

NANO GPT • TRAINING ENGINEERING

把 GPT 训练起来

nanoGPT 全仓工程精读

从数据与配置到训练、恢复和采样
沿着源码理解完整训练系统

原创中文学习书 · 机制、推导、代码与实验
基于 Andrej Karpathy 的 nanoGPT 公开源码

目录

把 GPT 训练起来：nanoGPT 全仓工程精读.....	6
怎样使用这本书	6
阅读路线	7
先运行哪个程序	7
查资料与改实验	7
第 1 章 沿一次训练看完整仓库	8
1.1 我们到底要把什么训练起来	8
1.2 训练的一个最小闭环	8
1.3 文件地图与阅读顺序	9
1.4 哪些结果已经实际运行	9
1.5 练习：先建立可核对的问题	10
第 2 章 字符与 BPE：三条数据准备流水线.....	11
2.1 同一段文字，token 可以完全不同.....	11
2.2 字符版：排序决定编号稳定性.....	11
2.3 Shakespeare BPE：复用 GPT-2 词表的必要性.....	12
2.4 OpenWebText：文档结束符与连续存储.....	12
2.5 uint16 是协议，也是一个硬边界	13
2.6 练习与检查	13
第 3 章 配置覆盖、类型检查与执行边界.....	14
3.1 一次运行为什么会有三份“配置”	14
3.2 命令行的类型检查到底检查了什么	14
3.3 把开关按它真正影响的东西分类	15
3.4 七份配置，各自省略了什么	16
3.5 练习：写出最终值，而不是凭文件名判断.....	16
3.6 默认值逐项参考	16
第 4 章 GPTConfig、参数规模与初始化	19
4.1 一个配置怎样变成一组矩阵	19
4.2 不靠打印的 M 数，自己数一遍.....	19
4.3 初始值如何进入残差网络.....	20
4.4 改结构时，先看哪个成本增长.....	20
第 5 章 逐层推导因果注意力、MLP 和残差	22
5.1 LayerNorm 沿哪个维度计算.....	22
5.2 Q、K、V 是三次线性投影，合成一次执行	22

5.3 掩码必须在 softmax 之前作用	23
5.4 名为 flash，不保证用了 FlashAttention 内核	24
5.5 MLP 改每个位置，注意力连接位置	24
第 6 章 权重共享、损失、预训练映射与上下文裁剪	26
6.1 从编号到每个位置的预测	26
6.2 交叉熵为什么要把前两维展平	26
6.3 共享不是拷贝，梯度会从两个角色汇合	27
6.4 从 GPT-2 导入时，名字和形状都要对齐	27
6.5 裁长度会换 Parameter，时机很重要	27
第 7 章 memmap 批次与下一 token 对齐	29
7.1 原始二进制没有自我说明能力	29
7.2 最后一扇窗口为什么从 N-T-1 开始	29
7.3 为什么磁盘上是 uint16，模型前是 int64	30
7.4 训练批次与验证批次共享随机数发生器	30
第 8 章 训练主循环、梯度累积、混合精度与裁剪	31
8.1 一步到底包含几次前向	31
8.2 优化器为什么分成两组	31
8.3 AMP 改计算精度，不等于把所有状态都减半	32
8.4 先还原梯度，再裁剪，再更新	32
8.5 日志中 loss 与时间的细节	33
第 9 章 学习率、验证、checkpoint 与恢复	34
9.1 逐段读学习率，而不是只说“余弦退火”	34
9.2 验证估计了什么	35
9.3 保存时机、best 字段与最后一步	35
9.4 checkpoint 有什么，没有什么	35
9.5 可信文件与版本边界	36
第 10 章 单机多卡与跨节点 DDP 的状态约束	37
10.1 多张卡复制模型，分担的是数据	37
10.2 配置里的累积次数先除以 world size	37
10.3 seed 不同，权重为什么仍应相同	38
10.4 为何只在最后一个微步同步	38
10.5 跨节点时，先核对进程数和通信条件	39
第 11 章 生成接口：温度、top-k、提示和随机性	40
11.1 采样从一个非空提示开始	40
11.2 一次循环只增加一个 token	40
11.3 温度改变概率比，不改变正温度下的排序	40
11.4 top-k 的阈值实现有平分边界	41

11.5 eval、no_grad 和 seed 各做一件事	41
11.6 采样入口的全部默认开关	42
第 12 章 微调与 GPT-2 基线：域差异如何影响比较	44
12.1 微调复用了哪一部分训练程序	44
12.2 配置注释中的 epoch 是估算	44
12.3 基线要对齐数据域与 tokenizer	44
12.4 小数据微调的失败方式	45
第 13 章 benchmark、MFU 与吞吐口径	46
13.1 bench.py 故意删掉了什么	46
13.2 每秒 token 数需要说明分子和分母	46
13.3 FLOPs 公式从哪里来	46
13.4 DDP 的 MFU 分子应与分母匹配	47
13.5 profiler 能回答另一类问题	47
13.6 benchmark 的全部默认开关	48
第 14 章 参数估算与 scaling laws：逐单元核对	49
14.1 先把 notebook 当成待验证的计算记录	49
14.2 sizing：精确参数、近似 FLOPs 与假设时间	49
14.3 scaling：参数口径中藏着的差异	51
14.4 scaling 的 FLOPs 不计哪两项	51
14.5 Approach 3：从固定预算推最优 N、D	51
14.6 Approach 2：对表格作回归，不是重新训练模型	53
14.7 练习：公式一致不等于问题一致	55
第 15 章 小规模复现、故障定位与仓库的历史边界	56
15.1 一条能在普通 CPU 上走完的路线	56
15.2 四个完整程序各负责一件事	57
15.3 真实输出怎样支持结论	57
15.4 按失败发生的位置排查	58
15.5 仓库外围文件也有工程含义	59
15.6 能继续研究的边界	59
参考资料索引	61
第 1 章 沿一次训练看完整仓库	61
第 2 章 字符与 BPE：三条数据准备流水线	61
第 3 章 配置覆盖、类型检查与执行边界	61
第 4 章 GPTConfig、参数规模与初始化	62
第 5 章 逐层推导因果注意力、MLP 和残差	62
第 6 章 权重共享、损失、预训练映射与上下文裁剪	62
第 7 章 memmap 批次与下一 token 对齐	62

第 8 章 训练主循环、梯度累积、混合精度与裁剪	63
第 9 章 学习率、验证、checkpoint 与恢复	63
第 10 章 单机多卡与跨节点 DDP 的状态约束	63
第 11 章 生成接口：温度、top-k、提示和随机性	63
第 12 章 微调与 GPT-2 基线：域差异如何影响比较	63
第 13 章 benchmark、MFU 与吞吐口径	64
第 14 章 参数估算与 scaling laws：逐单元核对	64
第 15 章 小规模复现、故障定位与仓库的历史边界	64
完整实验代码	65
examples/check_notebooks.py	65
examples/make_figures.py	66
examples/tiny_pipeline.py	67
examples/verify_mechanics.py	70

把 GPT 训练起来：nanoGPT 全仓工程精读

怎样使用这本书

本书从一次训练请求出发：配置怎样决定模型和数据，token 怎样组成批次，损失怎样变成参数更新，检查点保存了什么，以及生成阶段怎样使用训练结果。目标不是记住几个启动命令，而是能沿着源码解释一次成功或失败的运行。

范围固定在 nanoGPT 提交 3adf61e154c3fe3fca428ad6bc3818b27a3b8291。该版本 README 已说明项目弃用并指向 nanocat。本书仍聚焦这一份 nanoGPT 的实现；它有利于学习小而完整的训练系统，但不把历史安装建议、硬件报告或依赖版本包装成当前生产推荐。

正文同时区分三种证据：固定版本源码直接表达的行为、本书真实运行的微型 CPU 实验，以及依赖 GPU 或多节点的未运行路径。没有下载 GPT-2 大模型权重或完整预训练语料；原 README 报告的损失和速度只代表上游在其条件下的结果。

读者可从数据准备与模型开始，随后运行最小训练、恢复和采样，再回头检查梯度累积、调度和检查点的边界。完整代码与运行条件附后。复用的上游代码保留 Andrej Karpathy 的版权和 MIT 许可，本书的解释、图和补充实验明确标注。

项目入口：[nanoGPT](#)。

阅读路线

这本书沿着一次真实训练的执行顺序展开。先把输入和状态认清，再讨论速度与扩展。每章都可以独立回查；第一次阅读建议按顺序进行。

阶段	章节	阅读后应能回答的问题
搭起整体认识	1—3	数据在哪里变成 token？配置怎样进入程序？
解释模型	4—6	参数怎样组织？注意力为何不看未来？权重为何共享？
走完训练闭环	7—9	标签如何错位？一次更新看了多少数据？恢复了哪些状态？
扩展与使用	10—12	多卡如何改变全局批次？生成和微调各需要什么？
判断成本与结果	13—15	基准测了什么？规模公式依赖什么假设？怎样复现与排错？

先运行哪个程序

读完第 7—9 章后，按第 15 章准备独立环境，运行 `examples/tiny_pipeline.py`。这条微型 CPU 路线包含训练、保存、恢复和采样。随后运行 `verify_mechanics.py` 检验因果性、梯度累积等关键性质；读第 14 章时运行 `check_notebooks.py` 复算公式，最后用 `make_figures.py` 重绘本书图形。

本书用合成小数据验证程序机制。微型验证损失下降，不等于复现了 GPT-2 的语言能力。GPU、多节点、大语料与预训练权重路径的未验证边界，在相关章节中分别注明。

查资料与改实验

正文在对应观点附近提供固定提交源码和原始论文链接，书末再集中列参考资料。完整原创实验代码在附录逐文件列出；ZIP 中还保留运行所需的固定上游源码及许可。首次实验保留默认断言，之后一次只改一个条件，记录结果和首次不一致的位置。

第 1 章 沿一次训练看完整仓库

1.1 我们到底要把什么训练起来

假设手里只有一段文字，希望模型读到“ab”以后，能给下一个字符一个概率分布。模型不直接保存一张“前缀到答案”的字典，而是把文字变成整数、把整数查成向量，再经过多层计算。训练时，程序知道每个位置后面实际出现的 token，用它来纠正模型。生成时，程序从模型给出的概率分布中抽取一个 token，接回输入，继续预测。

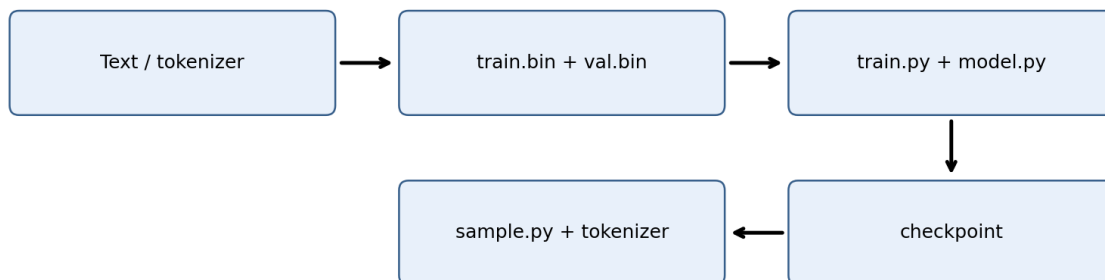
nanoGPT 把这条链分成了几个文件。`data/*/prepare.py` 负责把文字写成整数文件，`model.py` 定义可求导的计算，`train.py` 组织数据、梯度、优化器和保存，`sample.py` 负责把训练结果再变成文字。最值得学习的是这些文件之间的约定：训练脚本相信二进制文件使用正确的整数类型，模型相信编号没有超过词表，采样脚本相信解码器与训练时一致。任何一处“各自看起来没错”的改动，都可能破坏整条链。[固定版本仓库](#)

本书讲解固定提交 `3adf61e154c3fe3fca428ad6bc3818b27a3b8291`。该版本 README 已在 2025 年 11 月注明项目旧且已弃用，并指向 nanochat。本书保留这个历史边界：我们用 nanoGPT 学会核对一个训练系统，不把它当作持续更新的生产训练框架，也不悄悄把章节改写成 nanochat 使用说明。

1.2 训练的一个最小闭环

全书用同一组符号。 B 是一个微批次中的序列数， T 是序列长度， V 是词表大小， C 是每个位置的特征维数， L 是 Transformer 层数， H 是注意力头数。输入 x 和目标 y 都是形状 (B, T) 的整数张量。模型产生 (B, T, V) 的 logits，即尚未归一化的分数。

例如整数流为 `[2, 3, 1, 4, 2]`，取长度为 4 的窗口：输入为 `[2,3,1,4]`，目标为 `[3,1,4,2]`。一个前向传播同时产生四道“下一个 token 是什么”的题。因果掩码保证第一道题看不到后面的答案，但训练仍能把四道题并行计算。交叉熵把分数压缩成一个损失，反向传播求参数梯度，AdamW 用梯度和历史统计量更新参数。这才是一次优化器更新。



图中 checkpoint 并不是一个只装权重的袋子。它还装有优化器状态、模型结构参数、迭代编号和部分训练配置；但没有保存完整随机数状态。第 9 章会用实际恢复实验说明，这个差别为什么会让“能接着训练”与“逐位复现未中断训练”成为两个问题。

1.3 文件地图与阅读顺序

文件或目录	它回答的问题	重点章节
data/ 三条流水线及说明	token 从哪里来，如何切分和写盘	2、7
configurator.py 与 7 个配置文件	谁可以覆盖默认值，何时生效	3、10、12
model.py	每层算什么，参数如何组织	4—6、11、13
train.py	哪一步读取数据、求导、评估和保存	7—10
sample.py	如何加载并生成，如何还原文字	11
bench.py	测的到底是哪段时间	13
两份 notebook	参数、算力和数据规模怎样估算	14
README、许可、Git 配置及资产	历史报告、分发和证据边界	1、15

仓库不是按执行顺序排列的课程。直接从 `train.py` 第一行读到最后，容易在不熟悉模型时陷入配置细节，也容易在没有数据约定时误读损失。本书先确认数据和配置，再拆模型，再回到训练循环。每章都能指回源文件；第 14 章把两份 notebook 的 43 个单元逐一登记，第 15 章给出完整测试入口和覆盖台账。

1.4 哪些结果已经实际运行

本书在 Python 3.12.14、PyTorch 2.6.0+cpu、NumPy 2.3.5 上使用单 CPU 线程验证。完整入口 `examples/tiny_pipeline.py` 会复制四份固定源码到隔离目录，构造五字符合成语料，运行原版训练、恢复和采样。训练使用 $L = 2, C = 16, H = 2, T = 8, B = 2$ ，每次更新累积两个微批次；未下载权重或语料，未调用在线日志。

真实输出中，初始验证损失为 1.6728，step 20 为 1.0560，恢复后的 step 25 为 0.9835。这说明程序能在这组简单模式上降低交叉熵，也说明保存、加载和生成链路接通了。它没有证明模型会写自然语言，更没有复现 README 报告的 GPT-2 训练。GPU、NCCL、混合精度和大规模性能在本书中是源码分析，不是实测结论。

读者可先运行闭环，再回来逐章解释每个开关：

```
python examples/tiny_pipeline.py
python examples/verify_mechanics.py
python examples/check_notebooks.py
```

配套包包括固定源码及 MIT 许可，路径均相对于本书根目录；依赖版本和安装边界见第 15 章。所有书中新实验、图解、推导和习题均为教学补充。原始 notebook 的已有输出若未经此次重跑，不作为本次实测。

1.5 练习：先建立可核对的问题

补充练习。 修改模型宽度后，旧 checkpoint 一定能恢复吗？只把 `sample.py` 的提示换成中文，又可能在哪里失败？

提示。 先问“谁定义了张量形状”，再问“谁定义了字符编号”。

答案。 恢复训练时，脚本会用 checkpoint 覆盖宽度等结构参数，所以命令行改宽度通常不会产生新结构；手工强行加载不匹配的权重会报形状错误。字符模型的 `stoi` 只包含训练字符，中文提示若不在其中会触发 `KeyError`。这两类失败分别发生在参数协议和编码协议，不能都归为“模型不够大”。

本章来源

- 固定 README：定位、运行路径、历史数字与弃用通知
- 固定 `train.py`
- 本书实测：`qa/author/tiny_pipeline.json` 及 `tiny_train.log`、`tiny_resume.log`、`tiny_sample.log`。

第 2 章 字符与 BPE：三条数据准备流水线

2.1 同一段文字，token 可以完全不同

模型处理的是整数编号，编号的意义由编码器规定。字符编码把一个字符变成一个编号；GPT-2 的 BPE 把字节序列按已有词表组合成 token。BPE token 不等于一个词，字符 token 也不等于一个 UTF-8 字节。若比较两个模型“看了多少 token”或“每 token 损失多少”，必须先确认它们使用相同编码。

固定仓库提供三条准备流水线。两条使用同一份 Shakespeare 文本，区别在编码方式；另一条按文档处理 OpenWebText。它们最后都写出 `train.bin` 和 `val.bin`，但文件名相同不意味着语义可互换。三条数据目录

流水线	切分单位与方式	编码及边界	额外元数据
shakespeare_char	文本前 90% / 后 10%	排序后的唯一字符集；无额外终止符	meta.pkl 存词表、正反映射
shakespeare	文本前 90% / 后 10%	GPT-2 <code>encode_ordinary</code> ；无额外终止符	不写字符映射
openwebtext	文档随机切分，seed 2357，验证比例 0.0005	每文档独立 BPE，尾部添加 EOT，再串接	中间数据包含 <code>ids</code> 和 <code>len</code>

2.2 字符版：排序决定编号稳定性

`chars = sorted(list(set(data)))` 先去重再排序。排序很重要：集合的遍历顺序不应成为模型词表的隐含随机源。之后 `stoi` 把字符映到整数，`itos` 把整数映回字符；`encode` 是逐字符查表，`decode` 是逐编号查表并拼接。

用本书合成词表 `"\n abc"` 举例，换行是 0，空格是 1，a、b、c 分别是 2、3、4。`encode("ab c")` 得到 `[2,3,1,4]`，反向还原必须得到原字符串。这里的整数没有大小语义：4 不代表比 2 更强、更晚或更接近另一个字符。嵌入层会为它们分别学习向量。

原脚本先从完整文本建立字符集，再切分训练和验证。这意味着验证集中的字符身份参与了词表定义，但验证序列没有作为训练目标。这种做法不等于完全严格的“仅从训练集拟合预处理”；如果研究要求严格隔离，应由训练集定义词表，并为未见字符设计未知符号策略。原脚本没有未知符号，查询缺失字符会直接报错。字符脚本

字符版原脚本读写 `input.txt` 时没有显式指定编码，而 BPE 版指定 UTF-8；在默认编码不同的系统上，这也是需要固定的输入条件。OpenWebText 的大型处理主体放在 `if __name__ == "__main__"` 下，适配其多进程运行方式，但编码器初始化仍在文件顶层，不能把 `import` 当成毫无副作用的动作。

切分点为 $s = \lfloor 0.9n \rfloor$ ，其中 n 是 Python 字符串长度。训练读取 `data[:s]`，验证读取 `data[s:]`。两边各自编码、各自写文件，因此训练窗口不会跨入验证集。不过戏剧的连续内容可能高度相关；这是一种固定的顺序留出实验，不是跨作者、跨作品的泛化评估。

脚本最后用 `pickle` 保存 `vocab_size`、`stoi`、`itos`。这三个字段分别供训练构造输出层、采样编码提示、采样解码生成编号。`pickle` 能执行反序列化逻辑，不能读取陌生人发来的未审查 `meta.pkl`。本书只加载自己刚写出的元数据。

2.3 Shakespeare BPE：复用 GPT-2 词表的必要性

BPE 版在原始字符串上完成 90/10 切分，再调用 `enc.encode_ordinary`。这使两边的 token 化各自独立；切分点附近的 token 边界不一定等于“先整体编码再切 token 数量”的结果。`encode_ordinary` 把看似特殊 token 的文字当普通内容处理，不自动把它识别成控制符。

为什么微调 GPT-2 要用这个版本？预训练嵌入矩阵的第 i 行已经对应 GPT-2 的第 i 个 token。如果拿字符编号代替 BPE 编号，即使所有编号都在 0 到 50256 之间、程序完全不报错，模型看到的仍是另一种语言编码。形状检查不能发现这种语义错位。

此脚本不写 `meta.pkl`，采样脚本因而默认回到 GPT-2 BPE。默认值建立在“确实使用 GPT-2 编码”这个前提下。自制另一套 BPE 后仅省掉元数据，并不会让采样程序自动推断你的编码器。[BPE 脚本](#)

