大语言模型与生成式基础模型

基础、系统、对齐与应用

尹诚

2026 年 6 月 14 日

前言

本书是《Large Language and Generative Foundation Models: Foundations, Systems, Alignment, and Applications》的中文可读版。中文稿以原英文稿为蓝本，在保持技术含义、章节顺序、术语边界和引用依据一致的前提下，使用更适合中文读者的表达方式组织句子。它不是“核心论点摘要”，也不是只保留结论的缩写版；它的目标是让中文读者能够沿着同一条论证路径，理解每一章为什么这样安排、每个概念解决什么问题、每种工程选择带来什么代价。本书坚持一个基本立场：大模型不是孤立算法，而是一套由数据、目标函数、结构、系统、推理策略、评测和治理共同定义的工程系统。随着多模态大模型、统一理解-生成模型、扩散语言模型、视频生成模型、Omni 任意到任意接口和 vision-language-action 模型的发展，这个立场还需要进一步扩展：文本自回归只是生成式基础模型的一种重要形式，而不是整个领域的边界。理解现代基础模型的关键不是记住模型家族名称，而是能说清楚一个选择优化了什么、牺牲了什么、如何测量，以及测量本身可能在哪里失真。本书文字、结构、例子和论证为原创教材性写作。书中研究判断尽量连接到公开论文、技术报告或课程材料。第三方幻灯片、视频转写、数据样本、代码清单和图像不作为正文内容复制。

伦理与来源说明

本书只复述公开研究中的概念、方法和结论，不复制第三方课程材料、私有数据、模型输出、代码或图像。涉及模型报告、数据集、软件项目、法规和政策文件时，均把它们作为参考来源，而不是作为可直接搬运的内容。读者在复现实验时应特别关注数据来源、许可证、个人信息、评测污染和安全边界。一个可以训练的系统不等于一个可以发布的系统；一个可以发布的系统也不等于一个可以在高风险场景中不受约束使用的系统。

目录

第一部分

基础

第一章　什么使语言模型变大

1.1　研究对象

大语言模型的基本对象仍然是条件概率分布。给定一段 token 序列，自回归模型把联合概率分解为

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

训练通常最小化下一个 token 的平均负对数似然；若损失用自然对数计量，perplexity 就是该损失的指数。这个目标看起来简单，却把语言、代码、数学、工具调用、对话格式和多模态描述都压缩进同一个预测界面。Transformer 让这种目标在大规模上可行，因为 self-attention 能并行建模远距离 token 依赖 [233]。GPT 式因果模型随后表明，只要语料足够多样，单一从左到右目标也能迁移到广泛任务 [205, 14]。BERT 式 masked model 仍然适合表示学习和检索，密集检索又把表示质量连接到开放问答和 RAG 系统 [33, 134, 147]。

“大”不只表示参数多。参数规模、训练 token 数、数据多样性、上下文长度、推理预算、工具能力、服务吞吐和安全约束都会改变模型行为。一个参数更小但上下文更长、检索更强、推理预算更大的系统，可能在真实任务中比一个参数更多但系统设计较弱的模型更有用。因此，“大模型”的规模不是一个单轴数字。总参数量说明容量，激活参数量说明每个 token 实际花费的计算，训练 token 数说明模型见过多少统计证据，数据来源说明这些证据是否合法、可靠并且可复现。上下文长度说明模型一次前向可以条件化多少信息，但 KV cache、注意力代价和位置泛化决定了这个长度能否经济地服务。推理预算说明模型在预训练之后还能通过采样、检索、工具调用、验证器和搜索花多少额外计算。任何只报告参数量的比较，都无法解释现代系统的真实能力。

本书把模型看成一个“条件计算系统”。这种说法比“一个神经网络”更准确，因为同一个 checkpoint 在不同 tokenizer、chat template、解码参数、检索器、工具权限和安全过滤器下会表现出不同系统行为。模型发布也不是一个权重文件，而是一串契约：数据来源约束 tokenizer，tokenizer 约束序列长度和 special token，结构和损失约束可训练形态，优化和分布式系统约束成本与稳定性，后训练约束交互行为，推理服务约束延迟、上下文、工具和安全策略，评测与治理再把线上事故反馈回数据和训练。

基础模型学习的是文本延续分布；助手系统还要定义谁在说话、哪些指令有更高优先级、哪些内容必须拒绝、哪些外部证据可以信任，以及哪些工具可以被调用。

1.2 预测为什么成为通用接口

下一个 token 预测的核心优势是密度。每个文档中的几乎每个 token 都能成为训练目标，因此网页、书籍、代码、论文、问答和对话可以在没有人工标签的情况下转化为监督信号 [14]。这个目标并不直接奖励真实性、有用性、校准或无害性 [189]，但语言本身承载事实、过程、论证、代码、指令和交互格式，所以分布式延续训练能先学到大量潜在任务。

这个接口也很适合实现。模型读入 token id，映射为 embedding，经由多层 attention 与前馈网络混合信息，再投影到词表 logits。训练时，logits 与右移后的标签做交叉熵；生成时，从概率分布中采样或选择下一个 token。后续章节会从张量形状、mask、残差、归一化、位置编码和 logits 解释这一流程。第一章提前强调这一点，是为了防止把“会聊天”误解为模型结构中有一个独立聊天模块。

预测接口的代价是语义边界必须由数据格式和模板补上。系统消息、用户消息、助手回答、工具调用、工具返回、引用证据和拒答策略本质上都只是 token 序列。若模板写错、label mask 写错或训练数据把角色混在一起，模型就会学习错误接口。很多“模型不听指令”的问题，并不是基础模型容量不足，而是数据、模板和损失边界没有被当成系统规格管理。

1.3 规模改变了研究问题

现代 scaling law 把自然语言处理从任务特定数据集和模型结构，推向了参数、数据和算力的联合预算问题。Kaplan 等工作展示了损失随模型规模、数据规模和计算量近似按幂律变化 [132]；Chinchilla 风格计算最优训练进一步指出，许多早期大模型相对于其算力预算是 undertrained 的，参数变大并不自动等于训练充分 [50]。因此，一个严肃训练计划必须同时预算参数量、训练 token、序列长度、硬件吞吐、数据质量和评测协议。

涌现能力是 scaling 讨论中最容易被误读的部分。若某个能力在小模型上近似不存在、在大模型上突然出现，曲线看起来像阈值 [241]。但阈值并不是自然常数；它会随数据质量、模型家族、prompt 格式、finetuning、推理方法和指标改变。Exact match 或 accuracy 可能隐藏 log probability 的连续改善，直到答案格式刚好跨过评分器阈值。可信的涌现主张应报告完整模型序列、训练 compute 或参数规模、数据和 prompt 协议、随机基线、指标选择、不确定性，以及 loss 或校准 likelihood 是否早于 headline score 发生平滑变化。

开放模型报告让这种系统视角更清楚。LLaMA 说明较小模型如果数据和 token 预算选择得好，也能有竞争力 [230]；Llama 2 把预训练、SFT、偏好数据和安全评测放进同一报

告 [231]；Llama 3 进一步扩展到长上下文、多语言、代码、安全模型和大规模后训练 [43]。DeepSeek-V3 和 Kimi K2 等系统又显示，MoE 路由、特殊 attention 机制、代码/agent 后训练和服务成本正在共同定义 frontier open-weight 系统 [31, 137]。这些报告有价值，但仍应被当作技术报告阅读，而不是替代独立复现。

1.4 从基础模型到助手

基础模型通过预训练学习广泛的语言和知识模式，但它并不天然知道应该如何作为用户助手响应。后训练阶段通过监督指令微调、偏好学习、安全数据和工具协议，把基础模型转化为面向用户的系统。InstructGPT 和后续 RLHF 工作使这一点变得清晰：可用性不是预训练困惑度的直接结果，而是接口、偏好和安全目标共同塑造的结果 [189, 9]。近年来，OpenAI o1/o3、DeepSeek-R1、Qwen3 和 Kimi K2 等系统又把“推理时间”变成产品能力的一部分。推理不再只是模型内部能力，也包括采样、搜索、验证、工具调用和预算分配 [30, 250, 137]。这也解释了为什么预训练损失、聊天主观评分和真实部署质量之间不存在简单等价关系。预训练损失下降，可能意味着模型更好地预测了训练分布；但如果训练数据包含评测污染、错误网页、过期知识或有害模式，下游行为可能并不更可信。聊天偏好评分上升，可能说明模型语气更符合标注者期待；但它也可能学会冗长、奉承、空泛免责声明或过度拒答。真实部署则还要面对延迟、成本、权限、日志、监控和事故响应。

偏好优化也不是“人类价值函数”的同义词。RLHF 依赖特定标注协议、候选回答分布和奖励模型；DPO 等方法省去了显式在线强化学习循环，但仍然依赖 chosen/rejected 数据是否覆盖真实使用场景 [206]。标注者偏好的语气、长度、礼貌、拒答边界和安全策略都会进入模型行为。若报告只写“aligned model”，而不说明偏好数据、rubric、拒答类别和评测切片，读者无法判断它到底对齐了什么。

参数高效适配改变了谁能做后训练。LoRA 冻结 base model，只在选定线性层学习低秩更新 [53]；QLoRA 把低秩适配与量化 base weight 结合，使小硬件也能做指令微调 [32]。它们不是无成本魔法：rank、target modules、量化误差、optimizer state、合并策略和评测范围都会改变失败模式。若适配数据很窄，模型可能在目标格式上更听话，却在通用能力、安全边界或多语言表现上退化。

助手还需要检索和工具。RAG 把外部证据放进上下文，工具调用让模型使用搜索、计算器、数据库或代码执行环境 [147, 256]。这些系统把部分知识从参数记忆转移到运行时 grounding，但也引入 retriever recall、reranker precision、上下文排序、引用忠实性、权限边界和 prompt injection 风险。比较助手系统时，checkpoint 只是其中一个组件；检索索引、工具 schema、日志、监控和安全策略同样定义用户可见行为。

1.5 推理使测试时计算成为模型问题

推理不是单一机制。Chain-of-thought prompting 诱导中间自然语言步骤，self-consistency 采样多条路径，tree search 在候选 partial solution 上分支，verifier-guided 方法对推理轨迹打分或过滤 [242, 255]。这些轨迹可能提升任务表现，但不一定是模型内部计算的忠实解释，因此应被当作生成 artifact 评测，而不是透明思维记录 [163]。

推理型强化学习又增加了一层控制问题：奖励最终答案、过程步骤、单元测试、代码执行结果，还是工具轨迹？GRPO/RLVR 一类方法强调可验证结果，DeepSeek-R1 等系统让这种路线更受关注 [30, 243]。但分数提升可能来自更好的策略，也可能来自更多采样、更长轨迹、更强 verifier、搜索预算或污染。后续讨论 PPO、DPO、GRPO、过程奖励模型和 Monte Carlo tree search 时，本书都会追问四个问题：奖励什么，约束什么分布，信任哪个验证器，测试时花了多少额外计算 [22, 220, 254]。

1.6 本书路线

第一部分讨论建模对象、token 和数据。第二部分讨论 Transformer、GPT、LLaMA 类结构、优化、分布式训练和推理服务。第三部分讨论 SFT、LoRA、QLoRA、领域和语言适配。第四部分讨论 RAG、工具、agent、偏好学习、推理型训练、多模态与生成式基础模型、评测、安全和治理。最后一章把 CS336 式从零实现纪律与 2025–2026 年前沿研究连接起来 [222]。因此，本书虽然以大语言模型为主线，但并不把研究边界限死在纯文本模型。文本模型提供了最清晰的训练目标、系统接口和后训练范式；多模态理解模型、统一理解-生成模型、扩散/流匹配生成模型和 Omni 模型则说明，同一套数据、系统、评测和治理问题正在扩展到图像、音频、视频、语音和交互式生成。后文讨论这些模型时，会尽量避免把它们写成模型名称列表，而是把它们放回目标函数、表示、训练、服务和评测的共同框架中。

1.7 发布级阅读协议

高质量模型书稿应教读者审问模型报告。一个模型发布可以读成

$$\mathcal{R} = (D, \tau, A, O, S, P, I, E, G),$$

其中 D 是数据混合，τ 是 tokenizer 与模板，A 是架构，O 是优化配方，S 是训练系统，P 是后训练过程，I 是推理系统，E 是评测证据，G 是治理约束。任何能力主张如果缺少其中一项，都是欠规格的。例如，长上下文分数如果没有 I，就无法说明 KV cache 成本和调

度可行性；推理分数如果没有 P 和 I，就看不出增益来自强化学习、更多样本、更长轨迹还是更强验证器。

这个协议也解释了本书为什么优先使用原创 synthesis diagram，而不是复制论文中的架构图。教材图的目的不是复刻 model card 截图，而是暴露可比较的不变量：GPT decoder、MoE 系统、RAG 管线、多模态连接器和推理时控制器如何在同一组分析维度下被理解。引用说明来源，图形则服务于新的解释框架。

1.8 深入展开：规模、系统与评测

本章的重点不是给“大模型”下一个参数量阈值，而是拆解“规模”这个词背后的多个轴。参数量只是容量的一种表示；训练 token 数决定模型见过多少统计证据；数据来源决定这些证据是否可靠、合法、可复现；上下文长度决定推理时能条件化多少信息；推理预算决定模型能否在预训练之后通过采样、检索、验证、工具和搜索继续花计算。一个只报告“多少 B 参数”的模型比较，通常遮蔽了真正的系统差异。这也是为什么本书从一开始就把大语言模型写成系统，而不是单个神经网络。模型权重、tokenizer、chat template、检索器、工具运行时、安全过滤器和服务调度器共同定义用户实际接触到的行为。两个团队使用同一 base checkpoint，如果一个团队有更好的检索、上下文构造、引用约束和安全回归测试，另一个团队只有普通聊天模板，它们在真实任务中的可靠性会完全不同。

本章还强调，预训练的“基础模型”并不会自动成为助手。基础模型学到的是延续文本的能力，而助手需要知道谁在说话、哪些指令优先级更高、什么时候需要拒答、什么时候应引用证据、什么时候应调用工具。SFT、RLHF、DPO、RLAIF 和推理型强化学习都是把基础模型转化为可交互系统的后训练机制，但它们优化的都是代理目标，不能被误解为“已经对齐人类价值”。

最后，评测从第一章就被放进主线。模型越大，越容易在公开 benchmark 上出现污染、调参和过拟合；上下文越长，越容易让人误以为模型真正利用了所有证据；推理采样越多，越容易用 pass@k 掩盖单次可靠性不足。因此，本书后续每章都反复追问同一个问题：某个指标到底测量了什么，遗漏了什么，是否足以支撑发布决策。

1.9 章节细节

1.9.1 研究对象

语言模型研究的对象首先是一个条件分布，而不是一个聊天界面。自回归分解把长文本生成化为一串局部预测，但这些局部预测在规模足够大、数据足够多、接口足够通用时，

会表现为翻译、问答、代码、推理和工具调用等能力。理解这一点可以避免把每个能力都误读为单独训练出来的模块。

1.9.2 预测为什么成为通用接口

下一个 token 预测之所以重要，是因为它能容纳几乎所有可被序列化的任务。任务说明、输入材料、推理草稿、函数调用、程序输出和最终答案都能被写成上下文的一部分。它的代价是边界必须由数据格式和模板显式定义，否则模型会混淆角色、证据、指令和输出。

1.9.3 规模改变了研究问题

规模让研究从“模型是否能做某类任务”转向“系统如何稳定、经济、可解释地做任务”。参数、token、上下文、数据质量、后训练和服务预算共同决定能力。只讨论参数量会忽略训练 token 不足、推理成本过高、上下文利用率低和评测污染等关键问题。

Compute-optimal planning 还要求训练计划明确 N、D、C 和序列长度的关系。一个 70B dense 模型如果训练 token 不足，可能比参数更少但 token 更充分的模型差；一个 MoE 模型总参数很大，但每个 token 激活参数有限，服务瓶颈可能来自路由、通信和缓存，而不是名义参数量。

1.9.4 从基础模型到助手

基础模型只是学会了延续文本，并不天然知道怎样服务用户。助手需要学习对话角色、拒答边界、格式约束、工具协议和安全策略。SFT、偏好学习和安全训练的意义正是在这个接口层面重塑模型行为。

1.9.5 推理使推理阶段成为训练的一部分

推理型模型把测试时计算纳入能力定义。模型可以生成多个候选、调用工具、使用验证器、搜索解空间或按难度分配 token。这样一来，比较模型时不能只问 checkpoint 有多强，还要问系统允许它在推理时花多少计算、用哪些外部资源、怎样选择答案。

1.9.6 全书路线

本书的路线从概率建模和数据开始，再进入 Transformer、GPT、LLaMA 类结构、预训练、分布式训练和推理服务，随后讨论指令微调、PEFT、领域适配、RAG、偏好学习、推理、多模态以及治理。这个顺序反映了一个判断：现代大模型不是单点算法，而是从数据到部署的一条完整链路。

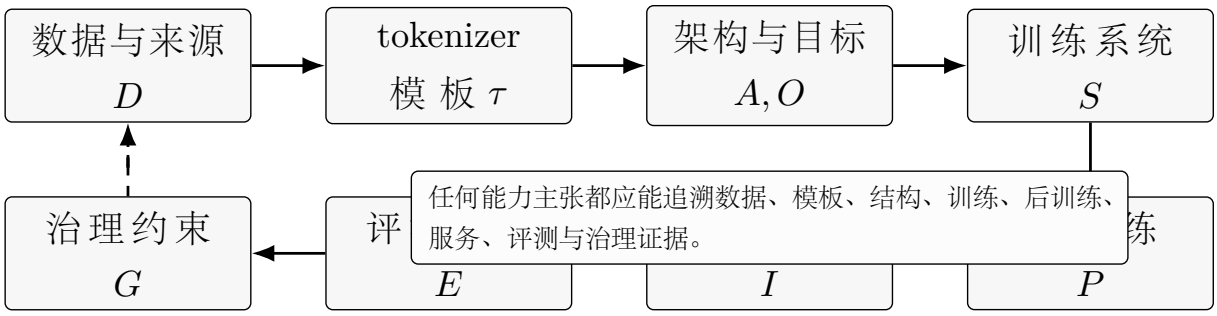

图 1.1: 大模型发布是系统生命周期，而不是单个权重文件。该图是对 GPT 式预训练、InstructGPT/RLHF、RAG 与发布治理实践的原创综合 [14, 189, 147]。

1.9.7 发布证据而不是模型名

读模型报告时，不要从模型名开始，而要从证据开始。数据、tokenizer、架构、优化、训练系统、后训练、推理系统、评测和治理缺一项，能力主张就缺少可解释边界。发布级证据应能回答：训练了什么、怎样训练、如何服务、用什么测量、哪些风险被排除、哪些风险仍然存在。

Hugging Face Hub 的 model card 文档把这件事落成了更具体的发布对象：模型仓库的 `README.md` 会被渲染为 model card，顶部 YAML metadata 支持发现、筛选和复用；字段可以记录库名、base lineage、版本迁移、训练数据、任务标签、许可证、结构化评测结果与来源、论文链接、受众限制和碳排放等信息 [78, 96]。因此，发布协议不应只停留在“报告写得清楚”：仓库元数据本身就是可机器读取的证据层。一个模型如果没有明确基座关系、训练数据链接、许可证、任务标签、评测来源和版本迁移路径，即使权重可下载，也不能被当成完整发布物。

Gated model 也应被解释为权限和分发控制，而不是安全结论。Hub 的 gated-model 机制可以记录 pending、accepted、rejected 访问请求，下载 access report，并用额外申请字段和提示文本收集用途、机构、国家或许可确认 [74]。这些字段能说明谁被允许获取权重以及如何撤销访问，但不能证明模型已经安全。发布报告还应把 gate policy 同 license、允许用途、model card 限制、滥用监控、撤权流程和事故响应连接起来；否则“需要申请访问”只是一道门，不是治理证据。图 1.1 把本章的系统视角压缩成一个发布级生命周期：能力来自链路，而不是某个孤立 checkpoint。

1.10 关键术语、实现要点与练习

关键术语。条件语言模型指在给定前文时预测下一个 token 的概率模型；base model 指经过大规模预训练但尚未面向助手接口后训练的模型；assistant model 指经过指令、偏

好、安全、工具或领域信号塑造的用户系统；scaling law 描述损失随参数、数据和算力变化的经验规律；active parameters 表示稀疏模型每个 token 实际激活的参数；后训练包括 SFT、偏好优化、安全调优、推理强化和领域适配；推理预算指检索、采样、验证、搜索和工具调用在部署时消耗的额外计算；系统边界指模型权重之外的 tokenizer、模板、检索、工具、过滤器和服务运行时；release evidence 指支撑模型能力和风险主张的数据、训练、服务、评测与治理证据；model card 是模型仓库的可读说明与 YAML 元数据；repo card metadata 是可机器读取的发布字段；gated model 是带访问请求、审批和撤销记录的分发控制。

实现要点。读者应能写出一个最小自回归损失，并解释为什么 label 需要右移；能区分参数量、激活参数量、训练 token 数、上下文长度和推理预算；能为一个模型报告列出数据、tokenizer、架构、优化、训练系统、后训练、推理系统、评测和治理字段；能审查 model card YAML 中的 `library_name`、`base_model`、`datasets`、`pipeline_tag`、`license`、`model-index` 和 `new_version`；能说明 gated access 记录、额外申请字段和撤权流程为什么只是分发证据而不是安全证明；能说明一个 benchmark 分数可能被数据污染、prompt、答案抽取器、采样预算或验证器改变。

练习。

1. 给定一个四 token 序列，写出自回归分解，并说明每一步条件是什么。

2. 比较两个系统：一个模型更大但无检索，另一个模型较小但有高质量 RAG。列出至少五个不能只看参数量的原因。

3. 为一个助手系统画出边界图：模型、tokenizer、模板、检索器、工具、安全过滤和日志分别在哪里。

4. 找一篇模型技术报告，标注其中哪些数字是能力数字，哪些是系统成本数字，哪些是治理证据。

5. 给出一个“涌现能力”主张，列出完整模型序列、prompt、metric、随机基线、置信区间和 smoother likelihood 证据应该如何报告。

6. 把一个模型发布写成 $\mathcal{R} = (D, \tau, A, O, S, P, I, E, G)$，并指出如果缺少 I 或 E，哪些结论不能成立。

7. 为一个模型仓库写最小 model card metadata，要求包含 `library_name`、`base_model`、`datasets`、`pipeline_tag`、`license`、`model-index` 评测来源和 `new_version`，并说明每个字段对应 $\mathcal{R}$ 的哪一项。

规模轴	它实际改变什么	常见误读
总参数量	模型容量、存储、加载成本	参数越多一定越强
激活参数量	每个 token 实际计算量	MoE 总参数可直接和 dense 比
训练 token 数	统计证据和覆盖面	token 多就一定数据好
上下文长度	推理时可条件化的信息量	能放入就等于能利用
推理预算	采样、搜索、检索、验证、工具成本	多想一定更正确
服务吞吐	用户可用性和成本	离线 benchmark 足够代表部署

表 1.1: 规模轴、实际改变对象与常见误读检查表。

8. 设计一个 gated model 发布流程，列出申请字段、审批状态、访问报告、撤权条件和事故响应，并说明哪些结论仍不能由访问门禁证明。

9. 解释为什么 chain-of-thought 正确不等于解释忠实，并设计一个用 verifier 和人工复核共同检查推理轨迹的方案。

1.11 结构化检查表

1.11.1 规模轴检查表

表 1.1 把规模轴和常见误读压成可引用的审查入口。

第二章　Token、语料与训练信号

2.1　数据就是规格

数据不是模型之外的附件。它定义了模型可见的语言、知识、风格、许可证边界和风险。训练语料中的重复内容会增加记忆风险；低质量网页会教会模型无意义模板；代码数据会改变推理和格式能力；多语言数据会影响 token 效率和跨语言泛化。一个严肃的数据报告至少应记录来源、时间、许可证、语言分布、过滤规则、去重方法、敏感信息处理、合成数据生成器、评测集重叠检查和数据混合权重。没有这些信息，模型行为很难解释，评测结论也很难复现。

数据混合权重本身就是训练目标的一部分。把更多 token 分配给代码，会提升代码补全、格式遵循和某些结构化推理能力，但也会改变自然语言风格和安全风险。把更多 token 分配给数学和竞赛题，可能提升推理 benchmark，却也可能让模型在普通对话中更爱展开冗长步骤。多语言数据不仅影响语言覆盖，还影响 token fertility：同样的中文、阿拉伯文或印地语内容，如果被 tokenizer 切得更碎，就会占用更多训练和推理计算。

过滤不是单纯“清洗脏数据”。它会改变模型世界观和能力边界。过强过滤可能删掉少数语言、非正式文本、敏感但合法的讨论和真实错误案例；过弱过滤则会保留垃圾模板、个人信息、仇恨内容、恶意代码和重复材料。高水平数据管线应把过滤规则视为可检查决策，而不是一次性脚本。

可出版的数据说明应把 manifest 当作主产物，而不是把语料管线藏在日志里。每条文档至少应有稳定 id、来源、采集时间、许可证类别、语言、内容类型、过滤状态、去重状态、是否命中评测重叠、以及进入哪一个数据 split。原始语料、过滤后语料、tokenized 语料和 packed token stream 应有不同 manifest；否则训练后很难追查某个行为来自来源本身、过滤规则、采样权重，还是打包时的边界错误。

混合权重还需要用训练 token 数解释。若组件 i 有可用 token 数 D_i，训练总预算为 S，采样概率为 p_i，则预期曝光量为 $E_i = Sp_i$，重复率为 $r_i = E_i/D_i$。小而高质量的数据可以被上采样，但 r_i 过高会提高记忆和过拟合风险。数据报告只给“包含代码、数学和网页”是不够的；它应说明各组件的 token 规模、采样概率、重复率、质量分层和污染风险。

同一段原始文本不能随意复用到所有数据角色中。预训练语料会改变模型参数；检索

数据 artifact	主要契约	常见失败
Tokenizer 语料	定义可复用单位和 token 效率	罕见文字、代码或领域术语被切得过碎
预训练语料	提供密集自监督目标	重复、低质、私有数据或 benchmark 泄漏
指令数据	展示输入到输出的行为	prompt 被计入 loss，格式支配能力
偏好数据	在同一 prompt 下比较候选回答	标注 rubric 不清，奖励冗长或迎合
检索语料	推理时提供可引用证据	chunk 过期、无许可、切分差或实际未检索到
评测集	估计泛化、安全或鲁棒性	污染、prompt 过拟合、rubric 弱或分布漂移

表 2.1: 数据 artifact、主要契约与常见失败模式检查表。

语料在推理时提供证据；偏好数据改变回答分布；评测集用于估计泛化。若这些角色没有来源边界，模型可能在训练中见过评测答案，也可能在 RAG 中引用无授权材料，或者把偏好标注中的冗长风格误当作事实能力。数据 manifest 因此应按 artifact 类型建表，而不是只给一个总目录。

把这个契约落到代码层，预处理脚本也应可审计。以网页语料为例，报告不能只写“使用 OpenWebText”；还应记录数据加载器、文档数、训练/验证切分比例、随机种子、tokenizer 名称、是否使用 ordinary encoding、是否在每个文档后追加 end-of-text token、写入二进制流时的 dtype、分片数和写入顺序。一个脚本若用 `train_test_split(test_size=0.0005, seed=2357)` 生成验证集，用 GPT-2 BPE 编码并在文档末尾追加 `eot_token`，再按 1024 个连续 shard 写入 `uint16` memmap，这些都不是实现细节而是数据证据。它们决定验证集是否可复现、文档边界是否进入训练信号、token id 是否能被 dtype 容纳，以及后续 dataloader 能否按同一顺序读到相同 token。

表 2.1 将数据角色和失败模式整理为本章的可复核契约。

图 2.1 展示了数据从来源到训练 token stream 的可审计路径：重点不是复刻某个私有数据集流程，而是让 manifest、过滤、去重、tokenizer、packing 和污染检查之间的责任边界可检查。

过滤和去重也应保存成可查询数据。一个实际管线可以在文档表里保留语言通过、质量通过、去重保留、评测重叠、许可证类别和 PII 风险等字段。这样后续发现某个模型行

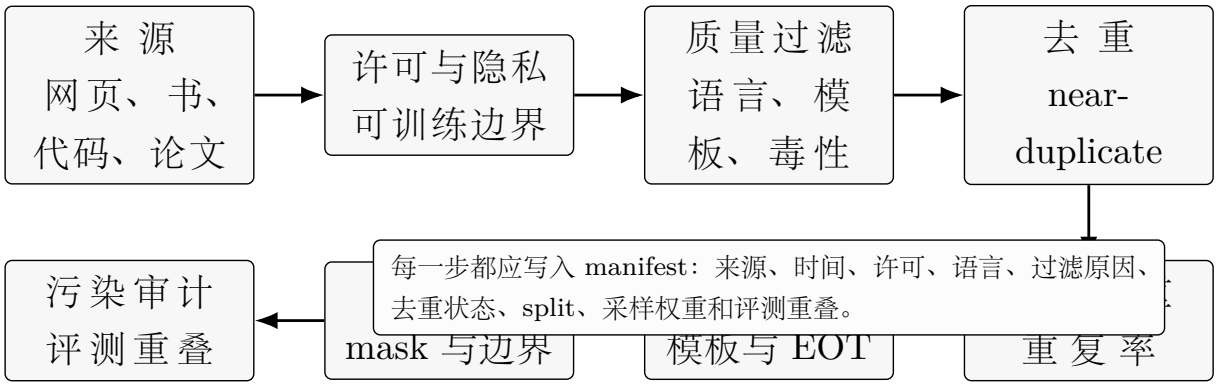

图 2.1: 可审计数据管线把训练信号写成证据链。该图是对分词、语料去重和评测污染文献的原创综合 [213, 140, 145, 218, 247]。

为或版权问题时，团队能解释文档为什么进入或离开训练流，而不是只能说“清洗脚本当时这么做”。近重复去重尤其要写清范围：训练集内部去重降低过度加权，训练-评测去重保护 benchmark，有些跨许可证去重则是为了防止禁止来源通过镜像重新进入语料 [145]。

Dataset card 和 data manifest 不是同一个东西。Manifest 面向训练复现，保存每条记录的 lineage、过滤、去重、split 和 tokenization 状态；dataset card 面向使用者和发布方，解释数据内容、创建过程、许可证、语言、规模、任务类别、偏差和适用边界 [67]。前者回答“这条样本为什么进入训练流”，后者回答“这个数据集能否被理解和负责任地使用”。若只有 dataset card 而没有 manifest，训练事故难以追溯；若只有内部 manifest 而没有对外说明，发布者又把使用风险留给下游猜测。

发布边界也应写进数据契约。训练团队内部可以保留原始抓取、过滤中间态和审计日志，但对外发布时往往只能公开派生统计、脱敏样本、许可证说明和可复核的处理规则。报告应区分 raw、derived、redacted、trainable 和 publishable 几类状态，并说明哪些来源允许训练、哪些允许再分发、哪些要求撤回或删除响应。否则“我们有数据”与“我们可以发布模型和数据证据”会被混成同一件事。

2.1.1 数据集对象与流式训练验收

把数据发布到 Hub、写入本地 Arrow 表，或以压缩 JSONL/Parquet 形式流式读取，本质上是三种不同的数据对象合同。Hugging Face 的 dataset card 文档要求数据集说明帮助使用者理解内容、创建方式、偏差和使用背景，并鼓励记录 license、language、task categories 与 size categories 等可发现元数据 [67]。Datasets 主类文档则把 `DatasetInfo`、features、version、split、hash 和 Arrow-backed `Dataset` 写成库级元数据对象 [72]。这提醒读者：对外 dataset card、内部 manifest、库对象元数据和训练 run card 不是同义词，不能互相替代。

流式加载尤其容易被低估。Datasets streaming 文档说明，`streaming=True` 可以从 Hub 或本地文件边迭代边读取，适合超大数据、磁盘不足、只探索少量样本，或只读取 Parquet 的部分列和过滤条件；但它创建的是 `IterableDataset`，不适合需要随机访问样本的任务 [73]。训练报告若使用流式数据，应写清数据文件模式、选取列、过滤条件、shuffle buffer、shard 数、worker 分配、epoch seed 和是否调用 `set_epoch`。否则同一个"数据集名称"在复现时可能变成不同的样本顺序、不同的过滤结果和不同的训练 token 暴露量。

验收对象	必须记录	常见发布风险
Dataset card	license、language、任务、规模、创建过程、偏差与限制	对外用户只看到名称，看不到适用边界
DatasetInfo	version、features、split、hash、样本数、生成时间	库对象更新后，报告仍引用旧元数据
Hub dataset repo	repo id、revision、README YAML、文件名、Viewer 状态、访问权限	split/config 推断与训练代码不一致
Arrow Dataset	schema、索引、cache 文件、fingerprint、列类型	本地缓存与远端版本不一致
IterableDataset	文件模式、shards、worker 分配、buffer、epoch seed	流式顺序不可复现，分布式 rank 重复读样本
Parquet 过滤	columns、filters、row-group 元数据、过滤前后规模	只训练了筛选子集，却按全量数据解释结果
Run card	tokenizer、special token、split seed、过滤版本、污染检查	训练证据无法连接到数据发布说明

Hub dataset 仓库也应视为发布验收对象。Hugging Face 上传与分享文档把 dataset repository 写成保存数据文件、版本、提交历史、差异、元数据、dataset card 和 Dataset Viewer 的发布单元；数据可以公开或私有，`push_to_hub`、CLI 上传和 README card 都会进入下游复现合同 [98, 99]。Hub 会按文件名、目录名或 README YAML 的 `configs/data_files` 推断 train、validation、test、subset 和 config；split 名称还需要用下划线、连字符、空格、点或数字等 delimiter 与其他词隔开，`test-data.csv` 与 `testdata.csv` 的含义不同 [70, 64]。Dataset Viewer 不是装饰：它暴露 schema、split 和样本预览，也会因文件过大、首行过大或 Parquet row group/page index 不合适而失败。发布验收应检查 Viewer 是否开启、私有访问策略、shard 大小、Parquet 行组、README YAML 和 `load_dataset` 行为是否一致 [69, 130]。

Cache fingerprint 给数据变换一条机器可比对的身份链。Datasets cache 文档说明，缓存会保存已下载和已处理的数据，fingerprint 初始来自 Arrow 表或文件，后续由前一步 fingerprint 与 transform hash 组合，`map` 参数也进入 hash [68]。如果 transform 不能被 dill

或 pickle 序列化，库会退回随机 fingerprint 并警告；禁用 caching 时缓存文件只在临时目录保留，会话结束后删除，需要 `save_to_disk` 才能保存变换结果。因此 run card 不应只写“用 Datasets 清洗”，而应记录每步代码版本、函数可序列化性、fingerprint、cache 路径、`save_to_disk` 位置、Hub revision 或 commit，以及上传时的 shard 配置和访问 token 策略。

2.2 分词是建模选择

Tokenizer 把文本转成离散 token。BPE、SentencePiece 和 byte fallback 等方案不只是预处理工具，它们改变序列长度、词表大小、嵌入矩阵形状和模型可学习的单位 [213, 140]。对中文、代码、数学符号和低资源语言而言，token fertility 尤其重要：同样一句话被切成更多 token，会占用更多上下文和训练计算。

Tokenizer 与模型权重绑定。更换 tokenizer 会改变 token id、特殊 token、聊天模板和长度分布。生产系统应把 tokenizer、chat template、special tokens 和模型权重一起版本化。教学 BPE 代码和 GPT 风格 tokenizer 不能混为一谈。教学实现常会先 lowercase，用 word-punctuation tokenizer，把未知词拆成带 SOW/EOW 边界的子词，并保留 UNK/PAD；GPT 风格 byte-level tokenizer 则通常要求任意字节都可表示，避免 unknown-token 路径，并在训练中把文本打包成连续 token stream 而不是每条样本固定 padding。数据报告应写明 tokenizer 家族、是否归一化或小写化、special token 由谁插入，以及 encode/decode 是否精确可逆。

更细地看，tokenizer pipeline 至少包含 normalization、pre-tokenization、model 和 post-processing 四层 [107]。Normalization 可能做 Unicode 规范化、去重音或小写化；pre-tokenizer 决定空格、标点、数字和代码符号如何先被切开；model 层才是 BPE、Unigram、WordPiece 或 WordLevel 的词表规则；post-processor 负责插入 `[CLS]`、`[SEP]`、BOS、EOS 或对话边界。改变 normalizer 或 pre-tokenizer 通常意味着应重新训练 tokenizer；只改 post-processor 不一定改变词表，却会改变模型实际看到的序列。因此出版报告不能只写 tokenizer 名称，还要说明四层配置和它们是否参与训练。

教育版 BPE 的价值在于暴露边界，而不是替代生产 tokenizer。若实现把常见词放入 word vocab，把罕见词拆成 SOW、若干 BPE n-gram、EOW，并用 UNK 和 PAD 处理失败与定长 batch，那么它已经引入了 lowercase、strip、word tokenizer、required tokens、strict inverse decoding、fixed-length 截断和 padding 等协议。读者应能从实验报告中看见这些协议：`vocab_size` 如何在 word vocab 与 BPE vocab 间分配，`ngram_min/max` 允许哪些合并，空字符串会不会只剩 padding，未知标点会不会变成 UNK，`inverse_transform` 遇到第二个 SOW 或孤立 EOW 时是报错还是容错。否则“我训练了一个 BPE tokenizer”这

句话无法说明模型实际学到的 token 语言。

Tokenizer 测试应覆盖空字符串、纯空白、中文姓名、emoji、代码缩进、数学符号、坏 Unicode、markup，以及系统、用户、助手、工具消息中真实使用的 special token。BOS、EOS、padding 和角色标记由谁插入必须写清楚；双重插入 BOS/EOS 会让标签整体错位，漏插终止符会让生成边界不稳定。词表扩展也不是元数据修改。新增 token 需要初始化并训练新的 embedding 和输出矩阵行；若输入输出权重 tied，读入和生成都会同时受影响。

Chat template 是 tokenizer 合同的一部分，而不是 UI 字符串。现代 chat tokenizer 会把 `messages` 列表渲染成模型预期的 role marker、换行、结束符和 assistant 起始符；有的模板还接收 tool schema 或检索文档 [63]。训练时通常不应添加 generation prompt，因为目标回答已经在样本中；推理时则常需要 `add_generation_prompt=True`，或在 assistant prefill 场景使用继续最后一条消息的语义。如果先把模板渲染成字符串再单独 tokenize，必须明确是否禁用额外 special token，否则 BOS/EOS 或 role marker 可能被重复插入。模板中的空白、Jinja 分支、工具变量和用户文本中的伪 special token 都应进入回归测试。

一个可计算的 tokenizer 诊断是 token 效率。对语言或领域切片 g，可记录

$$\rho_g = \frac{\sum_{d \in \mathcal{C}_g} \mathrm{bytes}(d)}{\sum_{d \in \mathcal{C}_g} |\tau(d)|}.$$

这个值越高，通常表示同样上下文窗口能容纳更多原始文本；但它不是质量指标，只是成本和覆盖面的线索。中文、数学、代码和表格文本的 ρ_g 差异会直接变成训练 token 预算和推理延迟差异。若一个模型报告宣称多语言公平，却没有展示不同语言的 token fertility、截断率和生成 token 成本，就缺少数据层证据。

比较 tokenizer 也不能只看平均 fertility。出版级实验应按语言、领域和任务切片报告 truncation rate、prompt budget、生成 token 成本、罕见标识符、数字串、复制任务和代码格式稳定性。词表变大可能降低序列长度，却会增加 embedding 和 softmax 参数、加载成本与稀有 token 学习难度；词表过小则会把少数语言、长数字、URL 和代码变量切碎。好的结论应说明 tokenizer 在训练成本、推理成本和任务可靠性之间怎样取舍。

2.3 预训练之外的信号

监督指令数据把结构化对话序列化为 token；偏好数据把一个 prompt 下的候选回答排序；验证器数据用单元测试、数学答案或工具执行结果给出奖励；检索语料在推理时提供外部证据。它们都不是简单文本，必须记录格式、掩码、边界和评测污染风险。评测污染是现代大语言模型研究的核心问题。模型可能见过原题、改写、答案解析、翻译版本或合成变体。任何高分都应回答一个问题：这个分数测量的是泛化能力，还是训练和开发流程已经接触过测试信息 [247]？去重同样有多层含义。训练集内部去重可以避免重复文档被

过度加权；训练集和评测集之间去重可以保护 benchmark 有效性；跨许可证去重可以避免禁止来源通过镜像或转载重新进入语料。近重复比精确重复更难处理，因为一个答案解析、论坛转载或自动翻译版本都可能携带测试信息。对于高价值评测，团队应同时做字符串级、token n-gram、文档来源和语义模板级检查。

预训练、SFT、偏好学习、验证器训练和 RAG 的数据 schema 应彼此隔离。SFT 中 prompt token 通常不计入 assistant loss，标签应把用户、系统、工具观察和 padding 位置设为 ignore index；偏好数据必须保留 chosen/rejected 的成对关系；工具轨迹应区分自然语言、工具名、参数、返回值和错误通道；检索语料应保留来源、日期、许可证和访问权限。把这些数据全都拼成普通文本，会让训练目标看似统一，却丢掉最重要的监督语义。

SFT 数据整理还要记录模板和特殊 token 的来源。许多实操代码会先补齐 pad、eos、bos 和 unk token，必要时调用 `resize_token_embeddings`，再把 source 前加 BOS、target 后加 EOS，并用 `add_special_tokens=False` 避免 tokenizer 再插一次。若 `train_on_source=False`，source span 的 label 要改成 `-100` 或 ignore index；若 `train_on_source=True`，模型会被训练去复述 prompt。这些选择必须进入 run card，因为它们决定模型学的是“回答用户”，还是“续写整段对话模板”。

打包训练序列时，硬件希望得到密集矩形张量，但文档长度不同。简单 padding 容易理解却浪费计算；连续拼接和固定 block 切片能提高 token 利用率，却会把无关文档放到同一上下文窗口。是否插入 EOS、是否允许跨文档注意、padding label 是否设为 ignore index、position id 如何处理 left padding 和 cache，都会改变训练信号。很多异常低 loss 不是模型更聪明，而是边界、mask 或标签右移写错。

