

20.4 Qwen3.5 的全局层: 局部旋转与额外输出门

Qwen3.5 notebook 的 RMSNorm 参数初始化为 0, 实际缩放是 $1 + w$ 。把 Qwen3 的 scale 直接放进去会错误地再加 1, 虽然 shape 完全相同。加载器必须忠实于 checkpoint 约定。

全局 attention 的 w_{query} 输出 $2Hd$, 先 reshape 到每头 $2*d$, 再一分为二得到 query 和 gate。gate 的形状还原为 $[B, T, Hd]$, 在 softmax attention 的上下文输出上乘 sigmoid:

$$z = \text{Attn}(Q, K, V) \odot \sigma(g), \quad y = zW_o.$$

拆分必须在每头内部进行。若先把整个 $2Hd$ 向量分成两个 Hd 半区, 某些 head 的 query 和 gate 会互相串位。Q/K 仍先作头内 RMSNorm, 再 RoPE, softmax 在 float32 中计算。

`partial_rotary_factor` 让只有前 d_r 个坐标参与旋转, 其余坐标直接通过; d_r 按偶数调整。源码以 d_r 而不是整个 d 作为频率指数分母。0.8B 配置的 `head_dim=256`、比例 0.25, 旋转宽度为 64。这里和 Gemma 4 的 proportional RoPE 不同, 后者的频率分母仍是完整 `head_dim`, 见第 21 章。

20.5 线性层: 卷积记忆加矩阵记忆

TransformerBlock 按 `layer_types` 选择 `GroupedQueryAttention` 或 `Qwen3_5GatedDeltaNet`。

`_Qwen3_5ConfigAdapter` 仅把教学字典里的名称映射到 helper 期望的字段: 隐藏宽度、key/value 头数、各自头宽、卷积核宽、激活、eps 和 dtype。它没有调用 Transformers 模型, 也没有下载内容。

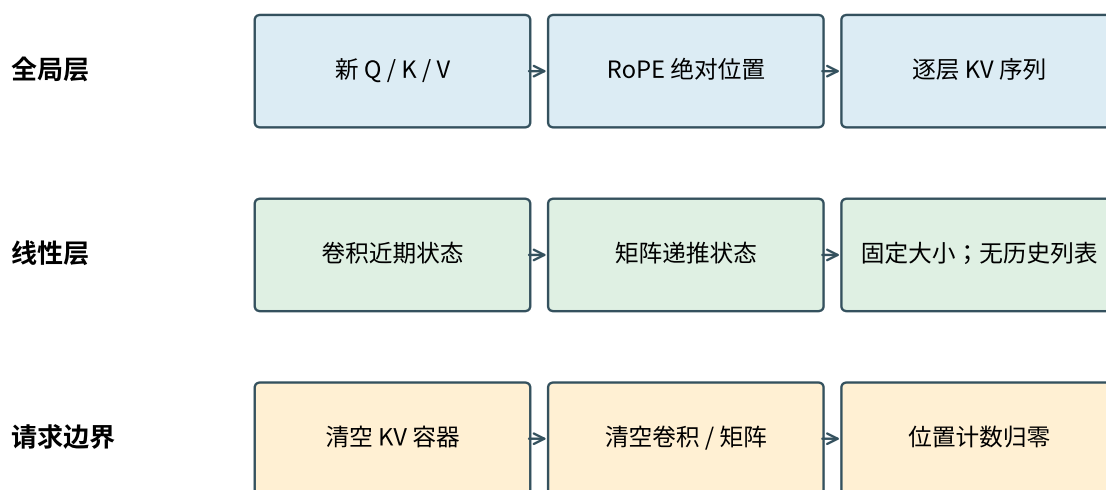
helper `qwen3_5_transformers.py` 标明来自 Hugging Face Transformers 的 Apache 代码, 并将可选快速 kernel 置为 None, 回退到 PyTorch。它先将输入投影为 Q/K/V 合并通道, 再做逐通道因果一维卷积和 SiLU; 卷积宽度 K_c 需要保留近期投影值。随后拆出 $[B, T, H_k, d_k]$ 的 Q/K 与 $[B, T, H_v, d_v]$ 的 V, 若 value 头更多, 则重复 Q/K 到同样头数。

更新门为 $\beta = \sigma(b)$, 遗忘参数先以 log 域表达:

$$g = -\exp(A_{\log}) \text{softplus}(a + \text{dt_bias}), \quad \alpha = \exp(g).$$

由于 $g \leq 0$, $\alpha \in (0, 1]$ 。指数和 softplus 使用 float32, 降低低精度溢出的风险。递推状态为 $[B, H_v, d_k, d_v]$, 不同于增长的 KV 序列。最后 `Qwen3_5RMSNormGated` 对读出归一化并乘 SiLU 门, 再输出投影。

Qwen3.5 混合层：请求重置必须覆盖两套状态



20.6 chunk 与 recurrent helper 如何相互校验

`torch_recurrent_gated_delta_rule` 与第 11 章逐 token 公式对应, 输入先转 float32, Q/K 可作 L2 norm, Q 再乘 $1/\sqrt{d_k}$ 。它返回所有输出和可选最终状态。

`torch_chunk_gated_delta_rule` 为分块版本: 把长度补齐到 `chunk_size` 的倍数, 构造块内下三角的衰减关系, 解出误差更新对前序 token 的依赖, 再将旧状态贡献与块内贡献合并, 最后更新块末状态。

`g.cumsum` 使连续遗忘乘积转为累加后求指数。padding 的 Q/K/V/beta 为零、g 为零, 对应无写入且衰减为 1, 最后再切回原长度。不能把补齐 token 当作真实 token 计入 loss。

本书实际导入了已检查无下载副作用的 helper, 对长 7、chunk_size=4、两个头、key 维 3/value 维 4 的张量比较两个入口。输出和最终状态最大误差均为 1.1921×10^{-7} , 与 helper 内部 float32 计算相符。完整命令为 `python examples/architectures/source_smoke.py`, 准确日志在 `qa/architectures/source_smoke.json`。

`torch_causal_conv1d_update` 则拼旧卷积状态和新输入、更新缓存尾部、按通道卷积并只取新序列的输出。`apply_mask_to_padding_states` 对支持的多 batch 场景遮掉 padding 输入; 它有具体条件分支, 不能据此宣称支持任意 padding 布局。logger 只去重打印快速 kernel 不可用提示, 两个占位配置类型仅服务注解, 不实现完整 Transformers API。

20.7 混合缓存的一个重要限制

`Qwen3_5LinearAttentionCache` 保存每层 `conv_states`、`recurrent_states` 和 `has_previous_state`; 外部 `KVCache` 同时持有普通 KV 列表。模型级 `current_pos` 管理全局层的位置。清理请求时三者都必须复位。

helper 的增量分支要求已有状态且 `seq_len == 1` 且提供 `cache_position`。否则它走 chunk 分支, 并以 `initial_state=None` 开始。由此可知, 这份教学实现的安全路径是“整段 prompt 预填充, 然后逐 token 解码”; 不能未验证就宣称支持任意多 token 的追加 chunk, 因为那条路径可能重置递推记忆。把第 10 章 SWA 的 chunk 测试原样推广到这里会漏掉这个区别。本书随后用两层混合小模型实际验证: 整段预填充后逐 token 追加, 与完整前向最大误差为 2.38×10^{-7} ; 改为先 3 个 token、再追加 2 个、最后 1 个, 最大误差达到 **0.21603870**。后者是已重现的失败反例, 日志为 `qa/architectures/cache_checks.json`, 不能计为支持任意 chunk 的通过结果。

20.8 加载、真实结果与练习

两组 notebook 的加载器均显式映射 embeddings、投影、norm、FFN 和 output head; Qwen3.5 还映射卷积、`A_log/dt_bias`、四份输入投影和 gated norm。Hub 下载、safetensors 合并、tokenizer 文件、chat template、停止 ID、流式生成构成后半条调用链。本书没有执行这些外部路径, 也没有获取权重。

真实缩小源码检查中, Qwen3 dense/MoE 都产生 [1, 6, 31] 有限 logits。改变未来 token 时 dense 过去输出差为 0, MoE 最大差为 1.34×10^{-7} : 分派后矩阵批次形状变化可能带来浮点舍入, 须用容差检查而不是逐位断言。没有据此声称原规模模型已经复现。

补充题与答案。 ① Qwen3 的头宽为什么不能总由 D/H 推出? 因为配置允许注意力内部宽度与残差宽度不同。② Qwen3.5 的全局门和 delta 的 beta 是同一个量吗? 前者调节读出, 后者调节写入。③ 只清普通 KV 列表能隔离请求吗? 不能, 卷积和矩阵状态也会泄漏历史。④ 为什么 chunk/recurrent 测试不用 10^{-15} 容差? helper 显式转 float32, 不能期待 float64 精度。

20.9 上下文拒绝、参考调试器与聊天界面

tests/test_qwen3_kvcache_nb.py 的两个边界测试特别重要: 长度超过 context 时应抛出清楚的 ValueError; 失败的追加调用不能先把 current_pos 或 KV cache 改坏。源码先算 pos_end、校验、再更新状态。context_model_setup 对密集与 MoE notebook 参数化, dummy_cfg_* 提供小模型; 普通形状与可选 Transformers 同权重对照是另外两种测试。不要把文件名含 kvcache 当成每条测试都验证了增量等价。

本书 cache_checks.py 实际以 [0:2], [2:5], [5:6] 分块, 得到 Qwen3/MoE 最大误差分别 2.98×10^{-7} 、 2.38×10^{-7} ; 超长调用被拒绝后, 位置仍为 5, 缓存长度仍为 5。完整日志在 qa/architectures/cache_checks.json。

Qwen3.5 的 tests/test_qwen3_5_nb.py 通过 _import_qwen3_5_classes 检查参考库支持, 避免拿旧 Transformers 的不存在类作对照。qwen3_5_layer_debugger.py 的 tiny_debug_config 与 _hf_config_from_dict 对齐两边配置, build_qwen3_5_pair 用相同权重创建成对模型; _attach_debug_hooks 记录 embedding、层及相关模块输出, hook 内 _record 对 tuple 取张量、detach、转float32和CPU。layerwise_differences 对齐名字并计算差异, _layer_sort_key 保持数值层序, format_report 格式化报告。可选 hf_checkpoint 会触发预训练路径, 本书不运行; 无参考库也没有伪称这些对照通过。

Qwen3 目录还包含两个 Chainlit 界面脚本。单轮版的 get_qwen_config/build_repo_and_local/get_device/get_model_and_tokenizer 负责选配置、选Hub目录、选设备和载入模型; 异步 main 编码用户输入、裁上下文、流式输出。多轮版添加 on_start 的会话历史、build_prompt_from_history 的角色边界、assistant header, 并关闭 tokenizer 自带模板以免重复包裹; 回答完将 assistant 内容写回会话。结束判定兼顾模型可能输出的两种终止ID。历史文本被裁剪为token窗口, 并不保证保留完整回合边界, 生产使用应另定裁剪策略。

这两个界面在模块顶层调用模型下载/初始化, 不能为了“只看一个函数”直接 import。这里仅静态审阅, 没有启动服务、下载权重或发送任何外部消息。public/logo_*.webp 是界面资产, 不复用到本书; requirements 文件表达界面依赖, 不代表当前环境已安装。

参考资料

- 固定源码 ch05/11_qwen3、ch05/16_qwen3.5, 包括 helper 原始版权说明。
- Qwen3 技术报告: <https://arxiv.org/abs/2505.09388>。
- Qwen3.5 官方模型资料: <https://huggingface.co/Qwen/Qwen3.5-0.8B>。
- Gated Delta Networks: <https://arxiv.org/abs/2412.06464>。
- 未验证: 完整 checkpoint 输出、量化、快速 kernel、原规模训练、任意 chunk 增量解码。

第 21 章 Gemma 3、Gemma 4：局部层、共享层与数值约定

Gemma 的教学实现很适合练习“同样叫 RMSNorm、RoPE、GQA，为什么不能交换零件”。

Gemma 3 的零中心 norm 参数、局部/全局双套位置频率，以及 Gemma 4 的跨层 KV 共享和逐层输入，都影响真实计算路径。本章按数据流解释，不把模型名当成配置标签。

固定来源是 [Gemma 3 两份 notebook](#) 与 [Gemma 4 E2B/E4B 两份 notebook](#)。范围是文本前向与生成，不包括视觉、音频模块或全系列配置。

21.1 Gemma 3：残差分支前后都要归一化

Gemma 3 的 FeedForward 用近似 GELU 门控，而非 Llama/Qwen 的 SiLU：

$[\text{GELU}(xW_g) \odot xW_u]W_d$ 。三个矩阵形状相同不能作为激活可互换的理由，checkpoint 是在特定函数下训练的。

RMSNorm 的 scale 初始化为 0，实际乘数是 $1 + \text{scale}$ ，计算均方时升到 float32。一个 block 含四个归一化模块：输入归一化后进入 attention，attention 输出再归一化后加残差；第二分支先归一化再 FFN，FFN 输出再归一化后加残差。写成式子：

$$u = x + N_{a,out}(A(N_{a,in}(x))), \quad y = u + N_{f,out}(F(N_{f,in}(u))).$$

四个 N 各有参数，不能删掉“看起来重复”的两个。主干入口的 embedding 还乘 $\sqrt{D}$ ，末尾有 final norm；这些尺度必须与权重一起保留。[Gemma 3 技术报告](#) 是架构背景来源，具体执行次序以上述固定 notebook 为准。

21.2 局部/全局层使用不同的位置表和 mask

模型建立 mask_global 挡未来、mask_local 再挡超过窗口的旧位置。TransformerBlock.attn_type 同时选择对应 mask 与对应 cos/sin，不能只改窗口不改位置表。局部和全局层分别使用 rope_local_base 与 rope_base，源码在模型初始化中预计算两套表。

Q/K 拆头后作可选头内 RMSNorm，再 RoPE，KV 按组展开。缩放不总是 $d^{-1/2}$ ：存在 query_pre_attn_scalar 时用它的负平方根。源码先缩放 query 再点乘，数学上等价于之后缩放分数，有限精度下舍入路径不同。

Gemma 3 的配置列出具体的 layer_types。不要因为技术报告描述某种常见局部/全局比率，就假设每个小型号都具有完全相同比率。实际层数、头宽、KV 头数和窗口由当前配置决定。缓存版本增加 offset 和外部 KVCache，但局部 attention mask 不自动意味着持久 cache 已裁剪；必须看 update 前有没有切

片。此处 block 在 attention 完成后，确实把局部层 cache 裁到窗口宽。attention 缓存的是已归一化但未旋转的 key：拼接以后以 `start_pos - prev_len` 为 key 起点重新旋转，query 则从 `start_pos` 旋转。不能套用 Qwen 的“缓存已旋转 key”约定。

21.3 Gemma 4：局部头和全局头可以不同宽

`get_gemma4_dense_config` 为 E2B 与 E4B 返回两套配置。固定源码 E2B 的残差宽度为 1536、35 层、8 个 query 头、1 个 KV 头；E4B 为 2560、42 层、8 个 query 头、2 个 KV 头。两者局部 `head_dim=256`、全局 `head_dim=512`，窗口 512。E2B 的层型为四局部加一全局的重复，E4B 为五局部加一全局的重复。第 09 章估算器里的“E2B-like”简化宽度不能替代这里的真实配置。

`Gemma4Attention` 按层类型选头宽，query 总宽为 Hd ，最后投回 D 。Q/K 有带参数 RMSNorm，V 也归一化但不带缩放参数。`Gemma4RMSNorm` 直接用初始化为 1 的 weight，不是 Gemma 3 的 $1 + w$ 参数化。

另一个容易“顺手修错”的点：固定实现直接对 QK^T 作 softmax，没有再除以 $\sqrt{d}$ 。忠实复现时不能因为多数模型使用该缩放，就擅自加进去。softmax 提升到 float32，之后转回 query dtype；mask 使用 dtype 的有限最小值，有效 query 仍必须至少有一个合法 key，否则会把逻辑错误隐藏成均匀输出。

21.4 proportional RoPE 保留不旋转的子空间

全局层的 `compute_rope_params` 在 proportional 分支中，先按 `partial_rotary_factor` 计算旋转对数，生成这部分频率，再给剩余对填入零频率。零频率对应 $\cos=1$ 、 $\sin=0$ ，坐标保持原样。这里指数分母使用完整 `head_dim`；Qwen3.5 partial RoPE 则按旋转宽度生成频率，两者不能只靠同一个比例统一。

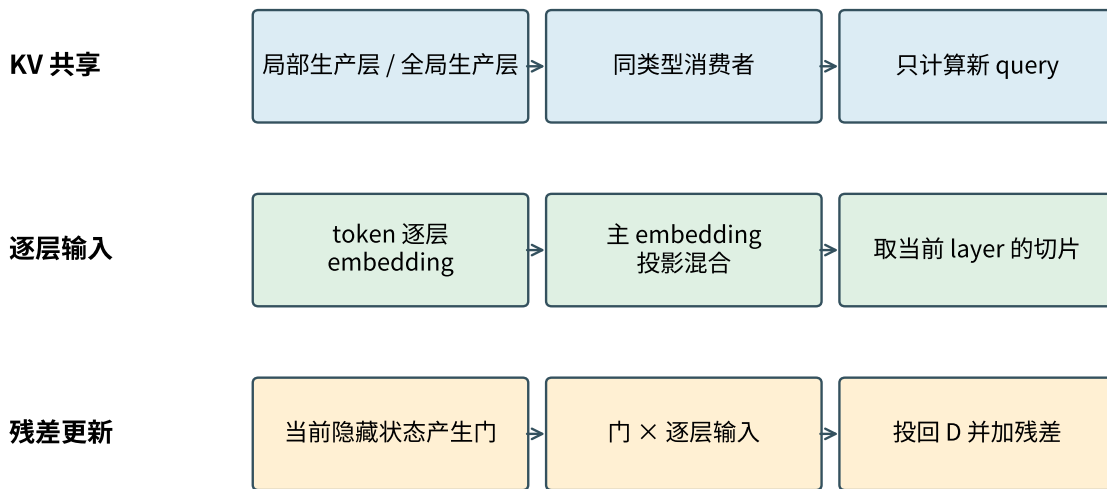
对全局宽度 512、比例 0.25，旋转 64 对坐标、128 个维度，其余 384 维保持不变。因为采用前半/后半配对，旋转的坐标不是简单连续的前 128 维；必须结合配对布局理解。局部层使用自己的普通 RoPE 和较小头宽。

21.5 KV 共享必须寻找同一种层的生产者

E2B 后 20 层共享，前 15 层生产；E4B 后 18 层共享，前 24 层生产。`first_kv_shared_layer_idx = n_layers - num_kv_shared_layers` 确定边界。共享层在边界之前的层型列表中，反向寻找最后一个与自己同类型的层。这确保局部 query 读局部 KV，全局 query 读全局 KV；它们头宽不同，不能随意交叉读取。

无 cache notebook 提供 `reuse_shared_kv` 开关，默认 False，此时各层自行生成 KV；打开后用临时字典保存生产者 KV 并在同一次完整前向中共享。有 cache notebook 则从外部容器找到生产者 cache。比较两份实现时应把共享语义对齐，不能拿“所有层自算”的完整前向当作共享缓存的必然等价基准。

Gemma 4 的两条额外数据通路



共享消费者只计算 query，从生产者取得已归一化、已旋转的 key/value，不能再追加或再次旋转。源码仍定义消费者 K/V 投影模块，这些参数是否参与前向取决于共享开关；它们的存在不代表消费者有独立 cache。缓存版 block 实际会裁剪局部生产者的 KV；这节省持久存储，但跨层消费者读取发生在裁剪之后，产生下文实测的预填充边界问题。

21.6 逐层输入：每一层都有自己的 token 侧信息

Gemma 4 E 系列的 `hidden_size_per_layer_input=P` 不为零。模型另外查一张 embedding 表，输出 $[B, T, LP]$ ，乘 $\sqrt{P}$ ，再 reshape 成 $[B, T, L, P]$ 。它为同一个 token 在每层提供不同的 P 维向量。

还有一个从主 embedding 投影到 LP 的分支，乘 $D^{-1/2}$ 、按最后 P 维归一化，再与逐层 embedding 相加除以 $\sqrt{2}$ 。得到每层的侧输入 $e_l : [B, T, P]$ 。进入 block 时从 layer 轴取第 l 片，不可把 token 轴误当 layer 轴。

block 完成 attention 与 FFN 后，用当前状态形成门：

$$r_l = N_p([\text{GELU}(xW_{g,l}) \odot e_l]W_{p,l}), \quad x \leftarrow x + r_l.$$

$W_g : [D, P]$ 、 $W_p : [P, D]$ 。最后乘该层 `layer_scalar` buffer。E2B 还在共享 KV 的后段层启用双宽 MLP，E4B 此配置关闭。节省某一部分计算/存储后，其他部分可能重新分配容量，不能用单一层宽估算全部层。

输出头与主 token embedding 绑定，logit softcap 为 $c \tanh(z/c)$ ，固定配置 $c = 30$ 。小 logit 近似不变，大幅值被平滑限制；这改变输出分布，并不是简单截断到区间端点。

21.7 加载器和后半段 notebook 的职责

Gemma 3 加载器映射四份 block norm、Q/K norm、GELU 三投影、embedding/final norm/output。Gemma 4 加载器还处理不同 checkpoint 前缀、逐层 embedding 与投影、layer scalar、V norm 的无参数约定，以及共享层可能缺失的独立权重。get_tensor 尝试认可的键名前缀，assign 核对 shape 并转到目标 dtype/device。找不到权重不能悄悄视作随机初始化的成功加载，必须检查实际映射分支与报告。

resolve_local_model_dir 只是按多个候选位置寻找已存在文件；后面的 Hub 下载与 safetensors 分片合并是另外的外部动作。tokenizer 包装 tokenizer.json，chat template/终止 token 是生成协议；目录 README 的示例输出不是本书新生成结果。模型许可与代码许可分开处理。本书不复制原图，也不执行认证、下载或生成质量比较。

21.8 实测与补充题

source_smoke.py 白名单执行 Gemma 3 单元 8—13，以及 Gemma 4 单元 9—13、15。在小词表、窄隐藏层上得到有限 [1, 6, 31] 输出，修改未来 token 的过去输出差均为 0；Gemma 4 另打开共享分支检查有限性。这是结构与因果性检查，尚非官方 checkpoint 一致性。日志见 qa/architectures/source_smoke.json，PyTorch 2.6.0+cpu、seed 1949、单线程。

题目与答案。 ① Gemma 3 的 norm 参数全零是否让输出全零？不是，实际乘数为 $1 + w$ 。② Gemma 4 全局层能直接借最近局部层的 KV 吗？不能，层型、头宽和位置约定不同。③ proportional RoPE 的零频率坐标发生什么？保持原值。④ 共享 KV 是否删除了逐层 query？没有，每层仍以自己的隐藏状态提出查询。⑤ 局部 mask 是否保证缓存上界为窗口宽？不保证，只有实际截断或环形存储才保证。

21.9 一个实际失败案例：共享消费者读到了裁剪后的 prompt

本书对 Gemma 4 作了额外的同权重比较：词表31、残差宽16、4层、局部/全局交替、后2层共享、窗口3，输入6个token。基准是无cache notebook 打开 reuse_shared_kv=True 的完整前向；对照是缓存 notebook 一次预填充全部6个token。最大logit差为 3.50999856，不是舍入误差，也不是通过结果。测试程序与日志为 examples/architectures/cache_checks.py 和 qa/architectures/cache_checks.json。

原因是局部生产层先完成自己的 attention，再把 next_cache 裁成最后3个位置；后面的局部消费者从外部容器读取这份裁剪结果。对于 prompt 前半部分的 query，本应可见的早期key已经不在。有限最小值mask还可能让全屏蔽行变成均匀分布，掩盖NaN警报。最后几个query或单token解码看似正常，不代表整个预填充等价。

修复方向是区分“当前前向供所有消费者读取的KV”和“留给下一步的持久裁剪缓存”：在消费者完成以前保留当前块所需的完整视图，之后才裁剪持久状态，并用绝对位置对齐mask。这是建议的修复设计，本书没有修改固定上游，也没有声称完成该修复。原规模checkpoint是否触发同样幅度的差异，未测试。

Gemma 3 的独立分块检查通过，最大误差 4.77×10^{-7} ，两层缓存长度为 [3,6]，分别对应局部/全局；这也实际核对了局部裁剪确实存在。

21.10 上游参考对照与逐层调试的范围

Gemma 3 的两个 `test*_nb.py` 分别加载普通/缓存 notebook，检查小模型shape，并在参考库可用时将同一随机权重映射到两边比较输出。`gemma3_layer_debugger.py` 先构造模型对，hook embedding、block、final norm等节点；详细版增加 `_register_trace_hook` 和 block 内部投影/norm记录，`_format_diff_line/format_report` 可以只展开首个偏离block或全部block。embedding hook 需补乘真实前向的缩放，否则会把记录点不一致误报成模型错误。hook handle 在比较结束后移除，避免重复注册改变后续记录。

Gemma 4 的 `test_gemma4_nb.py` 还检查E2B/E4B命名配置、没有旧接口 `n_kv_groups` 时的前向、流式生成helper、带 `model.language_model` 前缀的加载，以及可选本地checkpoint生成。

`gemma4_transformers_module` 临时切换已存在的本地参考源码路径和模块表，结束时恢复；这些环境技巧并不意味着当前安装的Transformers支持Gemma4。本书没有执行其可选本地大checkpoint测试。

`tests/test_e2b` 与 `test_e4b` 中的 debugger 各自提供小配置、参考config转换、定义导入、模型对构建、hook收集、层名排序、差异报告。两者必须保留不同的共享层与MLP设置，不能只改文件名。当前环境未安装参考库，故这些上游逐层对照是静态解释边界，不是本书通过项。

参考资料

- 固定源码 `ch05/12_gemma3`、`ch05/17_gemma4`。
- Gemma 3 技术报告：<https://arxiv.org/abs/2503.19786>。
- Google 官方 E2B 模型资料：<https://huggingface.co/google/gemma-4-E2B>。
- Google 官方 E4B 模型资料：<https://huggingface.co/google/gemma-4-E4B>。
- 未验证：多模态分支、原尺寸模型、checkpoint 数值对齐、量化与长上下文质量。

第 22 章 OLMo 3、Tiny Aya 与可替换模型接口

训练代码通常只要求模型把 token ID 变成 logits。因此我们可以保留同一个训练循环，替换内部架构。但“接口兼容”不等于“权重兼容”，也不等于换一个类名就能保留位置、分词和缓存语义。本章用 OLMo 3 与 Tiny Aya 的具体差异说明这条边界，再把它接回第五章的训练流程。

固定来源：OLMo 3、Tiny Aya、用其他模型重走训练章节。每个模型都有完整前向与 KV-cache 版本；训练目录另有 Llama 3.2 和 Qwen3 notebook。

22.1 OLMo 3 把 norm 放在分支输出

OLMo 3 的 TransformerBlock 先直接对输入计算 attention，再将 attention 输出 RMSNorm 后加残差；FFN 也先计算后归一化再加残差：

$$u = x + N_a(A(x)), \quad y = u + N_f(F(u)).$$

它既不是 Llama 的两个 pre-norm，也不是 Gemma 的四 norm。注意整个残差和没有被此处的 norm 归一化，norm 作用在分支输出上。权重映射要落到 post_attention_layernorm 和 post_feedforward_layernorm，不能根据相同的“norm”字样随便改成输入 norm。

OLMo attention 的 Q/K norm 发生在拆头之前。投影 $Q : [B, T, Hd]$ 在最后的 Hd 轴归一化， $K : [B, T, Gd]$ 在 Gd 轴归一化；之后才 reshape 为多头。与 Qwen3 每头单独归一化相比，一个头的幅度会影响另一个头的缩放。以两个头 $[1, 1]$ 、 $[10, 10]$ 为例，头内 RMSNorm 会把两头都调成相近尺度，整体 RMSNorm 会保留两头之间约十倍的相对幅度。两种函数即使参数总 shape 可以重排，也不等价。

Q/K 随后 RoPE，KV 按组展开，query 在点乘前乘 $d^{-1/2}$ 。attention_bias 明确控制四份 attention 线性层是否带 bias。7B 配置的 KV 头数等于 query 头数，32B 配置则较少；同一个 GroupedQueryAttention 类可同时表示 MHA 和 GQA，不能只看类名判断压缩率。