两个 Shakespeare 下载器都采用“文件存在就不再下载”的策略，没有校验 HTTP 状态、哈希和内容类型。错误页面可能被保存为 `input.txt`，之后仍能被编码和训练。正式准备数据时应先验证响应、编码、许可证和内容摘要；本书不执行这条联网路径，而用自建小语料验证下游协议。

2.4 OpenWebText：文档结束符与连续存储

`process(example)` 有两个输出：`ids` 是正文的 token 列表加一个 EOT，`len` 是加完后的长度。EOT 是一个可预测的符号，不是一堵自动隔离注意力的墙。文档被串接后，窗口可能跨过 EOT；因果注意力仍能访问窗口内前一篇文档。若任务要求文档间绝不互看，需要额外的注意力边界逻辑，原仓库没有实现。

脚本用 Hugging Face 的 `train_test_split` 按文档打乱并分出验证集，再把 `test` 重命名为 `val`。`.map(process, remove_columns=['text'])` 把原始文本替换为编号及长度。`num_proc` 控制 token 化进程数量，`num_proc_load_dataset` 控制加载阶段并发；它们不是训练 GPU 数量，也不应凭“越大越快”盲目设置。[OpenWebText 准备脚本](#)

写盘先用 `np.uint64` 求总长度，避免几十亿 token 的计数溢出；再建立形状为总 token 数的一维 `memmap`。1024 个连续 shard 依次转为 NumPy 数据、拼接 `ids`、写入 `[idx:idx+len(arr_batch)]`，随后更新 `idx`。这里的 `idx` 是已写 token 数，而不是字节偏移；每个 `uint16` 占两个字节。最后 `flush()` 将修改刷到文件。

固定 1024 个 shard 适合上游大数据路径，却不是任意小数据都能照用的常数。若把数据缩到几个文档，某些 shard 可能为空，`np.concatenate` 就可能没有数组可拼。缩小实验必须同时调整分片方式，不能只把下载数量改小。

2.5 uint16 是协议，也是一个硬边界

无符号 16 位整数可表示 0 到 $2^{16} - 1 = 65535$ 。若词表编号从 0 连续排列，则词表大小最多为 65536；若编号稀疏，应检查最大编号而不是只查词表条目数。GPT-2 最大编号 50256 在范围内。将更大词表强行转为 `uint16` 可能发生溢出、回绕或转换错误，取决于转换入口及 NumPy 版本。本书明确测试了先建 `int64` 再 `.astype(np.uint16)`：65536 变成了 0。这会把一种 token 静默改成另一种。

如果改成 `uint32`，准备脚本和所有读取器都必须一起修改，包括 `train.py` 与 `bench.py`。只改写入端会让读取器把一个 32 位编号拆成两个 16 位编号，损失仍可能下降，却是在训练损坏的数据。

上游三个 readme 报告的 token 数是历史语料结果：字符版训练 1,003,854、验证 111,540；BPE Shakespeare 训练 301,966、验证 36,059；OWT 训练约 90 亿、验证约 443 万。这些数字用于解释存储规模，不是本书重新下载所得。

2.6 练习与检查

补充练习 1。 为什么 `decode(encode(s)) == s` 通过，还不能证明训练和采样词表一致？

答案。 每套自洽映射都可能往返成功。还要比较训练时保存的映射与采样实际加载的映射，最好记录词表文件哈希；交换两个 token 的编号仍能保持各自的往返性质。

补充练习 2。 训练文件有 7000 个 `uint16` token，理论原始文件长度是多少？改为 `uint32` 呢？

答案。 分别为 14000 和 28000 字节，不含任何头部。`tofile` 写的是原始数组，不携带 `dtype`、端序、词表或形状说明，因此应另外保存这些约定。完整本地数据生成代码见 `examples/tiny_pipeline.py`，边界反例见 `examples/verify_mechanics.py`。

本章来源

- 字符准备与 `encode/decode`
- Shakespeare BPE 准备
- OpenWebText 准备与 `process`
- tiktoken 官方实现

第 3 章 配置覆盖、类型检查与执行边界

3.1 一次运行为什么会有三份“配置”

`train.py` 开头先定义普通 Python 全局变量，随后执行 `configurator.py`，最后把选中的全局值收集成 `config` 字典。配置不是独立对象：脚本后面直接使用 `batch_size`、`learning_rate` 等名字。默认值、配置文件、命令行参数依次覆盖，顺序决定最终结果。[configurator.py](#)

例如：

```
python train.py config/train_shakespeare_char.py --batch_size=12 --device=cpu --
dtype=float32 --compile=False
```

先载入默认 `batch_size=12`，配置文件改为 64，末尾参数又改回 12。如果把同一个参数放在配置文件之前，后读的配置文件会覆盖它。这里没有“命令行永远优先”的特殊层级，只有从左到右执行。

`config_keys` 在执行配置之前，从全局变量中挑选非下划线开头、类型属于整数、浮点数、布尔或字符串的键。之后建立的 `config` 是这些键的快照。配置文件新引入的变量未必被记录；DDP 稍后再修改的每进程累积次数也不会反写这个快照。日志中的原始设置和运行时派生值有意分开，但读 checkpoint 时要知道自己看的是哪一个。

3.2 命令行的类型检查到底检查了什么

包含等号的参数必须以 `--` 开头，随后 `key, val = arg.split('=')`。`literal_eval` 尝试把字符串变成 Python 字面值；失败时才保留原字符串。最后检查 `type(attempt) == type(globals()[key])`。

输入	若原值类型如下	实际结果
<code>--batch_size=3</code>	<code>int</code>	通过，得到整数 3
<code>--batch_size=3.0</code>	<code>int</code>	<code>AssertionError</code>
<code>--compile=False</code>	<code>bool</code>	通过
<code>--compile=false</code>	<code>bool</code>	小写不是 Python 布尔字面值，成为字符串，类型不符
<code>--unknown=3</code>	键不存在	<code>ValueError</code>
<code>--start=a=b</code>	<code>str</code>	拆成三段，解包失败

表中反例已由 `examples/verify_mechanics.py` 执行，不是按理想解析器推测。外层 shell 去掉引号以后，参数里依然包含第二个等号，因此简单加引号不能修复最后一行；提示中需要等号时可使用 `FILE:` 文件方式。

类型检查不是范围检查。`batch_size=-1`、`temperature=0.0`、`top_k=0` 都可能通过各自的类型关，却在后续分配或采样时失败。也不是安全校验：配置文件走的是 `exec`，可以导入模块、写文件或执行任意 Python。`literal_eval` 只约束单个命令行值，不能给配置文件提供沙箱。只运行可信、审查过的配置；来自网页或陌生仓库的“配置”应按代码处理。

3.3 把开关按它真正影响的东西分类

组别	源码字段	谁使用它
数据与批次	<code>dataset</code> , <code>batch_size</code> , <code>block_size</code> , <code>gradient_accumulation_steps</code>	文件路径、取窗、模型长度、累积循环
模型	<code>n_layer</code> , <code>n_head</code> , <code>n_embd</code> , <code>dropout</code> , <code>bias</code>	新建模型；恢复时部分被 checkpoint 覆盖
优化器	<code>learning_rate</code> , <code>weight_decay</code> , <code>beta1</code> , <code>beta2</code> , <code>grad_clip</code>	AdamW、学习率和梯度裁剪
日程	<code>max_iters</code> , <code>decay_lr</code> , <code>warmup_iters</code> , <code>lr_decay_iters</code> , <code>min_lr</code>	停止条件和逐步学习率
评估与保存	<code>out_dir</code> , <code>eval_interval</code> , <code>eval_iters</code> , <code>eval_only</code> , <code>always_save_checkpoint</code> , <code>init_from</code>	初始化、验证与 checkpoint
系统	<code>backend</code> , <code>device</code> , <code>dtype</code> , <code>compile</code>	分布式、设备、自动混合精度和编译
在线日志	<code>wandb_log</code> , <code>wandb_project</code> , <code>wandb_run_name</code>	仅启用时导入和连接 W&B

`dtype='float32'` 在本书 CPU 测试中是明确配置。原脚本 CPU 路径使用 `nullcontext()`，不会因为选择了 `bfloat16` 就自动把模型变成 BF16。默认 `dtype` 是根据 CUDA 支持情况决定的；没有 CUDA 时默认落入 `float16`，随后 `GradScaler` 可能给出禁用警告。把设备、`dtype` 和编译开关一起明确写出，比靠默认值猜行为容易审计。

3.4 七份配置，各自省略了什么

字符训练配置把层数、头数、宽度设为 6、6、384，长度 256，批次 64，累积 1，dropout 0.2；学习率 10^{-3} ，最小 10^{-4} ，warmup 100，训练上限与衰减终点都是 5000，`beta2=0.99`。它每 250 步评估，200 个批次估计损失，每 10 步打印，只在验证改善时保存。`bias`、优化器其他字段仍从主脚本继承，因此不能把配置文件单独当作完整实验说明。

GPT-2 训练配置设 `gradient_accumulation_steps=40`、`B=12`、`T=1024`，600000 的训练与衰减终点，每 1000 步评估，评估 200 批，每 10 步打印，权重衰减 0.1。它主动启用 W&B，读者运行前若不想外联必须追加 `--wandb_log=False`。第 10 章解释 40 如何在 8 卡下变成每进程 5 次。

四份评估配置只有模型型号不同：`gpt2`、`gpt2-medium`、`gpt2-large`、`gpt2-xl`；共同设批次 8、评估 500 批、`eval_only=True`、关闭 W&B。它们没有指定数据目录，继承默认 OWT。微调配置指定 Shakespeare BPE、`gpt2-xl`、批次 1、累积 32、最多 20、固定学习率 3×10^{-5} ，每 5 步评估 40 批，只保存改善的模型。它没有把 dropout 改成 0.1，仍继承默认 0.0。注释建议与实际赋值应分开读。[配置目录](#)

3.5 练习：写出最终值，而不是凭文件名判断

补充练习。 执行 `python train.py --batch_size=4 config/train_shakespeare_char.py --n_embd=128` 后，批次、宽度和头数各是多少？能开始模型前向吗？

答案。 批次 64、宽度 128、头数 6。配置文件覆盖了先前批次，最后参数只覆盖宽度。128 不能被 6 整除，注意力初始化断言失败。修复时应连同头数调整，例如 4；配置正确解析不代表结构有效。

3.6 默认值逐项参考

下面是固定 `train.py` 的全部 33 项默认配置，读者可用它补齐覆盖文件中没写出的值。它们是源码事实，不是面向任意硬件的推荐参数。相同字段在 `sample.py`、`bench.py` 有各自默认值，不能跨入口推断。

字段	固定源码默认值	含义及边界
<code>out_dir</code>	<code>'out'</code>	checkpoint 目录；复用旧目录前先确认 <code>init_from</code>
<code>eval_interval</code>	2000	相邻评估的迭代间隔；影响开销、保存和 RNG
<code>log_interval</code>	1	打印间隔；过密日志会增加同步与 I/O
<code>eval_iters</code>	200	每个 split 的抽样批次；精度与耗时折中

字段	固定源码默认值	含义及边界
eval_only	False	仅在 iter_num=0 的原版分支退出，见 9.3
always_save_checkpoint	True	每次评估都可保存；best 字段语义见 9.3
init_from	'scratch'	scratch、resume 或 gpt2*；控制权重及优化器来源
wandb_log	False	启用会外联日志服务；本书始终关闭
wandb_project	'owt'	在线日志项目名；关闭日志时不使用
wandb_run_name	'gpt2'	在线日志运行名；不决定本地输出目录
dataset	'openwebtext'	data 下的子目录名；编码必须与模型一致
gradient_accumulation_steps	5 * 8	DDP 分配前的总累积数 A；必须整除 W
batch_size	12	每进程每微步序列数；不是全局 batch
block_size	1024	每序列长度；限制窗口和位置表
n_layer	12	block 数；主干成本近似线性增长
n_head	12	注意力头数；宽度须能整除它
n_embd	768	残差宽度 C；主矩阵成本约按 C 平方增长
dropout	0.0	训练态随机丢弃比例；评估态关闭
bias	False	Linear/LN 是否带偏置；GPT-2 导入强制 True
learning_rate	0.0006	最大或固定学习率；每步写入所有参数组
max_iters	600000	原版是更新后检查 >；有多执行一步边界
weight_decay	0.1	矩阵参数组的 AdamW 解耦衰减
beta1	0.9	Adam 一阶矩平滑系数
beta2	0.95	Adam 二阶矩平滑系数
grad_clip	1.0	全局梯度范数阈值；0.0 关闭裁剪

字段	固定源码默认值	含义及边界
decay_lr	True	是否调用 warmup+cosine; False 用固定值
warmup_iters	2000	warmup 分支终点; 首步不是零
lr_decay_iters	600000	余弦终点; 须大于 warmup
min_lr	6e-05	衰减结束后保持的学习率
backend	'nccl'	进程组后端; 改 gloo 不自动支持 CPU DDP
device	'cuda'	单进程设备; DDP 会改为本地 CUDA 卡
dtype	支持 CUDA BF16 则 bfloat16, 否则 float16	自动混合精度选择; CPU 不使用 autocast
compile	True	是否 torch.compile; 本书关闭, 不测编译收益

七份配置的在线日志字符串也只是标签: 字符训练使用项目 `shakespeare-char`、运行名 `mini-gpt`、输出 `out-shakespeare-char`; GPT-2 训练使用 `owt`、`gpt2-124M`, 输出仍继承 `out`; 微调使用 `shakespeare`、由当前时间拼出的 `ft-...`、输出 `out-shakespeare`。时间戳只改变日志名, 不设随机 `seed`。四份评估文件不定义输出路径和项目名, 继续继承默认。逐项赋值、行号及原表达式见配套 `research/author/coverage.json` 的 `config_assignments`, 可以核对是否遗漏继承项。

本章来源

- `configurator.py` 的执行与解析
- `train.py` 默认配置与 `config_keys`
- 七份原始配置
- 本书解析反例: `qa/author/mechanics.json`。

第 4 章 GPTConfig、参数规模与初始化

4.1 一个配置怎样变成一组矩阵

GPTConfig 是 dataclass，保存长度、词表、层数、头数、宽度、dropout 和 bias。默认值描述一个 GPT-2 小模型量级的结构： $T_{\max} = 1024, V = 50304, L = 12, H = 12, C = 768$ 。其中 50304 是把 GPT-2 的 50257 向上补到 64 的倍数，不是另一套已经定义了 50304 个有效 token 的编码器。多出的输出行仍会参加 softmax，解码时未必有对应 token。GPTConfig 与 GPT 初始化

GPT.__init__ 先构造 ModuleDict：wte 是 token embedding，wpe 是绝对位置 embedding，drop 是输入 dropout，h 是长度为 L 的 ModuleList，ln_f 是最后的 LayerNorm。再创建无偏置的输出 lm_head，并让输入嵌入与输出矩阵共享一个 Parameter。用这些容器不是为了多一层包装；它们保证子模块被 PyTorch 注册，能出现在参数遍历、设备迁移和 state dict 中。

对象	参数形状	前向用途
token embedding	(V, C)	输入编号查行
position embedding	$(T_{\max}, C)$	位置编号查行
注意力 QKV 投影	$(3C, C)$	一次线性映射生成 Q、K、V
注意力输出投影	(C, C)	混合拼接后的各头特征
MLP 第一层	$(4C, C)$	扩展特征宽度
MLP 第二层	$(C, 4C)$	回到残差宽度
每个 LayerNorm	(C) ，可另有 (C) 偏置	逐位置归一化后缩放平移
输出头	与 token embedding 同一个 (V, C)	特征转为词表分数

表中 Linear 权重写成 PyTorch 的存储形状：输出维在前，输入维在后。计算 xW^T 时， (B, T, C) 才能变成 $(B, T, 3C)$ 。不了解这个约定，读预训练转置代码时很容易误以为它在“修正数学”。

4.2 不靠打印的 M 数，自己数一遍

先令 bias=False。一个 block 的注意力有 $3C^2 + C^2 = 4C^2$ 个矩阵参数，MLP 有 $4C^2 + 4C^2 = 8C^2$ ，两个 LayerNorm 各有 C 个缩放参数。因此整块为 $12C^2 + 2C$ 。嵌入共 $VC + T_{\max}C$ ，最后 LayerNorm 再加 C ，输出头因为共享不重复计数：

$$N_{\text{all}} = VC + T_{\max}C + L(12C^2 + 2C) + C.$$

若 `bias=True`，每块的四个线性层多出 $3C + C + 4C + C = 9C$ 个偏置，两个 LayerNorm 多出 $2C$ ，最后 LayerNorm 多出 C ，得到：

$$N_{\text{all}} = VC + T_{\text{max}}C + L(12C^2 + 13C) + 2C.$$

本书小测试采用 $V = 7, T_{\text{max}} = 6, L = 2, C = 8$ 、`bias=True`。计算为 $56 + 48 + 2(768 + 104) + 16 = 1864$ 。实际 `sum(p.numel() for p in model.parameters())` 为 1864。`get_num_params()` 默认再减去位置嵌入 48，报告 1816。它所谓的“non-embedding”并没有去掉 token embedding，因为同一矩阵也承担输出投影。这是本仓库专用的计数口径，不能只凭函数名把它等同于“扣掉所有嵌入”。

头数 H 在固定 C 时不出现在上述参数公式里。增加头数把每头维数 $d = C/H$ 变小，QKV 总宽度仍是 $3C$ 。这不意味着速度、注意力分布或内存完全不变，只说明参数数目没有随 H 直接倍增。

4.3 初始值如何进入残差网络

`_init_weights` 给 Linear 和 Embedding 的权重使用均值 0、标准差 0.02 的正态分布，Linear 偏置设 0。LayerNorm 构造时已经把缩放设 1、偏置设 0。这样 LayerNorm 初始时执行标准归一化而不额外改变尺度。

随后源码再次遍历参数，把所有名字以 `c_proj.weight` 结尾的参数设为标准差 $0.02/\sqrt{2L}$ 。注意力输出投影和 MLP 输出投影都叫 `c_proj`，两者都命中。每个 block 往残差流加两次输出，所以缩放中有 $2L$ 。可用一个粗略独立性假设理解：若 $2L$ 个增量方差相近且独立，求和方差约乘 $2L$ ；将每个增量尺度缩小 $\sqrt{2L}$ ，可以抑制初始残差方差随深度增长。真实网络的分支不是严格独立，所以这是初始化设计直觉，不是训练全程方差守恒定理。

共享权重在 `self.apply` 前建立，因此初始化遍历会经过 Embedding 和 Linear 两个模块所引用的同一参数。这可能导致该共享矩阵被初始化两次，但最后仍只有一份存储、一份梯度；不能因此把参数数目加倍。随机种子与模块初始化次序也会影响后续随机状态，所以“配置相同”之外，还需固定实现和执行顺序才能比较初始权重。

4.4 改结构时，先看哪个成本增长

把 C 翻倍，block 主矩阵参数约增到四倍；把 L 翻倍，这部分约增到两倍。把词表 V 翻倍，输入/输出共享矩阵加倍，训练时 logits 的最后一维也加倍。把最大长度翻倍，只让位置参数线性增长，却可能让手工注意力矩阵按长度平方增长。参数数量不是训练显存的完整估算，第 14 章会把激活和优化器状态分开。

补充练习。 小测试把最大长度从 6 裁成 4，参数减少多少？把头数从 2 改为 4、宽度仍为 8 呢？

答案。 裁位置表减少 $(6 - 4) \times 8 = 16$ 个参数；头数改变不改参数总量，但每头宽度从 4 变成 2。前者已在 `verify_mechanics.py` 断言，后者可直接用本节公式核算。恢复已有 checkpoint 时不能仅根据“参数总数一样”认定语义相同，头的划分已经改变。

本章来源

- `model.py`: GPTConfig、构造、计数和初始化
- 参数估算 notebook
- 本书实测: `qa/author/mechanics.json` 中 `parameters`。

第 5 章 逐层推导因果注意力、MLP 和残差

5.1 LayerNorm 沿哪个维度计算

一个位置的特征为 $x \in \mathbb{R}^C$ 。LayerNorm 计算该位置内的均值 $\mu = C^{-1} \sum_c x_c$ 和方差 $\sigma^2 = C^{-1} \sum_c (x_c - \mu)^2$ ，再输出

$$y_c = \gamma_c \frac{x_c - \mu}{\sqrt{\sigma^2 + 10^{-5}}} + \beta_c.$$

γ 是长度 C 的可学习缩放， β 在 `bias=False` 时不存在。每个批次、每个时间位置独立归一化；它不把不同句子或不同时间位置混在一起。因此训练和生成时不需要像 BatchNorm 那样维护运行均值。源码自定义 LayerNorm 的主要原因是提供可选偏置接口，归一化运算仍调用 `F.layer_norm`。LayerNorm