玩具数据和网页数据的差别也应进入报告。字符级 Shakespeare 预处理会把所有字符排序成 65 个符号，保存 `stoi/itos/vocab_size` 到 `meta.pkl`，因此采样脚本必须优先从 checkpoint 的数据目录加载这个元数据；GPT-2 BPE 版本则没有字符级映射，而是默认使用 `tiktoken.get_encoding("gpt2")`。同一个 `train.bin` 文件名在这两种实验中含义完全不同：前者每个 id 是字符表索引，后者每个 id 是 GPT-2 BPE token。若 checkpoint 没有记录 dataset、tokenizer 和 meta 路径，采样脚本可能退回错误编码器，生成仍然会产生字符串，但它已经不再是原训练分布下的模型。

污染审计应当是不对称的。不能只查训练集中是否含有原题，还要查答案键、解析、论坛转载、翻译版、同模板题、合成指令变体和开发过程中反复调榜形成的隐性泄漏。一个实用审计可以先做规范化字符串匹配，再做 token n-gram overlap，对高价值 benchmark 继续做来源级和语义模板级人工复核 [218, 247]。如果某个 benchmark 被用于模型选择，它的 prompt、答案和抽取规则就应像普通机器学习里的 test set 一样限制访问。

公开 benchmark 一旦反复参与过滤器、模板、后训练数据或解码策略选择，就已经部分变成开发信号。严肃报告应区分 training data contamination、development leakage 和

inference-time assistance：前者来自参数训练，第二类来自人和脚本围着榜单调系统，第三类来自检索器、工具或答案格式器在测试时提供额外信息。时间切分、私有 holdout、人工复核和失败样本记录应一起报告，避免把调参收益误写成泛化能力。

2.4 深入展开：数据管线不是预处理

本章把数据称为“规格”，意思是数据管线不是训练前的杂务，而是决定模型行为的第一层工程。一个语料混合会显式或隐式规定模型应该擅长哪些语言、哪些知识领域、哪些文本风格、哪些代码习惯和哪些安全边界。网页、书籍、论文、代码、问答、论坛、对话、数学题和工具轨迹都携带不同信号；把它们混在一起之前，必须知道每类数据的来源、时间、许可证、清洗规则和权重。

Tokenizer 在本章中被视为建模选择，而不是无害工具。词表大小影响 embedding 和输出矩阵；切分规则影响序列长度；byte fallback 影响罕见字符和多语言覆盖；special token 影响聊天角色、工具调用和多模态占位符。对中文而言，同一句话如果被切成更多 token，就会在训练和推理中花费更多步数；对代码而言，缩进、括号、标识符和空白的切分会影响生成格式稳定性。换 tokenizer 等同于改变模型输入语言，不能在预训练后随意替换。本章还区分了多种训练信号。预训练数据提供延续信号，SFT 数据提供示范信号，偏好数据提供比较信号，RLVR 数据提供验证信号，RAG 语料提供推理时证据。它们的共同点是都必须被序列化为模型可处理的 token 或上下文，但它们的语义完全不同。把偏好数据当作普通文本训练，会丢掉 chosen/rejected 关系；把工具轨迹当作普通对话训练，会丢掉 schema、权限和错误通道。污染控制是这一章的另一个中心。本章强调污染不是二元问题。模型可能见过原题、答案、解析、相似模板、翻译版本、合成变体或论坛讨论。越是被频繁用于模型选择的公开 benchmark，越需要像科学实验中的 test set 一样保护。严肃团队应在数据进入训练前建立 manifest，并在发布报告中说明污染检查方法，而不是只写一句“we deduplicated the data”。

从出版角度看，数据章节还要能解释“为什么这个模型可以被相信”。读者应能从文本中复原四件事：哪些来源以什么许可进入训练；每类来源在多少训练 token 中出现了多少次；tokenizer 与模板如何把文本变成监督信号；评测样本、答案、解析和派生版本如何被隔离。只报告最终 loss、benchmark 分数和几个样例回答，会把数据责任转移给读者猜测；高质量书稿应把这些隐含约定显式化。

2.5 章节细节

2.5.1 数据作为规格

数据决定模型能看见什么、不能看见什么、反复看见什么，以及以什么风格看见。语料来源、时间、许可证、重复率、语言分布、代码比例和安全过滤都会进入最终行为。把数据称为规格，是因为它像 API 合同一样定义系统边界。

2.5.2 分词作为建模选择

Tokenizer 决定文本被切成哪些基本单位。它影响序列长度、上下文消耗、嵌入矩阵大小、特殊 token 设计和多语言公平性。中文、代码、数学符号和低资源语言尤其容易受到 token fertility 的影响，因此 tokenizer 不能被视为无关预处理。

2.5.3 语料构建与混合

语料混合不是把更多文件堆在一起，而是决定不同数据源的训练权重。网页、书籍、论文、代码、论坛、对话和合成数据各自带来不同能力和风险。好的数据报告应说明每类来源如何过滤、去重、抽样和加权。

每个组件都应记录 token 数、采样概率、预期曝光量和重复率。重复率高的组件可能很有价值，例如数学推导或代码评测训练集，但它也更容易被记忆。若验证集来自同一来源或同一模板，重复率还会让验证 loss 显得过于乐观。

2.5.4 预处理产物

数据预处理的最终产物通常包括原始文档索引、过滤决策表、切词分片、`train.bin` 与 `val.bin`、分词器元数据和运行清单。出版稿应说明这些产物的依赖关系：二进制训练流依赖 tokenizer 版本、special-token 约定、dtype、文档边界策略和 split seed；`meta.pkl` 或 tokenizer 配置依赖训练语料和词表构造；验证集依赖切分规则和是否在切分前去重。缺少其中任一项，实验仍可能运行，却无法证明同一个模型可以被复现、审计或合法再分发。

2.5.5 把文本打包成训练序列

Packing 提高吞吐，但也容易制造边界错误。多个文档被放进同一序列时，模型不应跨文档泄漏信息；padding、attention mask、position id 和 loss mask 必须一致。很多看似神秘的训练异常，实际来自序列边界和标签右移错误。二进制 token stream 也必须被版本化。字符级玩具数据和 GPT-2 BPE 因为 token id 小于 65,536，可以用 `uint16` 保

存；但十万级多语言词表就不能直接复用这个 dtype。预处理产物应记录 dtype、tokenizer 或 `meta.pkl`、vocab size、special-token ids、训练/验证切分方式，以及文档之间是否插入 end-of-text token。用错误 tokenizer 读取 `uint16` 文件，张量形状仍然正常，但 token id 语义已经错了。固定块二进制数据集还要说明 header 与分片规则。稳健的 shard 会记录 magic、version、dtype、chunk size 和 block size，再把 token ids 写成可 memory-map 的定长块。分布式 dataloader 必须同时按 process rank 和 worker id 切分文件或 chunk；只按 worker id 切分会让不同数据并行 rank 读到同一批样本。如果 chunk 预填 separator token、按 block index shuffle，或从多个数据集按权重混合，separator id、随机种子、worker cursor 和混合权重都应进入 run manifest 与 checkpoint。

因果语言建模常用长度 $T+1$ 的片段构造输入 $x_{1:T}$ 和标签 $x_{2:T+1}$。这个右移规则很简单，也最容易被写错。Padding label 应设为 ignore index，而不是 padding token id；chat SFT 中则通常只让 assistant span 参与 loss。若把 user token 也放进 label，模型会被训练成模仿用户，而不是回答用户。

2.5.6 预训练之外的训练信号

现代模型使用的不只是 continuation signal。指令数据提供示范，偏好数据提供比较，验证器数据提供可执行奖励，检索语料在推理时提供证据。每类信号都有自己的格式、掩码和风险，不能简单当作普通文本混入。

2.5.7 污染、评测与来源

评测污染包括原题、答案、解析、翻译、改写和合成变体。模型越强、语料越大，越难保证公开 benchmark 未被接触。来源记录、近重复检测和时间切分不是形式主义，而是让评测仍然有解释力的前提。

污染检查应是不对称的：不仅问训练集中是否有原题，还要问是否有答案键、题解、论坛讨论、翻译版本、同模板生成题和合成指令变体。被用于模型选择的 benchmark 应像测试集一样限制访问；若开发过程反复根据公开榜单调过滤器、prompt 或后训练数据，评测已经部分转化为训练信号。

2.6 关键术语、实现要点与练习

关键术语。 Token fertility 表示同一文本被切成 token 的密度；data manifest 记录数据来源、时间、许可证和过滤规则；dataset card 面向使用者说明数据内容、创建过程、许可证、偏差和适用边界；DatasetInfo 记录数据集名称、版本、features、split、hash 和样本统

计等库级元数据；IterableDataset 表示按迭代顺序流式读取的数据对象；数据 artifact 指 tokenizer 语料、预训练语料、指令数据、偏好数据、检索语料和评测集等承担不同契约的数据对象；preprocessing artifact 指切分、tokenizer、二进制 shard、元数据和运行配置等可复现产物；normalizer、pre-tokenizer、tokenizer model 和 post-processor 构成 tokenization pipeline；chat template 把角色消息、工具和检索文档序列化成模型输入；generation prompt 标记模型应开始生成 assistant 回复；重复率表示某个数据组件在训练预算中被预期看见的次数；byte fallback 让 tokenizer 能用字节单元表示罕见或异常文本；sequence packing 把变长文档拼成固定训练块以提高利用率；end-of-text token 标记文档边界或生成终止；去重包括训练集内部、训练-评测之间和跨许可证去重；评测污染指训练或调参过程接触到测试信息；label mask 表示哪些 token 参与损失。

实现要点。数据管线应输出可复现 manifest，并用 dataset card 对外说明用途、许可证、语言、规模、偏差和限制；过滤、PII、许可证、去重和评测重叠决策应保存为可查询字段；若使用 Hugging Face Datasets，应记录 DatasetInfo、features、fingerprint、split、cache 或 streaming 文件模式；对外 Hub 数据集还应记录 repo id、revision、README YAML、文件命名、Viewer 状态、访问权限、上传方式与 shard 大小；流式训练还应记录 shards、shuffle buffer、worker/rank 分配和 epoch seed；tokenizer、normalizer、pre-tokenizer、post-processor、special tokens 和 chat template 必须版本化并做可逆性测试；预处理脚本应记录 split seed、tokenizer 名称、EOT 插入、dtype、shard 顺序和 `meta.pkl` 路径；训练用 chat template 通常不加 generation prompt，推理用模板要说明是否添加 assistant 起始符或继续最后一条 assistant 消息；SFT、偏好、检索和评测数据应保持不同 schema；packing 应检查右移标签、ignore index、position id 和分布式切分；污染检查应包含 exact match、n-gram、来源和语义模板检查。

练习。

1. 选取中文、英文、代码和数学各三句话，比较同一 tokenizer 的 token fertility。

2. 设计一个数据 manifest 表格，包含来源、许可证、时间、语言、过滤规则和权重。

3. 解释为什么“删除完全重复”不足以防止 benchmark contamination。

4. 写出一个 SFT 样本的 label mask，并说明为什么用户 token 不应计算 assistant loss。

5. 给定 $D = (50, 500, 5000)$ million tokens、$p = (0.5, 0.25, 0.25)$ 和训练预算 $S = 2$ billion tokens，计算每个组件的预期曝光量和重复率。

6. 设计 tokenizer 回归测试集合，覆盖中文、代码、emoji、坏 Unicode、BOS/EOS 和 chat role marker，并说明每项测试能发现什么错误。

7. 说明为什么用错误 tokenizer 读取形状正确的二进制 token stream 仍然会破坏训练。

8. 为 tokenizer 语料、预训练语料、指令数据、偏好数据、检索语料和评测集各写一个 manifest 字段清单，并指出哪些字段不能跨 artifact 复用。

9. 对比字符级 Shakespeare、GPT-2 BPE Shakespeare 和 OpenWebText 预处理，列出每个实验必须写进 run card 的 split、tokenizer、dtype、special token、二进制文件和元数据字段。

10. 设计一个过滤和去重决策表，至少包含语言、质量、PII、许可证、近重复、评测重叠和保留理由；说明如何用它追查一个训练后发现的版权或污染问题。

11. 为同一个两轮对话写出训练模板和推理模板，标出 BOS、EOS、role marker、generation prompt、assistant label span 和 ignore-index span。

12. 写一张 tokenizer pipeline 审计表，分别列出 normalizer、pre-tokenizer、model 和 post-processor 的配置、是否需要重新训练、以及会影响哪些下游指标。

13. 为一个用 `streaming=True` 读取 Parquet 网页语料的训练任务写 run card 字段，覆盖 columns、filters、shards、shuffle buffer、worker/rank 分配、epoch seed、DatasetInfo 和 dataset card 链接。

14. 为一个包含 train、validation、test 和两个 config 的 Hub 数据集设计目录结构与 README YAML，说明哪些 split 会被 Dataset Viewer 自动识别，哪些需要手写 `data_files`，并把 cache fingerprint、`save_to_disk` 路径和 Hub revision 写进发布验收表。

2.7 结构化检查表

2.7.1 数据管线报告清单

字段	应记录内容
来源	域名、数据集名、采集时间、镜像关系、是否合成
许可证	训练、再分发、商业使用、撤回和检查要求
语言与领域	语言分布、代码/数学/网页/论文/对话比例
过滤	质量过滤、PII 处理、安全过滤、低质模板删除
去重	exact、near-duplicate、跨 split、跨许可证去重
污染检查	benchmark 原题、答案、解析、翻译和合成变体重叠
混合权重	各来源采样概率、预期曝光量、重复率、epoch 或 token 上限

第二部分

架构、优化与系统

第三章　Transformer 机制

3.1　张量契约

Transformer 的实现首先是张量契约。输入 token id 通常具有形状 $B \times T$，其中 B 是 batch size，T 是序列长度；embedding table 把每个 id 映射为宽度 d_{model} 的向量，得到

$$X \in \mathbb{R}^{B \times T \times d_{\text{model}}}.$$

标准 decoder block 的关键不变量是外层形状保持不变：attention 改变不同位置之间的信息交换，MLP 改变每个位置内部的表示，残差流从第一层到最后一层都保持 $B \times T \times d_{\text{model}}$。许多训练错误不是数学思想错误，而是 mask、padding、batch 维度、head reshape 或 dtype 处理错误。理解 Transformer 时，最可靠的方法不是背图，而是追踪张量。

原始 Transformer 把 self-attention、前馈子层、残差连接和归一化组合成能并行处理序列位置的架构 [233]。现代语言模型修改了位置编码、归一化、MLP、初始化和服务路径，但仍保留同一个中心契约：模型维护一条 residual stream，并反复执行“内容相关的位置混合”和“逐位置的非线性变换”。如果一个实现无法清楚说明 batch 维、序列维、head 维、hidden 维和 vocab 维在哪里出现，它就很难被调试。

把这一契约写成报告，比只画 block 图更有用。一个 decoder block 至少应说明

$$C_{\text{block}} = (B, T, d_{\text{model}}, H, d_h, V_{\text{vocab}}, M_{\text{mask}}, P_{\text{pos}}),$$

其中 H 是 query head 数，d_h 是每个 head 的宽度，V_{vocab} 是输出词表大小，M_{mask} 是可见性规则，P_{pos} 是位置策略。这个向量不能替代完整实现，但它能迫使读者区分“隐藏维切成 head”“序列维进入 attention score”“词表维只出现在 logits”这些常被混淆的地方。

形状检查还要覆盖 silent broadcasting。很多深度学习库会把 $B \times 1 \times T \times T$ 的 mask 自动广播到所有 head，也会把 $1 \times T$ 的 position ids 广播到整个 batch。广播本身没有错，但如果某个样本使用 left padding、另一个样本使用 packed sequence，同一个广播规则就可能让两个样本共享错误位置或错误可见性。因此，单元测试不能只用 batch size 为 1 的整齐样本。

3.2 嵌入与位置

Token embedding 只提供内容，不提供顺序。如果两个序列包含相同 token 但顺序不同，没有位置信息的注意力层无法可靠地区分它们。位置信息可以通过多种方式进入模型。Learned absolute position embedding 给每个位置加一个可训练向量，形式简单，但最大上下文长度和未见位置外推都受训练配置约束。Sinusoidal encoding 使用多频率正弦和余弦函数，给模型一个确定性位置基底。RoPE 在 query/key 坐标上施加位置相关旋转，让相对距离直接影响注意力分数 [224]。ALiBi 等 bias 方法则不把位置向量写入 residual stream，而是在 attention score 中加入随距离变化的偏置 [193]。

位置 id 不总等于数组列号。Packed sequence、left padding、cached decoding、sliding window 和长上下文裁剪都会让逻辑位置与 tensor column 分离。鲁棒实现应把 position ids 当作显式张量或由经过测试的 cache 更新规则生成。一个常见回归是：full prefill 路径表现正常，但逐 token decode 因为 RoPE 位置或 KV cache append 偏移而逐步漂移。短样本可能看不出问题，长文档和多轮对话会首先暴露。

位置策略也应写进模型报告。至少要说明训练最大长度、评测最大长度、是否使用 RoPE scaling、是否有 sliding window 或 attention sink、padding 在左侧还是右侧、cache 中保存的是物理列号还是逻辑位置。若只写“支持 128k 上下文”，却没有说明训练长度分布、位置外推方式和长文档切片评测，这个声明无法解释模型是真的使用远处证据，还是只是在更大的输入窗口内读取近处片段。

3.3 缩放点积注意力

对查询 Q、键 K、值 V，注意力计算为

$$\mathrm{Attention}(Q,K,V)=\mathrm{softmax}\big((QK^{\top}/\sqrt{d_k})+M\big)V.$$

Self-attention 从同一条 residual stream 产生 $Q=XW_Q$、$K=XW_K$ 和 $V=XW_V$。多头注意力会把投影结果 reshape 成 $B\times H\times T\times d_h$，其中 $Hd_h=d_{\text{model}}$；attention score 的形状是 $B\times H\times T_q\times T_k$。softmax 后乘以 V，再把 heads 拼回 $B\times T\times d_{\text{model}}$ 并经过 output projection。多头不是简单复制同一个计算，每个 head 有自己的相似度空间、value 空间、位置偏置和 mask 视角。

除以 $\sqrt{d_k}$ 是为了控制点积分布尺度。head 维度变大时，未缩放的点积方差会增长，softmax 容易过早变尖，梯度信号变差。完整 attention 的主要计算量是 $O(BT^2d_{\text{model}})$，显式 attention weights 的内存是 $O(BHT^2)$。因此 context length 不是单纯的模型超参，也是训练和服务的系统预算。长上下文模型若没有说明 attention kernel、KV cache、padding/packing 和批处理策略，只报告最大长度是不完整的。

实现中的 attention kernel 要同时守住数学等价和 I/O 预算。标准写法逻辑上有 $T_q \times T_k$ 的 score 矩阵，但高效 kernel 不一定把完整矩阵写回显存。FlashAttention 通过分块和 online softmax 避免不必要的高带宽内存读写，同时保持精确 attention 结果 [28]。因此报告 attention 优化时，应区分“改变可见 token 集合”的稀疏或窗口方法，以及“保持同一数学结果但改变内存访问”的 kernel 方法。

数值等价也要被测试。若一个 fused attention kernel、标准 eager attention 和带 KV cache 的 decode 路径在同一小 batch 上输出差异很大，不能简单归因于浮点误差。排查顺序通常是 mask 极性、position ids、dtype promotion、dropout 状态、causal 标志、padding 长度和 cache layout。只有这些不变量固定后，才讨论 BF16、FP8 或不同 GPU kernel 带来的容差。

把公式落到当前 PyTorch 时，最直接的接口是 functional SDPA。PyTorch 2.12 把 `query`、`key`、`value`、`attn_mask`、`dropout_p`、`is_causal`、`scale` 和 `enable_gqa` 都写成显式参数 [200]。`attn_mask` 可以是布尔 mask 或加性 mask：布尔 mask 中 `True` 表示该位置允许参与注意力，非布尔 mask 则作为 bias 加到 score 上。若 `is_causal=True`，函数会构造下三角因果 bias；教学实现中不应再同时传入含义不清的普通 mask，否则同一个可见性约定会在两个入口重复表达。

`dropout_p` 也不是模块状态的隐式细节。SDPA 文档明确说明该函数会按传入的 `dropout_p` 应用 dropout，因此 evaluation 路径必须显式传 `0.0`，而不是假设外层模块调用 `eval()` 后 attention 内部自动关闭随机性。若 full forward 与 cached decode 的等价测试没有固定 `dropout_p`，测试会把随机差异误判成 cache 或 mask 错误。若启用 `enable_gqa`，等价实现会按 query head 与 key/value head 的比例重复 K/V head；这说明 GQA 的张量契约，但服务端 KV cache 仍应保留原始 KV head 布局，不能为了对齐 query head 而无谓展开缓存。

Kernel 选择也应进入 run card。当前 SDPA 后端可以通过 `sdpa_kernel` 上下文管理器约束，`SDPBackend` 包括 `MATH`、`FLASH_ATTENTION`、`EFFICIENT_ATTENTION` 和 `CUDNN_ATTENTION` 等后端 [199]。因此 benchmark 表格除了吞吐和显存，还应记录 backend、dtype、head dimension、mask 类型、是否有 dropout、是否启用 GQA、是否使用 `torch.compile`，以及容差如何随 kernel 变化。否则“attention 更快”无法区分算法、后端 dispatch 和批处理策略的贡献。

图 3.1 把本节的主要张量契约画成一条形状保持路径。这个图的目的不是替代公式，而是提醒实现者同时检查残差形状、head reshape、mask 形状、kernel 选择和 cache 路径。

mask M 决定 token 能看见什么。因果语言模型使用下三角可见模式，保证第 t 个位置不能看到未来 token。这个 mask 是自回归训练和生成的根本约束。如果训练时第 t 个位置能看到 x_{t+1}，loss 会异常好看，但推理时未来 token 不存在，生成会崩坏。

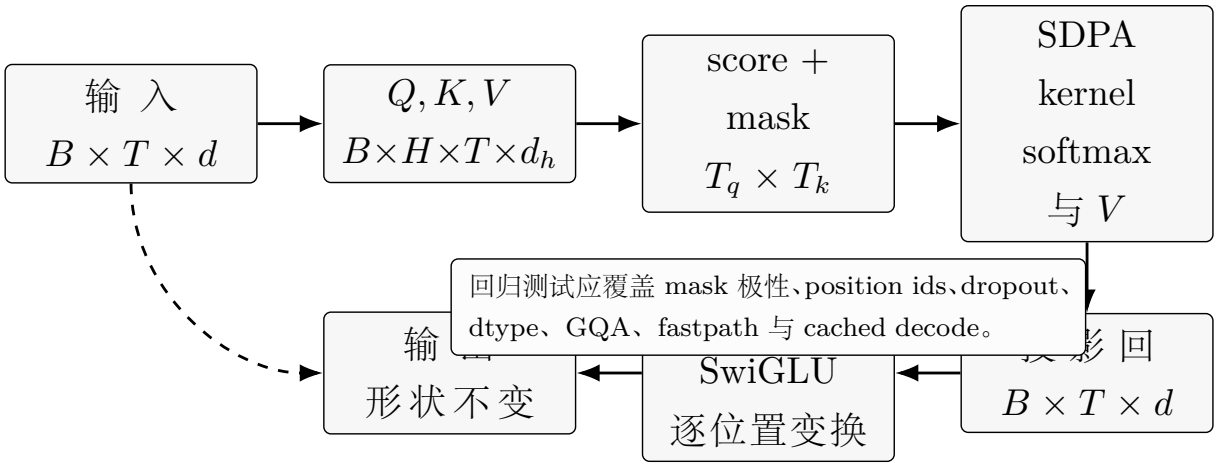

图 3.1: Transformer block 的实现首先是张量契约。该图是对原始 Transformer、RoPE 和 FlashAttention/SDPA 实现约束的原创综合 [233, 224, 28]。

mask 约定是工程上最容易出错的部分之一。某些库用布尔值 True 表示允许注意，另一些库用 True 表示屏蔽；加性 mask 用一个极小负数近似 $-\infty$，乘性 mask 则改变概率归一化前后的语义。混合精度下，如果 softmax 前没有减去行最大值，或者 masked position 的数值不够小，就可能产生 NaN、未来信息泄漏或训练损失异常偏低。教学版 encoder-decoder 代码通常会把约定写得很具体：padding mask 是 $B \times 1 \times T_q \times T_k$ 的布尔张量，True 表示允许注意；decoder self-attention 再把 padding mask 和下三角 no-peek mask 相与，然后在 softmax 前执行 `masked_fill(mask == 0, very_negative)`。这个写法本身没有问题，但前提是所有 mask 极性一致。Cross-attention 的 mask 还是 $T_{\text{dec}} \times T_{\text{enc}}$ 的矩形，因此只测试方阵 self-attention mask 不能覆盖 encoder-decoder 实现里的全部错误。

PyTorch 的两个常用入口还刻意保留了不同极性：SDPA 的布尔 `attn_mask=True` 表示允许参与注意力，`nn.MultiheadAttention` 的布尔 `attn_mask=True` 表示对应位置不允许注意，`key_padding_mask=True` 则表示该 key 位置应被忽略 [201]。若同时传 `attn_mask` 和 `key_padding_mask`，两者类型还应匹配。`MultiheadAttention` 的 `is_causal` 是对 `attn_mask` 的因果提示，错误提示会导致错误执行或兼容性问题；`need_weights=False` 才更容易走优化的 SDPA 路径。迁移代码时，应把每个 mask 的极性、形状、广播路径和是否取反写进测试名，而不是只在注释里写“causal mask”。

`MultiheadAttention` 的推理 fastpath 也不是无条件可用。官方文档把它限制在自注意力、三维 batch、`batch_first=True`、关闭训练、关闭 autograd 或没有 requires-grad 参数、`add_bias_kv=False`、`add_zero_attn=False`、`kdim=vdim=embed_dim`、autocast 关闭等条件下；若传入 `NestedTensor` 表达 padding，还不能同时传 `key_padding_mask` 或 `attn_mask`。这类条件应出现在性能声明旁边，因为一次看似无害的参数变化就可能让同一模型从 fastpath 回到普通路径。

一个最小 mask 回归集应包含四类样本：没有 padding 的因果序列、左 padding 的短序列、packing 后的两段文档、以及 encoder-decoder 的矩形 cross-attention。测试不只比较 mask 张量形状，还要改动被屏蔽 token 的内容并断言允许位置的 logits 不变。这样可以直接发现未来泄漏、跨文档泄漏、padding 泄漏和 cross-attention 极性错误。

3.4 残差、归一化与前馈层

深层 Transformer 可训练，是因为每层在 residual stream 上写入增量信息，而不是完全替换隐藏状态。Pre-norm decoder 常写成

$$Y = X + \mathrm{Attention}(\mathrm{Norm}_1(X)), \qquad Z = Y + \mathrm{MLP}(\mathrm{Norm}_2(Y)).$$

残差路径给梯度提供穿过深度的直接通道，归一化控制进入 attention 和 MLP 子层的激活尺度。Post-norm 把归一化放在 residual add 之后，原始 Transformer 使用这种顺序，但深层 decoder 通常更偏向 pre-norm，因为子层输入分布更稳定。

LayerNorm 会减去均值并除以标准差；RMSNorm 去掉均值项，只用 root mean square 重缩放隐藏向量 [261]。这个差别在公式上很小，在实现上却重要：RMSNorm 少一次均值归约，常见于 LLaMA 类 decoder。归一化不等于尺度问题消失。残差分支、初始化、学习率、activation function 和混合精度仍会决定激活是否保持在可用范围内。训练早期记录每层 activation norm 和 gradient norm，往往比只看总 loss 更早发现 mask、初始化或 dtype 错误。

前馈层是逐位置的非线性变换。经典 FFN 是两层 MLP，通常先扩展到数倍 hidden width，再投影回 d_{model}。SwiGLU 等门控结构把输入投影分成 gate branch 和 value branch，用逐元素乘法增加表达能力，已成为许多 LLaMA 类模型的默认选择 [215]。这也改变参数账：简单 $4d$ MLP 与门控 MLP 的中间宽度不能直接比较，报告模型结构时应写清 input projections、gate projection、output projection 和是否使用 bias。

前馈层的调试指标也不同于 attention。Attention 错误常表现为泄漏、NaN 或 cache 不一致；MLP 错误更常表现为激活尺度漂移、门控饱和、初始化不匹配或张量并行切分错误。对 SwiGLU，应检查 gate 分支和 value 分支的形状、激活分布、乘法前后的范数，以及 output projection 是否把中间宽度正确投回 residual stream。若这些指标缺失，参数量相同也不能说明两个 block 可比较。

小型翻译教学实现不能直接等同于现代 decoder 配方。原始 Transformer 风格的代码常用 sinusoidal absolute position、ReLU 前馈、子层后 dropout，以及“sublayer、dropout、residual add、LayerNorm”的 post-norm 顺序；这对六层 encoder-decoder 教学模型是合理的。但迁移到 LLaMA 类 decoder 时，往往要同时改成 pre-norm RMSNorm、RoPE、门控

MLP、无 cross-attention、KV cache 感知的位置更新，以及不同初始化和优化器日程。迁移时应保留 shape 与 mask 测试，但不要无意继承旧实现的 norm 顺序和位置机制。

注意力权重可以辅助诊断，但不能被当作完整解释。一个 head 关注某个 token，并不证明该 token 导致了最终输出；value projection、后续层、MLP 和残差路径都可能改变或擦除信息。因此，分析 Transformer 时应结合注意力可视化、受控干预、held-out loss、下游任务和生成行为。

3.5 深入展开：从张量形状理解 Transformer

本章的写法是“从张量契约推导结构”。输入 token id 的形状通常是 B×T，embedding 后变成 B×T×d。多头注意力再把隐藏维度拆成 head 数和每个 head 的维度，形成 query、key、value。只要这些形状的含义不清楚，模型实现就会在 mask、padding、cache 或并行切分处出错。缩放点积注意力中的 head 维度平方根缩放不是装饰项。head 维度变大时，未缩放的点积方差会增长，softmax 容易过早变尖，梯度信号变差。softmax 的数值实现也必须稳定：通常先减去每行最大值，再加入 mask，并在混合精度中谨慎处理极小负数。许多“模型不收敛”的问题，实际是 attention score overflow、mask 方向写反或 padding 被模型看见。因果 mask 是语言模型目标的一部分。如果训练时第 t 个位置能看到未来 token，训练 loss 会异常好看，但生成时模型无法访问未来信息，结果会崩坏。packed sequence、left padding、sliding window 和 KV cache 会让“数组位置”和“逻辑位置”不同，因此 position ids 应被显式构造和测试。长上下文扩展时，位置处理错误会在短样本上看不出来，却在长文档中造成严重退化。残差流是理解深层 Transformer 的核心。每层 attention 和 MLP 都不是替换隐藏状态，而是在残差路径上增量写入信息。Pre-norm 让子层输入分布更稳定，也让梯度更容易穿过深层网络。Attention heads、MLP neurons、残差通道和 normalization 共同塑造行为，所以解释模型时不能把某个 attention map 直接等同于“模型理由”。

3.6 从图到测试

架构图只有转化为测试才真正有用。高质量实现应比较 full forward 和 cached token-by-token decoding 在同一 prefix 上的 logits，断言 harmless padding 不改变真实 token logits，并验证单 token continuation 使用的 RoPE position 与完整上下文一致。若 $f_\theta^{\text{full}}(x_{1:t})$ 与 $f_\theta^{\text{cache}}(x_{1:t})$ 在同一前缀上差异显著，原因往往不是 attention 理论，而是 position id、mask、dtype、cache layout 或 batch 拼接不一致。

$$\max_{t \le T} \| f_\theta^{\text{full}}(x_{1:t}) - f_\theta^{\text{cache}}(x_{1:t}) \|_\infty \le \epsilon.$$

回归项	最小构造	暴露的问题
Full/cache 等价	同一 prefix 分别走完整前向和逐 token decode，比较 logits	position ids、cache append、dtype 或 layout 漂移
Padding 不变性	在真实 token 外加入左/右 padding，断言真实位置 logits 不变	padding 泄漏到 attention 或 loss
Packed 边界	把两段文档 packing 到同一行，屏蔽跨段可见性	样本之间互相偷看证据
Mask 极性	修改被屏蔽 token 内容，断言允许位置输出不变	boolean/additive mask 约定写反
Norm/MLP 尺度	记录每层 activation norm、gradient norm 和 gate 范数	初始化、学习率、混合精度或门控饱和错误

表 3.1: Transformer 最小回归项、构造方式与暴露问题检查表。

这里的 ϵ 应按 dtype 和 kernel 选择给出，而不是事后随意放宽。测试还应固定 dropout、采样开关、padding side、batch 排列和 stop token；否则同一个断言可能同时混入随机性、模板差异和真正的 cache 错误。

训练循环也有清晰契约。Teacher forcing 中，decoder input 是目标序列去掉最后一个 token 的前缀，labels 是同一序列左移后的 token；loss mask 必须忽略 target padding，而不是随手复用 source vocabulary 中的 pad id。若源/目标 tokenizer 在玩具实验中刚好共享 `<pad>`，错误会被隐藏；迁移到独立 tokenizer、多语言词表或 chat template 后，这类 bug 会直接污染 loss 和停止行为。

3.7 最小回归套件

把 Transformer 写对，不能只靠最终训练 loss。出版级实现说明应把最小回归套件写进实验记录，让读者知道每条关键路径已经被独立验证。这个套件不需要大模型，也不需要昂贵数据；它需要的是小 batch、确定性输入、固定随机种子、关闭 dropout、固定 dtype 和清楚的容差。若小模型在这些测试上不稳定，扩大参数、增加数据或换用更快 kernel 只会把错误隐藏到更昂贵的运行中。

表 3.1 汇总最小回归套件，帮助读者先查实现契约再扩大训练规模。

这些回归项应在 eager attention、fused attention、训练路径、prefill 路径和 decode 路径之间重复。若不同路径只在 BF16 舍入范围内略有差异，可以把容差写进 run card；若差异随序列长度累积，就应优先排查位置更新和 cache 对齐，而不是先怀疑模型能力。长上下文尤其需要这种纪律：短 prefix 通过并不能证明 8k、32k 或 128k 上下文下的 RoPE scaling、sliding window、attention sink 和 KV block 管理仍然一致。

测试还应包含负例。把未来 token 改掉而 logits 不变，说明因果 mask 生效；把允许 token 改掉而 logits 变化，说明测试确实有敏感度；把 padding token 改成高频词而真实 token logits 不变，说明 padding 没有泄漏。没有负例的单元测试容易变成“只检查代码能运行”，而不是检查模型契约成立。

3.8 章节细节

3.8.1 张量契约

Transformer 的核心首先是形状不变量。输入 token id、embedding、query、key、value、attention logits、head 合并和输出投影都有明确维度。实现者如果不能逐步说清这些维度，就很难发现 mask、cache、tensor parallel 切分和 silent broadcasting 中的错误。

出版级架构描述应把形状不变量和测试不变量放在一起：每个 block 的输入输出形状、每个 projection 的权重形状、每种 mask 的广播形状、每条服务路径的 position update，都要能被一个小 batch 复现。这样读者才能从公式走到实现，而不是停留在“有 attention 和 MLP”的抽象图。

3.8.2 嵌入与位置

Token embedding 把离散 id 映射为向量，位置机制让模型知道序列顺序。绝对位置、sinusoidal encoding、RoPE 和 attention bias 把顺序信息注入模型的方式不同。长上下文扩展时，位置外推、训练长度分布和 cache position 更新会直接影响模型能否真正利用远距离信息。

3.8.3 缩放点积注意力

注意力用 query 与 key 的相似度选择 value 的加权组合。除以 head 维度的平方根是为了控制点积分布尺度，避免 softmax 过早饱和。数值稳定实现需要处理 mask、极小值、混合精度和行最大值，否则训练可能在大模型上失控。使用 SDPA 时还要记录 backend、`dropout_p`、GQA 和 `is_causal`，因为这些参数决定数学路径、随机性和可见性。

3.8.4 Mask 与因果性

因果 mask 定义了语言模型的任务：当前位置只能看见过去。Padding mask、防泄漏 mask、packed sequence mask 和 sliding-window mask 都会改变可见信息。训练时

一次 mask 写反，就可能得到虚假的低 loss 和不可用的生成模型。PyTorch SDPA 与 `MultiheadAttention` 的布尔 mask 极性不同，迁移时必须显式取反并测试。

3.8.5 残差流与归一化

残差连接让每层在已有表示上写入增量信息，而不是完全替换隐藏状态。Pre-norm 让梯度更容易穿过深层网络，RMSNorm 和 LayerNorm 则控制激活尺度。解释模型时应理解残差流、attention 和 MLP 的共同作用，而不是只看某个注意力图。

3.8.6 前馈块

MLP 或 FFN 是逐位置的非线性变换，通常占 Transformer 参数和计算的大头。SwiGLU 等门控结构提高表达能力，但也改变参数量、activation memory 和初始化尺度。MoE 可以把 FFN 变成稀疏专家集合，从而扩大总容量但增加路由和通信问题。

3.8.7 调试不变量

小模型调试应检查形状、mask、loss 右移、梯度、初始 loss、过拟合小 batch 和生成样例。一个能在小数据上收敛并复现实验不变量的实现，才值得扩展到昂贵训练。CS336 式训练纪律的核心就是先让每个局部假设可测。

3.9 关键术语、实现要点与练习

关键术语。 Residual stream 是贯穿所有层的隐藏状态；causal mask 阻止当前位置看到未来 token；multi-head attention 让模型在多个投影子空间中比较 token；scaled dot-product attention 用 query/key 点积、$\sqrt{d_h}$ 缩放、mask 和 softmax 得到 value 加权和；`scaled_dot_product_attention` 是 PyTorch 中把 SDPA、mask、dropout、causal 标志和 GQA 暴露为显式参数的函数；`SDPBackend` 和 `sdpa_kernel` 描述或约束 SDPA 后端；`MultiheadAttention` 是参考式多头注意力模块，其布尔 mask 极性与 SDPA 不完全相同；attention kernel 是实现 attention 数学的具体计算与内存访问路径；position ids 表示 token 的逻辑位置而不一定等于数组列号；pre-norm 把归一化放在子层之前；RMSNorm 用均方根重缩放隐藏向量；tokenwise MLP 对每个位置独立应用同一前馈网络；prefill/cache equivalence 要求完整前向和逐 token 缓存解码在同一前缀上输出一致；attention weights 是诊断信号但不是完整解释。

实现要点。每个实现都应测试张量形状、mask 方向、padding 处理、position ids 和 cache 等价性；softmax 前应做数值稳定处理；attention kernel 优化应说明是否保持精确数学结果，

并记录 `SDPBackend`、dtype、head dimension、`dropout_p` 和 GQA；从 `MultiheadAttention` 迁移到 SDPA 时要核对 `attn_mask`、`key_padding_mask` 与 `is_causal` 的语义；packed sequence 和 KV cache 下逻辑位置可能不等于数组位置；teacher forcing 的 input、label 和 loss mask 必须写成单元测试；解释模型行为时应结合干预而不是只看注意力图。

练习。

1. 给定 $B = 2, T = 4, d = 8, h = 2$，写出 Q, K, V 和 attention score 的形状。

2. 写一个因果 mask 矩阵，并说明 additive mask 和 boolean mask 的差异。

3. 解释 pre-norm 为什么通常比 post-norm 更容易训练深层 decoder。

4. 设计一个测试，检测模型是否在训练时偷看未来 token。

5. 比较 learned absolute position、sinusoidal encoding、RoPE 和 ALiBi，说明位置信息分别进入 residual stream、query/key 或 attention score 的哪个位置。

6. 设计一个 full forward 与 KV-cache decoding 的一致性测试，写出需要固定的 position ids、mask、dtype 和容差。

7. 给定 $B = 4, T = 2048, H = 16$，估算显式 attention weight 张量的元素数，并说明为什么 kernel 可以不 materialize 完整矩阵但仍保持精确 attention。

8. 为 left padding、packed sequence 和 cross-attention 各写一个最小 mask 测试样本，说明改动哪个被屏蔽 token 能暴露泄漏。

9. 查阅 PyTorch SDPA 与 `MultiheadAttention` 文档，写出两者布尔 mask 极性的差异，并给出从一个接口迁移到另一个接口时的取反测试。

10. 设计一个 SDPA benchmark run card，列出必须记录的 backend、dtype、head dimension、`dropout_p`、GQA、`is_causal` 和 `need_weights` 条件。

3.10 结构化检查表

3.10.1 Transformer 实现检查

组件	必须验证	典型失败
Embedding	token id、padding、special token	特殊 token 与模板不一致
Attention mask	causal、padding、packed sequence	偷看未来或只看 padding
Attention API	SDPA/MHA mask 极性、`is_causal`、`dropout_p`	迁移后取反错误或 eval 仍有 dropout
SDPA backend	`SDPBackend`、dtype、head dimension、GQA	性能与容差不可复现
Position ids	left padding、cache、长上下文	位置漂移导致长文档退化
Softmax	行最大值、mask 数值、dtype	overflow、NaN、异常低 loss
Residual/norm	pre-norm 顺序、残差形状	深层训练不稳定
KV cache	cache append、位置更新、释放	生成错位或显存泄漏

第四章　从 Transformer 到 GPT

4.1　Decoder-only 契约

GPT 类模型去掉 encoder-decoder 结构，只保留因果 decoder。它的统一接口是：给定前文，预测下一个 token。问答、翻译、代码补全、函数调用和对话都可以被写成同一个序列建模问题。这种统一接口的优势是简单和可扩展。模型不需要为每个任务定义新的输出头，只要把任务、上下文、角色和目标格式写进 token 序列即可。代价是所有边界都必须被序列化：system message、user message、assistant answer、工具调用、工具返回、图像占位符、结束符和拒答格式都必须变成模型能学习的 token 模式。

更形式化地，给定 token 序列 $x_{1:T}$，decoder-only 语言模型把联合概率分解为

$$p_\theta(x_{1:T}) = \prod_{t=1}^{T} p_\theta(x_t \mid x_{<t}).$$

这个分解让任意文本位置都变成监督目标，也解释了为什么网页、代码、对话和结构化片段可以进入同一个预训练接口。它同时带来一个强约束：位置 t 的计算不能看到未来 token，否则训练时的目标信息会泄漏到输入里，生成时又不可用。Causal mask 因此不是实现细节，而是训练目标和自回归生成之间的契约。