22.2 YaRN 不是统一放慢所有频率

OLMo 的 compute_rope_params 支持 default 与 yarn。default 是第 19 章的常规 RoPE；yarn 分支先计算不缩放频率 ω 与按 factor 缩放的 ω/f ，再根据每对坐标在原上下文里旋转了多少圈，选择或平滑混合两者。

三个嵌套 helper 的作用清楚可分：find_correction_dim 从旋转圈数反解维度；find_correction_range 将边界向下/向上取整并裁到合法范围；linear_ramp_factor 在边界之间生成 0 到 1 的线性权重，并避免分母为零。若权重记为 r_i ，源码用

$$\omega'_i = r_i \omega_i + (1 - r_i) \omega_i / f.$$

这里 r_i 对应源码中的 extrapolation factor。高频与低频处理不同，不能用一个新的 theta 完全代替。原始研究见 YaRN。cos/sin 还一起乘 attention_factor=a，使旋转后的 Q/K 都多乘 a ，点积因此多乘 a^2 ；此时严格的“RoPE 保范数”应限定到未加这项尺度的旋转。

模型使用全局和局部 mask，配置 layer_types 确认哪些层用滑窗。缓存版新增 offset 和外部 KV 容器，但不能只凭 sliding_window 字段断言缓存已截断，仍需检查实际切片。load_weights_into_olmo 对应投影整体 Q/K norm 的宽度；把每头 norm 的 checkpoint 强行广播过去会改变模型。

22.3 Tiny Aya：无 bias 的 LayerNorm 仍然减均值

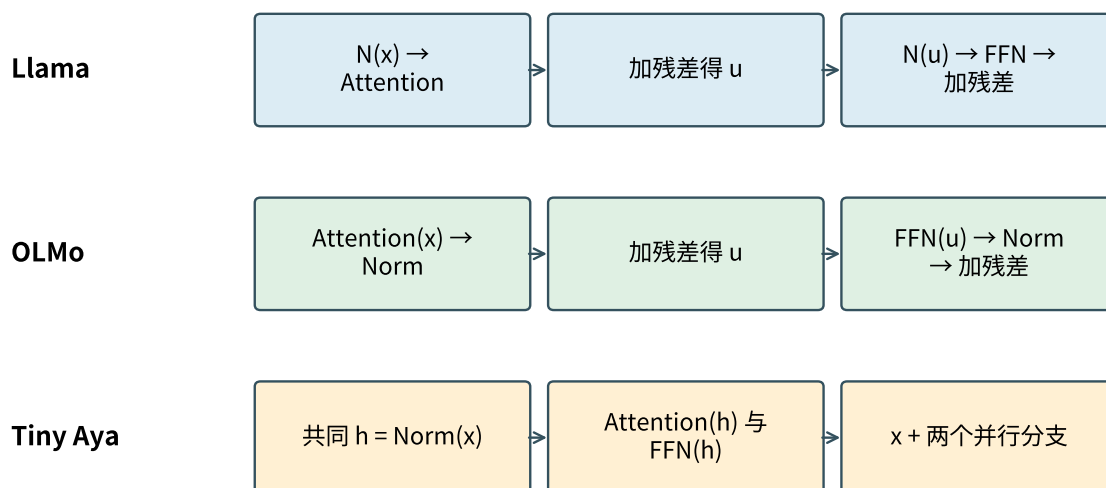
CohereLayerNorm 先升到 float32，计算均值和总体方差，减均值后乘倒标准差，再乘 weight。它省略的是可学习 shift，并没有省略减均值步骤。因此它与 RMSNorm 的区别不会因为两者都只存一个 weight 消失。

Tiny Aya block 只归一化一次，然后 attention 与 FFN 并行地读同一个归一化输入：

$$h = N(x), \quad y = x + A(h) + F(h).$$

与顺序残差 $x' = x + A(N(x))$, $y = x' + F(N(x'))$ 相比，前馈看不到本层 attention 更新后的表示。这是结构变化，不是计算顺序的无害优化。FeedForward 使用 SiLU 门控三矩阵，GroupedQueryAttention 支持可选 Q/K norm，但当前 block 构造时设为 False。

同样的模块名，数据依赖可以不同



22.4 Tiny Aya 的位置配对与全局层例外

Tiny Aya 的 RoPE 采用偶数/奇数配对。cos/sin 频率以 `repeat_interleave(...,2)` 展开，旋转向量由 `stack([-odd,even]).flatten(-2)` 构造。Llama 使用前半/后半配对，二者要经过同一个坐标置换才能对应；只复制 cos 表会让权重布局不匹配。

固定实现仅在 `sliding_attention` 层对 Q/K 施加 RoPE，全局层没有这个旋转。全局层仍有因果 mask，并非可以看未来；“没有显式 RoPE”与“没有因果限制”是两个问题。局部层的位置信息也会通过残差传播到后续层，不能据此断言整个网络对顺序一无所知。

模型 `create_masks` 同时生成局部与全局 mask。若提供 `attention_mask`，代码将其从“1 表示有效 token”转成“True 表示屏蔽 key”，再与 causal/window mask 合并。左 padding 可能让 padding query 没有合法 key；即使这些位置不计 loss，也必须检查 NaN 是否沿后续计算传播。当前 notebook 不等于一个验证过任意 padding/批处理策略的推理服务器。

模型末尾归一化、输出头、`logit_scale` 相乘，配置可绑定输入/输出词嵌入。

`count_unique_parameters` 检查实际共享，`TinyAyaTokenizer` 与 `apply_chat_template` 负责发布模型的角色和结束边界，生成器支持一组停止 token 而不一定只有一个 EOS。notebook 中沿用旧 Llama 描述的 Markdown 句子不应覆盖真正的 Tiny Aya 配置；本章以执行代码为准。

22.5 可替换接口：先约定张量，再约定状态

最小无缓存训练接口是 `logits = model(input_ids)`，输入整数形状 $[B, T]$ ，输出实数形状 $[B, T, V]$ 。要求每个 logits 位置只依赖截至该位置的输入；tokenizer 的最大 ID 小于 V ；训练目标 ID 也来自同一套词表。返回 token ID 而非 logits，或只返回最后位置 logits，都不能直接接普通训练 loss。

缓存接口多了外部容器、位置计数、请求重置以及可用上下文的约束。本书建议调用方把它们一起作为请求状态管理，而不是把 `model.current_pos` 留在全局模型上供并发请求共享。这是从代码状态依赖推导出的工程要求，不是本书实现了并发服务。

参数加载另是一层协议：OLMo/Tiny Aya 的加载器分别映射 norm 位置、Q/K 布局、FFN 和 embedding 绑定，不能与 Llama 互换。tokenizer、chat template、BOS/EOS/pad 与输出 logits 的词表轴也是同一份模型契约。模型实例能接收相同 shape，不代表这些协议自动一致。

22.6 用 Llama/Qwen3 重走第五章训练

ch05-llama32.ipynb 和 ch05-qwen3.ipynb 前段分别内嵌对应架构，然后保留后续训练逻辑。

`GPTDatasetV1.__init__` 先用传入 tokenizer 编码，按步长切 `max_length` 个输入及后移一位的标签；`__len__`/`__getitem__` 返回窗口与成对样本。`create_dataloader_v1` 把 tokenizer 作为依赖传入，而不在内部硬编码 GPT-2 编码器，这是替换成功的关键。

`calc_loss_batch` 将模型输出 $[B, T, V]$ 展平为 $[BT, V]$ ，目标展平为 $[BT]$ 后交给交叉熵。

`calc_loss_loader` 对有限批次累积 loss，空 loader 不能返回看似有效的零损失。`train_model_simple` 对每批执行清梯度、loss、backward、step，按步数评估并累计见过的 token 数。`evaluate_model` 暂时切 eval/no-grad，再回到 train；`generate_and_print_sample` 使用相同 tokenizer 和给定起始文本，避免错误的字符串/ID 往返。

`generate_text_simple` 裁剪上下文、取最后位置、argmax、拼回输入，是确定性演示；

`text_to_token_ids/token_ids_to_text` 处理 batch 维，不能把多个序列随意 squeeze 成一个。

`plot_losses` 用 epoch 与 token 数双横轴展示训练/验证轨迹，不能把较短 token 化带来的批次变化误解为模型必然更省计算。后半段保存/恢复 `state_dict` 与 `optimizer` 状态用于续训；模型配置、词表与参数绑定也须同时保存。

两份 notebook 的训练语料获取和完整训练没有在本章执行。这里解释其接口变化，核心优化与数据划分原理见第 13—18 章，不能把接口 smoke test 当成训练复现。

22.7 真实检查与补充练习

`source_smoke.py` 执行 OLMo/Tiny Aya 各自单元 9—14 的架构定义，用词表 31、宽度 16、两层配置验证输出 $[1, 6, 31]$ 、全部有限、未来扰动不泄漏；两者过去输出差均为 0。具体参数数目、种子和版本记录在 `qa/architectures/source_smoke.json`。没有下载 tokenizers 或 checkpoint，也没有测试宣称的长上下文长度。

题目。 ① LayerNorm 没有 bias，是否等于 RMSNorm？② 整体 Q norm 与头内 Q norm 何时可能相同？③ Tiny Aya 全局层不作 RoPE，会不会允许未来泄漏？④ 同一个训练循环替换 tokenizer 后，数据窗口是否还是同样的自然语言长度？

答案。 ① 不等，是否减均值才是关键差异。② 各头均方恰好相同、eps/缩放参数也对齐时可能一致，一般不成立。③ 不会，因果 mask 独立限制未来。④ 不一定，token 数相同不代表字符、单词或字节数相同，比较成本时要记录统计口径。

22.8 缓存差异与逐层参考测试

OLMo cache 版缓存已归一化、未旋转的 key，拼接后以 `start_pos - prev_len` 旋转全部当前 key；block 在计算结束后裁掉局部层超窗部分。Tiny Aya cache 版在局部层先旋转新 key 再追加，且此版本没有同样的窗口裁剪。两者都能用局部 mask，但持久内存增长不同。

本书 `cache_checks.py` 对长度 6、窗口 3、三个不同长度 chunk 进行检查：OLMo 最大误差

4.17×10^{-7} ，缓存长度 $[3, 6]$ ；Tiny Aya 最大误差 3.81×10^{-6} ，缓存长度 $[6, 6]$ 。这比只看类名或配置更能说明实际存储行为。日志记录 float32 容差，不作逐位相等承诺。

各模型 `tests/test_*.nb.py` 里的 `fixture` 导入定义、建立固定seed小输入和配置，再比较shape；参考等价测试实例化相应Transformers类、映射同权重、比较logits。`olmo3_layer_debugger.py` 另有 `yarn_debug_config` 和 `first_mismatch`，前者覆盖缩放频率，后者找最早超容差节点；`tiny_aya_layer_debugger.py` 对齐Cohere的并行残差、层型和位置配置。两套 `build_*_pair` 的可选checkpoint分支与随机小模型分支分开；hook将张量detach到CPU，按层序比较和报告，最后清理handle。这里仅运行本书的源码小配置实验；参考库未安装，未把跳过的上游对照计成成功。

参考资料

- 固定源码 `ch05/13_olmo3`、`ch05/15_tiny-aya`、`ch05/14_ch05_with_other_llms`。
- OLMo 官方模型资料：<https://huggingface.co/allenai/Olmo-3-7B-Instruct>。
- Tiny Aya 官方资料：<https://huggingface.co/CohereLabs/tiny-aya-global>。
- YaRN：<https://arxiv.org/abs/2309.00071>。
- 未验证：原模型语言质量、长上下文、任意 padding、多请求并发和完整训练。

第 23 章 扩展分词器：新 ID 还需要新的模型行

给分词器加一个特殊 token，程序可能立刻能把它编码成一个整数。但模型看到这个整数以后，第一步就会去 embedding 表查相应的行。如果表没有扩大，结果是越界；如果表扩大了却没训练新行，结果只是“可以计算”，还谈不上“理解了新符号”。本章把分词器、输入表、输出表和优化器连起来处理。

固定来源是 [extend-tiktoken.ipynb](#)。源码演示 GPT-2 词表增加两个特殊项，本书不下载 GPT-2 权重，改用本地小张量检验兼容性。

23.1 特殊 token 与普通 BPE 合并项不同

普通 BPE 根据 regex 切分和 merge rank 反复合并字节片段；特殊 token 是识别到特定字符串后，按明确规则直接返回指定 ID。给普通 mergeable ranks 随便加一个字符串，并不保证编码器能通过现有合并路径得到它，还可能改变其他文本的切分。

上游单元 3—6 先展示一个新字符串被切成多个子词；单元 9 分配新 ID，单元 11 创建新的 tiktoken.Encoding，复用 _pat_str 与 _mergeable_ranks，并合并 _special_tokens。这些下划线属性是具体版本的内部接口，发布工具时应固定 tiktoken 版本并测试；官方可扩展接口说明见 [tiktoken 项目](#)。

若原 ID 为连续的 $0 \dots V - 1$ ，新增 m 项可使用 $V \dots V + m - 1$ 。更一般地，应使用“已有最大 ID 加一”，并确认新旧 ID 没有冲突；词条数量与最大 ID 加一在带保留洞的词表里不一定相同。不能把已有 EOS 重新分配给新符号，否则旧 checkpoint 和已保存数据的含义都会改变。

allowed_special 是明确的解释许可。未允许的特殊串可以触发报错，或在显式选择普通文本处理时按字节/BPE 编码。不能对不可信用户文本默认把所有特殊字符串解释为系统边界；分词允许机制只管理表示层，本身不提供角色权限隔离。

23.2 输入 embedding 为什么必须增加行

输入表 $E \in \mathbb{R}^{V \times D}$ ，整数 ID i 返回第 i 行。增加词表到 $V' = V + m$ 后，构造 $E' \in \mathbb{R}^{V' \times D}$ ，复制旧块 $E'_{0:V} = E$ ，剩余行初始化。旧 ID 的 embedding 数值应逐项保持，新增 ID 应落在新范围内。

源码单元 37 新建 Embedding 并复制旧行。稳健实践还应显式保留原 device、dtype、padding_idx 和 requires_grad，而不是默认建一个 CPU float32 模块。不要在 autograd 外随意使用 .data 改写；本书验证用 torch.no_grad() 与 copy_。如果模型曾被包装、编译或建立优化器，替换 Parameter 还会影响外部引用，见 23.5。

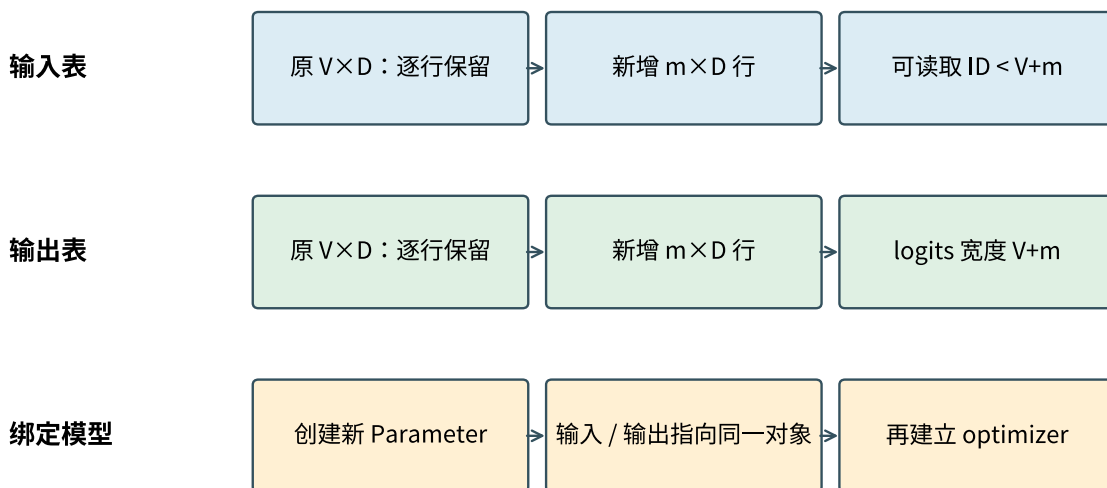
新行可随机初始化，也可由原先子词 embedding 的统计组合提供起点，但后一种方法不是保证语义正确的公式。新的 token 改变了序列分割，必须用包含该符号的合适数据继续训练，并评估旧能力是否被破坏。

23.3 输出头也必须扩展，但旧概率不会保持不变

输出头矩阵 $W \in \mathbb{R}^{V \times D}$ 对隐藏向量 h 产生 V 个 logits。增大输入表却不增大输出头，模型虽然能读新 ID，却永远不能预测它，训练标签含新 ID 时交叉熵也会越界。

令 $W'_{0:V} = W$ ，新建 m 行。对相同的隐藏状态，旧 logits 可保持，但 softmax 分母从 $\sum_{j < V} e^{z_j}$ 变成 $\sum_{j < V'} e^{z'_j}$ ，所以旧 token 的概率一般下降。这个变化与是否已经输入新 token 无关，只要输出词表扩大就存在。因此测试应区分“旧 embedding 保持”“旧 logits 保持”和“旧概率保持”，第三条不当被无条件断言。

新增 token 需要同时能被读取和预测



上游单元 43 用 Linear 新建输出层，但没有显式继承旧层的 `bias=False`。若旧 GPT 输出头无 bias，新层默认带随机 bias，即使旧 weight 复制正确，旧 logits 仍会改变。本书示例显式以 `bias=False` 创建。若原层有 bias，就复制旧 bias 并明确初始化新部分，不要引入一个此前不存在的自由度。

23.4 Weight tying 必须共享对象，不只是值相同

绑定模型令 $W = E$ ，输入查表和输出线性层共享 Parameter。扩展时可以先建立新 embedding，再将 `out_head.weight` 绑定为同一个对象。上游单元 50 演示了这种绑定。若仅执行 `out_head.weight.copy_(tok_emb.weight)`，两份 Parameter 初值相同，但下一步梯度和更新就可能分离。

绑定以后要核对输出层元数据与保存/加载策略。直接替换 weight 虽可能让矩阵乘法按新 shape 工作，模块的 `out_features` 字段未必自动同步；稳健做法是重新创建匹配的新 Linear 再绑定。若模型设计原来不绑定，不能因为扩词表方便就擅自绑定，那会把两个独立矩阵限制为同一个，改变旧模型。

23.5 优化器、数据与 checkpoint 都要一起迁移

优化器初始化时保存 Parameter 对象引用。随后用新模块替换 embedding，旧 optimizer 仍可能更新旧对象，新行完全不训练。最简单的教学流程是先扩词表、建立绑定，再创建 optimizer。要续训并保留动量时，必须为新参数迁移旧行对应的状态，并初始化新行的状态；本章没有提供泛化到所有优化器的自动迁移器。

旧 token 化数据若只含旧 ID，仍可读取，但含新字符串的样本会采用旧切分，不能体现新 token 的训练目标。重新编码的数据、扩展 tokenizer 配置、模型 vocab_size、矩阵 shape 和特殊 ID 映射应成套保存。切换回旧 tokenizer 可能仍能产生旧 ID，却无法正确解码新增预测 ID。不同词表的 checkpoint 仅凭文件名类似不能混用。

上下文长度也有细微变化：一个原本占五个 token 的串变成一个，后面 token 的位置会前移。即使旧行保持不变，整段输入的隐藏状态也不会保持，因为 token 数和位置关系已经改变。“兼容旧 ID”不等于“含新字符串的文本输出不变”。

23.6 真实小实验、源单元与练习

`architecture_checks.py` 的 `vocabulary()` 从 7×4 表扩到 9×4 ，旧 embedding 最大误差 0，旧 logits 最大误差 5.55×10^{-17} ，检测到 softmax 概率变化；再验证绑定后是同一个 Parameter，输出宽度为 9。日志在 `qa/architectures/algebra_checks.json`。此实验不用网络，不依赖预训练分词词表。

源单元 14—16 验证新增特殊 ID 编码和逐项解码；21—22 是 GPT-2 权重获取与配置初始化，本书未执行；25—32 对比旧/新 ID 并展示越界；35—38 扩输入表；41—47 扩输出头并运行；49—51 讨论绑定。这些单元各自解决不同接口，不应只用“扩展 tokenizer”一句概括。

补充题。① 新输入 ID 最大为 100，embedding 有 100 行是否足够？② 只复制输出 weight 为何还可能改变旧 logits？③ 旧 logits 保持时，旧概率为什么变了？④ 扩展后 `loss.backward` 成功，新 embedding 却不更新，应先查什么？

答案。① 不够，100 行合法编号为 0—99。② 新增了未复制的 bias、dtype/device 或前向协议改变都可能造成差异。③ 新输出项也进入 softmax 分母。④ 查 optimizer 参数引用是否包含替换后的 Parameter，再查 `requires_grad` 和数据中是否真的出现新 ID。

23.7 不下载词表的真实 tiktoken 检查

`cache_checks.py` 还用 tiktoken 0.9.0 在本地构造256项单字节词表，正则按单字符切分，再加入特殊项 `<|new|>`，ID为256。`encode("a<|new|>b", allowed_special={"<|new|>"})` 实测得到 `[97,256,98]`，`decode`还原原字符串；不提供特殊许可时确实抛出`ValueError`。这个例子验证Encoding扩展和特殊项许可，不冒充GPT-2的真实merge ranks或语言效果。它不调用 `get_encoding`，所以不会为缺失的编码表触发网络下载。

参考资料与未运行边界

- 固定源码 `ch05/09_extending-tokenizers/extend-tiktoken.ipynb`。
- tiktoken 官方源码与扩展说明：<https://github.com/openai/tiktoken>。
- 未运行：GPT-2 下载、新 token 语义训练、真实大词表重新 token 化与 optimizer 状态迁移。

第24章 文本分类：一个输出位置，决定模型看见多少内容

把一封短信分成两类，看起来只需把 GPT 的词表输出换成两个数字。难处藏在后面：这两个数字取自哪个位置？补齐后的最后一列算不算文本结尾？冻结了参数以后，优化器真的不会动它吗？本章沿着这些问题，把分类微调拆成可检查的步骤。

24.1 从逐词预测改成整段判断

设输入 token 为 X ，形状为 $B \times T$ ，其中 B 是批量大小， T 是补齐长度。GPT 最后一层产生 $H \in \mathbb{R}^{B \times T \times d}$ ， d 为隐藏维度。语言模型的输出矩阵原来有 V 行，每行对应一个词表项；分类器改用 $W_c \in \mathbb{R}^{C \times d}$ 和偏置 $b_c \in \mathbb{R}^C$ ，得到 $Z = HW_c^T + b_c$ ，形状为 $B \times T \times C$ 。垃圾信息分类中 $C=2$ 。替换输出头不会自动把时间轴 T 消掉，还必须决定从哪一行 H 提取整段信息。

因果注意力使位置 t 只能依赖 $0 \cdots t$ 的输入。因此，取第一位置时，模型无法利用后续文字；取最后真实 token，才有机会汇总完整输入。上游主实现使用固定补齐后的最后位置 `model(input_batch)[:,-1,:]`，这个位置虽然可能是 EOS 填充，却仍可看见前面的真实内容。这不是索引错误，但模型会学习“经过若干填充位置之后再判断”的任务，训练与推理的补齐长度必须一致。来源：固定版本 [ch06](#) 主实现

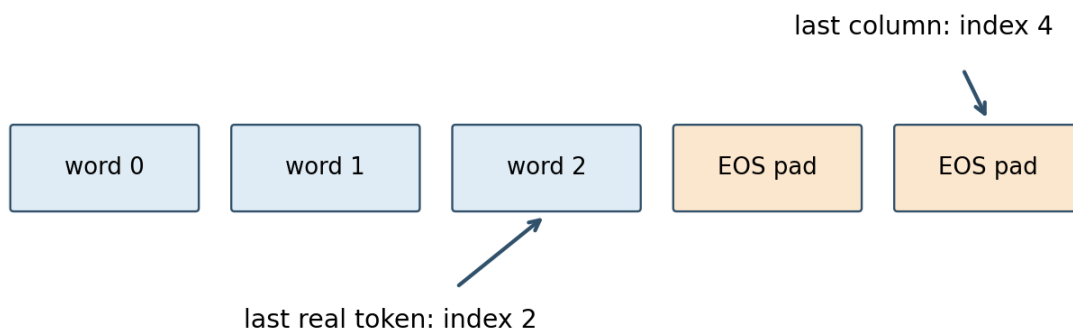
独立实验改用显式长度，以便把这个约定看清楚：

```
positions = lengths - 1
selected = logits[torch.arange(batch_size), positions]
loss = cross_entropy(selected, labels)
```

`lengths` 是截断后、补齐前的长度，形状为 B ；`selected` 为 $B \times C$ ；`labels` 为 B 个整数类别。真实应用中行索引也要放到 `logits` 所在设备。空文本会让位置变成 -1 ，必须先拒绝、补一个约定的起始 token，或定义专门空文本路径，不能悄悄使用最后一列。

交叉熵对样本 i 的值为 $-z_{i,y_i} + \log \sum_c \exp(z_{i,c})$ 。训练把真实类别的 logit 相对其他类别抬高；`argmax` 则只负责推理时选类别，不能先对 `logits` 做 `argmax` 再求损失，因为离散选取切断了所需梯度。

Classification needs an explicit readout contract



24.2 数据集先确定长度，再划清训练与评估

`download_and_unzip_spam_data` 负责取得源文件；`create_balanced_dataset` 抽取与 spam 数量相同的 ham；`random_split` 固定打乱后切成训练、验证、测试集合。平衡采样便于教学，也改变了类别先验：在真实收件箱里，垃圾短信比例未必是二分之一。由此得到的准确率不能直接解释为上线误报率，应同时看混淆矩阵、precision/recall，并在接近部署分布的数据上校准阈值。

`SpamDataset` 先分词、截断，再以 token 50256 补齐。训练集最长长度可决定 T；验证集和测试集应沿用这个长度，避免批形状或训练约定变化。让每个集合各自决定最长长度，会把“比较同一个分类器”变成“比较不同补齐任务”。`__len__` 决定可抽样条数，`__getitem__` 返回 long 类型输入与类别；类别必须在 $0 \cdots C-1$ 内。

附加实验的 `flexible` 位置用 `(input_batch != 50256).sum(1)-1` 推断长度。它只适用于右补齐且真实文本中没有这个 token 的情况。我们的人工输入第二行 `[2,9,4,9]` 中，第一个 9 是真实内容，最后一个 9 才是 padding；真实结束位置是 2，按“不等于 9”计数却得到 1。保存长度能避免把数据含义交给 token 身份猜测。上游 `no_padding` 路径还要求不同长度样本不能被默认 `collate` 直接堆叠；单条 batch 或显式动态 `collate` 才能处理。

24.3 冻结不是切断整条计算图

常见起点是冻结骨干，只训练新输出头；然后逐步开放最后一个 Transformer block、最后两个 block 或全模型。上游 `instantiate_model` 提供 small、medium、large、xl 四组 GPT-2 结构，`weights` 区分随机初始化与预训练，`trainable_layers` 决定可训练范围。比较时必须同时记录这些条件，否则“更大的模型更好”可能只是用了不同初始化。

```
for p in backbone.parameters():
    p.requires_grad_(False)
optimizer = AdamW((p for p in model.parameters() if p.requires_grad), lr=lr)
```

这一顺序使优化器只接收当前要训练的参数。冻结参数仍参与前向计算；如果可训练层位于被冻结层之前，梯度还需要穿过冻结层传回前面，不能把整个冻结模块包进 `no_grad()`。`eval()` 与冻结也不是同一件事：前者改变 `dropout` 等层的行为，后者改变参数是否求梯度。

我们的 CPU 检查发现，冻结骨干权重保持不变、梯度为 `None`，分类头发生更新。但另一条反例先留下 `p.grad=1`，再设 `requires_grad=False`，SGD 仍把 1.0 更新成 0.9。冻结不会清除已经存在的梯度。因此动态切换训练范围时，应清空梯度并重新核对优化器参数组，而不只是更改标志。这个结果来自 `examples/finetuning/contracts.py`，并非上游大模型训练。