分母中的 10^{-5} 防止方差过小时除零。若 $x = [1, 1]$ ，中心化后都为 0；输出是偏置，而不是 NaN。训练时 γ 和 β 获得梯度，输入也通过均值、方差参与求导；不能把统计量当常数手写一个不完整反向过程。

5.2 Q、K、V 是三次线性投影，合成一次执行

输入 X 为 (B, T, C) ，`c_attn` 输出 $(B, T, 3C)$ ，沿最后一维拆成 Q、K、V。令每头宽度 $d = C/H$ ，每个张量先 `view` 为 (B, T, H, d) ，再把时间和头轴交换，得到 (B, H, T, d) 。断言 $C \bmod H = 0$ 就是在保证这个分组有整数大小。

对某个批次与头，查询位置 i 对键位置 j 的分数为

$$s_{ij} = \frac{q_i \cdot k_j}{\sqrt{d}}.$$

点积把 d 个乘积相加；若这些分量方差近似稳定，未经缩放的点积方差随 d 增大。除以 $\sqrt{d}$ 有助于避免 softmax 一开始就极端尖锐。随后对每个查询行做 softmax，得到注意力权重 a_{ij} ，输出 $o_i = \sum_j a_{ij} v_j$ 。Q 和 K 决定“从哪里读”，V 提供被读出的内容；这三个角色是计算上的区别，不意味着某些神经元永久代表固定语义。

操作	输入形状	输出形状
<code>q @ k.transpose(-2, -1)</code>	(B, H, T, d) 与 (B, H, d, T)	(B, H, T, T)
行 softmax	(B, H, T, T)	相同，每行和为 1
<code>att @ v</code>	(B, H, T, T) 与 (B, H, T, d)	(B, H, T, d)
合并各头	转轴、连续化、view	(B, T, C)

操作	输入形状	输出形状
输出投影	(B, T, C)	(B, T, C)

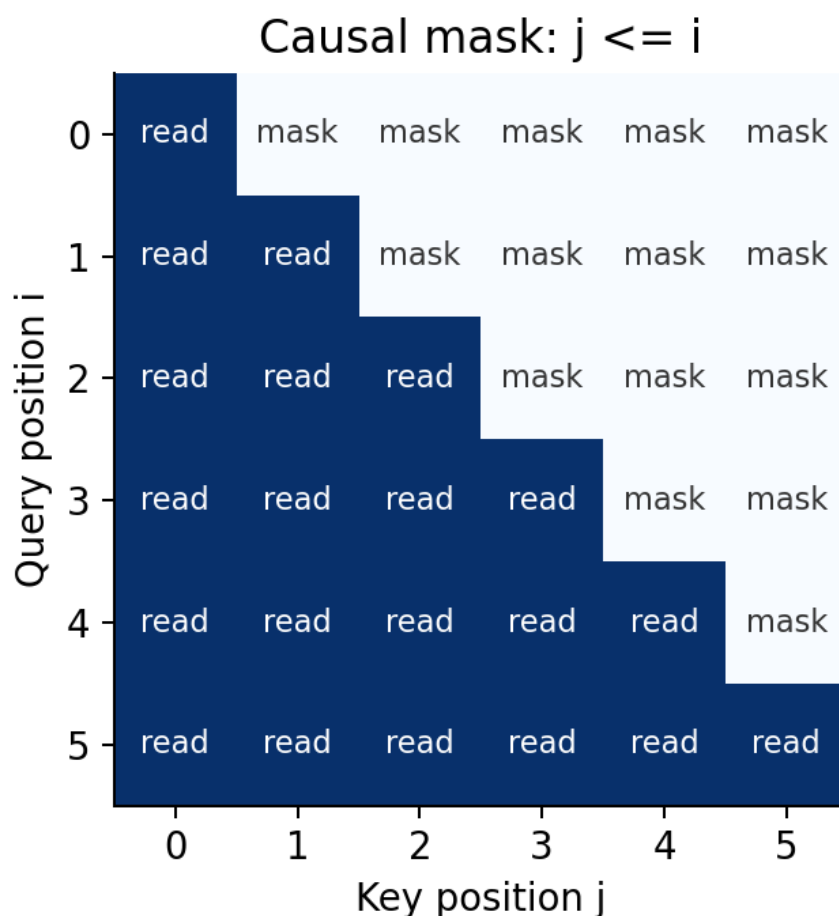
`transpose` 通常只改变步幅，不把数据重新排成连续存储。源码在合并头之前显式 `.contiguous()`，然后 `.view(B,T,C)`；如果省掉连续化，`view` 可能因步幅不兼容报错，或者不得不用能复制的 `reshape`。

5.3 掩码必须在 softmax 之前作用

自回归训练在位置 i 只能访问 $j \leq i$ 。手工路径创建下三角 0/1 掩码，把未来位置的分数改成负无穷，再做 softmax：

$$a_{ij} = \begin{cases} \frac{e^{s_{ij}}}{\sum_{k=0}^i e^{s_{ik}}}, & j \leq i, \\ 0, & j > i. \end{cases}$$

例如第二个位置可以看到第一个和第二个位置。若未归一化分数为 $[0, \log 2, 100]$ ，屏蔽未来后权重为 $[1/3, 2/3, 0]$ 。未来位置即使有极大分数也不应改变结果。如果先对三项 softmax 再乘掩码，保留项的和会远小于 1，已经不是同一个注意力运算。



源码的手工实现把 mask 注册为 buffer，形状 $(1, 1, T_{\max}, T_{\max})$ ，可在批次和头维广播；它不是可学习参数，但会随模型移动设备。masked_fill 是对分数设置禁区，不是把未来的输入 embedding 清零。层与层都保持同一因果关系，才能保证深层也不泄漏。

本书修改输入后半段、比较前半段 logits，最大误差实际为 0。这检验的是“在固定权重下，前缀输出不依赖后缀”，比只看掩码图更接近真正需要的性质。测试时关闭 dropout，否则两次前向可能因随机掩码不同而出现差异。

5.4 名为 flash，不保证用了 FlashAttention 内核

self.flash 只是检查 PyTorch 是否有 scaled_dot_product_attention 函数。存在时传入 is_causal=True，训练态使用配置 dropout，评估态明确传 0。这个标记不探测硬件，也不证明 CPU 上运行 CUDA FlashAttention。具体后端由 PyTorch、设备、dtype 和形状决定；本书 CPU 测试只能验证 SDPA 接口结果。[固定实现](#)

为了核对两条路径，本书复制同一个 double 模型，给副本显式注册下三角 buffer，再设 flash=False。仅翻转标记会失败，因为初始化时有 SDPA 的模型没有建立手工 mask。两条路径 logits 最大差为 2.78×10^{-17} ，属于浮点舍入量级。带 dropout 时随机数消费顺序可能不同，不能要求逐位相等。

手工路径显式保留 (B, H, T, T) 分数和权重，长度增大时内存增长快。优化后端可以减少中间张量读写，但不会改变模型所表达的因果注意力定义。性能结论必须另测，不能从误差测试推出加速倍数。

5.5 MLP 改每个位置，注意力连接位置

MLP 的第一层把 C 扩为 $4C$ ，经 GELU，再投影回 C ，最后 dropout。GELU 可写为 $\text{GELU}(x) = x\Phi(x)$ ， Φ 是标准正态分布累积分布函数。它是平滑非线性，不是简单截断负数的 ReLU。若去掉非线性，两层线性映射可以合成为一个线性映射，扩宽再压回的表达作用会大幅改变。

MLP 对每个位置使用同一组参数，但位置之间不直接通信。位置间混合由注意力承担，特征间非线性组合由 MLP 承担。Block.forward 依次执行

$$U = X + \text{Attention}(\text{LN}_1(X)), \quad Y = U + \text{MLP}(\text{LN}_2(U)).$$

这是 pre-LN 结构：先归一化，再计算分支，最后加回原始残差流。每次相加双方都是 (B, T, C) 。残差让梯度包含一条恒等路径，减少必须连续穿过所有非线性变换的压力；它不保证任何深度、学习率都稳定。

补充练习。 将 attention 输出投影改成 $(C/2)$ 维，再保留 $x + \text{attn}(\dots)$ ，会怎样？若把因果 mask 去掉而训练 loss 更低，能说模型更好吗？

答案。 残差相加的形状不兼容，需要额外投影才能定义运算，而那已经改变结构。去掉 mask 后模型可读到目标对应的未来输入，训练任务变得泄漏；更低损失不能作为自回归生成改善的证据。用前缀不变性测试可直接揭示这种反例。

本章来源

- `model.py`: `LayerNorm`、`CausalSelfAttention`、`MLP`、`Block`
- PyTorch SDPA 官方接口
- 本书核对程序: `examples/verify_mechanics.py`, 结果 `qa/author/mechanics.json`。

第 6 章 权重共享、损失、预训练映射与上下文裁剪

6.1 从编号到每个位置的预测

`GPT.forward(idx, targets=None)` 先读出批次和长度，要求 $T \leq T_{\max}$ ，再创建位置编号 $0, \dots, T-1$ 。token embedding 为 (B, T, C) ，position embedding 为 (T, C) ，相加时位置向量在批次维广播。经过输入 dropout、 L 个 block 和最终 LayerNorm，隐藏状态仍是 (B, T, C) 。`forward`

这套位置编码是学出来的绝对位置表。生成时若只保留最近窗口，留下来的 token 会重新从位置 0 编号；原代码没有旋转位置编码，也没有保留无限增长的全局位置。不要把别的模型的位置机制套到这里。

有 `targets` 时，输出头计算全部位置的 logits，形状为 (B, T, V) 。无 `targets` 时，只取 $x[:, [-1], :]$ ，保留长度为 1 的时间维，返回 $(B, 1, V)$ ，损失为 `None`。优化只省掉最后输出头对早期位置的计算；各层注意力仍处理整个输入窗口，因此它不是 KV cache。

6.2 交叉熵为什么要把前两维展平

令 z_{btv} 是位置 (b, t) 对词表项 v 的分数，目标编号为 y_{bt} 。单位置损失为

$$\ell_{bt} = -z_{bt, y_{bt}} + \log \sum_{v=0}^{V-1} e^{z_{btv}}.$$

对所有非忽略位置求平均得到最终 loss。展平 logits 为 (BT, V) 、targets 为 (BT) ，就是把每个位置当一条分类样本。若 logits 全相同，每项概率 $1/V$ ，损失为 $\log V$ 。五字符合成实验中，这个均匀基线是 $\log 5 \approx 1.6094$ ；随机初始化的实际初始验证损失 1.6728 略高，并不异常。

`ignore_index=-1` 允许把某些目标标为不参与损失，但输入编号依然不能是 -1：嵌入查表和目标掩码是不同接口。若所有目标都被忽略，均值没有有效分母，不应期待一个正常的训练损失。若有不同数量的有效目标，等权平均微批次损失也不再自动等于对全部有效 token 的平均，第 8 章讨论这个条件。

源码使用 `targets.view(-1)`，要求内存步幅兼容。我们用 `y[:, :4]` 截取原来长度 6 的二维张量时，真实触发了 `RuntimeError`；调用 `.contiguous()` 后通过。原 `get_batch` 用 `stack` 生成连续目标，因此标准训练不受影响。自己改 loader 或对目标裁切时，应检查连续性，或在修改版中明确使用 `reshape`。不能把这个接口失败误诊为 checkpoint 损坏。

6.3 共享不是拷贝，梯度会从两个角色汇合

`self.transformer.wte.weight = self.lm_head.weight` 令两处引用同一个 Parameter。前向时它一边提供输入向量，一边把最终特征映回词表；反向时这两条使用路径对同一个参数的梯度相加。若只复制数值而创建新 Parameter，两份权重下一步就可能分离，那不叫权重共享。

以共享矩阵元素 $E_{2,3}$ 为例，本书 double 测试的自动梯度为 -0.11059136136513018。将同一个元素增加和减少 $\epsilon = 10^{-5}$ ，用

$$\frac{\mathcal{L}(E + \epsilon) - \mathcal{L}(E - \epsilon)}{2\epsilon}$$

得到 -0.11059136064694995，误差 7.18×10^{-10} 。有限差分同时改变输入和输出两处使用，才与自动梯度比较的是同一个函数。这里没有分别给两套共享引用施加扰动。

参数遍历默认去重，`configure_optimizers` 又从 `named_parameters` 建字典；本书检查优化器内 Parameter 身份没有重复。否则同一权重可能被更新两遍，训练异常却很难从 state dict 的重复名字看出来。

6.4 从 GPT-2 导入时，名字和形状都要对齐

`from_pretrained` 只接受四个型号，并仅允许覆盖 dropout。它固定词表 50257、长度 1024、`bias=True`，再按型号确定 (L, H, C) ：小型 (12, 12, 768)，medium (24, 16, 1024)，large (36, 20, 1280)，xl (48, 25, 1600)。这些是结构约束，不能在导入权重时任意改宽度。导入路径

源码从 Hugging Face `GPT2LMHeadModel` 取 state dict，排除注意力 mask buffer，核对键数量，再逐键核对形状。四类权重需要转置：`attn.c_attn.weight`、`attn.c_proj.weight`、`mlp.c_fc.weight`、`mlp.c_proj.weight`。原因是上游 GPT-2 使用的 Conv1D 包装与本地 Linear 存储方向不同。例如从 $(C, 3C)$ 转到 $(3C, C)$ ，计算语义才一致。其余权重直接拷贝，不应为了“统一”把 embedding 也转置。

拷贝在 `torch.no_grad()` 下进行，因为导入是状态赋值，不是优化目标中的可求导操作。键数量相等只是第一关；键名、形状、词表语义和层结构都匹配才有意义。本书没有下载这些权重，这部分是固定源码静态核对，不声称验证了当前 Transformers 版本兼容性或输出一致性。

6.5 裁长度会换 Parameter，时机很重要

`crop_block_size(new_T)` 要求新长度不大于旧长度，更新 config，截取位置表前 `new_T` 行并包装成新的 Parameter；手工注意力路径的 mask 也截到相同大小。它不能扩展位置表，不能让一个只学过 1024 位置的模型无代价支持更长上下文。

裁剪应在创建优化器之前完成，原 `train.py` 正是如此。因为位置表 `Parameter` 被替换，先创建的优化器可能还持有旧对象。恢复训练另有一层风险：如果先裁小模型，再加载原 checkpoint 的优化器状态，位置表的一、二阶矩形状仍可能对应旧长度，后续更新会冲突。权重裁剪成功不等于优化器状态也已裁好；需要同步迁移状态或明确重建优化器。

补充练习。 无 `targets` 的前向输出只有最后一个位置，能否用它直接计算整个训练窗口的交叉熵？

答案。 不能。它只有 $(B, 1, V)$ ，而整窗目标为 (B, T) 。要传 `targets` 走训练分支，或修改前向返回所有 logits。不能靠广播把缺少的预测补出来。

本章来源

- `forward`、`crop_block_size`、`from_pretrained`
- 恢复与裁剪的顺序
- 本书梯度、共享身份及连续性反例：`examples/verify_mechanics.py`。

第 7 章 memmap 批次与下一 token 对齐

7.1 原始二进制没有自我说明能力

`get_batch(split)` 把 `train.bin` 或 `val.bin` 以 `dtype=np.uint16, mode='r'` 打开。memmap 让操作系统按需映射文件页，不要求先把整个语料读成一个常驻 NumPy 数组。它并不消除磁盘读取，也不保证任何随机访问模式都快。源码每次取批重新建立映射，注释将原因指向一个内存增长问题；本书把它当作该版本的工程选择，不把旧讨论里的现象推成所有 NumPy 版本的普遍定律。[get_batch](#)

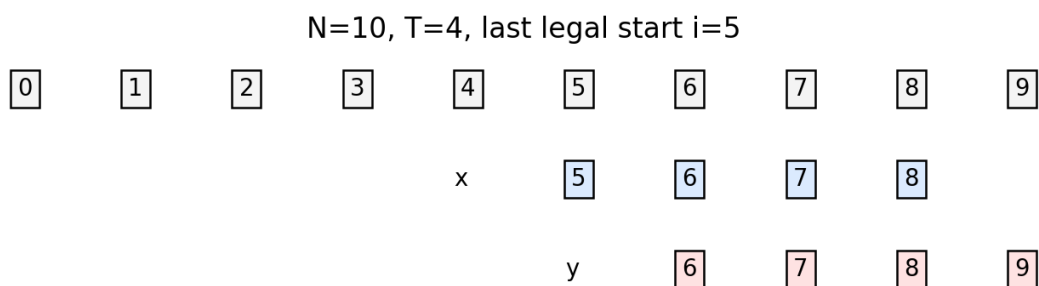
文件里没有 `dtype` 或词表标识。若读错 `dtype`，模型不一定立刻报错。第一步检查应包括：文件字节数能否被元素字节数整除、token 总数是否大于 `block_size`、编号是否非负且小于词表、训练与验证文件是否真的来自不同切分。随机抽几个值只能发现显眼错误，完整范围检查要在准备数据时完成。

7.2 最后一扇窗口为什么从 $N-T-1$ 开始

设文件有 N 个 token，窗口长度为 T ，起点为 i 。输入读取索引 i 到 $i+T-1$ ；目标读取 $i+1$ 到 $i+T$ 。目标最后一项必须满足 $i+T \leq N-1$ ，所以

$$0 \leq i \leq N - T - 1.$$

`torch.randint(N-T, (B,))` 的上界不包含在内，恰好得到这组合法起点。不是少取了一项，而是要给下一 token 留出一个位置。若 $N = T$ ，输入虽然刚好能装满一个窗口，目标却少一项，`randint(0,...)` 会失败。必须有 $N \geq T + 1$ 。



图中 $N = 10, T = 4$ ，最后合法起点是 5。输入 $[5, 6, 7, 8]$ ，目标 $[6, 7, 8, 9]$ 。如果错误地允许起点 6，输入有四项，目标只剩三项；如果把目标也从 5 开始，则模型在做复制当前 token，不再是预测下一个。两种错误的损失都不能与正常训练相提并论。

7.3 为什么磁盘上是 uint16，模型前是 int64

代码逐窗 `.astype(np.int64)`，然后 `torch.from_numpy`，最后 `stack` 成 (B, T) 。磁盘的紧凑类型节省存储；嵌入查表和交叉熵目标用长整型编号。`astype` 会创建转换后的数组，因此张量并不是直接引用整个 `memmap` 的 16 位视图。`stack` 又把这些窗口组织成连续批次。

CUDA 路径先 `.pin_memory()`，再 `.to(device, non_blocking=True)`。固定页内存有助于支持主机到设备的异步复制；`non_blocking=True` 是允许异步的请求，实际重叠程度还取决于设备、流和工作安排。CPU 路径直接 `.to(device)`。MPS 虽然在配置注释中列为设备示例，但脚本把不含 `cuda` 的设备都划到 `device_type='cpu'` 分支，本书没有验证其性能或完整支持情况。

本书完整流水线的两个文件分别含 7168 与 1792 个 token，长度远大于 $T = 8$ 。每批两个随机起点，不按 epoch 顺序扫描，不保证一个 token 在若干步后恰好被看过一次。所谓“看了多少遍语料”，只能用累计训练 token 数除以文件 token 数作粗略曝光量估计。

7.4 训练批次与验证批次共享随机数发生器

`get_batch` 使用 PyTorch 的全局随机数发生器。`estimate_loss()` 也调用同一函数。因此改变 `eval_iters` 或 `eval_interval`，即使不改变模型结构和初始 `seed`，也会改变后续训练窗口的随机序列。这解释了一个常见困惑：“我只是多验证几次，为什么之后训练不完全一样？”评估本身不求梯度，却会消耗随机状态。

最初的 x, y 在主循环前已经取好。每次前向之后又预取下一批，再执行反向；循环结束时还留有一批等待下次使用。若要设计严格可恢复训练，仅保存 RNG 还不够，还要记录或重建这份已预取批次的位置。本书第 9 章将这项状态与 `checkpoint` 一起核对。

补充练习。 数据长度为 9、窗口长度为 8，合法起点有几个？改成长度为 8 呢？

答案。 前者只有起点 0，一个输入窗和一个错位目标窗；后者没有合法起点。增大 `batch_size` 会重复抽同一起点，不会凭空创造新样本。`verify_mechanics.py` 穷举了 $N = 10, T = 4$ 的全部合法起点并检查对齐。

本章来源

- `train.py`: `memmap` 与窗口
- `bench.py` 的另一份 `get_batch`，差异见第 13 章。
- 本书对齐测试: `qa/author/mechanics.json` 的 `last_legal_window`。