GPT-2 的关键做法是把自然网页文本视为潜在多任务数据：翻译、摘要、问答、阅读理解和代码样例以文本形式混在语料中 [205]。GPT-3 进一步把 zero-shot、one-shot 和 few-shot 写成推理时上下文条件，而不是梯度更新 [14]。因此，一个 few-shot 报告若只写“给了若干示例”是不够的；它还应说明示例数、分隔符、示例采样、上下文预算、答案抽取和 likelihood 归一化规则。

4.2　训练循环

一个最小 GPT 训练循环包括：取样 token block，前向计算 logits，右移标签，计算交叉熵，反向传播，梯度裁剪，优化器更新，周期性验证和 checkpoint。损失曲线应按数据源、长度和任务切片观察，而不是只看全局平均。训练循环的很多错误只在细节中出现。标签右

移错一位会让模型预测当前 token 而不是下一个 token；padding token 没有 mask 会让模型学习填充模式；gradient accumulation 没有正确缩放 loss 会改变有效学习率；checkpoint 没有保存 optimizer state 会使恢复训练后的曲线出现跳变。小模型调试阶段应该专门写测试检查这些不变量。

Teacher forcing 的输入和标签也必须写成测试：decoder 输入应是去掉最后一个 token 的目标前缀，label 应是同一序列左移一位后的目标 token；loss mask 应忽略 target padding，而不是随手使用 source vocabulary 里的 pad id。玩具翻译任务里源/目标 `<pad>` id 可能刚好相同，错误会被隐藏；一旦迁移到独立 tokenizer、多语言词表或 chat template，这类 bug 会直接污染 loss 和生成停止行为。

实际实现通常从长度为 $T+1$ 的 packed token 片段构造一个训练样本：输入是前 T 个 token，标签是后 T 个 token。若 logits 形状为 $B \times T \times V$，loss 应只在有效标签上平均：

$$\mathcal{L} = -\frac{1}{N_{\text{valid}}} \sum_{b,t} m_{b,t} \log p_\theta(y_{b,t} \mid x_{b,\le t}).$$

这里的 $m_{b,t}$ 既处理 padding，也处理文档边界、prompt mask 和指令微调中的非 assistant 片段。一个最小单元测试可以直接喂入 `[10,20,30,40]`：若输入为前三个 token，标签应对齐为后三个 token；若标签等于输入，模型训练成了近似 autoencoder，而不是因果语言模型。

把这个不变量落到 nanoGPT 的数据路径时，通常会先把 GPT-2 BPE token 写入 `train.bin` 和 `val.bin` 两个 `uint16` memmap，并在文档边界放入 end-of-text token。批处理不是重新解析文本，而是在 memmap 上随机取 offset，构造 `x = data[i:i+T]` 和 `y = data[i+1:i+1+T]`。若数据目录有 `meta.pkl`，训练脚本从中读取真实词表大小；否则常用 padding 到 64 倍数的 `50304`，这和 GPT-2 checkpoint 的真实 `50257` 词表要区分。

分布式训练的有效 token 数也应写成可审计量。若 micro-batch 为 B，上下文长度为 T，梯度累积步数为 A，进程数为 W，一次 optimizer step 看到的 token 数约为 $BTAW$。DDP 启动后，常把全局 `gradient_accumulation_steps` 按 world size 拆到每个进程；每个 micro-step 的 loss 先除以累积步数，只在最后一个 micro-step 打开梯度同步。混合精度下还要先 unscale 再 clip gradient；checkpoint 至少应保存 `model_args`、optimizer state、iteration、best validation loss 和配置，否则 resume 后的曲线没有可比性。

困惑度只是 $\exp(\mathcal{L})$ 的损失尺度，不是通用能力分数。Tokenizer、语料、上下文长度、文档 separator、去重和验证集切分都会改变它。一个字符级模型和一个 BPE 模型对同一句话产生的 target 数不同，困惑度不能直接横向比较。出版级训练图应同时给出 train loss、validation loss、token 数横轴、数据切片和生成样例，否则读者无法判断曲线下降来自建模进步、数据泄漏还是评测预处理差异。

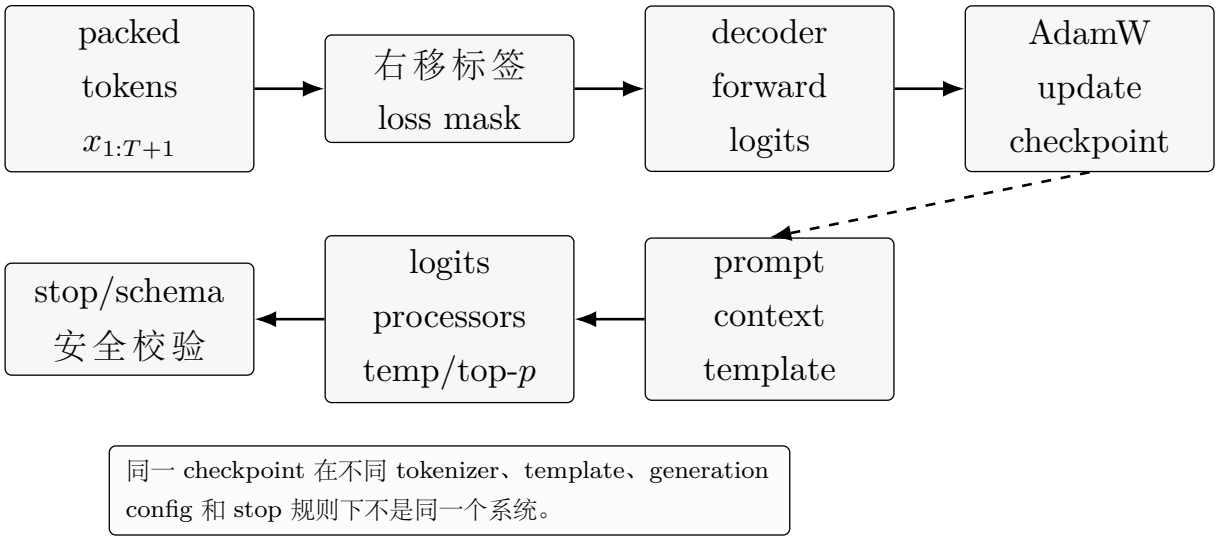

图 4.1: GPT 是统一的自回归接口，但训练和生成有不同的证据边界。该图是对 GPT-2/GPT-3 训练与上下文学习实践的原创综合 [205, 14]。

4.3 从 logits 到文本

生成时，模型把 logits 转成分布。温度控制分布尖锐程度；top-k 和 top-p 控制候选集合；重复惩罚影响循环文本；stop token 和 chat template 控制响应边界。解码策略不是无关细节，它直接影响事实性、创造性、长度、成本和安全过滤。结构化输出使解码更复杂。JSON、SQL、函数调用和工具参数都要求模型在自然语言之外遵守语法约束。生产系统通常需要增量解析、schema validation、失败修复和安全沙盒。一个模型在自由文本中表现良好，并不代表它能稳定产生可执行、可解析、可授权的结构化动作。

Transformers 的 `GenerationConfig` 把这些选择从“调用时随手传参”提升为可保存的生成配置：输出长度、`max_new_tokens`、`stop_strings`、采样或 beam、cache 实现、logit processor、repetition penalty、token healing、watermarking 和相关默认值都应进入同一个配置对象，并可通过 `generation_config.json` 保存和复用 [76]。第 4 章中的最小 GPT 虽然可以只写一个短 `generate` 函数，但发布报告不能只保存 prompt 和输出文本；它还要保存模型仓库默认 generation config、运行时覆盖字段、随机种子、停止字符串、cache 配置和验证结果。否则同一个 checkpoint 在 notebook、pipeline 和服务框架里可能使用不同默认值，评测差异会被误认为模型能力差异。

图 4.1 把 GPT 的训练循环和生成循环放在同一张图中。训练时 label shift、loss mask 和 optimizer state 定义学习目标；生成时 logits processor、cache、停止规则和校验器定义用户可见行为。

解码控制应有确定顺序。常见路径是先取得最后位置 logits，再应用 logit bias 或禁用

token，接着执行重复惩罚等 logit processor，然后除以 temperature，再做 top-k 或 top-p 截断，最后采样或取 argmax。改变顺序会改变概率分布，所以实验报告应把顺序写清楚。Stop 规则也要区分 token 级停止和字符串级停止；后者可能在 detokenization 后才发现分隔符，容易被空格、多语言标点或半个 token 边界影响。

温度不是孤立的“创造力旋钮”。低温会放大模型最高概率偏好，可能减少事实错误，也可能把错误模式固定下来；高温会增加多样性，也会增加尾部低质量 token。Top-k 使用固定候选数，忽略不同 step 的不确定性；top-p 使用累计概率 nucleus，更自适应，但依赖模型校准。若要比较解码策略，应固定 prompt、最大长度、停止规则、随机种子集合和评价指标，并报告失败样例，而不是只挑一个最好看的输出。

教学版 `sample.py` 通常只暴露最关键的生成契约：从 checkpoint 或 GPT-2 预训练权重初始化，按 `meta.pkl` 选择字符级编码或退回 GPT-2 tokenizer，固定 seed、device 和 dtype，接受字符串 prompt 或 `FILE:prompt.txt`，再用 `max_new_tokens`、temperature 和 top-k 调用 `generate`。这类脚本适合做烟测：同一 prompt 在固定 seed 下是否可复现，EOS 或最大长度是否能停止，checkpoint resume 后是否仍能输出合法文本。它不是服务端基准，因为没有持续 batching、取消请求、KV cache 分页和结构化输出修复。

4.4 深入展开：GPT 是统一接口

本章把 GPT 类模型解释为一种极简但强大的接口：只保留因果 decoder，让所有任务都变成“给定前缀，预测下一个 token”。翻译、问答、摘要、代码补全、函数调用、对话和推理轨迹都可以被写成序列。这个统一接口极大降低了任务工程复杂度，也使模型能从不同任务的序列模式中迁移。

GPT-2 把这个接口落实为“网页文本中的潜在多任务学习”：翻译对、摘要、问答和阅读理解片段不是人工标成任务，而是作为自然文本出现 [205]。GPT-3 进一步把评测协议写清楚：zero-shot、one-shot 和 few-shot 是推理时的上下文条件，不是微调。报告 few-shot 结果时，应说明任务描述、分隔符、示例数量 K、示例采样规则、上下文窗口预算、答案抽取规则，以及选择题打分是否使用原始 likelihood、长度归一化 likelihood 或其他校准方式 [14]。

但统一接口也把很多责任推给数据格式。系统消息和用户消息要有明确边界；assistant 的回答要有终止符；工具调用要能和自然语言区分；多轮对话要避免角色混淆；多模态输入要有图像、音频或视频占位符；结构化输出要能被解析器验证。本章强调，chat template 不是 UI 文案，而是训练和推理共同依赖的协议。训练循环看似简单，实际上有许多不变量。标签必须右移，loss mask 必须只覆盖应预测的 token，gradient accumulation 必须正确缩放，clip 应在 unscale 之后执行，scheduler 应和 optimizer step 对齐，checkpoint 应包

含 optimizer 和随机状态。

结构化输出还需要“反向模板”证据。最新 Transformers response parsing 文档把工具调用、reasoning 字段和最终回答视为需要解析回标准 message dict 的模型输出；支持该能力的 tokenizer 可以用 `parse_response()` 把解码文本解析成包含 `role`、`content`、`thinking` 或 `tool_calls` 的结构，底层由 `response_schema`、正则和 JSON parser 描述 [97]。这改变了第 4 章对结构化输出的验收口径：不仅要验证模型能吐出一段像 JSON 的字符串，还要验证 tokenizer 是否声明了 response schema、schema 是否随 tokenizer 保存、unsupported tokenizer 是否明确失败、解析后的工具名和参数是否匹配授权 schema，以及解析结果能否安全追加回下一轮 chat history。

nanoGPT 风格实现把这些约定压缩在很少代码里 [133]。一个最小 GPT block 通常是 pre-norm 残差结构：先 LayerNorm，再自注意力；再 LayerNorm，再 4 倍宽度 GELU MLP。注意力层把 Q/K/V 融合在一个 `c_attn` 投影里，再 split 成多头张量。若 PyTorch 提供 `scaled_dot_product_attention` 且可用 causal fast path，可以直接传 `is_causal=True`；否则要显式注册下三角 mask，避免当前位置看到未来 token。

模型侧还有几个容易漏掉的契约。它使用 learned token embedding 和 learned absolute position embedding，把 token embedding 与 LM head 权重绑定；训练时有 targets 就返回所有位置 logits，推理时可以只对最后一个位置做 LM head 以省计算；`crop_block_size` 必须同时裁剪位置 embedding 和注意力 mask。导入 GPT-2 checkpoint 时，配置要回到原始 `50257` 词表和 `1024` 上下文，并把 Conv1D 风格的投影权重转置后赋值；而从零训练的小模型常把词表 padding 到 `50304` 来改善硬件对齐。

从 logits 到文本的最后一步同样是建模。贪心解码确定但容易僵硬；温度提高多样性但增加错误；top-p 截断尾部但会改变罕见事实的出现概率；重复惩罚可能减少循环，也可能破坏代码或数学符号；stop sequence 如果按文本而非 token 处理，可能截断过早或过晚。服务级生成循环必须把采样、停止、流式输出、结构验证和取消机制放在同一个状态机里。

重复惩罚是解码期 logit 变换，不是训练目标。CTRL 式 penalized sampling 会降低已生成前缀中出现过的 token 分数；惩罚系数为 1 时不改变 logits，系数更大时更抑制复用 [135]。它能缓解贪心或低温解码中的循环，但也会误伤变量名、引号、列表符号、诗歌韵脚、法律条款和代码缩进等合法重复。生成报告应写明精确变换、是否把 prompt token 也算作已重复、它与 temperature、top-k、top-p、logit bias 的执行顺序，以及哪些样例显示它抑制了有效重复。

4.5 章节细节

4.5.1 Decoder-only 契约

GPT 类模型保留因果 decoder，把任务统一成前缀条件下的下一个 token 预测。它没有任务专用输出头，所有任务边界都通过序列化格式表达。这个契约简单、可扩展，但也让模板和特殊 token 变得极其重要。

4.5.2 因果语言建模

训练时，模型看到前缀 token 并预测当前位置 token。标签右移、loss mask 和 padding 处理必须正确；否则模型可能学会预测当前 token 或填充 token。因果语言建模的简洁性掩盖了大量实现细节。

4.5.3 最小 GPT Block 栈

一个最小 GPT 包括 token embedding、位置机制、若干 decoder block、最终归一化和输出头。每个 block 通常由 attention、MLP、残差和归一化组成。实现时应确认参数初始化、权重 tying、dropout、dtype 和 checkpoint 保存都与训练目标一致。

nanoGPT 还提醒读者，初始化也是契约的一部分。普通线性层和 embedding 可以用固定标准差初始化，但残差投影常按层数用 `0.02/sqrt(2L)` 这类尺度缩小；否则层数增加时残差流方差可能增长过快。参数统计也要区分 position embedding、tied token embedding 和输出头，否则“非 embedding 参数量”会被重复或漏算。

导入和恢复训练也属于最小实现的一部分。GPT-2 家族权重里若某些矩阵来自 Conv1D 包装，形状语义和普通 `nn.Linear` 相反，复制前必须转置；从 `torch.compile` 模型保存的 checkpoint 恢复时，也可能需要去掉参数名前的 `_orig_mod.`。checkpoint 不应只保存权重，还要保存 optimizer、当前 iteration、best validation loss、模型参数和数据配置，这样才能解释恢复后 loss 是否连续。

4.5.4 Batching、Packing 与 Loss 曲线

Batching 决定吞吐，packing 决定 token 利用率，loss 曲线决定训练健康。全局平均 loss 往往掩盖不同数据源、长度和语言上的差异。应按切片观察验证 loss，并用生成样例检查模型是否只是学会了格式。

packed stream 训练特别容易隐藏边界问题。OpenWebText 这类数据会把文档连续写入 token 流，并用 end-of-text 作为软边界；训练样本可能跨文档，因此报告应说明是否允

许跨文档 attention、是否在边界处 reset position、以及 validation split 是否按文档而不是按 token 随机切分。若为了 batch 对齐引入 padding，必须确认 label mask 忽略 padding；若完全用连续 memmap，则仍要确认 EOS、截断和最长上下文的行为。

4.5.5 从 Logits 到文本

Logits 经过温度、top-k、top-p、重复惩罚和停止规则才变成文本。采样策略直接影响创造性、事实性、循环、长度和安全性。结构化输出还需要解析、schema 检查、失败重试和工具权限控制。

4.5.6 高效注意力与生成

FlashAttention、KV cache、GQA/MQA 和连续 batching 让大模型生成可服务化。训练和推理的计算形态不同：训练并行处理序列，推理逐步扩展前缀。理解这种差别是分析延迟、显存和吞吐的基础。教学版 generate loop 可以故意不实现 KV cache：它把不断增长的上下文裁剪到最后 `block_size` 个 token，完整 forward 一次，只取最后位置 logits，按 temperature 缩放，再做 top-k 截断和 multinomial sample。这种代码适合验证模型行为，但不是服务性能基准；它每一步都重新计算前缀，采样顺序本身也必须写入实验报告。

服务级生成通常分成 prefill 和 decode。Prefill 把 prompt 当作完整序列处理，为每层建立 key/value 状态；decode 每次只处理新 token，并复用已有 KV cache。对 dense MHA decoder，KV cache 的近似字节数为

$$2BLTHd_h s,$$

其中 B 是活动 batch，L 是层数，T 是当前上下文长度，H 是 KV head 数，d_h 是 head 维度，s 是每个元素字节数。GQA/MQA 通过减少 KV head 数降低 cache，而不是免费提高模型质量。长上下文服务中，KV cache 容量、带宽、分页策略和取消请求后的释放逻辑，常常比权重大小更先成为瓶颈。

FlashAttention 改善的是精确 attention 的内存访问方式，而不是把 softmax attention 换成近似公式 [28]。它通过分块和 IO-aware 计算避免显式物化巨大注意力矩阵，因此在合适 mask 和精度下应保持同一数学操作。这个区分很重要：有些高效注意力方法改变语义，有些只是更好地执行相同语义。报告推理加速时，应说明使用的是训练路径、prefill 路径还是 decode 路径，因为三者的瓶颈完全不同。

4.5.7 评测提醒

GPT 能生成流畅文本不等于它可靠。评测应区分格式遵循、事实正确、推理正确、代码可执行、安全边界和长上下文利用。尤其在开放式任务中，主观偏好分数不能替代可复现的错误分析。

评测还应区分 teacher-forced loss、生成样本质量和任务分数。一个模型可能 validation loss 更低，但在某个 temperature 下生成更差；也可能 benchmark 分数更高，却因为污染、答案抽取规则或 prompt 模板差异而无法部署。随机解码下单样本估计很弱，pass@k 或 self-consistency 又消耗更多推理预算。报告应把样本数、随机种子、解码参数、metric、答案归一化和成本一起写出。

4.5.8 Chat 模型接口契约

Chat model 不是只在 base decoder 后面加一点指令数据。它是 tokenizer、chat template、loss mask、停止符、工具 schema 和运行时策略共同定义的接口。若训练时使用 `<assistant>` 作为回答起点，服务时却换成另一套 role marker，模型可能不是能力下降，而是接口不匹配。若训练标签没有覆盖 end-of-turn token，模型可能学不会在正确位置停止。

因此，chat template 应像 API 一样做版本管理和回归测试。测试样例应覆盖空 system message、多轮对话、工具观察、结构化输出、拒答、截断、EOS 和 stop sequence。每个样例都应能打印 token ids、decoded text、label mask 和最终 stop 行为。这个检查属于第 4 章的 GPT 契约，也会在后面的 SFT、偏好学习、RAG、工具调用和多模态章节反复出现。

4.5.9 发布前验收矩阵

GPT 发布前验收应把训练目标、生成接口和服务状态放在同一张表里，而不是分别展示 loss、样例和延迟。Hugging Face 的 chat template 文档把 `add_generation_prompt`、`continue_final_message`、special token 是否重复插入等细节明确为 tokenizer 行为 [63]；KV cache 文档又把 dynamic、static、quantized、offloaded、sliding-window 等 cache strategy 写成可选运行时策略 [62]。这说明第 4 章的 GPT 契约已经超出“模型前向”本身：同一个 checkpoint 在不同模板、停止规则、cache 实现和结构化输出修复循环下，可能呈现不同能力、成本和失败模式。

因此，出版级报告应至少记录下面六类证据。

表 4.1 把 GPT 发布证据放在同一张验收矩阵中，防止把接口问题误读成能力问题。

这张矩阵的作用不是替代后续服务章节，而是防止读者把 GPT 写成孤立模型。第 4 章只要能在小模型上打印这些字段，后面扩展到 LLaMA 类结构、SFT、偏好学习、RAG

验收项	必须固定的字段	常见回归
Tokenizer 与模板	special token、chat template、generation prompt、prefill 字段	重复 BOS/EOS，assistant 边界错位
Label mask	assistant span、工具观察、padding、EOS	训练 loss 下降但不会停止
采样策略	温度、top-p、top-k、logit 处理器、seed	同一结果不可复现或合法重复被惩罚
Cache 策略	dynamic、static、quantized、offloaded、sliding window、释放规则	长上下文显存失控或 decode 漂移
结构化输出	schema、增量解析、修复循环、权限检查	JSON 可解析但越权调用工具
评测复现	prompt、样本数、答案抽取、成本、失败样例	pass@k 掩盖单样本质量和预算差异

表 4.1: GPT 发布验收项、必须固定字段与常见回归检查表。

或多模态时，就能保留同一套证据语言。

发布证据。 把 GPT 作为可发布系统时，不能只给出一条 loss 曲线或一组聊天样例。最小证据包应同时包含模型配置、tokenizer 和 special-token 表、训练/验证数据切分、packing 与文档边界策略、label mask 规则、checkpoint 恢复字段、采样参数、stop 规则和 chat template 版本。若这些字段没有绑定在同一个 run card 中，读者无法判断同一个 checkpoint 的训练损失、推理样例和服务行为是否来自同一套契约。尤其是从零训练的小模型和导入 GPT-2 权重的模型，词表大小、position embedding、权重转置、padding 到硬件倍数和真实 tokenizer id 都可能不同；报告必须说明哪些差异是训练优化，哪些差异会改变模型语义。

故障定位。 GPT 调试应先排除接口错误，再解释能力问题。若 loss 很快下降但生成复制输入，优先检查标签是否右移；若验证 loss 异常好，检查训练/验证去重、文档切分和 prompt 泄漏；若模型不会停止，检查 EOS 是否参与训练、stop string 是否跨 token、以及服务端是否在 detokenization 之后才截断；若 few-shot 分数不稳定，检查示例顺序、分隔符、答案抽取和长度归一化；若结构化输出偶发不可解析，检查 schema token、修复循环和工具权限，而不是只提高 temperature 或更换 prompt。出版级教材应把这些失败模式写成可复现诊断，而不是把它们归为“模型太小”或“数据不够”。

从本章到后续章节。 第 4 章的核心不是 GPT 这个名字，而是一组后续章节都会复用的接口不变量：从左到右目标、序列化边界、mask、checkpoint、采样、停止和运行时状态。

第 5 章会把同一契约换成 LLaMA 类默认结构；第 6、7 章会把 token/step、checkpoint 和稳定性放进训练系统；第 8 章会把 prefill、decode 和 KV cache 放进服务系统；第 9 章以后则把 chat template、loss mask、偏好数据、工具调用和安全边界写成后训练协议。读者若能在一个小 GPT 上打印并验证这些字段，后续面对更大的模型名时就不会把系统接口误认为纯粹的参数规模。

4.6 关键术语、实现要点与练习

关键术语。 Decoder-only model 只用因果 decoder 建模前缀到下一个 token；causal language modeling 只根据更早 token 预测当前位置 token；teacher forcing 在训练时为所有位置提供真实前缀；in-context learning 在推理 prompt 中用指令和示例指定任务，而不更新权重；weight tying 共享 token embedding 和输出投影；tokens per iteration 描述一次 optimizer step 消耗的有效 token；padded vocab 把词表大小对齐到硬件友好的倍数；gradient accumulation 用多个 micro-batch 模拟更大 batch；KV cache 保存历史 key/value 以避免重复计算前缀；cache implementation 表示 dynamic、static、quantized、offloaded 或 sliding-window 等缓存策略；FlashAttention 是精确 attention 的 IO-aware 实现；chat template 定义系统、用户、助手和工具消息的序列化方式；generation prompt 标记 assistant 回复应开始的位置；GenerationConfig 是保存长度、停止、采样、cache 和 logit 处理默认值的生成配置；response schema 定义如何把模型输出解析回结构化 message；`parse_response()` 是 tokenizer 级结构化响应解析接口；sampling policy 包括 temperature、top-p、top-k、stop sequence 和 repetition penalty；structured output 要求模型输出满足 schema。

实现要点。最小训练循环必须验证右移标签、loss mask、gradient accumulation、checkpoint、tokenizer/meta 文件和采样顺序；memmap 数据集应说明 `x/y` 切片、EOS、padding、截断和 validation split；GPT-2 权重导入要区分 padded vocab 与真实词表，并处理 Conv1D 权重转置；checkpoint resume 要保存 optimizer、iteration、best validation loss 和模型配置；困惑度报告必须说明 tokenizer、验证集、文档边界和上下文长度；服务级生成循环应区分 prefill、decode、KV cache、stop 状态和流式输出；生成调用应保存 `generation_config.json`、运行时覆盖、停止字符串、cache 配置、随机种子和配置校验结果；结构化输出需要增量解析、response schema、`parse_response()`、schema 验证和失败恢复；chat template 必须和 tokenizer、loss mask、EOS、工具 schema 一起版本化。

练习。

1. 把一个三轮对话序列化成 chat template，并标出哪些 token 计算 loss。

2. 比较 greedy、temperature sampling 和 top-p sampling 的失败模式。

3. 为同一分类任务设计 zero-shot、one-shot 和 few-shot prompt，并写出分隔符、示例采样、答案抽取和 likelihood 归一化规则。

4. 在 generation loop 中加入 repetition penalty，说明 prompt token 是否参与重复统计、它与 temperature/top-p 的执行顺序，以及一个合法重复被误伤的例子。

5. 设计一个 JSON 输出 schema，并说明模型输出不合法时系统应如何处理。

6. 写出一个最小 generation loop 的状态字段：token ids、stop 条件、采样参数、stream handle 和取消标志。

7. 用 $B = 8$、$L = 24$、$T = 4096$、$H = 16$、$d_h = 64$、BF16 估算 KV cache 字节数，并说明 GQA 把 KV head 减到 4 时如何变化。

8. 构造一个 `[10,20,30,40]` 的右移标签单元测试，写出输入、标签、ignore 位置和预期 logits 形状。

9. 给定 `block_size=1024`、`batch_size=12`、`gradient_accumulation_steps=40` 和 8 个 DDP 进程，计算一次 optimizer step 的 token 数，并说明每个进程应如何缩放 loss。

10. 设计一个 `train.bin`/`val.bin` memmap 读取测试，验证随机 offset、`x/y` 右移、文档 EOS 和 padding mask 是否符合预期。

11. 从 GPT-2 checkpoint 导入一个最小模型时，列出需要固定的配置项、需要转置的权重类型，以及为什么从零训练时可以使用 `50304` 而导入时不能替代 `50257`。

12. 用 `FILE:prompt.txt`、固定 seed、temperature 和 top-k 写一个 generation smoke test，检查最大长度、EOS、重复惩罚和 resume checkpoint 后输出格式是否稳定。

13. 为 GPT 发布前验收写一张表，覆盖 chat template、label mask、generation config、cache implementation、结构化输出和评测复现字段。

14. 为一个支持工具调用的 chat model 写 response parsing smoke test，要求覆盖 raw decoded output、`parse_response()`、`response_schema`、tool name、arguments JSON、unsupported tokenizer 失败路径和追加回 chat history 的格式。

15. 比较模型仓库默认 `generation_config.json` 与运行时覆盖参数，列出哪些字段会改变长度、停止、采样、cache、logit 处理、watermark 或 token healing，并说明如何防止 notebook 与服务端默认值漂移。

4.7 结构化检查表

4.7.1 生成策略对照

策略	适用场景	风险
Greedy	需要确定性、低成本输出	僵硬、可能陷入局部高概率错误
Temperature	创意写作、多样回答	事实性和格式稳定性下降
Top-p	控制长尾采样	截断可能丢掉罕见正确项
Self-consistency	数学、推理、多路径任务	成本上升, pass@k 掩盖 pass@1
Verifier selection	代码、数学、可检查任务	验证器漏洞和选择偏差
Structured decoding	JSON、SQL、工具调用	解析失败、schema 漏洞、修复循环

第五章　LLaMA 类架构

5.1　现代 decoder 默认配置

LLaMA 类模型通常使用 pre-norm、RMSNorm、RoPE、SwiGLU、无 bias 线性层、grouped-query attention、长上下文和更复杂的数据混合 [230, 231, 43]。这些选择的价值不是单独存在的，而是在训练稳定性、推理成本和后训练兼容性之间形成组合。

一个 LLaMA 类 decoder block 可以先写成残差流上的两个更新：

$$y^{(\ell)} = x^{(\ell)} + \mathrm{Attn}(\mathrm{RMSNorm}(x^{(\ell)})), \tag{5.1}$$

$$x^{(\ell+1)} = y^{(\ell)} + \mathrm{SwiGLU}(\mathrm{RMSNorm}(y^{(\ell)})). \tag{5.2}$$

它仍然是第 3 章的 residual-stream 结构，只是默认组件发生了变化。RMSNorm 改变归一化成本和数值行为；RoPE 改变位置信息进入 attention 的位置；SwiGLU 改变 MLP 表达能力和参数分配；GQA 改变 KV cache 预算。训练和服务十亿级参数时，这些差异不是装饰。

RoPE 把位置信息写进 query/key 的旋转结构，使相对位置关系更自然地进入注意力分数。RMSNorm 用均方根尺度归一化隐藏状态，减少均值项处理并常用于大规模 decoder。SwiGLU 通过门控提升前馈层表达能力。GQA/MQA 减少 KV cache，但也改变注意力头共享方式。教材应该把这些组件解释为服务于稳定性、长上下文和推理成本的组合，而不是模型报告中的装饰词。

出版级架构描述还应把 LLaMA 类模型看成一组接口契约。层数、隐藏维、head 数、KV head 数、FFN 宽度、RoPE base、norm ϵ、词表大小、special token、chat template 和最大上下文长度，都会进入 checkpoint shape 或推理语义。只说“LLaMA-like”不能说明模型能否被正确加载、能否复用服务 kernel、能否安全地扩展上下文，也不能说明某个能力提升来自结构、数据还是后训练。

一个更可复现的写法是给出配置向量

$$C_{\mathrm{llama}} = (L, d, H_q, H_{\mathrm{kv}}, d_h, m, V, T_{\max}, \theta_{\mathrm{rope}}, \epsilon_{\mathrm{norm}}, P_{\mathrm{cache}}),$$

其中 L 是层数，m 是 SwiGLU 或专家 FFN 的中间宽度，V 是真实词表而不是 padded vocab，P_{cache} 描述 cache 表示、滑动窗口和释放策略。这个向量仍然不能替代完整 checkpoint card，但它能把架构、tokenizer 和服务路径放在同一张表里。若模型报告只列出总参数和上下文长度，读者就无法复现 attention head 分组、MLP 宽度取整、RoPE scaling 或 KV cache 成本。

本地 lit-llama 风格实现把这些字段写成了代码接口。配置里有 block size、真实词表、padded vocab、层数、head 数和 hidden size；若未显式给出 padded vocab，会把真实词表向 64 的倍数取整。Embedding 和 LM head 都按 padded vocab 建表，这能让矩阵 kernel 更整齐，但采样和损失仍必须屏蔽 padding 行。另一个轻量 HF 配置还显式记录 attention bias 关闭、SiLU 激活、KV head 数、RMSNorm ϵ、RoPE base、BOS/EOS id、输入输出 embedding 不绑定，以及启用 cache。这些字段不是实现噪声；它们共同决定 checkpoint shape、logit 数值、是否能复用缓存解码，以及 tokenizer 与模型权重是否同一接口。

官方 Transformers 模型文档把这类字段进一步公开成配置合同。LlamaConfig 不只列模型名，而是列出词表、隐藏维、FFN 中间维、层数、attention head、KV head、激活函数、最大位置、RMSNorm ϵ、BOS/EOS、embedding tying、RoPE 参数、attention bias、MLP bias 和 head dimension 等字段 [85]。其中 KV head 的语义尤其重要：KV head 等于 query head 是 MHA，等于 1 是 MQA，介于两者之间才是 GQA。把一个 checkpoint 转成 GQA 时，文档还要求按组对原 K/V head 做均值池化，而不是任意丢弃或复制。也就是说，`num_key_value_heads` 不是内存优化标志，而是权重形状、head 分组和质量风险的共同定义。

Qwen3Config 和 Gemma3TextConfig 说明，现代 LLaMA 类报告不能只复述 LLaMA 字段。Qwen3 显式给出 head dimension、滑动窗口开关、窗口大小、最大窗口层数和 layer types，用来区分哪些层采用全局 attention、哪些层采用局部 attention [95]。Gemma3TextConfig 又暴露更大的词表、不同的 head/KV-head 组合、embedding tying、RoPE 参数、局部 RoPE base、滑动窗口模式、hybrid cache、attention/logit softcapping 和多模态占位 token [75]。这些差异说明，“decoder-only”只是大类；出版级架构表应保存原始 config JSON、推导出的 head 维度、每层 attention 类型、embedding 是否共享、RoPE scaling 或 RoPE 参数、cache 实现和特殊 token 索引。否则模型能加载并生成文本，也不代表架构比较、参数量、KV cache 成本或多模态占位语义是可复现的。

Llama4 文档把这种趋势推到更完整的接口层：模型配置同时包含文本配置、视觉配置、图像起止 token 和图像占位 token；文本配置又把 KV head、MoE 每 token 专家数、本地专家数、MoE 层位置、router logits、辅助 router loss、QK norm、无 RoPE 层、分块注意力大小和 layer types 写成显式字段 [86]。这说明 LLaMA-like 已经不能只被理解为“RMSNorm+RoPE+SwiGLU+GQA”。对于这类 checkpoint，架构审计应同时记录 attention

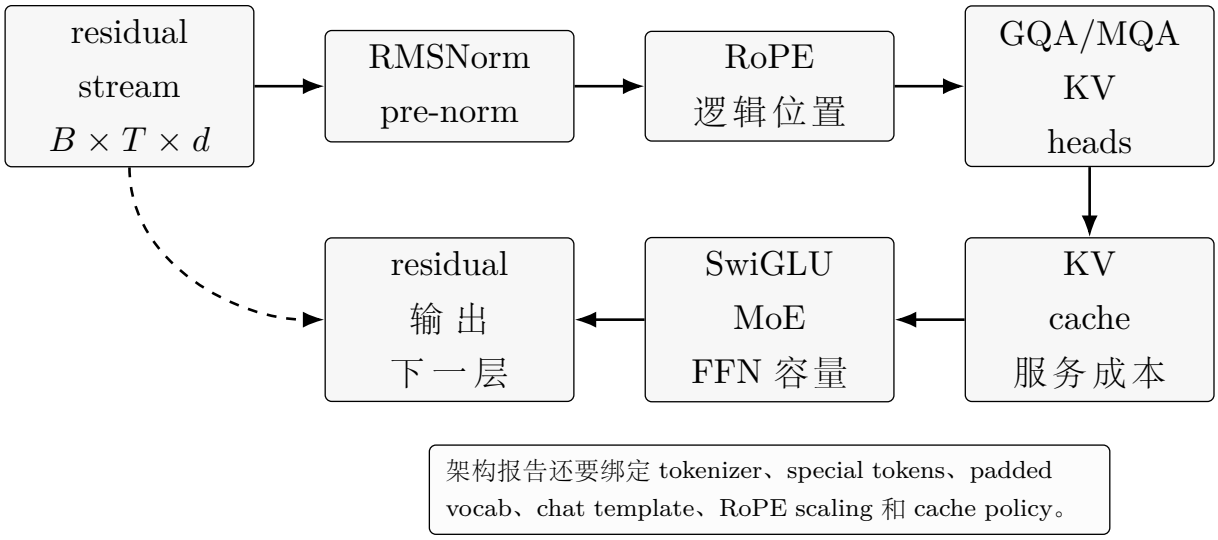

图 5.1: LLaMA 类架构是一组可加载、可训练、可服务的契约。该图是对 LLaMA、Llama 3、DeepSeek-V3 和 Kimi K2 类报告的原创综合 [230, 43, 31, 137]。

backend、chunk size、router 噪声或辅助损失、图像 token 索引、视觉预处理器以及长上下文示例依赖的 kernel 条件；否则同一个模型在不同 backend、tensor parallel、图像预处理或 cache 策略下，可能有不同的可运行长度、显存曲线和多模态 token 对齐语义。

图 5.1 把 LLaMA 类 decoder block 画成“结构、缓存、服务”三者的共同契约。它突出 RMSNorm、RoPE、GQA/KV cache、SwiGLU/MoE FFN 与 tokenizer/template 之间的关系，而不是只把这些词作为模型家族标签。

RMSNorm 对隐藏向量 $h \in \mathbb{R}^d$ 做尺度归一化：

$$\mathrm{RMSNorm}(h) = g \odot \frac{h}{\sqrt{\frac{1}{d}\sum_{j=1}^{d} h_j^2 + \epsilon}}.$$

它不显式减均值，常见实现也不使用 bias。节省的单次计算不大，但每个 token、每一层、每次生成都会重复执行。更重要的是，它和 pre-norm 一起让子层在受控尺度上工作，同时让残差流继续累积信息。混合精度实现中，平方均值通常应在足够稳定的 dtype 中累积；ϵ、cast 顺序和 scale 应用位置若与 checkpoint 不一致，长序列生成中会放大小的 logit 差异。

Pre-norm 的工程意义在于每个子层看到稳定尺度，而 residual stream 可以继续承载未归一化的累积状态。最后接 LM head 前的 RMSNorm 也不能省略或随意移动，因为 logits 的尺度会影响采样温度、logit bias、重复惩罚和结构化输出约束。迁移 checkpoint、量化或写 fused RMSNorm kernel 时，应对齐原模型的 dtype、ϵ、是否有 bias、scale 参数形状以及 final norm 位置。

RMSNorm 还有一个容易被教材略过的性质：对正尺度缩放近似不变。若 $s > 0$，则

$sx/\mathrm{RMS}(sx) = x/\mathrm{RMS}(x)$。这说明 RMSNorm 主要约束方向和相对特征幅度，而不是像 LayerNorm 那样显式移除均值。反向传播中，输入缩放会改变梯度尺度；混合精度训练、残差累积和梯度裁剪都会放大这种差异。实现上应写清平方均值的累积 dtype、ϵ 加在平方均值内还是根号后、scale 参数是否参与权重衰减，以及量化前后是否使用同一个 fused kernel。

SwiGLU 前馈层可以写成

$$\mathrm{FFN}(h) = W_o(\mathrm{SiLU}(W_g h) \odot W_u h).$$

它有 gate、up、down 三个主要矩阵。若隐藏宽度为 m，忽略 bias 后参数量约为 $3dm$；标准 $4d$ GELU MLP 约为 $8d^2$。因此许多 LLaMA 类模型把 m 设在接近 $(8/3)d$ 的位置，再向硬件友好的倍数取整。取整会改变参数量、激活内存和 checkpoint 大小，报告中不应只写“使用 SwiGLU”。

FFN 往往是 dense decoder 中最大的参数和计算来源，所以 SwiGLU 的宽度不是无关紧要的实现细节。若为了 tensor core 对齐把 m 向上取整，参数量、激活 buffer、optimizer state 和 checkpoint 体积都会一起变化；若换成 MoE FFN，还要同时报告总专家参数和每 token 激活参数。架构表应给出每层 attention 参数、FFN 参数、embedding/output head 参数，而不是只给一个模型总参数。

从功能上看，SwiGLU 是 token-wise 的可学习特征选择。Attention 负责让不同 token 交换信息；FFN 则在每个 token 的隐藏向量内部做非线性变换。SwiGLU 的 gate 矩阵不是普通激活函数替换，而是让模型按 token 和 feature 选择哪些候选特征通过。与 MoE 的 router 相比，SwiGLU gate 在每个 token 内连续地选择 feature；MoE router 则在专家集合上做离散或稀疏选择。出版稿把二者都称为“门控”时，应说明门控对象不同：一个是 feature 维度，一个是 expert 路径。

YaRN 类 RoPE 扩展进一步说明，长上下文不是简单把最大长度调大。它从频率分段角度处理 RoPE：高频维度保留原始旋转以保护短距离位置建模，低频维度做内插以扩展长距离范围，中间频段平滑混合，并通过 attention score rescale 避免长序列下注意力过度发散 [191]。因此，报告长上下文模型时应说明原训练长度、目标长度、缩放因子、频段分割规则、推理时是静态还是动态 scaling，以及短上下文能力是否回归。

PI 和 NTK-aware RoPE 也不能被理解成严格的 train-free 外推。PI 把位置 m 替换成 m/s，使新位置落回更熟悉的相位范围，但它会同时压缩高频和低频维度，可能破坏短距离位置模式。NTK-aware scaling 通过放大 RoPE base，让内插强度从高频到低频逐渐增大，比统一缩放更保护短距离相位，但仍然改变了 query/key 分布。因此，长上下文报告若采用 PI、NTK-aware RoPE 或类似规则，通常应把继续训练或适配训练写入配方，除非给出强 no-training 证据并同时覆盖短上下文和长上下文切片。

Baichuan 2 提醒读者，位置机制是架构选择，而不是模型家族标签。它的 7B 模型使用

RoPE, 13B 模型使用 ALiBi, 用相关模型同时研究旋转式和 bias 式 attention 路径 [251]。因此, 不能只用“LLaMA-like”概括一个 checkpoint; 位置机制会影响 kernel 兼容性、attention mask、长上下文外推和 KV cache 解码一致性测试。