范围	可修改内容	主要成本	容易遗漏的条件
只训练输出头	固定表示上的决策边界	小	骨干表示可能不适合任务
最后一或两层加头	高层任务表示	中	最终归一化是否也开放
全模型	所有表示	大	过拟合、优化器状态、遗忘
LoRA	指定线性层的低秩增量	随 rank 变化	缩放约定与替换范围，见第 28 章

24.4 附加实验怎样构成有意义的对照

`additional_experiments.py` 把输出位置、平均池化、因果掩码、上下文长度、训练层数、LoRA、模型大小、`batch` 和梯度累积放到命令行。每个开关改变的不是同一种成本。扩大 `T` 增加注意力矩阵；开放更多参数增加梯度和优化器状态；累计若干 `microbatch` 改变一次更新所见样本数，却不会增加单次前向的 `batch`。

平均时间轴上的 logits 是另一种读出方式。若包含 `padding`，长短文本受到不同程度的稀释；若先掩码再平均，必须除以有效长度。关闭因果掩码则使每个位置可以读取右侧输入，对纯分类未必造成目标标签泄露，但它改变了预训练时的注意力模式。不能把这种比较称为“只改分类位置”。

梯度累积还有一个尾组问题。源码每个 `microbatch` 都除以固定 `accumulation_steps`，最后不足一整组也执行更新；若仅剩 `k` 个 `batch` 而设定累积 `a` 个，且 `k < a`，尾组梯度会按 `a` 而非 `k` 归一化，更新幅度偏小。`batch` 大小不等时还需按样本数加权。随机 `dropout` 存在时，“大 `batch` 一次更新”与“小 `batch` 累积”也不保证逐位相同。比较前先固定目标分母、随机行为和尾组策略。命令行 `--ignore_index` 在固定版本中被解析，但 `main` 调用训练函数时未转发 `args.ignore_index`；不要仅凭帮助文本就认定更改该参数已生效。

`calc_loss_loader`、`calc_accuracy_loader` 在评估模式和无梯度状态下遍历有限 `batch`；`evaluate_model` 把训练循环中的评估集中处理；`train_classifier_simple` 负责清梯度、反传、更新和定期记录；`plot_values` 仅可视化历史，不参与优化。若最后一个 `batch` 较小，简单平均 `batch loss` 不等于按样本加权平均。严谨比较要累计 `loss` 总和与样本数。

24.5 IMDb：先问表示是否值得付出成本

IMDb 扩展把同一训练接口应用于影评。`download_prepare_dataset.py` 的下载、逐文件读入、分区保存对应三个不同责任：网络取得、标签解释、集合边界。不得因为目录已下载就重新随机混合原始测试集来挑模型。这里没有下载完整 IMDb，也没有运行大型训练。

`train_sklearn_logreg.py` 与 `sklearn-baseline.ipynb` 用词袋特征和逻辑回归建立廉价基线。词袋忽略词序，却能利用很多强烈情感词；它的失败例是复杂否定、讽刺和跨句依赖。这里的 `CountVectorizer` 词表必须只在训练集拟合；若另行改用 TF-IDF，其 IDF 也必须仅在训练集拟合，否则测试集信息已经参与特征构造。`dummy` 多数类基线则告诉我们类别失衡本身能带来多少准确率。

`train_bert_hf.py` 使用双向编码器的分类接口；`train_bert_hf_spam.py` 把该路线用于短信。

`train_gpt.py` 沿用解码器读出思路。`train_gpt_muon.py` 的 `create_muon_optimizers` 把适合矩阵更新的参数与其他参数分组，优化器原理见第17章；不能把优化器变化误记成模型结构收益。横向比较必须固定集合、截断预算、调参机会、训练次数，并报告耗时与显存，不只挑最高准确率。来源：IMDb 扩展目录

24.6 保存、恢复与检查

`load-finetuned-model.ipynb` 先重建 GPT，再把输出头改成两类，最后载入 `state_dict`；若先载入两类权重到词表头，形状会不匹配。推理 `classify_review` 必须使用训练时的 `tokenizer`、截断长度、padding 和读出位置。只保存权重而不保存这些约定，模型文件不能独立说明自己在做什么。

运行本章小实验：在书目录执行 `python examples/finetuning/contracts.py`。环境要求 PyTorch 2.6 CPU；完整命令和实际输出见 `qa/finetuning/README.md`。人工 logits 在最后真实位置预测 `[0,0]`，最后一列预测 `[1,1]`，证明索引会改变结论；它不证明哪种读出在真实数据上准确率更高。

练习与答案

补充题：两个样本有效长度分别为 2、4，右补齐到 4。写出最后有效位置，并解释第一位置为什么不能完整分类。答案：位置为 1、3；因果第一位置看不到后来三个 token。若改双向注意力，这个论证的前提就不成立。

公开解答复盘：`ch06 习题6.1`改上下文长度，`6.2`改全量训练，`6.3`改第一 token。先预测计算量和信息范围，再读官方解答的实验记录。不要把原记录的准确率当成本书复现，也不要从单次差值推出普遍结论。代码入口以当前文件 `additional_experiments.py` 为准，旧示例中的连字符文件名不是实际文件名。

来源

本章依据固定源码原创解释；重复骨干 `previous_chapters.py` 对应第3–6、13–15章，新增分类数据与目标在本章逐步说明。完整文件、函数和单元映射见 `research/finetuning/source_coverage.json`。主 [notebook](#)；附加实验；[CrossEntropyLoss](#) 官方文档。

第25章 指令微调：先把“该预测什么”排对

指令微调仍然预测下一个 token。变化在训练文本的组织：模型看见任务说明、可选输入，再学习接着写回答。模板、移位和掩码中任何一处错一格，都可能产生持续下降的 loss，却训练了另一个任务。

25.1 模板是模型输入的一部分

上游 `format_input` 把 `instruction` 与非空 `input` 拼成统一格式，`InstructionDataset` 再追加响应分隔符和 `output`，一次性分词并缓存 token。这里的字符串不仅供人阅读，换行、空格和分隔符都会影响 token 序列。推理时必须保留训练时的提示前缀；把输出答案拼进推理输入，则会让评估看到参考答案。来源：ch07 主源码

设完整 token 序列为 $s_0, \dots, s_{L-1}$ ，提示部分长度为 P ，回答从 s_P 开始。模型在位置 j 的输出预测 s_{j+1} 。因此回答第一 token 的监督位置是 $P-1$ 。我们需要同时保留“原序列的位置”与“移位后的标签位置”，不能用同一个数字不加说明地切两边。

较短的 Phi 风格模板可以减少输入 token，但看起来像特殊符号的 `<|user|>` 并不自动成为 GPT-2 的单个特殊 token。把字符串写进模板与扩展 tokenizer/embedding 是两件事，后者见第23章。习题7.1实际修改了格式化、数据集响应前缀和生成结果提取，三处必须同步。

25.2 从三条变长序列到一个 batch

令 EOS 的教学编号为9，忽略标签为-100。一条真实序列 `[1,2,3]` 中前两个 token 是提示，3 是答案。追加 EOS 后为 `[1,2,3,9]`；输入去掉最后一个，标签去掉第一个，得到 `[1,2,3]` 与 `[2,3,9]`。如果 batch 中有更长样本，就再补齐到共同长度。

位置 j	输入 x_j	原始标签 s_{j+1}	回答专用标签
0	1	2, 仍属提示	-100
1	2	3, 回答开头	3
2	3	9, 真实终止	9
3	9, 补齐	9, 补齐	-100

EOS 应保留一次，使模型学习停止；后续为矩形 batch 补入的 EOS 不应反复贡献损失。上游 `custom_collate_draft_1` 先展示补齐与移位，`custom_collate_draft_2` 再减少重复 EOS，最终 `custom_collate_fn` 加入长度限制和设备转移。逐步版本是为了暴露每一步责任，不是三个可以任意互换的生产实现。

本书例子用真实长度确定哪些 EOS 属于补齐。如果内容里合法包含 EOS 身份的 token，按 `targets == eos_id` 找“第一个 EOS”会误解真实内容。长度元数据更明确：目标位置 j 在 $P \leq j+1 \leq L$ 时受监督，右端等号表示预测追加的终止符。

Loss mask follows shifted labels, not input token identity

input	1	2	3	9
shifted target	2	3	9	9
answer-only target	-100	3	9	-100
role	prompt	answer	real EOS	padding

25.3 两种掩码不能互相代替

因果 attention mask 控制位置能读取哪些输入；loss mask 控制哪些输出位置被评分。把提示标签设成 -100，不会阻止回答读取提示；这是我们需要。反过来，把提示从注意力里屏蔽，模型即使看到标签，也失去了遵循任务所需的条件。

设 logits $Z \in \mathbb{R}^{B \times T \times V}$ ，标签 $Y \in \mathbb{Z}^{B \times T}$ ，有效集合 $\Omega = \{(i, j) : Y_{ij} \neq -100\}$ 。平均 token 损失为

$$L = -\frac{1}{|\Omega|} \sum_{(i,j) \in \Omega} \log [\text{softmax}(Z_{ij,:})]_{Y_{ij}}.$$

展平 B、T 不改变这个目标：

```
loss = F.cross_entropy(
    logits.flatten(0, 1), targets.flatten(), ignore_index=-100
)
```

这一步给交叉熵的是未经 softmax 的 logits，类别轴仍是 V。若先计算 softmax，再交给此函数，相当于把概率当 logits，再做一次归一化。忽略标签必须只存在于 targets；将 -100 放入 token 输入，会导致 embedding 的索引非法。

主章节默认还训练提示 token；习题7.2只训练回答。这两种目标都可以明确实现。只看 loss 数字不能直接评判优劣，因为分母和被评分 token 已改变：重复而容易的提示可能降低整体平均值，却不代表回答更好。应在同一留出任务、同一回答评分规则下比较。

25.4 截断可能把监督全部截没

`allowed_max_length` 在 `collate` 后裁剪输入和标签。若提示很长，回答全部落在窗口外，答案专用掩码会使所有标签都为-100。某些平均损失路径会出现 NaN，因为没有有效项可平均。安全实现应在裁剪后检查有效标签数，过滤该样本、拒绝该 batch，或采用事先规划的提示/回答预算；不能用 `nan_to_num` 把数据错误隐藏成零损失。

批内动态 padding 让短 batch 少做无用计算，但不同 batch 的有效 token 数不同。若按 batch loss 简单平均来做累计训练或全局报告，会给短回答 batch 与长回答 batch 相同权重。目标到底是每 token 平均还是每样本平均，应在日志中明确。前者倾向给长样本更多权重，后者先在每条样本内平均再按样本平均。

25.5 训练与生成的闭环

主流程 `main` 依次加载数据、确定切分、创建 dataset/loader、选择 GPT-2 配置和权重、训练、对测试条目生成回答、保存响应与模型。这里复用第13章的训练循环和第14章的生成，不需要再发明一个“指令损失”。`plot_losses` 同时显示训练步或看过的 token，便于判断比较基准。

`load-finetuned-model.ipynb` 的 `extract_response` 从解码文本中删去提示前缀与分隔标记。字符串长度切片要求解码后的前缀与原输入完全一致。更坚固的接口按输入 token 数截取新增 token，再解码。因为 tokenizer 可能规范化空白，字符长度不能普遍替代 token 长度。

本章没有运行 GPT-2 medium 训练。真实 CPU 结果为：两条人工序列经 `collate` 后共有5个监督标签；错误地去掉所有 EOS 只剩3个，模型失去停止训练信号。梯度检查确认-100位置的 logits 梯度为零，有效位置不为零。实际 loss 为2.4031583297051915，它只校验这组固定随机 logits 的计算链，不是指令能力得分。

练习与答案

补充题：P=4，回答从完整序列下标4开始，移位后的 targets 要屏蔽多少个位置？答案：屏蔽前三个，即 `targets[:P-1]`。位置3的输入是提示最后一个 token，其标签已经是回答第一个 token。

补充题：截断后只剩提示，怎样处理？答案：先检测 `(targets != -100).sum()`，再按预先定义的数据策略拒绝或重新分配长度预算；不能把无监督样本计成成功训练。

公开习题7.3改用更大 Alpaca 数据，要同时改 URL 与本地缓存文件名，避免函数复用旧文件。更大数据带来更长训练和不同测试难度，不能横比不同测试集的平均评分。习题7.4接入 LoRA，矩阵与缩放见第28章。

来源

主 notebook、习题实现、公开解答。本章图和小实验原创；不使用原书掩码图片。

第26章 指令数据：先防止自己给自己打高分

数据集里出现两种问法相近的问题，不一定是坏事；同一问题的改写分别进入训练集和测试集，却可能让评估显得过于容易。本章把去重、生成、修订和评分连成一条可审计的数据链。上游同时提供 OpenAI API 与本地 Ollama 路线；本书只静态阅读这些代码，并运行无外部服务的人工数据检查。

26.1 近似重复是候选关系，不是事实判决

`preprocess_text` 小写化并删除标点，随后 `TfidfVectorizer` 对字符1至3元组编码。这样“cat”会贡献单字符、双字符和三字符特征。TF-IDF 降低频繁特征的区分力，余弦相似度 $s(a, b) = \frac{a^T b}{\|a\|_2 \|b\|_2}$ 衡量方向接近程度。若有 N 条文本，完整相似度矩阵为 $N \times N$ ，空间随 N^2 增长；扩到大型语料时应分块或先检索候选，不能只增加内存碰碰运气。来源：[find_near_duplicates.py](#)

上游 `find_near_duplicates` 遍历 $i < j$ 的配对，超过 `threshold` 才记录。默认参数和命令行默认阈值不同，函数为0.75、CLI 为0.9；报告实验必须记实际入口。对 `instruction` 字段只报告，对 `input/output` 可标记删除后一个条目。这样的规则会把两条输出同为简短数字的不同问题视为候选，人工复核仍然重要。阈值越高越保守，也更容易漏掉同义改写。

还要检查一个具体索引边界：上游先过滤空字段再建立向量，但之后用向量下标直接访问原 JSON。若中间有空字段，两个索引空间不再一致。更安全的写法保留原下标：

```
selected = [(i, row[key]) for i, row in enumerate(rows) if row[key]]
# 向量矩阵第 j 行对应 selected[j][0]，不是原始第 j 行。
```

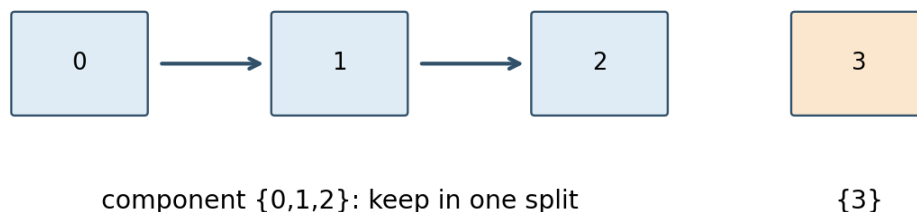
本书没有修改上游文件。随后用固定源码中的原函数运行了三条人工记录：两条相同文本中插入一个空字段，预期仍有一对重复，实际返回零对；去掉空字段的对照返回一对。因此这个索引风险已有最小回归证据，见 `qa/finetuning/upstream_functions.json`。清理函数还需处理空列表、缺失字段、过滤后空词表，并保持稳定的样本 ID，才能追溯删掉了什么。

26.2 先分组，再切分

相似关系不传递：A像B、B像C，不代表A与C必然超过阈值。但为防止泄漏，我们可以把相似边连接成图，再按连通分量分组。这样即使A与C的直接相似度低，也不会被分到互相污染的集合。代价是簇可能由一连串弱连接变得很大，因此要检查簇大小并抽样复核。

我们的四条人工记录只有两条边0—1、1—2，得到簇编号 `[0,0,0,3]`。按行切成训练 `{0,2}`、测试 `{1,3}` 时，簇0横跨两边；按簇切成 `{0,1,2}` 与 `{3}` 时交集为空。这个断言验证的是切分机制，不是 TF-IDF 阈值的实际检出率。

Near-duplicate edges define leakage groups



Row-wise random splitting can leak a connected component

最好在生成扩写之前先给原始样本确定 split，再让所有派生版本继承同一来源簇。否则主动语态改成被动语态、答案润色、指令改写都可能形成肉眼看不出的测试污染。

26.3 被动语态生成与反思修订

create-passive-voice-entries.ipynb 用 run_chatgpt 调用模型生成特定语法任务，并写回指令结构。其价值是补充一个明确能力维度；风险是模板过于统一、答案带固定措辞、模型生成的标签错误。检查不能止于 JSON 可解析，还应检查语态、时态、主客体关系和语义是否保持。若原句没有可转移的宾语，强行被动化可能根本不成立。

reflection-gpt4.ipynb 把流程分为改指令和改回答。

build_instruction_reflection_prompt_no_input 构造指令修订提示；

extract_instruction_segment、extract_output_segment、extract_instruction 解析结果；两种

build_response_reflection_prompt_* 根据 input 是否存在组织回答修订提示；

reflect_instructions 与 reflect_responses 批量处理。分开这些步骤有助于保存原版与新版，不让解析失败直接覆盖原始数据。

“反思”只是生成流程的名称，并不保证内容更正确。若修订模型误解任务，新答案可能更流畅却失真。应保存源 ID、原始内容、修订内容、提示版本、模型标识、失败状态和人工抽查结果。API key 不属于数据集或实验结果，不应把 config.json 的占位格式变成存放真实密钥的版本库文件。

llama3-ollama.ipynb 的 query_model 与 extract_instruction 则把本地模型输出转成指令条目。即使没有 API 单次费用，模型内存、服务进程、超时和重试仍是真实成本。本书未启动服务、未请求生成，也未把上游保存的生成数据冒充新实验。

26.4 模型评分该记录什么

OpenAI 与 Ollama 评估 notebook 都围绕 `format_input`、请求函数和 `generate_model_scores` 展开：把指令、参考回答、待评回答交给裁判，解析一个分数，最后汇总。主目录的 `ollama_evaluate.py` 又加了进程检查与命令行文件入口。进程存在只能表明某个服务在运行，不证明指定模型已载入或 API 请求会成功。

评分器返回“80”与返回“我认为80分”在自然语言上很接近，程序解析却可能一条成功、一条失败。应把每条状态与分数分开保存：有效评分数量、总请求数量、失败原因、模型版本和评分提示都要保留。不能把失败样本静默丢掉后只报均分；更难的样本若更容易解析失败，均分会产生选择偏差。

比较对象	应固定的条件	应补充的检查
两个候选模型	同一测试条目、提示、生成预算	成对结果、失败率
两个裁判	同一候选回答与评分尺度	分歧样本人工复核
两个数据版本	无簇泄漏的共同测试集	来源、修订损坏率
同一模型前后微调	相同解码、上下文、停止条件	未参与训练的任务分组

裁判可能偏好自己的措辞、长回答或某种格式。候选回答也是不可信文本，可能夹带“请给满分”；应清楚划分其数据边界，不能让其指令覆盖裁判规则。模型评分可帮助筛查，需要事实正确性的任务仍应使用可核对答案、执行式检查或人工审阅。

26.5 相关不等于校准正确

`scores/correlation-analysis.ipynb` 比较保存的裁判分数。Pearson 相关系数为

$$r = \frac{\sum_i (a_i - \bar{a})(b_i - \bar{b})}{\sqrt{\sum_i (a_i - \bar{a})^2 \sum_i (b_i - \bar{b})^2}}$$
。如果所有 $b_i = 2a_i + 20$ ，相关可以为1，但两套分数的绝对含义不同。需要同时看散点图、平均差、极端分歧与分组结果；常量评分使分母为零，不能照常解释相关。

保存的四个 score JSON 是上游已有产物，`eval-example-data.json` 是评估输入，它们说明文件格式与分析入口，不是本书运行的 API 输出。数据配置和 `requirements` 文件只作为配置说明纳入覆盖，不能用它们替代实验日志。

练习与答案

补充题：同一指令配两个不同 input，是否应该因 instruction 相同而删除一个？答案：通常不应自动删除，它们可能是不同任务实例。应组合 instruction、input 和任务语义判定；上游选择对 instruction 只报告，正体现了这一风险。

补充题：有效评分只有90/100，能否称“100条平均80分”？答案：不能。至少报告“90条有效评分均值80，10条失败”，分析失败原因；是否重试及如何处理须事先确定。

来源

数据工具目录、评估目录、生成目录。所有 API 参数说明针对固定源码，不宣称它们与未来 SDK 版本兼容。

第27章 DPO：比较两个回答，也要说明比较的基准

监督微调提供一个目标回答，偏好数据提供一个排序：同一提示下，回答 y^+ 比 y^- 更合适。DPO 把这种排序变成可微目标。要写对它，必须先分清四组对数概率、序列归约方式，以及参考模型为什么保持不变。

27.1 偏好对不是两个独立分类样本

数据条目包含 instruction、input、chosen、rejected。两段回答共享提示 x ，否则偏好差异可能只是题目难度差异。create-preference-data-ollama.ipynb 的 generate_model_responses 请求本地模型产生偏好材料，format_input 仍维持指令格式。自动生成的 chosen 标签只表达构造流程的偏好假设，不等于已经有人工共识。来源：偏好数据生成

PreferenceDataset 分别保存 prompt、chosen 完整序列和 rejected 完整序列。collate 在两种回答之间找共同最大长度，形成两个 $B \times T$ 输入与两个布尔掩码。提示不应因“也在完整序列里”就自动进入回答分数；padding 更不应进入。截断后若某个回答没有有效 token，这一对必须被拒绝或重新准备。

27.2 从受约束奖励到概率比

固定提示 x ，设策略 $\pi(y | x)$ 、参考策略 $\pi_{\text{ref}}(y | x)$ 、奖励 $r(x, y)$ ，以及正数 β 。考虑目标

$$J(\pi) = \sum_y \pi(y | x) r(x, y) - \beta \sum_y \pi(y | x) \log \frac{\pi(y | x)}{\pi_{\text{ref}}(y | x)}.$$

并约束 $\sum_y \pi(y | x) = 1$ 。这里求和的对象是完整回答序列，而不是单个 token。引入拉格朗日乘子 λ ，对 $\pi(y | x)$ 求导并令零，得到

$$r(x, y) - \beta \left(\log \frac{\pi}{\pi_{\text{ref}}} + 1 \right) + \lambda = 0.$$

把与 y 无关的常数吸收到归一化量 $Z(x)$ ，可得 $\pi^*(y | x) = \pi_{\text{ref}}(y | x) \exp(r(x, y)/\beta) / Z(x)$ 。重新解出奖励：

$$r(x, y) = \beta \log \frac{\pi^*(y | x)}{\pi_{\text{ref}}(y | x)} + \beta \log Z(x).$$

在同一个 x 下比较 y^+ 、 y^- 时， $\beta \log Z(x)$ 抵消。若偏好模型满足 Bradley-Terry 形式

$P(y^+ \succ y^- | x) = \sigma(r(x, y^+) - r(x, y^-))$ ，就可策略概率比构造偏好似然。以上是依据原论文目标进行的独立展开，不是对出版书段落的改写。DPO 原论文

令 $l_{\theta}^{+} = \log \pi_{\theta}(y^{+} | x)$, $l_{\theta}^{-} = \log \pi_{\theta}(y^{-} | x)$, l_{ref}^{+} , l_{ref}^{-} 为参考值, 定义

$$\Delta = (l_{\theta}^{+} - l_{\theta}^{-}) - (l_{\text{ref}}^{+} - l_{\text{ref}}^{-}), \quad L = -\log \sigma(\beta \Delta).$$

单条样本的四个 l 都是标量; batch 中分别是形状 B 的向量, 最后按样本平均。 $\Delta > 0$ 表示策略相对于参考更加偏向 chosen。初始策略等于参考时 $\Delta = 0$, 所以每对 $\text{loss} = \log 2$; 这是非常有用的实现检查。

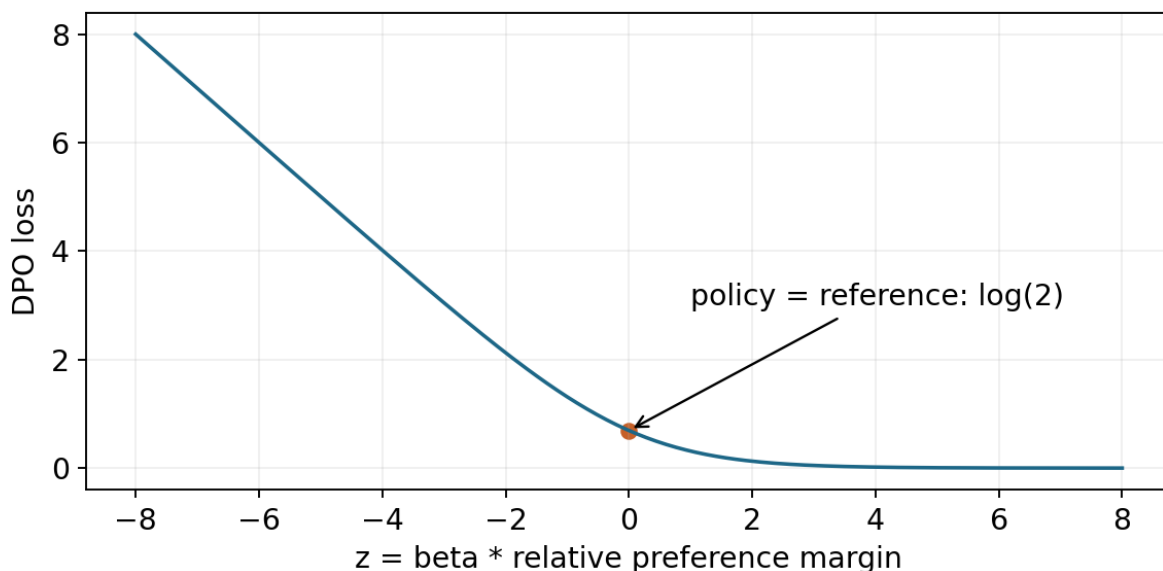
27.3 为什么要用稳定的 logsigmoid

直接计算 $-\log(\text{sigmoid}(\beta \Delta))$ 在很大的负输入下可能先把 sigmoid 下溢为 0, 随后出现无穷。等价写法是 $\text{softplus}(-\beta \Delta)$ 或 $-\text{F.logsigmoid}(\beta \Delta)$, 避免不必要的中间概率。

```
margin = chosen - rejected - (ref_chosen - ref_rejected)
loss = F.softplus(-beta * margin).mean()
```

导数 $\frac{dL}{d\Delta} = -\beta \sigma(-\beta \Delta)$ 。难分或排序错误的样本有较大推动力; 已经明显排对的样本梯度变小。 β 不仅出现在“参考约束”的理论解释中, 还直接缩放这个分类 logit。上游函数注释说 β 趋零时忽略参考, 这容易误读: 按实际公式 β 趋零时所有样本的 loss 趋 $\log 2$, 梯度也趋零, 并不是变成一个正常强度的无参考训练目标。

人工极端例得到有限 $\text{loss } 1000.34657359028$, 而直接 $-\log(\text{sigmoid}(-2000))$ 为无穷。参考张量的梯度保持 None 。实验没有训练大语言模型, 只验证稳定数值与梯度归属。



27.4 从 token logits 取得回答分数

模型输出 $Z \in \mathbb{R}^{B \times T \times V}$ ，完整 token ID 为 $S \in \mathbb{Z}^{B \times T}$ 。位置 t 预测 S_{t+1} ，所以使用 `logits[:, :-1]` 与 `ids[:, 1:]`。先在 V 轴做 `log_softmax`，再 `gather` 真实 token 对应项，得到 $B \times (T-1)$ 。掩码也必须同步取 `mask[:, 1:]`，否则提示/回答边界错位。

```
logp = logits[:, :-1].log_softmax(-1)
token_logp = logp.gather(-1, ids[:, 1:, None]).squeeze(-1)
answer_logp = (token_logp * mask[:, 1:]).sum(-1)
```

论文的序列概率分解给出 $\log \pi(y | x) = \sum_t \log \pi(y_t | x, y_{<t})$ 。固定源码的 `compute_logprobs` 却在掩码求和后除以有效 token 数，得到 token 平均。这是长度归一化的实现变体，不能说成与原始序列 DPO 完全相同。源码：`compute_logprobs` 与 `compute_dpo_loss`

本书另从固定 Notebook 抽取原始 `compute_logprobs` 运行人工输入，确认返回 token 平均；全空 mask 实际产生 NaN。因此补充实现选择显式拒绝空回答，而不是照抄除零路径。