第 8 章 训练主循环、梯度累积、混合精度与裁剪

8.1 一步到底包含几次前向

在 nanoGPT 里, `iter_num` 主要计数优化器更新; 一个更新内部可以包含多个微批次。令本进程累积次数为 a , 第 r 个微批次平均损失为 ℓ_r , 目标是优化它们的平均值

$$\mathcal{L} = \frac{1}{a} \sum_{r=1}^a \ell_r, \quad \nabla_{\theta} \mathcal{L} = \frac{1}{a} \sum_{r=1}^a \nabla_{\theta} \ell_r.$$

因此循环每次先把 loss 除以 a , 再 backward, 直到全部微批次完成才 step。梯度默认累加进 `.grad`; 若在每个微批次后清零, 就只剩最后一个微批次的梯度。若忘记除以 a , 梯度尺度增大 a 倍, 裁剪和优化器行为都会改变。训练循环

这里的等价性需要条件: 微批次包含相同数量的有效 token, 模型对样本没有跨批次统计依赖, 随机层处理方式也需可比。若一个微批次只有一个未忽略目标, 另一个有 100 个, 简单平均两项“各自均值”会给前者过大权重。应按有效 token 数加权。dropout 开启时, 整批和分批的随机掩码消费也可能不同, 不宜要求逐位一致。

本书在 dropout=0、double、相同样本下比较整批四条序列与两次各两条序列, 参数梯度最大差为 5.55×10^{-17} 。它验证了计算逻辑, 不是显存或吞吐测试。

8.2 优化器为什么分成两组

`configure_optimizers` 先保留 `requires_grad=True` 的去重参数, 再按维数分类。维数不少于 2 的矩阵使用配置的 weight decay; 一维及标量参数不衰减。于是 token 和位置 embedding、各线性矩阵被衰减, LayerNorm 缩放及各种偏置不衰减。它不是按参数名字写死一张排除表, 也不是“所有 embedding 都不衰减”。优化器构造

AdamW 维护梯度的一阶矩 m_t 和二阶矩 v_t , 用 β_1, β_2 控制历史平滑。省略实现细节后, 更新可理解为

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

这里 θ_t 是第 t 次更新前的参数, $\hat{m}_t$ 、 $\hat{v}_t$ 是经过初始偏差校正的矩估计, $\epsilon > 0$ 是避免分母过小的常数。若把当前梯度记为 g_t 、从全零矩状态开始并令更新次数 t 从 1 计数, 则 $m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$ 、 $v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$, 校正为 $\hat{m}_t = m_t / (1 - \beta_1^t)$ 、 $\hat{v}_t = v_t / (1 - \beta_2^t)$; 平方、平方根和除法均逐

参数元素进行。 η_t 为学习率， λ 为该参数组的权重衰减。衰减与梯度统计解耦；它不等同于先向 loss 加一个 L2 项再任由 Adam 的自适应缩放处理。不同组的 λ 可以为 0，但本脚本每步把同一学习率写入两组。

脚本通过检查 AdamW 签名是否含 fused，并要求 `device_type=='cuda'` 才启用 fused 实现。签名检测解决的是 API 是否提供此参数，不证明特定硬件、dtype 或版本一定能成功运行。CPU 本书实测输出为 `using fused AdamW: False`。

8.3 AMP 改计算精度，不等于把所有状态都减半

CUDA 上的 `torch.amp.autocast` 为部分运算选择指定精度。模型默认参数仍是 FP32，并没有调用 `model.half()`。因此不能简单把参数、梯度、Adam 状态、激活全部按两字节估算。CPU 分支用空上下文，运行普通 FP32；本书梯度核对额外把模型转成 double。

`GradScaler` 仅在 dtype 为 float16 时启用。它把 loss 放大后反向，使某些很小的梯度较不容易下溢；更新后再恢复真实尺度。若出现非有限梯度，scaler 可跳过优化器更新并调整缩放因子。BF16 与 FP16 的数值范围不同，本脚本 BF16 不启用这个 scaler。不要把“低精度”视为一种统一行为。

固定代码调用 `torch.cuda.amp.GradScaler`，在测试版本产生 API 弃用警告，但 CPU、dtype=float32 下它是禁用的空操作。本书保留原版执行并记录警告，没有借机替换源码再声称原版通过。

8.4 先还原梯度，再裁剪，再更新

全局梯度范数为

$$g = \sqrt{\sum_p \sum_j (\partial \mathcal{L} / \partial p_j)^2}.$$

当阈值为 c 时，梯度整体乘以不大于 1 的系数，约为 $\min(1, c/g)$ 。这样保留方向、限制范数。源码在裁剪前调用 `scaler.unscale_(optimizer)`，否则阈值会作用到人为放大的梯度上，实际裁剪会随 loss scale 改变。然后 `clip_grad_norm_`，再 `scaler.step`、`scaler.update`，最后 `zero_grad(set_to_none=True)`。

裁剪不会恢复已经出现 NaN 的前向，也不会保证每步都收敛。它限制梯度，不直接限制 AdamW 的最终参数更新，因为后者还依赖历史矩和权重衰减。把阈值调小到训练几乎不动，不是解决根因的可靠方法。

8.5 日志中 loss 与时间的细节

循环打印 `loss.item() * gradient_accumulation_steps`，撤销最后一个微批次的缩放。它只得到最后一个微批次损失，不是这次更新所有微批次损失的精确平均。要记录精确平均，应在循环中累计每个未缩放 loss，但这可能增加同步成本或状态管理。

`loss.item()` 会要求把标量带回 CPU，CUDA 中常形成同步点。`dt` 的测量又与前后循环、验证和保存交错，所以不能把普通训练日志中的每个时间都当成纯模型前后向耗时。稳定性能测量用第 13 章专门说明的预热、同步和口径。

补充练习。 不改变全局 token 数，把 `batch_size` 减半、累积次数加倍，是否保证训练完全不变？

答案。 理想、无随机层、等长有效样本情况下，平均梯度可以相等；实际数据抽样次序、dropout、浮点加法顺序和系统开销会改变。更小微批次也可能降低硬件利用率。应分别验证梯度近似、训练指标和速度，不用“等效 batch”一词包办三项结论。

本章来源

- `train.py`: `autocast`、`GradScaler`、累积和裁剪
- `model.py`: AdamW 参数分组
- PyTorch AdamW 官方接口
- 本书累积与去重测试: [qa/author/mechanics.json](#)。

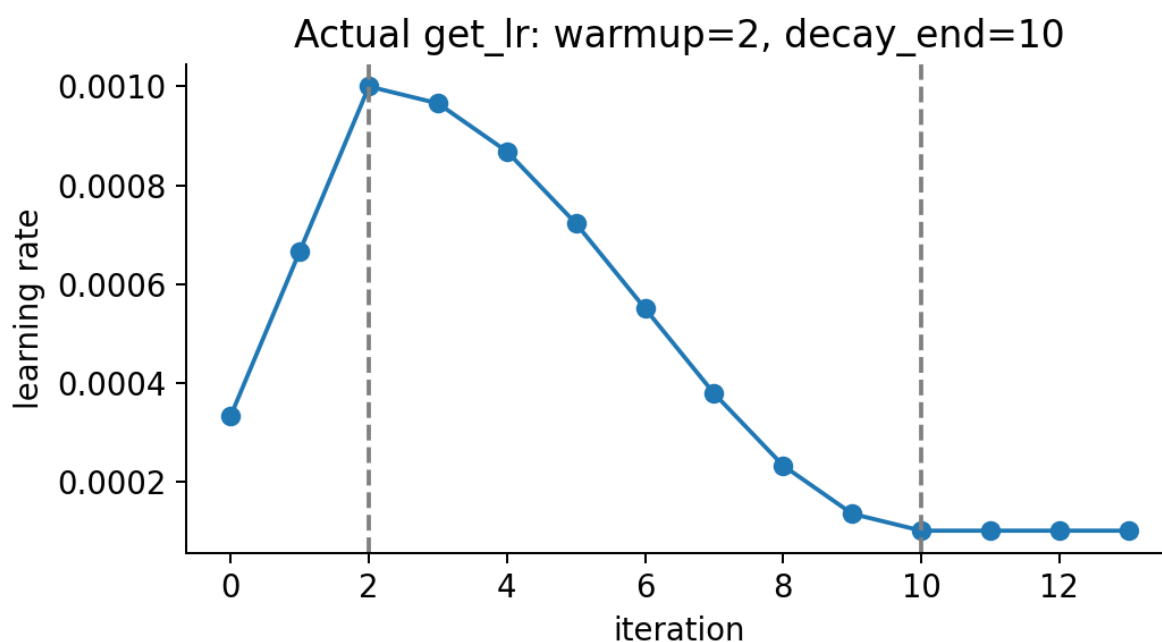
第9章 学习率、验证、checkpoint与恢复

9.1 逐段读学习率，而不是只说“余弦退火”

令 i 为当前迭代编号， w 为 warmup 终点， d 为衰减终点， $\eta_{\max}$ 与 $\eta_{\min}$ 为学习率上下限。固定 `get_lr` 的精确行为是

$$\eta(i) = \begin{cases} \eta_{\max}(i+1)/(w+1), & i < w, \\ \eta_{\min} + (\eta_{\max} - \eta_{\min}) \frac{1 + \cos(\pi(i-w)/(d-w))}{2}, & w \leq i \leq d, \\ \eta_{\min}, & i > d. \end{cases}$$

第一步不是 0。若 $w=2, d=10, \eta_{\max}=0.001, \eta_{\min}=0.0001$ ，实际六个边界值为： $i=0$ 时 0.0003333，1 时 0.0006667，2 时 0.001，6 时 0.00055，10 与 11 时都是 0.0001。warmup 的最后一个分支点与余弦起点要分清。`get_lr`



$d=w$ 会使余弦分母为 0，本书已测试其 `ZeroDivisionError`； $d < w$ 也不构成合理日程。`decay_lr=False` 直接使用固定学习率，不调用这段函数。`max_iters` 决定程序何时停，`lr_decay_iters` 决定学习率何时降到底，两者只是通常设为相近，不存在自动绑定。

9.2 验证估计了什么

`estimate_loss` 加了 `@torch.no_grad()`，暂时切到 `eval`，分别对 `train`、`val` 各取 `eval_iters` 个随机批次，把各批次 `loss` 存进向量后取平均，最后切回 `train`。关闭梯度节省图和内存，`eval` 关闭 `dropout`；两者是不同开关。

这是随机窗口平均损失估计，不是遍历验证文件中每个 `token` 一次的精确值。相邻窗口可以重叠，短文件也会重复抽样。增大 `eval_iters` 通常降低抽样波动，但增加耗时并消耗更多 RNG。指标下降很小的时候，需要判断差值是否超过抽样噪声；原脚本没有置信区间或多 `seed` 比较。

本书初次运行 `step 20` 的 `val` 为 1.0560，恢复后重新评估同一 `checkpoint` 的 `step 20` 为 1.0557。权重相同也能得到不同抽样估计，这正是上述机制的可见结果，不应伪造为逐步严格下降曲线。

9.3 保存时机、best 字段与最后一步

每轮先设置学习率，再按间隔验证和保存，然后训练更新。`checkpoint` 中 `iter_num=20` 表示保存时尚尚未执行编号 20 的更新。在从 0 开始的正常路径上，此刻已经完成编号 0 到 19 的 20 次更新。

保存条件是“验证改善，或者 `always_save_checkpoint=True`”。进入这个分支时，代码无条件把 `best_val_loss` 改成当前值。因此打开 `always_save` 时，这个字段其实是最近保存评估值，不保证是历史最优；关闭它才更接近名字所说的意义。另有 `iter_num>0` 限制，所以 `step 0` 不保存。

停止条件位于更新后：先 `iter_num += 1`，再判断 `iter_num > max_iters`。设 `max_iters=20`、从 0 开始，会实际执行编号 0 到 20，共 21 次更新。最后 `checkpoint` 若在更新前 `step 20` 保存，它不包含这最后一次更新。训练正常退出不意味着最后内存状态已经落盘。这是原版边界，若自行改为 `>=`，应同时明确你想让编号代表已完成步数还是当前索引。

`eval_only` 也不是无条件退出：源码只有 `iter_num == 0 and eval_only` 时 `break`。因此从迭代 20 的 `checkpoint` 恢复并设 `eval_only=True`，不会按直觉只评估一次，而可能继续训练。对预训练初始化的评估配置，`iter_num` 为 0，才符合原设计。

9.4 checkpoint 有什么，没有什么

保存字段	作用	不应推断出的能力
<code>model</code>	参数及 <code>buffer</code>	不包含 <code>tokenizer</code> 文件本体
<code>optimizer</code>	AdamW 统计量和参数组	不等于保存全部 AMP 状态
<code>model_args</code>	重建结构	不允许任意修改层数仍兼容
<code>iter_num</code>	恢复计数和日程位置	不说明最后一次内存更新已保存

保存字段	作用	不应推断出的能力
best_val_loss	保存判断参考	always-save 时不保证历史最小
config	部分原始配置快照	不是全部运行时状态

恢复时先从 checkpoint 覆盖层数、头数、宽度、block_size、bias、vocab_size, dropout 可使用本次配置。重新建模型、清除 state dict 键的 _orig_mod. 前缀, 再载权重和优化器状态。原配置与新配置冲突时, 要检查实际采用的是哪一方。加载优化器还会恢复它的参数组设置; 随后学习率每步被日程重新赋值, 但其他组字段不能假设必然使用命令行的新值。

checkpoint 未保存 Python、NumPy、CPU/CUDA RNG 状态、GradScaler 状态、数据抽样游标或已经预取的 X/Y。恢复又重新执行 seed 初始化和模型构造, 随机状态与未中断路径不同。因此这是“恢复模型和优化器继续训练”, 不是严格逐位续跑。若要后者, 应设计完整状态协议, 并用连续运行与中断运行的参数哈希比较检验, 而不是仅看 loss 大致接上。保存与恢复代码

9.5 可信文件与版本边界

torch.load 和 pickle 元数据都有信任边界。本书只加载自身生成的文件; 在 PyTorch 2.6.0 中, 这个由张量和简单字典组成的 checkpoint 能通过默认加载, 检查脚本还显式使用 weights_only=True。这不代表任意旧 checkpoint 都可安全加载。遇到安全加载错误时, 不应把“设 weights_only=False”当成对陌生文件的通用修复。PyTorch 序列化说明

补充练习。 max_iters=7、eval_interval=5、从零开始, 若始终保存, 最后 checkpoint 的编号是什么? 共更新几次?

答案。 step 0 不保存, step 5 保存; 编号 0 到 7 共更新 8 次。最后 checkpoint 是更新编号 5 之前的状态, 后续三次更新并未保存。解决方法可以是增加明确的退出前保存, 但那是源码改动, 需说明与原版的差异。

本章来源

- train.py 全部状态转移
- 本书恢复记录: qa/author/tiny_pipeline.json、tiny_resume.log。
- 本书学习率边界: qa/author/mechanics.json。

第 10 章 单机多卡与跨节点 DDP 的状态约束

10.1 多张卡复制模型，分担的是数据

DDP 的基本思路是每个进程持有完整模型，用不同数据求局部梯度，再同步梯度，使各副本作相同更新。它并不把一个大模型的参数自动拆到多张卡上；如果单卡放不下模型和优化器状态，增加 DDP 卡数未必解决。固定 README 把 FSDP 列为未来工作，不能把它视为已经实现的能力。[固定训练入口](#)

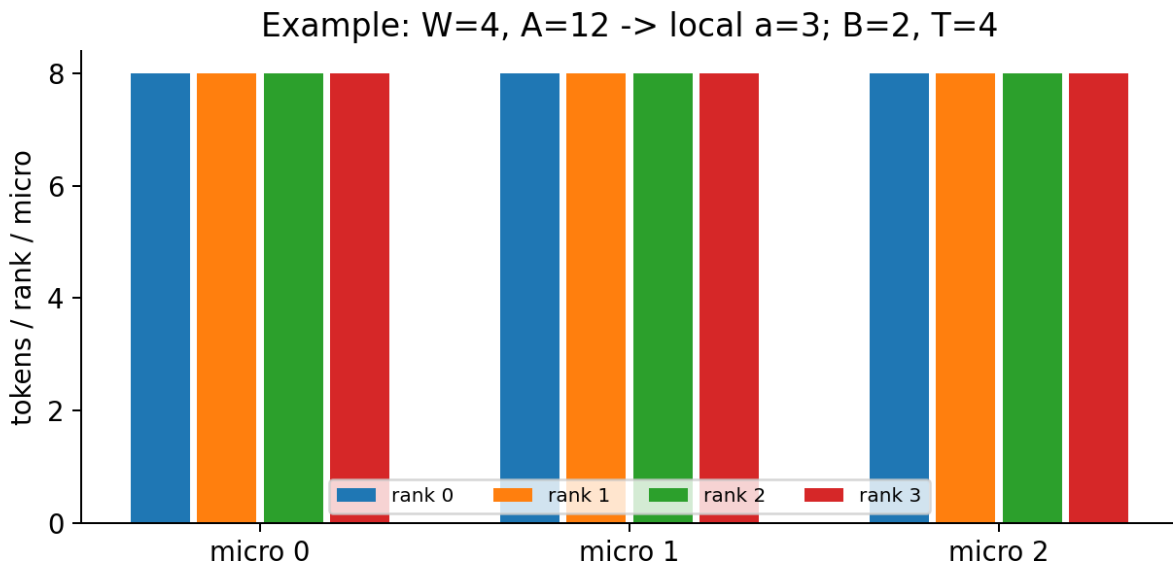
脚本通过环境变量 RANK 判断是否进入 DDP。全局 RANK 标识全部进程中的编号，LOCAL_RANK 是当前节点上的编号，WORLD_SIZE 是进程总数。初始化进程组后，脚本把设备强制设为 `cuda:LOCAL_RANK`，选卡，令 rank 0 负责主要日志和保存。即使把 backend 改成 gloo，也不意味着这份 DDP 分支就成为 CPU DDP：它仍显式设置 CUDA 设备。本书未启动此路径。

10.2 配置里的累积次数先除以 world size

令命令行配置的累积次数为 A ，进程数为 W 。代码要求 A 能被 W 整除，然后每进程使用 $a = A/W$ 。每个微批次 B 条、每条 T 个 token，则一次全局更新 token 数为

$$K = W \cdot a \cdot B \cdot T = A \cdot B \cdot T.$$

`train_gpt2.py` 设置 $A = 40, B = 12, T = 1024$ 。8 个进程时每进程累积 5 次，总 token 数 $8 \times 5 \times 12 \times 1024 = 491520$ 。单进程则累积 40 次，token 总量相同，等待时间和硬件并行程度不同。不能在读配置时误算成 40×8 次累积。[GPT-2 配置](#)



图示另一组小数值： $W = 4, A = 12$ ，所以本地 $a = 3$ ； $B = 2, T = 4$ ，每进程每微步 8 token，全局每更新 96 token。这是计数图，不是已运行的四卡实验。

在各进程、各微批次有效 token 数相等时，DDP 平均梯度后得到

$$g = \frac{1}{W} \sum_{r=1}^W \frac{1}{a} \sum_{j=1}^a \nabla \ell_{rj}.$$

如果设备数改变而配置 A 不可整除，会直接触发断言。静默向下取整会改变全局 batch，所以这里宁愿拒绝。若不同 rank 的有效 token 数不同，平均“各进程均值”也需重新加权，不能默认是全局 token 均值。

10.3 seed 不同，权重为什么仍应相同

脚本使用 `torch.manual_seed(1337 + rank)`。不同 rank 会从不同随机状态生成初始化、数据和 dropout；之后创建 DDP 包装时会同步模型状态，使各进程从一致权重出发。数据抽样仍由各自 RNG 驱动，不存在显式 `DistributedSampler`，也没有保证不同 rank 永不抽到同一窗口。

rank 0 额外运行评估，消耗更多随机数，因此它与其他 rank 的抽样流还会进一步分开。不同数据流是需要的，但这不构成一个记录完整、可逐位恢复的数据调度器。复现实验要固定进程数、评估频率、软件和硬件环境，并保存必要状态。