RoPE 调试应关注 logical position，而不只是 tensor shape。Prefill 阶段通常使用位置 $0, \ldots, T-1$; cached decode 阶段新 token 必须使用前缀之后的下一个逻辑位置。如果实现滑动窗口、prefix cache 复用或多轮工具调用拼接，位置编号、attention mask、cache 写入 slot 和实际可见 token 集合必须保持一致。一个 off-by-one 错误在短 prompt 上可能看不出来，却会在长文档问答或持续对话中造成明显退化。

RoPE 的最小回归测试应同时覆盖三类输入。第一类是无 padding 的短前缀，用来比较 full prefill 与逐 token decode 的 logits; 第二类是 left padding batch，用来验证 position ids 不随物理列号错位; 第三类是超过训练长度或滑动窗口边界的长样本，用来检查远处证据、短上下文回归和 cache 滚动语义。若只用单条短 prompt 测试，很多 RoPE base、动态 scaling、prefix cache 和 input position 的错误都会被隐藏。

代码级 RoPE 还要检查缓存形状。一个常见实现按 `head_dim` 的偶数坐标构造 θ，为所有位置预先生成 cos/sin cache，并在应用时把最后一维 reshape 成二维旋转对。若实现为了模拟低精度复数行为把 cache 转成 fp16，而模型主体是 bf16 或 fp32，就要确认 cast 顺序与原 checkpoint 一致。轻量实现中 `input_pos` 会从 RoPE cache 选择逻辑位置，也会从 mask cache 选择对应行; 如果 cache 已经向左滚动，RoPE 逻辑位置、KV 写入位置和 attention 可见窗口必须一起滚动。只检查张量维度相等，无法证明 cached decoding 与 prefill 等价。

5.2 KV Cache 与注意力变体

推理时, 已生成 token 的 key/value 可以缓存, 避免每步重复计算全部前缀。KV cache 大小随层数、头数、维度、上下文长度和 batch 增长。GQA/MQA 通过共享 key/value 头减少缓存成本，但可能改变质量和长上下文行为 [1]。

对 dense decoder，若层数为 L，batch 为 B，上下文长度为 T，KV head 数为 H_{kv}，head 维度为 d_h，每个 cache 元素占 s bytes，则近似有

$$\mathrm{bytes}_{\mathrm{KV}} = 2BLTH_{\mathrm{kv}}d_h s.$$

其中 2 来自 key 与 value。MHA 中 $H_{\mathrm{kv}} = H_q$，MQA 中 $H_{\mathrm{kv}} = 1$，GQA 则介于两者之间。这个公式解释了为什么 GQA 会成为服务端默认选择：模型权重按副本加载一次，而 KV cache 随活动请求数和上下文长度线性增长。长上下文服务中，cache 容量和带宽常常比权重大小更先成为瓶颈。

表 5.1 以 KV cache 为中心比较注意力变体，使架构选择和服务成本保持同一口径。

注意力变体	KV head 结构	主要收益	主要风险
MHA	$H_{\text{kv}} = H_q$	每个 query head 有独立 KV 表示	decode cache 和带宽最大
MQA	$H_{\text{kv}} = 1$	cache 最小、decode 带宽低	某些规模或任务上质量下降
GQA	$1 < H_{\text{kv}} < H_q$	质量和服务成本折中较好	checkpoint 和 head 分组必须一致
Latent attention	模型特定压缩状态	改变 cache 表示并节省状态	语义和 kernel 更难横向比较

表 5.1: 注意力变体、KV head 结构、主要收益与风险检查表。

DeepSeek-V3 这类模型进一步把注意力缓存问题推向“表示设计”。Multi-head Latent Attention 不是简单减少 KV head 数，而是把一部分 key/value 状态压缩进低秩 latent，再在注意力计算中恢复所需表示 [31]。这类设计必须说明 latent 维度、RoPE 作用在哪些分量、缓存的是 latent 还是重构后的 key/value、是否依赖专门 kernel。否则读者无法判断它节省的是 cache 字节、内存带宽，还是只是在某种请求形态下改变了常数。

MLA 实现还要明确缓存对象。若缓存重构后的多头 key/value，大部分显存收益就消失；服务路径通常应缓存压缩后的 KV latent 和解耦出来的 RoPE key，再在注意力计算时重构或吸收剩余投影。RoPE 分离不是形式细节：如果位置敏感的 key 分量留在低秩重构内部，decode 每一步都可能要对整个 prefix 重新计算旋转后的 key。训练后把 key-up 或 value-up 矩阵吸收到 query/output 路径，也属于推理契约，而不只是线性代数化简。长上下文并不只增加注意力计算，它还增加服务系统中的状态。一个请求生成到第 10,000 个 token 时，服务器需要保留大量 KV block；如果用户取消、超时或切换工具，系统还要释放或复用这些 block。架构选择和调度策略在这里相互耦合：同一个模型结构，在不同 batch、不同上下文长度和不同请求混合下，成本曲线可能完全不同。

用缓存元素数看，MLA 的关键不是“少一个 head”，而是把普通的两份 K/V 表示换成 latent 加少量位置敏感 key。若上下文长度为 l，KV latent 维度为 d_c，解耦 RoPE key 维度为 d_h^R，则每层每 token 近似缓存 $d_c + d_h^R$ 个元素，而普通 MHA 近似缓存 $2H_q d_h$ 个元素。这个数字只有在服务路径真的缓存 latent 和 decoupled RoPE key 时才成立。若为了实现方便先 up-project 成 full K/V 再缓存，报告中的节省就只是纸面节省。矩阵吸收也要写入 run card：哪些 W^{UK}、W^{UV}、query-up 或 output projection 在训练后被折叠，折叠前后是否做数值等价测试，以及 tensor parallel 如何切分 latent 与 head 维度。

GQA 和 MLA 的比较也不能只看 cache 字节。GQA 主要改变 KV head 数，通常还能沿用普通 attention 语义；MLA 改变 cache 表示和投影吸收路径，可能需要模型特定 kernel。发布报告应分别给出 prefill 延迟、decode TPOT、cache bytes/token、最大可承

载 active tokens、短 prompt 开销、长文档检索准确率和 kernel 依赖。若一种方法只在长 prompt 上节省显存，却让短请求增加明显常数开销，服务端可能需要按流量分布选择，而不是把它写成无条件更优。

生成时的 cache 还要区分“模型结构要求”和“运行时策略”。Transformers KV cache 指南把 DynamicCache、StaticCache、Quantized Cache、offloaded cache 等实现分开，并指出对 sliding-window 或 chunked attention 层，cache 到达窗口或 chunk 大小后就不再按完整上下文增长 [62]。因此，一个长上下文系统的 run card 至少应写清：模型层本身是 MHA、GQA、MLA、sliding-window 还是 hybrid；运行时使用动态、静态、量化、offloaded 或 hybrid cache；哪些层受窗口限制；cache 是否支持编译、offload 和断点恢复；取消请求后是否释放 block。若只报告“启用 KV cache”，读者无法判断延迟收益来自架构、cache 类、滑动窗口上限、offload，还是调度器减少了 active tokens。

cache 选择还应写成验收矩阵，而不是一个布尔开关。DynamicCache 适合通用生成，但增长模式会跟请求长度绑定；StaticCache 预分配固定上限，便于编译和稳定形状，却可能在短请求上浪费 attention 计算；offloaded cache 用主机内存换 GPU 显存，收益取决于 PCIe/NVLink 带宽、prefetch 和 batch 形态；Quantized Cache 降低存储，但量化 backend、位宽、分组轴和残差窗口都会影响长上下文复制、引用和数字任务。对 sliding-window、chunked attention 或 hybrid 层，run card 还要逐层列出 cache 上界，因为某些层不会随完整上下文继续增长。这样读者才能把“架构节省的 KV head”“运行时节省的驻留显存”和“系统调度节省的 active token”分开。

5.3 MoE 与替代序列模型

MoE 通过稀疏激活增加总容量，每个 token 只选择部分专家。它改变了“模型大小”的含义：必须区分总参数、激活参数、路由成本和专家通信。DeepSeek-V3 与 Kimi K2 显示了 MoE 在开放权重前沿系统中的重要性 [31, 137]。

DeepSeek-MoE 类设计还提醒我们，router 是架构的一部分。共享专家、路由专家、top-k 选择、序列级负载均衡、专家并行放置和负载均衡偏置，都会改变哪些 token 更新哪些专家，以及哪些设备成为瓶颈。MoE 报告除了总参数和激活参数，还应给出专家数、每 token 选中专家数、共享专家容量、路由分布、负载均衡策略、token 丢弃或重路由规则，以及 all-to-all 通信成本。

教学实现很容易把 sparse MoE 写成“所有专家都 forward，然后把非 top-k 权重置零”。这能展示数学形式，却没有减少实际计算。可服务实现通常要经历 dispatch、expert compute、combine 三步：先按 top-k 把 token 分发给对应专家，再让每个专家批量处理自己的 token，最后按路由权重把输出组合回原 token 顺序。这个 dispatch-combine 语义会

进入通信图、padding/capacity 策略和反向传播。若报告只说“top-2 MoE”，但不说明是否真正只计算被选专家、是否有容量上限、溢出 token 如何处理，就不能解释训练吞吐或服务延迟。

Top-k 也不是一个可以忽略的可导性细节。实际框架的 top-k 反向传播通常只把梯度传给被选中的门控分数，未选专家在该 token 上没有梯度。长期训练中，这会让热门专家更热门，冷专家更冷。负载均衡损失因此要同时观察 token count 和 routing weight：每个专家收到多少 token、收到的总概率质量是多少、top-k 权重是否过度集中、每层还是全模型计算均衡目标。专家并行下，这些统计直接对应设备负载；均衡损失不是正则化装饰，而是让稀疏容量可训练、可调度的系统约束。

RetNet、Mamba、Mamba-2 和 Titans 等工作探索了注意力之外的长序列机制 [225, 44, 29, 11]。但替代注意力的主张必须在相同数据、tokenizer、训练预算、上下文任务和服务负载下验证。

MoE 报告尤其要避免误导。总参数大不等于每 token 计算大，激活参数小也不等于服务便宜，因为专家路由、all-to-all 通信、负载均衡和冷专家问题都会影响吞吐。替代序列模型同样不能只报告困惑度；它们必须证明自己在检索、聚合、矛盾处理、后训练、工具调用和安全评测中仍然可靠。

5.4 深入展开：LLaMA 类设计的组合意义

本章并不把 LLaMA 类架构写成组件清单，而是解释这些组件为什么经常一起出现。具体说，RMSNorm 简化归一化，RoPE 写入相对位置信息，SwiGLU 提升前馈层表达。GQA 与 MQA 主要降低 KV cache 成本；无 bias 线性层和 pre-norm 则服务于训练稳定性。这些选择单独看都不神秘，组合在一起才形成现代 decoder 默认配方。

RoPE 和长上下文尤其需要谨慎。扩大 context window 不是简单提高一个超参数。位置编码外推、attention score 范围、训练长度分布、KV cache 显存和 RAG 上下文构造都会影响长文档表现。一个模型能接受长输入，并不代表它能稳定利用中间证据、跨章节聚合信息或抵抗干扰文档。

KV cache 是架构和服务交汇处。训练时所有 token 并行计算；生成时每一步只新增一个 token，但要保留历史 key/value。GQA/MQA 通过共享 key/value 头减少缓存，但可能影响模型质量和长上下文行为。服务系统中的 batch、分页缓存、请求取消和多租户隔离，都依赖这些结构选择。

lit-llama 这类独立实现把这个交汇点写成显式 API [153]。通常会预先构造 RoPE cache 和下三角 mask cache；prefill 使用前 T 个位置，cached decode 则用 `input_pos` 选择逻辑位置并写入 KV cache。如果运行时设置了固定 `max_seq_length`，cache 满时可能把 key/value

向左滚动并覆盖最后一格。这不是中性的加速技巧，而是 sliding-window 策略，意味着一部分早期 token 已经不可见，报告中必须说明。手写 LLaMA/GQA demo 时还有几处很容易错。RoPE 的维度是 `head_dim`，不是模型 hidden width；一份 sin/cos cache 通常在所有 head 间共享，并作为不训练的 buffer 保存。RoPE 作用在 query 和 key 上，不作用在 value 上；缓存的 key 应是 decode 时 attention 实际读取的同一种旋转表示。head split 之后，attention score 应除以 head 维度的平方根，而不是模型 hidden width 的平方根。GQA 中用 `repeat_interleave` 展开 KV head 适合教学展示共享关系，但服务端 cache 应只保存 H_{kv} 个 head；如果把缓存里的 K/V 物理扩展回 H_q 个 head，就会抵消大部分显存和带宽收益。

MoE 则改变“模型规模”的语言。一个 MoE 模型可以有巨大的总参数量，但每个 token 只激活部分专家。质量、成本和延迟取决于 router、top-k 专家、负载均衡、专家并行和通信拓扑。因此要求读者看到模型报告中的总参数、激活参数和硬件条件，而不是被单一数字误导。

5.5 章节细节

5.5.1 什么使 Decoder 成为 LLaMA 类

LLaMA 类模型通常组合 pre-norm、RMSNorm、RoPE、SwiGLU、GQA、无 bias 线性层和高质量数据配方。它不是一个单一组件，而是一套现代 decoder 工程默认值。理解这种组合比记住某个模型名更重要。

5.5.2 RMSNorm 与 Pre-Normalization

Pre-norm 在子层前归一化输入，使深层训练更稳定。RMSNorm 使用均方根尺度，不显式减均值，常用于大规模 decoder。归一化选择会影响梯度流、数值稳定和后训练兼容性。

5.5.3 SwiGLU 前馈网络

SwiGLU 用门控结构增强前馈层表达能力。相比普通 ReLU 或 GELU FFN，它通常带来更好的质量-计算权衡。实现时需要注意 hidden dimension、参数量和初始化，因为 FFN 往往是计算和参数的主要来源。

5.5.4 旋转位置嵌入

RoPE 通过旋转 query 和 key 把相对位置信息写入注意力分数。它与长上下文扩展、频率缩放和位置外推密切相关。仅仅把最大长度调大，不等于模型真的学会了长距离检索和聚合。

YaRN 的实践要点是保留高频、内插低频、混合中频，并在更长上下文中调整注意力分数尺度。这个细节提醒读者：长上下文扩展必须同时看位置编码、attention 分布、继续训练数据和短上下文回归。

5.5.5 KV Cache 与注意力变体

推理时缓存历史 key/value 能避免重复计算前缀。GQA 和 MQA 通过共享 key/value 头减少 cache 体积，但可能改变质量和长上下文行为。服务报告应说明 cache 成本、上下文长度、batch 形态和注意力实现。

5.5.6 Tokenizer 与词表契约

LLaMA 类 checkpoint 与 tokenizer、special tokens 和 chat template 绑定。更换 tokenizer 会改变输入 id 和输出空间，通常等同于换了模型接口。多语言和代码任务尤其依赖词表覆盖与切分效率。

词表大小也是系统参数。Embedding 和输出矩阵的规模约为 Vd_{model}，训练时最终 softmax 要在 V 个词表项上归一化，推理时 logit processor 和采样器也要逐步处理这些 logits。实现常把词表 pad 到硬件友好的倍数，但 padding 行不能变成可采样 token。报告 tokenizer 时，应同时说明真实词表大小、padded vocab size、special token id、chat template 和是否有 byte fallback。

Baichuan 2 把多语言 tokenizer 契约写得更具体：词表扩展到 125,696，使用 SentencePiece BPE，不做文本归一化，把数字拆成单个 digit，为代码加入只含空白的 token，并用 byte fallback 覆盖罕见字符 [251]。这些不是普通预处理细节。数字切分影响算术和数字复制，空白 token 影响代码格式，byte fallback 决定混合文字和罕见姓名如何进入模型，词表变大也会增加 embedding、LM head 和 softmax 成本。

checkpoint 转换也是词表和架构契约的一部分。轻量实现可能把 Q/K/V 存成一个 stacked projection，而来源 checkpoint 可能分成独立矩阵或多个 model-parallel shard；转换时必须按正确维度 concat、重排 QKV shard、保留 tokenizer 文件，并区分真实 vocab size 与 padded vocab size。一个 checkpoint 能生成文本，不代表它的参数统计、sampling mask 或 tensor-parallel merge metadata 都正确。

转换验收应从“能加载”提升到“语义等价”。至少应固定原始 checkpoint、转换后 checkpoint、tokenizer、模板和配置哈希，用同一批短 prompt、含 padding 的 batch、长前缀和特殊 token 边界样例比较 logits、生成文本、stop reason 和采样屏蔽。若 QKV 排列、embedding tying、padded vocab、RoPE base、KV head 分组或 tensor-parallel merge 有一项错位，模型可能仍能输出通顺文本，却在长上下文、结构化输出或后续 LoRA 训练中系统性偏移。

部署接口还要求 tokenizer 与聊天模板一起版本化。BOS、EOS、system、user、assistant 标记、工具调用占位符、图像或文件占位符、停止词和 padding 方向都会影响模型看到的 token 序列。Adapter、reward head 或安全分类头若绑定在旧模板上，换模板后可能仍能运行，却在角色边界、停止条件或工具参数上产生系统性错误。发布记录应把 tokenizer 文件、模板文本、special-token id、padded vocab 和 embedding/head tied 状态放在同一处。

5.5.7 长上下文

长上下文能力包括可接受长输入、在长输入中定位证据、跨段落整合信息和抵抗无关干扰。很多模型报告只证明了第一点。严肃评测应包括 needle、multi-hop、矛盾证据、长文档摘要和真实 RAG 工作负载。

因此，长上下文声明最好拆成三层。可运行长度回答“张量能否前向通过”；检索长度回答“模型能否在干扰文档中找到证据”；综合长度回答“模型能否把远处、多处、甚至互相冲突的证据组织成正确答案”。三层指标可以同时存在也可以互相矛盾：一个模型可能接受 128k token，却只稳定使用最近几千 token；也可能在 needle 测试中命中唯一字符串，却在多表格、多章节或反事实证据任务上失败。

长上下文评测还应同时报告成本。Prefill 延迟、KV cache bytes/token、decode TPOT、cache eviction、检索切片策略、引用忠实性和短上下文回归，都是同一个 claim 的组成部分。一个 128k 模型若只能在低并发下通过 needle 测试，却在真实多文档问题上忽略中间证据或显著拉高 p95 延迟，就不能简单宣传为“支持 128k 有效上下文”。

发布长上下文模型时，最好把失败切片按距离和证据形态分层记录。近端、中段、远端、跨表格、冲突证据、重复干扰、工具返回和 RAG 拼接边界应分别给出样例与指标；服务侧则应说明 chunk-prefill、prefix cache、sliding window、attention sink、cache eviction 和取消请求后的释放策略。这样读者才能区分位置外推失败、检索组织失败、缓存策略丢证据和单纯预算不足，而不是把所有问题混成“长上下文不稳定”。

5.5.8 MoE 变体

MoE 用稀疏专家扩大总参数量，每个 token 只激活部分专家。它带来容量优势，也带来路由、负载均衡、all-to-all 通信和专家退化问题。报告 MoE 时应区分总参数、激活参数、训练 FLOPs 和服务延迟。

MoE 层可以抽象为 router 对 token hidden state h_t 打分，选择 top-k 专家集合 S_t，再按路由权重组合专家输出。这个抽象必须落到指标上：每个专家收到多少 token、路由熵是多少、容量因子是否导致 token dropping、共享专家与路由专家如何分工、负载均衡损失或 bias 更新规则是什么、all-to-all 时间占 step time 或 decode latency 的多少。没有这些数据，总参数量会遮住真正的计算和通信成本。

对于课堂代码，还应区分 dense MoE、sparse MoE 和 efficient sparse MoE。Dense MoE 用所有专家的加权和，主要用于理解“专家特征组合”；sparse MoE 先取 top-k，但如果仍然执行所有专家，只是数学稀疏而非计算稀疏；efficient sparse MoE 才把 token dispatch 到少数专家以减少计算。三者 loss 可能都能下降，但系统含义完全不同。把这个区别写清楚，可以防止读者把 notebook demo 的正确性误读成生产实现的吞吐收益。

5.5.9 超越 LLaMA 类 Decoder

RetNet、Mamba、Mamba-2、Titans 和扩散语言模型等工作探索注意力之外的序列建模。它们提出不同的长序列和生成机制，但必须在相同数据、预算、上下文任务和后训练流程下比较。单一困惑度不能证明替代结构已解决真实系统问题。

5.5.10 评测提醒

架构比较最容易被不公平设置污染。数据、tokenizer、训练 token、上下文长度、推理预算和后训练都可能比结构本身更影响结果。评测应固定关键变量，或者明确承认比较的是完整配方而非单一架构。

开放模型报告是重要的一手资料，但不能自动等同于独立复现。一个报告可能同时改变结构、数据过滤、语料比例、后训练、评测模板和安全策略；如果把全部增益归给某个组件，就需要消融证据。部署时也应评测实际 checkpoint 和解码路径，而不是只看“LLaMA-like”“MoE”或“长上下文”标签。

一个出版级消融表至少应包含 dense/MoE 或 MHA/GQA/MLA 的 matched budget 对照。可固定 tokenizer、训练 token、数据顺序、优化器、上下文分布和后训练流程，只改变目标组件；若无法固定，就应明确写成“完整系统比较”。评测列也不能只放平均 benchmark 分数，还应加入长上下文切片、多语言/代码切片、复制和引用任务、服务延迟、峰值显存、失败样例和可复现配置。这样读者才能判断结构变化是在提高模型能力，还是在移动成本、

数据或推理预算。

5.5.11 架构核算

对 dense decoder，设隐藏宽度为 d、query head 数为 H_q、KV head 数为 H_{kv}、head 维度为 d_h、SwiGLU 宽度为 m，单层参数可粗略写为

$$P_{\text{layer}} \approx d(H_q d_h) + 2d(H_{\text{kv}} d_h) + d(H_q d_h) + 3dm.$$

前四项分别对应 W_Q、W_K、W_V 和 W_O，最后一项对应门控前馈网络。这个近似忽略 embedding、norm、bias 和取整，但能说明关键点：GQA 主要减少 KV projection 和服务 cache，dense 参数常由 FFN 主导。若是 MoE 层，$3dm$ 应分别按所有专家统计总参数，并按 top-k 激活专家统计每 token 活动参数。出版级架构章节必须说明自己报告的是哪个量。

5.6 关键术语、实现要点与练习

关键术语。 RoPE 用旋转方式编码位置信息；YaRN 通过频段化 RoPE scaling 扩展上下文并保护短上下文能力；RMSNorm 用均方根归一化隐藏状态；SwiGLU 是门控前馈结构；configuration contract 指 config JSON 中决定权重形状、位置语义、cache 行为和 token 接口的字段集合；GQA/MQA 通过共享 key/value 头降低 KV cache；MLA 用 latent 表示压缩注意力状态；sliding-window attention 只让某些层看到有限窗口内的历史 token；chunked attention 把注意力历史限制在分块语义内；attention backend 指 eager、SDPA、FlashAttention、Flex Attention 或模型专用 kernel 等实际执行路径；QK norm 表示在 attention score 前对 query/key 做归一化的结构选择；cache implementation 指动态、静态、量化、offload 或 hybrid KV cache 的运行时选择；cache policy 定义缓存对象、释放、滚动窗口和复用边界；multimodal placeholder token 是文本序列中代表图像、视频或其他媒体 embedding 插入位置的特殊 token；MoE 通过稀疏专家提升总容量；router auxiliary loss 用额外目标约束专家负载或路由行为；activated parameters 表示特定 token 或 forward pass 实际使用的参数；load balancing 表示防止少数专家过载而其他专家闲置的 router 机制；matched-budget ablation 指在数据、token、优化器、上下文和推理预算尽量一致时比较单一结构变化。

实现要点。报告架构时应写清层数、隐藏维、head 数、KV head 数、head dimension、latent 维度、FFN 维度、位置编码、词表和上下文长度；config 审计应保存 `hidden_size`、`intermediate_size`、`num_attention_heads`、KV head、RoPE 参数、sliding-window 或 chunked 层、attention backend、QK norm、embedding tying、attention/MLP bias、cache implementation 和 special-token id；词表报告应说明真实词表、padded vocab、special token、

chat template、byte fallback、图像占位 token 和每张图对应的媒体 token 数；MoE 报告必须区分总参数、激活参数、router、top-k 专家、共享专家、router 辅助损失、负载均衡和专家并行；长上下文主张必须配合有效上下文评测。

实现审计还应保留最小等价测试：prefill 与逐 token decode 的 logits 对齐；真实 vocab 与 padded vocab 的采样屏蔽正确；RMSNorm 的 ϵ 和 dtype 与 checkpoint 一致；SwiGLU 宽度和取整规则能复原参数量；GQA cache 只保存 KV head 而不是物理展开后的 query head；MLA cache 保存 latent 与 decoupled RoPE key；MoE 统计真实 dispatch 后的专家 token count、routing weight 和 all-to-all 时间。

练习。

1. 比较 MHA、MQA 和 GQA 的 KV cache 大小。
2. 解释为什么长上下文能力不能只看最大输入 token 数。
3. 比较 PI、NTK-aware RoPE 和 YaRN 扩展长上下文时如何保护短上下文能力。
4. 给一个 MoE 模型报告写检查清单：哪些数字可能误导读者?
5. 阅读一个 DeepSeek-V3 类模型报告，区分 MLA、MoE 路由、FP8 训练、自定义 kernel 和专家并行分别属于架构主张还是系统主张。
6. 比较 attention、state-space 和 recurrent memory 方案在长文档任务中的评测要求。
7. 把一个 GPT-style block 改写成 LLaMA-style block，列出 norm、bias、位置处理、MLP、attention cache 假设分别如何变化。
8. 令 $d = 4096$，比较隐藏宽度为 $4d$ 的 GELU MLP 与隐藏宽度约为 $(8/3)d$、并向 256 的倍数取整的 SwiGLU MLP 参数量。
9. 设计一个消融实验，判断 GQA 的质量变化是否独立于总参数量、训练 token 数和 tokenizer。
10. 为一个 LLaMA 类 checkpoint 写配置向量，至少包括 $L, d, H_q, H_{\text{kv}}, d_h, m, V, T_{\max}$、RoPE scaling、norm ϵ、chat template 和 cache policy。
11. 设计三组 RoPE/KV cache 回归测试，分别覆盖无 padding 短前缀、left padding batch 和超过滑动窗口边界的长样本。
12. 给一个 padded vocab 的实现写采样屏蔽测试，说明真实词表、padded 行、embedding、LM head 和 tokenizer 文件如何共同约束 checkpoint。

13. 把一个 toy MoE 从“所有专家都计算再置零”的版本改成 dispatch-compute-combine 版本，并报告专家 token count、routing weight 和有效 FLOPs。

14. 为 MLA 设计一个缓存等价测试：比较缓存 full K/V、缓存 latent 加 RoPE key、以及矩阵吸收后的 logits、cache bytes/token 和 decode 延迟。

15. 读取一个 Llama、Qwen3 或 Gemma3 的 config JSON，写出会改变权重形状、cache 大小、位置语义、embedding tying、滑动窗口和 special-token 语义的字段，并说明每个字段的回归测试。

16. 比较 DynamicCache、StaticCache、Quantized Cache、offloaded cache 和 hybrid cache 在长上下文服务中的适用边界，说明哪些收益来自模型结构，哪些来自运行时策略。

17. 读取一个 Llama4TextConfig，把 MoE 层、每 token 专家数、本地专家数、QK norm、chunked attention、无 RoPE 层和图像占位 token 分到“权重形状”“位置语义”“路由语义”“多模态接口”和“运行时 backend”五类。

18. 设计一组 cache 策略回归：同一 prompt 下比较 dynamic、static、offloaded 和 quantized cache 的 logits 差异、峰值显存、TTFT、TPOT、长上下文复制错误和取消请求后的资源释放。

5.7 结构化检查表

5.7.1 LLaMA 类架构报告清单

报告项	必填内容	缺失时的风险
Block 契约	pre-norm、RMSNorm ϵ、bias、残差顺序、final norm、QK norm	checkpoint 可加载但 logits 与原实现漂移
Attention	H_q、H_{kv}、head dim、RoPE 分量、sliding/chunked 层、cache 表示、backend	KV cache 预算和长上下文行为无法复现
FFN 或 MoE	SwiGLU 宽度、取整规则、专家数、top-k、共享专家、router loss、负载均衡	总参数、激活参数和服务成本被混在一起
Tokenizer 与媒体接口	真实词表、padded vocab、special token、chat template、byte fallback、图像占位 token、媒体 token 数	微调、采样、工具调用、多模态输入或安全策略语义错位
长上下文	原训练长度、目标长度、scaling 规则、继续训练、短上下文回归	只证明可运行长输入，不能证明有效利用证据
评测归因	数据、token 预算、优化器、后训练、推理预算和硬件条件	把完整配方收益误写成单一架构收益

第六章　优化与预训练

6.1　预训练问题

预训练配方是架构、token 预算、数据混合、优化器、精度、batch、上下文长度、硬件拓扑和停止规则的耦合选择。Scaling laws 显示损失随参数、数据和计算可预测变化；Chinchilla 进一步说明许多早期大模型相对数据量而言参数过多 [132, 50]。预训练损失必须按 token 解释。一个 batch 中不同数据源、不同长度和不同语言的 token 贡献不同，平均 loss 可能掩盖某个语料切片已经退化。训练计划还要区分 tokens seen、unique tokens、effective batch size、sequence length 和 gradient accumulation。两个运行即使总 FLOPs 接近，也可能因为数据顺序、warmup、上下文长度或验证集不同而不可比较。一些新配方还会在标准 next-token prediction 之外加入辅助目标。DeepSeek-V3 的 multi-token prediction（MTP）把未来多个 token 的预测作为训练辅助：主目标仍预测下一个 token，但额外模块预测向后两步、三步等更远位置，并用加权损失影响共享表示 [31]。这类目标要和推理阶段的多 token 解码区分开：辅助 head 可以只用于训练并在发布前移除，也可以保留为服务加速路径。训练报告必须写清它属于目标函数、checkpoint 结构还是运行时 decoder。

因果 decoder 的基本目标可以写成对有效位置集合 $\mathcal{M}$ 的平均负对数似然：只要 token 位置被 mask 掉，就不应贡献梯度。基础预训练通常让几乎所有非 padding 位置参与 loss；指令微调则常常屏蔽 prompt token。这个差异意味着，预训练 loss 衡量的是语料混合上的分布压缩，不等价于有帮助、真实或安全。报告 loss 时必须同时说明 tokenizer、document separator、padding、packing、loss mask 和验证语料，否则数值相同也可能代表完全不同的训练目标。

训练计划应先写清单位。设参数量为 N、训练 token 为 D、序列长度为 T、单卡 micro-batch 为 B_μ、梯度累积步数为 G、数据并行规模为 P，则一次 optimizer update 消耗的 token 近似为 $B_\mu TGP$，更新步数约为 $D/(B_\mu TGP)$。Dense decoder 的一阶训练计算常用 $6ND$ 估算。这个公式忽略 attention 细节、重算、数据加载和系统损耗，但足以提醒读者：参数、数据、上下文长度和硬件效率必须一起规划。

这个记账也决定日志的横轴。若一次恢复训练后改变了序列长度、packing 效率、数据并行规模或梯度累积步数，step 编号就不再代表同样数量的 token。出版级训练记录应把

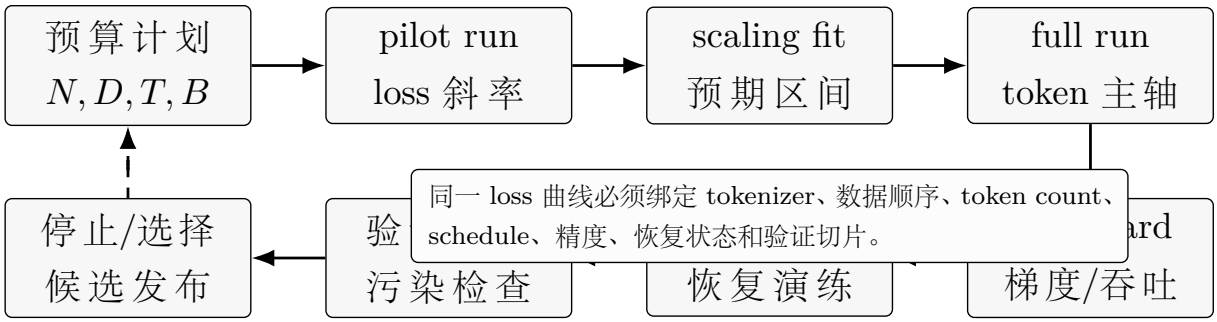

图 6.1: 预训练是实验控制环，而不是一次性批处理。该图是对 scaling law、compute-optimal training 和现代大模型训练报告的原创综合 [132, 50, 31]。

token count 作为主时间轴，把 optimizer step 作为辅助时间轴，并在学习率、验证 loss、吞吐和 checkpoint 事件旁边同时标出两者。否则两个看似“同样训练 10 万步”的运行，可能已经消耗了完全不同的训练证据。

预训练 loss 的可比性还依赖 tokenizer 和边界语义。相同文本在不同 tokenizer 下会产生不同 target 数；同一 tokenizer 若改变 end-of-text、文档拼接、padding 或跨文档 mask，loss 单位也会变化。因此 perplexity 最适合在同一实验族内部比较。跨模型报告时，它只能作为诊断线索，不能单独证明能力、事实性或安全边界更好。

预训练更像一个受控实验，而不是一次不可回头的批处理任务。计划阶段要先估算参数、token、上下文、batch 和硬件吞吐；pilot run 用来验证 loss 斜率、吞吐、显存和数据加载；scale-law 拟合给出预期 loss 区间；full run 再用 dashboard、checkpoint、验证切片和停止规则闭环控制。若没有 pilot-derived expectation，训练到一半才发现 loss 偏离、验证集泄漏或 checkpoint 不能恢复，已经无法把最终模型解释成干净实验。

图 6.1 把预训练写成实验控制环。高水平模型报告不只给最终 checkpoint，还应说明计划、pilot、scale-law、full run、dashboard、checkpoint 和验证切片如何互相约束。

辅助 MTP 目标也应落到公式和日志中。一个简化写法是

$$\mathcal{L}_{\text{pretrain}} = \mathcal{L}_{\text{NTP}} + \sum_{j=1}^{J} \lambda_j \mathcal{L}_{\text{MTP},j},$$

其中第 j 个辅助头预测更远的未来 token，λ_j 控制它对共享表示的影响。报告时应说明未来 token 偏移、辅助 head 是否共享 embedding 或 LM head、梯度是否回传到主干、训练结束后是否删除，以及验证 loss 是否分开记录 NTP 和 MTP。否则读者无法判断 MTP 是表示学习技巧、训练正则项，还是会改变推理结构的并行解码机制。

验证集必须在训练前固定，并按来源而不是随机 token 切片拆分。随机切片容易与训练语料共享近重复文档，使 validation loss 过于乐观。一个严肃 dashboard 至少应展示 general web、code、math、books、academic text、多语言桶、对话或合成数据等切片 loss。

全局 loss 平稳时，某个切片的异常上升可能提前暴露数据比例漂移、tokenizer 对代码不友好、坏 shard、课程边界或污染检查失败。

验证切片还要和下游主张分离。预训练 validation loss 说明模型是否压缩了指定语料分布；代码、数学、安全、长上下文和事实性 benchmark 则说明候选 checkpoint 是否值得进入后训练或发布。若把同一个公开 benchmark 同时用于数据筛选、checkpoint 选择和最终宣传，分数就不再是独立证据。run card 应记录每个评测集在训练、调参和最终报告中的角色。

配套 LLaMA 预训练脚本把这些原则落实为可审计的数据混合。RedPajama 预处理说明把总量约 1.2T tokens 的语料拆成 CommonCrawl、C4、GitHub、Books、ArXiv、Wikipedia 和 StackExchange 等来源；训练脚本再用 arxiv 2.5、book 4.5、c4 15.0、commoncrawl 67.0、github 4.5、stackexchange 2.0、wikipedia 4.5 的权重构造 CombinedDataset。这里的权重不是注释，而是目标函数的一部分：同一个 7B 架构若把 GitHub 或 ArXiv 比例调高，代码、数学和学术语言的梯度占比就会改变，最终模型不能再被称为同一次预训练的简单复现。

PackedDataset 也体现了预训练的工程含义。脚本用 $T+1$ 个 token 组成一个样本，前 T 个作为输入，后 T 个作为 target；这样位置 t 的 label 才是 $t+1$。在多进程训练中，dataset 还接收 world size、global rank、随机种子和 chunk 数，确保不同 worker 读到不同 shard。若这些参数没有进入训练记录，读者无法判断一个 loss 曲线是来自真实的数据并行，还是来自不同进程重复消费同一批 token。

6.2 AdamW、学习率与稳定性

AdamW 将权重衰减与自适应梯度更新解耦，是现代大语言模型训练的常见优化器 [161]。Warmup 防止早期更新过大；cosine decay 或类似 schedule 控制后期收敛；梯度裁剪和混合精度处理数值稳定性。BF16 常用于大模型训练，因为指数范围比 FP16 更稳。稳定性不是单个超参数能保证的。pre-norm、初始化、残差尺度、激活函数、attention score 范围、MoE router、长上下文位置外推和低精度格式都会影响梯度分布。训练 dashboard 应区分优化健康和模型质量：前者包括 loss、gradient norm、update norm、learning rate、clipping rate、tokens/s、显存和通信时间；后者包括代码、数学、多语言、安全、事实性和生成样例。

AdamW 的“W”不是装饰。经典 L_2 正则把 $\lambda\theta$ 加进梯度，经过 Adam 的自适应分母后，有效正则会随二阶矩估计变化；AdamW 则把权重衰减作为独立的参数收缩项。参数组也属于算法：常见做法对二维以上权重矩阵使用 decay，对 bias、norm 参数和有时 embedding 关闭 decay。报告中只写“使用 AdamW”并不充分，还应写明 $(\beta_1, \beta_2, \epsilon)$、weight decay、参

数组规则、clipping 阈值、batch token 数和学习率 schedule。

AdamW 的矩估计应写进读者能复现的算法描述。若 $g_t = \nabla_\theta \mathcal{L}_t(\theta_t)$，则

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

经过 bias correction 后，参数更新再叠加解耦的权重衰减项。常见参数组规则可粗略写成：二维以上矩阵进入 decay 组，bias、norm scale 和有时 embedding 进入 no-decay 组。这个规则不是代码风格，而是正则化定义；一个实现若把 RMSNorm scale 放进 decay 组，或者把 embedding 与输出 head 的 tying 关系处理错，就会改变训练轨迹。

学习率 schedule 决定优化预算如何被花掉。线性 warmup 让动量和激活尺度先稳定下来；cosine decay 让更新幅度逐渐降低；非零最小学习率或末段 annealing 则会改变最后阶段的探索方式。Batch size 应以 token 为单位报告，因为同样的样本数在不同 context length 下代表完全不同的梯度统计。若恢复训练后改变 sequence length、packing 效率、数据并行规模或累积步数，step 轴就不再可比，日志应把 token count 作为主时间轴。

配套 nanoGPT 脚本给出了最小但完整的 token batch 记账：`tokens_per_iter = gradient_accumulation_steps * ddp_world_size * batch_size * block_size`。DDP 模式下，脚本先把全局累积步数按 world size 整除，保证每个进程只负责自己的 micro-step；日志再把乘回 world size 后的 token 数作为一次迭代的证据量。LLaMA 预训练脚本用 `batch_size // devices` 得到每进程 batch，再用 `process_batch_size // micro_batch_size` 得到梯度累积次数。两个写法表面不同，实质都在回答同一个问题：一次 optimizer step 到底平均了多少 token 的梯度。

梯度累积的实现顺序也必须写清。正确做法是在每个 micro-batch 上把 loss 除以累积步数，再 backward；DDP 或 FSDP 训练时，非最后一个 micro-step 应关闭跨卡同步，最后一步再同步、裁剪、更新和清梯度。若先 backward 完整 loss 再累积，等价于把学习率放大了累积步数；若每个 micro-step 都同步，则通信开销会吞掉 batch 扩大的收益。这些不是性能小技巧，而是优化算法定义的一部分。

最新 Accelerate 文档把这个细节进一步具体化：对 causal LM 这类 token 级任务，跨 micro-batch 的正确归一通常应按整个累积窗口内的非 padding token 总数计算，而不是把每个 batch 已经平均过的 loss 再取平均 [58]。在多卡训练中，非 padding token 数还要跨设备聚合；否则长短样本混合、最后一个不满累积窗口的 batch、或不同 rank 的 padding 差异，都会让同一个 global batch 得到不同的有效梯度。出版级 run card 因而要记录 loss reduction 是 sum 还是 mean、忽略标签如何定义、token normalizer 是否跨设备聚合、非最后 micro-step 是否 no-sync，以及 dataloader 末尾是否强制同步。Transformers Trainer 的最新接口也把这一点暴露为训练契约：自定义 `compute_loss_func` 可以接收整个累积 batch 的 item 数，`TrainingArguments` 中还有 gradient accumulation、跨设备 token 平均、输入 token 计数、max grad norm、warmup、scheduler、checkpoint 与评测切片等字段 [113]。

因此“用了 Trainer”不能替代训练证据；报告必须说明这些字段最终如何解析到 token 级目标。

一个常见 schedule 可分段写成：前 t_w 步从 0 线性升到 $\eta_{\max}$，随后按 cosine 衰减到 $\eta_{\min}$，之后保持最小学习率。末段如果更换数据 mixture 做 annealing，应视为新的训练阶段，而不是原曲线的自然延续。它可能提升某些目标评测，也可能掩盖主语料上的退化。因此报告中应把 annealing 数据、持续 token 数、学习率范围和能力回归单独列出。

AdamW 的超参数也隐含梯度噪声假设。许多大模型配方使用 β_1 接近 0.9，而把 β_2 设得低于 Adam 原始默认值，以便二阶矩在长训练和变动数据混合中更快适应 [138]。ϵ 在低精度训练里也不是无关常数：当二阶矩很小时，它会改变有效步长。比较优化器或复现实验时，应把 $(\beta_1, \beta_2, \epsilon)$、weight decay、no-decay 规则、clip 阈值、warmup token 数、峰值学习率和最小学习率一起列出。