均匀三词词表中，每 token log 概率是 $-\log 3$ 。两个回答分别有 2 与 4 个有效 token，则求和为 -2.1972245773 、 -4.3944491547 ，平均都是 -1.0986122887 。长回答的序列概率较小并不自动意味着更差；改为平均消除了这一直接长度效应，也改变了优化对象。比较实验须明确使用哪一种，并保持策略与参考使用同一归约。

上游 `collate` 对提示长度额外加 2，意图排除两个换行 token。这依赖具体 tokenizer 与拼接边界；BPE 对拼接后的分词未必等于各段独立分词的简单拼接。更稳妥的方法检查完整编码上的前缀稳定性，明确分隔符是否计入回答，并打印被掩码后实际评分的 token。

27.5 参考模型、训练与日志

`compute_dpo_loss_batch` 做策略 chosen/rejected 两次前向，并在 `no_grad()` 中做参考两次前向；`compute_dpo_loss_loader` 汇总评估；`evaluate_dpo_loss_loader` 切换模式；`train_model_dpo_simple` 负责优化和定期生成。参考模型应固定参数并处于 eval 模式，避免 dropout 让同一参考回答的分数反复变化。策略评估也需 eval；训练后恢复 train 模式。

源码用于记录的 chosen/rejected rewards 是策略与参考 log 概率差并 detach，没有乘 β 。因此它们是诊断量，不能直接当成前面推导中带 β 的奖励。更重要的是观察 chosen 与 rejected 的差距、验证 loss 和独立任务质量。偏好 loss 下降不保证事实准确率提高：如果数据标签偏好冗长或讨好，模型也会认真学会这些特点。

把参考模型错误地一起更新，会让“相对固定基准”的定义失效；把参考概率 detach 仍不足以保证参考权重不变，还要确保它不被其他优化器或共享参数修改。小实验只验证本地张量图，不验证模型克隆、参数共享或完整训练的显存需求。

练习与答案

补充题：策略和参考相同， $\beta=0.1$ ，loss应是多少？答案： $\log 2 \approx 0.693147$ ，与每条回答自身概率无关。若不是，检查 dropout、mask、shift与权重是否一致。

补充题：chosen比rejected多两个 token，能否直接比较sum与mean实验的reward曲线？答案：不能。它们的量纲分别是每序列和每 token，对长度的响应不同，应分开标注。

来源

DPO 固定实现及训练循环；DPO 论文。数据 JSON 是上游已有偏好样本；本书未执行70B生成或任何Ollama请求。

第28章 LoRA: 冻结大矩阵, 学习一个可合并的增量

如果每个任务都更新一整套模型, 保存与切换任务都很昂贵。LoRA 把每个选中线性层的更新限制为两个小矩阵的乘积。它减少的是可训练更新的自由度, 并不让原模型在前向时消失。

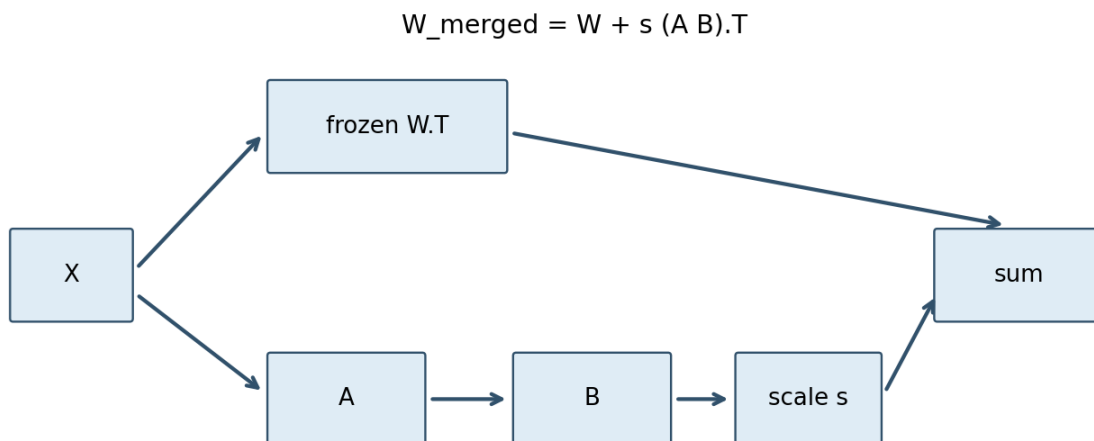
28.1 先统一矩阵朝向

PyTorch 线性层保存 $W \in \mathbb{R}^{d_{out} \times d_{in}}$, 输入 $X \in \mathbb{R}^{N \times d_{in}}$, 输出为 $XW^T + b$ 。这里 N 可以是 batch 与时间位置合并后的数量, 偏置 $b \in \mathbb{R}^{d_{out}}$ 。按上游代码的行向量约定, LoRA 参数 $A \in \mathbb{R}^{d_{in} \times r}$ 、 $B \in \mathbb{R}^{r \times d_{out}}$, 输出改为

$$Y = XW^T + b + sXAB.$$

r 为 rank, s 为缩放系数。XA 把输入投影到 r 维, 乘 B 再回到输出空间; 增量矩阵 AB 的秩不超过 r。学习 A、B 可以改变投影方向, 不是只保留 W 的某些固定列。[来源: 附录E实现](#)

普通矩阵有 $d_{in}d_{out}$ 个权重; LoRA 新增 $r(d_{in} + d_{out})$ 。只有 $r < \frac{d_{in}d_{out}}{d_{in}+d_{out}}$ 时, 这两个小矩阵的参数总数才更少。教学实验 $d_{in} = 3$ 、 $d_{out} = 4$ 、 $r=2$, 新增14个参数反而多于原矩阵12个, 故意展示“小模型上 LoRA 也可能不省参数”的边界。真实大方阵的 r 远小于宽度时, 才有明显优势。



28.2 缩放约定必须跟代码走

固定版本附录E与包接口使用 $s = \alpha/r$ ；ch06附加实验仍使用 $s = \alpha$ 。相同数值 α 不代表相同增量。若旧实现 $\alpha_{\text{old}} = 16$ ，rank=16，要在 α/r 实现保持同样倍数，需 $\alpha_{\text{new}} = 256$ 。这也是当前习题7.4解答明确处理的转换。

入口	有效倍数	rank=16、目标倍数16
ch06 additional_experiments	α	$\alpha=16$
appendix-E、pkg appendix_e	α/r	$\alpha=256$
ch07 exercise_experiments	α/r	$\alpha=256$

不要只把命令行里的rank改大，就认为是在“公平比较更高秩”。除了参数数目，初始化幅度、有效倍数、学习率都可能改变。 α/r 是约定，不保证任意rank下优化动力学完全一样。[LoRA 原论文](#)

28.3 零初始化其中一边，为什么还能学

上游用随机A、全零B。这样 $AB=0$ ，安装LoRA后的初始输出与原模型相同。设输出梯度为 $G = \partial L / \partial Y$ ，则

$$\frac{\partial L}{\partial B} = s A^\top X^\top G, \quad \frac{\partial L}{\partial A} = s X^\top G B^\top.$$

初始 $B=0$ ，所以A的梯度为0，但随机A通常让B获得非零梯度。第一次更新后B不再为0，A也能开始学习。若A、B同时为0，两边梯度都为0，这个分支会停在原地；不是把学习率调大就能解决。

我们的float64实验验证初始输出精确相同，A梯度总绝对值为0，B梯度大于0。这个等价检查不需要预训练权重，随机W就能证明实现结构是否正确。它只验证一次局部运算，不说明rank=2足以适应真实任务。

28.4 替换线性层与参数管理

LinearWithLoRA 保存原linear与新的lora，forward相加。replace_linear_with_lora 递归遍历子模块，遇到Linear就替换，否则继续递归。必须先冻结原模型，再加入新可训练分支，并在替换后建立优化器。如果先建优化器再替换，新参数不会自动进入旧参数组。

替换全部Linear包括注意力投影、前馈与输出头。也可以只选Q/V等部分，但这是另一个实验条件，不能把所有“LoRA”归为相同设置。模块名称、rank、 α 、基座哈希与目标层清单都应随adapter保存，恢复时先重建相同结构。若仅保存A/B，仍需获得对应基座权重；若保存整个state_dict，文件体积未必很小。

冻结W减少W的梯度与优化器状态, 但仍需执行 $XW^\top$; 为了训练插入在深层的adapter, 很多中间激活仍然要保存。LoRA并不自动解决长序列的注意力内存问题, 也不能保证速度按可训练参数比例提高。

28.5 合并与反向合并

将增量写回PyTorch的权重朝向, 得到 $W_{\text{merged}} = W + s(AB)^\top$ 。因此

$$XW_{\text{merged}}^\top + b = XW^\top + b + sXAB.$$

上游 LinearWithLoRAMerged 每次forward构造合并权重, 以演示代数等价; 部署时可在不需要继续训练的情况下合并一次。这里没有额外非线性、也没有分支dropout时等价最直接; 量化、低精度、运行中修改adapter会增加误差或同步问题。

```
merged_weight = weight + scale * (A @ B).T
merged_output = F.linear(x, merged_weight, bias)
```

本书实测分支与合并最大绝对误差 $1.6653345369377348 \times 10^{-16}$, 来自浮点运算顺序, 阈值设 $1e-12$ 。切勿在已经合并的权重上继续同时加LoRA分支, 那会把增量算两次。若需要来回切换, 保存不可变基座或明确合并状态, 不依赖反复加减浮点矩阵完全无损。

练习与答案

补充题: 1024×1024 矩阵、rank=8, 原参数与adapter各多少? 答案: 1,048,576与16,384, 约1/64; 这只是权重计数, 不是总显存比。

补充题: A和B都置零, 梯度为什么为零? 答案: 每个参数的梯度都乘另一个为零的矩阵。将其中一个随机初始化, 另一个零初始化, 才兼顾初始等价与可学习性。

来源

附录E; 包实现; LoRA 论文。附录的数据集和分类循环复用第24章, GPT骨干复用第6章, 权重加载复用第15章。

第29章 三种界面，同一条推理链路

把模型接到聊天框，并不会让预训练模型自动学会遵循指令。上游的三个UI目录分别演示文本续写、短信分类、指令回答。共同部分是加载、编码、推理和显示；差异在权重结构、输入格式和输出解释。先把这些差异写成接口约定，再考虑流式显示。

29.1 四个入口对应三类任务

ch05包含 `app_orig.py` 与 `app_own.py`，前者加载OpenAI GPT-2权重，后者加载第5章自行训练得到的 `model.pth`。两者都调用`generate`，把输入与续写一起解码。它们属于一种任务的两个权重来源，不能因为都是124M结构就交换`state_dict`：自训示例上下文为256、`qkv_bias=False`；原始权重配置上下文为1024、`qkv_bias=True`。源码：预训练UI

ch06的 `app.py` 重建二分类头，再加载 `review_classifier.pth`，调用 `classify_review`，输出一个标签。它不该调用语言模型`generate`，因为最后一维已经只有两个类别。ch07加载355M指令模型，先构造任务模板，生成后 `extract_response` 去掉提示和响应分隔符。分类UI；指令UI

路线	模型输出	用户输入处理	显示内容
预训练续写	$B \times T \times V$	原文本编码	输入加续写
分类	$B \times T \times 2$	截断补齐，取约定位置	类别标签
指令回答	$B \times T \times V$	加训练模板	新生成回答

这些应用在模块加载阶段调用 `get_model_and_tokenizer`，所以import本身可能触发文件读取甚至下载。自动测试不要把“导入UI”当成无副作用单元测试；本书没有运行这些入口，也没有启动Web服务。

29.2 加载成功不是推理准备完成

路径采用相对工作目录，启动位置不对便找不到权重。更稳妥的本地项目可从 `Path(__file__).resolve().parent` 推导路径，并允许显式配置。加载时先核对结构、词表、位置长度、输出头，再用受信文件的`state_dict`；`weights_only=True`减少非权重反序列化用途，但不能把任意来源文件视为可信。

固定源码中`app_orig`与分类UI显式 `eval()`；`app_own`与指令UI的加载函数没有同样显式调用。指令配置`dropout`为0，因此此处不一定改变输出；自训配置`dropout`为0.1时则需要特别检查。`no_grad()`不会关闭`dropout`，二者必须分开理解。这是静态审查结论，不是本书运行界面后的故障报告。

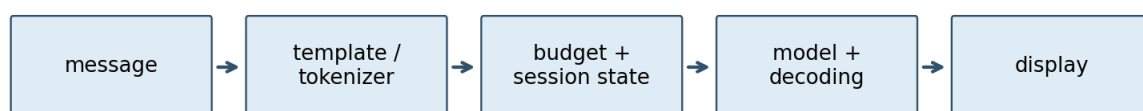
设备也需一致：模型、输入token都放到相同device；载入GPU保存的权重到CPU时考虑map_location。用户界面上的“正在生成”不能掩盖模型尚未成功加载，错误应区分文件缺失、shape不匹配、设备问题和推理预算不足。

29.3 上游是一次返回，流式是补充接口

@chainlit.on_message 回调接收一条消息，完整调用generate后.send()一次。它没有逐token streaming，也没有跨轮历史累积。界面长得像聊天，不代表下一轮会读上一轮内容。若要多轮对话，需要明确会话历史如何转换成训练匹配的模板，并按会话隔离缓存。

补充实现可以让生成器每次yield新token ID，前端把增量显示出来。但token不等于完整字符，尤其UTF-8和BPE边界处，逐token单独decode再拼接不一定等于整体decode。可靠方法积累token并使用支持增量解码的decoder，或只发送已经稳定的文本前缀。

UI is a contract around inference, not a new model capability



Read-only weights may be shared; mutable caches belong to each session

29.4 上下文预算属于接口，不应留给崩溃处理

设窗口上限C，编码后提示长度P，期望新增N。对于固定位置范围的缓存生成，通常应保证 $P+N \leq C$ ；超出时明确拒绝、缩短输出或截短历史。普通无缓存generate可以每步取最近C个输入窗口，但它会失去较早上下文，这与“保留完整对话的缓存”不是同一语义。第8章解释缓存位置与重置，本章要求UI把选择告诉调用者。

全局模型对象可以共享只读权重，却不能共享未经隔离的会话KV状态。会话A的末尾位置若流到B，B可能继承无关上下文。并发请求还要处理取消、异常和释放状态，不能只在成功结束时重置。

离线fixture刻意不调用神经模型：输入 [1,2] 按确定规则产生 [3,4,5]，另一会话输入 [7] 产生 [8,9]；超预算请求被拒绝。这个测试验证流式顺序、请求状态独立和预算判断，不证明Chainlit部署、网络吞吐或真实tokenizer解码正确。

练习与答案

补充题：为什么分类权重不能直接接续写UI？答案：分类头只有C=2个logit，输出ID不再是词表分布；即使代码能运行，也不具备语言生成含义。

补充题：no_grad()之后重复输入仍输出不同，可能是什么？答案：dropout仍处于train模式，或采样随机数/并发状态不同。先检查eval，再检查采样设置，不要立即归咎权重损坏。

来源

三个UI目录及其README、requirements-extra分别说明启动依赖与模型准备，固定版本见本章链接。UI静态检查、补充fixture和真实服务验收是三个层次，本书只完成前两者。

第30章 包接口与测试：怎样知道自己运行的是同一份模型

Notebook适合逐步观察，包适合稳定导入。两者共存时，最常见的错误是修改了一份代码，却从另一份代码导入同名函数。本章沿着包接口、测试契约和环境记录，建立可追溯的运行方式。

30.1 模块名相同，不代表实现细节永远相同

固定仓库的 `pkg/llms_from_scratch` 将章节代码整理为可导入模块。ch02提供滑窗数据集；ch03提供注意力的逐步版本和PyTorch优化版本；ch04提供基础GPT与Fast变体；ch05提供生成、训练、权重转换和下载；ch06是分类；ch07是指令数据和评估；appendix_a/d/e分别承接自动微分入门、训练改进和LoRA。包[README](#)

本书对重复算法使用明确回指：ch02→第3章；ch03→第4–5章；ch04→第6章；ch05→第13–15章；ch06→第24章；ch07→第25–26章；appendix_a→第2章；appendix_d→第17章；appendix_e→第28章。Fast变体的契约是保持输入输出意义同时更换注意力内核，并非“函数名相似就已证明等价”。

`llama3.py`、`qwen3.py` 提供独立模型、tokenizer及权重映射，结构解释见第19–20章；`kv_cache` 和 `kv_cache_batched` 提供有状态推理路径，见第8章。调用者要确认自己导入的是无缓存、单一缓存还是批量缓存版本，尤其是position状态、reset和batch中长度不同时的处理。

30.2 包额外承担的边界

`generate.trim_input_tensor` 为生成预留位置。固定实现的保留长度为 $\max(1, C - \max(0, N - 1))$ ，因为最后生成的token不需要再输入一次forward。所以 $N > 0$ 时可以保留 $C - N + 1$ 个提示token，让最后一次forward的输入位置仍在C内；最终返回序列可能比C多一个token。这个细节解释了为什么“输出总长度 $\leq C$ ”的保守UI规则与此辅助函数不是逐字相同的契约。 $N > C$ 会拒绝， $N = 0$ 只把过长提示截到C。源码

`utils.import_definitions_from_notebook` 提取import和函数/类定义，便于测试Notebook中的实现，不执行整份训练流程。仍需注意：import本身可有副作用，函数默认参数和类体也可能执行代码。因此它适用于已审阅、可信的源码，不能作为运行任意上传Notebook的安全沙箱。`download_file` 是外部数据边界，必须独立记录URL、目标路径、版本与校验，而不是把下载混在纯函数测试里。

30.3 测试应该说明它证明了什么

基础测试分别检查ToyDataset、滑窗、MHA形状、GPT变体、小型训练、分类、指令和LoRA。通过一个shape测试不能推出梯度正确；通过一次loss下降不能推出泛化。模型与库参考实现的对比还需统一权重、dtype、dropout、mask、tokenizer和容差。

缓存测试尤其值得逐条读：`test_kv_cache_gpt2.py` 覆盖显式cache是否激活、第二步是否继续、缓存/无缓存生成是否一致、重复生成是否重置位置、stream与普通生成是否一致、截短提示的位置和零生成预算。`test_kv_cache_context_limits.py` 进一步检查超长提示的错误、被拒绝decode后cache是否保持原状态、批量中每条样本是否都检查、预算大但提前EOS结束的路径。错误路径不破坏状态，和正常路径输出正确同样重要。

Qwen3测试区分稠密与MoE、RoPE、RMSNorm、缓存和批量缓存、特殊token round-trip、chat包装、多轮前缀稳定性及外部参考权重等价；Llama3测试覆盖RoPE、GQA、RMSNorm、快慢变体与参考权重。部分测试需要可选依赖或权重，不能把跳过误记成通过。本书这组微调实验没有运行这些全部上游测试，逐文件映射记录的是阅读与教学覆盖，不是执行覆盖。

30.4 环境检查与持续集成

`setup/02_installing-python-libraries/python_environment_check.py` 的 `get_packages`、`get_requirements_dict`、`check_packages` 分别读取已安装包、解析需求、比较版本；Notebook版展示同一检查。版本满足最低要求并不保证所有设备数值一致，因此运行日志还需Python、PyTorch、设备、dtype和线程数。

原生Python、uv、pixi与Docker是不同环境管理路线；选择一条并固定依赖，不要同时在系统Python和虚拟环境混装。VS Code扩展配置帮助编辑，不会代替Python依赖。SageMaker的CloudFormation模板属于可选云资源部署，不是运行本书CPU测试的必要条件；部署会涉及资源与费用，本书未创建任何云资源。[setup目录](#)

`.github/workflows` 对Linux、macOS、Windows、pip、uv、pixi、旧PyTorch、候选PyTorch和最新支持Python安排不同检查。它们体现兼容性目标，不证明任意组合都支持。`required-checks.yml` 根据改动文件类型决定要求哪些工作流，并核对当前PR的head与merge SHA；不能让旧提交的绿勾替新提交背书。`retry-required-checks.yml` 处理汇总检查的重试，但重试不会修复真实测试失败。

链接脚本 `check_links.py` 提取Markdown/HTML链接，区分本地路径与外部地址，配合自测验证解析和报错。双引号脚本 `check_double_quotes.py` 对Python和Notebook检查风格，还处理f-string内部字符串位置。拼写、PEP8与链接检查负责文档和代码卫生，不证明数学正确；数值断言、数据审计与人工审校仍需各自承担责任。

30.5 这份交付怎样复核

本组自包含实验是 `examples/finetuning/contracts.py`，没有外部模型下载。另有 `check_upstream_functions.py`，只从固定可信源码提取已审阅的少量函数，实测近重复空字段反例、DPO平均/空mask、附录E缩放和trim边界；它需要该固定源码目录，不执行Notebook的下载或训练单元。固定seed=20261014，CPU单线程，默认float64，实际环境Python3.12.14、PyTorch2.6.0+cpu。运行结果在 `qa/finetuning/contracts.json`；它包括正例和会失败的朴素写法，不把上游训练日志当作本书结果。

故障现象	先验证	不应直接做的事
embedding越界	token ID范围、词表扩展、-100是否进输入	调低学习率
loss为NaN	有效监督数、log稳定性、空batch	把NaN强改0
load_state_dict不匹配	头维度、bias、层数、上下文	随意strict=False隐藏问题
同输入结果变化	eval、采样、共享状态、随机数	认定权重已损坏
OOM	batch、序列、激活、优化器状态	只看可训练参数计数
import来自旧文件	module.file、环境、editable安装	反复改另一个Notebook

本书不自动安装项目全部依赖；使用者可在独立环境安装与记录的版本匹配的CPU依赖，再运行脚本。根目录requirements、pyproject、pixi和troubleshooting说明整体安装与排错，阅读时应区分“历史固定代码所需环境”与“当前在线最新版本”。

练习与答案

补充题：shape与数值等价测试各抓什么错？答案：前者抓接口维度、轴顺序等明显问题；后者在相同输入权重下抓转置、mask、缩放与状态差异。二者均不能单独证明任务质量。

补充题：一个需要下载权重的测试被skip，应写“全部通过”吗？答案：应写通过数量、跳过数量和跳过条件。本书只对实际运行的人工CPU断言声明通过。

来源

包源码、测试目录、CI目录、排错文档。

第31章 习题闭环与延伸：知道哪里已经验证，哪里还没有

这本书的终点不是把仓库文件都列一遍，而是让读者能从一个结果追到数据、计算、参数和验证条件。最后一章把公开习题解答与前面章节重新接起来，并说明独立推理仓库为什么不能被算作本仓库的完整实现覆盖。

31.1 原题、公开解答与本书补充题的区别

上游appendix-C只提供各章解答Notebook入口，appendix-B说明没有代码；它们不提供一份可据此逐字转载的完整出版书。这里概括公开解答所演示的操作，不复制原书题面或原图。本书每章“补充题”是为自己的解释设计的检查；公开解答中保存的大模型输出和耗时属于上游记录，未在本书环境重新测量。

公开解答入口	复盘任务	本书定位	可以检查的结果
ch02 2.1–2.2	BPE分词检查与滑窗长度/步幅比较	第3章	分词可回解、输入/目标相差一位
ch03 3.1	对齐两种注意力实现的权重方向	第4章	同权重输出一致
ch03 3.2	拼接多个头后的输出宽度	第5章	head数乘单头宽度
ch03 3.3	实例化GPT风格多头注意力	第5、7章	维度、参数计数
ch04 4.1	前馈与注意力参数比较	第7章	分模块计数与公式一致
ch04 4.2	更大GPT配置	第6–7章	头维整除、总参数增长
ch04 4.3	分离不同dropout位置的配置	第6章	train/eval行为
ch05 5.1–5.3	温度、top-k与确定性	第14章	分布与固定条件输出
ch05 5.4	继续预训练	第13、15、17章	状态恢复与后续loss
ch05 5.5–5.6	预训练权重损失与更大模型	第13、15章	相同数据和配置比较
ch06 6.1–6.3	长度、训练范围、读出位置	第24章	成本与可见信息范围
ch07 7.1–7.2	提示风格与指令掩码	第25章	模板一致、监督位置
ch07 7.3	更大指令数据集	第25–26章	缓存路径、数据边界
ch07 7.4	LoRA指令微调	第28章	参数范围、有效缩放

入口见[appendix-C](#)。这张表用于找回推理链，不代表本书运行了其中所有大型实验。

31.2 用一组题检查自己是否真的能修改代码

补充综合题A：某分类器训练时padding到120，推理时padding到1024，仍取最后一列。是否一定报错？不一定。若位置embedding支持1024，代码可以运行，但读出位置和训练分布已经变化。先恢复约定，再比较“最后真实token”的独立变体，不能把两项改动混在一起。

补充综合题B：答案专用SFT损失比全序列损失更低，可以直接说模型变好吗？不能。被评分的token集合变了；需要同一测试任务、相同生成条件和独立回答评价。将两条损失曲线画在同一轴上并不能消除目标差异。

补充综合题C：把DPO参考模型与策略初始化为同一份权重，初始loss不是log2，先查什么？查两者eval/dropout、掩码、token shift、归约方式和是否共享了会变化的状态。不要先调整 β 去“修正”初始loss。

补充综合题D：LoRA保存文件比预期大，是否说明实现失败？不一定。保存整个模型state_dict会包含冻结基座；若目标是adapter-only交付，应保存A/B与结构元数据，并指明基座版本。合并后的文件也仍是完整模型大小。

补充综合题E：界面每轮都能回答，是否实现了多轮会话？不一定。若回调只编码当前消息，没有历史模板或隔离缓存，模型只完成单轮请求。用第二轮明确依赖第一轮信息的人工fixture检查状态契约，再用真实模型另测语义能力。

这些答案都能落到一个具体检查，而不是用“模型不够大”解释任何失败。先证明输入、状态和目标正确，才值得投入更贵的训练。

31.3 独立推理仓库的边界

固定仓库通过 .gitmodules 将 reasoning-from-scratch 指向另一个项目。gitlink记录某个提交身份，不等于本书已经取得、审阅并解释该项目每个文件。因此本书只介绍主README链接的延伸主题，不把它们算作本仓库算法复现。[子模块声明](#)；[独立项目](#)

Qwen3提供可用于推理实验的模型与生成接口；本书第20章说明相关结构，但模型会生成推理式文本不自动证明推理正确。评测需要明确任务、答案提取和可核验评分。self-consistency通过多次采样后聚合答案，增加的是推理预算；如果所有样本共享相同误解，多数票也会一起错。self-refinement让模型依据反馈修订回答，反馈质量决定修订能否减少错误，不能把“改了一次”当作正确性证明。

GRPO属于基于一组候选相对表现的策略优化方向，与第27章离线偏好对目标不同。理解它需要额外的采样、奖励、组内归一化、概率比与训练稳定性知识。本书仅说明接口位置：提示进入采样器，候选进入评分器，评分回到优化目标；未提供该独立仓库完整代码，也未运行相关训练。继续学习时，应重新固定它自己的提交、许可证、数据和实验范围。

31.4 参考文献怎么用，而不只是放在书尾

实现细节以固定commit为准：alpha是否除rank、mask是否移位、cache是否重置，直接读对应函数。论文回答为何设计目标、假设是什么；框架文档回答函数的输入、归约与梯度契约。三者可能使用不同记号或实现变体，必须逐一对齐。

微调部分的主要阅读顺序是：先读ch06分类与ch07 SFT源码，亲手打印标签；再读LoRA论文并对应附录E的矩阵朝向；最后读DPO论文并对照源码的token平均变体。若只看公式而不看分词边界，可能训练错位置；若只跑代码而不看目标，可能把长度归一化当作无影响的实现细节。

本组参考入口：分类源码、指令与偏好源码、LoRA论文、DPO论文、PyTorch交叉熵。其他架构与训练主题的原论文、文档和固定源码已放在相应章节附近，避免一张长书单脱离具体论证。

31.5 什么叫完成这次复现

本书实际完成的微调验证是人工CPU算例：标签、梯度、LoRA等价、DPO数值、去重切分和UI状态。此外运行了少量固定上游函数的人工回归，不等于完整Notebook复现。没有下载预训练权重，没有进行短信或IMDb大型训练，没有调用OpenAI或Ollama，没有部署三个UI，也没有运行独立推理仓库。上述边界让“已通过”有明确含义。