10.4 为何只在最后一个微步同步

每次微步都进行梯度 all-reduce 会多次支付通信成本。源码将 `model.require_backward_grad_sync` 设为“是否最后一个微步”，之前只在本地累加，最后一次统一同步。官方通常推荐用 `no_sync()` 上下文；该仓库直接改属性，是作者为缩短代码选择的实现方式，不应推广为跨版本稳定扩展接口。

forward 也必须处于相应同步策略下，因为 DDP 在前向时会准备部分反向同步状态。原代码在 forward 之前设置属性，顺序不可随意后移。所有 rank 还必须执行相同数量、相容次序的训练同步操作；一个 rank 提前退出或抛异常，其他 rank 可能一直等通信。

rank 0 单独评估也值得审查：现代 SDPA 分支没有手工注意力 mask buffer，而旧手工分支存在注册 buffer。DDP 默认可能在 forward 广播 buffer；仅 rank 0 多做评估前向、其他 rank 进入训练，可能造成 collective 次序不匹配。这是由源码状态推得的兼容风险，本书未进行旧版多卡复现，不把它报告为所有运行必然死锁。设计更稳健的验证流程时，应明确让所有 rank 协同评估，或在受控同步边界使用合适的非 DDP 模型调用，并实际测试。

10.5 跨节点时，先核对进程数和通信条件

`torchrun --nproc_per_node=8 --nnodes=2` 表示每节点 8 个进程、共 16 个，不是“总共 8 张卡”。两个节点的 `node_rank` 不同，`master` 地址与端口应相同且可达。源码开头有“四卡”文字与八进程命令不一致的注释，计算应以具体参数为准。

默认 $A = 40$ 不能被 16 整除，因此直接用默认累积值启动这个两节点示例会在断言处失败。要选择明确的可整除 A ，同时承认全局 `batch` 会按新值变化。这比照抄示例后把失败归咎于 NCCL 更有效。

网络带宽、延迟、节点间可达性会影响训练；README 提到的 InfiniBand 与 `NCCL_IB_DISABLE` 是历史环境建议，不应未经诊断就一律关闭某种传输。先验证设备、进程组和最小通信，再测训练。所有 `worker` 使用同一数据、源码和模型配置，`rank 0` 可写输出目录；共享路径是否真实共享也要确认。

补充练习。 从 8 卡改到 16 卡，保留 $B = 12, T = 1024$ ，想保持 491520 token/更新，这份原版累积设计能做到吗？

答案。 需要 $A = 40$ ，但 $40/16 = 2.5$ 不是整数，因此不能在每 `rank` 相同整数微步数的原设计下做到。可改微批次大小，例如 $B = 6, A = 80$ ，全局 token 数仍相同，每 `rank` 累积 5；但性能和随机分批会改变，必须重新验证。

本章来源

- `train.py`: DDP 初始化、同步与销毁
- `config/train_gpt2.py`
- PyTorch DDP 官方接口
- 本章 DDP 行为为静态审计；没有 GPU、NCCL 或跨节点实测。

第 11 章 生成接口：温度、top-k、提示和随机性

11.1 采样从一个非空提示开始

`sample.py` 默认从 `out_dir/ckpt.pt` 恢复，按 `model_args` 建模、处理 `_orig_mod.` 前缀、载入状态，然后 `eval()` 并移到设备。也可从四种 GPT-2 型号初始化，但这会走预训练权重路径，本书不执行。脚本随后查 checkpoint 中的 dataset 名称，寻找对应 `meta.pkl`；找到就使用字符映射，否则假定 GPT-2 BPE。采样脚本

这段默认逻辑有两个容易遗漏的条件。第一，自定义 BPE 不能只凭 `meta.pkl` 文件存在就自动工作，因为代码明确期待 `stoi` 和 `itos` 字符表。第二，即使用字符表，脚本顶层仍导入 `tiktoken`，所以运行原版 `sample.py` 需要安装该包，但字符路径不会调用 BPE 编码或下载词表。

`start` 若以 `FILE:` 开头，就把后面的路径作为 UTF-8 文本文件读入；否则直接编码字符串。得到的一维编号加一个批次维，成为 $(1, T_0)$ 。 T_0 必须大于零，原脚本没有为空提示补默认 token。未知字符会在字符查表时报错，空序列会在模型最后位置索引等环节失败。GPT-2 路径允许 `<|endoftext|>` 这个特殊 token，不等于允许任何任意特殊标记。

11.2 一次循环只增加一个 token

`GPT.generate` 循环 `max_new_tokens` 次。先取最近不超过 `block_size` 的输入窗口，前向得到最后位置 logits，再按温度和 top-k 调整，softmax 后用 `torch.multinomial` 抽一项，拼回完整 `idx`。因此返回长度为 $T_0 + M$ ，其中 M 是 `max_new_tokens`；旧提示保留在结果中，不只返回新生成部分。

`generate`

窗口裁短只影响“下一步拿什么作条件”，不截断已经累积的输出。原版没有遇到 EOT 就提前停止的判断，没有批内不同长度终止，也没有 KV cache。对每个新 token，它重新计算当前窗口的所有层。生成速度与训练吞吐因此不能用同一个 tokens/s 简单比较。

11.3 温度改变概率比，不改变正温度下的排序

令原 logits 为 z_v ，温度为 $\tau > 0$ 。采样概率为

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

两项概率比为 $p_i/p_j = \exp((z_i - z_j)/\tau)$ 。若分数是 $[0, \log 2]$ ，温度 1 给概率 $[1/3, 2/3]$ ；温度 0.5 给 $[1/5, 4/5]$ ；温度 2 给约 $[0.414, 0.586]$ 。降低正温度强化已有偏好，升高则更平坦，但不会把模型不知道的知识补进来。

原接口没有检查温度为正。零会产生除零、无穷或 NaN，负温度甚至反转排序，都不是通常所说的温度采样。若想确定性贪心，应显式用 `argmax`，而不是给一个未经验证的零温度。有限精度下非常低的温度也可能造成数值极端。

11.4 top-k 的阈值实现有平分边界

源码先取 $\min(\text{top_k}, V)$ 个最大分数，拿第 k 个分数当阈值，把严格小于它的项设负无穷。再 softmax，相当于在保留集合上重新归一化。若 $z = [1, 1, 0]$ 、 $k=1$ ，两项并列最大都不小于阈值，实际上会保留两个候选；本书测试已验证。所以“`top_k=1` 一定等于 `argmax`”在这个实现中有并列例外。

k 超过词表被截为 V ，等于不排除任何项。 $k=0$ 或负数无效，源码没有提供友好的参数范围错误。正温度缩放保持排序，所以在数学上先缩放再取 top-k 与先按原分数选集合一致；浮点溢出或极端数值不在这个理想前提里。

设置	主要作用	常见代价
较低正温度	更偏向高分项	重复与僵化可能增加
较高温度	更多低分项被抽到	语法或内容错误可能增加
较小 top-k	限制候选范围	正确但暂时低分的项可能被排除
无 top-k	保留全词表概率	长尾噪声也参与抽样

这些是采样机制的方向性影响，不是某个温度对所有任务更好的经验定律。比较生成应使用同一模型、提示和明确 seed，并看多个样本。

11.5 eval、no_grad 和 seed 各做一件事

`eval()` 关闭 dropout，`no_grad()` 不建立反向图，seed 控制随机抽样流。只设 seed 不会关闭 dropout；只设 eval 仍会从概率分布随机抽样。`generate` 方法有 `no_grad` 装饰器，但不会替调用方自动设 eval，因此直接调用模型方法时要自己确认状态。

本书两次单独启动 `sample.py`，使用 `seed=17`、温度 0.8、`top-k=3`、提示 `ab`、生成 32 token，完整 stdout 一致。生成片段为：

```
ab
aab
cab
```

```
cabcab caab
aab
cab ca
```

它能看出合成训练数据的局部片段，却仍有重复和拼接错误。这是 25 步小实验的真实输出，不用“像语言”包装成自然语言能力。若同一次进程里连续抽多个样本，RNG 状态会前进，通常得到不同结果；并不是每个样本前都会重新 seed。

补充练习。 提示长度已超过 block_size, max_new_tokens=2, 返回值会只有 block_size+2 吗？

答案。 不会。计算时仅裁条件窗口，完整 idx 保留原提示，因此返回原提示长度加 2。每次条件窗口内位置重新从 0 开始，老 token 离开窗口后不能再直接参与注意力。

11.6 采样入口的全部默认开关

字段	固定源码默认值	含义及边界
init_from	'resume'	resume 或四种 gpt2* 型号；采样只加载模型，不创建或恢复优化器，不支持 scratch
out_dir	'out'	checkpoint 目录；复用旧目录前先确认 init_from
start	'\n'	字面提示或 FILE:路径；必须编码成非空序列
num_samples	10	连续生成次数；循环内不会每次重新设 seed
max_new_tokens	500	每个样本新增 token 数；不含提示长度
temperature	0.8	正温度缩放 logits；原接口不验证范围
top_k	200	按第 k 大阈值排除候选；并列可能多于 k
seed	1337	初始化抽样 RNG；跨环境不保证逐位一致
device	'cuda'	采样模型和输入所在设备；此入口没有 DDP 分支
dtype	支持 CUDA BF16 则 bfloat16, 否则 float16	自动混合精度选择；CPU 不使用 autocast
compile	False	是否 torch.compile；本书关闭，不测编译收益

这些是原版默认，完整 CPU 实验显式覆盖设备、dtype、长度、样本数、温度、top-k 和 seed。

本章来源

- `sample.py`: 加载、编码、文件提示和输出
- `model.py`: `generate`
- 本书样本及重复性: `qa/author/tiny_sample.log`、`tiny_sample_repeat.log`; 并列阈值反例: `qa/author/mechanics.json`。

第 12 章 微调与 GPT-2 基线：域差异如何影响比较

12.1 微调复用了哪一部分训练程序

在本仓库中，微调仍是下一 token 交叉熵训练，使用同一个 train.py、同一个 AdamW 和同一个因果模型。关键变化是从预训练权重开始、使用目标领域数据、通常采用更小学习率和更短日程。它没有新增 LoRA、指令模板、偏好优化或聊天角色机制。把原始戏剧文本微调成能续写戏剧，不能直接推成“训练了一个会遵循指令的助手”。[微调配置](#)

`init_from='gpt2-xl'` 导入的是权重，不是此前 GPT-2 训练的优化器状态。本地会新建 AdamW。
`init_from='resume'` 才从本地 checkpoint 恢复优化器。这两种继续训练起点并不相同：一个保留参数知识、重新开始局部优化统计，另一个试图延续已有优化轨迹。

12.2 配置注释中的 epoch 是估算

微调配置设 $B = 1, A = 32, T = 1024$ ，单次更新处理 32768 token。上游 BPE Shakespeare 训练文件有 301966 token，所以除法给出约 9.2 次更新相当于一份文件大小的 token 曝光量。但 `get_batch` 是随机窗口采样，可能重叠且可能遗漏，程序没有完成一次严格遍历的 epoch 事件。

最多 20 的设置按原停止条件会从 0 到 20 执行 21 次更新。每 5 步评估、仅在改善时保存，输出目录 `out-shakespeare`。学习率固定 3×10^{-5} ，`decay_lr=False`；模型为最大 GPT-2 型号，所以内存需求并不“小”。把迭代次数设少不能解决一次前向就放不下模型的问题。

缩小模型型号、微批次或 `block_size` 各有不同作用。换型号改变参数和初始知识；减小微批次主要减少激活，但可用累积补回全局 token 数；减小长度减少上下文与注意力成本，也改变学习任务。三种调整不能只比较“是否不再 OOM”。

12.3 基线要对齐数据域与 tokenizer

固定 README 报告了 OpenWebText 上的 GPT-2 基线：

型号	上游标注参数量	上游 train loss	上游 val loss
gpt2	124M	3.11	3.12
gpt2-medium	350M	2.85	2.84
gpt2-large	774M	2.66	2.67

型号	上游标注参数量	上游 train loss	上游 val loss
gpt2-xl	1558M	2.56	2.54

这是固定 README 的历史结果，未在本书 CPU 环境重新复现。四个 `eval_gpt2*.py` 文件分别指定型号，其他设置相同：批次 8、评估 500 批、`eval_only=True`、关闭在线日志。评估数据仍继承 `train.py` 默认 OWT。基线报告

为什么从头训练的 124M 模型报告约 2.85，而原 GPT-2 是约 3.12，不能立刻判定实现“比 GPT-2 强”？GPT-2 的原训练语料 WebText 未公开，OWT 是另一个尽力接近它的集合。模型直接在 OWT 上训练，天然适应该分布；原权重未经适应就评估，比较包含数据域差异。README 还报告把 GPT-2 权重在 OWT 上继续训练后也能接近 2.85，因此更合理的比较应把这种适应机会考虑进去。

即使验证损失相同，也不保证生成风格、事实正确率、长文一致性或下游任务相同。交叉熵只评价给定数据上真实下一个 token 的概率。改变 tokenizer 后，单 token loss 更不能直接比较：字符一个位置与 BPE 一个位置携带的信息量不同。若一定跨 tokenizer 比较，需要定义共同单位，例如按原始字节归一化的负对数似然，并认真处理编码边界。

12.4 小数据微调的失败方式

验证越来越差而训练越来越好，可能意味着过拟合，也可能暴露训练与验证分布差异。只看训练 loss 选择 checkpoint 会奖励记忆训练集。原配置采用验证改善保存是一种基本保护，但随机窗口验证有噪声，频繁比较也会增加挑中偶然好结果的机会。

提示编码错配会让微调无效，即使 loss 在下降；第 2 章已说明必须保持预训练 token 身份。过高学习率可能迅速破坏预训练表示，过低则在少数更新内几乎不改变。较大 dropout 也不是无条件更好，尤其原配置实际上未启用它。可信调参应先固定数据和验证方法，每次只改变清楚定义的因素，记录结果和资源成本。

本书没有运行 GPT-2 微调，不提供捏造的最佳学习率或训练时长。合成小模型的恢复实验只能验证通用状态流，不能替代加载预训练权重、领域适应和大模型内存测试。

补充练习。 模型 A 在字符编码验证集上 `loss=1.5`，模型 B 在 BPE 验证集上 `loss=2.5`，A 一定更好吗？

答案。 不能这样排序。它们预测单位不同，还要确认原始文本、切分、上下文覆盖、归一化和评估协议。即使将两者都指数化为 perplexity，单位差异也不会消失。

本章来源

- [finetune_shakespeare.py](#)
- [四份 GPT-2 评估配置](#)
- [固定 README 的复现、基线和微调部分](#)

第 13 章 benchmark、MFU 与吞吐口径

13.1 bench.py 故意删掉了什么

bench.py 保留取批、前向、反向和优化器更新，去掉验证、checkpoint、学习率日程和 DDP 管理。默认仍是 12 层、12 头、768 宽、长度 1024、词表 50304 的模型。配置解析器能覆盖已定义的全局变量，却没有提供 `n_layer` 等全局开关；不能用 `--n_layer=2` 就把原版 bench 变成小模型。本书没有运行默认大 benchmark。bench.py

`real_data=True` 时，它只打开训练 memmap 一次，`get_batch(split)` 忽略 `split` 参数，始终从训练数据取窗，并无条件使用 `pin_memory` 和设备搬运。这与 `train.py` 每次重建映射、区分 CPU/CUDA 的实现不同。`real_data=False` 则预先生成固定随机 `x/y`，每次返回同一对张量，减少数据路径干扰，但测到的任务也不同。

简单计时分两段，先 10 步预热，再 20 步测量；阶段前后调用 `torch.cuda.synchronize()`，保证排队的 CUDA 工作完成再读时间。即使指定 `device=cpu`，这个无条件 CUDA 调用仍使原脚本不是直接可用的 CPU benchmark。不能把本书的小 CPU 计时写成原 bench.py 性能。

13.2 每秒 token 数需要说明分子和分母

若 S 次更新，每次处理 K 个训练 token，总用时 Δt ，训练吞吐为 $SK/\Delta t$ 。分子应说明是否含所有 rank、是否含累积微批次；分母应说明是否含数据读取、验证、保存、编译和日志。生成吞吐通常只数新生成 token，而且逐 token 自回归，不能直接与并行训练吞吐比较。

bench 的计时循环包含取批、优化器、`loss.item()` 和打印，因此它测的是一段具体循环，而非纯矩阵内核。真实数据与固定数据两种模式之间的差值也不一定全部等于“磁盘瓶颈”，缓存、传输、同步都可能参与。可靠对照应固定 batch、长度、精度、模型、编译和机器状态。

13.3 FLOPs 公式从哪里来

`estimate_mfu(fwdbwd_per_iter, dt)` 使用

$$F_{\text{token}} = 6N + 12LHQ T = 6N + 12LCT,$$

其中 N 是减掉位置嵌入后的参数数， $Q = C/H$ 为每头宽度， T 使用配置 `block_size`。再乘 T 得到每条序列的前后向 FLOPs，乘 `fwdbwd_per_iter` 得到本次计时覆盖的序列总数，除以秒数得到 FLOPs/s。

MFU 实现

$6N$ 的直觉是多数权重矩阵每 token 前向约两次浮点操作，反向约前向两倍，合计约六次。注意力中的 QK 和 AV 还有随上下文长度增长的工作量，第二项作补充。这个计算把乘和加计作两个 FLOP；若硬件资料用 MAC 为单位，直接代入会差一个倍数。它忽略或简化了归一化、softmax、激活、优化器与实际内核细节。

MFU 再除以源码硬编码的 312×10^{12} ，这是所选 A100 BF16 峰值口径。换 CPU、其他 GPU、精度或稀疏峰值，分母应重新定义。把 CPU 实验打印出的微小 MFU 当作“CPU 利用率”没有意义；它只是除错了比较对象。

13.4 DDP 的 MFU 分子应与分母匹配

训练循环传 `batch_size * 本地累积次数`，估算单 rank 的工作量，再与单卡峰值比较。只有各 rank 工作量相等、用时代表共同速度时，这才可近似理解为整体同构集群的平均利用率。不能把全局 token 工作量除以单卡峰值，否则数值会随卡数虚增；也不能本地工作量除以全机峰值，否则虚降。

dt 的测量不一定等于严格同步的 GPU 计算窗口，前五个本地迭代也不计算 MFU，后续使用 0.9 旧值加 0.1 新值平滑。初始 `running_mfu=-1` 会打印 -100%，这是“尚未测量”的哨兵值，不是负利用率。

若实际输入长度小于 `config.block_size`，估算仍用配置长度，FLOPs 会高估。测试 crop 后应确认 config 已更新。若运行激活重计算，真实执行 FLOPs 可增加，而“模型理论工作量”不一定随之增加；MFU 与硬件 FLOPs 利用率的口径也应分开。

13.5 profiler 能回答另一类问题

`profile=True` 时设置 5 步等待、5 步预热、5 步活动记录；活动包括 CPU 和 CUDA，输出 TensorBoard trace。`with_flops=True` 请求估计部分算子 FLOPs，`record_shapes=False`、`profile_memory=False`、`with_stack=False` 关闭部分额外记录。关闭堆栈减少开销，却也减少定位信息；不能既不记录形状和显存，又希望报告回答所有显存问题。

`torch.compile` 的首次编译成本应与稳态收益分开。README 中约 250ms 到 135ms 的数字是 2022 年末某组历史实验，不是当前机器通用承诺。编译还可能改变 state dict 包装前缀，恢复代码因此有清理逻辑。性能变化、数值误差和保存兼容性应分别测试。

补充练习。 两份报告分别写“100000 token/s”和“50% MFU”，能直接判断谁更快吗？

答案。 不能。需要模型规模、上下文、全局/本地口径、硬件及峰值定义。较大模型即使 token/s 较低，也可能执行更多 FLOPs 并达到更高 MFU。先把分子分母补齐，再比较相同任务。

13.6 benchmark 的全部默认开关

字段	固定源码默认值	含义及边界
batch_size	12	每进程每微步序列数；不是全局 batch
block_size	1024	每序列长度；限制窗口和位置表
bias	False	此处新建模型的 Linear/LN 是否带偏置；不走预训练导入
real_data	True	True 读 OWT；False 复用固定随机张量
seed	1337	初始化抽样 RNG；跨环境不保证逐位一致
device	'cuda'	benchmark 的单进程设备；此入口没有 DDP 分支
dtype	支持 CUDA BF16 则 bfloat16，否则 float16	自动混合精度选择；CPU 不使用 autocast
compile	True	是否 torch.compile；本书关闭，不测编译收益
profile	False	启用 CPU/CUDA trace；否则预热后简单计时

模型层数、头数和宽度在构造处写死；表中不包含它们，是因为它们不是这个入口支持的命令行配置键。

本章来源

- [bench.py](#) 的数据、预热和 profiler 路径
- [model.py](#): `estimate_mfu`
- PaLM 原论文，源码引用其 FLOPs 口径
- 本章没有 GPU 性能实测；数值公式重跑见第 14 章。