配套 GPT 实现采用的默认组合是一个典型小规模预训练基线：峰值学习率 6×10^{-4}，warmup 2000 iter，cosine decay 到 6×10^{-5}，AdamW 的 $(\beta_1, \beta_2) = (0.9, 0.95)$，weight decay 为 0.1，global norm clip 为 1.0。模型代码把所有二维及以上张量放进 decay 组，把一维 bias 和 norm 参数放进 no-decay 组，并在 CUDA 上优先使用 fused AdamW。这个例子适合作为读者实现检查：参数组数量、decay 参数量、no-decay 参数量和是否启用 fused optimizer 都应该打印进日志。

参数组审计应具体到张量名称。embedding、LM head、RMSNorm scale、bias、MoE router、专家矩阵和 LoRA adapter 是否 decay，都会改变正则化含义。一个可靠实现不仅打印 decay/no-decay 参数量，还应在配置中保存匹配规则，并在 checkpoint 恢复时验证同一规则仍然选中同一组张量。否则代码重构后模型可能继续训练，却已经不是同一个优化实验。

框架默认值也要被当作显式选择来审计。Transformers Trainer 的优化器、学习率调度、参数衰减过滤、评测 dataset 字典、分布式 sampler、data seed、mixed precision、torch compile、gradient checkpointing、DeepSpeed/FSDP 配置和 Hub 保存策略都通过 `TrainingArguments` 暴露 [113]。这些字段会影响目标、日志和恢复语义：例如参数衰减过滤若按 LayerNorm 类型、bias 或 norm 名称排除，换成 RMSNorm、adapter、router 或自定义模块时就要确认规则仍然正确；评测集若用字典传入，metric 名称会带前缀，checkpoint 选择也必须绑定到某一个切片；IterableDataset 在分布式训练中需要可控 generator 或 `set_epoch`，否则每个 epoch 的样本顺序不可复现。高质量书稿应教读者读取解析后的配置，而不是只看命令行默认值。

Logit 尺度也是稳定性问题。Baichuan 2 报告了两个实用机制：对输出 head 做归一化，使 logits 不过度依赖 embedding norm；再加入 max-z loss，惩罚过大的最大 logit [251]。这类辅助项的目的不是简单降低训练 loss，而是让 repetition penalty、logit processor 和

softmax 在推理时更不脆弱。报告中应说明系数、使用阶段，以及移除该项后生成行为是否变化。

max-z penalty 可以写成 $z_{\max} = \max_i \ell_i$ 与 $\mathcal{L}_{\text{max-}z} = \lambda z_{\max} z_{\max}$，其中 ℓ_i 是词表 logits。它惩罚的是过大 logit 尺度，而不是某个具体答案。若使用这类项，报告应同时观察训练 loss、validation loss、logit norm、重复率、采样温度敏感性和安全拒答边界；否则可能把稳定解码的收益误写成模型知识提升。

Loss spike 需要定位，而不是迷信单个补丁。首先判断 spike 是否只出现在某个 worker、某个 shard、某个语料成分，还是整个任务同时异常；其次判断被污染的是权重、optimizer moments，还是只是单个坏 batch。如果 optimizer state 已经出现 inf 或 NaN，从 spike 前 checkpoint 恢复通常比继续训练更稳。若验证 loss 永久改变，这次运行应被视为新分支，而不是原曲线上的小波动。

梯度裁剪是护栏，不是稳定性的根因。Global norm clipping 先计算

$$\|g\|_2 = \sqrt{\sum_i \|g_i\|_2^2},$$

若超过阈值 c，就把所有梯度按 $c/(\|g\|_2 + \epsilon)$ 缩放。混合精度下应先 unscale 再裁剪，否则裁剪阈值和 loss scaling 会纠缠。clipping rate 也应记录：偶发裁剪可能只是坏 batch，持续裁剪则说明学习率、batch、初始化或数据分布存在问题。

混合精度是另一条稳定性边界。FP16 指数范围有限，常需要 loss scaling 避免梯度下溢 [166]；BF16 尾数更短但指数范围更宽，常让大模型训练少依赖显式 loss scaling；FP8 能提升吞吐和带宽效率，却要求记录 E4M3/E5M2 角色、scale 粒度、scale 更新时间、饱和统计、舍入策略、accumulation dtype 和 BF16/FP16 基线。报告只写“使用低精度训练”不足以说明数值路径，也不足以支撑质量可比。

Accelerate 的低精度训练文档把 FP8 从“一个 dtype”拆成后端和 recipe：当前推荐的维护路径包括 TransformersEngine 和 torchao，MS-AMP 被保留为 legacy 兼容但不再推荐；配置中需要说明 backend、FP8 format、amax 计算、history length、margin、interval、哪些线性层保持高精度，以及 eval 是否也启用 autocast [59]。这意味着 FP8 训练报告不能只写硬件是 H100 或“开启 fp8”。它应列出哪些 tensor 真的进入 8-bit 路径、哪些层保持 BF16/FP32、通信或 optimizer state 是否降精度、是否有饱和/amax 日志、是否在相同数据和 seed 下与 BF16 baseline 对齐，以及 quality gate 是按 loss、能力切片还是后训练兼容性判断。若 FP8 只提升 tokens/s，却让某些验证切片或后续 SFT 稳定性退化，它不是可发布优化。

nanoGPT 训练脚本中，若 CUDA 支持 BF16，就优先选择 bfloat16；否则使用 float16 并启用 GradScaler。脚本还打开 TF32 matmul 和 cuDNN TF32，以提高 Ampere 及以后 GPU 上的吞吐。LLaMA 预训练脚本则直接使用 `bf16-mixed` Fabric 精度。报告这些选择

时要区分参数存储、矩阵乘累积、梯度缩放和 optimizer state 的 dtype；否则“BF16 训练”和“FP16 加 loss scaling”会在同一个表格里被误认为等价。

6.3 可复现训练记录

训练报告应记录随机种子、数据 manifest、tokenizer、模型配置、optimizer state、精度、并行策略、checkpoint 频率、验证集、损失切片、硬件和软件版本。没有这些记录，一个损失曲线只能说明“某次运行发生过”，不能支持科学解释。

checkpoint 不是简单保存权重。要可靠恢复训练，还需要优化器状态、scheduler 状态、随机数状态、数据加载位置、并行切分配置和 tokenizer/chat template 版本。大规模运行中，失败恢复本身就是训练系统的一部分：checkpoint 太频繁会浪费 I/O，太稀疏会增加故障成本；只让 rank 0 写日志也要保证所有 worker 在 barrier 和种子上保持一致。中间 checkpoint 还可以是研究 artifact。Baichuan 2 计划释放从约 220B tokens 到 2.64T tokens 的 7B 中间 checkpoint，用来观察 benchmark 切片随训练 token 增长如何提升或平台化 [251]。这种 checkpoint 序列只有在 tokenizer、数据 manifest、评测 harness 和 token 计数一致时才有解释力，否则训练动态可能只是评测协议变化。

checkpoint cadence 是成本决策。频繁保存降低故障后丢失的 token，但会吃掉 I/O 带宽和存储；间隔过长则让一次节点故障变成大量重复计算。实用方案通常保留一小段 rolling recovery checkpoint，同时按更稀疏间隔导出 archival checkpoint 给评测和分析。若 checkpoint 是分片保存的，报告还应说明 shard layout、tensor parallel 元数据、optimizer state 是否同样分片、以及能否在不同并行规模下恢复。

恢复流程应在昂贵训练前演练。一个小型 dry run 可以训练数步、保存、恢复，再检查 loss、learning rate、data-loader 位置和随机数轨迹是否连续。常见错误包括只恢复权重而忘记 optimizer state，scheduler 从头开始，数据 shard 被重复或跳过，dropout seed 改变，或者用略有差异的架构加载 checkpoint。这个演练不是运维琐事，而是验证训练曲线能被解释的前提。

Accelerate 的 `save_state`/`load_state` 边界正好说明 checkpoint 类型不能混用：训练恢复需要模型、optimizer、scaler、RNG generators 和已注册对象的状态，且主要面向同一训练环境中的恢复；`save_model` 或普通权重导出则更接近发布/评测 artifact [55, 60]。自动 checkpoint 命名、保存上限、旧 checkpoint 删除、safetensors、每节点保存策略和自定义 pre-hook 都会影响恢复证据。run card 因此应分两列写清：recovery checkpoint 用来从故障继续训练，必须证明 optimizer、scheduler、token cursor 和随机数连续；canonical checkpoint 用来评测、后训练或推理，必须证明 tokenizer、config、权重 shard、adapter、special tokens 和生成样例一致。若只验证其中一种，就不能声称训练既可恢复又可发布。

配套 GPT checkpoint 是一个具体例子：它保存 model state、optimizer state、model args、当前迭代数、best validation loss 和运行 config；resume 时还会从 checkpoint 强制恢复层数、head 数、embedding 维度、block size、bias 和 vocab size，并清理编译后可能出现的 `_orig_mod.` 前缀。这个策略比“加载权重继续跑”严格得多，因为它防止读者在不知情的情况下用不同架构、不同词表或不同上下文长度接续同一条 loss 曲线。

恢复演练还应覆盖导出路径。训练 checkpoint 可能是 ZeRO/FSDP shard、tensor-parallel shard 或带 optimizer moments 的内部格式，而评测和发布需要 canonical weights、tokenizer、配置和安全模板。一个通过的 dry run 应证明四件事：同一 world size 能继续训练；必要时能转换 world size 或导出 full state dict；导出权重能被推理脚本加载并生成固定样例；恢复后没有重复或跳过大段 token。没有这些证据，checkpoint 只能证明“文件被写出”，不能证明训练可恢复。

导出 checkpoint 也应绑定评测环境。若评测脚本加载的是合并后的 full weights，而训练恢复用的是分片 optimizer state，二者需要共享同一 tokenizer、position 配置、RoPE scaling、vocab padding 和 chat template。导出后至少应跑一个固定 prompt、一个短验证集和一个参数 shape 清单，确认 canonical artifact 与训练分支一致。否则发布包可能通过训练恢复测试，却在真实推理栈中语义漂移。

数据加载正确性也应成为 run card 的验收项。最小证据不是只打印 dataset 名称，而是保存 shard manifest、每个来源的 token 预算、实际消费 token 数、worker/rank 分配、epoch 或 sample cursor、document boundary 策略、跨文档 loss mask、重复和跳过样本检查，以及验证集隔离证据。恢复训练后，应能证明每个 rank 从同一全局样本位置继续，既没有重复消费一个大 shard，也没有因为 worker 重排跳过一个来源桶。若数据 loader 使用随机 buffer、weighted sampling 或 curriculum，报告还要给出采样分布的实测直方图，而不是只写配置权重。

训练异常复盘要进入同一套证据链。一次 loss spike、NaN、吞吐骤降或验证切片漂移，至少应记录发生 token count、optimizer step、学习率、gradient norm、update norm、clipping rate、混合精度 scale、数据 shard id、rank id、最近 checkpoint、是否跳过 batch、是否回滚、以及恢复后的若干 step 是否回到预测区间。这样的 incident record 能防止团队把“训练继续跑完了”误当成“训练科学上仍然连续”。若回滚点、数据顺序或 optimizer state 改变，后续模型应标为新的训练分支，并在报告中说明与原曲线的关系。

停止和分支决策也需要可审计。一个 checkpoint 可能在 validation loss 上仍有下降，但边际收益已低于继续训练的计算、时间、数据污染和机会成本；另一个 checkpoint 可能全局 loss 稍差，却在代码、数学或低资源语言切片上更适合后训练。run card 因此应保存候选 checkpoint 列表、每个候选的 token count、成本、验证切片、污染检查、安全探针、后训练兼容性和服务成本估计。最终选择哪个 checkpoint 进入 SFT、对齐或发布，不应只

由最后一步 loss 决定。

6.4 深入展开：预训练是统计目标和工程配方的耦合

本章把预训练写成一个耦合系统。交叉熵目标定义了 token 级预测，optimizer 决定参数如何更新，data mixture 决定梯度来自哪里，batch size 和 sequence length 决定显存和统计效率，learning-rate schedule 决定训练早期和后期的稳定性，精度格式决定数值范围和吞吐。任何一个选择改变，都可能让“同样的模型”变成不同实验。

数据顺序是优化变量，不是 dataloader 的细节。大语料通常混合网页、书籍、代码、论文、多语言文本、数学、对话和合成数据。代码与数学的梯度统计不同于普通网页；低质量重复可能降低训练 loss，却伤害下游鲁棒性；benchmark-like 文档还会造成污染 [218, 247]。严肃训练应保存 mixture weights、token counts、dedup 策略、过滤策略、shard 顺序、随机种子和验证集来源。

RedPajama 例子可以帮助读者理解“数据集名称”为什么不够。文档中完整语料由数千个 jsonl 文件组成，示例样本只有少量文件；GitHub 子集还限定在特定开源许可证内。训练脚本假设使用 LLaMA 风格的 32k SentencePiece tokenizer，并把不同来源通过路径前缀、权重和随机种子组合起来。若论文或书稿只写“使用 RedPajama 训练”，却不写具体文件集合、tokenizer、许可证过滤、采样权重和 shard 顺序，那么复现者即使用同名数据集也可能得到不同目标分布。

Packed sequence training 能提高 accelerator 利用率，但会制造边界选择。若忽略文档边界，模型可能学习互不相关文档之间的虚假过渡；若每个文档都插入 boundary token，tokenizer 和词表必须稳定支持该标记；若跨边界做 loss mask，分布式 shuffle 后 token ids 和 mask 必须保持同步。验证集也应按来源拆分，而不是只从训练语料中随机切片，否则近重复文档会让 validation loss 看起来过于健康。

validation dashboard 应按 mixture component 展开，而不是只给一个全局 loss。代码、数学、书籍、网页、多语言、学术文本、对话和合成数据的损失曲线可能朝不同方向变化。若全局 loss 平稳但代码 loss 上升，可能是数据比例或 tokenizer 对代码不友好；若某个语言桶突然下降，可能是近重复泄漏；若某个 shard 的 loss 爆炸，可能是文件损坏或编码错误。按来源拆分的验证集，是把优化问题和数据问题分开的最低要求。

Scaling laws 的作用不是崇拜规模，而是帮助规划计算。Kaplan 风格 scaling law 显示 loss 随参数、数据和计算呈规律变化；Chinchilla 进一步说明许多早期模型相对训练 token 不足，参数过多而数据不够。实践中，团队需要在给定预算下决定参数量、token 数、训练步数和停止点，而不是只追求最大模型。

可审计规划表至少应包含 N、D、T、token batch、预估 $6ND$ FLOPs、attention 与

重算开销、目标 MFU、packing 效率、预计故障和 checkpoint 损耗、验证 cadence，以及 go/no-go 条件。小规模吞吐测试通常高估全量训练，因为它没有包含数据加载、checkpoint、验证、网络拥塞、straggler 和重启损耗。规划表应在 pilot run 后更新，而不是把理论峰值当作交付承诺。

Emergent abilities 是 smooth loss scaling 的另一面。Wei 等把它定义为"小模型上不存在、大模型上出现"的能力，因此下游任务曲线看起来像跨过阈值，而不是从小模型平滑外推 [241]。但这个阈值不是能力本身的物理常数，它会随数据质量、模型家族、prompt、微调、推理方法和指标移动。Exact match 或 accuracy 可能掩盖 log probability 的渐进改善，直到某个规模后分数突然上升。因此报告 emergent ability 时，应给出完整模型序列、训练 compute 或参数规模、数据与 prompt 协议、随机基线、指标选择、置信区间，以及 loss 或 calibrated likelihood 是否在 headline score 前已经改善。

优化稳定性来自许多小机制的共同作用。AdamW 解耦权重衰减；warmup 防止早期大更新；cosine decay 控制后期学习率；gradient clipping 限制异常梯度；BF16 提供比 FP16 更大的指数范围；FP8 需要额外缩放和验证。FP8 也不是一个单一格式：深度学习 FP8 提案区分 E4M3 和 E5M2，常见做法是权重/激活偏向 E4M3，梯度偏向 E5M2，矩阵运算仍在更高精度累积，并在转成 FP8 前对张量做 scaling [167]。因此 FP8 报告应写清张量角色、scaling 粒度和更新规则、饱和或舍入行为、accumulation dtype，以及对齐的 BF16/FP16 基线。MoE、长上下文和多模态输入都会改变激活和梯度分布，因此"默认超参数"不能脱离架构版本。

本章还强调训练 dashboard 应分成两类指标。优化健康看 training loss、validation loss、gradient norm、update norm、tokens/s、显存、通信时间和数据加载等待；模型质量看代码、数学、多语言、事实性、安全、长上下文和生成样例。一个训练运行可能在优化指标上很健康，却在某个能力切片上退化。只有把两类指标分开，才能判断问题来自训练系统还是数据和目标。

停止规则应写在训练开始前。它可以是最大 token 预算、目标 validation loss、预算上限，或某个边际收益不再值得继续的决策点。没有 stop rule，团队容易因为 loss 仍在下降而继续训练，哪怕部署目标更需要数据清洗、后训练、评测或推理优化。可审计 run card 至少应给出 N、D、T、token batch、MFU、packing 效率、故障与 checkpoint 损耗、验证切片和停止规则；只给参数量和 benchmark 分数，不足以支持可复现训练主张。

停止规则也应允许选择非最后 checkpoint。最后一步 loss 最低不一定意味着发布价值最高：某个早期 checkpoint 可能在安全探针、低资源语言、代码切片、后训练兼容性或服务成本上更平衡。候选选择表应列出 token count、训练成本、验证切片、污染检查、后训练 pilot、推理成本和回滚风险。这样读者才能理解"为什么停止"和"为什么选择这个 checkpoint"是两个相关但不同的决策。

run card 还应把“运行是否健康”和“模型是否值得发布”分开。前者记录 loss、梯度、吞吐、显存、通信、I/O、恢复事件和告警；后者记录能力切片、生成样例、污染检查、安全探针、长上下文、代码和数学。一个运行可能系统健康但模型质量一般，也可能能力提升但服务成本不可接受。出版稿若只报告 benchmark 分数，读者无法判断这是优化配方成功、数据更好、评测泄漏，还是后训练阶段补救了 base model 的缺陷。

可审计 run card 可以概括为

$$\mathcal{C}_{\text{run}} = (N, D, T, B_{\text{tok}}, \eta_{\text{mfu}}, \eta_{\text{pack}}, \eta_{\text{fail}}, \mathcal{V}, \mathcal{S}),$$

其中 η_{mfu} 是模型 FLOPs 利用率，η_{pack} 是 packing 效率，η_{fail} 是因故障、checkpoint 和恢复损失的时间比例，$\mathcal{V}$ 是验证切片集合，$\mathcal{S}$ 是停止规则集合。这个元组不替代完整日志，但能防止模型报告只给参数量和 benchmark 分数而省略训练证据。

6.5 章节细节

6.5.1 预训练问题

预训练把模型结构、数据混合、token 预算、优化器、batch、上下文长度、精度和硬件绑定在一起。交叉熵目标简单，但训练配方高度耦合。一个实验结果必须说明这些条件，才有可复现意义。

6.5.2 目标记账与数据顺序

训练 token 数、unique token 数、epoch、sequence length、effective batch size 和数据顺序都影响学习。数据重复会让某些样本被过度加权，数据顺序会影响训练早期梯度。报告“训练了多少 token”仍不足以解释模型行为。

配套代码中的最小记账式是 $B_{\text{tok}} = B_{\mu}TGP$，对应到实现就是 micro-batch、block size、梯度累积步数和数据并行进程数的乘积。LLaMA 预训练还要把 $T+1$ 的 packed sample 区分开：输入长度是 T，多出来的一个 token 只用于构造向右平移的 target。

6.5.3 辅助目标与 MTP

多 token 预测等辅助目标会改变每个位置提供的梯度信号。它可能改善表示或支持后续并行解码研究，但不能直接等同于线上提速。记录时应说明辅助 loss 权重、未来 token 偏移、head 是否共享、训练后是否移除，以及对验证 loss 和下游任务的影响。

6.5.4 AdamW 与参数组

AdamW 解耦权重衰减和自适应梯度更新，是大模型训练常用优化器。实践中通常对 bias、norm 参数和 embedding 设置不同 decay 规则。参数组错误会改变正则化，导致难以解释的收敛差异。

6.5.5 Schedule、Warmup 与 Batch Size

Warmup 防止训练初期更新过大，cosine decay 等 schedule 控制后期收敛。Batch size 增大可以提高硬件效率，但也会改变优化噪声和学习率需求。有效 batch 还受到 gradient accumulation 和数据并行规模影响。

在实现中，gradient accumulation 的 loss 缩放、跨卡同步开关和裁剪顺序必须作为同一个算法块检查。正确日志应能回答：每个 optimizer step 处理多少 token，是否只在最后一个 micro-step 同步梯度，是否先 unscale 再裁剪，以及 learning rate 是按 optimizer step 还是按 token count 更新。

若样本长度可变，检查表还应加入“按 token 而非按 batch 平均”的验收项。一个通过的最小测试可以把同一个 global token batch 分成不同 micro-batch 切法，确认最终梯度、loss normalizer 和 optimizer step 相同；在多卡场景下，还要确认有效 token 数跨设备聚合，最后一个部分累积窗口不会改变学习率或梯度尺度。

6.5.6 梯度裁剪与稳定性

梯度裁剪限制异常更新，混合精度需要 unscale 后再裁剪。NaN、inf、loss spike、激活爆炸和 router collapse 都应被监控。稳定训练不是一个技巧，而是初始化、归一化、精度、优化器和数据共同作用的结果。

6.5.7 Checkpoint 与可复现性

可恢复训练需要保存模型权重、优化器状态、scheduler、随机数状态、数据加载位置、并行配置和 tokenizer 版本。只保存权重无法复现训练轨迹。大规模训练中，checkpoint 频率还必须平衡 I/O 成本和故障恢复成本。

小规模代码也应养成同样纪律：checkpoint 至少包含模型参数、optimizer state、模型配置、当前 step、最佳验证 loss 和运行 config；resume 时要检查架构字段、vocab size、block size 和编译前后参数名前缀是否一致。

同时要区分“恢复态”和“发布态”。恢复态关注训练是否能从同一个并行/精度环境继续，发布态关注权重是否能被普通推理或后训练脚本消费。两者应分别有验收命令、固定样例和失败回滚路径。

6.5.8 计算最优规划

Scaling laws 帮助在给定计算预算下选择参数量和 token 数。Chinchilla 结果提醒研究者，过大参数而训练 token 不足会浪费计算。规划预训练时应先确定目标损失、数据规模和可用硬件，而不是只追求最大参数。

6.5.9 监控、评测与停止规则

Dashboard 应区分优化健康和能力质量。前者看 loss、梯度、更新、吞吐、显存和通信；后者看代码、数学、多语言、安全和事实性切片。停止训练不应只看训练 loss，还应看验证趋势、能力回归和成本收益。

6.5.10 实现笔记

从小模型开始实现可以暴露标签、mask、dtype 和 checkpoint 错误。扩大规模前，应先验证过拟合小 batch、恢复训练一致性、数据切片 loss 和生成样例。实现纪律比单次大规模运行更能培养可靠判断。

几个实现错误应写进单元测试。第一，target shift 必须确保位置 t 预测 $t+1$，否则模型可能学习复制当前 token。第二，padding 和跨文档边界必须按目标设计进入或退出 loss。第三，gradient accumulation 应在 backward 前按累积步数缩放 loss，或者等价地在 optimizer step 前缩放梯度，不能让有效学习率随 accumulation 改变。第四，validation 必须关闭 dropout 并固定数据顺序。第五，最后一个不满累积步数的 micro-batch 要么丢弃，要么按真实累积数修正缩放。

6.6 关键术语、实现要点与练习

关键术语。Scaling law 描述 loss 随参数、数据和计算变化的规律；Chinchilla-optimal 指给定计算下参数和数据的更优平衡；MTP 是预测多个未来 token 的辅助训练目标；packed sequence 把文档切分或拼接成固定长度训练块；validation slice 是按来源、语言、任务或质量桶拆出的验证子集；AdamW 解耦权重衰减；parameter group 定义哪些参数接受 decay；warmup 防止早期大更新；global norm clipping 限制异常梯度；mixed precision 在吞吐和数值稳定之间权衡；max-z loss 惩罚过大的最大 logit；FP8 scaling 指把高精度张量映射到 FP8 可表示范围的缩放策略；MFU 衡量模型相关 FLOPs 对硬件峰值的利用率；run card 是记录训练目标、数据、系统、验证和停止规则的可审计摘要。

实现要点。训练日志应分离优化健康和模型质量；token count 应作为主时间轴；MTP、max-z、annealing、低精度和参数组都要作为目标或优化配方记录；RedPajama 这类混合

语料要记录来源权重、tokenizer、文件集合、许可证过滤和 shard 顺序；gradient accumulation 要记录 loss reduction、有效 token normalizer、同步时机、clip 顺序和 token batch；Trainer/Accelerate 配置要保存解析后的 optimizer、scheduler、mixed precision、checkpoint 与评测字段；checkpoint 应区分恢复态和发布态，并包含权重、优化器、scheduler、随机数状态、数据位置、分片元数据和 tokenizer 版本；实验记录应保存 tokenizer、数据 manifest、模型配置、精度、并行策略、验证切片、停止规则和恢复/导出演练结果。

练习。

1. 解释为什么两个相同 FLOP 的训练运行可能不可比较。
2. 设计一个训练 dashboard，列出至少八个优化健康指标和八个质量指标。
3. 写出恢复训练所需的 checkpoint 字段。
4. 给定 `batch_size=12`、`block_size=1024`、`gradient_accumulation_steps=40`、`world_size=8`，计算一次 optimizer step 消耗多少 token，并说明 DDP 中每个进程实际执行多少个累积 micro-step。
5. 为 next-token loss 加一个 two-token-ahead 辅助损失，说明 loss 权重、label 对齐和训练后是否保留辅助 head。
6. 说明 BF16 相对 FP16 的优势和 FP8 的额外风险。
7. 把 RedPajama 的七个来源写成一张 mixture 表，说明为什么采样权重、tokenizer 和许可证过滤都属于训练目标的一部分。
8. 为一次 FP8 训练实验写 run-card 条目，说明哪些张量使用 E4M3/E5M2、哪里保持高精度累积、scaling factor 如何选择、用哪个 BF16/FP16 基线验证。
9. 给一个恢复训练 dry run 写验收标准：loss、learning rate、数据位置、随机数状态和 optimizer moments 应如何连续?
10. 设计一个 validation dashboard，至少包含六个数据来源切片，并说明每个切片能提前暴露哪类失败。
11. 设计一个 max-z loss 消融：需要记录哪些 logit、重复率、采样稳定性和安全回归指标，才能判断它改善的是解码稳定性而不是掩盖质量退化?
12. 为一个 7B dense 模型和一个 7B active-parameter MoE 模型各写一张预训练规划卡，比较 N、D、token batch、MFU、checkpoint 损耗和验证切片应如何解释。

类别	指标
优化健康	training loss、validation loss、gradient norm、update norm、learning rate、clipping rate、tokens/s、data-loader wait、communication time、checkpoint time
模型质量	代码、数学、多语言、长上下文、事实性、安全、拒答边界、RAG faithfulness、生成样例、回归测试
系统成本	GPU 利用率、MFU、显存峰值、网络带宽、I/O、失败恢复时间、能耗、美元成本

表 6.1: 预训练 dashboard 指标按优化健康、模型质量和系统成本分层。

6.7 结构化检查表

6.7.1 训练 dashboard

表 6.1 把预训练 dashboard 分成优化、质量和系统三类，避免只用 loss 解释训练状态。

6.7.2 预训练 run card 最小字段

字段	必填内容	缺失时的风险
目标与规模	N、D、T、token batch、optimizer steps、$6ND$ 估算	参数量和 token 预算不可比较
数据与目标	mixture weights、packing、loss mask、MTP/max-z 等辅助 loss	loss 数字无法解释
优化配方	AdamW 超参数、decay 组、schedule、warmup、clip、annealing	收敛差异无法复现
精度与系统	BF16/FP16/FP8 角色、MFU、packing 效率、故障和 checkpoint 损耗	吞吐和质量 claim 脱节
验证与停止	验证切片、污染检查、go/no-go 条件、停止规则	训练是否应继续只能凭直觉
恢复与导出	save/resume dry run、world-size 转换、canonical checkpoint、固定生成样例	checkpoint 可写但不可恢复或不可发布

第七章　分布式训练系统

7.1　内存与并行

训练内存由参数、梯度、优化器状态、激活、临时 buffer 和通信 buffer 组成。数据并行复制模型并对应梯度；ZeRO/FSDP 切分参数、梯度和优化器状态 [207, 267]；tensor parallelism 切分层内矩阵；pipeline parallelism 切分深度；expert parallelism 处理 MoE 专家路由。每种并行方式都在移动瓶颈。数据并行简单，但每个 worker 仍要保存完整模型状态；FSDP/ZeRO 节省显存，但增加 gather/scatter 和 checkpoint 复杂度；tensor parallelism 降低单卡矩阵尺寸，但引入频繁层内通信；pipeline parallelism 可以容纳更深模型，却产生 bubble 和调度复杂度；expert parallelism 适合 MoE，但 all-to-all 和负载均衡会成为核心问题。

大规模预训练不是把教材里的优化器放进更大的循环。训练系统会决定可承受的 batch、可用的序列长度、稳定的精度、检查点写入频率和失败恢复速度。数学上清楚但放不进显存、通信效率极低或无法恢复的配方，不是可运行配方。

核心设计问题是把计算图映射到显存有限、互联带宽有限的 accelerator 集群上。如果模型能放进单卡但 batch 放不下，数据并行可能足够；如果优化器状态太大，就需要 sharding；如果单层矩阵乘法太大或太慢，就要考虑张量并行；如果深度或激活显存占主导，则流水线并行或激活检查点可能更合适；如果模型是稀疏 MoE，还要额外处理路由和负载均衡。现代前沿训练通常组合多种策略，而不是只选一个名词。

因此，并行计划应被写成系统映射，而不是写成“启用了三维并行”。最小映射包括模型层到 stage 的对应关系、tensor parallel group 是否留在同一高速互联岛内、data parallel group 是否跨节点、expert 放置是否与 all-to-all 拓扑匹配、FSDP/ZeRO wrap 粒度、activation checkpointing 范围、bucket 大小、checkpoint 格式和导出路径。这个映射一旦改变，训练的显存峰值、step time、失败恢复和可发布 checkpoint 都会改变。高质量报告应让读者能从这些字段推导为什么该模型能训练，而不是只相信最终模型名。

图 7.1 把分布式训练画成计算图到集群资源的映射。读者应从图中追问每种并行策略移动了什么状态、引入了什么 collective、怎样恢复和怎样导出，而不是只记住 FSDP、TP、PP 或 EP 的缩写。

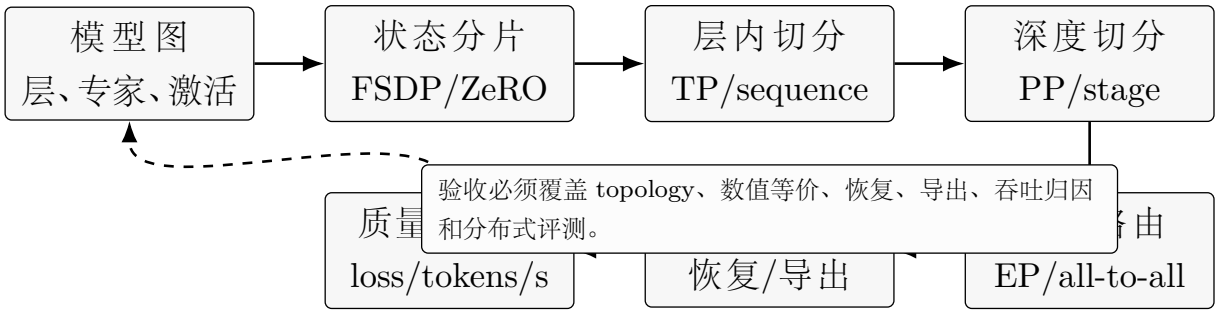

图 7.1: 分布式训练是模型图、内存、通信、checkpoint 和评测的系统映射。该图是对 ZeRO/FSDP、Megatron-style tensor parallelism 和 MoE 训练报告的原创综合 [207, 219, 31]。

7.2 通信与吞吐

大规模训练不是只买更多 GPU。互联带宽、all-reduce、all-to-all、pipeline bubble、激活检查点、kernel fusion 和数据加载都会限制吞吐。报告训练效率时，应同时给出 tokens/s、step time、MFU、GPU 利用率、显存峰值和失败恢复策略。

DeepSeek-V3 的训练资料把这些问题集中到一个案例中：MoE 架构需要 expert parallel 和 all-to-all；pipeline parallel 需要减少 bubble；FP8 训练需要缩放和高精度累积；专用通信与 kernel 决定理论 FLOPs 能否变成真实 tokens/s [31]。因此，本章不把并行策略写成名词表，而是要求读者说明每个策略移动了哪些字节、增加了哪些 collective、又通过什么 overlap 或调度把通信隐藏起来。吞吐还必须和样本效率一起看。某个并行方案可能让 tokens/s 更高，却迫使 batch size 变大、学习率重调或数据顺序改变，最终在相同 token 预算下质量更差。系统优化的正确比较应固定训练目标和质量阈值，报告达到目标 loss 或目标 benchmark 所需的总时间、总能耗、总成本和失败次数。

通信原语描述的是语义，不是链路算法。All-reduce 让每个 rank 得到规约结果，reduce-scatter 让每个 rank 得到规约结果的一片，all-gather 把分片聚成完整张量，all-to-all 让每个 rank 向其他 rank 发送不同切片。底层可以是 ring、tree、分层节点内/节点间算法或硬件专用路径。大消息通常更受带宽限制，小 bucket 更受 latency 限制；如果 bucket 太大，通信启动太晚，overlap 失败；如果 bucket 太小，kernel launch 和 collective latency 又会吃掉收益。因此通信优化必须给出 profiler 时间线证据，而不是只写“异步”或“开启 overlap”。

7.3 实现纪律

CS336 把资源核算、GPU、kernel、Triton、并行和 scaling laws 放在核心位置，是因为语言模型工程的很多结论只有在实现后才显现 [222]。一个研究者应能从 CPU 小模型调试开始，再逐步加入 GPU、混合精度、FlashAttention、checkpointing 和分布式训练。这种从

零实现不是为了重复造轮子，而是为了识别抽象边界。使用成熟框架时，很多关键决定被隐藏在 dataloader、collator、attention kernel、optimizer wrapper 和 distributed launcher 中。只有写过最小版本，研究者才容易发现 loss mask 错误、padding 泄漏、通信等待、显存碎片和 checkpoint 不可恢复等问题。

7.4 深入展开：分布式训练的核心是记账

本章的主线是“accounting”。参数、梯度、优化器状态、激活、临时 buffer 和通信 buffer 都占显存；矩阵乘法、attention、all-reduce、all-to-all、检查点 I/O 和数据加载都占时间。所谓并行策略，就是把这些字节和时间从一个位置移到另一个位置。说某个方法“省显存”是不够的，必须说明省的是参数、梯度、优化器还是激活，又增加了什么通信和重算。

数据并行最容易理解：每张卡保留模型副本，处理不同 batch，再对应梯度。它简单但显存重复。ZeRO/FSDP 把参数、梯度和优化器状态切分到不同 worker，节省显存但增加 gather、scatter 和检查点复杂度。张量并行切分层内矩阵，适合单层太大无法放入一张卡的情况；流水线并行切分网络深度，但会产生 pipeline bubble；专家并行服务于 MoE，但 all-to-all 和负载均衡会成为瓶颈。

本章要求训练报告给出 tokens/s、step time、MFU、GPU 利用率、显存峰值、通信时间和失败恢复策略。吞吐不能脱离质量解释：一个方案提高 tokens/s，却强迫使用过大 batch 或改变数据顺序，可能在相同 token 预算下质量更差。正确比较应报告达到某个 loss 或能力阈值所需的总时间、总成本和失败率。实现纪律还包括故障处理。大规模训练必然遇到节点失败、网络抖动、文件系统慢、数据样本损坏和非确定性重启。检查点频率、rank 对应、日志归并和恢复测试都应在训练开始前设计。一个不能稳定恢复的训练系统，即使单步吞吐很高，也不适合大规模运行。

有用的系统设计要把“保存了哪些字节、增加了哪些 collective、如何 overlap、检查点如何写、哪种失败模式更可能出现”同时说清楚。若只报告“用了 FSDP”或“用了三维并行”，读者无法判断结果是算法贡献、通信拓扑优势、数据加载优势，还是检查点/恢复策略不同造成的。可复现训练报告应把并行度、rank 映射、bucket 大小、通信后端、精度策略、数据吞吐和恢复时间写入同一张 run card。

发布级训练系统还应把并行计划写成可检查对象，而不是一串 launcher 参数。一个完整计划至少包含 data/tensor/pipeline/expert parallel degree、每个维度的 process group、节点内和节点间 rank mapping、每个 rank 持有的参数或专家、global batch 的计算方式、gradient accumulation 步数、activation checkpointing 策略、精度策略、通信 bucket 大小、checkpoint 格式和导出路径。只有这些字段齐全，读者才知道一次训练能否换 world size 恢复、能否导出普通推理权重，以及某个吞吐数字是否来自拓扑优势。

7.5 章节细节

7.5.1 为什么训练系统是模型的一部分

训练系统决定可训练模型的大小、数据吞吐、故障恢复和成本。相同架构在不同并行策略、精度和数据管线下可能得到不同结果。系统不是外部工程细节，而是模型配方的一部分。

7.5.2 内存记账

显存由参数、梯度、优化器状态、激活、临时 buffer 和通信 buffer 组成。优化器状态常常比参数本身更占空间，激活随 batch 和序列长度增长。没有内存记账，就无法选择并行策略。

以 AdamW dense training 为例，BF16 参数约 2 bytes，BF16 梯度约 2 bytes，两个 FP32 moment 各 4 bytes，持久状态已经接近每参数 12 bytes；若还保留 FP32 master weights，预算会继续上升。70B dense 模型仅这些状态就需要数百 GB，尚未包含激活、CUDA allocator 碎片、kernel workspace 和通信缓冲区。长上下文或较大 micro-batch 下，激活项常常成为限制项；只看参数量会严重低估真实训练显存。

临时显存是最容易漏算的一类。矩阵乘法库会申请 workspace，attention kernel 可能需要 scratch buffer，collective 可能需要 staging buffer，layout 转换会产生短暂副本，显存碎片还可能让“总空闲显存足够”时仍然分配失败。因此生产训练不会把单卡显存填到最后一 MB，而会保留可解释的 headroom。

容量规划可以先写成表格式 run card：参数、梯度、优化器状态、master weights、激活、communication buffer、kernel workspace、dataloader staging 和保留 headroom 分别占多少；其中哪些随参数量 N 增长，哪些随 $B_\mu T$ 增长，哪些随 collective bucket 或 sequence shard 增长。这样做的价值不是得到一个精确常数，而是在扩容前发现谁是第一瓶颈。如果持久状态已经超过单卡预算，先考虑 ZeRO/FSDP；如果单层矩阵乘法无法高效执行，考虑 tensor parallel；如果激活项随长上下文爆炸，考虑 checkpointing、sequence parallel 或 context parallel；如果 I/O 或 checkpoint 时间接近 step time，就必须先修数据管线和存储，而不是继续增加 GPU。

并行策略的选择顺序应从瓶颈分类开始，而不是从框架默认配置开始。若瓶颈是冗余 optimizer state，就先切状态；若瓶颈是层内 GEMM 尺寸或单层延迟，就先切矩阵；若瓶颈是深度、激活或 stage 放不下，就考虑 pipeline 与重算；若瓶颈是 MoE 路由，就先画 expert placement 和 all-to-all 路径；若瓶颈是数据或 checkpoint I/O，增加并行度只会把等待时间放大。这个决策树能防止把所有问题都误写成“需要更多 GPU”。

7.5.3 数据并行

数据并行让每个 worker 处理不同 batch，再对应梯度。它实现简单，扩展性好，但每个 worker 保存完整模型和优化器状态。规模继续增大时，显存重复会成为瓶颈。

同步数据并行的主要 collective 是梯度 all-reduce。实现通常把梯度分 bucket，使后层反向传播产生的梯度能在前层反向计算时开始通信。它的优点是保留本地模型代码，缺点是每个 worker 仍复制参数、梯度和优化器状态。对于中小模型这是最便宜的扩展方式；对大模型，数据并行常常先被显存而不是算力限制。

数据并行还会改变优化问题。如果随着 worker 数增加而放大全局 token batch，在固定 token 预算下 optimizer update 次数会减少，warmup、learning rate schedule 和梯度噪声都需要重调。因此扩容不是单纯的系统参数变化，也会改变训练动力学。

训练报告应明确 global batch 的定义。常见写法是 $B_{\text{global}} = B_{\mu} \times P_{\text{dp}} \times G$，其中 B_{μ} 是每 rank micro-batch，P_{dp} 是数据并行组大小，G 是 gradient accumulation 步数；若还按序列打包或按 token 预算动态 batch，则应报告每 step 实际 token 分布。loss 是否除以 accumulation 步数、是否按有效 token 数归一、padding token 是否被 mask、梯度裁剪在 accumulation 前还是后执行，都会影响数值等价性。单卡和多卡同 global batch 的 loss 对齐测试，是发现这些错误的最低成本手段。

数据并行的扩容还应记录优化状态是否真的等价。梯度同步前后的 norm、梯度裁剪比例、有效 token 数、学习率 step 计数、warmup 进度和随机数据顺序都应可检查。若 global batch 随 GPU 数线性增大，模型看到的 optimizer update 数会减少，验证曲线可能因为噪声下降而看似更平滑，却在同 token 预算下泛化更差。出版级比较应同时报告 fixed-step、fixed-token 和 fixed-quality 三种视角，避免把更大 batch 的吞吐优势误写成模型算法优势。

nanoGPT 的 DDP 脚本把这些约束压缩成很少代码 [133]。单机多卡用 `torchrun` 启动并指定每节点进程数；多节点还要显式给出节点数、节点序号、主地址和主端口；若没有 InfiniBand，还要让 NCCL 配置与真实网络一致。脚本从 `RANK`、`LOCAL_RANK` 和 `WORLD_SIZE` 推导 device、master process 与 seed offset，并要求 gradient accumulation steps 能被 world size 整除。这些字段不是 launcher 噪声，而是可复现实验的一部分。