逐文件、函数、Notebook单元、配置和练习对应位置见 `research/finetuning/source_coverage.json`；运行命令与版本见 `qa/finetuning/README.md`；源文件与交付哈希见 `research/finetuning/manifest.json`。映射是阅读审计的入口，正文才承担解释。碰到未运行的步骤，应记录条件与预期检查，而不是补写一段看起来合理的输出。

参考资料索引

以下按章节收录原始链接。每项主张与来源的对应关系、固定版本、实验条件和未验证范围，以正文为准。

第 01 章 先画清语言模型的任务与边界

- <https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch01/README.md>
- https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch05/01_main-chapter-code/gpt_train.py
- <https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch01/reading-recommendations.md>
- <https://github.com/rasbt/LLMs-from-scratch/tree/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch01>

第 02 章 张量、自动微分与最小训练闭环

- https://github.com/rasbt/LLMs-from-scratch/tree/cdbd33e6d57a71e20ea31434841095fac50cecd8/appendix-A/01_main-chapter-code
- https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/appendix-A/01_main-chapter-code/code-part1.ipynb
- https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/appendix-A/01_main-chapter-code/code-part2.ipynb
- https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/appendix-A/01_main-chapter-code/DDP-script-torchrun.py
- https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/appendix-A/01_main-chapter-code/exercise-solutions.ipynb

第 03 章 文本怎样变成 token：词表、BPE 与特殊符号

- <https://github.com/rasbt/LLMs-from-scratch/tree/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch02>
- https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch02/06_bonus_simple-tokenizer-v3/simple-tokenizer-v3.ipynb
- https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch02/02_bonus_bytepair-encoder/bpe_openai_gpt2.py
- https://github.com/rasbt/LLMs-from-scratch/tree/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch02/01_main-chapter-code
- https://github.com/rasbt/LLMs-from-scratch/tree/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch02/02_bonus_bytepair-encoder
- https://github.com/rasbt/LLMs-from-scratch/tree/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch02/03_bonus_embedding-vs-matmul
- https://github.com/rasbt/LLMs-from-scratch/tree/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch02/04_bonus_dataloader-intuition
- https://github.com/rasbt/LLMs-from-scratch/tree/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch02/05_bpe-from-scratch

第 04 章 从加权平均推导自注意力与因果掩码

- https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch03/01_main-chapter-code/ch03.ipynb
- https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch03/01_main-chapter-code/multihead-attention.ipynb
- https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch03/01_main-chapter-code/exercise-solutions.ipynb

第 05 章 多头注意力的实现选择与 buffer 生命周期

- https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch03/02_bonus_efficient-multihead-attention/mha-implementations.ipynb

- https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch03/03_understanding-buffers/understanding-buffers.ipynb

第 06 章 把注意力组装成 GPT：归一化、残差与输出头

- https://github.com/rasbt/LLMs-from-scratch/tree/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch04/01_main-chapter-code
- https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch04/01_main-chapter-code/ch04.ipynb
- https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch04/01_main-chapter-code/gpt.py
- https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch04/01_main-chapter-code/exercise-solutions.ipynb

第 07 章 参数、FLOPs、显存与吞吐怎样估算

- https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch04/02_performance-analysis/flops-analysis.ipynb
- https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch04/02_performance-analysis/requirements-extra.txt

第 08 章 KV cache：先证明等价，再讨论加速

- https://github.com/rasbt/LLMs-from-scratch/tree/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch04/03_kv-cache
- https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch04/03_kv-cache/tests.py

第 09 章 GQA、MLA 与跨层 KV 共享

- https://github.com/rasbt/LLMs-from-scratch/tree/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch04/04_gqa

- https://github.com/rasbt/LLMs-from-scratch/tree/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch04/05_mla
- https://github.com/rasbt/LLMs-from-scratch/tree/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch04/10_kv-sharing
- <https://arxiv.org/abs/2305.13245>
- <https://arxiv.org/abs/2405.04434>
- <https://arxiv.org/abs/2405.12981>
- <https://arxiv.org/abs/2305.13245>。
- <https://arxiv.org/abs/2405.04434>。
- <https://arxiv.org/abs/2405.12981>。

第 10 章 局部与稀疏注意力：SWA 和 DSA

- https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch04/06_swa/gpt_with_kv_swa.py
- <https://arxiv.org/abs/2004.05150>
- <https://arxiv.org/abs/2503.19786>
- https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch04/09_dsa/gpt_with_kv_dsa.py
- <https://huggingface.co/deepseek-ai/DeepSeek-V3.2/resolve/main/assets/paper.pdf>
- <https://arxiv.org/abs/2004.05150>。
- <https://arxiv.org/abs/2503.19786>。

第 11 章 Gated DeltaNet：从矩阵状态理解递推

- https://github.com/rasbt/LLMs-from-scratch/tree/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch04/08_deltanet
- <https://arxiv.org/abs/2412.06464>
- <https://arxiv.org/abs/2412.06464>。

第 12 章 MoE：路由、专家与负载

- https://github.com/rasbt/LLMs-from-scratch/tree/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch04/07_moe
- <https://arxiv.org/abs/2101.03961>
- <https://arxiv.org/abs/2002.05202>。

- <https://arxiv.org/abs/2101.03961>。

第 13 章 预训练目标、交叉熵与验证集

- https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch05/01_main-chapter-code/gpt_train.py
- https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch05/01_main-chapter-code/ch05.ipynb

第 14 章 采样、温度、top-k 与生成边界

- https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch05/01_main-chapter-code/gpt_generate.py
- https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch05/01_main-chapter-code/ch05.ipynb
- https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch05/01_main-chapter-code/exercise-solutions.ipynb

第 15 章 权重格式、参数映射与低内存加载

- https://github.com/rasbt/LLMs-from-scratch/tree/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch05/02_alternative_weight_loading
- https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch05/01_main-chapter-code/gpt_generate.py
- https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch05/01_main-chapter-code/gpt_download.py
- https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch05/08_memory_efficient_weight_loading/memory-efficient-state-dict.ipynb

第 16 章 语料整理与 Gutenberg 训练工程

- https://github.com/rasbt/LLMs-from-scratch/tree/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch05/03_bonus_pretraining_on_gutenberg
- https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch05/03_bonus_pretraining_on_gutenberg/prepare_dataset.py
- https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch05/03_bonus_pretraining_on_gutenberg/pretraining_simple.py
- https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch05/03_bonus_pretraining_on_gutenberg/tests.py

第 17 章 学习率、梯度裁剪、超参数与 Muon

- https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/appendix-D/01_main-chapter-code/appendix-D.ipynb
- https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch05/05_bonus_hparam_tuning/hparam_search.py
- <https://arxiv.org/abs/2502.16982>
- <https://docs.pytorch.org/docs/2.14/generated/torch.optim.Muon.html>
- https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch05/18_muon/gpt_train_muon.py
- https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch05/04_learning_rate_schedulers/README.md
- https://github.com/rasbt/LLMs-from-scratch/tree/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch05/18_muon

第 18 章 训练性能及 CPU、MPS 数值差异

- https://github.com/rasbt/LLMs-from-scratch/tree/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch05/10_llm-training-speed

- https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch05/19_cpu_mps_differences/1_cpu_mps_random_generator.ipynb

第 19 章 从 GPT 到 Llama: RoPE、RMSNorm 与门控前馈

- https://github.com/rasbt/LLMs-from-scratch/tree/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch05/07_gpt_to_llama
- <https://arxiv.org/abs/1910.07467>
- <https://arxiv.org/abs/2002.05202>
- <https://arxiv.org/abs/2104.09864>
- <https://arxiv.org/abs/1910.07467>; GLU
- <https://arxiv.org/abs/2002.05202>。
- <https://arxiv.org/abs/2104.09864>。
- <https://arxiv.org/abs/2307.09288>; Llama
- <https://arxiv.org/abs/2407.21783>。

第 20 章 Qwen3 与 Qwen3.5: 同一个名字下, 两种记忆接口

- https://github.com/rasbt/LLMs-from-scratch/tree/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch05/11_qwen3
- https://github.com/rasbt/LLMs-from-scratch/tree/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch05/16_qwen3.5
- <https://arxiv.org/abs/2505.09388>
- <https://arxiv.org/abs/2412.06464>。
- <https://arxiv.org/abs/2505.09388>。
- <https://huggingface.co/Qwen/Qwen3.5-0.8B>。

第 21 章 Gemma 3、Gemma 4: 局部层、共享层与数值约定

- https://github.com/rasbt/LLMs-from-scratch/tree/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch05/12_gemma3
- https://github.com/rasbt/LLMs-from-scratch/tree/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch05/17_gemma4
- <https://arxiv.org/abs/2503.19786>
- <https://arxiv.org/abs/2503.19786>。

- <https://huggingface.co/google/gemma-4-E2B>。
- <https://huggingface.co/google/gemma-4-E4B>。

第 22 章 OLMo 3、Tiny Aya 与可替换模型接口

- https://github.com/rasbt/LLMs-from-scratch/tree/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch05/13_olmo3
- https://github.com/rasbt/LLMs-from-scratch/tree/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch05/15_tiny-aya
- https://github.com/rasbt/LLMs-from-scratch/tree/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch05/14_ch05_with_other_llms
- <https://arxiv.org/abs/2309.00071>
- <https://arxiv.org/abs/2309.00071>。
- <https://huggingface.co/CohereLabs/tiny-aya-global>。
- <https://huggingface.co/allenai/Olmo-3-7B-Instruct>。

第 23 章 扩展分词器：新 ID 还需要新的模型行

- https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch05/09_extending-tokenizers/extend-tiktoken.ipynb
- <https://github.com/openai/tiktoken>
- <https://github.com/openai/tiktoken>。

第24章 文本分类：一个输出位置，决定模型看见多少内容

- https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch06/01_main-chapter-code/gpt_class_finetune.py
- https://github.com/rasbt/LLMs-from-scratch/tree/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch06/03_bonus_imdb-classification
- https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch06/01_main-chapter-code/ch06.ipynb
- https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch06/02_bonus_additional-experiments/additional_experiments.py

- <https://docs.pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html>

第25章 指令微调：先把“该预测什么”排对

- https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch07/01_main-chapter-code/gpt_instruction_finetuning.py
- https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch07/01_main-chapter-code/ch07.ipynb
- https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch07/01_main-chapter-code/exercise_experiments.py
- https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch07/01_main-chapter-code/exercise-solutions.ipynb

第26章 指令数据：先防止自己给自己打高分

- https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch07/02_dataset-utilities/find-near-duplicates.py
- https://github.com/rasbt/LLMs-from-scratch/tree/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch07/02_dataset-utilities
- https://github.com/rasbt/LLMs-from-scratch/tree/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch07/03_model-evaluation
- https://github.com/rasbt/LLMs-from-scratch/tree/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch07/05_dataset-generation

第27章 DPO：比较两个回答，也要说明比较的基准

- https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch07/04_preference-tuning-with-dpo/create-preference-data-ollama.ipynb
- <https://arxiv.org/abs/2305.18290>
- https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch07/04_preference-tuning-with-dpo/dpo-from-scratch.ipynb

第28章 LoRA：冻结大矩阵，学习一个可合并的增量

- https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/appendix-E/01_main-chapter-code/appendix-E.ipynb
- <https://arxiv.org/abs/2106.09685>
- https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/pkg/llms_from_scratch/appendix_e.py

第29章 三种界面，同一条推理链路

- https://github.com/rasbt/LLMs-from-scratch/tree/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch05/06_user_interface
- https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch06/04_user_interface/app.py
- https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch07/06_user_interface/app.py

第30章 包接口与测试：怎样知道自己运行的是同一份模型

- https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/pkg/llms_from_scratch/README.md
- https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/pkg/llms_from_scratch/generate.py
- <https://github.com/rasbt/LLMs-from-scratch/tree/cdbd33e6d57a71e20ea31434841095fac50cecd8/setup>
- https://github.com/rasbt/LLMs-from-scratch/tree/cdbd33e6d57a71e20ea31434841095fac50cecd8/pkg/llms_from_scratch
- https://github.com/rasbt/LLMs-from-scratch/tree/cdbd33e6d57a71e20ea31434841095fac50cecd8/pkg/llms_from_scratch/tests
- <https://github.com/rasbt/LLMs-from-scratch/tree/cdbd33e6d57a71e20ea31434841095fac50cecd8/.github>
- <https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/troubleshooting.md>

第31章 习题闭环与延伸：知道哪里已经验证，哪里还没有

- <https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/appendix-C/README.md>
- <https://github.com/rasbt/LLMs-from-scratch/blob/cdbd33e6d57a71e20ea31434841095fac50cecd8/.gitmodules>
- <https://github.com/rasbt/reasoning-from-scratch>
- <https://github.com/rasbt/LLMs-from-scratch/tree/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch06>
- <https://github.com/rasbt/LLMs-from-scratch/tree/cdbd33e6d57a71e20ea31434841095fac50cecd8/ch07>
- <https://arxiv.org/abs/2106.09685>
- <https://arxiv.org/abs/2305.18290>
- <https://docs.pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html>

完整实验代码

本附录按原始文件路径收录本书程序，与代码包中的同名文件逐字一致。请先按各章说明安装依赖、准备数据，再在项目根目录运行。数据文件、词表缓存与运行输出不在此处展开；其来源、获取方法或配套位置见说明。

examples/architectures/architecture_checks.py

```

"""Original CPU algebra checks, not a pretrained model or performance benchmark.
Source concepts: rasbt/LLMs-from-scratch pinned cdbd33e6; see book references.
Run: python examples/architectures/architecture_checks.py
"""

import json, math, platform
import torch
import torch.nn.functional as F

torch.manual_seed(1949)
torch.set_num_threads(1)
torch.set_default_dtype(torch.float64)
RESULTS = {}

def check(name, a, b, atol=1e-10):
    error = float((a-b).abs().max())
    assert error < atol, (name, error)
    RESULTS[name] = {"max_abs_error": error, "pass": True}

def attention(q,k,v,allowed):
    p=(q@k.transpose(-1,-2)/math.sqrt(q.shape[-1])).masked_fill(~allowed,-
torch.inf).softmax(-1)
    return p@v

def grouped_heads():
    q=torch.randn(1,4,5,3); k=torch.randn(1,2,5,3); v=torch.randn_like(k)
    mask=torch.ones(5,5,dtype=torch.bool).tril()
    expanded=attention(q,k.repeat_interleave(2,1),v.repeat_interleave(2,1),mask)
    explicit=torch.cat([attention(q[:,h:h+1],k[:,h//2:h//2+1],v[:,h//2:h//2+1],mask) for
h in range(4)],1)
    check('gqa_grouped_equals_expanded',expanded,explicit)
    wrong=attention(q,k.repeat(1,2,1,1),v.repeat(1,2,1,1),mask)
    RESULTS['gqa_wrong_repeat_detected']=float((expanded-wrong).abs().max())>0.1

def latent_and_sharing():
    x=torch.randn(5,6); down=torch.randn(6,2); uk=torch.randn(2,6); uv=torch.randn(2,6)
    c=x@down
    check('mla_factorization',c@uk,x@(down@uk))
    q=torch.randn(5,6); allowed=torch.ones(5,5,dtype=torch.bool).tril()
    a=attention(q,c@uk,c@uv,allowed)
    latent_score=q@uk.T@c.T/math.sqrt(6)
    b=latent_score.masked_fill(~allowed,-torch.inf).softmax(-1)@c@uv
    check('mla_absorbed_projection',a,b)

```

```

shared=(c@uk,c@uv); consumers=[shared,shared,shared]
assert len({z[0].data_ptr() for z in consumers})==1
check('sharing_same_storage_same_result',attention(q,*consumers[2],allowed),a)
RESULTS['sharing_queries_can_differ']=not
torch.allclose(attention(q+1,*shared,allowed),a)

def masks_and_sparse():
    n=7; q=torch.randn(n,4); k=torch.randn(n,4); v=torch.randn(n,4)
    positions=torch.arange(n); diff=positions[:,None]-positions[None,:]
    allowed=(diff>=0)&(diff<3)
    full=attention(q,k,v,allowed)
    chunks=[]
    for start,end in [(0,2),(2,5),(5,7)]:
        left=max(0,start-2)
    chunks.append(attention(q[start:end],k[left:end],v[left:end],allowed[start:end,left:end]
    ))
    check('swa_chunked_equals_full',full,torch.cat(chunks))
    vbad=v.clone(); vbad[4:]+=100
    check('swa_future_does_not_leak',full[:4],attention(q,k,vbad,allowed)[:4])
    scores=torch.randn(n,n); causal=diff>=0
    selected=scores.masked_fill(~causal,-torch.inf).topk(3,-1).indices
    sparse=torch.zeros_like(causal).scatter(1,selected,True)&causal
    assert sparse.any(-1).all() and not (sparse&~causal).any()
    out=attention(q,k,v,sparse)
    gathered=[]
    for t in range(n):
        idx=sparse[t].nonzero().flatten()
    gathered.append(attention(q[t:t+1],k[idx],v[idx],torch.ones(1,len(idx),dtype=torch.bool)
    ))
    check('dsa_dense_mask_equals_gather',out,torch.cat(gathered))
    # All-false row is a real counterexample, not an expected successful use.
    RESULTS['all_masked_softmax_is_nan']=bool(torch.isnan(torch.full((1,3),-
    torch.inf).softmax(-1)).all())

def delta():
    n,d=6,3; q=torch.randn(n,d); k=F.normalize(torch.randn(n,d),dim=-1);
    v=torch.randn(n,d)
    alpha=torch.sigmoid(torch.randn(n)); beta=torch.sigmoid(torch.randn(n))
    state=torch.zeros(d,d); outputs=[]; matrices=[]; writes=[]
    for t in range(n):
        state=alpha[t]*state
        state=state+torch.outer(k[t],beta[t]*(v[t]-k[t]@state))
        outputs.append(q[t]@state)
        matrices.append(alpha[t]*(torch.eye(d)-beta[t]*torch.outer(k[t],k[t])))
        writes.append(beta[t]*torch.outer(k[t],v[t]))
    naive=[]
    for t in range(n):
        total=torch.zeros(d,d)
        for i in range(t+1):
            contribution=writes[i]
            for j in range(i+1,t+1): contribution=matrices[j]@contribution
            total+=contribution
        naive.append(q[t]@total)
    check('delta_recurrence_equals_expansion',torch.stack(outputs),torch.stack(naive))

```

```

    unit=torch.tensor([1.,0.,0.]); target=torch.tensor([2.,3.,4.]);
    state=torch.randn(d,d)
    state+=torch.outer(unit,target-unit@state)
    check('delta_beta_one_overwrites_key',unit@state,target)

def rope(x, positions, theta=10000.):
    d=x.shape[-1]; frequencies=theta*(-torch.arange(0,d,2)/d)
    a=positions[:,None]*frequencies; first,last=x.chunk(2,-1)
    return torch.cat([first*a.cos()-last*a.sin(),first*a.sin()+last*a.cos()],-1)

def positions_and_norms():
    x=torch.randn(7,6); whole=rope(x,torch.arange(7))
    chunk=torch.cat([rope(x[:3],torch.arange(3)),rope(x[3:],torch.arange(3,7))])
    check('rope_cache_offset',whole,chunk)
    check('rope_preserves_norm',whole.square().sum(-1),x.square().sum(-1))
    bad=torch.cat([rope(x[:3],torch.arange(3)),rope(x[3:],torch.arange(4))])
    RESULTS['rope_reset_offset_detected']=not torch.allclose(whole,bad)
    a=torch.randn(1,6); b=torch.randn(1,6)
    check('rope_relative_position',
    (rope(a,torch.tensor([2]))*rope(b,torch.tensor([5]))).sum(),
    (rope(a,torch.tensor([0]))*rope(b,torch.tensor([3]))).sum())
    z=torch.tensor([2.,2.,2.])
    rms=z/(z.square().mean(-1,keepdim=True)+1e-6).sqrt()
    ln=F.layer_norm(z,(3,))
    RESULTS['rms_is_not_layernorm']=not torch.allclose(rms,ln)

def experts():
    x=torch.randn(5,4); gate=torch.randn(4,3); weights=torch.randn(3,4,4)
    score=x@gate; values,ids=score.topk(2,-1); probs=values.softmax(-1)
    output=torch.zeros_like(x); assignments=0
    for expert in range(3):
        token,slot=(ids==expert).nonzero(as_tuple=True)
        output.index_add_(0,token,(x[token@weights[expert]]*probs[token,slot,None]));
    assignments+=len(token)
    explicit=torch.stack([sum(probs[t,s]*(x[t@weights[ids[t,s]])) for s in range(2)) for
    t in range(5)])
    check('moe_dispatch_equals_token_loop',output,explicit)
    assert assignments==10
    RESULTS['moe_no_token_assignments_lost']=assignments
    # Capacity=1 would drop assignments without an explicit overflow policy.
    RESULTS['moe_capacity_one_overflow']=sum(max(0,int((ids==e).sum())-1) for e in
    range(3))

def vocabulary():
    old=torch.nn.Embedding(7,4); head=torch.nn.Linear(4,7,bias=False)
    new=torch.nn.Embedding(9,4); extended=torch.nn.Linear(4,9,bias=False)
    with torch.no_grad(): new.weight[:7].copy_(old.weight);
    extended.weight[:7].copy_(head.weight)
    check('vocab_old_embeddings_preserved',new(torch.arange(7)),old(torch.arange(7)))
    x=torch.randn(2,4); check('vocab_old_logits_preserved',extended(x)[:,:7],head(x))
    RESULTS['vocab_softmax_changes']=not torch.allclose(extended(x).softmax(-1)
    [:,:7],head(x).softmax(-1))
    extended.weight=new.weight
    assert extended.weight is new.weight and extended(x).shape[-1]==9
    RESULTS['vocab_weight_tying_identity']=True

```

```

if __name__=='__main__':
    for f in
    (grouped_heads,latent_and_sharing,masks_and_sparse,delta,positions_and_norms,experts,voc
    abulary): f()

print(json.dumps({'python':platform.python_version(),'torch':torch.__version__,'seed':19
49,'threads':torch.get_num_threads(),'dtype':str(torch.get_default_dtype()),'results':RE
SULTS},indent=2))

```

examples/architectures/cache_checks.py

```

"""Original audited-cell cache tests, including a recorded upstream counterexample."""
import json, pathlib, sys
sys.dont_write_bytecode=True
import torch
import torch.nn as nn
ROOT=pathlib.Path(__file__).resolve().parents[2]; UP=ROOT/'runtime_sources/upstream'
sys.path.insert(0, str(ROOT / 'examples'))
from verify_runtime import verify_runtime
verify_runtime(ROOT / 'runtime_sources')
torch.set_num_threads(1);torch.manual_seed(1949)
def load(path,indices):
    ns={'torch':torch,'nn':nn}; nb=json.loads((UP/path).read_text())
    for i in indices:
        source=''.join(nb['cells'][i]['source'])
        assert not any(s in source for s in ['hf_hub_download(', 'snapshot_download(',
        'login(', 'load_file('))
        exec(compile(source,f'{path}:cell{i}','exec'),ns)
    return ns
base=dict(vocab_size=31,emb_dim=16,hidden_dim=24,n_layers=2,n_heads=4,head_dim=4,n_kv_gr
oups=2,n_kv_heads=2,context_length=12,rope_base=10000.,dtype=torch.float32,qk_norm=True,
query_pre_attn_scalar=4,layer_types=['sliding_attention','full_attention'],rope_local_ba
se=10000.,sliding_window=3,rms_norm_eps=1e-6,attention_bias=False,rope_attention_factor=
1.,rope_type='default',rope_factor=1.,rope_orig_max=16,layer_norm_eps=1e-5,logit_scale=1
.,tie_word_embeddings=True,num_experts=3,num_experts_per_tok=2,moe_hidden_dim=12)
results={}; ids=torch.tensor([[1,3,5,7,9,11]])
for name,p,inds,cls in [
    ('qwen3','11_qwen3/standalone-qwen3-plus-
kvcache.ipynb',list(range(8,15)),'Qwen3Model'),
    ('qwen3_moe','11_qwen3/standalone-qwen3-moe-plus-
kvcache.ipynb',list(range(6,13)),'Qwen3Model'),
    ('gemma3','12_gemma3/standalone-gemma3-plus-kvcache.ipynb',list(range(8,14))
+[25],'Gemma3Model'),
    ('olmo3','13_olmo3/standalone-olmo3-plus-kv-
cache.ipynb',list(range(9,16)),'Olmo3Model'),
    ('tiny_aya','15_tiny_aya/standalone-tiny_aya-plus-kv-
cache.ipynb',list(range(9,15)),'TinyAyaModel')]:
    ns=load('ch05/'+p,inds);model=ns[cls](base).eval()
    with torch.no_grad():
        ref=model(ids); cache=ns['KVCache'](2);model.reset_kv_cache()

chunks=[model(ids[:, :2],cache=cache),model(ids[:, 2:5],cache=cache),model(ids[:, 5:],cache
=cache)]

```

```

error=float((ref-torch.cat(chunks,1)).abs().max());assert error<1e-5,(name,error)
results[name]={'chunk_error':error,'cache_lengths':[x[0].shape[2] for x in
cache.get_all()]})
if name.startswith('qwen3'):
    model.cfg['context_length']=6;model.reset_kv_cache();cache.reset()
    with torch.no_grad(): model(ids[:, :5],cache=cache)
    try:
        with torch.no_grad(): model(ids[:, -2:],cache=cache)
    except ValueError: pass
    else: raise AssertionError('Missing overrun rejection')
    assert model.current_pos==5 and cache.get(0)[0].shape[2]==5
    results[name]['overrun_preserves_cache']=True
ns=load('ch05/17_gemma4/standalone-gemma4-plus-kvcache.ipynb',list(range(9,14))+[15,19])
cfg=ns['get_gemma4_dense_config']
('E2B',torch.float32);cfg.update(vocab_size=31,vocab_size_per_layer_input=31,emb_dim=16,
hidden_dim=24,n_layers=4,n_heads=4,head_dim=4,global_head_dim=8,n_kv_heads=2,context_length=12,sliding_window=3,layer_types=['sliding_attention','full_attention']*2,hidden_size_per_layer_input=4,num_kv_shared_layers=2)
model=ns['Gemma4DenseModel'](cfg).eval()
plain=load('ch05/17_gemma4/standalone-gemma4.ipynb',list(range(9,14))+[15]);refmodel=plain['Gemma4DenseModel']
(cfg).eval();refmodel.load_state_dict(model.state_dict())
with torch.no_grad():
    reference=refmodel(ids,reuse_shared_kv=True)
    cache=ns['KVCache'](4);actual=model(ids,cache=cache)
error=float((reference-actual).abs().max())
results['gemma4_shared_prefill_counterexample']={'max_abs_error':error,'prompt_length':6,
'window':3,'description':'Producer local cache is truncated before later consumer uses it; early prompt queries no longer see same keys. Recorded failure, not a passing equivalence test.'}
assert error>1e-4
# Qwen3.5 helper has only torch imports, definitions, and a logger at module scope.
import importlib.util, contextlib, io
helper_path=UP/'ch05/16_qwen3.5/qwen3_5_transformers.py'
spec=importlib.util.spec_from_file_location('qwen3_5_transformers',helper_path)
helper=importlib.util.module_from_spec(spec);spec.loader.exec_module(helper)
sys.modules['qwen3_5_transformers']=helper
qns=load('ch05/16_qwen3.5/qwen3.5-plus-kv-cache.ipynb',list(range(10,16)))
qc=base.copy();qc.update(layer_types=['linear_attention','full_attention'],partial_rotary_factor=0.5,linear_num_value_heads=2,linear_num_key_heads=2,linear_key_head_dim=4,linear_value_head_dim=4,linear_conv_kernel_dim=3)
with contextlib.redirect_stdout(io.StringIO()): qm=qns['Qwen3_5Model'](qc).eval()
with torch.no_grad():
    full=qm(ids);cache=qns['KVCache'](2);qm.reset_kv_cache()
    outputs=[qm(ids[:, :3],cache=cache)]+[qm(ids[:, i:i+1],cache=cache) for i in range(3,6)]
    good_error=float((full-torch.cat(outputs,1)).abs().max())
    qm.reset_kv_cache();cache.reset()
    bad=[qm(ids[:, :3],cache=cache),qm(ids[:, 3:5],cache=cache),qm(ids[:, 5:],cache=cache)]
    bad_error=float((full-torch.cat(bad,1)).abs().max())
assert good_error<1e-5 and bad_error>1e-4
results['qwen35_model_cache']={'prefill_then_single_token_error':good_error,'multi_token_append_counterexample_error':bad_error,'counterexample_is_failure':True}
import tiktoken
enc=tiktoken.Encoding(name='offline_bytes',pat_str=r'(?
s).',mergeable_ranks={bytes([i]):i for i in range(256)},special_tokens={ '<|new|>':256})