第 14 章 参数估算与 scaling laws：逐单元核对

14.1 先把 notebook 当成待验证的计算记录

transformer_sizing.ipynb 有 17 个单元，scaling_laws.ipynb 有 26 个。它们混合了可执行公式、实验假设、历史测量、绘图和未解决疑问。不能因为某个公式印在 notebook 里，就把旁边的 GPU 时间也当成自己测过。

本书用 examples/check_notebooks.py 按原顺序执行全部代码单元，只删除 Jupyter 专用的 %matplotlib inline 显示指令，图使用离线 Agg 后端保存。没有改变模型公式或网格范围，没有下载数据或权重。每个单元的 stdout 单独保存在 qa/author/notebook_execution.json；Markdown 单元只阅读和解释，不标为执行成功。下文单元编号从 0 开始，与原始 JSON 索引一致。两份 notebook 的固定版本

14.2 sizing：精确参数、近似 FLOPs 与假设时间

14.2.1 参数与存储：单元 0—7

单元 0 声明目标是估参数、FLOPs、峰值内存和 checkpoint。实际后续并没有完成完整峰值内存模型，最后还留下带宽 TODO；开头的目标不能替代实现。

单元 1 引入 OrderedDict，保证分项表顺序稳定。单元 2 选

$T = 1024, V = 50257, L = 12, H = 12, C = 768$ 并断言 bias=False。它与 GPTConfig 默认的词表 50304、bias=True 不同，所以比较参数前必须统一设置。

单元 3 的 params() 按第 4 章的无偏置公式累加：位置、token embedding、注意力 LN/QKV/投影、MLP LN/扩展/压缩、最后 LN。dense=0 是因为共享矩阵已在 token embedding 算过。表中 embedding、block 等是小计，不能把所有行再相加，否则重复计数。源码把 position 键拼成 embedding/position，其他函数沿用同一键，虽拼写不规范却不影响程序。本次得到 124337664，与单元硬编码的期望数一致。

单元 4 估计 FP32 参数占 $4N$ 字节，AdamW 两个状态张量再占 $8N$ ，合计 $12N = 1492051968$ 字节，约 1.49 GB。它不包括训练时梯度、激活、临时 workspace，也不是任意 optimizer 的通用公式。单元中的 1542470366 字节“实测 checkpoint”是上游历史常量；此次只重新做了除法，没有生成或测量那个大 checkpoint。

单元 5 引出显存比例，6 用 $40e9$ 字节作 GPU 容量分母，得参数加优化器约占 3.73%；7 据此讨论小模型中激活可能占较大份额。这里的 40 GB 使用十进制近似，不是通过设备查询得到的可用容量。完整训练显存还应算梯度约 $4N$ 、前向保存激活、通信 bucket、分配器缓存等；占比小不保证某个 batch 一定放得下。

14.2.2 FLOPs：单元 8—10

单元 8 过渡到前向成本，9 定义 `flops()`。一次 $(m, k) \times (k, n)$ 矩阵乘法按 $2mkn$ FLOPs 计。每条长度 T 的序列，一层 QKV 投影为 $6TC^2$ ，输出投影为 $2TC^2$ ，两个 MLP 矩阵共 $16TC^2$ ，所以矩阵主干共 $24TC^2$ 。QK 分数和 AV 聚合各 $2T^2C$ ，合计 $4T^2C$ 。最后输出头为 $2TCV$ 。于是

$$F_{\text{forward}} = L(24TC^2 + 4T^2C) + 2TCV.$$

反向被近似为前向两倍，训练为前向三倍。这个估算未逐算 LayerNorm、softmax、GELU、dropout 与优化器，也按完整方阵成本计注意力，不模拟具体因果内核跳过哪些操作。本次前向 291648307200 FLOPs，前后向合计 874944921600 FLOPs，都是公式结果，不是 profiler 读取的硬件计数。

单元 10 的 `palm_flops()` 用 $6N + 12LCT$ 乘序列长度， N 扣位置嵌入但保留共享 token 矩阵。本次得到 875062886400，与逐矩阵估算差 117964800，比例约 1.000135。差值主要对应简化参数口径中被 $6N$ 算入的 LayerNorm 缩放项。两式接近增加了算术一致性的信心，不能证明二者都精确模拟实际芯片指令。

14.2.3 历史速度与预算：单元 11—16

单元 11 说明作者曾以总 batch=100、单次 0.755 秒测量；12 将这些固定数字代入 $F \times 100/0.755$ ，再除 A100 312 TFLOPS；13 表达希望利用率更高。此次重跑复算的是历史输入，不是重新测出 0.755 秒。对本书 CPU 套这个分母没有设备利用率意义。

单元 14 用 $6ND$ ，假定 $D = 300 \times 10^9$ 、8 张 A100、MFU=0.3，估算总时间。本次计算是 3.459358974 天。单元 15 提到作者约四天的历史训练并给一篇解释链接；它与估算接近，但不是同一测量。单元 16 明确还没有估 LOAD/STORE 成本，说明整份 notebook 不能用来证明模型一定是算力受限而非带宽受限。

sizing 单元	本书解释位置	本次证据或边界
0、1、2	14.2.1 的目标、容器、无偏置配置	1、2 已执行；0 为说明
3	14.2.1 的参数分项与共享	已执行，124337664
4、5、6、7	14.2.1 的 checkpoint 与显存口径	4、6 复算；历史字节数未实测
8、9、10	14.2.2 的逐矩阵与 PaLM 近似	9、10 已执行
11、12、13	14.2.3 的历史 MFU	12 复算；无 GPU 实测

sizing 单元	本书解释位置	本次证据或边界
14、15、16	14.2.3 的预算和未完成带宽估计	14 已执行；15 历史，16 TODO

14.3 scaling：参数口径中藏着的差异

单元 0 就承认无法精确匹配所有 Chinchilla 数字，这个限制应随结果保留。单元 1 导入 matplotlib、NumPy、pandas 和 notebook 显示 magic；本书仅移除 magic。单元 2 引出参数分析，3 定义 gpt_params，4 声称与 124M 相符。scaling_laws.ipynb

gpt_params 虽然计算了 embeddings，最终却不加它，而另外加一个 CV 的输出 dense。共享权重下，这样保留了一份 token 矩阵，却遗漏位置表。它还包含线性和 LayerNorm 偏置。对 GPT-2 小配置得到 123653376，正是有偏置模型扣掉位置表后的数，不是完整 `sum(model.parameters())` 的 124439808。单元 4 的“完全匹配”文字应按这个口径纠正，不能只把四舍五入后的 124M 当精确一致。

单元 5 的 chinchilla_params 加入一项相对位置参数 $C^2 + 2C$ ，MLP 宽度改为传入值，其余仍作简化；参数 num_heads 没有参与公式。单元 6 存入从论文表格转录的 50 个模型，每行包括参数量、模型宽度、MLP 宽度、KV 宽度、头数、层数；7 只打印末五行估算对照；8 承认近似。计算函数没有接收独立 KV 宽度，部分表行的投影维数关系不能靠 C 一项完整表达，这也是不能宣称精确重建所有架构的原因。50 行数据被完整执行和保留在配套 notebook，不为正文逐行复制大表。

14.4 scaling 的 FLOPs 不计哪两项

单元 9 引出 FLOPs，10 的 chinchilla_flops 把每头宽度设为 `d_model // num_heads`，计算 QKV、QK、softmax、AV、输出投影及 MLP。它还算了 embedding 和 logits 两项，但最终 forward_flops 只取层数乘 attention+MLP，不把这两项加进去。源码注释把这个选择归于作者通信，本书没有独立核实该私人通信，不将其升级成“论文官方勘误”。

因此这里的“精细 FLOPs”与 sizing notebook 含输出头的公式不完全同口径。单元 11 构造论文表 A4 的六组结构，计算上述 FLOPs 除以 $6DN$ 的比值，再用 pandas 表展示；12 评论相近但参数尚不精确。本次六个比值依次约为 1.0250、1.1008、1.0829、1.0441、1.0329、0.9941。相近不等于精确，“exact”变量名也不能替代误差边界。

在小模型、大词表下，省掉输出头可能很重要；在大模型主干占比很高时，相对影响可能变小。若拿这些估算安排预算，至少先说明是否包含 embedding、输出 logits、反向、优化器和重计算。

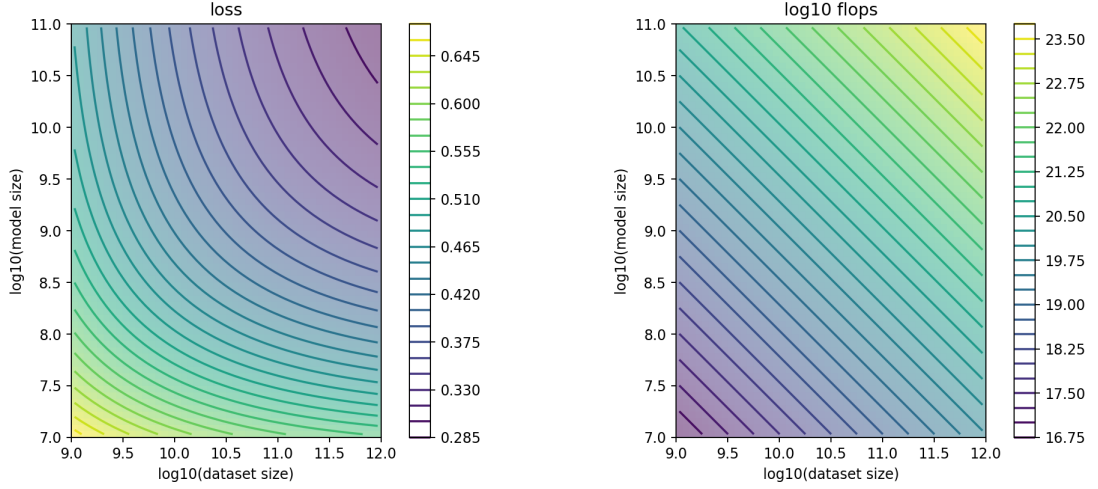
14.5 Approach 3：从固定预算推最优 N、D

单元 13 提出损失拟合，14 定义

$$\mathcal{L}(N, D) = AN^{-\alpha} + BD^{-\beta} + E,$$

其中 N 是参数数量, D 是训练 token 数, 原单元取

$A = 406.4, B = 410.7, \alpha = 0.34, \beta = 0.28, E = 1.69$ 。这些是特定论文实验下的拟合常数, 不是任意语料、tokenizer、模型的自然常数; 特别不能把 E 当所有自然语言共有、已证明的熵下界。单元同时画损失与 $6ND$ 网格图。图展示这个拟合函数, 不展示本书训练所得数据。左图源码先取了 $\log_{10}$, 因此色条表示 $\log_{10} \mathcal{L}$, 不是原始 loss; 标题只写 loss 不够精确, 读图时须保留这个换算。[Chinchilla 原文](#)



单元 15 问固定预算 C_{budget} 时怎样选模型和数据。用 $C_{\text{budget}} = 6ND$ 消去 D :

$$D = \frac{C_{\text{budget}}}{6N}, \quad \mathcal{L}(N) = AN^{-\alpha} + B\left(\frac{6N}{C_{\text{budget}}}\right)^{\beta} + E.$$

对 N 求导并令零:

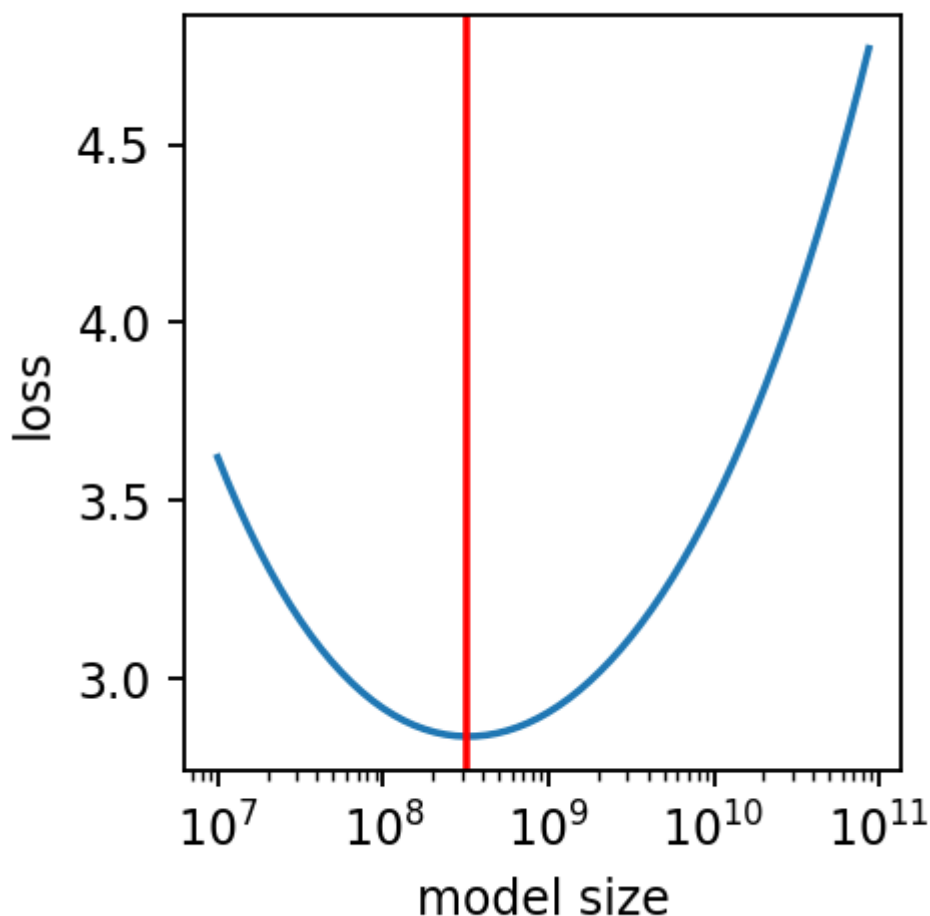
$$-\alpha AN^{-\alpha-1} + \beta B(6/C_{\text{budget}})^{\beta} N^{\beta-1} = 0,$$

所以

$$N_* = \left[\frac{\alpha A}{\beta B} \left(\frac{C_{\text{budget}}}{6} \right)^{\beta} \right]^{1/(\alpha+\beta)}, \quad D_* = C_{\text{budget}} / (6N_*).$$

这一步展示了权衡: 模型太小, 第一项偏大; 模型太大, 同预算下数据变少, 第二项偏大。 N_* 随预算约按 $C_{\text{budget}}^{0.4516}$ 增长, D_* 约按 $C_{\text{budget}}^{0.5484}$ 增长。它不是在任何规模上都精确要求“每参数 20 token”。

单元 16 用 2.21×10^{19} FLOPs 和离散网格搜最小点, 17 指出结果与引用表格仍不一致。本书补充的解析解为约 326.12M 参数、11.294B token, 拟合 loss 2.83719; 原离散网格点更接近 316.23M。解析解去除了网格离散误差, 仍不会自动消除拟合与论文其他方法的差异。



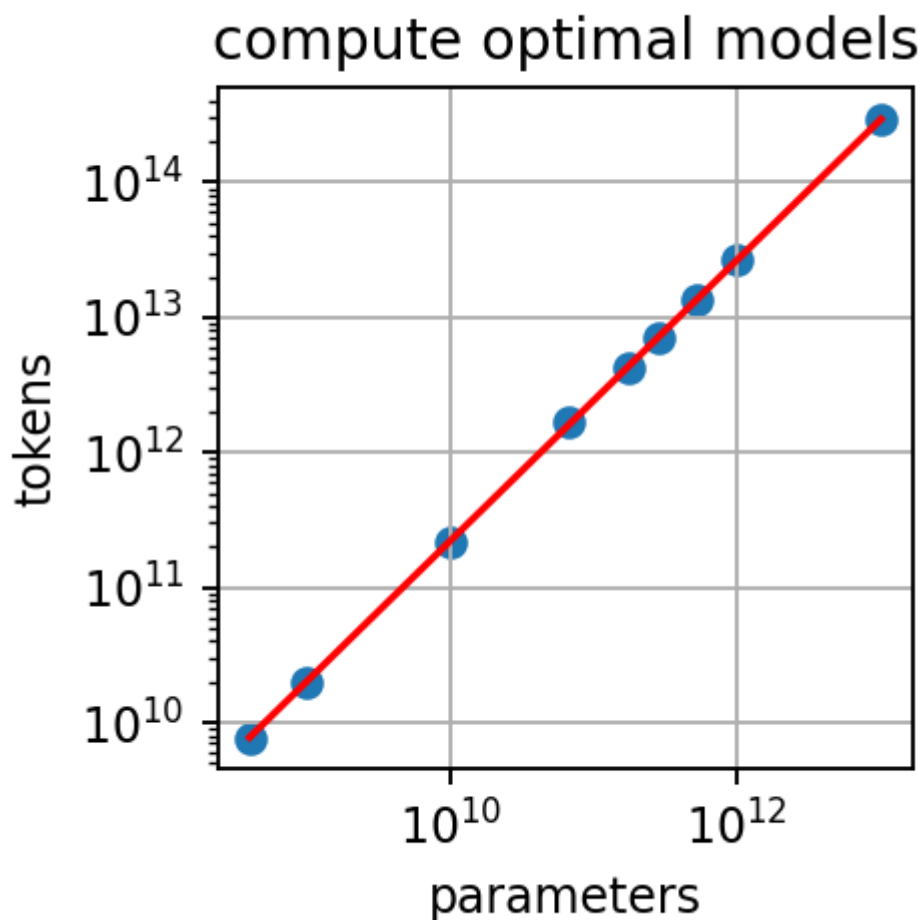
单元 18 对从 10^{17} 到不足 10^{26} 的 2304 个预算点重复网格搜索，19 找到接近 400M 的模型并输出其数据与计算量，20 再指出与另一表值不同。最近邻查询既含参数网格误差，也含预算网格误差；结果不应包装成精确的最优训练处方。

14.6 Approach 2：对表格作回归，不是重新训练模型

单元 21 描述另一种按固定算力实验拟合的思路，22 录入九组参数/token 对；注释还保留另一组 Approach 1 数据，但实际运行使用 Approach 2。23 在对数空间最小二乘拟合

$$x = \log_{10} N, \quad y = \log_{10} D, \quad y = mx + c,$$

设计矩阵每行为 $[x_i, 1]$ 。本次得到 $m = 1.0409573169995878$ 、 $c = 0.9353887152390967$ ，所以 $D = 10^c N^m$ 。24 绘原始点与拟合线，25 查询 $N = 124 \times 10^6$ ，得到约 2.2924×10^9 token。



最后单元打印文字把 parameters 和 tokens 的角色写反了；按代码，输入是参数量，输出是预测 token 数。124M 还小于录入表的最小模型 400M，因此这是向下外推，不是数据区间内插值。这个约 2.29B 也不与 train_gpt2 配置的约 300B 构成代码矛盾：一个是特定近似下的计算最优建议，另一个是仓库为历史 GPT-2 复现目标设定的训练曝光量。目标不同，预算和评价也不同。

scaling 单元	本书解释位置	执行与限定
0、1、2	14.1、14.3	1 执行，显示 magic 移除；0、2 为说明
3、4	14.3 GPT 参数口径	3 执行；纠正 4 的精确匹配措辞
5、6、7、8	14.3 Chinchilla 参数近似	5—7 执行，50 行数据保留；8 为说明
9、10、11、12	14.4 FLOPs 排除项与六行对照	10、11 执行；9、12 为说明
13、14、15	14.5 损失拟合及预算	14 执行并保存图；13、15 为说明
16、17	14.5 单预算网格与解析解	16 执行；17 的不一致保留
18、19、20	14.5 多预算与近邻	18、19 执行；20 为差异提醒

scaling 单元	本书解释位置	执行与限定
21、22、23、24、25	14.6 表格回归、图及查询	22—25 执行，纠正 25 的打印标签

14.7 练习：公式一致不等于问题一致

补充练习 1。 为什么 sizing 的 124337664 与 scaling 的 123653376 不同，不能只归咎于舍入？

答案。 前者无偏置、包含位置表，后者有偏置、不含位置表。两者字段口径不同，差值是可逐项计算的，远不只是 M 单位显示精度。

补充练习 2。 若预算固定，直接把 N 翻倍而不改变 D ，还在同一条预算曲线上吗？

答案。 不在， $6ND$ 翻倍。保持预算就要把 D 减半，再评估两项损失变化。给推理部署使用的模型选择还涉及推理成本和延迟，本 notebook 的训练 FLOPs 最小损失目标没有包括这些约束。