DDP 下的梯度累积也有隐藏契约。为了保持目标 global batch 不变，脚本会按 world size 缩小每个进程的 accumulation steps，并把 `tokens_per_iter` 写成 accumulation、world size、micro-batch 和 block size 的乘积。每个 micro-step 的 loss 先除以 accumulation steps，只有最后一个 micro-step 才设置 `require_backward_grad_sync` 触发梯度同步；否则通信会在每个 micro-step 发生，既改变吞吐，也可能改变梯度裁剪与日志解释。报告中应写明 no-sync 规则、loss normalizer、梯度裁剪位置和 master-only checkpoint 规则。

代码层面还有两个常见陷阱。第一，随机 `memmap` offset 取 batch 时，如果只靠每个 rank 的 seed offset 区分样本，短跑实验可能足够，但长跑复现仍应记录数据位置、随机种

子和 checkpoint 中的迭代号。第二，编译或包装模型后，checkpoint key 可能出现包装前缀，恢复时需要剥离或转换；因此 checkpoint 报告应说明保存的是 raw model、DDP wrapper、compiled model，还是已经清理过的 canonical state dict。

7.5.4 ZeRO 与 FSDP

ZeRO 和 FSDP 切分参数、梯度和优化器状态，显著降低单卡显存。代价是更多 gather、scatter、通信和检查点复杂度。它们适合模型状态过大但通信网络足够强的场景。这里必须区分“内存分片”和“计算分片”。ZeRO-3/FSDP 在模块调用之间可以只保存参数分片，但某个模块真正执行本地 forward/backward 时，仍然需要 all-gather 出该模块的完整参数视图。它解决的是持久模型状态显存，而不是自动把一个矩阵乘法切成多卡协同计算。如果瓶颈是优化器状态或权重放不下，先考虑 ZeRO/FSDP；如果瓶颈是单层矩阵太大或单层计算太慢，仍然需要张量并行。

ZeRO 的阶段可以按切分对象记忆：Stage 1 只切优化器状态；Stage 2 再切梯度；Stage 3 连参数也切。FSDP 接近 ZeRO 第三阶段的精神：模块大部分时间只保留本地参数分片，在前向和反向传播前聚合当前模块需要的完整参数，用完后释放；梯度通过规约分散回到各 rank 的分片。wrap 粒度决定峰值显存与 collective 频率，过粗会一次实例化太多参数，过细会让通信和调度开销上升。

ZeRO 后续方案体现了课件中“用时间和通信换空间”的工程主线。ZeRO-Offload 把部分优化器状态、梯度和计算移到 CPU 侧，缓解 GPU 显存压力，但只有在 CPU 计算和 PCIe/NVLink 传输能与 GPU 计算重叠时才真正有效 [210]。ZeRO-Infinity 进一步把 CPU 与 NVMe 纳入异构内存层次，预取粒度、存储带宽和故障恢复也会成为训练算法的一部分 [208]。ZeRO++ 则从通信量入手，用权重量化 all-gather、分层 partition 和量化梯度交换降低 ZeRO 的通信瓶颈 [238]。因此，分布式训练报告不能只写“用了 ZeRO-3”，还应报告主机内存、PCIe/NVLink 带宽、NVMe 吞吐、collective 体积、overlap 效率，以及是否和未压缩基线做过质量对齐。

DeepSpeed/Accelerate 的配置文件也不是启动脚本附录。它至少决定 distributed type、进程数、ZeRO stage、BF16/FP16 开关、每 GPU micro-batch、梯度累积步数、all-gather bucket、是否 offload optimizer 或 parameter，以及这些字段是否由框架在运行时自动填充。若 run card 只写命令行而不写解析后的 DeepSpeed JSON、Accelerate state 和实际 world size，后续读者无法判断显存节省来自 ZeRO stage、量化、梯度累积，还是来自更小的真实 batch。配套实操脚本中把 `zero_stage`、混合精度、GPU 数、offload 设备和梯度累积写进 tracker 的做法，正是为了把 launcher 状态变成可审计证据。

最新 Accelerate DeepSpeed 文档把这条证据链拆成两层：plugin 路径暴露 ZeRO stage、梯度累积、梯度裁剪、optimizer/parameter offload device、NVMe path、ZeRO-3 初始化和

16-bit 保存、混合精度、MoE layer class、hostfile、multinode launcher，以及 MS-AMP/FP8 相关的开关和 opt level；config-file 路径还会把 optimizer、scheduler、all-gather/reduce bucket、communication overlap、effective batch 和若干 `auto` 字段交给 DeepSpeed JSON 解析 [56]。因此配置审计不能只保存 `accelerate launch` 命令。它还要保存解析后的 accelerate config、DeepSpeed config file、实际 world size、per-device batch、有效 train batch、offload 介质、多节点启动器、FP8/MS-AMP 开关和运行时展开后的 `auto` 数值；否则同一实验换机器或换 launcher 后，OOM、吞吐差异、低精度数值差异和优化器更新语义都无法追溯。

ZeRO-3 和 PEFT 结合时，检查点导出本身就是一条工程链路。训练目录里保存的可能是分片优化器状态和 adapter state，而不是推理端能直接加载的完整模型。发布前通常要先从 ZeRO shards 重建 full-precision state dict，再抽取 PEFT adapter 权重，保留 reward model 的 score head 等额外模块，最后决定只发布 adapter，还是把 LoRA merge 到 base model。Tokenizer、chat template 和 special-token 配置必须跟着导出，否则 adapter 虽然能加载，生成语义也可能已经错位。

7B/70B 后训练脚本进一步说明同一套分布式外壳下的作业并不等价。SFT、reward model、DPO 和 PPO 都可能传入 DeepSpeed 配置、BF16、gradient checkpointing、per-device batch、梯度累积和输出目录，但 PPO 还会引入 reference/policy 关系、value head、reward scoring、生成长度、KL 目标和 minibatch 切分；QLoRA 的 NF4 量化能降低 base weight 显存，却不会自动消除激活、reward 前向、logprob 缓存和 adapter 导出的成本。因此 70B run card 应按组件记账：policy、reference 或共享 reference、reward model、value head、adapter、optimizer state、KV/activation 和生成缓存分别占多少，保存的是 ZeRO 训练状态、PEFT adapter，还是 merge 后模型。

FSDP 检查点也要区分用途。Full state dict 便于评测和发布，但 materialize 成本高；sharded state dict 适合恢复训练，但依赖 world size、parameter flattening 和 sharding layout 元数据。可靠流水线必须分别测试 resume 和 export：节点失败后能恢复训练，训练结束后也能导出推理或后训练所需的 canonical checkpoint。

Accelerate FSDP 指南把这个边界落成配置和保存接口：sharding strategy、offload params、auto-wrap policy、prefetch、state-dict type、use-orig-params 和 sync-module-states 都会改变训练状态结构；checkpoint 推荐用 sharded state dict 与 `save_state` 保存分片模型、优化器、scheduler 和随机状态，而发布时还要通过 Accelerator 的 get-state-dict 接口或 merge utility 形成可加载权重 [57]。所以 FSDP run card 不应只写“保存了 ckpt”。它要分开列出 resume artifact、export artifact、state-dict 类型、是否 rank0-only materialize、是否 CPU offload、merge 脚本和固定推理样例。

FSDP 配置本身也会进入模型契约。Mixed precision 的 param、reduce 和 buffer dtype，是否使用 full-shard 或 hybrid-shard，是否 flatten parameters，哪些模块被排除在 wrapping

外，shared embedding 和 tied LM head 如何处理，都会影响显存峰值和 checkpoint 结构。若训练中加入 LoRA、reward head 或额外 tokenizer token，还要确认这些模块没有被错误地丢进只读分片或导出时被漏掉。高质量实验记录应包含一次小模型 dry run：保存、重启、改 world size 转换、导出、再用推理脚本加载并生成固定样例。

PyTorch 的 FSDP2 把这些约束进一步显式化为 `fully_shard` 与 `DTensor` 契约 [198]。初始化时，参数会从普通 tensor 原地转换为位于 device mesh 上的 sharded `DTensor`；forward/backward hook 在模块计算前 all-gather 参数，把它们临时暴露成普通 tensor，计算后再释放完整参数并恢复成分片表示。因此 optimizer 应在 `DTensor` 参数上构造，带输入的普通模型调用必须触发 hook，bottom-up wrapping 顺序会决定 all-gather group。FSDP2 也改变 checkpoint 预期：报告不能只写 full state dict，而要说明分片 `DTensor` 如何经由 `DTensor.full_tensor()`、reshard 或 PyTorch Distributed Checkpoint API 导出 [195]。

这类接口的出版价值在于暴露失败点。若训练代码绕过模块调用路径而直接调用 `forward`，hook 可能不会执行；若优化器在分片封装前构造，优化器状态可能不对应分片参数；若根模块和层块的封装顺序不清楚，参数聚合粒度和峰值显存就不可复现。FSDP2 run card 应记录设备网格维度、每层封装顺序、优化器创建位置、检查点接口、重分片或导出路径，以及一个固定 batch 的保存、恢复和导出验证。

FSDP2 的通信边界不是一个独立的 bucket 超参数，而是由 `fully_shard` 应用到哪些模块决定。若只对根模块调用一次，前向前可能变成一次大 all-gather，反向后再做一次大 reduce-scatter，几乎没有机会与层内计算 overlap；若按 Transformer block 自底向上包裹，每个 block 会形成更细的通信组，下一层参数 all-gather 才可能在当前层计算时提前发起。报告应把 root group、block group、embedding/output projection group 分开列出，并说明 FQN 是否保持不变、哪些参数被排除在子模块组之外、是否调用 `model(input)` 触发 hook、是否需要显式 `unshard` 或注册额外 forward 方法。这样才能解释显存峰值和通信时间来自 wrap 设计，而不是来自一个笼统的“FSDP2 开关”。

FSDP2 的验证不能只看 loss 能下降。一个小规模验收应先在同一输入上比较未分片模型与 FSDP2 模型的 logits 子集、loss、梯度范数和参数更新方向；再在 checkpoint 后重启，确认 optimizer state、scheduler step、token count 和 sampler offset 都连续；最后把 sharded `DTensor` state 重新 materialize 或 reshard 成发布形态，用普通推理脚本加载并跑固定 prompt。这样可以区分“训练能继续跑”和“权重能被另一个系统正确消费”。若只测试 save/resume，而不测试 export，发布时才发现 tied embedding、LM head、adapter 或 tokenizer 配置缺失，就已经太晚。

设备网格也应写成训练规格。二维 mesh 可以把 tensor parallel 放在节点内高速互联，把 FSDP 放在跨节点维度；三维或更高维布局还可能同时包含 pipeline stage、expert group 和 data-parallel replica。报告应给出 global rank 到 node、GPU、TP rank、FSDP rank、

PP stage 和 EP group 的映射，以及每个维度使用的通信后端和带宽假设。这个映射影响 collective 延迟、checkpoint shard 命名、故障恢复和导出脚本。若换一批机器或改变 rank 顺序后结果不同，问题常常不是模型，而是原报告没有把 topology contract 写出来。

导出验收应有单独的 go/no-go 表。至少检查参数总数、每个权重 dtype、张量形状、shared/tied weights、LoRA 或 reward/value head、position embedding 或 RoPE 配置、tokenizer special token、chat template hash、模型配置 JSON 和固定输入 logits。对 sharded checkpoint，应记录 merge 脚本版本、输入 shard 清单、输出 artifact 路径、是否允许 world-size 转换、以及导出后是否还能反向追溯到训练 run id。这样，分布式训练章节才真正覆盖“可训练、可恢复、可评测、可发布”四个状态，而不是只覆盖前两个。

lit-llama 的预训练脚本提供了一个具体 FSDP 契约 [153]。它用 Transformer-block auto-wrap policy 把 `Block` 作为 wrap 单元，同时开启 block 级 activation checkpointing 和 all-gather 节流，再通过 Fabric 设定 `bf16-mixed` 精度。这个配置说明 wrap 单元、重算范围、all-gather 节流和参数 dtype 是同一套系统选择；若只写“用了 FSDP”，读者无法推导峰值显存、通信频率或恢复成本。

同一个脚本还展示了 FSDP 与 batch 语义的耦合。全局 `batch_size` 会先除以 device 数得到每进程 batch，再除以 `micro_batch_size` 得到 gradient accumulation 次数；训练循环用 `fabric.no_backward_sync` 在累积期间避免同步，只在真正 optimizer step 前裁剪梯度、更新参数并清零。验证、日志和 checkpoint 通过 `fabric.global_rank`、`fabric.print` 与 barrier 协调。出版级 run card 应把这些字段和实际 tokens/s/device、保存间隔、验证间隔放在一起。

Baichuan 2 展示了稠密多语言模型的一种实用混合布局：单机内做张量并行，跨机器做 ZeRO 式数据并行；对大词表 cross entropy 等显存压力高的计算做 tensor splitting；再配合拓扑感知 rank placement 和混合/分层参数切分，减少大规模集群通信压力 [251]。可迁移的不是某个固定集群配置，而是放置原则：层内张量并行尽量留在高速互联岛内，冗余状态在更大的数据并行组里切分，rank mapping 和检查点导出都属于训练配方。

7.5.5 Tensor Parallelism

张量并行切分层内矩阵，使单层计算分布到多张卡。它能容纳更宽模型，但引入频繁通信。是否值得使用取决于矩阵大小、互联带宽和 kernel 实现。

Megatron 风格的基本技巧是列切分和行切分：前者切输出列，各卡得到输出分片；后者切输入行，各卡算局部结果后需要求和。MLP 中，上投影常用列切分，各卡得到扩展 hidden 维度的一段，激活函数可以本地执行；下投影再用行切分，各卡计算回原 hidden 维度的局部结果，最后用 all-reduce 求和。Attention 的 Q/K/V 可以按 head 切分，输出投影再做行切分规约。这个细节很重要，因为张量并行的代价不是“用了几张卡”，而是每一

层前后究竟有 all-reduce、all-gather 还是 reduce-scatter。

OSLO 的 tensor model parallelism 教程把这一点进一步显式化：除 Megatron 风格 1D 行列切分外，还提供 2D、2.5D 和 3D tensor partitioning，并把张量并行 size、mode、depth 和进程组布局作为运行时配置 [252]。这些模式不是简单开关；它们会改变 collective 模式、检查点分片方式和可用 rank 组合。这个教程还暴露出几个工程约束：模型先在 CPU 上创建再切分，`tensor_parallel_size` 必须为正、不超过 GPU 数且通常要求为 2 的幂；`tensor_parallel_depth` 只对 2.5D 模式有意义；示例中的张量并行路径不和流水线并行混用。默认 `save_pretrained` 会写出带 TP、PP、EP rank 标识的分片检查点；若要得到普通推理栈可加载的权重，需要显式 merge。训练日志应记录张量并行 degree、mode、depth、data/pipeline/expert parallel degree、rank mapping，以及保存的是按 TP rank 分片的检查点，还是已经 merge 成普通推理栈可加载的模型。

PyTorch 较新的 tensor-parallel 教程把同一件事写成 `DeviceMesh`、`DTensor` 和模块级 parallel style [152, 262]。`DeviceMesh` 给进程组维度命名，因此二维 mesh 可以把张量并行放在单机高速互联内，把 FSDP 放在跨机器维度上，而不需要手写每个进程组。Parallelization plan 再把子模块映射到 `ColwiseParallel`、`RowwiseParallel`、`SequenceParallel` 或显式输入输出 layout 转换。这样 shape 契约会更清楚：Q/K/V 的 column sharding 会切 head 维度，因此 reshape、RoPE buffer 和 local head 数必须要么由 DTensor 感知，要么按 local shape 重写。如果模型在切分前连 CPU 内存都放不下，meta-device 初始化或逐层初始化就属于训练配方，而不是次要实现细节。

PyTorch 2.12 的 TP API 进一步把这张 plan 表具体化：模块并行化入口根据用户给出的 plan 把参数替换成 `DTensor`，运行时再按每个 style 的输入/输出 layout 插入 `allreduce`、`allgather` 或 `reduce_scatter` 等通信；`PrepareModuleInput`、`PrepareModuleOutput` 和 `PrepareModuleInputOutput` 可只负责 activation layout 转换，而不切分模块参数 [197]。因此 TP run card 应逐子模块列出 module path、parallel style、参数 placement、输入 placement、输出 placement、预期 collective、是否依赖偶数分片、以及该 layout 在 attention mask、RoPE、vocab shard 和 fused kernel 中的消费位置。否则一段代码即使能 forward/backward，也可能在局部 head 数、mask 广播或输出 gather 处悄悄改变语义。

这个 plan 表还应记录默认 layout 约定。`ColwiseParallel` 默认让输出沿最后一维分片，后续算子若假设完整 hidden 维或 vocab 维，就必须显式重写 local shape 或插入 gather；`RowwiseParallel` 默认消费最后一维分片输入，并常把输出还原成 replicated tensor；`SequenceParallel` 目前主要适用于 LayerNorm、Dropout 和 RMSNorm 这类兼容模块，且带权重的归一化层如果做过自定义初始化，需要在并行化前后广播以保持副本一致。这些细节决定了一个 Llama block 的 Q/K/V、RoPE、MLP 上投影、下投影、norm 和 loss 是否处在同一个 layout 语言里。出版稿不应只给出 `tp_size`，还应给出每条边的 DTensor

placement 和从分片输出回到普通 tensor 的位置。

序列并行、loss 并行和 context parallelism 应分开理解。序列并行通常让 Transformer block 的输入输出沿序列维保持分片形式，在归一化或 dropout 上节省激活显存，再在注意力与 FFN 前插入 layout 转换。Loss parallelism 则让大词表 logits 保持按 vocab 维切分，直接在分片 logits 上计算 cross entropy，避免每张卡聚合全量输出；PyTorch 的 `loss_parallel` 要求输入 logits 是沿 class/vocab 维分片的 `DTensor`，target 和 weight 按 mesh 复制，且当前不支持 label smoothing，API 仍标为 experimental [197]。Context parallelism 面向超长序列，把 Q/K/V 或其他序列 buffer 沿序列维切开，并用 ring attention 替代普通 scaled-dot-product attention；PyTorch prototype 暴露了 all-gather pass-KV 和 all-to-all pass-KV 两种旋转路径 [244]。RoPE 频率这类位置相关 buffer 必须纳入同一个 sequence sharding plan，否则张量形状可能都对，但 attention 数值已经错了。

张量并行最常见的线上问题是“形状正确但语义错”。Local head 数、RoPE offset、vocab shard、attention mask shard、loss normalizer 和 logits gather 位置都可能让训练继续运行，却让 loss 或评测慢慢漂移。调试时不要只看 step 是否能跑通，而要比较 TP=1 和 TP>1 在同一输入上的 logits 子集、loss、梯度范数和 checkpoint 导出结果。若使用编译器或 fused kernel，还要确认 layout 转换没有把通信重新串行化。

7.5.6 Pipeline Parallelism

Pipeline parallelism 按深度切分模型，让不同 stage 处理不同 microbatch。它能容纳更深网络，但会产生 pipeline bubble 和调度复杂度。microbatch 数、stage 平衡和激活传输决定效率。

PyTorch 的 pipeline parallelism 文档把这个问题拆成 split frontend 和 distributed runtime [196]。前者根据 specification 或 policy 切分模型并记录数据流关系，后者处理 microbatch 切分、schedule、通信和梯度传播。文档仍把该包标成 alpha，因此发布报告要同时记录 API 版本和 schedule 语义。GPipe、1F1B、interleaved 1F1B 和 Looped BFS 不是单纯的性能选项；它们会改变 activation 生命周期、optimizer step 时刻、跨主机通信、checkpoint 映射和调试日志的解释。

Pipeline stage 也有明确的 shape/dtype 契约。`PipelineStage` 负责分配通信 buffer、建立 send/recv 操作，并保存尚未被消费的中间输出；因此它必须提前知道 stage 输入输出的静态形状，运行时形状不匹配会触发 shape error。若流水线与 TP、FSDP、mixed precision 或 context parallel 组合，stage 边界的 dtype 和 layout 也会随之改变，不能只按原始模型的 tensor 形状填写。手工切分时，报告应记录每个 stage 保留或删除哪些层、是否用 meta-device 避免初始化 OOM、stage 输出 shape/dtype、microbatch chunk spec 和 backward 归属；自动切分时，还要记录 `torch.export` 可追踪性、split point、图分区策略和与其他并

行技术组合时的 workaround。

TorchTitan 的 Llama 训练栈展示了现代训练系统的组合压力：FSDP2、tensor parallel、pipeline parallel、context parallel、activation checkpointing、distributed checkpointing、float8、structured per-rank logging 和 checkpointable data loading 会同时出现 [229]。这说明出版级系统章节不应只列出“三维并行”，而应解释哪些维度在节点内、哪些跨节点、哪些组件负责 checkpoint，哪些日志能证明 loss、吞吐和恢复都处在预期状态。

简单 pipeline 的 bubble 可以粗略写成 $(S-1)/(M+S-1)$，其中 S 是 stage 数，M 是 microbatch 数。增加 microbatch 能降低空闲比例，但也会增加激活 bookkeeping，改变有效 batch 约束，并可能让 optimizer step 更晚发生。Interleaved schedule 可以让一个设备承担多个 stage chunk，改善负载平衡，但调度和 checkpoint 映射也更复杂。

stage balance 是关键。embedding、output projection、长上下文 attention 和 MoE layer 会让“相同层数”对应完全不同的运行时间。好的 partitioner 应按真实运行时间、峰值显存和 stage 间 activation 传输切分，而不是简单平均层数。Pipeline 还会增加调试难度：某个 stage 产生的 NaN 可能只在下游暴露；shape bug 可能只在 stage 边界出现；日志必须包含 stage rank、microbatch id 和 schedule step。

DualPipe 类 schedule 的核心动机是把 forward、backward 和通信更充分地重叠，尤其适合同时存在 pipeline 和 expert parallel 的稀疏模型。判断它是否有效，不能只看平均 step time，而要看每个 stage 的空闲比例、activation 传输、all-to-all 时间、microbatch 排队和最慢 stage。

流水线并行的 checkpoint 和日志也要按 stage 设计。每个 stage 应记录负责的层范围、参数 shard、activation 边界、microbatch 顺序和随机数状态；发生 NaN、OOM 或 shape mismatch 时，日志应能定位到 stage、microbatch 和 schedule step。若最终要导出无 pipeline 切分的模型，发布流程必须能按原始层顺序合并 stage 权重，并验证 tied weights、position embedding 和输出头没有重复或遗漏。

7.5.7 Expert Parallelism

Expert parallelism 服务于 MoE，把不同专家放在不同设备上。每个 token 经 router 分配到专家，通常需要 all-to-all 通信。负载均衡和专家利用率是核心监控指标。专家负载不是只靠一个平均利用率能看清。应记录每个专家收到的 token 数、路由熵、容量溢出、重路由或丢 token、all-to-all 时间，以及每个专家是否持续收到梯度。序列级负载均衡在长 prefill、batch size 为 1、或领域分布偏斜时可能表现不同，所以负载均衡机制本身也要在目标服务形态下评测。

MoE 的容量因子、top-k 路由、shared expert、token dropping 和 auxiliary loss 都会改变系统形态。容量因子太小会丢 token 或强制重路由，太大又会浪费显存和 all-to-all 带宽；

某些专家长期低负载可能说明路由塌缩，也可能说明数据域分布不均。训练 dashboard 应把语言、任务域、序列长度和专家负载关联起来看，否则平均 perplexity 可能掩盖了某个领域专家冷启动或过载的问题。

7.5.8 激活检查点与重算

激活检查点不保存部分中间激活，在反向传播时重算，从而用计算换显存。它适合显存瓶颈明显的训练，但会降低 tokens/s。报告应说明节省了多少显存、增加了多少计算。

重算的代价不是固定百分比。它会和 pipeline schedule、tensor-parallel collective、dropout 随机数状态、attention kernel 与 compiler graph capture 相互作用。若重算段包含随机操作，随机数状态必须可复现；若包含通信，重算增加的不只是算术，还有网络流量。实践中应按 memory profile 选择保存哪些 activation，而不是对所有 block 使用同一策略。

选择 checkpoint 范围时要比较“保存少量贵张量”和“重算大量便宜张量”的差别。归一化、dropout、attention score、MLP 中间激活、MoE dispatch buffer 和跨 rank layout conversion 的成本并不相同。一个看似节省显存的均匀 block checkpoint 策略，可能把通信重算、随机数恢复和 compiler graph break 引入反向传播。报告应给出开启前后的 peak memory、tokens/s、collective time、重算 FLOPs 估计和数值对齐切片。

7.5.9 精度与通信

BF16、FP16、FP8 和量化通信改变吞吐与稳定性。低精度能提高速度、降低带宽，但会引入溢出、下溢和缩放问题。通信压缩也必须检查最终质量是否受损。

FP8 报告不能只写“使用 FP8”。它应说明哪些张量以 FP8 存储、哪些 GEMM 使用 FP8 输入、哪些累积保持 BF16/FP32、缩放因子如何更新、哪些验证切片和 BF16 基线对齐。对 MoE 模型，还要检查 router logits、负载均衡损失、专家梯度和通信压缩是否因为低精度产生不稳定。通信原语还要区分语义和底层实现。All-reduce、reduce-scatter、all-gather、all-to-all、broadcast、scatter/gather 和点对点 send/recv 描述的是各 rank 最终看到什么张量；实际链路上可能用 ring、tree、分层节点内/节点间算法或 NVLink/NVSwitch 专用路径。Ring all-reduce 对大消息带宽友好，但小 bucket 可能受 latency 支配。非阻塞通信也不是自动加速：只有在 `wait` 之前安排了真正独立的计算，才有 overlap；同一 process group 中各 rank 仍必须按兼容顺序调用 collective，否则异步调度会变成死锁。

通信 overlap 应以时间线证据证明。一个 profiler trace 至少要能看到 backward compute、gradient bucket ready、reduce-scatter/all-reduce 启动、参数 all-gather/prefetch、optimizer step 和 checkpoint I/O 的相对位置。若所有 rank 在同一 collective 前等待，说明 overlap 没有发生；若小 bucket 过多，kernel launch 和 collective latency 会吞掉收益；若大

bucket 过少，通信开始太晚。报告 bucket size、通信 backend、拓扑、平均和 p95 collective 时间，比只写“开启异步通信”更有价值。

通信测试也应在真实拓扑和真实 batch 形状下运行。单独的 all-reduce microbenchmark 可以暴露带宽上限，却不能说明反向传播 bucket 何时 ready、哪个 process group 与 pipeline send/recv 竞争、checkpoint 上传是否抢占网络，或某个 rank 是否因为数据等待而错过 overlap 窗口。出版级结果应记录节点拓扑、NCCL 或通信后端版本、bucket 配置、每类 collective 的 p50/p95/p99 时间，以及 profiler 是否覆盖 warmup 后的稳定区间。

7.5.10 吞吐记账

吞吐不只是 tokens/s，还包括 step time、MFU、GPU 利用率、数据加载等待、通信时间和失败重启。高吞吐如果伴随质量下降或故障频繁，并不代表系统更好。正确比较应看达到目标质量所需总成本。

端到端 step time 包含数据加载、host-to-device 传输、forward、backward、optimizer step、communication、validation、checkpoint 和 restart。只测 GEMM 的 microbenchmark 对 kernel 开发有用，但不能规划训练成本；反过来，一个低 tokens/s 数字也不能直接说明瓶颈在哪里。profile 至少要分开 data time、compute time、collective time、optimizer time 和 I/O time。

同步训练会被最慢 rank 拖住。慢 dataloader worker、降频 GPU、拥塞网络路径或文件系统暂停，都可能让整轮训练吞吐下降。日志若只记录 master rank 的平均 step time，straggler 会长期隐藏。出版级系统报告应给出 per-rank timing 或至少给出慢 rank 诊断方式。

吞吐 dashboard 不应只服务于调参，还应服务于归因。最小字段包括每 step token 数、samples/s、input pipeline wait、forward、backward、optimizer、collective、checkpoint、validation、显存峰值、分配失败次数、重试次数和每 rank 温度/功耗。若质量曲线同时记录 loss、eval metric、梯度范数、学习率和数据位置，就能判断吞吐变化是否伴随优化状态变化。没有这些字段，训练慢下来时只能猜是网络、数据、kernel、checkpoint 还是某个 rank 被拖慢。

性能回归验收应同时有短跑 profile 和长跑漂移检查。短跑能发现 kernel、bucket 和 stage balance 的直接问题；长跑才会暴露文件系统缓存失效、远程数据 backpressure、GPU 温度降频、checkpoint 队列堆积、验证作业挤占资源和慢 rank 的尾部效应。若一个系统只在前几百 step 报告 tokens/s，却没有两小时、一天或一次完整 checkpoint 周期后的吞吐曲线，读者无法判断它是否适合真实预训练。

7.5.11 运行可靠性

大规模训练必然遇到节点故障、网络抖动、文件系统瓶颈和坏样本。可靠系统需要 checkpoint 恢复测试、日志归并、错误隔离和监控告警。不能恢复的训练系统不适合长时间运行。

可靠 runbook 应说明单 rank 失败、重复 OOM、NaN、慢 checkpoint 写入、网络降级和 validation regression 时分别如何处理。Data loader 也属于可靠性边界：tokenized shard 应有 checksum、长度和格式版本；sampler 应能按 token 或 shard 位置恢复；若各 worker 独立 shuffle 而没有统一 seed 与 epoch 定义，恢复后可能重复或跳过数据。

PyTorch Distributed Checkpoint 把 checkpoint 从“保存一个文件”变成显式的分布式协议 [195]。State dict 可以来自 FSDP、`fully_shard`、DDP、tensor parallel 或它们的组合，但仍要说明 storage writer、planner、metadata、进程组和 load path。异步保存也不能被写成“免费”。`async_save` 会区分 staging completion 和 upload completion；默认路径可能仍在主线程把状态从 GPU 拷到 CPU，再由后台线程写存储。较新的 staging helper 可以把这段 D2H copy 移到后台线程，但 runbook 仍要预算 CPU 内存、未完成写入、同步点和上传中断时的恢复策略 [194]。

DCP 的 API 还会影响发布格式。`storage_writer` 决定 checkpoint id 如何映射到文件系统或对象存储，`planner` 决定 state dict 条目如何分片、放置和写入，`process_group` 决定哪些 rank 协调保存；若关闭跨 rank collectives 或走单 rank 保存，文档也把这类格式变化标成需要谨慎处理的实验路径。异步保存返回的 future 不是日志装饰品：runbook 应限制并发异步 checkpoint 数，等待上一轮 staging 后再允许 optimizer 修改同一份参数，等待 upload 后再把 metadata 标记为 durable，并在最后一个 checkpoint 完成后关闭异步写入器。否则训练可能继续前进，但最近一次“可用 checkpoint”并不一定真的可加载。

实验追踪要区分高频指标和 durable artifact。高频指标包括 step time、loss、gradient norm、显存和通信时间；durable artifact 包括 config、data manifest、评测报告、checkpoint 和环境描述。前者帮助在线排障，后者支撑事后科学解释。

Accelerate 类训练器常把这些 artifact 分散在多个接口里：tracker 记录训练配置和分布式配置，`save_state` 保存 optimizer/scheduler/model 状态，`save_pretrained` 负责发布形态，PEFT 还可能需要非严格加载或额外保存 adapter。可靠性审计要检查这些路径是否引用同一个 checkpoint 目录、同一个 tokenizer 和同一个代码版本。若 tracker 里缺少 git ref、DeepSpeed stage、offload 设备或真实进程数，事后很难解释同一脚本为什么在另一台机器上 OOM 或质量不一致。

Accelerator API 还把梯度同步、数据恢复和分布式评测分片暴露为运行时接口。`no_sync` 会临时关闭 DDP 梯度同步，`trigger_sync_in_backward` 用于多个 forward 后在下一次 backward 强制同步；`skip_first_batches` 可把 dataloader 移到恢复位置；

`split_between_processes` 能把评测输入分到不同进程，若为了 gather 采用 padding，还要在聚合后丢弃重复尾部样本 [60]。因此 run card 应记录同步边界、跳过 batch 数、padding/去重规则和每 rank 输入分片。否则一次恢复或分布式评测看似成功，实际可能重复样本、漏样本，或在错误 micro-step 同步梯度。

数据管线恢复是可靠性的一部分。Tokenized shard 应记录 checksum、样本数、token 数、格式版本和构建命令；sampler 应记录 epoch、shard offset、sample offset 或 token offset；断点恢复后应能证明没有大段重复或跳过。远程存储还要记录 cache 命中率、预取队列长度、读取错误、重试和 backpressure。很多“GPU 利用率低”的问题，本质上是数据和 I/O 没有按训练规模设计。

混合语料 dataloader 也要能恢复。RedPajama 类脚本常把 arXiv、book、C4、Common Crawl、GitHub、StackExchange 和 Wikipedia 等来源按权重合成一个 `CombinedDataset`，再用 `PackedDataset` 按 world size 与 global rank 切分读取。这里的来源权重、文件 glob、chunk 数、shuffle seed、rank-local cursor 和 effective block size 都属于训练状态。若 checkpoint 只保存模型权重而不保存数据混合状态，恢复后的 loss 曲线可能连续，数据分布却已经改变。

可靠性 runbook 应在启动前演练，而不是事故后补写。一次长训练至少需要 launch manifest、不可变代码版本、环境快照、数据 manifest、checkpoint cadence、健康检查、告警阈值和停止规则。Runbook 应分别写明单 rank 失败、重复 OOM、NaN、文件系统写入变慢、网络链路降级、验证集突然回退和数据 shard 损坏时的处理路径。若团队没有测试过从最近 checkpoint 恢复并继续 loss 曲线，所谓 fault tolerance 只是配置项，不是已验证能力。

故障演练要保留可复查证据。最小演练不是简单重启一次，而是在固定 step、固定数据 shard 和固定随机种子下杀掉一个 worker，确认调度器停止或重排作业、最近完整 checkpoint 被选中、partial write 被丢弃或隔离、optimizer/scheduler/token count/sampler offset/RNG state 全部恢复，并在恢复后的若干 step 内让 loss、gradient norm、tokens/s 和 per-rank 显存回到预期区间。若 checkpoint 采用异步上传，还要记录 staging 完成时间、durable metadata 写入时间、后台上传失败处理和下一次启动选择哪个 checkpoint。

导出验收也应进入训练系统章节。训练能恢复不代表权重可发布：FSDP、ZeRO、tensor parallel、pipeline stage、expert shard、LoRA adapter、reward/value head 和 tied embedding 都可能让训练 checkpoint 与推理 checkpoint 结构不同。发布前应运行固定脚本完成 shard merge 或 reshard，校验参数数目、dtype、tied weights、tokenizer special token、chat template hash、adapter 合并策略和位置编码配置，再用固定 prompt 比较导出前后的 logits 子集或短生成。这个验收把“保存成功”升级为“可恢复、可评测、可加载、可发布”。

7.5.12 实现笔记

实现分布式训练时，应先在单机验证数值，再逐步加入数据并行、FSDP、tensor parallel 和 pipeline。每加一层并行，都要重新检查 loss、梯度、吞吐和 checkpoint。并行不是越多越好，而是为具体瓶颈服务。

最小分布式脚本也隐藏很多契约：环境变量定义 rank、local rank、world size、master 地址和通信 backend；sampler 必须避免非预期重复样本；gradient accumulation 下 loss scaling 必须正确；随机种子在初始化、数据顺序和 dropout 上要有明确语义；共享 artifact 只能由指定 rank 写入，但所有 rank 又必须参加 checkpoint 协议需要的 barrier。每次扩展规模前，都应做单 GPU 与多 GPU 同 global batch 的 loss 对比、有限梯度检查、峰值显存分类统计、保存恢复测试和目标序列长度 profile。

分布式评测也要按训练系统的标准处理。多卡评测脚本常用 `torchrun` 启动，初始化 NCCL 后从 rank 和 world size 划分样本，例如让第 i 个样本只由满足 $i \bmod W = r$ 的 rank 处理；每个 rank 写自己的 JSONL，目录创建和写入前后用 barrier 协调，最后 gather count、reward、accuracy 或 tree-search 指标。报告应给出每 rank 样本数、总样本数、跳过/重复样本检查、聚合公式和随机生成参数。否则评测吞吐看似提高，实际可能是样本漏跑、重复统计或只有 rank 0 的日志被当成全量结果。

可发表的系统设计清单至少应暴露 data/tensor/pipeline/expert parallel degree、每 rank 峰值显存、每 step collective 时间、checkpoint 或数据加载 I/O 时间，以及故障后的实测恢复时间。这些不是次要工程细节，而是决定另一个团队能否复现同一训练配方的核心证据。

最小回归套件应覆盖四类场景：数值等价、资源边界、恢复导出和性能归因。数值等价包括单卡/多卡同 global batch 的 loss 与 logits 对比；资源边界包括峰值显存、OOM 降级和 headroom；恢复导出包括 save/resume、world-size 转换、sharded 到 full checkpoint 合并和推理加载；性能归因包括短跑 profile、长跑吞吐漂移、straggler 注入和 checkpoint 写入压力。只有这些测试存在，分布式训练系统才算是可维护的研究基础设施。

这些回归不必一开始覆盖最大模型，但必须覆盖同一类接口。小模型可以验证 loss normalizer、rank-local sampler、FSDP wrap、TP shape、PP stage 边界、expert routing、checkpoint planner 和导出脚本；中等规模试运行再验证真实显存、通信和 I/O 压力。若只在玩具配置验证数学正确性，而把真实并行布局留到正式训练首次运行，失败时就很难区分是算法问题、配置问题还是集群问题。

可以把训练系统报告压缩成一个向量：

$$\mathcal{S}_{\text{train}} = (P_{\text{dp}}, P_{\text{tp}}, P_{\text{pp}}, P_{\text{ep}}, M_{\text{peak}}, C_{\text{step}}, T_{\text{io}}, R_{\text{restart}}).$$

其中四个 P 分别是数据、张量、流水线和专家并行度，M_{peak} 是每 rank 峰值显存，C_{step}

观察到的症状	可能根因	优先检查证据
启动即 OOM	参数、优化器状态或初始化峰值超过预算	持久状态估算、分片包裹粒度、惰性初始化、显存余量
长上下文 OOM	激活、attention 临时量或通信 buffer 主导	激活剖析、重算范围、序列/上下文并行、bucket 大小
吞吐随时间下降	数据、网络、温度、checkpoint 或慢 rank 拖慢同步 step	每 rank 步时、数据等待、collective p95、GPU 温度、I/O 队列
恢复后 loss 跳变	优化器、scheduler、数据游标或随机数状态未完整保存	state dict、sampler offset、学习率步数、token 计数、恢复短跑曲线
训练能跑但导出失败	分片布局、共享权重、adapter 或 tokenizer 契约缺失	分片元数据、合并脚本、固定提示加载测试、模板 hash
MoE 尾延迟很慢	专家负载不均或 all-to-all 拥塞	每专家 token 数、路由熵、容量溢出、all-to-all 时间线

表 7.1: 分布式训练症状、可能根因与优先检查证据检查表。

是每步 collective 时间，T_{io} 是 checkpoint 或数据加载 I/O 时间，R_{restart} 是故障后的实测恢复时间。这个向量不能替代完整报告，但能迫使作者说明训练结果依赖哪些系统条件。

系统诊断应从症状回到可测量原因。OOM、tokens/s 下降、loss 不连续或 checkpoint 无法导出，通常不是单个开关能解释的问题，而是内存、通信、数据、精度和恢复契约共同作用的结果。下面的表把常见症状映射到优先排查项，帮助读者把“训练慢了”或“保存失败了”改写成可以验证的工程假设。

表 7.1 把分布式训练症状映射到证据项，便于把系统事故写成可验证假设。

7.5.13 发布前验收矩阵

分布式训练章节最后应落到验收，而不是只停在策略名称。一个训练系统能跑通，不等于它适合写进论文、书稿或模型发布说明。出版级验收至少要覆盖六类证据：并行布局、数值等价、恢复能力、导出能力、吞吐归因和分布式评测。每一类证据都应有明确输入、命令、日志、artifact 和失败处置。

并行布局验收要先固定 topology contract。报告应写明节点数、每节点 GPU 数、global rank 到 node/GPU 的映射、data/tensor/pipeline/expert/FSDP 维度、每个 process group 的用途，以及哪些 collective 留在节点内高速互联。若使用 `DeviceMesh`、FSDP2 或 TorchTitan 类组合栈，还要把 mesh 维度、`fully_shard` 包裹顺序、pipeline stage、context parallel 维度和 checkpointable dataloader 状态放进同一张表 [198, 229]。这样，读者才能判断一次训练结果依赖的是算法、硬件拓扑、rank placement，还是框架默认值。

数值等价验收应从小规模开始。固定 tokenizer、数据 batch、随机种子和 global batch

后，先比较单卡、DDP、FSDP/FSDP2、tensor parallel 或组合并行的 logits 子集、loss、梯度范数和一次 optimizer update 方向。TP 版本还要单独比较 plan 中每个 layout 边界的本地张量形状、placement、vocab/class 维 normalizer、`loss_parallel` 输出和导出后 logits；这能把“通信/切分正确”与“语言模型语义正确”分开。允许 BF16、FP8 或通信压缩带来小的数值差异，但这些差异必须有容忍阈值和验证切片。若多卡版本 loss 能下降却无法和单卡基线在同一输入上对齐，后续大规模训练的质量变化就无法归因。

恢复和导出要分开验收。恢复验收关注训练能否继续：checkpoint 后重启，optimizer state、scheduler step、token count、sampler offset、RNG state、data-mixing state 和 per-rank 显存应连续。导出验收关注权重能否被其他系统消费：sharded `DTensor`、FSDP、ZeRO、TP、PP、EP、LoRA adapter、reward/value head 和 tied embedding 都要合并或重分片成明确的发布形态，并用普通推理脚本加载固定 prompt [195]。异步 checkpoint 还要记录 staging 完成、upload 完成、durable metadata 和 partial write 清理规则；否则一次“保存成功”可能只说明数据进入了暂存区，并不说明下一次启动能安全恢复。

吞吐归因验收要证明性能数字来自目标系统，而不是短跑假象。至少同时记录 warmup 后稳定区间和跨 checkpoint 周期的长跑区间；字段包括 tokens/s、step time、data wait、forward、backward、optimizer、每类 collective、checkpoint I/O、validation、GPU 温度/功耗、显存峰值和每 rank p95/p99。若平均 tokens/s 提升但慢 rank、checkpoint 队列或数据 backpressure 同时恶化，这个系统并没有可靠提升。出版稿应把吞吐和质量目标绑定：固定 token 预算、固定质量阈值和固定 wall-clock 成本下分别报告结果。

分布式评测也需要验收矩阵。每个样本应有全局唯一 id、rank 分配规则、per-rank JSONL 输出、断点续跑策略、barrier、聚合脚本和重复/漏跑检查。生成式评测还要记录采样参数、seed、stop rule、模型裁判版本和人工校准集。一个 8 卡评测任务如果只收集 rank 0 日志，或恢复后重新评了部分样本，就会把吞吐和指标都写错。训练章节把这点说清楚，是为了提醒读者：分布式系统错误同样会污染 benchmark。

验收域	必备证据	不通过时的处理
Topology	rank mapping、mesh、process group、并行度	固定布局后重跑 dry run
数值等价	单卡/多卡 logits、loss、梯度与 update 对齐	缩小到一层或一个 batch 定位
恢复	optimizer、scheduler、token、sampler、RNG 连续	丢弃 partial checkpoint，回退到完整版本
导出	shard merge、dtype、tied weights、tokenizer、固定 prompt	修正导出脚本并重新验收
吞吐归因	per-rank timing、collective、I/O、温度、长跑漂移	区分数据、网络、kernel 和 checkpoint 瓶颈
分布式评测	样本 id、rank 输出、聚合、漏跑/重复检查	重建评测清单并重新聚合指标

7.6 关键术语、实现要点与练习

关键术语。 Data parallelism 复制模型并对应梯度；FSDP/ZeRO 切分参数、梯度和优化器状态；FSDP2 用 `fully_shard`、`DTensor` 和 hook 明确分片参数何时 all-gather、何时重新分片；Distributed Checkpoint 用分布式 state dict、planner、storage writer 和 metadata 保存并恢复分片训练状态；asynchronous checkpointing 把 staging 和 upload 拆开，允许与训练重叠但需要跟踪未完成写入；process group 定义某个 collective 涉及的 rank 集合；rank mapping 指 rank 的节点、GPU、stage、expert 或 shard 放置；process rank 是训练进程在设备集合中的身份；global batch 由 micro-batch、数据并行度和 gradient accumulation 共同决定；no-sync 表示累积期间暂缓梯度同步；sync boundary 指触发跨 rank 梯度同步的 backward 或 optimizer step；all-reduce 让所有 rank 得到规约后的完整结果；all-gather 把分片聚成完整张量；reduce-scatter 把规约结果分回各 rank；ZeRO Offload 把部分状态和计算移到 CPU 侧；Accelerate config contract 记录 plugin/config-file、launcher、offload、同步和导出；DeepSpeed run card 记录 ZeRO stage、offload、bucket、精度、world size、batch 和展开后的 `auto` 字段；tensor parallelism 切分层内矩阵；tensor-parallel plan 把子模块、parallel style、placement 和 layout 转换写成可审计表；张量并行检查点按 TP rank 保存张量分片，恢复或导出依赖布局元数据；loss parallelism 在分片 vocab/class logits 上计算交叉熵；sequence parallelism 沿序列维切分部分激活；context parallelism 面向长上下文切分序列状态；activation checkpointing 用反向重算换取激活显存；communication overlap 把通信安排到计算背后而不是串行执行；pipeline bubble 是 pipeline stage 填充和排空造成的空闲时间；pipeline parallelism 切分网络深度；expert parallelism 支持 MoE 专家路由；rank-local data shard 表示某 rank 实际读取的数据分片和游标；sharded state dict 保存

分片权重和布局元数据；canonical checkpoint 是普通推理或后训练栈可加载的标准权重；distributed evaluation contract 规定样本分片、per-rank 写入、barrier 和指标聚合；launch manifest 记录代码、环境、数据、并行度和恢复策略；straggler 是同步训练中拖慢全局 step 的慢 rank 或慢组件。

实现要点。所有并行策略都要说明节省哪些字节、增加哪些通信、如何 checkpoint、如何恢复；DDP/FSDP/DeepSpeed 报告应记录 launcher、解析后的配置、rank 环境变量、world size、local rank、gradient accumulation 缩放、no-sync 规则、loss normalizer、master-only 写入规则和数据游标；FSDP2 报告还要记录 `fully_shard` 自底向上包裹顺序、通信组、是否调用 `model(input)` 触发 hook、optimizer 创建位置和 DTensor 导出路径；Tensor parallel 报告应记录 `DeviceMesh` 维度、每个子模块的 `ColwiseParallel/RowwiseParallel/SequenceParallel` 或 prepare-style layout、通信边界、local head/vocab 形状、`loss_parallel` 条件、checkpoint merge 和导出 logits 验证；pipeline 报告要记录 stage 输入输出 shape/dtype、microbatch spec、schedule 和 stage 权重合并规则；Accelerate/DeepSpeed 审计还要记录 plugin 或 config-file 路径、offload 介质、NVMe path、多节点 launcher、FP8/MS-AMP 开关、运行时展开后的 `auto` 字段、sync boundary、跳过 batch 数和 padding/去重规则；后训练任务还要分开记录 policy、reference、reward、value head、adapter 和导出路径；分布式评测要记录样本分片、per-rank 输出、barrier、gather 和漏跑/重复检查；吞吐报告应包含 tokens/s、step time、MFU、GPU 利用率、显存和失败恢复；系统比较应固定质量目标。

练习。

1. 估算一个 7B 模型在 AdamW 下参数、梯度和优化器状态的显存。

2. 解释为什么 ZeRO-3 能让模型放进显存，但不能自动减少单个 dense layer 的矩阵乘法计算量。什么时候仍然需要 tensor parallelism?