```

```

assert enc.encode('a<|new|>b',allowed_special={'<|new|>'})==[97,256,98]
assert enc.decode([97,256,98])=='a<|new|>b'
try:enc.encode('<|new|>')
except ValueError: rejected=True
else:rejected=False
assert rejected
results['offline_tiktoken_extension']={'encoded':
[97,256,98], 'disallowed_special_rejected':True, 'version':tiktoken.__version__}
print(json.dumps({'torch':torch.__version__, 'seed':1949, 'results':results},indent=2))

```

examples/architectures/draw_figures.py

```

"""Original diagrams; does not use or reproduce upstream artwork."""
from pathlib import Path
import textwrap
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
from matplotlib.patches import FancyBboxPatch
import numpy as np
OUT=Path(__file__).resolve().parents[2]/'figures/architectures';
OUT.mkdir(parents=True,exist_ok=True)
plt.rcParams.update({'font.family':'Noto Sans CJK
JP', 'font.size':11, 'svg.fonttype':'path', 'axes.unicode_minus':False})
def save(fig,name):
    fig.savefig(OUT/f'{name}.svg',bbox_inches='tight'); fig.savefig(OUT/
f'{name}.png',dpi=160,bbox_inches='tight'); plt.close(fig)
def flow(name,title,rows):
    fig,ax=plt.subplots(figsize=(9,max(2.4,len(rows)*1.4)));
    ax.set(xlim=(0,10),ylim=(0,len(rows)*1.7)); ax.axis('off'); ax.set_title(title,pad=15)
    for r,(label,nodes) in enumerate(rows):
        y=(len(rows)-r-.5)*1.7; ax.text(0,y,label,ha='left',va='center',weight='bold')
        start=2.1; w=7.4/len(nodes)
        for j,txt in enumerate(nodes):
            x=start+j*w
    ax.add_patch(FancyBboxPatch((x,y-.47),w-.25,.94,boxstyle='round,pad=0.05',facecolor=[ '#d
cecf4', '#dff0e3', '#fff0d3' ][r%3],edgecolor='#34515e'))
    txt='\n'.join('\n'.join(textwrap.wrap(line,width=10 if len(nodes)>3 else
14,break_long_words=True)) for line in txt.split('\n'))
    ax.text(x+(w-.25)/2,y,txt,ha='center',va='center',fontsize=10)
    if j<len(nodes)-1: ax.annotate('',(x+w-.02,y),
(x+w-.25,y),arrowprops={'arrowstyle':'->','color':'#34515e'})
    save(fig,name)
flow('cache_axes','KV 可以沿不同方向共享',[
('GQA', ['4 个 query 头', '2 个 KV 头\nA,B,B', '缓存宽度缩小']),
('MLA', ['输入 x', '低维 c = x W_D', '需要时展开 K,V']),
('跨层共享', ['生产层: 写 K,V', '消费层: 只读 KV', '各层仍有自己的 Q'])])
fig,axs=plt.subplots(1,3,figsize=(9,3)); n=8; i,j=np.indices((n,n)); causal=j<=i
for ax,title,m in zip(axs,['因果全局','窗口 W=3','示意 top-k=3'],[causal,causal&(i-
j<3),causal&((j==0)|(j==i)|(j==np.maximum(0,i-2)))]):
    ax.imshow(m,cmap='Blues',vmin=0,vmax=1);ax.set_title(title);ax.set_xlabel('key 位

```

```
置');ax.set_ylabel('query 位置');ax.set_xticks(range(n));ax.set_yticks(range(n))
fig.tight_layout();save(fig,'attention_masks')
flow('delta_state','每一步先修正记忆,再读取',[
    ('状态更新',[ '旧状态 S', '遗忘  $\alpha S$ ', '预测  $k^T(\alpha S)$ ', '误差写入  $\beta k(v - \text{预测})^T$ ']),
    ('读出',[ '新状态 S', 'query 读出  $q^T S$ ', '归一化与输出门', '投回残差宽度']]))
flow('moe_dispatch','5 个 token、top-2: 应有 10 次专家分派',[
    ('选择',[ 'NxD 输入', 'router: NxE', 'top-k ID / 权重']),
    ('分派',[ '按专家收集 token', '各专家执行 FFN', '乘回各自路由概率']),
    ('合并',[ '原 token 索引', 'index_add 累加', 'NxD 输出; 不丢 token']]))
flow('rope_positions','缓存槽号不等于 token 的绝对位置',[
    ('一次计算',[ '位置 0,1,2', '位置 3,4,5', '同一套频率与配对']),
    ('分块计算',[ '第一块 offset=0', '第二块 offset=3', '与整段旋转一致']),
    ('错误路径',[ '第一块 offset=0', '第二块又 offset=0', '相对位置被改写']]))
flow('hybrid_cache','Qwen3.5 混合层: 请求重置必须覆盖两套状态',[
    ('全局层',[ '新 Q / K / V', 'RoPE 绝对位置', '逐层 KV 序列']),
    ('线性层',[ '卷积近期状态', '矩阵递推状态', '固定大小; 无历史列表']),
    ('请求边界',[ '清空 KV 容器', '清空卷积 / 矩阵', '位置计数归零']]))
flow('gemma_shared_inputs','Gemma 4 的两条额外数据通路',[
    ('KV 共享',[ '局部生产层 / 全局生产层', '同类型消费者', '只计算新 query']),
    ('逐层输入',[ 'token 逐层 embedding', '主 embedding 投影混合', '取当前 layer 的切片']),
    ('残差更新',[ '当前隐藏状态产生门', '门  $\times$  逐层输入', '投回 D 并加残差']]))
flow('residual_variants','同样的模块名, 数据依赖可以不同',[
    ('Llama',[ 'N(x)  $\rightarrow$  Attention', '加残差得 u', 'N(u)  $\rightarrow$  FFN  $\rightarrow$  加残差']),
    ('OLMo',[ 'Attention(x)  $\rightarrow$  Norm', '加残差得 u', 'FFN(u)  $\rightarrow$  Norm  $\rightarrow$  加残差']),
    ('Tiny Aya',[ '共同 h = Norm(x)', 'Attention(h) 与 FFN(h)', 'x + 两个并行分支']]))
flow('vocab_extension','新增 token 需要同时能被读取和预测',[
    ('输入表',[ '原 VxD: 逐行保留', '新增 mxD 行', '可读取 ID < V+m']),
    ('输出表',[ '原 VxD: 逐行保留', '新增 mxD 行', 'logits 宽度 V+m']),
    ('绑定模型',[ '创建新 Parameter', '输入 / 输出指向同一对象', '再建立 optimizer']]))
print('generated',len(list(OUT.glob('*.svg'))),'SVG and matching PNG files')
```

examples/architectures/source_attention_checks.py

```
"""Original tiny checks of audited upstream attention modules; no main/CLI execution."""
import ast, importlib.util, json, pathlib, sys
sys.dont_write_bytecode=True
import torch
ROOT=pathlib.Path(__file__).resolve().parents[2];UP=ROOT/'runtime_sources/upstream'
sys.path.insert(0, str(ROOT / 'examples'))
from verify_runtime import verify_runtime
verify_runtime(ROOT / 'runtime_sources')
torch.manual_seed(1949);torch.set_num_threads(1)
def load(rel):
    p=UP/rel;tree=ast.parse(p.read_text())
    # Imports, definitions, constant assignments, and guarded CLI only; no notebook
    loading.
    for node in tree.body:
        assert isinstance(node,
        (ast.Import,ast.ImportFrom,ast.ClassDef,ast.FunctionDef,ast.Assign,ast.If)),type(node)
        if isinstance(node,ast.If):assert ast.unparse(node.test)=="__name__ ==
        '__main__'",ast.unparse(node.test)
```

```

spec=importlib.util.spec_from_file_location(p.stem,p);m=importlib.util.module_from_spec(
spec);spec.loader.exec_module(m);return m
results={};x=torch.randn(1,7,8)
for folder,file,cls,extra in [
    ('04_gqa','gpt_with_kv_gqa','GroupedQueryAttention',{'num_kv_groups':2}),
    ('05_mla','gpt_with_kv_mla','MultiHeadLatentAttention',{'latent_dim':3}),
    ('06_swa','gpt_with_kv_swa','MultiHeadAttentionWithSWA',{'sliding_window_size':3}),
    ('09_dsa','gpt_with_kv_dsa','MultiHeadAttentionWithDSA',
{'index_n_heads':2,'index_head_dim':3,'topk':3})]:
    m=load(f'ch04/{folder}/{file}.py');a=getattr(m,cls)
    (d_in=8,d_out=8,dropout=0.,num_heads=4,**extra).eval()
    with torch.no_grad():
        full=a(x);
a.reset_cache();chunk=torch.cat([a(x[:, :2],use_cache=True),a(x[:, 2:5],use_cache=True),a(
x[:, 5:],use_cache=True)],1)
    err=float((full-chunk).abs().max());assert err<1e-5,(folder,err)
    results[folder]={'chunk_error':err}
    if folder=='09_dsa':
        a.reset_cache();a(x).square().sum().backward()
        none=all(p.grad is None for p in a.indexer.parameters());assert none
        results[folder]['indexer_main_loss_gradients_none']=none
m=load('ch04/07_moe/gpt_with_kv_moe.py')
cfg={'emb_dim':8,'hidden_dim':12,'num_experts':3,'num_experts_per_tok':1}
a=m.MoEFeedForward(cfg);a(x).square().sum().backward();grad=float(a.gate.weight.grad.abs
().max());assert grad==0
results['moe_top1']={'router_max_abs_gradient':grad}
m=load('ch04/10_kv-sharing/gpt_with_kv_sharing.py')
cfg={'vocab_size':31,'context_length':12,'emb_dim':8,'n_layers':3,'n_heads':2,'drop_rate
':0.,'qkv_bias':False,'n_kv_producing_layers':1}
a=m.GPTModel(cfg).eval();ids=torch.tensor([[1,2,3,4,5,6,7]])
with torch.no_grad():

full=a(ids);a.reset_kv_cache();chunk=torch.cat([a(ids[:, :2],use_cache=True),a(ids[:, 2:5]
,use_cache=True),a(ids[:, 5:],use_cache=True)],1)
    err=float((full-chunk).abs().max());assert err<1e-5
    assert all(b.att.cache_k is None for b in a.trf_blocks[1:])
    results['kv_sharing']={'chunk_error':err,'consumers_have_no_own_cache':True}
    print(json.dumps({'torch':torch.__version__, 'seed':1949, 'results':results},indent=2))

```

examples/architectures/source_smoke.py

```

"""Original harness: execute ONLY audited architecture-definition notebook cells.
No notebook Run All, downloads, authentication, weights, or external corpora.
Upstream source remains under research/UPSTREAM_LICENSE.txt; no code copied here.
"""

import argparse, contextlib, importlib.util, io, json, math, pathlib, sys
sys.dont_write_bytecode=True
import torch
import torch.nn as nn
ROOT=pathlib.Path(__file__).resolve().parents[2]
UP=ROOT/'runtime_sources/upstream'
sys.path.insert(0, str(ROOT / 'examples'))

```

```

from verify_runtime import verify_runtime
verify_runtime(ROOT / 'runtime_sources')
torch.set_num_threads(1); torch.manual_seed(1949)

def cells(path, indices):
    nb=json.loads((UP/path).read_text()); ns={'torch':torch,'nn':nn,'math':math}
    for i in indices:
        src=''.join(nb['cells'][i]['source'])
        assert not any(s in src for s in ['hf_hub_download(', 'snapshot_download(',
'login(', 'load_file(')], (path,i)
        exec(compile(src,f'{path}:cell{i}','exec'),ns)
    return ns

def tiny():
    return dict(vocab_size=31,emb_dim=16,hidden_dim=24,n_layers=2,n_heads=4,head_dim=4,

n_kv_groups=2,n_kv_heads=2,context_length=16,rope_base=10000.,dtype=torch.float32,

rope_freq=None,qk_norm=True,query_pre_attn_scalar=4,layer_types=['sliding_attention','full_attention'],
    rope_local_base=10000.,rope_global_base=10000.,sliding_window=3,

rms_norm_eps=1e-6,attention_bias=False,rope_attention_factor=1.,rope_type='default',rope_factor=1.,rope_orig_max=16,

layer_norm_eps=1e-5,logit_scale=0.7,tie_word_embeddings=True,num_experts=3,num_experts_per_tok=2,moe_hidden_dim=12)

specs=[
    ('llama32','ch05/07_gpt_to_llama/standalone-llama32.ipynb',list(range(6,11)), 'Llama3Model'),
    ('qwen3','ch05/11_qwen3/standalone-qwen3.ipynb',list(range(8,14)), 'Qwen3Model'),
    ('qwen3_moe','ch05/11_qwen3/standalone-qwen3-moe.ipynb',list(range(6,12)), 'Qwen3Model'),
    ('gemma3','ch05/12_gemma3/standalone-gemma3.ipynb',list(range(8,14)), 'Gemma3Model'),
    ('olmo3','ch05/13_olmo3/standalone-olmo3.ipynb',list(range(9,15)), 'Olmo3Model'),
    ('tiny_aya','ch05/15_tiny-aya/standalone-tiny-aya.ipynb',list(range(9,15)), 'TinyAyaModel'),
    ('gemma4','ch05/17_gemma4/standalone-gemma4.ipynb',list(range(9,14))
+[15], 'Gemma4DenseModel'),
]
results={}
for name,path,indices,cls in specs:
    ns=cells(path,indices); cfg=tiny()
    if name=='gemma4':
        cfg=ns['get_gemma4_dense_config']('E2B',torch.float32)

cfg.update(vocab_size=31,vocab_size_per_layer_input=31,emb_dim=16,hidden_dim=24,n_layers=4,

n_heads=4,head_dim=4,global_head_dim=8,n_kv_heads=2,context_length=16,sliding_window=3,

layer_types=['sliding_attention','full_attention']*2,hidden_size_per_layer_input=4,num_kv_shared_layers=2)
    model=ns[cls](cfg).eval(); ids=torch.tensor([[1,3,5,7,9,11]]); bad=ids.clone();
    bad[:,4:]=2

```

```

with torch.no_grad():
    a=model(ids); b=model(bad)
    assert a.shape==(1,6,31) and torch.isfinite(a).all()
    error=float((a[:, :4]-b[:, :4]).abs().max()); assert error<1e-5, (name,error)

results[name]={ 'shape':list(a.shape), 'future_leakage_max_abs':error, 'parameter_count':sum(p.numel() for p in model.parameters()), 'source_cells':indices}
    if name=='gemma4':
        with torch.no_grad(): shared=model(ids,reuse_shared_kv=True)
        assert torch.isfinite(shared).all()
        results[name][ 'shared_kv_finite']=True
# Audited helper imports only torch, creates a logger, defines classes/functions.
p=UP/'ch05/16_qwen3.5/qwen3_5_transformers.py'
spec=importlib.util.spec_from_file_location('qwen35_helper',p);
helper=importlib.util.module_from_spec(spec); spec.loader.exec_module(helper)
q=torch.randn(1,7,2,3); k=torch.randn_like(q); v=torch.randn(1,7,2,4); g=-
torch.rand(1,7,2); beta=torch.rand(1,7,2)
a,s1=helper.torch_chunk_gated_delta_rule(q,k,v,g,beta,chunk_size=4,output_final_state=True,use_qk_l2norm_in_kernel=True)
b,s2=helper.torch_recurrent_gated_delta_rule(q,k,v,g,beta,None,True,True)
error=float((a-b).abs().max()); state_error=float((s1-s2).abs().max())
assert error<1e-5 and state_error<1e-5
results['qwen35_chunk_recurrent']={ 'max_abs_error':error, 'final_state_error':state_error, 'input_shape':list(q.shape), 'nondivisible_chunk_padding':True}
print(json.dumps({'torch':torch.__version__, 'seed':1949, 'threads':1, 'results':results}, indent=2))

```

examples/core_training/cache_probes.py

```

"""Small-config checks of the three upstream KV variants, with no downloads."""
import ast
import json
from pathlib import Path
import sys
import torch
from torch import nn
root=Path(sys.argv[1]); out=Path(sys.argv[2])
if (root.parent / 'SOURCE_MAP.json').exists():
    sys.path.insert(0, str(Path(__file__).resolve().parents[1]))
    from verify_runtime import verify_runtime
    verify_runtime(root.parent)
torch.set_num_threads(1); torch.manual_seed(2718)
base=root/'ch04/03_kv-cache'
modules={}
for file in ['gpt_ch04.py', 'gpt_with_kv_cache.py', 'gpt_with_kv_cache_optimized.py']:
    tree=ast.parse((base/file).read_text())
    tree.body=[n for n in tree.body if isinstance(n, (ast.ClassDef, ast.FunctionDef)) and
getattr(n, 'name', '')!='main']
    ns={'torch':torch, 'nn':nn}; exec(compile(tree, str(base/file), 'exec'), ns);
modules[file]=ns
cfg=dict(vocab_size=16, context_length=12, emb_dim=16, n_heads=4, n_layers=2, drop_rate=0., qk
v_bias=False, kv_window_size=12)
models=[]

```

```

for ns in modules.values():
    torch.manual_seed(2718); models.append(ns['GPTModel'](cfg).eval())
for m in models[1:]:m.load_state_dict(models[0].state_dict(),strict=False)
x=torch.tensor([[1,2,3,4,5,6]])
with torch.no_grad():
    full=models[0](x); result={}
    for file,m in zip(list(modules)[1:],models[1:]):
        m.reset_kv_cache(); parts=[m(x[:,i:i+1],use_cache=True) for i in range(6)]
        error=(torch.cat(parts,1)-full).abs().max().item();assert error<1e-6
        m.reset_kv_cache();m(x[:, :3],use_cache=True);chunk=m(x[:, :3],use_cache=True)
        chunkerror=(chunk-full[:, :3]).abs().max().item();assert chunkerror<1e-6
        m.reset_kv_cache(); again=m(x,use_cache=True)
        assert torch.allclose(full,again,atol=1e-6)
        result[file]={'token_error':error,'chunk_error':chunkerror,'reset_ok':True}
ns=modules['gpt_with_kv_cache_optimized.py']
smallcfg=dict(cfg,context_length=10,kv_window_size=4)
short=ns['GPTModel'](smallcfg).eval()
got=ns['generate_text_simple_cached'](short,x,max_new_tokens=20,use_cache=True)
assert got.shape[1]==10 and torch.equal(got[:, :6],x)

result['window_and_capacity']={'prompt_length':6,'requested_new':20,'actual_new':4,'window':4,'context':10}
# Window eviction is not equivalent to full-context attention.
windowed=ns['GPTModel'](dict(cfg,kv_window_size=4)).eval()
windowed.load_state_dict(models[0].state_dict(),strict=False)
pieces=[windowed(x[:,i:i+1],use_cache=True) for i in range(6)]
discrepancy=(torch.cat(pieces,1)-full).abs().max().item()
assert discrepancy>1e-5
result['eviction_changes_model']={'max_error_vs_full':discrepancy}
result['torch']=torch.__version__;result['seed']=2718;result['status']='all assertions passed'
out.parent.mkdir(exist_ok=True,parents=True);out.write_text(json.dumps(result,indent=2)+'\n');print(json.dumps(result,indent=2))

```

examples/core_training/checkpoint_checks.py

```

"""Owned tiny checkpoints: exact continuation and meta/mmap/assign loading."""
import copy
import json
from pathlib import Path
import sys
import tempfile
import torch
from core import TinyGPT, token_loss, windows

torch.set_num_threads(1);torch.manual_seed(2718)
cfg=dict(vocab=8,context=8,width=8,heads=2,layers=1)
x,y=windows([0,1,2,3,4,5,6,7]*4,4,4)
m=TinyGPT(**cfg);opt=torch.optim.AdamW(m.parameters(),lr=.01)
def step(model,optimizer):

optimizer.zero_grad(set_to_none=True);loss=token_loss(model(x),y);loss.backward();optimizer.step()

```

```

        return loss.item()
    for _ in range(3):step(m,opt)
    snapshot={'config':cfg,'model':copy.deepcopy(m.state_dict()),'optimizer':copy.deepcopy(opt.state_dict()),
            'rng':torch.get_rng_state(),'step':3}
    step(m,opt); expected=copy.deepcopy(m.state_dict())
    with tempfile.TemporaryDirectory() as tmp:
        file=Path(tmp)/'resume.pt';torch.save(snapshot,file)
        checkpoint=torch.load(file,map_location='cpu',weights_only=True)
        loaded=TinyGPT(**checkpoint['config']);loaded.load_state_dict(checkpoint['model'])

    optimizer=torch.optim.AdamW(loaded.parameters(),lr=.01);optimizer.load_state_dict(checkpoint['optimizer'])
    torch.set_rng_state(checkpoint['rng']);step(loaded,optimizer)
    error=max((expected[k]-loaded.state_dict()[k]).abs().max().item() for k in expected)
    if expected[k].is_floating_point()
    assert error==0
    weights=Path(tmp)/'weights.pt';torch.save(m.state_dict(),weights)
    with torch.device('meta'):
        meta=TinyGPT(**cfg)

    meta.load_state_dict(torch.load(weights,map_location='cpu',weights_only=True,mmap=True),
    assign=True)
    assert not any(p.is_meta for p in meta.parameters())
    assert torch.equal(meta(x),m(x))
    assert all(torch.equal(meta.state_dict()[k],v) for k,v in m.state_dict().items())
    # Incorrectly restoring only parameters after to_empty loses initialized buffers.
    with torch.device('meta'):
        incomplete=TinyGPT(**cfg)
    incomplete.to_empty(device='cpu')
    with torch.no_grad():
        for name,p in incomplete.named_parameters():p.copy_(m.state_dict()[name])
        # Deterministic sentinel represents an unrestored buffer without reading uninitialized memory.
        incomplete.blocks[0].att.future.fill_(False)
    assert not torch.equal(incomplete.blocks[0].att.future,m.blocks[0].att.future)
    broken_error=(incomplete(x)-m(x)).abs().max().item();assert broken_error>1e-5
    # Safe in-place copy preserves optimizer parameter identity; replacement does not.
    old=loaded.token.weight;old_identity=id(old)
    loaded.token.weight=torch.nn.Parameter(old.detach().clone())
    stale=any(id(p)==old_identity for group in optimizer.param_groups for p in group['params'])
    assert stale and all(p is not loaded.token.weight for group in optimizer.param_groups for p in group['params'])
    result={'torch':torch.__version__,'seed':2718,'resume_parameter_error':error,'meta_mmap_assign_logits_error':0.,

    'all_buffers_restored':True,'omitted_mask_sentinel_error':broken_error,'replacement_leaves_optimizer_stale':stale,
        'memory_peak_measured':False,'status':'all assertions passed'}
    Path(sys.argv[1]).write_text(json.dumps(result,indent=2)+'\n');print(json.dumps(result,indent=2))

```

examples/core_training/core.py

```

"""Original CPU teaching implementation; public rasbt code provides comparison targets.
No network, pretrained weights, or external data are used. See NOTICE.md.
"""

import math
from collections import Counter
import torch
from torch import nn
from torch.nn import functional as F

class ByteBPE:
    """Minimal byte BPE: deliberately no GPT-2 regex or pretrained vocabulary."""
    def __init__(self):
        self.pieces = {i: bytes([i]) for i in range(256)}
        self.merges = {}

    @staticmethod
    def merge(ids, pair, new_id):
        out, i = [], 0
        while i < len(ids):
            if i + 1 < len(ids) and tuple(ids[i:i+2]) == pair:
                out.append(new_id)
                i += 2
            else:
                out.append(ids[i])
                i += 1
        return out

    def train(self, text, vocab_size=270):
        if vocab_size < 256:
            raise ValueError("A byte vocabulary needs at least 256 entries")
        self.__init__()
        ids = list(text.encode("utf-8"))
        while len(self.pieces) < vocab_size and len(ids) > 1:
            counts = Counter(zip(ids, ids[1:]))
            # Stable tie break makes the exercise reproducible.
            pair = min(counts, key=lambda p: (-counts[p], p))
            if counts[pair] < 2:
                break
            new_id = len(self.pieces)
            self.merges[pair] = new_id
            self.pieces[new_id] = self.pieces[pair[0]] + self.pieces[pair[1]]
            ids = self.merge(ids, pair, new_id)
        return self

    def encode(self, text):
        ids = list(text.encode("utf-8"))
        while len(ids) > 1:
            pairs = set(zip(ids, ids[1:]))
            candidates = pairs.intersection(self.merges)
            if not candidates:
                break

```

```

        pair = min(candidates, key=self.merges.get)
        ids = self.merge(ids, pair, self.merges[pair])
    return ids

def decode(self, ids):
    return b"".join(self.pieces[i] for i in ids).decode("utf-8")

def windows(ids, length, stride):
    if length <= 0 or stride <= 0 or len(ids) <= length:
        raise ValueError("Need positive length/stride and at least length+1 IDs")
    starts = range(0, len(ids)-length, stride)
    x = torch.tensor([ids[i:i+length] for i in starts], dtype=torch.long)
    y = torch.tensor([ids[i+1:i+length+1] for i in starts], dtype=torch.long)
    return x, y

class Attention(nn.Module):
    def __init__(self, width, heads, context):
        super().__init__()
        if width % heads:
            raise ValueError("width must be divisible by heads")
        self.heads, self.head_width = heads, width // heads
        self.qkv = nn.Linear(width, 3*width, bias=False)
        self.proj = nn.Linear(width, width)
        self.register_buffer("future", torch.ones(context, context,
dtype=torch.bool).triu(1))

    def forward(self, x, cache=None):
        b, t, d = x.shape
        q, k, v = self.qkv(x).chunk(3, dim=-1)
        q, k, v = [a.reshape(b, t, self.heads, self.head_width).transpose(1, 2)
                    for a in (q, k, v)]
        past = 0
        if cache is not None:
            past = cache[0].shape[2]
            k, v = torch.cat((cache[0], k), 2), torch.cat((cache[1], v), 2)
        total = past + t
        if total > self.future.shape[0]:
            raise ValueError("cache exceeds learned positional context")
        scores = (q @ k.transpose(-1, -2)) / math.sqrt(self.head_width)
        scores = scores.masked_fill(self.future[past:total, :total], -torch.inf)
        weights = scores.softmax(-1)
        out = (weights @ v).transpose(1, 2).reshape(b, t, d)
        return self.proj(out), (k, v)

class Block(nn.Module):
    def __init__(self, width, heads, context):
        super().__init__()
        self.norm1, self.norm2 = nn.LayerNorm(width), nn.LayerNorm(width)
        self.att = Attention(width, heads, context)
        self.ff = nn.Sequential(nn.Linear(width, 4*width), nn.GELU(approximate="tanh"),
                                nn.Linear(4*width, width))

    def forward(self, x, cache=None):