本章来源

- [transformer_sizing.ipynb](#) 固定源码
- [scaling_laws.ipynb](#) 固定源码
- [Training Compute-Optimal Large Language Models](#)
- [PaLM: Scaling Language Modeling with Pathways](#)
- 全单元 stdout、数值表、解析解：qa/author/notebook_execution.json；完整可运行入口 examples/check_notebooks.py。

第 15 章 小规模复现、故障定位与仓库的历史边界

15.1 一条能在普通 CPU 上走完的路线

本书的首要验证目标是让原版的数据读取、模型、训练、保存、恢复和生成接成闭环，并用独立断言检查关键数学性质。运行规模刻意很小：五字符合成序列、2 层、2 头、宽度 16、窗口 8、微批次 2、累积 2。它不需要网络、API、预训练权重或大型语料。

配套目录应保留以下结构：

nanogpt_book/	
chapters/	分章正文
examples/	四个完整可运行入口
figures/	原创图与 notebook 重绘图
requirements-cpu.txt	本次验证依赖
research/upstream/	固定 MIT 源码与 notebook
research/UPSTREAM_LICENSE.txt	
research/author/	逐文件、定义、单元和配置覆盖
qa/author/	实验 JSON；日志与隔离运行目录由执行生成

在可信的本地目录解压。建议创建独立虚拟环境，依赖安装需联网，但下面四个实验本身离线。下面命令对应本书已经验证的 Linux CPU 环境，PyTorch 使用其 CPU 官方源；不要把不同 CUDA 构建的结果混为本次环境。

```
python -m venv .venv
. .venv/bin/activate
python -m pip install torch==2.6.0 --index-url https://download.pytorch.org/whl/cpu
python -m pip install -r requirements-cpu.txt
export OMP_NUM_THREADS=1
export MKL_NUM_THREADS=1
export OPENBLAS_NUM_THREADS=1
export MPLBACKEND=Agg
export MPLCONFIGDIR="$PWD/qa/author/mpl_cache"
python examples/tiny_pipeline.py
python examples/verify_mechanics.py
python examples/check_notebooks.py
python examples/make_figures.py
```

macOS 不应照抄上面的 Linux CPU 索引安装行。按 [PyTorch 2.6.0 官方历史安装页](#) 的 OSX 路径，可将该行替换为 `python -m pip install torch==2.6.0`，其余依赖另装；本书不需要 torchvision 或 torchaudio。是否有匹配的 wheel 还取决于 Mac 架构与 Python 版本，若提示无匹配版本，先核对官方支持条件。本书没有在 macOS 或 MPS 上实测，四个教学入口仍明确使用 CPU。

配套 ZIP 不包含 checkpoint、虚拟环境、缓存或 `tiny_run` 隔离运行目录；这些由运行脚本在本机创建。

本次真实环境是 Linux、Python 3.12.14、PyTorch 2.6.0+cpu、NumPy 2.3.5、Matplotlib 3.10.8、pandas 2.2.3、tiktoken 0.9.0，CPU 线程 1。版本约束用于复现记录，不声称这些版本对所有操作系统都是最合适的新安装选择。跨硬件、BLAS 或库版本时，不要求 loss 最末位与本文一致，应优先核对断言和误差范围。

15.2 四个完整程序各负责一件事

`tiny_pipeline.py` 从固定上游复制 `train.py`、`model.py`、`sample.py`、`configurator.py` 和 LICENSE 到隔离 `qa/author/tiny_run`，四份 Python 的执行副本哈希与源文件逐一相等。它生成 `train/val` 二进制及字符元数据，写小配置，然后用 `subprocess` 运行原 `train.py`。训练 seed 为原版的 1337，采样 seed 为 17，子进程限制 CPU 线程并禁用在线日志。脚本不会执行原数据下载器。

第一阶段 `max_iters=20`，checkpoint 在 step 20 保存；第二阶段从该文件恢复，设置 `max_iters=25`，确认保存编号变成 25、至少一个权重发生变化。然后两次独立启动 `sample.py` 并比较完整输出。只加载本次自身创建的 checkpoint，不接收任意文件路径作为反序列化入口。

`verify_mechanics.py` 检查参数计数、共享身份、优化器去重、因果性、SDPA/手工路径、整批/累积梯度、共享参数有限差分、上下文裁剪、非连续目标反例、学习率边界、窗口尾端、uint16 和配置解析反例。它用 AST 从可信 `train.py` 抽取 `get_lr`，而不是 `import train.py` 后意外启动训练。所有数学比较都记录实际误差。

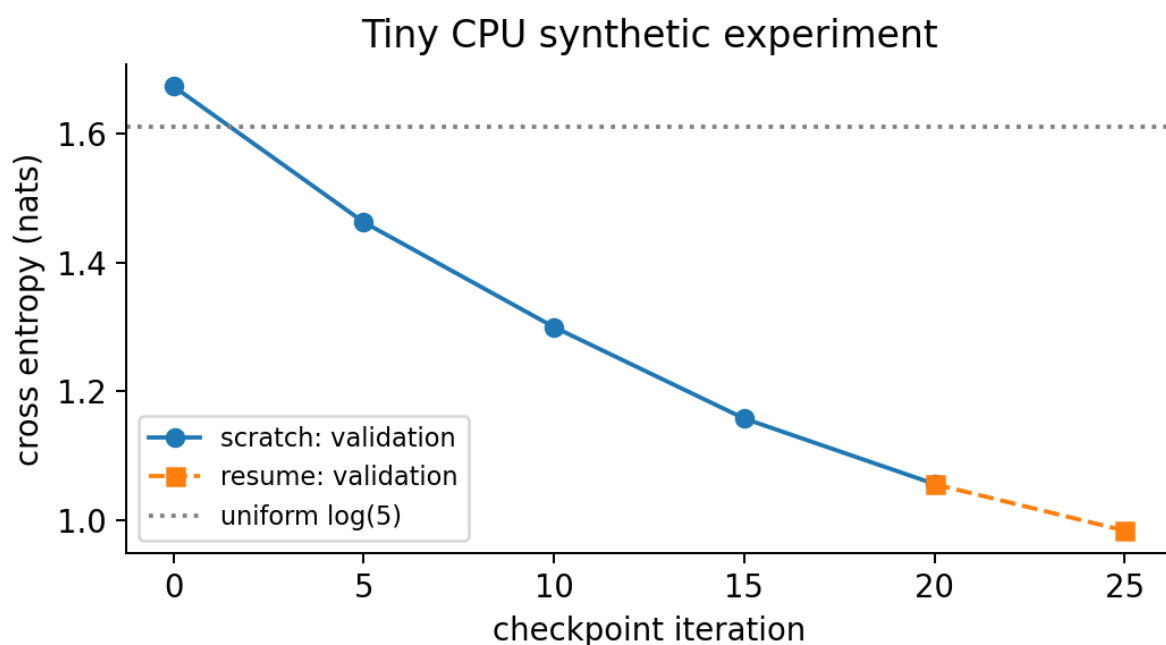
`check_notebooks.py` 顺序执行两份 notebook 全部代码单元，单元级记录 `stdout`，保留原公式和表格。它只是复算历史 GPU 数字，不执行 GPU 训练。`make_figures.py` 绘制数据流、因果掩码、窗口错位、学习率、微型验证曲线和 DDP 计数图；图中 DDP 是解释图，不是性能记录。

所有入口都按 `__file__` 找本书根目录，不依赖作者机器上的绝对路径。源码包保留这些完整文件，可打印附录由清单的 `appendix=true` 项生成；必要数据、原始大源码、图和日志分别列出，不把一份日志冒充另一份程序。

15.3 真实输出怎样支持结论

检查	结果	能支持什么
小模型参数计数	1864，全数；1816，扣位置	公式和实现计数一致
修改后缀后的前缀误差	0	测试输入下没有未来泄漏
SDPA 对手工 attention	最大差 2.78×10^{-17}	double、dropout=0 时数值一致

检查	结果	能支持什么
累积对整批梯度	最大差 5.55×10^{-17}	等有效样本、无 dropout 的平均梯度一致
共享参数有限差分	误差 7.18×10^{-10}	同时涵盖输入/输出路径的梯度正确
训练→恢复 checkpoint	20→25	原版状态加载和继续更新通畅
同 seed 两次采样	完整 stdout 一致	本机同环境的这次抽样可复现
两份 notebook	所有代码单元执行	公式可运行；不等于硬件历史重测



曲线的 step 20 有两个接近但不同的评估值，是恢复时重新抽样造成的。首次 train loss 从 1.6792 降到 1.0649，val 从 1.6728 降到 1.0560；恢复 step 25 的 train 为 1.0005，val 为 0.9835。时间日志保留供审计，但这里不把数毫秒的小模型循环转写为 GPU 性能承诺。

15.4 按失败发生的位置排查

现象	先核对什么	不要直接做什么
Unknown config key	参数名是否由该入口定义	不要假定 train、sample、bench 开关相同
类型断言失败	Python 字面值、大小写、原类型	不要把所有数字都加小数点
randint 上界错误	文件 token 数是否大于窗口	不要只看文件存在

现象	先核对什么	不要直接做什么
embedding 越界	dtype、最大编号、词表一致性	不要盲目扩词表掩盖读盘损坏
targets.view 报错	裁切后连续性	不要怀疑所有权重都损坏
恢复模型形状不符	model_args、词表、前缀清理	不要随意忽略缺失键
恢复后优化器形状不符	是否载了位置表而没迁移状态	不要认为权重加载成功就足够
loss 极低但生成很差	目标错位、因果泄漏、编码器	不要只增大模型
最后状态未保存	保存间隔、更新前保存、停止条件	不要把正常退出等同最终落盘
多卡挂起或断言	A/W、rank、设备、collective 次序	不要立即改网络安全设置

排查时先复现最小失败，再分清数据、数学、状态与系统。第 6 章的非连续目标就是一例：原 loader 正常，自己切片后失败。用具体张量步幅解释，比泛泛“升级 PyTorch”更有效。

15.5 仓库外围文件也有工程含义

.gitignore 忽略 .DS_Store、IDE 目录、notebook checkpoint、Python 缓存、*.bin、*.pkl、*.pt、input.txt 以及 env/venv。它防止常见大数据和运行产物意外进入版本库，却不会自动备份这些文件，也不能保证敏感数据永不提交。复现实验应另行保存数据来源、哈希、配置和必要 checkpoint；Git 工作区干净不代表实验可重建。

.gitattributes 将 *.ipynb 标为 linguist-generated，只影响 GitHub 语言统计等展示判断，不代表 notebook 没有教学内容、更不代表它会自动生成或经过测试。本书因此仍逐单元审计。

assets/nanogpt.jpg 是 README 的展示图，assets/gpt2_124M_loss.png 是上游历史损失图。它们在覆盖清单中保留路径、哈希与用途，正文不直接复用未单独核验的图片权利，也不从像素反推出精确训练数值。本书图形来自原创绘制或 MIT notebook 公式重绘。

LICENSE 为 MIT，版权归属 Andrej Karpathy，完整副本在 research/UPSTREAM_LICENSE.txt。重分发上游实质性源码时保留版权及许可，正文的新解释、习题与图解不暗示得到上游作者认可。README 的安装、快速开始、GPT-2 复现、基线、微调、采样、效率、待办、排错、社区和致谢均在覆盖清单中映射到相应章节；社区入口和赞助致谢是项目背景，不是运行依赖。

15.6 能继续研究的边界

原 README 待办包括 FSDP、零样本评测、微调调参、动态 batch、其他位置编码、优化器状态分离和更多训练健康日志。它们解释了固定仓库尚未覆盖的方向，不是已经验证的功能。本书也没有完成 GPU AMP、fused AdamW、编译收益、NCCL、跨节点、GPT-2 权重导入或完整 OWT 训练。

下一步若有合适硬件，可以从一个清楚的问题开始：同一小模型的 GPU 前向是否与 CPU 在容差内一致？双卡全局 batch 是否与单卡累积一致？恢复运行保存哪些额外状态才能匹配连续运行？每次只扩大一个边界，记录实际证据，比直接启动几天训练后只留下一个 loss 数字更可控。

补充练习。 为“严格恢复”设计一个验收，而不改训练算法。

答案。 固定代码、硬件、数据及全部随机源，比较连续运行 K 步与先运行 J 步、保存完整状态、再恢复 K-J 步。逐步比较下一批数据、学习率、scaler、参数及优化器状态。若任何一项不同，先定位首次分叉点。原版 checkpoint 不满足完整状态要求，因此这应标为后续扩展实验，不能在本书现有恢复测试上直接打“严格复现通过”。

本章来源

- 固定 README 与完整文件树
- MIT LICENSE
- Git 忽略规则、Git 属性
- 本书完整结果：qa/author/tiny_pipeline.json、mechanics.json、notebook_execution.json；
逐项覆盖：research/author/coverage.json。

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 1 章 沿一次训练看完整仓库

- <https://github.com/karpathy/nanoGPT/tree/3adf61e154c3fe3fca428ad6bc3818b27a3b8291>
- <https://github.com/karpathy/nanoGPT/blob/3adf61e154c3fe3fca428ad6bc3818b27a3b8291/README.md>
- <https://github.com/karpathy/nanoGPT/blob/3adf61e154c3fe3fca428ad6bc3818b27a3b8291/train.py>

第 2 章 字符与 BPE：三条数据准备流水线

- <https://github.com/karpathy/nanoGPT/tree/3adf61e154c3fe3fca428ad6bc3818b27a3b8291/data>
- https://github.com/karpathy/nanoGPT/blob/3adf61e154c3fe3fca428ad6bc3818b27a3b8291/data/shakespeare_char/prepare.py
- <https://github.com/karpathy/nanoGPT/blob/3adf61e154c3fe3fca428ad6bc3818b27a3b8291/data/shakespeare/prepare.py>
- <https://github.com/karpathy/nanoGPT/blob/3adf61e154c3fe3fca428ad6bc3818b27a3b8291/data/openwebtext/prepare.py>
- <https://github.com/openai/tiktoken>

第 3 章 配置覆盖、类型检查与执行边界

- <https://github.com/karpathy/nanoGPT/blob/3adf61e154c3fe3fca428ad6bc3818b27a3b8291/configurator.py>
- <https://github.com/karpathy/nanoGPT/tree/3adf61e154c3fe3fca428ad6bc3818b27a3b8291/config>
- <https://github.com/karpathy/nanoGPT/blob/3adf61e154c3fe3fca428ad6bc3818b27a3b8291/train.py>

第 4 章 GPTConfig、参数规模与初始化

- <https://github.com/karpathy/nanoGPT/blob/3adf61e154c3fe3fca428ad6bc3818b27a3b8291/model.py#L108-L148>
- <https://github.com/karpathy/nanoGPT/blob/3adf61e154c3fe3fca428ad6bc3818b27a3b8291/model.py#L108-L168>
- https://github.com/karpathy/nanoGPT/blob/3adf61e154c3fe3fca428ad6bc3818b27a3b8291/transformer_sizing.ipynb

第 5 章 逐层推导因果注意力、MLP 和残差

- <https://github.com/karpathy/nanoGPT/blob/3adf61e154c3fe3fca428ad6bc3818b27a3b8291/model.py#L18-L27>
- <https://github.com/karpathy/nanoGPT/blob/3adf61e154c3fe3fca428ad6bc3818b27a3b8291/model.py#L29-L76>
- <https://github.com/karpathy/nanoGPT/blob/3adf61e154c3fe3fca428ad6bc3818b27a3b8291/model.py#L18-L106>
- https://pytorch.org/docs/stable/generated/torch.nn.functional.scaled_dot_product_attention.html

第 6 章 权重共享、损失、预训练映射与上下文裁剪

- <https://github.com/karpathy/nanoGPT/blob/3adf61e154c3fe3fca428ad6bc3818b27a3b8291/model.py#L170-L193>
- <https://github.com/karpathy/nanoGPT/blob/3adf61e154c3fe3fca428ad6bc3818b27a3b8291/model.py#L207-L261>
- <https://github.com/karpathy/nanoGPT/blob/3adf61e154c3fe3fca428ad6bc3818b27a3b8291/model.py#L170-L261>
- <https://github.com/karpathy/nanoGPT/blob/3adf61e154c3fe3fca428ad6bc3818b27a3b8291/train.py>

第 7 章 memmap 批次与下一 token 对齐

- <https://github.com/karpathy/nanoGPT/blob/3adf61e154c3fe3fca428ad6bc3818b27a3b8291/train.py#L114-L132>
- <https://github.com/karpathy/nanoGPT/blob/3adf61e154c3fe3fca428ad6bc3818b27a3b8291/bench.py#L31-L49>

第 8 章 训练主循环、梯度累积、混合精度与裁剪

- <https://github.com/karpathy/nanoGPT/blob/3adf61e154c3fe3fca428ad6bc3818b27a3b8291/train.py#L249-L327>
- <https://github.com/karpathy/nanoGPT/blob/3adf61e154c3fe3fca428ad6bc3818b27a3b8291/model.py#L263-L287>
- <https://github.com/karpathy/nanoGPT/blob/3adf61e154c3fe3fca428ad6bc3818b27a3b8291/train.py>
- <https://pytorch.org/docs/stable/generated/torch.optim.AdamW.html>

第 9 章 学习率、验证、checkpoint 与恢复

- <https://github.com/karpathy/nanoGPT/blob/3adf61e154c3fe3fca428ad6bc3818b27a3b8291/train.py#L230-L243>
- <https://github.com/karpathy/nanoGPT/blob/3adf61e154c3fe3fca428ad6bc3818b27a3b8291/train.py>
- <https://pytorch.org/docs/stable/notes/serialization.html>

第 10 章 单机多卡与跨节点 DDP 的状态约束

- <https://github.com/karpathy/nanoGPT/blob/3adf61e154c3fe3fca428ad6bc3818b27a3b8291/train.py>
- https://github.com/karpathy/nanoGPT/blob/3adf61e154c3fe3fca428ad6bc3818b27a3b8291/config/train_gpt2.py
- <https://pytorch.org/docs/stable/generated/torch.nn.parallel.DistributedDataParallel.html>

第 11 章 生成接口：温度、top-k、提示和随机性

- <https://github.com/karpathy/nanoGPT/blob/3adf61e154c3fe3fca428ad6bc3818b27a3b8291/sample.py>
- <https://github.com/karpathy/nanoGPT/blob/3adf61e154c3fe3fca428ad6bc3818b27a3b8291/model.py#L306-L332>

第 12 章 微调与 GPT-2 基线：域差异如何影响比较

- https://github.com/karpathy/nanoGPT/blob/3adf61e154c3fe3fca428ad6bc3818b27a3b8291/config/finetune_shakespeare.py

- <https://github.com/karpathy/nanoGPT/blob/3adf61e154c3fe3fca428ad6bc3818b27a3b8291/README.md#baselines>
- <https://github.com/karpathy/nanoGPT/tree/3adf61e154c3fe3fca428ad6bc3818b27a3b8291/config>
- <https://github.com/karpathy/nanoGPT/blob/3adf61e154c3fe3fca428ad6bc3818b27a3b8291/README.md>

第 13 章 benchmark、MFU 与吞吐口径

- <https://github.com/karpathy/nanoGPT/blob/3adf61e154c3fe3fca428ad6bc3818b27a3b8291/bench.py>
- <https://github.com/karpathy/nanoGPT/blob/3adf61e154c3fe3fca428ad6bc3818b27a3b8291/model.py#L289-L303>
- <https://arxiv.org/abs/2204.02311>

第 14 章 参数估算与 scaling laws：逐单元核对

- <https://github.com/karpathy/nanoGPT/tree/3adf61e154c3fe3fca428ad6bc3818b27a3b8291>
- https://github.com/karpathy/nanoGPT/blob/3adf61e154c3fe3fca428ad6bc3818b27a3b8291/scaling_laws.ipynb
- <https://arxiv.org/abs/2203.15556>
- https://github.com/karpathy/nanoGPT/blob/3adf61e154c3fe3fca428ad6bc3818b27a3b8291/transformer_sizing.ipynb
- <https://arxiv.org/abs/2204.02311>

第 15 章 小规模复现、故障定位与仓库的历史边界

- <https://pytorch.org/get-started/previous-versions/#v260>
- <https://github.com/karpathy/nanoGPT/tree/3adf61e154c3fe3fca428ad6bc3818b27a3b8291>
- <https://github.com/karpathy/nanoGPT/blob/3adf61e154c3fe3fca428ad6bc3818b27a3b8291/LICENSE>
- <https://github.com/karpathy/nanoGPT/blob/3adf61e154c3fe3fca428ad6bc3818b27a3b8291/.gitignore>
- <https://github.com/karpathy/nanoGPT/blob/3adf61e154c3fe3fca428ad6bc3818b27a3b8291/.gitattributes>
- <https://download.pytorch.org/whl/cpu>