3. 比较 FSDP 与 tensor parallelism 的通信模式。

4. 一个 tensor-parallel 库按 TP、PP 和 EP rank 保存 checkpoint。列出恢复训练需要的元数据，并解释为什么推理导出前可能还要 merge。

5. 为一个 2D `DeviceMesh` 上的 Llama block 写 tensor-parallel plan 表，至少列出 Q/K/V、输出投影、MLP 上投影、下投影、归一化和 loss 的 placement、collective 与验收样例。

6. 解释 pipeline bubble 产生的原因，并给出减少方法。

7. 为 MoE 训练列出 all-to-all、负载均衡和专家冷启动风险。

8. 为一个 pipeline+expert parallel 的稀疏模型设计监控面板，判断 DualPipe 类 overlap 是否真的提升端到端吞吐。

9. 一个 13B dense 模型使用 BF16 参数、BF16 梯度、FP32 Adam moments 和 FP32 master weights。估算持久训练状态显存，并比较 ZeRO-1、ZeRO-2、ZeRO-3 在 16 个 rank 下的每 rank 状态量。

10. 128 张 GPU 分布在 16 个节点、每节点 8 张 GPU。为 dense 模型选择 data、tensor 和 pipeline parallel degree，并说明哪些 collective 应尽量留在节点内。

11. 某训练任务 loss 稳定，但两小时后 tokens/s 下降 25%。列出至少六个日志或指标，用来区分数据加载、网络、checkpoint、温度降频和 straggler 问题。

12. 设计一个从 sharded training checkpoint 到推理可加载 checkpoint 的导出流程，列出需要合并或校验的权重、tokenizer、chat template、adapter、score head 和 tied weights。

13. 为一次 1000 GPU 小时的预训练 dry run 写出最小 runbook：启动前检查、在线监控、单 rank 故障处理、checkpoint 恢复测试、数据 shard 校验和停止条件。

14. 用 $\mathcal{S}_{\text{train}}$ 向量描述一个训练计划，并说明哪个分量最可能成为复现实验的瓶颈。

15. 设计一次 communication overlap 实验，要求用 profiler 证明 reduce-scatter 或 all-gather 没有和主计算串行化。

16. 对照一个 DDP 脚本和一个 FSDP 脚本，列出 rank 初始化、梯度累积、同步跳过、checkpoint 写入、数据切分和导出路径各自必须进入 run card 的字段。

17. 对照一个 DeepSpeed/Accelerate 配置，写出解析后的 run card，说明哪些字段会改变显存峰值，哪些字段会改变 optimizer update 语义。

18. 对照一个 Accelerate DeepSpeed plugin 配置和一个 DeepSpeed JSON，列出 `zero_stage`、offload、bucket、multinode launcher、effective batch、`auto` 字段展开值、sync boundary 和 checkpoint/export 证据。

19. 设计一个 8 卡分布式评测流程，要求每个样本只评一次、每个 rank 可恢复写入、最终指标能证明没有样本漏跑或重复统计。

20. 写一个 FSDP2 run card，要求包含 device mesh 维度、`fully_shard` wrapping 顺序、optimizer 创建位置、checkpoint API、reshard/export 路径，以及如何验证 `model(input)` 触发了必要的 all-gather hook。

21. 比较 GPipe 与 1F1B 两种 pipeline schedule。说明哪些指标能证明 schedule 提高了吞吐，同时没有破坏 activation 生命周期、optimizer step 时序和 checkpoint 重构。

22. 设计一次异步 Distributed Checkpoint 故障演练：区分 staging completion、upload completion、durable metadata 和 partial write，并说明下一次启动应从哪个 checkpoint 恢复。

7.7 结构化检查表

7.7.1 并行策略诊断表

策略	节省什么	增加什么
数据并行	提高 batch 吞吐	梯度对应、模型状态重复
FSDP/ZeRO	参数、梯度、优化器显存	gather/scatter、checkpoint 复杂度
Tensor parallel	单层矩阵显存和计算	层内通信、拓扑敏感
Pipeline parallel	单卡容纳深层模型	bubble、调度复杂度
Expert parallel	MoE 专家容量	all-to-all、负载均衡、冷专家
激活检查点	激活显存	重算时间、确定性要求
数据管线恢复	保证 token 顺序和复现实验	shard checksum、sampler offset、I/O backpressure
Checkpoint/export	支持恢复和发布	规模元数据、分片合并、推理加载校验

第八章　推理与服务

8.1　Prefill 与 Decode

推理服务不是“关闭 dropout 后跑模型”。Prefill 处理 prompt，代价随输入长度增长；decode 逐 token 生成，代价随输出长度和 batch 调度变化。真实服务必须处理流式输出、取消、超时、缓存、租户隔离和安全过滤。首 token 延迟主要受 prefill、排队和调度影响；后续 token 速度主要受 decode、KV cache 和 batch 形状影响。长文档总结、短聊天、代码补全、RAG 和 agent 工具循环对服务系统提出的压力完全不同。如果所有请求进入同一个 FIFO 队列，长上下文请求可能阻塞短请求；如果过度偏向短请求，长任务可能永远得不到足够资源。

自回归生成把服务拆成两个形态不同的阶段。给定长度为 T_p 的 prompt 和长度为 T_g 的输出，模型实际计算的是

$$p_\theta(y_{1:T_g} \mid x_{1:T_p}) = \prod_{t=1}^{T_g} p_\theta(y_t \mid x_{1:T_p}, y_{<t}).$$

Prompt token 可以在 prefill 阶段并行处理；输出 token 却有串行依赖。于是只报告“每秒生成 token”会隐藏 prompt 成本，只报告端到端延迟又会隐藏 decode 吞吐。一个可发布的服务报告至少要分开给出 TTFT（time to first token）、TPOT（time per output token）、端到端延迟、input tokens/s、output tokens/s、排队时间和取消率。TTFT 变差可能来自排队、长 prompt、chunk-prefill 策略或安全过滤；TPOT 变差常常来自 KV cache 带宽、batch occupancy、采样逻辑或流式 flush。

Prompt cache 和 prefix reuse 只能在 token 序列、模型版本、位置编码状态和系统提示完全一致时复用。共享 system prompt、检索证据前缀或多轮对话摘要可以改善首 token 延迟，但错误的 cache key 会把一个用户的上下文泄漏给另一个用户，或者在检索证据更新后继续使用旧前缀。因此，cache 命中率不是越高越好；它必须和权限、版本、租户边界和证据新鲜度一起报告。

服务仪表盘也要按阶段分层。请求到达后先经历 admission、排队、prefill、首个 decode、连续 decode、后处理和流式 flush；每一段都应有单独计时。只看 GPU utilization 容易误

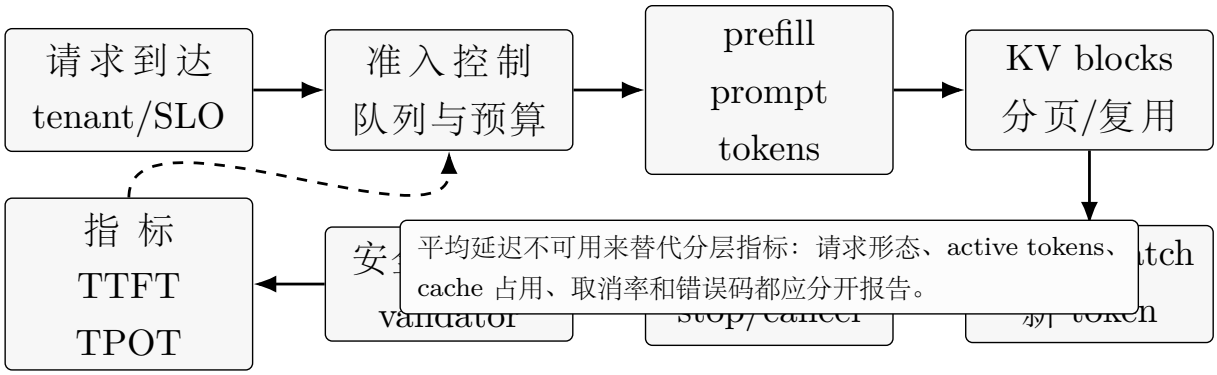

图 8.1: 推理服务是资源控制系统。PagedAttention/vLLM 的 KV-block 抽象启发了缓存管理，但服务闭环还必须包含准入、调度、流式输出和安全校验 [141]。

判：GPU 很忙可能是有效 batch，也可能是长 prompt、重复重试、安全分类器和工具等待把队列拖长。可审计的报告应给出每类流量的 TTFT、TPOT、E2E、active tokens、KV block 占用、拒绝率、取消率和错误码分布。

不同流量形态应先分开诊断，再合并看总体成本。短 prompt 短输出的瓶颈常在排队、kernel launch 和流式 flush；长 prompt 短输出主要压 prefill attention、tokenization、检索证据拼接和安全预检；短 prompt 长输出主要压 decode 带宽、KV cache 驻留和采样逻辑；多轮会话的风险在于 active token 越积越多，取消或超时后 cache 没有释放；共享 system prompt 的场景则要看 prefix cache 命中是否真实、是否跨租户安全。把这些流量混在一个平均 latency 里，会让优化方向失真。

因此，容量评估最好先定义 request class，再为每类写单独的准入规则和 SLO。短聊天可以优先保护 TTFT，长 RAG 可以限制 active token 和证据数量，代码生成可以限制最长 decode 与工具重试，agent 循环可以限制 tool step 和总 wall time。最终仪表盘再合并总体成本，而不是用总体平均值替代分层判断。

图 8.1 展示了推理服务中请求、prefill、decode、KV cache、流式输出和安全校验之间的闭环。它强调服务质量是资源控制问题，不是只把模型切到 evaluation mode。

表 8.1 将服务流量按瓶颈拆开，避免平均延迟掩盖真实容量问题。

图 8.2 把同一组方法放回服务瓶颈地图：一个加速器是否有用，取决于它命中的瓶颈是否正是当前流量的主导瓶颈。

服务接口还要明确 sampling contract。Temperature、top-k、top-p、logit bias、repetition penalty、stop sequence 和 seed 的应用顺序会改变输出；流式接口如果按 UTF-8 字节、token 或句子边界发送，也会改变客户端看到的部分结果。若产品承诺可复现，报告应说明相同请求在相同模型、模板、seed、采样参数和硬件后端下是否 bitwise deterministic，或者只承诺统计稳定。

流量形态	主要瓶颈	首先观察的指标
短聊天	排队、launch、stream flush	requests/s、TTFT p95、取消率
长 prompt QA	prefill compute、attention I/O、检索拼接	input tokens/s、TTFT、prefill queue depth
长输出生成	decode bandwidth、KV cache、采样	TPOT、output tokens/s、active tokens
多轮会话	cache 驻留、历史截断、租户隔离	KV blocks、cache lifetime、stop reason
共享前缀 RAG	prefix cache、证据版本、权限边界	cache hit rate、证据版本错配、复用失败率
混合优先级	调度策略、期限、反压	按租户/优先级切分的 p95/p99 和拒绝率

表 8.1: 推理服务流量形态、主要瓶颈与优先观察指标检查表。

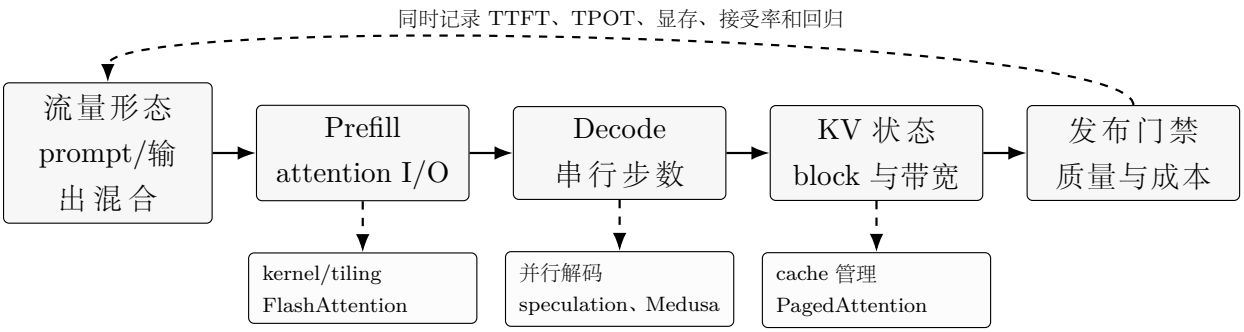

图 8.2: 推理加速器应按瓶颈选择，而不是按名称选择。该图综合 FlashAttention、PagedAttention/vLLM、speculative decoding、Medusa 和 Lookahead decoding 的服务契约 [28, 141, 146, 15, 41]。

结构化流式输出还要单独写 contract。JSON、tool-call 和 function argument 可以在生成中途处于不完整状态，客户端看到的是片段而不是最终对象。服务端应记录首个结构化片段时间、schema validator 失败率、repair 次数、工具参数撤回次数和最终提交时间；否则一个看似很快的 streaming API 可能只是把校验失败延迟到用户可见之后。

本地 Hugging Face、vLLM 和 lit-llama 推理示例说明，demo 能运行不等于服务契约完整。一个 HF 生成脚本往往同时包含 base checkpoint、LoRA adapter、tokenizer 路径、prompt template、NTK scaling、8-bit 加载、device map、EOS/PAD id 和 generation config。它还可能在 tokenizer 词表大于 base embedding 时 resize embedding。若这些字段没有进入发布记录，后来换 tokenizer、换 adapter 或换 prompt wrapper 时，输出变化会被误归因于模型能力，而不是接口改变。

vLLM 式批量生成示例通常把共享 prefix 和多个 prompt 放在同一批里，再用 sampling 参数控制输出。这很适合展示吞吐，但它仍缺少生产服务必须有的 admission、租户边界、prefix-cache key、取消释放和错误语义。温度设为 0 也不自动等于可复现服务：如果停止规则、tokenizer、chat template、kernel、随机种子或并发调度发生变化，同一请求仍可能产生不同 trace。出版稿应把“批量 demo”与“在线服务”分开说明。

最新 Transformers continuous batching 文档把这层区别写得更具体：批处理不是一次性把多个 prompt pad 到同一长度，而是在每个 generation step 重新调度，已完成请求离开 batch，新请求立即加入；服务式用法还要支持 `add_request`、`cancel_request`、streaming iterator、per-request temperature/top-k/top-p 和 `ContinuousBatchingConfig` 中的 KV 预算、block size、scheduler 与 prefix cache 选项 [65]。因此，continuous batching 的正确性不仅是“吞吐更高”，而是请求生命周期可追踪：每个 request id 的输入 token、采样参数、状态、部分输出、取消标志、stop reason 和 cache 释放都能被记录。

Transformers 的 continuous batching architecture 文档还把内部状态机写成 pending、prefilling、decoding 和 finished 四态：只有 token budget 与 cache budget 都允许时，请求才从等待队列进入 prefill；prompt 超过 `max_batch_tokens` 时，chunked prefill 会把长 prefill 拆成多个 step，并与已有 decode 交错 [66]。报告因此要记录 scheduler 类型、`max_batch_tokens`、block size、block 数或显存比例、cache 安全余量、是否允许 block sharing、cache 满时选择 offloading 还是 soft reset。soft reset 会把已生成 token 并回原 prompt、释放 KV block、再把请求放回队列；这不是普通重试，必须记录 stop reason、重排次数、额外 prefill 成本和用户可见延迟。

vLLM 的 serving 配置也应进入 run card，而不是只保留启动命令。当前官方 CLI 文档把 `kv_cache_memory_bytes`、`kv_cache_dtype`、prefix caching、prefix-cache hash、`max_num_batched_tokens`、`max_num_seqs`、chunked prefill、partial prefill 并发、long-prefill 阈值和 FCFS/priority scheduling 都显式暴露为服务参数 [236]。这些字段会改变容量、尾延迟、可复现性和安全边界。尤其是 prefix-cache hash：更快的非密码学 hash 可能带来碰撞和跨租户泄漏风险；若选择它，报告必须说明风险接受条件、租户隔离、证据版本和命中异常报警。

vLLM 的 prefix caching 设计进一步说明 cache key 不能只看文本前缀。每个完整 KV block 的 hash 由父 block hash、当前 block token 和额外 hash 组成；额外 hash 可以包含 LoRA id、多模态输入 hash 和多租户 cache salt [235]。因此，多模态请求中的图像占位 token 不能单独作为复用依据，还要绑定图像处理后的 hash。`sha256_cbor` 这类可复现 hash 适合跨环境确定性审计；非密码学 hash 即使更快，也要写明碰撞风险、租户隔离和泄漏监控。

PD 分离也不应被写成“天然更快”。vLLM 的 disaggregated prefilling 文档把主要动机

描述为分别调 TTFT 与 ITL、控制尾部 ITL，并明确提醒它本身不提升吞吐 [234]。这对书稿很重要：若实验只报告 tokens/s，就可能误判 PD 分离；发布报告应同时给出 prefill queue、decode queue、KV transfer latency、handoff 成功率、跨池 backpressure、版本一致性检查和失败后是否重新 prefill。PD 分离真正改变的是资源隔离和尾延迟治理，而不是消除 prefill 或 decode 的计算。

TGI 的官方文档提供了另一套服务验收口径。Launcher 参数把 `model_id`、Hub `revision`、`num_shard`、量化方法、`dtype`、`kv_cache_dtype`、并发上限、输入与总 token 上限、等待/运行 batch 比例、等待 token 上限、prefill 与 batch token 预算、CUDA graph batch、tokenizer config path、API key、payload limit、JSON log 和 OTLP endpoint 都暴露为运行时边界 [105]。这说明服务 run card 不能只说“使用 TGI”或“OpenAI-compatible API”：它必须记录 router 如何在等待请求和运行中 batch 之间取舍、prefill token 预算如何限制计算峰值、batch total token 如何绑定显存、客户端能请求多少 stop sequence 或 top-n token、是否加载 LoRA adapter、是否开启 grammar support，以及自定义 Hub 代码是否固定到具体 revision。

TGI 首页还说明该项目现在处于 maintenance mode，后续主要接受小修、文档改进和轻量维护，并建议面向新推理引擎关注 vLLM、SGLang 或本地兼容引擎 [102]。这不会降低 TGI 文档的教材价值，反而要求读者把它当作服务字段清单和迁移基线：记录哪些能力由当前 runtime 原生支持，哪些只是兼容 API，哪些需要换栈后重新验证。

接口层也应进入同一张验收表。TGI 支持 OpenAI Messages API 和 streaming 请求 [104]；guidance 可以通过 `/generate` 的 grammar 或 `/chat/completions` 的 tools 约束输出，工具调用在其实现中是 grammar 的一种特例 [103]。因此，结构化输出不是后处理说明，而是 decode contract：报告应保存 grammar 或 tool schema、编译失败率、validation latency、partial stream 语义、最终对象提交时间、repair 或拒绝策略，以及这些约束是否改变 sampling、stop reason 和安全过滤顺序。TGI 暴露的 Prometheus 指标还把 batch decode/filter/forward 时长、queue size、request duration、input length、max new tokens、mean time per token、validation duration 和 speculated tokens 等字段拆开 [106]；这些指标正好对应本章的 TTFT、TPOT、prefill/decode、过滤和投机解码验收，而不是额外监控装饰。

手写 decode loop 则暴露最小正确性不变量。典型循环会预分配 prompt 加最大新 token 的缓冲区，用 `input_pos` 指向当前逻辑位置，每步只把最近 token 送入模型，再用 top-k、temperature 和 multinomial 采样下一个 token，写回缓冲区并检查 EOS。这个流程的关键不是循环本身，而是 cache 是否按同一逻辑位置更新、生成结束后是否 reset cache、prompt length 是否从输出中正确剥离、tokens/s 是否只统计新增 token。否则一个单用户 demo 可以看起来正常，多用户服务却会出现位置漂移、旧 cache 泄漏或吞吐统计虚高。

Demo 字段	服务中应升级为	缺失后的典型后果
prompt wrapper	版本化 chat template 与 stop rule	SFT 与推理模板不一致
tokenizer path	tokenizer、special token 与 embedding shape 记录	resize 或 PAD/EOS 错误被忽略
sampling config	有序 logit-processor 与随机性契约	复现实验和线上输出不一致
device map/8-bit	runtime、量化和 kernel 配置	显存、延迟和质量无法归因
prefix batch	prefix-cache key、权限和证据版本	跨租户复用或旧证据复用
decode loop	cache position、reset 和 stop reason 测试	cache 泄漏、位置漂移或计费错误

8.2 KV Cache、批处理与调度

KV cache 是长上下文推理的主要内存压力。PagedAttention 等方法把 cache 管理变成类似操作系统内存分页的问题 [141]。服务系统需要在吞吐、首 token 延迟、尾延迟和公平性之间平衡。调度器需要 admission control。它必须在请求进入时估计 prompt length、max new tokens、并发数和 KV cache 上限；还要在生成过程中处理提前停止、用户取消、工具等待和安全检查。结构化输出和工具调用还会引入解析器、schema validator、沙盒执行和重试逻辑，这些都应计入延迟预算。最新推理课件中的 continuous batching、vLLM page cache 和 chunk-prefill 都指向同一个问题：服务系统要同时调度计算和 KV 状态。长 prompt 的 prefill 往往 compute-bound，逐 token decode 往往 memory-bound；chunk-prefill 把长 prompt 切成小块，避免一个长请求独占 GPU；连续 batching 则在请求完成和加入时动态维持 decode batch。评测这些方法时要分别看 TTFT、TPOT、KV block 利用率、取消释放是否及时，以及长短请求混合下的 p95/p99。教学版 continuous batching 实现把生产系统必须显式维护的状态暴露得很清楚。Request manager 需要记录 request id、prompt token、已生成 token、当前长度、最大长度和 waiting/running/completed 状态；KV cache manager 需要记录固定槽或分页槽、每个槽的序列长度、request 到 slot 的映射、slot 到 request 的映射以及空闲槽集合。Prefill 路径分配槽位，只在模型 batch 内 padding prompt，写入 prompt KV，并返回第一个生成 token 的 logits。Decode 路径只把每个活跃槽上一轮生成的 token 和当前位置送入模型，每层写入一个新的 KV，更新长度，检查 EOS 或长度停止条件，并在请求结束时立刻释放槽位。这些状态转移就是 continuous batching 的实现含义。

notebook 里可以简化的细节，在服务里会变成契约。如果请求生产者和推理主循环并发运行，请求队列必须线程安全；max batch size 和 max sequence length 应来自显存预算，而不是独立手填。形状为 layer、batch、sequence、head、head dim 的固定 tensor cache 适合教学，但会为短请求过度预留内存，也会在混合流量下浪费容量；paged cache 改变的是

分配粒度，并不取消 request 状态管理。某个 decode step 让请求结束后，系统必须在下一步前把它移出 active batch，并让 cache slot 可复用，同时不能把旧 token 暴露给其他租户。

KV cache 的容量可以先用简单公式估算。对一个 dense decoder，若层数为 L、key/value head 数为 H_{kv}、head 维度为 d_h、活动序列长度为 T、batch size 为 B、每个 cache 元素占 s bytes，则近似显存为

$$M_{\text{KV}} = 2BTLH_{\text{kv}}d_h s.$$

系数 2 来自 key 和 value。MHA 中 H_{kv} 通常等于 query head 数；MQA 可能只有 1；GQA 介于两者之间 [214, 1]。这意味着架构选择直接改变服务并发能力。一个 32 层、$H_{\text{kv}} = 8$、$d_h = 128$、BF16 cache 的模型，每个活动 token 大约需要 $2 \times 32 \times 8 \times 128 \times 2 = 131{,}072$ bytes。40 GiB 可用 KV 预算约能容纳 327k 活动 token，约等于 80 个 4096-token 对话，或 20 个 16k 长文档请求；混合流量下还要扣除 block overhead、碎片、保留空间和安全过滤占用。

当前 Transformers 文档把 cache 选择暴露为生成接口的一部分，而不是隐藏实现。GenerationConfig 中的 `cache_implementation` 可以选择 `dynamic`、`static`、`offloaded`、`offloaded_static` 或 `quantized`，并用 `cache_config` 传入 cache 类参数 [76]。服务验收因此要记录 cache implementation、max cache length、量化 backend、位宽、group size、residual length、offload 范围和 fallback 规则。Static cache 预分配上限，便于 `torch.compile` 和稳定 decode 形状，但会在短请求或长度分布很宽的流量中浪费被 mask 的位置；offloaded cache 用 CPU 内存和异步搬运换 GPU 显存；quantized cache 省驻留空间，却需要单独测长上下文复制、引用和数字任务质量 [62, 91]。如果模型包含 sliding-window、chunked 或 hybrid attention，某些层的 cache 不会按完整上下文继续增长，run card 应逐层列出上界，否则容量估算会把架构窗口和 runtime 策略混在一起。

准入控制不能只按请求数。一个短聊天和一个 100k token 文档问答对 GPU 的压力不同；一个已生成 4000 token 的请求比刚进入队列的请求更占 cache。保守策略按 prompt length 加 max new tokens 预留最坏情况，利用率低但稳定；激进策略按期望长度准入，同时在生成过程中执行硬上限、取消和降级。后者需要清楚的产品语义：超预算时是停止生成、返回部分结果、排队等待，还是转到低优先级 worker。没有这些规则，所谓“高吞吐”可能只是把失败转移给尾延迟和用户取消。

内存碎片也会成为线上故障源。变长序列不断分配和释放 cache 区域，可能出现总空闲显存足够、但没有连续区域可放长请求的情况。PagedAttention 用固定 block 降低碎片；cache arena、预分配池、slot compaction 和及时释放也是常见手段。对于 chat 会话，是否保留 warm prefix 取决于产品语义；对于隐私敏感系统，cache 生命周期和跨租户隔离是安全要求，不只是性能优化。

调度策略应写成可以复现的状态机。等待队列中的请求至少有 prompt tokens、max new

tokens、priority、deadline、tenant、sampling config 和 estimated KV reservation；运行中的请求还要记录已生成长度、当前 block table、last-token buffer、stream handle、cancellation flag 和 stop reason。每个 step 后都要更新状态并释放完成请求。若这些状态只散落在日志里，后续很难解释为什么 p99 变差、为什么 cache 没释放，或者为什么某个租户被低优先级流量挤占。

准入策略还应区分软预算和硬预算。软预算用于排队排序和容量估计，可以根据历史长度分布调整；硬预算用于防止单个请求越界，必须在 decode 中检查。超过软预算可以降级到低优先级、要求缩短上下文或转入离线队列；超过硬预算则应返回清楚的停止原因。这样做比简单 OOM 后重试更可控，也方便把成本和用户体验写入 SLA。

多租户服务还需要把配额和公平性写进调度器。租户配额不应只是每分钟请求数；更有用的单位是 input token、output token、active token、最大并发、工具调用次数和安全复核预算。高优先级租户可以获得更短队列或更高 decode 份额，但不应绕过硬性安全过滤和 cache 隔离。若只按请求数限流，长 prompt 或长输出租户仍可能占满 KV cache；若只按 token 计费，短而频繁的请求仍可能用 launch 和 streaming flush 拖高尾延迟。报告应说明配额的计量单位、窗口长度、超额后的降级方式，以及这些规则是否进入 load test。

公平性测试不能只看单租户峰值吞吐。候选调度器需要在同一随机种子下回放混合租户轨迹，比较每个租户桶的排队时间、拒绝率、active-token 占用、取消率和实际完成 token。若平均吞吐上升但低优先级租户长期饥饿，或者高优先级租户绕过安全复核预算，这个调度器就不应作为发布证据。

发布级服务记录应把这些状态字段固定下来，而不是只给一个吞吐数字。一次推理服务实验至少要记录模型版本、tokenizer 和 chat template、context limit、RoPE 或 NTK scaling、并行策略、attention kernel、cache dtype、block size、weight/KV quantization、sampling 默认值、stop rule、admission rule、batching policy、prefix-cache key、timeout、retry、rate-limit、安全过滤器版本、日志采样率和计费口径。缺少其中任一项，别人很难复现实验，也无法判断一次延迟变化来自模型、runtime、调度器、过滤器还是流量分布。

记录域	最小字段	缺失后的风险
模型与模板	checkpoint、tokenizer、chat template、context scaling	输出差异被误归因于服务栈
内存与 cache	implementation、config、dtype、block size、KV heads、eviction、tenant isolation	cache 泄漏、碎片或容量不可复现
调度与准入	admission rule、priority、batching、deadline、backpressure	p99 与拒绝率无法解释
解码与采样	采样参数、seed、停止规则、验证器	质量和可复现性不可比较
过滤与工具	prompt/final filter、tool timeout、retry、sandbox	安全延迟和失败模式被隐藏
成本与日志	input/output token 计费、trace 字段、sampling rate	容量规划和事故复盘缺证据

8.3 下一代推理系统

随着 MoE、长上下文和 disaggregated serving 发展，推理服务开始分离 prefill/decode、attention/FFN 或专家路由。Frontier simulator 这类工作说明，下一代服务系统必须能模拟专家并行、跨集群路由和异构资源 [39]。

PD 分离的核心动机是 prefill 和 decode 的资源形态不同：prefill 适合大矩阵吞吐，解码阶段受权重读取和 KV cache 带宽约束。把两者放到不同 worker 或硬件池可能降低混合流量尾延迟，但会引入 KV 状态传输、路由、失败恢复和跨租户隔离问题。MoE 推理还要额外监控专家 token 分布、all-to-all 时间和 compute-communication overlap，否则平均吞吐可能掩盖专家拥塞。生产设计里应把 PD 阶段目标写清楚。Prefill 池重点优化 input tokens/s、prompt batch 利用率和 TTFT；decode 池重点优化 output tokens/s、TPOT、KV cache 驻留数量和显存带宽。两阶段交接不是免费箭头，而是必须保持精确 token 前缀、位置状态和 KV 表示一致，并把 KV 传输时间、decode 池反压、失败重试计入服务指标。对于 DeepSeek-V3 类 MoE 服务，报告还应说明 MLA 式 cache 压缩、专家并行路由、FP8 或其他推理精度、MTP head 是否真的出现在 serving 路径，而不只是训练配方中的组件 [31]。生成式多模态系统还会扩大服务问题。图像编码、视频采样、ASR、语音合成、扩散步数、flow matching 调度和安全滤镜都可能与文本模型竞争资源。一个 Omni 系统的“延迟”不是单一模型 decode 时间，而是编码器、connector、LLM prefill、LLM decode、语音/图像/视频生成器和后处理共同构成的端到端时间。

MoE 服务的瓶颈不能只看整体 tokens/s。Router 可能把某些专家打满，让 all-to-all 成为尾延迟来源；专家并行的最佳拓扑也可能和 attention 或 dense FFN 的最佳拓扑不同。一个稀疏模型的服务报告应分开记录 prefill throughput、decode throughput、每专家 token

count、专家负载均衡、all-to-all 时间、cache movement、batch spill、失败重试和 tail latency。否则，某个 batch 策略看似提高平均吞吐，实际可能只是把专家拥塞藏在 p99。

PD 分离的失败模式也要提前列出。Prefill worker 生成的 KV 状态若和 decode worker 的模型版本、tokenizer、RoPE scaling、position offset、cache dtype 或 block layout 不一致，输出可能静默漂移；跨节点 KV 传输若没有 backpressure，会把 decode 池压垮；失败重试若重新 prefill 长 prompt，会把事故放大。发布报告应包含 handoff 成功率、KV transfer latency、decode-queue depth、跨池重试率和版本一致性检查。

对于多模态和工具系统，服务边界不止 LLM。图像 encoder、ASR、TTS、工具沙盒、检索服务、重排器、安全分类器和 citation checker 都会进入端到端延迟。若这些组件共享 GPU、网络或数据库连接池，单独测 LLM decode 没有意义。端到端报告应按组件归因，并说明哪些步骤可并行、哪些步骤必须串行、哪些失败可以重试，哪些失败必须中止。

8.4 深入展开：推理服务是另一套系统工程

本章把推理和服务从训练中分离出来。训练时目标是高吞吐地处理大量 token；服务时目标是满足用户请求的延迟、成本、可靠性和安全边界。Prefill 处理 prompt，一次性计算所有输入 token；decode 每次生成一个新 token，需要反复调度。短聊天、长文档总结、代码生成、RAG、agent 工具循环和语音对话的服务形态完全不同。

KV cache 是服务成本核心。每个活动请求都占用与层数、head 数、head 维度、上下文长度和 batch 大小相关的 cache。PagedAttention 把 cache 管理变成类似操作系统分页的问题，减少碎片并提高 batch 利用率 [141]。但调度器仍需决定哪些请求可以进入、如何混合长短请求、如何处理取消和超时、如何在多租户场景下保证隔离。采样和结构化输出也属于服务逻辑。温度、top-p、repetition penalty、logit bias、stop sequence 和随机种子应有确定顺序；其中 repetition penalty 应说明是否统计 prompt token、是否只作用于已生成 token、以及在 logit bias 和截断采样前后的位置。JSON、SQL、函数调用和工具参数需要增量验证；一旦输出不合法，系统要决定修复、重试、拒绝还是交给人工。安全过滤可能发生在 prompt 前、流式生成中和最终输出后，每一步都会增加延迟和失败模式。下一代推理系统还会遇到 MoE 和 disaggregated inference。Prefill 和 decode 的硬件需求不同，专家路由和 FFN 可能适合不同资源，长上下文请求可能需要特殊调度。本章强调，服务研究不应把模型当作固定黑盒，而应研究架构如何改变调度器、缓存、通信和成本模型。

8.5 章节细节

8.5.1 服务不是 Evaluation Mode

推理服务不是把模型切到 eval 后运行。服务系统要处理请求排队、流式输出、取消、超时、缓存、配额、日志、安全过滤和多租户隔离。用户体验由端到端系统决定，而不是单次 forward 决定。

8.5.2 Prefill 与 Decode

Prefill 处理输入 prompt，decode 逐 token 生成输出。Prefill 更像大矩阵并行计算，decode 更受调度和 KV cache 约束。长输入短输出、短输入长输出和多轮工具调用的瓶颈不同。

8.5.3 KV Cache

KV cache 保存历史 key/value，避免每步重复计算前缀。它随层数、头数、head 维、上下文长度和并发请求增长。长上下文服务的主要显存压力往往来自 cache，而不是模型权重。

8.5.4 Batching 与调度

连续 batching 让不同请求在生成过程中动态合并，提高 GPU 利用率。调度器必须兼顾长短请求、公平性、超时和吞吐。一个追求最大吞吐的策略可能让短请求延迟不可接受。

8.5.5 内存管理与准入控制

PagedAttention 等方法把 KV cache 管理成分页式 block，减少碎片。准入控制决定新请求能否进入系统，防止 cache 爆掉或延迟失控。服务质量依赖调度、缓存和配额共同控制。

8.5.6 量化与压缩

权重量化、KV cache 量化和剪枝可以降低成本，但也可能影响事实性、数学、代码和安全边界。量化评测应覆盖目标部署任务，而不是只看平均 benchmark。不同硬件对量化格式支持也不同。

量化不是一个单旋钮。报告应说明 bit width、per-tensor 或 per-channel scale、group size、calibration data、outlier 处理、activation 是否量化、KV cache 是否量化，以及目标硬件是否有对应 kernel。GPTQ、AWQ 和 SmoothQuant 等方法面向的误差来源不同 [40, 154, 245]。一个 4-bit checkpoint 如果没有高效 kernel，可能节省显存但不降延迟；一个校准集如果缺少代码、数学、多语言和长上下文样本，可能掩盖严重回归。出版级评测应同时报告质量指标和系统指标：perplexity、任务准确率、复制/引用行为、代码执行、长上下文召回、安全探针、模型加载时间、峰值显存、TTFT、TPOT、batch 容量和每百万 token 成本。

上线判据应把量化前后差异写成阈值。可接受的 perplexity 变化、关键任务准确率下降、代码执行失败率、长上下文 needle recall、拒答误差和 p95/p99 延迟变化都要预先定义。若量化只在短英文聊天上无损，却让长 RAG 引用、中文代码注释或 JSON 函数参数变差，就不能把平均分数当作发布证据。KV cache 量化还要单独检查长上下文和逐字复制，因为它直接影响旧 token 的注意力表示。