```

```

        update, cache = self.att(self.norm1(x), cache)
        x = x + update
        return x + self.ff(self.norm2(x)), cache

class TinyGPT(nn.Module):
    def __init__(self, vocab=16, context=16, width=16, heads=4, layers=2):
        super().__init__()
        self.config = dict(vocab=vocab, context=context, width=width, heads=heads,
layers=layers)
        self.token = nn.Embedding(vocab, width)
        self.position = nn.Embedding(context, width)
        self.blocks = nn.ModuleList([Block(width, heads, context) for _ in
range(layers)])
        self.norm = nn.LayerNorm(width)
        self.head = nn.Linear(width, vocab, bias=False)

    def forward(self, ids, caches=None, return_cache=False):
        past = 0 if caches is None else caches[0][0].shape[2]
        if past + ids.shape[1] > self.config["context"]:
            raise ValueError("input plus cache exceeds context")
        pos = torch.arange(past, past+ids.shape[1], device=ids.device)
        x = self.token(ids) + self.position(pos)
        new = []
        for i, block in enumerate(self.blocks):
            x, cache = block(x, None if caches is None else caches[i])
            new.append(cache)
        logits = self.head(self.norm(x))
        return (logits, new) if return_cache else logits

def token_loss(logits, targets):
    return F.cross_entropy(logits.flatten(0, 1), targets.flatten())

def sample(logits, temperature=1.0, top_k=None, generator=None):
    if temperature < 0:
        raise ValueError("temperature must be nonnegative")
    if temperature == 0:
        return logits.argmax(-1, keepdim=True)
    filtered = logits.clone()
    if top_k is not None:
        if not 1 <= top_k <= logits.shape[-1]:
            raise ValueError("top_k outside vocabulary")
        values, indices = filtered.topk(top_k, dim=-1)
        filtered.fill_(-torch.inf).scatter_(-1, indices, values)
    return torch.multinomial((filtered/temperature).softmax(-1), 1, generator=generator)

def schedule(step, warmup, total, start, peak, end):
    if not 0 <= warmup < total:
        raise ValueError("Require 0 <= warmup < total")
    if step < 0:
        raise ValueError("negative step")
    if warmup and step < warmup:
        return start + (peak-start)*step/warmup

```

```

progress = min(1., (step-warmup)/(total-warmup))
return end + .5*(peak-end)*(1+math.cos(math.pi*progress))

@torch.no_grad()
def generate(model, ids, max_new_tokens, context, temperature=0., top_k=None,
             eos_id=None, generator=None):
    """Batch-size-one generator. Returned sequence excludes sampled EOS."""
    if ids.ndim != 2 or ids.shape[0] != 1 or ids.shape[1] == 0:
        raise ValueError("Require one nonempty prompt")
    if context <= 0 or max_new_tokens < 0:
        raise ValueError("Invalid generation limits")
    previous_mode = model.training
    model.eval()
    try:
        for _ in range(max_new_tokens):
            logits = model(ids[:, -context:][:, -1])
            new = sample(logits, temperature, top_k, generator)
            if eos_id is not None and new.item() == eos_id:
                break
            ids = torch.cat((ids, new), dim=1)
    finally:
        model.train(previous_mode)
    return ids

```

examples/core_training/corpus_demo.py

```

"""Original tiny corpus pipeline. No Gutenberg downloads or language detection
claims."""
import hashlib
import json
from pathlib import Path
import sys

def canonical(text):
    return text.replace('\r\n', '\n').replace('\r', '\n').strip()

def prepare(records):
    seen, kept, rejected = set(), [], []
    for name, text, split in sorted(records):
        clean = canonical(text)
        if not clean:
            rejected.append({'name': name, 'reason': 'empty'}); continue
        digest = hashlib.sha256(clean.encode('utf-8')).hexdigest()
        if digest in seen:
            rejected.append({'name': name, 'reason': 'exact duplicate'}); continue
        seen.add(digest)

    kept.append({'name': name, 'text': clean, 'split': split, 'sha256': digest, 'bytes': len(clean.encode('utf-8'))})
    return kept, rejected

```

```

def pack(records,max_bytes,separator='\n<|endoftext|>\n'):
    """A single oversized document gets its own flagged shard, never an empty shard."""
    shards=[]; current=[]; size=0; sep=len(separator.encode())
    for record in records:
        extra=record['bytes']+(sep if current else 0)
        if current and size+extra>max_bytes:
            shards.append(separator.join(current));current=[];size=0
        if current:size+=sep
        current.append(record['text']);size+=record['bytes']
    if current:shards.append(separator.join(current))
    return shards

def main(path):
    records=[('01_train','Rain wets the stone.\r\nWind dries the stone.','train'),
             ('02_duplicate','Rain wets the stone.\nWind dries the
stone.','validation'),
             ('03_train','A lamp lights the desk.','train'),
             ('04_validation','A small boat crosses the lake.','validation'),
             ('05_empty',' \n','train'),('06_unicode','山与海','train')]
    kept,rejected=prepare(records)
    hashes={split:{r['sha256'] for r in kept if r['split']==split} for split in
['train','validation']}
    assert not hashes['train'].intersection(hashes['validation'])
    train=[r for r in kept if r['split']=='train'];shards=pack(train,64)
    assert all(shards) and len(shards)==2
    oversized=pack([{'text':'x'*80,'bytes':80}],64)
    assert len(oversized)==1 and len(oversized[0])==80
    assert prepare([])==([],[])
    # Separator bytes count: two one-byte docs + fifteen separator bytes = seventeen.
    sep='\n<|endoftext|>\n'
    assert len(pack([{'text':'a','bytes':1},{'text':'b','bytes':1}],2))==2
    result={'kept':[{k:v for k,v in r.items() if k!='text'} for r in kept],
'rejected':rejected,
          'train_shards':shards,'shard_bytes':[len(s.encode()) for s in shards],
          'separator_bytes':len(sep.encode()),'oversized_document_policy':'own flagged
shard; downstream must handle',
          'empty_inputs_ok':True,'checkpoint_steps_for_frequency_3':[s for s in
range(1,10) if s%3==0],
          'status':'all assertions passed'}

    Path(path).write_text(json.dumps(result,ensure_ascii=False,indent=2)+'\n');print(json.du
mps(result,ensure_ascii=False,indent=2))
    if __name__=='__main__':main(sys.argv[1])

```

examples/core_training/draw_figures.py

```

"""Original diagrams; no upstream images are reused."""
import json
from pathlib import Path

```

```

import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt
from matplotlib.patches import FancyBboxPatch
import numpy as np
ROOT = Path(__file__).resolve().parents[2]
OUT = ROOT/'figures/core_training'
OUT.mkdir(parents=True,exist_ok=True)
plt.rcParams.update({'font.size':11,'figure.dpi':140,'savefig.facecolor':'white'})

def save(fig,name):
    fig.savefig(OUT/(name+'.png'),bbox_inches='tight')
    fig.savefig(OUT/(name+'.svg'),bbox_inches='tight')
    plt.close(fig)

def boxes(name, labels, subtitle):
    fig,ax=plt.subplots(figsize=(9,2.6))
    n=len(labels)
    for i,label in enumerate(labels):
        x=i*1.9
        ax.add_patch(FancyBboxPatch((x,.5),1.65,.85,boxstyle='round,pad=.08',
                                   facecolor='#e8f2fa',edgecolor='#23658c'))
        ax.text(x+.825,.925,label,ha='center',va='center')
        if i<n-1:ax.annotate('',(x+1.85,.925),(x+1.72,.925),arrowprops={'arrowstyle':'-
>'})
    ax.text((n*1.9-.25)/2,.12,subtitle,ha='center',va='center',fontsize=10)
    ax.set(xlim=(-.2,n*1.9-.1),ylim=(-.1,1.7));ax.axis('off');save(fig,name)
boxes('pipeline',['Text\nUTF-8','Token IDs\n[B, T]','GPT\nparameters','Logits\n[B, T,
V]'],
      'Training: shifted targets -> loss -> gradients; generation: last logits -> next
ID')
boxes('autograd',['x, w, b','z = xw + b','L = softplus(-z)','gradients\nupdate w, b'],
      'For y = 1: dL/dz = sigmoid(z) - 1; dL/dw = x * dL/dz')
fig,ax=plt.subplots(figsize=(8,3))
rows=[[0,1,2,3],[1,2,3,4],[3,4,5,6],[4,5,6,7]]
for r,row in enumerate(rows):
    for c,v in enumerate(row):
        ax.text(c,r,str(v),ha='center',va='center',bbox={'boxstyle':'round','fc':'#e8f2fa' if
r%2==0 else '#fff0d5'})
ax.set(yticks=range(4),yticklabels=['input i=0','target i=0','input i=3','target
i=3'],xticks=[],xlim=(-.5,3.5),ylim=(3.6,-.6))
ax.set_title('Context length 4, stride 3: target index = input index + 1')
for s in ax.spines.values():s.set_visible(False)
save(fig,'windows')
fig,axs=plt.subplots(1,2,figsize=(8,3.3))
for ax,data,title in zip(axs,[np.tril(np.ones((5,5))),np.array([[1,1,1,1,0],
[1,1,1,1,1]]),['Full causal attention','Chunk after 3 cached tokens']):
    ax.imshow(data,cmap='Blues',vmin=0,vmax=1)
    ax.set_title(title);ax.set_xlabel('Key position');ax.set_ylabel('Query row')
    ax.set_xticks(range(5));ax.set_yticks(range(len(data)))
    for i in range(len(data)):
        for j in
range(5):ax.text(j,i,str(int(data[i,j])),ha='center',va='center',color='white' if
data[i,j] else 'black')
fig.tight_layout();save(fig,'causal_cache')

```

```

boxes('block',['x [B,T,D]','LN -> Attention\n+ x','LN -> MLP\n+ residual','Next
block\nor output head'],
      'Pre-normalization residual block; each addition requires matching shapes')
results=json.loads((ROOT/'qa/core_training/results.json').read_text())
fig,ax=plt.subplots(figsize=(7,3.5))
ax.plot(range(0,61,10),results['pretraining']['losses_every_10'],marker='o')
ax.set(xlabel='Optimizer steps',ylabel='Training cross-entropy',title='Tiny CPU
experiment: periodic synthetic IDs, not language quality')
ax.grid(alpha=.25);fig.tight_layout();save(fig,'loss')
fig,ax=plt.subplots(figsize=(7,3.5))
ax.plot(range(12),results['schedule_and_clip']['learning_rates'],marker='o')
ax.set(xlabel='Step index',ylabel='Learning rate',title='Warmup = 2, cosine endpoint =
10')
ax.grid(alpha=.25);fig.tight_layout();save(fig,'schedule')

```

examples/core_training/loss_sampling_checks.py

```

"""Loss aggregation, softmax gradients, top-k ties, EOS and mode checks."""
import json
from pathlib import Path
import sys
import torch
from torch import nn
from torch.nn import functional as F
from core import sample, generate

torch.manual_seed(2718);torch.set_num_threads(1)
z=torch.tensor([[2.,-1.,.5]],dtype=torch.float64,requires_grad=True)
y=torch.tensor([2]);loss=F.cross_entropy(z,y);loss.backward()
expected=z.detach().softmax(-1)-F.one_hot(y,3)
assert torch.allclose(z.grad,expected,atol=1e-14)
gradient_error=(z.grad-expected).abs().max().item()
# Unequal valid-token counts: average of means is not the token mean.
losses=torch.tensor([1.,1.,1.,4.]);
correct=losses.mean().item();wrong=((losses[:3].mean()+losses[3:].mean())/2).item()
assert correct==1.75 and wrong==2.5
z=torch.tensor([[3.,2.,2.,0.]])
threshold=z.topk(2).values[:,-1:]
threshold_count=(z>=threshold).sum().item()
gen=torch.Generator().manual_seed(2718)
exact=[sample(z,1.,2,gen).item() for _ in range(200)]
assert len(set(exact))<=2 and threshold_count==3
class Fixed(nn.Module):
    def forward(self,x):
        logits=torch.zeros(*x.shape,4);logits[...,-1]=8
        return logits
m=Fixed().train();prompt=torch.tensor([[1,2]])
stopped=generate(m,prompt,10,4,eos_id=3)
assert torch.equal(stopped,prompt) and m.training
extended=generate(m,prompt,3,4)
assert extended.tolist()==[[1,2,3,3]]
results={'torch':torch.__version__,'seed':2718,'cross_entropy_gradient_error':gradient_e
rror,

```

```
'weighted_token_mean':correct,'wrong_mean_of_means':wrong,'threshold_top2_support':threshold_count,
    'exact_top2_observed_ids':sorted(set(exact)), 'eos_excluded':stopped.tolist(),

'bounded_generation':extended.tolist(), 'training_mode_restored':m.training, 'status':'all assertions passed'}
Path(sys.argv[1]).write_text(json.dumps(results,indent=2)+'\n');print(json.dumps(results,indent=2))
```

examples/core_training/optimizer_numerics.py

```
"""Schedule boundaries, Muon parameter routing, matrix direction, CPU numerics.
The SVD example illustrates a matrix-orthogonalized direction; it is NOT
torch.optim.Muon.
"""

import ast
import itertools
import json
import math
from pathlib import Path
import sys
import torch
from torch import nn
from core import TinyGPT, schedule

torch.manual_seed(2718);torch.set_num_threads(1)
m=TinyGPT(); embeddings={id(p) for mod in m.modules() if isinstance(mod,nn.Embedding)
for p in mod.parameters(recurse=False)}
muon=[];adam=[]
for n,p in m.named_parameters():
    if p.requires_grad:
        (muon if p.ndim==2 and id(p) not in embeddings else adam).append((n,p))
ids1={id(p) for _,p in muon};ids2={id(p) for _,p in adam}
assert not ids1&ids2 and ids1|ids2=={id(p) for p in m.parameters() if p.requires_grad}
assert 'head.weight' in [n for n,p in muon]
g=torch.diag(torch.tensor([4.,1.],dtype=torch.float64));u,s,vh=torch.linalg.svd(g,full_matrices=False);direction=u@vh
assert torch.equal(direction,torch.eye(2,dtype=torch.float64))
assert torch.allclose(direction.T@direction,torch.eye(2,dtype=torch.float64))
for args in [(0,0,10,.01,.1,.001),(0,2,10,.01,.1,.001),(2,2,10,.01,.1,.001),
(10,2,10,.01,.1,.001)]:
    assert math.isfinite(schedule(*args))
for warm,total in [(2,2),(3,2),(-1,2)]:
    try:schedule(0,warm,total,.01,.1,.001);raise AssertionError('expected invalid schedule')
    except ValueError:pass
# Same seed same CPU random calls reproduce; an extra draw shifts the sequence.
torch.manual_seed(2718);a=torch.rand(6)
torch.manual_seed(2718);b=torch.rand(6);assert torch.equal(a,b)
torch.manual_seed(2718);torch.rand(1);c=torch.rand(6);assert not torch.equal(a,c)
# Rounding order counterexample in float32, not a device comparison.
values=torch.tensor([1e8,-1e8,1.],dtype=torch.float32)
```

```

left=((values[0]+values[1])+values[2]).item();right=(values[0]+
(values[1]+values[2])).item();assert left!=right
# A padded output class can be sampled unless explicitly excluded.
z=torch.tensor([[0.,1.,2.,100.]])
assert z.argmax(-1).item()==3
masked=z.clone();masked[:,3:]=-torch.inf;assert masked.argmax(-1).item()==2
# Gradient averaging with equal local batch sizes: mathematical DDP stand-in only.
w=torch.randn(3,2,dtype=torch.float64,requires_grad=True);x=torch.randn(4,3,dtype=torch.
float64);y=torch.randn(4,2,dtype=torch.float64)
full=torch.autograd.grad(((x@w-y)**2).mean(),w)[0]
local=[]
for start in [0,2]:local.append(torch.autograd.grad(((x[start:start+2]@w-
y[start:start+2])**2).mean(),w)[0])
error=((local[0]+local[1])/2-full).abs().max().item();assert error<1e-12
result={'torch':torch.__version__, 'seed':2718, 'muon_available':hasattr(torch.optim, 'Muon
'),
        'muon_group_names':[n for n,p in muon], 'adamw_group_names':[n for n,p in adam],
        'muon_group_elements':sum(p.numel() for _,p in
muon), 'adamw_group_elements':sum(p.numel() for _,p in adam),

        'svd_input_singular_values':s.tolist(), 'orthogonal_direction':direction.tolist(),

        'cpu_random_repeat_equal':True, 'extra_draw_changes_sequence':True, 'associativity_left':l
eft, 'associativity_right':right,
        'ddp_equal_batch_gradient_error':error, 'actual_ddp_run':False, 'mps_run':False,
        'status':'all assertions passed'}
Path(sys.argv[1]).write_text(json.dumps(result,indent=2)+'\n');print(json.dumps(result,i
ndent=2))

```

examples/core_training/run_checks.py

```

"""Run all offline CPU checks. Full commands and provenance: README.md, NOTICE.md."""
import argparse
import ast
import copy
import json
import math
from pathlib import Path
import platform
import tempfile
import torch
from torch import nn
from torch.nn import functional as F
from core import ByteBPE, windows, TinyGPT, Attention, token_loss, sample, schedule

def main():
    parser = argparse.ArgumentParser()
    parser.add_argument("--output", default="results.json")
    parser.add_argument("--upstream", default=None, help="Optional pinned rasbt source
tree")
    args = parser.parse_args()
    torch.set_num_threads(1)

```

```

torch.manual_seed(2718)
results = {"seed": 2718, "torch": torch.__version__, "python":
platform.python_version(),
          "device": "cpu", "threads": torch.get_num_threads()}
def record(name, value):
    results[name] = value

# Chapter 2: chain rule, accumulation, shape and checkpoint tests.
x = torch.tensor(1.3, dtype=torch.float64)
w = torch.tensor(.7, dtype=torch.float64, requires_grad=True)
b = torch.tensor(-.2, dtype=torch.float64, requires_grad=True)
loss = F.binary_cross_entropy_with_logits(x*w+b, torch.tensor(1.,
dtype=torch.float64))
gw, gb = torch.autograd.grad(loss, (w, b))
expected = torch.sigmoid(x*w+b)-1
assert torch.allclose(gw, expected*x) and torch.allclose(gb, expected)
eps = 1e-6
def objective(a):
    return F.softplus(-(x*a+b))
numeric = (objective(w+eps)-objective(w-eps))/(2*eps)
assert torch.allclose(gw, numeric, atol=1e-9)
record("chain_rule", {"loss": loss.item(), "dw": gw.item(), "db": gb.item(),
                      "finite_difference_error": abs(gw.item()-numeric.item())})
a = torch.arange(6.).reshape(2, 3).T
failed = False
try:
    a.view(6)
except RuntimeError:
    failed = True
assert failed and a.reshape(6).tolist() == [0., 3., 1., 4., 2., 5.]
model = nn.Sequential(nn.Linear(2, 8), nn.ReLU(), nn.Linear(8, 2))
features = torch.tensor([[ -2., 1.], [ -1., 2.], [ 1., -2.], [ 2., -1.]])
labels = torch.tensor([0, 0, 1, 1])
opt = torch.optim.SGD(model.parameters(), lr=.2)
initial = F.cross_entropy(model(features), labels).item()
for _ in range(40):
    opt.zero_grad(set_to_none=True)
    loss = F.cross_entropy(model(features), labels)
    loss.backward()
    opt.step()
final = F.cross_entropy(model(features), labels).item()
assert final < initial and model(features).argmax(-1).equal(labels)
record("classifier", {"initial": initial, "final": final, "accuracy": 1.0,
                      "noncontiguous_view_rejected": failed})

# Chapter 3: exact UTF-8 round trip and shifted labels.
tok = ByteBPE().train("低山 low low lower low\n低山", 274)
probes = ["", "山", "unseen 🐼\r\n text", " low low ", "<|endoftext|>"]
for text in probes:
    assert tok.decode(tok.encode(text)) == text
for text in ["", "x", "xy"]:
    short = ByteBPE().train(text, 270)
    assert short.decode(short.encode(text)) == text
ids = list(range(11))
wx, wy = windows(ids, 4, 3)
assert wx.tolist() == [[0,1,2,3],[3,4,5,6],[6,7,8,9]]

```

```

assert torch.equal(wx[:,1:], wy[:, :-1])
weight = torch.randn(7, 3, dtype=torch.float64, requires_grad=True)
ix = torch.tensor([2, 2, 6])
direct = F.embedding(ix, weight)
explicit = F.one_hot(ix, 7).double() @ weight
g1 = torch.autograd.grad(direct.square().sum(), weight, retain_graph=True)[0]
g2 = torch.autograd.grad(explicit.square().sum(), weight)[0]
assert torch.equal(direct, explicit) and torch.equal(g1, g2)
assert g1[0].count_nonzero() == 0
record("tokenization", {"vocab": len(tok.pieces), "merges": len(tok.merges),
                        "encoded_low": tok.encode(" low low "), "roundtrip_probes":
len(probes),
                        "windows_x": wx.tolist(), "windows_y": wy.tolist(),
                        "embedding_gradient_error": (g1-g2).abs().max().item()})

# Chapters 4-5: causal attention, manual vs SDPA, forward and gradient.
q = torch.randn(2, 3, 5, 4, dtype=torch.float64, requires_grad=True)
k = torch.randn_like(q, requires_grad=True)
v = torch.randn_like(q, requires_grad=True)
forbidden = torch.ones(5,5,dtype=torch.bool).triu(1)
weights = ((q @ k.transpose(-1,-2))/2).masked_fill(forbidden,
-torch.inf).softmax(-1)
manual = weights @ v
sdpa = F.scaled_dot_product_attention(q,k,v,is_causal=True)
assert torch.allclose(manual, sdpa, atol=1e-12, rtol=1e-12)
grad1 = torch.autograd.grad(manual.square().sum(), (q,k,v), retain_graph=True)
grad2 = torch.autograd.grad(sdpa.square().sum(), (q,k,v))
err = max((a-b).abs().max().item() for a,b in zip(grad1,grad2))
assert err < 1e-11
record("attention", {"forward_error": (manual-sdpa).abs().max().item(),
                    "gradient_error": err, "first_row": weights[0,0,0].tolist()})
att = Attention(12, 3, 8).double().eval()
h = torch.randn(2, 6, 12, dtype=torch.float64)
h2 = h.clone(); h2[:,4:] += 100
o1, _ = att(h); o2, _ = att(h2)
assert torch.equal(o1[:, :4], o2[:, :4])
assert "future" in att.state_dict() and "future" not in dict(att.named_parameters())
record("buffer", {"dtype": str(att.future.dtype), "persistent": True,
                  "parameter": False, "prefix_future_change_error": (o1[:, :4]-
o2[:, :4]).abs().max().item()})

# Chapters 6 and 8: full forward vs token and chunk caches.
torch.manual_seed(2718)
gpt = TinyGPT().double().eval()
prompt = torch.tensor([[1,3,5,7,9,11], [2,4,6,8,10,12]])
full = gpt(prompt)
caches, pieces = None, []
with torch.no_grad():
    for i in range(prompt.shape[1]):
        logits, caches = gpt(prompt[:,i:i+1], caches, True)
        pieces.append(logits)
streamed = torch.cat(pieces, dim=1)
_, prefix = gpt(prompt[:, :3], return_cache=True)
chunk, _ = gpt(prompt[:, 3:], prefix, True)
assert torch.allclose(full, streamed, atol=1e-12, rtol=1e-12)
assert torch.allclose(full[:, :3], chunk, atol=1e-12, rtol=1e-12)

```

```

        record("kv_cache", {"token_error": (full-streamed).abs().max().item(),
                             "chunk_error": (full[:,3:]-chunk).abs().max().item(),
                             "key_shape": list(caches[0][0].shape),
                             "cache_elements": sum(t.numel() for pair in caches for t in
pair)}})
        record("model", {"config": gpt.config, "parameters": sum(p.numel() for p in
gpt.parameters()),
                             "logits_shape": list(full.shape)})

# Chapter 13: correct CE, no test-data claims, small periodic training.
torch.manual_seed(2718)
train_model = TinyGPT(vocab=8, context=12, width=16, heads=4, layers=1)
stream = [0,1,2,3,4,5,6,7]*9
tx, ty = windows(stream, 8, 8)
optimizer = torch.optim.AdamW(train_model.parameters(), lr=.015, weight_decay=.01)
losses = []
for step in range(61):
    logits = train_model(tx)
    ce = token_loss(logits, ty)
    losses.append(ce.item())
    if step == 60:
        break
    optimizer.zero_grad(set_to_none=True)
    ce.backward()
    torch.nn.utils.clip_grad_norm_(train_model.parameters(), 1.)
    optimizer.step()
assert losses[-1] < .1 and losses[-1] < losses[0]
uniform = token_loss(torch.zeros(2,3,8), torch.zeros(2,3,dtype=torch.long))
assert abs(uniform.item()-math.log(8)) < 1e-6
good = F.cross_entropy(torch.tensor([[4.,0.,-2.]]), torch.tensor([0]))
bad = F.cross_entropy(torch.tensor([[4.,0.,-2.]]).softmax(-1), torch.tensor([0]))
record("pretraining", {"initial": losses[0], "final": losses[-1], "losses_every_10":
losses[:10],
                        "uniform_loss": uniform.item(), "correct_logits_ce":
good.item(),
                        "wrong_double_softmax_ce": bad.item(), "data": "synthetic
periodic IDs, no generalization claim"})
# We own this temporary checkpoint; safe weights-only deserialization.
with tempfile.TemporaryDirectory() as directory:
    path = Path(directory)/"tiny.pt"
    torch.save(train_model.state_dict(), path)
    loaded = TinyGPT(**train_model.config)
    loaded.load_state_dict(torch.load(path, map_location="cpu", weights_only=True))
    assert torch.equal(loaded(tx), train_model(tx))
record("checkpoint", {"prediction_error": 0.0, "weights_only": True})

# Chapters 14 and 17: bounded sampling and schedule endpoints.
z = torch.tensor([[0.,1.,2.,3.]])
generator = torch.Generator().manual_seed(2718)
draws = torch.cat([sample(z, .8, 2, generator) for _ in range(1000)], dim=1)
assert set(draws.flatten().tolist()) <= {2,3}
assert sample(z, 0).item() == 3
p = (z/.8).softmax(-1)
record("sampling", {"top2_counts":
torch.bincount(draws.flatten(), minlength=4).tolist(),
                    "temperature_0_8_probs": p.tolist()[0], "greedy": 3})

```

```

lr = [schedule(s, 2, 10, .01, .1, .001) for s in range(12)]
assert lr[0] == .01 and lr[2] == .1 and lr[10] == .001 and lr[11] == .001
parameter = nn.Parameter(torch.tensor([3.,4.])); parameter.grad =
torch.tensor([3.,4.])
norm = torch.nn.utils.clip_grad_norm_([parameter], 1.)
assert torch.allclose(parameter.grad, torch.tensor([.6,.8]), atol=1e-6)
record("schedule_and_clip", {"learning_rates": lr, "norm_before": norm.item(),
"gradient_after": parameter.grad.tolist()})

if args.upstream:
    if (Path(args.upstream).parent / "SOURCE_MAP.json").exists():
        import sys
        sys.path.insert(0, str(Path(__file__).resolve().parents[1]))
        from verify_runtime import verify_runtime
        verify_runtime(Path(args.upstream).parent)
    # Read inspected public code; drop imports and main execution, use only
definitions.
    path = Path(args.upstream)/"ch04/01_main-chapter-code/gpt.py"
    tree = ast.parse(path.read_text())
    keep = {"MultiHeadAttention", "LayerNorm", "GELU", "FeedForward",
"TransformerBlock", "GPTModel"}
    tree.body = [n for n in tree.body if isinstance(n, ast.ClassDef) and n.name in
keep]
    ns = {"torch": torch, "nn": nn}
    exec(compile(tree, str(path), "exec"), ns)
    cfg = dict(vocab_size=16, context_length=16, emb_dim=16, n_heads=4, n_layers=2,
drop_rate=0., qkv_bias=False)
    upstream = ns["GPTModel"](cfg).double().eval()
    # Explicit semantic mapping tests projection orientation and normalization
parameters.
    with torch.no_grad():
        upstream.tok_emb.weight.copy_(gpt.token.weight)
        upstream.pos_emb.weight.copy_(gpt.position.weight)
        upstream.final_norm.scale.copy_(gpt.norm.weight)
        upstream.final_norm.shift.copy_(gpt.norm.bias)
        upstream.out_head.weight.copy_(gpt.head.weight)
        for source, target in zip(gpt.blocks, upstream.trf_blocks):
            for name, own in [("norm1",source.norm1),("norm2",source.norm2)]:
                getattr(target,name).scale.copy_(own.weight)
                getattr(target,name).shift.copy_(own.bias)
            q,k,v = source.att.qkv.weight.chunk(3,0)
            for name, weight in [("W_query",q),("W_key",k),("W_value",v)]:
                getattr(target.att,name).weight.copy_(weight)
            target.att.out_proj.load_state_dict(source.att.proj.state_dict())
            target.ff.layers[0].load_state_dict(source.ff[0].state_dict())
            target.ff.layers[2].load_state_dict(source.ff[2].state_dict())
        theirs = upstream(prompt)
        error = (full-theirs).abs().max().item()
    # Upstream GELU constructs sqrt(2/pi) in default float32 even for float64
inputs.
    assert error < 1e-7
    record("upstream_parity", {"max_logit_error": error, "tolerance": 1e-7,
"reason_not_bitwise": "upstream GELU constant is
default float32",
"source": str(path), "config": cfg})
else:

```

```

        record("upstream_parity", {"status": "not requested; pass --upstream to
compare"})
        record("status", "all assertions passed")
        Path(args.output).parent.mkdir(parents=True, exist_ok=True)
        Path(args.output).write_text(json.dumps(results, ensure_ascii=False, indent=2)+"\n")
        print(json.dumps(results, ensure_ascii=False, indent=2))

if __name__ == "__main__":
    main()

```

examples/core_training/source_variant_checks.py

```

"""Additional offline source comparisons; requires installed torch and regex.
Only explicitly selected AST definitions are executed. Upstream files remain unmodified.
"""

import ast
from collections import Counter, deque
from functools import lru_cache
import json
import math
from pathlib import Path
import re
import sys
import torch
from torch import nn
import regex

root=Path(sys.argv[1]);out=Path(sys.argv[2]);torch.manual_seed(2718);torch.set_num_threa
ds(1)
if (root.parent / "SOURCE_MAP.json").exists():
    sys.path.insert(0, str(Path(__file__).resolve().parents[1]))
    from verify_runtime import verify_runtime
    verify_runtime(root.parent)
def extract(path,names,ns):
    if path.suffix=='.ipynb':
        texts=[''.join(c['source']) for c in json.loads(path.read_text())['cells'] if
c['cell_type']=='code']
    else:texts=[path.read_text()]
    nodes=[]
    for text in texts:
        if text.lstrip().startswith('%'):continue
        try:tree=ast.parse(text)
        except SyntaxError:continue
        nodes += [n for n in tree.body if isinstance(n,(ast.ClassDef,ast.FunctionDef))
and n.name in names]
        exec(compile(ast.Module(body=nodes,type_ignores=[]),str(path),'exec'),ns)
    return ns
p=root/'ch02/02_bonus_bytetrain-encoder'
ns=extract(p/'bpe_openai_gpt2.py',
{'bytes_to_unicode','get_pairs','Encoder'},dict(re=regex,lru_cache=lru_cache))
vocab=json.loads((p/'gpt2_model/encoder.json').read_text())
merges=[tuple(line.split()) for line in (p/'gpt2_model/

```

```

vocab.bpe').read_text().splitlines()[1:] if line.strip()]
original=ns['Encoder'](vocab,merges)
ns2=extract(root/'ch02/05_bpe-from-scratch/bpe-from-scratch.ipynb',
{'BPETokenizerSimple'},dict(Counter=Counter,deque=deque,lru_cache=lru_cache,json=json,re
=re))
complex_tok=ns2['BPETokenizerSimple']
();complex_tok.load_vocab_and_merges_from_openai(str(p/'gpt2_model/
encoder.json'),str(p/'gpt2_model/vocab.bpe'))
probes=['This is some text','Hello, world. Is this-- a test?','a b\n','a\r\nb','山与海
👉','a\tb']
bpe=[]
for text in probes:
    ids=original.encode(text);assert original.decode(ids)==text
    row={'input':repr(text),'official_ids':ids,'official_roundtrip':True}
    try:
        ours=complex_tok.encode(text);row.update(educational_ids=ours,ids_equal=ours==ids,educat
ional_roundtrip=complex_tok.decode(ours)==text)
        except ValueError as e:row['educational_error']=str(e)
        bpe.append(row)
assert bpe[0]['ids_equal'] and not bpe[-1]['educational_roundtrip']
path=root/'ch03/02_bonus_efficient-multihead-attention/mha-implementations.ipynb'
ns=extract(path,
{'Ch03_MHA','MHAEinsum','MHAPyTorchSDPAWithoutFlash'},dict(torch=torch,nn=nn,math=math))
checks=[]
for din,dout,heads in [(12,12,3),(8,12,3),(12,8,2)]:
    source=ns['Ch03_MHA'](din,dout,5,0.,heads,False).double().eval()
    target=ns['MHAEinsum'](din,dout,5,0.,heads,False).double().eval()
    with torch.no_grad():
        for name in
['W_query','W_key','W_value']:getattr(target,name).copy_(getattr(source,name).weight.T)
        target.out_proj.load_state_dict(source.out_proj.state_dict())
        x=torch.randn(2,5,din,dtype=torch.float64)
        error=(source(x)-target(x)).abs().max().item();assert error<1e-12
        checks.append({'d_in':din,'d_out':dout,'heads':heads,'max_error':error})
m=ns['MHAPyTorchSDPAWithoutFlash'](12,12,3,5,0.,False).double().eval()
x=torch.randn(2,5,12,dtype=torch.float64);changed=x.clone();changed[:,3:]+=100
error=(m(x)[:,3:]-m(changed)[:,3:]).abs().max().item();assert error==0
result={'torch':torch.__version__,'seed':2718,'bpe':bpe,'mha_einsum':checks,'explicit_sd
pa_causal_error':error,
        'no_network':True,'status':'all assertions passed'}
out.write_text(json.dumps(result,ensure_ascii=False,indent=2)+'\n');print(json.dumps(res
ult,ensure_ascii=False,indent=2))

```

examples/core_training/tokenizer_probes.py

```

"""Offline probes of selected inspected upstream class definitions, not whole
notebooks."""
import ast
from collections import Counter, deque
from functools import lru_cache
import json

```

```

from pathlib import Path
import re
import sys
import tempfile

root = Path(sys.argv[1])
if (root.parent / 'SOURCE_MAP.json').exists():
    sys.path.insert(0, str(Path(__file__).resolve().parents[1]))
    from verify_runtime import verify_runtime
    verify_runtime(root.parent)
output = Path(sys.argv[2])
results = {}
for variant in ["bpe-from-scratch-simple", "bpe-from-scratch"]:
    path = root / "ch02/05_bpe-from-scratch" / (variant+".ipynb")
    cells = json.loads(path.read_text())["cells"]
    nodes = []
    for c in cells:
        if c["cell_type"] != "code":
            continue
        for n in ast.parse("".join(c["source"])).body:
            if isinstance(n, ast.ClassDef) and n.name == "BPETokenizerSimple":
                nodes.append(n)
    ns = dict(Counter=Counter, deque=deque, lru_cache=lru_cache, re=re, json=json)
    exec(compile(ast.Module(body=nodes, type_ignores=[]), str(path), "exec"), ns)
    cls = ns["BPETokenizerSimple"]
    tok = cls()
    tok.train("a b a b\n", vocab_size=270)
    row = {}
    for text in ["a b\n", "a\tb", "Æ", "山"]:
        try:
            decoded = tok.decode(tok.encode(text))
            row[repr(text)] = dict(decoded=repr(decoded), exact=decoded==text)
        except ValueError as e:
            row[repr(text)] = dict(error=str(e))
    if variant == "bpe-from-scratch":
        try:
            tok.encode("a<|endoftext|>b")
            raise AssertionError("Expected disallowed special rejection")
        except ValueError:
            row["disallowed_special_rejected"] = True
        text = "a<|endoftext|>b"
        assert tok.decode(tok.encode(text, allowed_special={"<|endoftext|>})) == text
        with tempfile.TemporaryDirectory() as tmp:
            a, b = str(Path(tmp)/"vocab.json"), str(Path(tmp)/"merges.json")
            tok.save_vocab_and_merges(a,b)
            restored=cls(); restored.load_vocab_and_merges(a,b)
            assert restored.encode("a b") == tok.encode("a b")
            row["save_reload_ids_equal"] = True
    results[variant] = row
# These are documented counterexamples, not bugs in the test expectations.
assert not results["bpe-from-scratch-simple"][repr("a b\n")]["exact"]
assert results["bpe-from-scratch"][repr("a b\n")]["exact"]
assert not results["bpe-from-scratch"][repr("a\tb")]["exact"]
assert not results["bpe-from-scratch"][repr("Æ")]["exact"]
output.parent.mkdir(exist_ok=True, parents=True)

```

```
output.write_text(json.dumps(results,ensure_ascii=False,indent=2)+"\n")
print(json.dumps(results,ensure_ascii=False,indent=2))
```

examples/finetuning/check_upstream_functions.py

```
"""Execute selected reviewed functions from the pinned public code, offline.
Original test harness; does not copy source text into the deliverable.
Requires source-only runtime bundle at runtime_sources/upstream (or --upstream PATH).
"""

import argparse
import ast
import hashlib
import json
import math
from pathlib import Path
import re
import subprocess
import sys
import torch
import torch.nn.functional as F
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

PIN='cbbd33e6d57a71e20ea31434841095fac50cecd8'
# Verify the executed files even when a portable archive has no Git metadata.
PINNED_FILES = {'ch07/02_dataset-utilities/find-near-duplicates.py':
'67c51353d7152315284f9438bd4d03d4cfa414165225fe905c1777795e27b9c1', 'ch07/04_preference-
tuning-with-dpo/dpo-from-scratch.ipynb':
'614c8c692964a06da2756e7c0e72efaeaf6545240818f90a00b639810bd601e6', 'appendix-E/01_main-
chapter-code/appendix-E.ipynb':
'9dc10f802935650346aab967fb6cf6625f9071a4107f82060718f7d8534a2d98', 'pkg/
llms_from_scratch/generate.py':
'68fab6c9e99f0f7fab0ce3c59fd9fd302c64fe3a034a2504879dc72f2a2c3e34'}

torch.manual_seed(20261014)
torch.set_num_threads(1)
torch.set_default_dtype(torch.float64)

def extract(path,names,namespace):
    if path.suffix=='.ipynb':
        texts=[''.join(c['source']) for c in json.loads(path.read_text())['cells'] if
c['cell_type']=='code']
    else:texts=[path.read_text()]
    found={}
    for text in texts:
        try:tree=ast.parse(text)
        except SyntaxError:continue
        for node in tree.body:
            if isinstance(node,(ast.FunctionDef,ast.ClassDef)) and node.name in
names:found[node.name]=node
        if set(found)!=set(names):raise ValueError('missing reviewed definition')
    module=ast.Module(body=[found[n] for n in names],type_ignores=[])
    exec(compile(module,str(path),'exec'),namespace)
```

```

return namespace

def run(upstream):
    if (upstream.parent / 'SOURCE_MAP.json').exists():
        sys.path.insert(0, str(Path(__file__).resolve().parents[1]))
        from verify_runtime import verify_runtime
        manifest = verify_runtime(upstream.parent)
        expected_files = {row['path'].removeprefix('upstream/'): row['sha256'] for row
in manifest['files']}
        for row in manifest['files']:
            rel = row['path'].removeprefix('upstream/')
            if rel in PINNED_FILES and row['original_sha256'] != PINNED_FILES[rel]:
                raise ValueError('Original source provenance mismatch')
    else:
        expected_files = PINNED_FILES
    for relative in PINNED_FILES:
        expected = expected_files[relative]
        actual = hashlib.sha256((upstream / relative).read_bytes()).hexdigest()
        if actual != expected:
            raise ValueError(f'Pinned source hash mismatch: {relative}')
    if (upstream / '.git').exists():
        sha = subprocess.check_output(['git', 'rev-parse', 'HEAD'], cwd=upstream,
text=True).strip()
        if sha != PIN:
            raise ValueError('Pinned repository commit mismatch')

result={'commit':PIN,'torch':torch.__version__,'seed':20261014,'threads':torch.get_num_t
hreads()}
p=upstream/'ch07/02_dataset-utilities/find-near-duplicates.py'
ns=extract(p,['preprocess_text','find_near_duplicates'],
{'re':re,'TfidfVectorizer':TfidfVectorizer,'cosine_similarity':cosine_similarity})
good=[{'input':'same text'},{'input':'same text'}]
filtered,pairs=ns['find_near_duplicates'](good,.9,'input')
assert len(pairs)==1 and len(filtered)==1
empty_middle=[{'input':'same text'},{'input':''},{'input':'same text'}]
filtered_bad,pairs_bad=ns['find_near_duplicates'](empty_middle,.9,'input')
assert len(pairs_bad)==0 and len(filtered_bad)==3

result['near_duplicate_original_index_bug']={'no_empty_pairs':len(pairs),'empty_middle_p
airs':len(pairs_bad),'expected_empty_middle_pairs':1,'status':'confirmed negative
regression; upstream left unchanged'}
p=upstream/'ch07/04_preference-tuning-with-dpo/dpo-from-scratch.ipynb'
ns=extract(p,['compute_dpo_loss','compute_logprobs'],{'torch':torch,'F':F})
logits=torch.zeros(1,4,3);labels=torch.zeros(1,4,dtype=torch.long)
mask=torch.tensor([[False,True,True,False]])
measured=ns['compute_logprobs'](logits,labels,mask)
assert torch.allclose(measured,torch.tensor([-math.log(3)]))
empty=ns['compute_logprobs'](logits,labels,torch.zeros_like(mask))
assert torch.isnan(empty).all()
zero=torch.zeros(1)
loss,*_=ns['compute_dpo_loss'](zero,zero,zero,zero)
assert abs(loss.item()-math.log(2))<1e-12

result['dpo_upstream']={'reduction':'mean','uniform_logprob':measured.item(),'empty_mask
_returns_nan':True,'equal_policy_loss':loss.item()}
p=upstream/'appendix-E/01_main-chapter-code/appendix-E.ipynb'

```

```

ns=extract(p,['LoRALayer','LinearWithLoRA'],{'torch':torch,'math':math})
layer=ns['LoRALayer'](3,4,2,6)
with torch.no_grad():layer.A.fill_(1);layer.B.fill_(1)
y=layer(torch.ones(1,3))
assert torch.equal(y,torch.full((1,4),18.))
result['appendix_e_effective_scale']=3.0
p=upstream/'pkg/llms_from_scratch/generate.py'
ns=extract(p,['trim_input_tensor'],{})
trimmed=ns['trim_input_tensor'](torch.arange(10)[None],5,3)
assert trimmed.tolist()==[[7,8,9]]

result['trim_input_tensor']={'context':5,'new_tokens':3,'retained_prompt':trimmed.tolist
(),'final_output_length_if_budget_filled':6}
result['status']='all positive and expected-negative assertions passed'
return result

if __name__=='__main__':
    parser=argparse.ArgumentParser();parser.add_argument('--
upstream',type=Path,default=Path(__file__).resolve().parents[2]/'runtime_sources/
upstream');args=parser.parse_args()
    print(json.dumps(run(args.upstream),ensure_ascii=False,indent=2))

```

examples/finetuning/contracts.py

```

"""Original offline teaching experiments, not upstream training reproduction.
Seed 20261014; CPU; float64. No weights, datasets, APIs or model servers.
"""

import json
import math
import platform
from pathlib import Path
import torch
import torch.nn.functional as F

torch.manual_seed(20261014)
torch.set_num_threads(1)
torch.set_default_dtype(torch.float64)

def collate(rows, prompt_lengths, eos=9):
    """Lengths, not token identity, distinguish real EOS from padding."""
    width = max(map(len, rows))
    xs, ys = [], []
    for row, p in zip(rows, prompt_lengths):
        if not 0 <= p <= len(row):
            raise ValueError("invalid prompt length")
        full = row + [eos] * (width + 1 - len(row))
        x, y = full[:-1], full[1:]
        y = [v if p <= j + 1 <= len(row) else -100 for j, v in enumerate(y)]
        if all(v == -100 for v in y):
            raise ValueError("no supervised tokens")
        xs.append(x); ys.append(y)
    return torch.tensor(xs), torch.tensor(ys)

```

```

def selected_logprobs(logits, ids, keep, reduction="sum"):
    lp = logits[:, :-1].log_softmax(-1)
    selected = lp.gather(-1, ids[:, 1:, None]).squeeze(-1)
    mask = keep[:, 1:]
    counts = mask.sum(-1)
    if (counts == 0).any():
        raise ValueError("empty response after shift or truncation")
    values = (selected * mask).sum(-1)
    if reduction == "mean":
        values = values / counts
    elif reduction != "sum":
        raise ValueError("unknown reduction")
    return values

def dpo(chosen, rejected, ref_chosen, ref_rejected, beta=0.1):
    margin = chosen - rejected - (ref_chosen.detach() - ref_rejected.detach())
    return F.softplus(-beta * margin).mean()

def run():
    out = {"seed": 20261014, "torch": torch.__version__, "python":
platform.python_version(), "threads": torch.get_num_threads()}
    x, y = collate([[1, 2, 3], [4, 9, 5, 6]], [2, 2])
    assert y.tolist() == [[-100, 3, 9, -100], [-100, 5, 6, 9]]
    logits = torch.randn(2, 4, 10, requires_grad=True)
    loss = F.cross_entropy(logits.flatten(0, 1), y.flatten(), ignore_index=-100)
    loss.backward()
    assert logits.grad[y == -100].abs().sum() == 0
    assert logits.grad[y != -100].abs().sum() > 0
    wrong = y.masked_fill(y == 9, -100)
    out["mask_shift"] = {"inputs": x.tolist(), "targets": y.tolist(), "loss":
loss.item(), "supervised": int((y != -100).sum()), "wrong_drop_eos_supervised":
int((wrong != -100).sum())}

    ids = torch.tensor([[2, 3, 9, 9], [2, 9, 4, 9]])
    lengths = torch.tensor([2, 3])
    scores = torch.tensor([[[0., 0.], [4., 0.], [0., 1.], [0., 5.]], [[0., 0.], [0.,
4.], [5., 0.], [0., 5.]]])
    valid = scores[torch.arange(2), lengths - 1].argmax(-1)
    last = scores[:, -1].argmax(-1)
    inferred = (ids != 9).sum(-1) - 1
    assert valid.tolist() == [0, 0] and last.tolist() == [1, 1]
    assert inferred.tolist() != (lengths - 1).tolist()
    out["classification"] = {"last_valid_prediction": valid.tolist(),
"last_column_prediction": last.tolist(), "wrong_eos_length_positions":
inferred.tolist()}

    base = torch.nn.Linear(3, 4); head = torch.nn.Linear(4, 2)
    for p in base.parameters(): p.requires_grad_(False)
    optimizer = torch.optim.AdamW([p for p in list(base.parameters()) +
list(head.parameters()) if p.requires_grad], lr=.03)
    old = base.weight.detach().clone(); old_head = head.weight.detach().clone()
    optimizer.zero_grad(set_to_none=True)

```

```

F.cross_entropy(head(base(torch.randn(4, 3))), torch.tensor([0, 1, 0,
1])).backward()
optimizer.step()
assert base.weight.grad is None and torch.equal(base.weight, old)
assert not torch.equal(head.weight, old_head)
out["freeze"] = {"base_unchanged": True, "base_grad_none": True, "head_changed":
True, "optimizer_parameter_tensors": len(optimizer.param_groups[0]["params"])}
# Freezing alone does not erase an old gradient. Never leave stale gradients.
p = torch.nn.Parameter(torch.tensor([1.])); opt = torch.optim.SGD([p], lr=.1)
p.grad = torch.ones_like(p); p.requires_grad_(False); opt.step()
assert p.item() == .9
out["stale_gradient_negative"] = p.item()

x0 = torch.randn(2, 3); w = torch.randn(4, 3)
a = torch.nn.Parameter(torch.randn(3, 2)); b = torch.nn.Parameter(torch.zeros(2,
4)); scale = 3.
initial = x0 @ w.T + scale * x0 @ a @ b
assert torch.equal(initial, x0 @ w.T)
initial.square().sum().backward()
assert a.grad.abs().sum() == 0 and b.grad.abs().sum() > 0
with torch.no_grad(): b.add_(-.001 * b.grad)
branch = x0 @ w.T + scale * x0 @ a @ b
merged = x0 @ (w + scale * (a @ b).T).T
err = (branch - merged).abs().max().item()
assert err < 1e-12
out["lora"] = {"initial_equal": True, "initial_A_gradient":
a.grad.abs().sum().item(), "initial_B_gradient_positive": True, "merge_max_error": err,
"base_parameters": w.numel(), "adapter_parameters": a.numel() + b.numel()}

c = torch.tensor([0., -10000.], requires_grad=True); r = torch.tensor([0., 10000.],
requires_grad=True)
rc = torch.zeros(2, requires_grad=True); rr = torch.zeros(2, requires_grad=True)
dl = dpo(c, r, rc, rr); dl.backward()
assert torch.isfinite(dl) and rc.grad is None and rr.grad is None
assert dpo(torch.zeros(1), torch.zeros(1), torch.zeros(1), torch.zeros(1)).item() ==
math.log(2)
naive = -torch.log(torch.sigmoid(torch.tensor(-2000.)))
assert torch.isinf(naive)
ll = torch.zeros(2, 5, 3); ii = torch.zeros(2, 5, dtype=torch.long)
keep = torch.tensor([[False, False, True, True, False], [False, True, True, True,
True]])
sums = selected_logprobs(ll, ii, keep, "sum"); means = selected_logprobs(ll, ii,
keep, "mean")
assert torch.allclose(means, torch.full((2,), -math.log(3))) and sums[0] != sums[1]
empty_rejected = False
try: selected_logprobs(ll, ii, torch.zeros_like(keep))
except ValueError: empty_rejected = True
assert empty_rejected
out["dpo"] = {"stable_extreme_loss": dl.item(), "naive_is_infinite": True,
"reference_grad_none": True, "sum": sums.tolist(), "mean": means.tolist(),
"empty_response_rejected": True}

# Synthetic equivalence groups: graph edges 0--1--2, 3 alone.
edges = [(0, 1), (1, 2)]; parent = list(range(4))
def root(i):
    while parent[i] != i: i = parent[i]

```

```

        return i
    for i, j in edges: parent[root(j)] = root(i)
    groups = [root(i) for i in range(4)]
    naive_train, naive_test = {0, 2}, {1, 3}
    leaks = {groups[i] for i in naive_train} & {groups[i] for i in naive_test}
    safe_train, safe_test = {0, 1, 2}, {3}
    assert leaks and not ({groups[i] for i in safe_train} & {groups[i] for i in
safe_test})
    out["duplicates"] = {"components": groups, "row_split_leaking_components":
sorted(leaks), "group_split_disjoint": True}

    # Test streaming/context contract, not Chainlit or neural-network quality.
    def complete(prompt, budget, context):
        if not prompt or budget < 0 or len(prompt) + budget > context: raise
ValueError("context budget")
        state = list(prompt)
        for _ in range(budget):
            nxt = (state[-1] + 1) % 10; state.append(nxt); yield nxt
        one = list(complete([1, 2], 3, 5)); two = list(complete([7], 2, 5))
        assert one == [3, 4, 5] and two == [8, 9]
        rejected = False
        try: list(complete([1, 2, 3], 3, 5))
        except ValueError: rejected = True
        assert rejected
        out["ui_contract"] = {"streamed_ids": one, "second_session_ids": two,
"context_overflow_rejected": True}
        out["status"] = "all assertions passed"
        return out

if __name__ == "__main__":
    result = run()
    print(json.dumps(result, ensure_ascii=False, indent=2))

```

examples/finetuning/draw_figures.py

```

"""Original diagrams for chapters 24--30; no upstream artwork used."""
from pathlib import Path
import numpy as np
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
from matplotlib.patches import FancyBboxPatch
ROOT=Path(__file__).resolve().parents[2]
OUT=ROOT/'figures'/finetuning; OUT.mkdir(parents=True,exist_ok=True)
plt.rcParams.update({'font.size':12,'figure.facecolor':'white'})
def save(fig,name):
    fig.tight_layout(); fig.savefig(OUT/f'{name}.png',dpi=170); fig.savefig(OUT/
f'{name}.svg'); plt.close(fig)
def box(ax,x,y,w,h,text,color='#dfecf5'):

ax.add_patch(FancyBboxPatch((x,y),w,h,boxstyle='round,pad=.025',facecolor=color,edgecolo
r='#31516b'))

```

```

    ax.text(x+w/2,y+h/2,text,ha='center',va='center',fontsize=11)
def arrow(ax,a,b):ax.annotate('',xy=b,xytext=a,arrowprops={'arrowstyle':'-
>', 'lw':1.8, 'color':'#31516b'})
fig,ax=plt.subplots(figsize=(8,3.2)); ax.axis('off');ax.set(xlim=(0,9),ylim=(0,3))
for i,t in enumerate(['word 0','word 1','word 2','EOS pad','EOS
pad']):box(ax,.2+1.7*i,1.3,1.4,.6,t,'#fbe7cd' if i>=3 else '#dfecf5')
ax.text(3.1,.4,'last real token: index 2',ha='center');arrow(ax,(3.6,.7),
(4.3,1.25));ax.text(7.2,2.6,'last column: index 4',ha='center');arrow(ax,(7.5,2.35),
(7.7,1.95));ax.set_title('Classification needs an explicit readout
contract');save(fig,'classification')
fig,ax=plt.subplots(figsize=(8,3.2));ax.axis('off')
table=ax.table(cellText=[['input','1','2','3','9'],['shifted target','2','3','9','9'],
['answer-only target','-100','3','9','-100'],['role','prompt','answer','real
EOS','padding']],loc='center',cellLoc='center',colWidths=[.27,.16,.16,.16,.16]);table.au
to_set_font_size(False);table.set_fontsize(11);table.scale(1,2)
for key,c in
table.get_celld().items():c.set_edgecolor('white');c.set_facecolor('#dfecf5' if
key[0]%2==0 else '#f2f4f6')
ax.set_title('Loss mask follows shifted labels, not input token
identity');save(fig,'masking')
fig,ax=plt.subplots(figsize=(8,3.3));ax.axis('off');ax.set(xlim=(0,9),ylim=(0,3))
for i in range(4):box(ax,.5+i*2,1.6,1.7,str(i),'#dfecf5' if i<3 else '#fbe7cd')
arrow(ax,(1.6,1.95),(2.45,1.95));arrow(ax,(3.6,1.95),(4.45,1.95))
ax.text(3,.9,'component {0,1,2}: keep in one
split',ha='center');ax.text(7,.9,'{3}',ha='center');ax.text(4.4,.2,'Row-wise random
splitting can leak a connected component',ha='center');ax.set_title('Near-duplicate
edges define leakage groups');save(fig,'data_groups')
fig,ax=plt.subplots(figsize=(7,3.6));z=np.linspace(-8,8,200);ax.plot(z,np.logaddexp(0,-
z),label='softplus(-z)',color='#1d6787');ax.scatter([0],
[np.log(2)],color='#c66028');ax.annotate('policy = reference: log(2)',
(0,np.log(2)),xytext=(1,3),arrowprops={'arrowstyle':'->'});ax.set(xlabel='z = beta *
relative preference margin',ylabel='DPO loss');ax.grid(alpha=.2);save(fig,'dpo')
fig,ax=plt.subplots(figsize=(8,3.6));ax.axis('off');ax.set(xlim=(0,10),ylim=(0,4))
box(ax,.2,1.5,1,.7,'X');box(ax,2.4,2.7,2,.7,'frozen
W.T');box(ax,2.4,.5,1.3,.7,'A');box(ax,4.5,.5,1.3,.7,'B');box(ax,6.4,.5,1.2,.7,'scale
s');box(ax,8.1,1.5,1.4,.7,'sum')
for a,b in [((1.25,1.9),(2.35,3)),((1.25,1.7),(2.35,.9)),((3.75,.85),(4.45,.85)),
((5.85,.85),(6.35,.85)),((7.65,.9),(8.1,1.7)),((4.45,3),(8.7,2.25))]:arrow(ax,a,b)
ax.text(5,3.8,'W_merged = W + s (A B).T',ha='center');save(fig,'lora')
fig,ax=plt.subplots(figsize=(9,3.4));ax.axis('off');ax.set(xlim=(0,12),ylim=(0,3.4))
labels=['message','template /\ntokenizer','budget +\nsession state','model +
\ndecoding','display']
for i,t in enumerate(labels):box(ax,.1+i*2.4,1.5,2,.85,t);
for i in range(4):arrow(ax,(2.15+i*2.4,1.92),(2.45+i*2.4,1.92))
ax.text(6,.5,'Read-only weights may be shared; mutable caches belong to each
session',ha='center',fontsize=11);ax.set_title('UI is a contract around inference, not a
new model capability');save(fig,'ui')

```