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/check_notebooks.py

```

"""Execute every computational notebook cell from pinned nanoGPT offline.
Removes only the Jupyter %matplotlib inline display directive. Logs cells separately.
MIT upstream portions: Copyright (c) 2022 Andrej Karpathy, ../research/
UPSTREAM_LICENSE.txt.
"""

from pathlib import Path
import contextlib, io, json, math, os
os.environ.setdefault('MPLCONFIGDIR', str(Path(__file__).resolve().parents[1]/'qa/author/
mpl_cache'))
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
import numpy as np
ROOT=Path(__file__).resolve().parents[1]

def main():
    report={}
    for name in ['transformer_sizing', 'scaling_laws']:
        notebook=json.loads((ROOT/'research/upstream'/f'{name}.ipynb').read_text())
        ns={}; cells=[]
        for i,c in enumerate(notebook['cells']):
            if c['cell_type']!='code':
                cells.append(dict(cell=i, status='read_markdown_not_executed')); continue
            source=''.join(c['source'])
            source='\n'.join(line for line in source.splitlines() if line.strip()!
='%matplotlib inline')
            buf=io.StringIO()
            with
contextlib.redirect_stdout(buf):exec(compile(source, f'{name}:cell{i}', 'exec'), ns)
            cells.append(dict(cell=i, status='executed_cpu', stdout=buf.getvalue()))
            if plt.get_fignums():
                for j,n in enumerate(plt.get_fignums()):
                    fig=plt.figure(n); fig.tight_layout()
                    fig.savefig(ROOT/'figures'/f'notebook_{name}_{i}_{j}.png', dpi=170)
                plt.close('all')
            report[name]={'cells':cells}
            if name=='transformer_sizing':
                report[name]['numeric']={'parameters':ns['params']()
['total'], 'fwd_flops':ns['flops']() ['forward_total'],
                                     'train_flops':ns['flops']()
['total'], 'palm_flops':ns['palm_flops'](),

```

```

        'estimated_days':ns['time_needed_s']/86400}
    else:
        a,b,A,B=.34,.28,406.4,410.7;budget=2.21e19
        n=(a*A/(b*B)*(budget/6)**b)**(1/(a+b));d=budget/(6*n)
        report[name]
    ['numeric']={'analytic_N':n,'analytic_D':d,'analytic_loss':ns['L'](n,d),

    'approach2_slope':float(ns['m']),'approach2_intercept':float(ns['c']),
        'approach2_tokens_124M':float(ns['yquery']),
        'gpt_params_124m':ns['gpt_params'](*ns['gpt2']),

    'all_chinchilla_models':len(ns['chilchilla_models']),
        'grid_budget_count':len(ns['models']),
        'table4':ns['df'].to_dict(orient='records')}
    assert abs(math.log(n/(10**8.5)))<.1
    (ROOT/'qa/author/notebook_execution.json').write_text(json.dumps(report,indent=2))
    print(json.dumps({n:v['numeric'] for n,v in report.items()},indent=2))

if __name__=='__main__':main()

```

examples/make_figures.py

```

"""Original explanatory figures. Matplotlib, offline, no generated image service."""
from pathlib import Path
import json, os
ROOT=Path(__file__).resolve().parents[1]
os.environ.setdefault('MPLCONFIGDIR',str(ROOT/'qa/author/mpl_cache'))
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
from matplotlib.patches import FancyBboxPatch
import numpy as np
D=ROOT/'figures';D.mkdir(exist_ok=True)
plt.rcParams.update({'font.size':11,'figure.facecolor':'white','axes.spines.top':False,'
axes.spines.right':False})
def save(fig,name):
    fig.tight_layout();fig.savefig(D/(name+'.png'),dpi=180);fig.savefig(D/
(name+'.svg'));plt.close(fig)
def main():
    fig,ax=plt.subplots(figsize=(9,3));ax.set(xlim=(0,10),ylim=(0,3));ax.axis('off')
    boxes=[(.2,1.6,'Text / tokenizer'),(3.5,1.6,'train.bin + val.bin'),
(6.8,1.6,'train.py + model.py'),(6.8,.2,'checkpoint'),(3.5,.2,'sample.py + tokenizer')]
    for x,y,label in boxes:
        ax.add_patch(FancyBboxPatch((x,y),2.6,.7,boxstyle='round,pad=.08',fc='#e8f0fa',ec='#365e
8a'))
        ax.text(x+1.3,y+.35,label,ha='center',va='center',fontsize=10)
        for a,b in [((2.9,1.95),(3.4,1.95)),((6.2,1.95),(6.7,1.95)),((8.1,1.5),(8.1,1)),
((6.7,.55),(6.2,.55))]:
            ax.annotate('',xy=b,xytext=a,arrowprops=dict(arrowstyle='->',lw=2))
        save(fig,'pipeline')

fig,ax=plt.subplots(figsize=(5,4));a=np.tril(np.ones((6,6)));ax.imshow(a,cmap='Blues',vm

```

```

in=0,vmax=1)
    for i in range(6):
        for j in range(6):ax.text(j,i,'read' if j<=i else
'mask',ha='center',va='center',color='white' if j<=i else '#333',fontsize=9)
        ax.set(xlabel='Key position j',ylabel='Query position
i',xticks=range(6),yticks=range(6),title='Causal mask: j <= i');save(fig,'causal_mask')

fig,ax=plt.subplots(figsize=(8,2.5));ax.axis('off');ax.set(xlim=(-.8,9.8),ylim=(-.8,2.5)
)
    for j in
range(10):ax.text(j,2,str(j),ha='center',bbox=dict(boxstyle='square',fc='#f4f4f4'))
        for start,y,label,color in [(5,1,'x','#d8eafe'),(6,0,'y','#fee2e2')]:
            ax.text(start-1,y,label,ha='right')
            for j in
range(start,start+4):ax.text(j,y,str(j),ha='center',bbox=dict(boxstyle='square',fc=color
))
        ax.set_title('N=10, T=4, last legal start i=5');save(fig,'batch_alignment')
fig,ax=plt.subplots(figsize=(6,3.4));it=np.arange(0,14);lr=[]
    for i in it:
        lr.append(.001*(i+1)/3 if i<2 else (.0001 if i>10 else
.0001+.0009*.5*(1+np.cos(np.pi*(i-2)/8))))

ax.plot(it,lr,'o-');ax.axvline(2,color='gray',ls='--');ax.axvline(10,color='gray',ls='--
');ax.set(xlabel='iteration',ylabel='learning rate',title='Actual get_lr: warmup=2,
decay_end=10');save(fig,'learning_rate')
    report=json.loads((ROOT/'qa/author/tiny_pipeline.json').read_text())
    import re
    def evaluations(lines):
        rows=[re.match(r'step (\d+): train loss ([0-9.]+), val loss ([0-9.]+)',line) for
line in lines]
        return [int(r[1]) for r in rows],[float(r[3]) for r in rows]

xs,ys=evaluations(report['train_evaluations']);rx,ry=evaluations(report['resume_evaluati
ons'])
    fig,ax=plt.subplots(figsize=(6,3.4))
    ax.plot(xs,ys,'o-',label='scratch: validation');ax.plot(rx,ry,'s--',label='resume:
validation');ax.axhline(np.log(5),color='gray',ls=':',label='uniform
log(5)');ax.set(xlabel='checkpoint iteration',ylabel='cross entropy (nats)',title='Tiny
CPU synthetic experiment');ax.legend(fontsize=9);save(fig,'tiny_loss')
    fig,ax=plt.subplots(figsize=(7,3.5));x=np.arange(3);width=.25
    values=[[2,2,2],[2,2,2],[2,2,2],[2,2,2]]
    # Each column is a local micro step, all ranks contribute before the update.
    for rank in range(4):
        ax.bar(x+rank*.18,np.array(values[rank])*4,width=.16,label=f'rank {rank}')
    ax.set(xticks=x+.27,xticklabels=['micro 0','micro 1','micro 2'],ylabel='tokens /
rank / micro',title='Example: W=4, A=12 -> local a=3; B=2, T=4')
    ax.legend(ncol=4,fontsize=8);save(fig,'ddp_tokens')
if __name__=='__main__':main()

```

examples/tiny_pipeline.py

```

"""Original book harness. Executes unmodified, pinned nanoGPT in an isolated copy.
No network, external logging, pretrained weights, or large corpus required.

```

```

Run from any working directory: python examples/tiny_pipeline.py
"""
from pathlib import Path
import hashlib, json, math, os, pickle, re, shutil, subprocess, sys
import numpy as np
import torch

ROOT = Path(__file__).resolve().parents[1]
SRC = ROOT / 'research/upstream'
WORK = ROOT / 'qa/author/tiny_run'
LOG = ROOT / 'qa/author'

def sha(path):
    return hashlib.sha256(path.read_bytes()).hexdigest()

def run(label, args):
    env = os.environ.copy()
    env.update(OMP_NUM_THREADS='1', MKL_NUM_THREADS='1', OPENBLAS_NUM_THREADS='1',
               CUDA_VISIBLE_DEVICES='', WANDB_MODE='disabled', HF_HUB_OFFLINE='1')
    result = subprocess.run([sys.executable] + args, cwd=WORK, env=env,
                             text=True, capture_output=True, timeout=120)
    (LOG / f'{label}.log').write_text(result.stdout + '\nSTDERR\n' + result.stderr)
    if result.returncode:
        raise RuntimeError(f'{label} failed; inspect {LOG / (label + ".log")}')
    return result.stdout

def main():
    torch.set_num_threads(1)
    WORK.mkdir(parents=True, exist_ok=True)
    for name in ['train.py', 'model.py', 'sample.py', 'configurator.py', 'LICENSE']:
        shutil.copy2(SRC / name, WORK / name)
    folder = WORK / 'data/toy'
    folder.mkdir(parents=True, exist_ok=True)
    alphabet = '\n abc'
    stoi = {c:i for i,c in enumerate(alphabet)}
    itos = {i:c for c,i in stoi.items()}
    # Two independently generated streams of the same simple periodic grammar.
    for split, repeats in [('train', 1024), ('val', 256)]:
        text = ('ab cab\n' if split == 'train' else 'cab\nab ') * repeats
        np.array([stoi[c] for c in text], dtype=np.uint16).tofile(folder / (split +
        '.bin'))
        with (folder / 'meta.pkl').open('wb') as f:
            pickle.dump(dict(vocab_size=len(alphabet), stoi=stoi, itos=itos), f)
        config = '''out_dir = 'out-toy'
dataset = 'toy'
device = 'cpu'
dtype = 'float32'
compile = False
wandb_log = False
n_layer = 2
n_head = 2
n_embd = 16
block_size = 8
batch_size = 2
gradient_accumulation_steps = 2
dropout = 0.0

```

```

bias = False
learning_rate = 0.003
warmup_iters = 2
lr_decay_iters = 40
min_lr = 0.0003
max_iters = 20
eval_interval = 5
eval_iters = 4
log_interval = 1
always_save_checkpoint = True
'''

(WORK / 'tiny_config.py').write_text(config)
train = run('tiny_train', ['train.py', 'tiny_config.py'])
ckpt = WORK / 'out-toy/ckpt.pt'
# Only load this harness's own checkpoint. Never accept arbitrary pickle files.
state = torch.load(ckpt, map_location='cpu', weights_only=True)
before = {k:v.clone() for k,v in state['model'].items()}
checkpoint_iter_before = state['iter_num']
shutil.copy2(ckpt, LOG / 'tiny_before_resume.pt')
resume = run('tiny_resume', ['train.py', 'tiny_config.py', '--init_from=resume', '--
max_iters=25'])
state2 = torch.load(ckpt, map_location='cpu', weights_only=True)
sample = run('tiny_sample', ['sample.py', '--out_dir=out-toy', '--device=cpu',
                             '--dtype=float32', '--start=ab', '--num_samples=1', '--
max_new_tokens=32',
                             '--temperature=0.8', '--top_k=3', '--seed=17'])
sample2 = run('tiny_sample_repeat', ['sample.py', '--out_dir=out-toy', '--
device=cpu',
                                     '--dtype=float32', '--start=ab', '--num_samples=1', '--
max_new_tokens=32',
                                     '--temperature=0.8', '--top_k=3', '--seed=17'])
assert sample == sample2
assert checkpoint_iter_before == 20 and state2['iter_num'] == 25
assert any(not torch.equal(v, state2['model'][k]) for k,v in before.items())
assert not any('rng' in k.lower() for k in state2)
vals=[float(v) for v in re.findall(r'val loss ([0-9.]+)',train)]
assert all(math.isfinite(v) for v in vals) and vals[-1]<vals[0]
assert state2['optimizer']['state']
assert all(torch.isfinite(v).all() for v in state2['model'].values())
updates=len(re.findall(r'^iter \d+:',train,re.M))
resumed_updates=len(re.findall(r'^iter \d+:',resume,re.M))
assert updates==21 and resumed_updates==6
report = dict(torch=torch.__version__, numpy=np.__version__, python=sys.version,
              threads=1, cpu_only=True, seed_train=1337, seed_sample=17,
              source_hashes={n:sha(SRC/n) for n in
['train.py','model.py','sample.py','configurator.py']},
              copied_hashes={n:sha(WORK/n) for n in
['train.py','model.py','sample.py','configurator.py']},
              checkpoint_before=checkpoint_iter_before,
checkpoint_after=state2['iter_num'],
              checkpoint_keys=sorted(state2), sample_repeat_identical=True,
              first_process_updates=updates,
resumed_process_updates=resumed_updates,
              validation_decreased=True, optimizer_state_present=True,
parameters_finite=True,
              model_changed_after_resume=True, tokens_per_iteration=32,

```

```

        train_evaluations=[l for l in train.splitlines() if l.startswith('step
    ')],
        resume_evaluations=[l for l in resume.splitlines() if
l.startswith('step ')],
        sample=sample, commands_log='tiny_*.log',
        boundary='CPU tiny synthetic pipeline only; no exact RNG resume, GPU,
DDP, or quality claim')
    (LOG/'tiny_pipeline.json').write_text(json.dumps(report, ensure_ascii=False,
indent=2))
    print(json.dumps(report, ensure_ascii=False, indent=2))

if __name__ == '__main__':
    main()

```

examples/verify_mechanics.py

```

"""Original, offline tests for the book. Source snippets are AST-extracted from MIT
nanoGPT.
Copyright (c) 2022 Andrej Karpathy for extracted upstream code; see ../research/
UPSTREAM_LICENSE.txt.
"""

from pathlib import Path
import ast, contextlib, copy, io, json, math, sys
import numpy as np
import torch
import torch.nn.functional as F
ROOT=Path(__file__).resolve().parents[1]
SRC=ROOT/'research/upstream'
sys.path.insert(0, str(SRC))
from model import GPT, GPTConfig

def source_function(name):
    module=ast.parse((SRC/'train.py').read_text())
    node=next(n for n in module.body if isinstance(n, ast.FunctionDef) and n.name==name)
    node.decorator_list=[]
    return compile(ast.Module(body=[node], type_ignores=[]), '<extracted
train.py>', 'exec')

def main():
    torch.set_num_threads(1); torch.manual_seed(17)
    out={}

    cfg=GPTConfig(vocab_size=7, block_size=6, n_layer=2, n_head=2, n_embd=8, dropout=0.0, bias=True)
    m=GPT(cfg).double().eval(); B,T,C,V,L=4,6,8,7,2
    x=torch.randint(V, (B,T)); y=torch.randint(V, (B,T))
    logits, loss=m(x,y); assert logits.shape==(B,T,V)
    expected=V*C+T*C+L*(12*C*C+13*C)+2*C
    assert m.get_num_params(False)==expected
    assert m.get_num_params()==expected-T*C
    assert m.transformer.wte.weight is m.lm_head.weight
    opt=m.configure_optimizers(.1, .001, (.9, .95), 'cpu')
    ids=[id(p) for g in opt.param_groups for p in g['params']]

```

```

    assert len(ids)==len(set(ids))
    assert sum(p.numel() for g in opt.param_groups for p in g['params'])==expected

out['parameters']=dict(total=expected,non_position=m.get_num_params(),unique_tensors=len(
ids))
    changed=x.clone();changed[:,3:]=(changed[:,3:]+1)%V
    original=m(x,y)[0];other=m(changed,y)[0]
    causal=(original[:,3:]-other[:,3:]).abs().max().item();assert causal<1e-12
    out['causal_prefix_max_error']=causal
    # Explicitly install the legacy mask before forcing that branch. Merely flash=False
is insufficient.
    fallback=copy.deepcopy(m)
    for b in fallback.transformer.h:
        b.attn.flash=False
        b.attn.register_buffer('bias',torch.tril(torch.ones(T,T)).view(1,1,T,T))
    manual=fallback(x,y)[0]
    error=(original-manual).abs().max().item(); assert error<1e-12
    out['sdpa_manual_max_error']=error
    full=copy.deepcopy(m); micro=copy.deepcopy(m)
    full(x,y)[1].backward()
    for xx,yy in zip(x.chunk(2),y.chunk(2)):
        (micro(xx,yy)[1]/2).backward()
    err=max((a.grad-b.grad).abs().max().item() for a,b in
zip(full.parameters(),micro.parameters()))
    assert err<1e-12;out['accumulation_max_grad_error']=err
    # Finite differences on a shared embedding parameter test both input and output
uses.
    p=full.transformer.wte.weight;analytic=p.grad[2,3].item();epsilon=1e-5
    with torch.no_grad():
        old=p[2,3].item();p[2,3]=old+epsilon;plus=full(x,y)[1].item()
        p[2,3]=old-epsilon;minus=full(x,y)[1].item();p[2,3]=old
    numeric=(plus-minus)/(2*epsilon)
    assert abs(analytic-numeric)<1e-7

out['shared_gradient']=dict(autograd=analytic,finite_difference=numeric,error=abs(analyt
ic-numeric))
    before=m.get_num_params(False);m.crop_block_size(4)
    assert before-m.get_num_params(False)==16
    try:m(x[:,4],y[:,4])
    except RuntimeError:out['noncontiguous_targets_view_failure']=True
    else:raise AssertionError('expected noncontiguous target view failure')
    assert m(x[:,4],y[:,4].contiguous())[0].shape==(B,4,V)
    try:m(x,y)
    except AssertionError:out['oversize_context_rejected']=True
    else:raise AssertionError('expected context failure')
    # Original get_lr without importing and executing train.py.
    ns=dict(math=math,learning_rate=.001,min_lr=.0001,warmup_iters=2,lr_decay_iters=10)
    exec(source_function('get_lr'),ns)
    out['learning_rates']={str(i):ns['get_lr'](i) for i in [0,1,2,6,10,11]}
    assert ns['get_lr'](2)==.001 and ns['get_lr'](10)==.0001
    ns['lr_decay_iters']=2
    try:ns['get_lr'](2)
    except ZeroDivisionError:out['degenerate_schedule_rejected']=True
    # Token alignment at the final legal offset: randint(high=N-T) is exclusive.
    data=np.arange(10,dtype=np.uint16);t=4
    for i in range(len(data)-t):

```

```

xx=data[i:i+t].astype(np.int64); yy=data[i+1:i+1+t].astype(np.int64)
assert len(xx)==len(yy)==t and np.array_equal(xx+1,yy)
out['last_legal_window']=dict(start=5,x=data[5:9].tolist(),y=data[6:10].tolist())
out['uint16_max']=int(np.iinfo(np.uint16).max)
assert np.array([65536],dtype=np.int64).astype(np.uint16)[0]==0
# Configurator executes in the caller globals; only trusted source is supplied here.
code=(SRC/'configurator.py').read_text(); saved=sys.argv
results={}
try:
    for arg in ['--batch_size=3','--batch_size=3.0','--unknown=3','--
compile=false','--start=a=b']:
        ns={'batch_size':2,'compile':False,'start':'a'}
        sys.argv=['test',arg]
        try:
            with contextlib.redirect_stdout(io.StringIO()):exec(code,ns)
            results[arg]='ok'
        except Exception as e:results[arg]=type(e).__name__
    finally:sys.argv=saved
    out['config_cases']=results
    # Top-k threshold includes all ties, and the original interface does not validate
    zero temperature.
    z=torch.tensor([[1.,1.,0.]])
    cutoff=torch.topk(z,1).values[:,-1:]; kept=z>=cutoff
    assert kept.sum()==2;out['top_k_1_tie_survivors']=2

out['versions']={'torch':torch.__version__,'numpy':np.__version__,'seed':17,'threads':1}
(ROOT/'qa/author/mechanics.json').write_text(json.dumps(out,indent=2))
print(json.dumps(out,indent=2))

if __name__=='__main__':main()

```