8.5.7 投机与并行解码

Speculative decoding 用小模型提出候选，再由大模型验证，以减少大模型调用次数。它提高速度的前提是草稿模型足够匹配目标模型。并行解码和多 token prediction 也需要质量、延迟和复杂度权衡。这里要区分训练中的 MTP 和服务中的多 token 解码。训练报告里的 MTP 可以只是辅助损失，用来塑造表示，训练后移除额外 head；服务系统里的多 token head 或草稿路径则必须在推理时存在，并证明输出质量没有下降 [31]。因此，报告应说明未来 token 预测影响的是目标函数、发布 checkpoint、运行时 decoder，还是三者都有。实现层面还要说明“MTP”到底是哪一种。简单辅助 head 可以从当前 hidden state 直接预测向后第 i 个 token；DeepSeek-V3 风格 MTP 更接近递归的 next-token-prediction 栈，后续辅助模块消费 shifted embedding 和 latent feature，并在移位位置上做 next-token loss。若服务端保留这条路径，它仍然需要自己的 KV cache、接受规则和回滚逻辑。因此同样叫 MTP 的方法，必须区分训练 loss、checkpoint 结构和运行时解码算法。

投机解码报告不应只给 speedup。草稿 token 的 acceptance rate 应按短聊天、长 RAG、代码、数学、多语言和安全拒答分别统计，因为低接受率会让验证成本、回滚逻辑和 KV 写入抵消加速收益。若候选系统在高风险请求上关闭草稿路径，也要说明路由规则和降级后的延迟；否则平均吞吐会掩盖关键场景的真实成本。

8.5.8 注意力 Kernel 与长上下文

FlashAttention、滑动窗口、分块注意力和压缩上下文都会改变成本曲线。长上下文不是只改变最大长度，还改变 prefill 时间、cache 体积和调度策略。服务评测应使用真实长度分布。第 11 章加速课件可以按瓶颈重新分类。MQA/GQA 减少 KV head，从而降低 decode cache 与带宽；FlashAttention/FlashAttention-2 通过 tiling、I/O-aware 计算和更好的 work partition 提升 prefill 与训练式 attention 的效率 [28, 27]；PagedAttention 解决的是变长请求下的 KV block 管理和碎片；StreamingLLM 的 attention sink、滑动窗口和 LongLoRA 的 shifted sparse attention 改变长上下文中哪些 token 参与计算 [246, 21]；speculative decoding、Medusa 和 Lookahead decoding 则减少目标模型串行 decode 步数 [146, 15, 41]。FlashAttention 的关键不只是“用了 block”, 而是 online softmax 不变量：query 按 key/value block 流式扫描时维护每行 running maximum 和 normalizer，后续 block 改变最大值时重缩放已有部分输出，因此无需物化完整 score matrix 也能保持精确 attention 语义。这些方法不能只按“是否加速”比较，而要说明作用阶段、是否改变模型结构、是否保持输出分布、是否需要额外 head 或草稿模型，以及在真实长度分布下的 TTFT、TPOT、显存和质量变化。

方法族	例子	主要瓶颈	报告要点
KV head 减少	MQA, GQA	decode cache 与带宽	KV heads、cache bytes/token、质量变化
Kernel/tiling	FlashAttention 系列	prefill 与长序列注意力 I/O	是否精确、dtype、硬件、序列长度
Cache 管理	PagedAttention	碎片和连续 batching	block size、eviction、sharing、取消释放
窗口/汇聚注意力	窗口、sink、LongLoRA	长上下文内存和训练成本	哪些 token 可见、推理是否全注意力
并行解码	Speculative, Medusa, Lookahead	串行目标模型步数	acceptance rate、草稿/额外 head、质量保持

8.5.9 流式 API、取消与安全过滤

流式输出提升感知速度，但也要求中途安全过滤和可取消执行。用户取消后，系统必须释放 cache 和工具资源。安全过滤可能发生在输入、生成中和输出后，每个位置都有不同延迟和漏检风险。

流式 API 还需要定义 token 到文本的边界、partial JSON 或 tool-call 片段的发送规则、stop sequence 的截断语义，以及客户端断开时的清理路径。Timeout 也应拆开：queue timeout、prefill timeout、decode timeout 和 total timeout。排队超时几乎没有消耗 GPU；长 decode 超时可能已经消耗大量计算，盲目重试会在事故中放大负载。安全过滤和结构化 validator 不是免费后处理：如果 prompt classifier、streaming classifier、final classifier 或 JSON repair 共用同一 GPU 池，它们会和生成争抢容量，也必须进入 load test。

安全过滤的位置会改变可用性。Prefill 前的 prompt filter 能省 GPU，但可能误拒需要上下文澄清的请求；streaming filter 能更早拦截危险输出，但需要处理半句话和部分 JSON；final classifier 能看完整回答，却可能太晚才发现问题。服务报告应分别统计 unsafe compliance、false refusal、safe redirection、filter latency 和人工升级率，并说明被拦截请求是否释放 KV cache、是否写入审计日志、是否触发重试。

8.5.10 负载测试与成本记账

服务报告应给出 TTFT、TPOT、吞吐、p50/p95/p99 延迟、并发、上下文长度分布和成本。平均延迟不足以解释用户体验。成本也应分解为权重常驻、prefill、decode、cache、工具和后处理。

负载测试应模拟流量分布，而不是只用相同 prompt 打满服务器。真实请求有输入长度、输出长度、采样参数、租户优先级、cache 复用、取消率、工具调用和安全过滤差异。最小报告应包含 request/s、input tokens/s、output tokens/s、TTFT p50/p95/p99、TPOT p50/p95/p99、端到端延迟、错误率、取消率、GPU memory、GPU utilization、cache hit rate、active tokens 和每百万输入/输出 token 成本。慢请求 trace 应能分解到 queue、prefill、decode、postcheck、stream flush 和工具等待。只有这样，才知道尾延迟来自长 prompt、cache 碎片、straggler kernel、网络 backpressure、安全过滤还是重试风暴。

负载生成器还要说明到达过程。固定并发数、固定 request/s 和真实突发流量会触发不同瓶颈；Poisson 到达、批量上传、课后作业高峰、新闻事件洪峰和 agent 批量重试都不等价。测试脚本应记录到达时间戳、burst factor、warmup 窗口、稳态窗口和冷启动比例，并把客户端超时、主动取消和服务端 load shedding 分开统计。

成本模型应把输入 token 和输出 token 分开。长 prompt 消耗 prefill 计算和 KV 空间，长输出消耗串行 decode 和流式连接时间；二者不能用一个平均 token 价格解释。共享前缀、prefix cache、量化、speculative decoding 和批处理会改变边际成本，但也可能增加复杂度和失败重试。容量规划最好同时给出峰值并发、active-token 上限、可承受取消率、每租户配额和降级路径。

负载测试还应该包含事故演练。常见演练包括：长 prompt 洪峰到来时是否触发 admission rejection 而不是 OOM；客户端批量取消后 cache block 是否立刻释放；安全分类

器变慢时是否对生成 worker 形成 backpressure；工具服务超时是否触发指数退避而不是同步重试风暴；prefix cache 命中异常升高时是否能发现权限或证据版本错配。事故演练的结果应和正常负载测试放在一起，因为上线后的服务质量往往由退化路径决定，而不是由理想路径决定。

正式发布前还要区分 benchmark、load test、shadow traffic 和 canary。Benchmark 验证离线质量和 kernel 性能；load test 验证受控流量下的容量；shadow traffic 把真实请求复制到候选栈但不返回给用户，用来观察模板、tokenizer、过滤器和成本差异；canary 把少量真实用户导向候选栈，用来验证真实交互和回滚路径。每一步都应有 SLO：例如 TTFT p95、TPOT p95、错误率、取消后 cache 释放时间、安全过滤延迟和每百万 token 成本。若 canary 失败，回滚对象也要明确：模型权重、量化配置、adapter、chat template、检索索引、filter 版本、调度策略或整个 runtime。

场景	TTFT p95	E2E p95	输出 toks/s	成本/百万 token
短聊天				
长 prompt QA				
代码生成				
共享前缀 RAG				
混合生产轨迹				

8.5.11 实现笔记

实现服务时，应先用固定 prompt 和固定 seed 建立基线，再加入流式、batching、量化、RAG 和工具。每加一个功能都要重新测延迟、显存和失败模式。服务系统的正确性包括输出正确，也包括资源正确释放。小型推理 demo 还会暴露复现性缺口。如果运行时 prompt template 和 SFT template 不一致，报告应明确说明，不能只把输出归因于模型本身。如果 LoRA adapter 搭配扩展 tokenizer，生成前可能需要 resize base embedding 和 LM head。如果加载时打了 NTK 或 RoPE scaling patch，alpha 值和最大上下文假设就是 serving config 的一部分。流式包装器常常收到累计 token id，再按 prompt 长度切出新增部分；它必须处理 EOS、token 到文本的边界、用户取消和 cache 清理。按字符数截断历史不是合格的上下文管理，因为它可能切断模板、角色边界或 special token。

从实现审计角度看，推理脚本的输出文件也不是普通日志。批量预测写出的 JSON 应至少保留输入 prompt、模板化后的实际 token 序列或其 hash、输出、生成参数、模型/adapter/tokenizer 版本、停止原因和错误状态。只保存最终文本会让后续无法区分模型失败、模板失败、tokenizer 失败、采样失败还是服务失败。若脚本同时写出 generation config，发布记录应把它和 checkpoint、量化配置、adapter merge 状态放在同一 run card 中，而不是散落在不同目录。

交互式 demo 还容易隐藏多轮边界。很多脚本只支持单轮问答，却会被用户当作聊天服务使用。若没有 conversation id、历史裁剪、角色标记、工具观察、取消标志和 cache reset 规则，多轮输入可能只是把文本拼接到旧 prompt 后面。高质量书稿应提醒读者：单轮 inference、批量离线 prediction、流式在线 chat 和 agent tool loop 是四种不同接口，不应共用一个未经说明的生成函数。

最小回归套件应覆盖固定 prompt、长 prompt、共享 prefix、短输出、长输出、取消、超时、JSON schema、工具失败、安全拒答和多租户隔离。每个用例都要检查输出文本、stop reason、token 计数、KV block 释放、日志字段和计费字段。这样可以把“模型回答正确”和“服务资源正确”同时纳入测试，而不是上线后才发现取消请求仍占用 cache 或错误重试被计费。

上线运行还需要预热和回滚纪律。冷启动会把模型加载、kernel compilation、CUDA graph capture、static cache 编译、prefix cache 填充和安全分类器加载混在首批用户请求里，因此发布前应记录 warmup 脚本、预热 token 分布和预热后基线指标。Static cache 或 CUDA graph 路径还要固定 batch size、最大输出长度和 padding 桶；否则形状变化会触发重新初始化或重新编译。回滚不能只写“切回旧模型”：如果新版本改变了 tokenizer、chat template、RoPE scaling、cache implementation、cache dtype、量化 group size 或 tool schema，回滚时必须清理不兼容 cache，并让 trace 能区分旧请求、新请求和重试请求。缺少这些字段时，事故复盘只能看到延迟上升，无法判断问题来自模型、runtime、调度器还是外部工具。

服务 trace schema 应在上线前固定。每个请求至少记录 request id、tenant 或匿名化租户桶、模型和模板版本、tokenizer hash、admission decision、queue time、prefill time、decode step 数、TTFT、TPOT、输入/输出 token 数、active token 峰值、KV block 分配和释放、prefix-cache 命中或未命中原因、scheduler policy、安全过滤事件、工具调用事件、stop reason、错误码和计费口径。日志不应保存不必要的原文敏感内容；可以保存 token 序列 hash、长度、风险标签和抽样后的脱敏样例。这样既能支持容量规划，也能在隐私边界内复盘尾延迟和错误输出。

SLO 也要按阶段拆开。一个服务可以规定 TTFT p95、TPOT p95、端到端 p99、错误率、取消后 cache 释放时间、安全过滤延迟、工具超时率和每百万 token 成本上限，并为短聊天、长 RAG、代码生成和 agent 循环分别设阈值。Canary 失败的判据应写在发布前，而不是事故后临时解释：例如 TTFT 连续两个窗口超过阈值、prefix-cache 异常命中、unsafe compliance 上升、JSON repair 率异常或某个租户的拒绝率突增，都应触发降级或回滚。没有 error budget 的服务报告，很容易把一次平均吞吐提升误当成可发布改进。

发布门槛最好写成候选栈相对基线的比较，而不是孤立数字。候选版本需要在同一流量回放下满足质量、延迟、成本和安全阈值：例如关键任务准确率不降、TTFT p95 不超

过基线某个比例、取消后 KV 释放仍在预算内、JSON repair 率不升高、过滤器误拒不扩大。只有这些条件同时成立，系统优化才算可发布。

事故复盘应能把“请求结果”还原成“资源状态”。取消请求后 cache 是否释放、重试是否复用了过期 prefix、过滤器是否和生成 worker 抢 GPU、工具超时是否造成同步重试风暴、PD handoff 是否在版本不一致时继续 decode，这些问题都不是最终文本能说明的。复盘记录应包含慢请求 trace、同时段队列深度、KV block 水位、worker 版本、外部工具状态、过滤器延迟、限流决策和回滚动作。出版稿把这些字段写出来，可以防止读者把推理服务误解成单纯的生成循环。

8.6 关键术语、实现要点与练习

关键术语。 Prefill 处理 prompt；decode 逐 token 生成；KV cache 保存历史 key/value；paged attention 管理 KV block；prefix cache 复用完全相同的 token 前缀；prefix-cache hash 决定共享前缀如何映射到 cache key；cache salt 用于隔离多租户或多模态 prefix-cache key；continuous batching 在每个生成步重新组合活跃请求；chunked prefill 把长 prompt 拆成多个 prefill step；soft reset 表示 cache 满时把已生成内容并回 prompt、释放 KV block 后重排请求；offloading 把被逐出的 cache 状态移到 CPU 或其他存储再恢复；per-request processor 让同一 batch 内不同请求使用不同采样参数；StreamingLLM 保留 attention sink 与最近窗口；Medusa 用额外 head 并行提出未来 token；ITL 表示 inter-token latency；PD 分离把 prefill 和 decode 放到不同执行资源；KV transfer connector 负责把 prefill 产生的 cache 状态交给 decode 资源；admission control 决定请求能否进入系统；backpressure 表示下游容量不足时向上游施加压力；load shedding 表示主动拒绝或降级以保护系统；quota 表示按请求、token、active token、并发或工具调用定义的容量边界；SLO 是把延迟、错误率、成本和安全过滤目标写成可检查门槛；canary 是把少量真实流量导向候选栈的发布验证；rollback path 是候选栈失败时撤回模型、模板、索引、过滤器或 runtime 的路径；tail latency 反映最慢用户体验；stop reason 记录生成停止、超时、取消、过滤或预算耗尽的原因。

实现要点。 服务应分别测量排队、prefill、decode、工具、过滤和后处理延迟；调度器应处理长短请求混合、取消、超时、配额和多租户隔离；cache key 必须绑定模型、tokenizer、模板、位置状态、权限和证据版本；服务配置应记录 KV cache implementation、`cache_config`、容量、dtype、block size、prefix-cache hash、`max_batch_tokens`、cache budget、block sharing、soft reset/offload、partial prefill 并发、调度策略和每请求采样参数；prefix-cache hash 应写明完整 block、父 hash、额外 hash、cache salt 与确定性算法；PD 分离实验必须报告 TTFT、ITL/TPOT、KV 传输、handoff、backpressure 和失败恢复，而不能只报告吞吐；

结构化输出、安全过滤和工具调用都应进入成本模型；发布流程要包含 warmup、shadow traffic、canary 和 rollback 证据；回归测试要同时检查输出质量、资源释放、日志、计费和 stop reason。

练习。

1. 推导 $L = 32$、$H_{\text{kv}} = 8$、$d_h = 128$、BF16 cache、$B = 24$、活动长度 $T = 4096$ 时的 KV cache 显存；再说明 MQA 中 $H_{\text{kv}} = 1$ 时如何变化。
2. 设计一个混合流量调度策略：短聊天、长总结、代码生成和 RAG。分别写出 admission rule、batching rule、取消规则和优先级规则。
3. 说明为什么 p95/p99 latency 比平均 latency 更重要，并列出五种可能造成尾延迟的系统来源。
4. 一个 4-bit weight-only 模型比 BF16 省显存但 p95 延迟更差。列出五种可能原因，并为每种原因给出一个区分性测量。
5. 为一个 tool-use 服务写出失败恢复策略：参数非法、工具超时、返回冲突和权限不足。
6. 为长 RAG prompt 和短聊天混合流量设计 PD 分离方案，说明 KV 状态如何移动以及哪些指标能发现分离后尾延迟变差。
7. 设计一个服务回归测试表，覆盖取消、超时、streaming stop sequence、JSON 工具调用和安全过滤，并说明每项应检查哪些日志字段。
8. 给出一个 prefix-cache 发布方案，说明 cache key、权限边界、证据更新、命中率指标和错误复用的检测方法。
9. 设计一次 canary 发布计划，写出 warmup、shadow traffic、SLO、回滚对象、cache 清理和事故复盘字段。
10. 为 continuous batching 服务写一张 architecture run card，覆盖请求状态机、token budget、cache budget、scheduler、block sharing、soft reset/offload、prefix-cache hash 和多租户隔离。
11. 为 vLLM 或 Transformers continuous batching 服务写一张 run card，列出 KV cache 配置、prefix-cache hash、chunked-prefill 参数、调度策略、per-request sampling、取消语义和 streaming 日志字段。
12. 为 Transformers 的 `dynamic`、`static`、`offloaded`、`offloaded_static` 和 `quantized` cache 设计同机回归，比较 TTFT、TPOT、峰值显存、形状重编译、长上下文质量和 OOM fallback。

8.7 结构化检查表

8.7.1 推理服务成本分解

阶段	需要测量
排队	admission control、优先级、租户隔离、请求混合
Prefill	prompt tokens、视觉 token、检索证据、batch shape
Decode	输出长度、KV cache、采样、stop condition、streaming
工具	schema validation、网络延迟、超时、重试、沙盒
安全	prompt filter、streaming filter、final classifier、人工升级
发布运行	warmup、shadow traffic、canary、SLO、rollback path
后处理	citation check、JSON repair、logging、monitoring
回归证据	固定 seed 输出、stop reason、KV 释放、错误码、计费字段

第三部分

适配

第九章　监督指令微调

9.1　接口转换

SFT 把基础模型变成遵循指令的模型。一个样本包含系统消息、用户消息、上下文、工具观察和目标回答。训练时通常只对 assistant token 计算损失，prompt token 作为条件而非标签。

设一条样本由上下文 c 和目标助手回答 a 组成，chat template τ 把结构化消息序列化成 token 序列 $z_{1:T} = \tau(c, a)$。监督指令微调仍是自回归 next-token loss，但只在要模仿的回答位置上计入损失：

$$M_{\text{mask}} = \sum_{t=1}^{T} m_t, \qquad \mathcal{L}_{\text{SFT}}(\theta) = -\frac{1}{M_{\text{mask}}} \sum_{t=1}^{T} m_t \log p_\theta(z_t \mid z_{<t}).$$

其中 $m_t = 1$ 表示 assistant 目标 token，$m_t = 0$ 表示系统消息、用户消息、检索证据、padding 和通常不需要模仿的分隔符。许多训练库用 `-100` 作为 ignore label。这个小细节很关键：如果用户指令或系统策略也被计入 loss，模型会被训练去生成 prompt，而不是只生成回答。

chat template 是数据格式的一部分。相同语义样本如果被写成 system/user/assistant 消息、普通 prompt-completion、工具调用 JSON 或厂商私有格式，会训练出不同边界行为。训练和推理模板不一致，是让一个本来可用的模型表现异常的常见原因。模板应明确角色、轮次结束、工具观察、图像占位符和停止条件。指令微调和下游任务 SFT 要分开报告。指令微调的数据通常覆盖翻译、总结、抽取、代码、对话和拒答等通用任务，目标是让模型理解自然语言任务描述并形成通用助手接口；任务 SFT 则更窄，服务于某个产品流程、领域语料、schema 或客户风格，可以用全参微调，也可以用 LoRA/QLoRA 等 PEFT。两者使用的 loss 可能相同，但证据要求不同：前者要证明任务广度和未见指令泛化，后者要证明目标 workflow 变好且基础能力没有回归。图 9.1 概括了从文本延续模型到序列化助手接口的转换。

最新 Transformers chat template 文档把训练与推理的差别写得更具体 [63]。训练时通常应把完整消息序列渲染成样本，不额外添加 generation prompt；推理时才常用

`add_generation_prompt=True` 让模型进入 assistant 回复位置。若要让模型续写一个已经开头的 assistant 消息，应使用继续最后消息的语义，而不能和 generation prompt 混用。若先用 `apply_chat_template(tokenize=False)` 得到字符串再单独 tokenize，还必须禁用额外 special-token 注入，否则 BOS、EOS 或 role marker 可能重复。SFT 报告应保存训练模板调用方式、推理模板调用方式、是否使用 assistant prefill、是否允许工具/文档参数进入模板，以及每个路径下的渲染样例。

指令数据不是 prompt-answer 列表，而是行为契约。一条好样本应说明谁在说话、消息优先级如何、可用外部证据是什么、输出格式有什么约束、请求不可完成或不安全时应怎样回应。若这些信息隐含在自然语言里，模型学到的往往是数据集里的偶然相关，而不是部署中稳定的协议。

数据质量也要分层审计。样本层面要检查答案是否正确、完整并符合指令；数据集层面要检查任务、长度、语言、领域、拒答类别和工具格式是否均衡；来源层面要检查许可证、可再分发状态、隐私风险和 benchmark 污染。公开评测题常常本身就是指令格式，如果进入 SFT 训练，后续榜单分数就不能再当作泛化证据。

数据契约最好落到 manifest，而不是只保存在数据生成脚本里。每个样本至少应能追溯来源、创建时间、许可状态、语言、任务族、目标格式、是否含工具观察、是否含检索证据、是否为安全或拒答样本、是否来自教师模型、是否经过人工审核，以及与哪些评测集做过去重。没有这些字段，后续看到模型变好或变差时，很难判断是模板变化、任务混合、教师错误还是安全数据比例造成的。

SFT 混合数据还要同时报告样本数和 target-token 数。一个数据源如果样本数很少但答案很长，可能在 token-weighted loss 中占很大权重；一个短分类数据源如果被按样本均匀采样，也可能改变模型格式偏好。报告中应按来源画出 prompt 长度、assistant 长度、被截断比例、拒答比例、语言比例和自动验证器覆盖率。这样读者才能判断训练是在学习通用助手接口，还是在过拟合某个模板、某种口吻或某类题。

指令数据也应包含失败示范和恢复示范。工具调用样本不应只有成功轨迹，还应有工具报错、权限不足、参数不合法和证据缺失时的处理；结构化输出样本不应只有正确 JSON，还应包含用户要求不完整时如何追问；多轮对话样本应覆盖用户修正、上下文冲突和前文约束撤销。否则模型会把“每个请求都能一次性完成”当成隐含假设，部署时遇到异常状态就容易编造结果。

9.2 合成数据

Self-Instruct、WizardLM、Orca 和 STaR 显示了合成指令、复杂任务演化、教师解释蒸馏和自举推理数据的价值 [240, 248, 174, 260]。合成数据管线必须记录生成器、prompt、

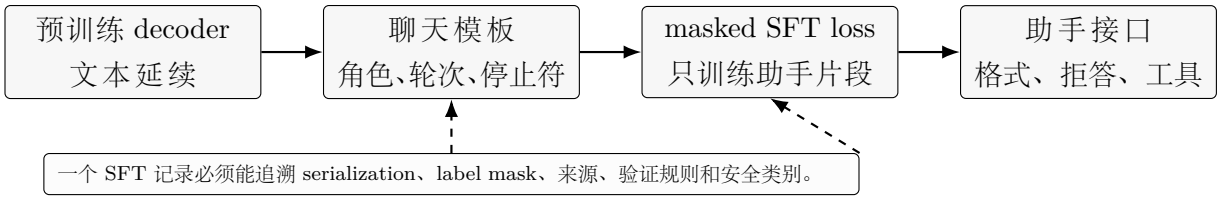

图 9.1: SFT 改变的是接口分布，而不是自回归语言建模的基本分解。

采样参数、过滤规则、拒绝原因和评测重叠。

Self-Instruct 的原始流程给出了可复现的数据契约：从 175 个种子任务开始，循环采样种子和已生成指令作为 in-context 示例，让模型生成新指令；再判断任务是否为分类任务，并用 input-first 或 output-first 模板生成实例；最后按相似度、格式、长度和输入输出一致性过滤。过滤后的公开运行约有 52K 条指令和 82K 个实例。因此报告不能只说“用了 Self-Instruct”，还应写明种子集、in-context 采样比例、任务类型判断 prompt、实例生成模板、相似度阈值、无效样本规则和最终指令/实例数量。

本地 Alpaca 生成脚本把这些字段进一步具体化。生成 prompt 会把若干种子样本序列化成 instruction、input、output 三段，空输入显式写成 no-input 标记；采样时记录模型名、温度、top-p、最大生成长度、批大小和停止串；后处理会丢弃被长度截断的最后一条、字段数不完整的样本、过短或过长的指令、以标点或非英文字符开头的指令，以及图像、音频、地图、流程图等不适合纯语言模型完成的任务。相似度过滤用 ROUGE-L，对已有指令最高相似度超过阈值的候选不保留。

这类脚本还说明“保留率”本身是数据质量信号。每轮请求应记录候选数、保留数、处理耗时、最相似历史指令和平均相似度；中间结果应写入可恢复文件，避免 API 中断后重启生成改变数据分布。若同一个 seed、prompt 和采样参数不能复现近似的候选池，后续 SFT 结果就很难归因。

中文 Alpaca 这类语言适配版 SFT 还要求报告语言和模板契约。它以中文 LLaMA 为基础，混合翻译数据、公开中文任务、Alpaca 风格数据和抓取/教师生成的 SFT 记录，Plus 版本进一步加入更多 STEM 与科学数据 [26]。该工作使用无 input 字段的 Alpaca prompt 模板，需要 input 时把 instruction 和 input 用换行合并。因此多语言 SFT 报告应写明模板变体、语言比例、数据是翻译还是原生目标语言、最大序列长度、dynamic padding 规则，以及是否排除了 benchmark-like 公共任务。

中文 Alpaca 数据文件本身也提醒我们不要只写“51K 指令”。样本字段仍是 instruction、input、output，但中文翻译、原生中文任务、长列表答案和空 input 样本的比例会改变 token 长度分布。训练脚本若要求特定中文 tokenizer 词表规模、额外 pad token 或扩展 embedding，就必须把 tokenizer 路径、词表大小、special-token 表和 embedding resize 规则写进 run card；否则同名 Alpaca 数据在不同 tokenizer 下不是同一个实验。

合成数据的风险随规模放大。教师模型会编造事实、写出不安全代码、误拒合法请求或把自身风格强加给学生模型。可靠管线通常需要四道门：生成 prompt 规定任务族、难度、来源和格式；自动过滤去掉重复、过短、格式错误或越界样本；任务验证器执行代码、SQL、JSON 或数学检查；人工抽样检查估计残余错误。目标不是让数据看起来多样，而是补上当前模型缺少且可验证的覆盖。

合成指令数据可以写成“提议分布加过滤器”。生成器从种子样本和生成策略中采样候选，再由一组验证器接受或拒绝；只有通过所有必需验证门的候选才进入接受集。因此，合成数据质量不由流畅度决定，而由验证门决定。代码样本需要执行测试，数学样本需要可检查答案，安全样本需要相邻合法/非法案例，RAG 样本需要证据支持。小而严筛的数据集有时比大而宽泛的合成数据更有价值。

蒸馏还会改变学习对象。学生模型可能只模仿教师表面风格，而没有获得教师的隐含能力；可能继承教师的安全边界；也可能因为接受集只保留容易成功的样本而变得脆弱。推理和工具使用样本最好同时包含问题、上下文、工具观察或验证器结果、最终答案和简短有效性说明。安全敏感领域的教师生成样本还需要额外人工审查，因为流畅语言会掩盖无证据的医疗、法律、金融或安全建议。

Platypus 的 Open-Platypus 混合数据聚焦 STEM 和逻辑，主要使用人工设计题，删除精确重复和高相似指令，并在 LoRA 微调和模型合并前检查训练题与 benchmark 题目的重叠 [143]。这不是要求所有模型都做 STEM 专家，而是提醒 SFT 报告必须给出来源级样本数、许可证、领域过滤规则、去重方法、benchmark contamination 检查，以及被合并 adapter 的训练数据是否可审计。

合成数据还需要“负面日志”。只公布通过过滤的样本，会让读者看不到生成器的真实错误率。更有用的报告应列出候选数、接受数、按规则拒绝的数量、最常见的拒绝原因、人工抽样错误率，以及不同任务族的接受率差异。如果代码样本主要因为测试失败被拒绝，说明教师模型或 prompt 还不可靠；如果安全样本主要因为边界不清被拒绝，说明需要更多 near-neighbor 对照，而不是继续扩大同类模板。

训练小模型时，合成数据的“容易样本偏置”尤其明显。过滤器倾向保留教师已经答对、格式规整、验证器容易检查的样本，这会让学生在标准题上进步，却不一定学会歧义处理、证据不足、长上下文约束或复杂工具恢复。一个实用做法是把接受集分成简单、困难、失败恢复和边界安全四类，分别监控验证集损失和错误类型，而不是把所有合成样本混成一个总分。

9.3 拒答、安全与评测

拒答数据不能只训练“正常问题回答、危险问题拒绝”的二元风格。真实系统需要更细边界：违法或危险操作细节应拒绝，附近的安全教育请求应回答，风险依赖上下文时应追问，拒答本身也不能变成空泛说教。有效安全样本应包含 near-neighbor 对比，例如“如何绕过访问控制”应拒绝，“解释访问控制原则”应回答，“审计我自己的配置”可能需要确认权限。

SFT 评测至少回答三个问题：模型是否遵循指令，基础能力是否保留，安全边界在分布外是否稳定。格式任务应优先用确定性验证器，例如 JSON parser、SQL 执行、单元测试或 schema validator；开放式问答可以用人工或高质量 rubric，但评审 prompt 不能泄漏目标答案。只看聊天主观评分容易奖励礼貌、冗长和迎合。

能力保留要和 base model 做对照。SFT 可能让聊天更自然，同时损害事实召回、翻译、代码、数学、校准或低资源语言。安全评测也要分开统计 under-refusal 与 over-refusal：一个敏感领域里什么都拒绝的模型并不真正可用，一个什么都回答的模型也不安全。评测集必须隔离训练数据；即使去掉原题，翻译、改写和合成变体仍可能泄漏。

9.4 局限

SFT 学会的是示范分布。它可能提升格式和礼貌，却不一定提升真实偏好、事实性或安全性。评估 SFT 不能只看指令榜单，也要看基础能力回归、拒答边界、长上下文、多语言和代码能力。

SFT 还会产生能力遗忘。一个模型在聊天格式上变好，可能同时降低翻译、代码、事实召回、数学或低资源语言能力。训练集越窄、越合成、越单一语言，回归风险越高。严肃报告应包含 base model 与 tuned model 的对照，并在未优化任务上保留回归测试。

9.5 深入展开：SFT 是接口训练，不是完整对齐

本章把监督指令微调定义为接口转换。预训练模型已经能延续文本，但它不知道什么是 system instruction、什么是用户请求、什么是可接受回答、什么是工具返回。SFT 通过示范把这些格式变成条件生成问题。训练时通常只对 assistant token 计算 loss，让用户问题、系统消息和检索证据作为条件，而不是让模型学习“预测用户”。

Chat template 是本章的关键。模板应明确角色、轮次、结束符、工具调用、工具观察、多模态占位符和安全上下文。训练模板与推理模板不一致，会让一个本来正常的模型出现角色混淆、提前结束、工具调用格式错误或拒答边界失效。因此要求把模板和 tokenizer、special tokens、模型权重一起版本化。合成数据扩展覆盖，但也复制教师错误。Self-Instruct

通过少量种子指令生成大量任务；WizardLM 类方法把任务演化得更复杂；Orca 类蒸馏让学生学习更丰富的教师解释；STaR 用正确答案筛选和迭代生成推理轨迹；Platypus 则强调小规模、高质量、去重和污染检查。它们共同说明，现代 SFT 数据管线是生成、过滤、验证和检查的系统，而不是一次性调用一个强模型。本章还提醒，模仿不能替代偏好和真实性。SFT 可以让回答更像人类示范，但示范本身可能过时、错误、风格单一或安全边界含糊。模型可能学会“看起来有帮助”的语气，却没有更强事实性；也可能学会教师模型的过度拒答。SFT 后必须做独立评测，覆盖事实、代码、数学、多语言、长上下文、安全和未优化任务。

9.6 章节细节

9.6.1 接口转换

SFT 把预训练模型转化为遵循指令的接口模型。它通过示范学习角色、格式、拒答、工具调用和回答风格。训练时通常只在 assistant token 上计算 loss，让其他消息作为条件。

9.6.2 指令数据作为契约

一条指令样本不只是 prompt 和 answer，还包含系统策略、角色边界、上下文、目标格式和安全期望。数据格式定义了模型未来如何解释用户请求。模板和 special tokens 必须版本化。

9.6.3 合成指令数据

合成数据能扩大任务覆盖，但会复制教师错误和风格偏置。Self-Instruct、WizardLM、Orca 和 STaR 展示了生成、演化、蒸馏和验证的不同策略。合成数据必须过滤、去重、检查和独立评测。

9.6.4 训练细节

SFT 训练要处理 loss mask、packing、学习率、epoch、长度分布和混合数据。过多 epoch 可能导致风格过拟合和能力遗忘。应比较 base model 与 tuned model 在未优化任务上的回归。课件里的几个实现坑应写进检查清单。第一，assistant span 必须包含 end-of-turn/EOS 监督；如果标签里没有停止 token，模型可能一直生成到 `max_new_tokens`。第二，训练时 right padding 和批量推理时 left padding 都可以成立，但 attention mask、position ids、labels 和 KV cache 初始化必须一致。第三，PEFT 的可训练子空间更小，常常不能只用

“同样 epoch 数”比较全参微调和 LoRA，应报告 target token 数、验证曲线、停止行为和回归指标。

Completion-only collator 让 loss mask 更容易实现，但也更容易被模板细节破坏。常见做法是寻找 `###Answer:` 这类 response delimiter 的 token ids，并把 delimiter 之前的标签全部设为 ignore。这个规则必须用同一个 tokenizer、同样的前导换行、空格和特殊 token 设置做单元测试。若同一项目同时有 packing=True 的 ConstantLengthDataset 路径和 delimiter-based collator 路径，报告应说明实际使用哪一个；前者提高 token 利用率，后者给出更干净的 assistant-only loss，把两者结合时必须显式检查样本边界。手写 chat template 时，token 级拼接通常比先拼字符串再 tokenize 更可靠。如果预处理先插入 `<EOS>` 或自定义 label-start marker，再调用 tokenizer，这个 marker 可能被拆成多个 token、被正规化，或者没有变成预期的特殊 token id。更稳的教学实现是分别 tokenize 角色前缀和内容，在 token-id 层拼接，同时构造 `is_label` 向量；collate 时再把非 label 位置设为 ignore index，并按框架约定做 label shift。最小单元测试应打印 decode 后的序列、特殊 token id、assistant mask 和最终 labels，否则 SFT 可能悄悄训练错 span，或者漏掉 EOS 监督。

另一类实现直接在预处理阶段构造 input ids 和 labels。Alpaca 风格脚本会先把 instruction 与可选 input 填进固定 prompt，再把 output 加上 EOS 作为 target；source tokens 对应的 labels 全部设为 ignore index，target tokens 保留为监督目标，最后按最大长度截断。collator 再用 pad token 补齐 input ids，用 ignore index 补齐 labels，并由非 pad 位置生成 attention mask。这里的关键不是代码长短，而是 source/target 的 token 化边界、EOS 是否进入 target、截断发生在哪一侧，以及 label 长度是否始终和 input ids 对齐。

TRL 风格 SFTTrainer 暴露了 packing 与 completion-only 两条不同路径。使用 constant-length packing 时，短样本被拼成固定长度块，吞吐更好，但必须说明样本边界、EOS 和跨样本 attention 语义；使用 response delimiter collator 时，loss mask 更清楚，但 delimiter 的 token ids 必须在真实 tokenizer 下验证。若训练脚本一处设置 tokenizer left padding，另一处又按 right padding 训练或生成，报告必须解释 attention mask、position ids 和 KV cache 如何保持一致。

官方 TRL SFTTrainer 文档还把这些选择变成了可检查配置 [128]。数据可以是普通文本、对话消息、提示-补全样本，或已经处理好的 token ids；因此 run card 应先写清数据 schema，而不是只写数据集名称。对对话数据，`assistant_only_loss=True` 依赖模板能返回 assistant token mask，通常需要 `{% generation %}` 与 `{% endgeneration %}` 包住 assistant 输出；对提示-补全数据，completion-only loss 是默认语义，若要训练全序列必须显式改变 `completion_only_loss`。这些开关决定 loss mask，不是普通训练超参数。

TRL 的 dataset formats 文档把这件事拆成“format”和“type”两层：standard 与 conversational 描述样本如何表示，language modeling、prompt-only、prompt-completion、pref-

erence、unpaired preference 和 stepwise supervision 描述训练任务需要哪些列 [117]。SFTTrainer 主要消费 language modeling 或 prompt-completion 类型；prompt-only 更适合生成或 RL 类 trainer，不能被误当成已有目标回答的 SFT 样本。同一组 messages 经过 `apply_chat_template` 后，prompt-only 会在末尾加入 assistant 起始位置，language modeling 则可能把输入当作完整序列结束。出版级报告因此应记录原始列、转换函数、转换前后样例、是否先转换再套模板，以及 `formatting_func` 是否把数据显式改成 language-modeling 语义。否则一个数据集"能被 trainer 接收"，也不代表 loss mask、生成边界和评测集构造是同一个任务。

SFTConfig 的 preprocessing 字段也应进入出版级验收。`dataset_text_field` 决定哪一列被当作文本；`chat_template_path` 可能替换 tokenizer 内置模板，若模板引用新 special token，就要扩展 tokenizer 并 resize embedding；`eos_token` 必须和模板里的轮次结束符一致，否则模型可能学会内容格式却不会停止；`pad_token` 缺失时常会退回 EOS，这会影响 attention mask 和生成边界。packing 还要记录 `packing_strategy`、`max_length`、eval 是否 packing，以及是否启用 padding-free 路径；后者依赖能处理展平 batch 的 attention kernel，不能只写成"更省显存"。

现代 SFT 训练还可能改变 loss 计算本身。`loss_type=chunked_nll` 与普通 NLL 数学目标一致，但通过跳过 ignored-label token 和分块交叉熵降低峰值 activation；视觉语言模型训练会默认把数据预处理留到运行时，并使用视觉语言 collator。若报告没有写明 causal LM 还是 VLM、是否 chunked loss、collator 在 padding 前还是后截断、是否使用 PEFT adapter、是否恢复已有 `PeftModel`，读者无法判断同一条 loss 曲线是否可复现。

工具调用和视觉语言 SFT 也已经进入同一个训练器接口，但它们的证据不能和纯文本样本混在一起。官方 SFTTrainer 文档要求工具调用样本包含带 `tool_calls` 的对话消息、`tool` role 返回，以及 `tools` 列中的 JSON schema；VLM 样本则需要 `image` 或 `images` 列，并提醒不要让截断删掉图像 token，必要时把 `max_length` 设为 `None` [128]。MoE 模型还有一个容易遗漏的训练契约：若要把 router load-balancing 或辅助损失计入最终 loss，模型配置需要输出 router logits。发布记录应把这些字段拆开写：文本 span、工具 schema、工具返回 span、媒体列、处理器、图像 token 是否被截断、router 辅助损失是否启用、以及日志里的非 mask token loss、entropy、mean token accuracy、num tokens 和 grad norm。这样才能区分"学会回答""学会调用工具""学会看图"和"路由负载稳定"四件事。

QLoRA 类脚本还把长度和标签策略变成显式参数。Source 与 target 可分别设置最大长度，`train_on_source` 决定 prompt 是否参与 loss，group-by-length 会改变批内长度分布和吞吐，MMLU 或其他回归回调会临时改变 source 长度预算。出版级报告应写明这些开关，而不是只给一个学习率和 epoch 数；否则读者无法判断提升来自数据、mask、长度截断、批组成，还是 LoRA 子空间。

长度控制也是训练目标的一部分。从右侧截断可能删掉目标答案，从左侧截断可能删掉指令，截断检索证据可能让正确答案变成无证据回答。预处理报告应给出被截断样本比例、丢失 target token 数、prompt/answer 长度分布，以及受影响最严重的样本。混合数据还要同时报告样本数和 target-token 数；少量长代码答案可能主导 token-weighted loss，大量短分类样本也可能主导 example-weighted sampling。

PEFT 版本的 SFT 还要把 checkpoint 语义写清楚。Chinese-LLaMA 类脚本会指定 LoRA 作用到哪些投影层、rank、alpha、dropout，并把 embedding 与 LM head 放入额外保存模块；QLoRA 脚本可能只保存 adapter checkpoint，并删除普通全量权重文件。若继续训练只恢复 adapter，而没有恢复 optimizer、scheduler 和随机状态，曲线就不是同一次实验的延续。若使用 DeepSpeed ZeRO，还可能需要先从 ZeRO checkpoint 取回 FP32 state，再抽取 PEFT 权重并保存 tokenizer。

因此 SFT 的恢复测试至少要覆盖三条路径：从最近 checkpoint 继续训练，导出 adapter 并在基础模型上加载，必要时把 adapter merge 或转换成推理栈可加载的权重。每条路径都应生成同一个固定 prompt，检查模板、special tokens、EOS、pad token 和 stop sequences 是否一致。一个只保存了 adapter 文件但没有保存 tokenizer 和模板的结果，不足以支持发布。

9.6.5 拒答与安全数据

安全数据定义模型何时拒答、如何提供安全替代、如何处理敏感但合法的请求。过少会造成越界，过多会造成误拒。拒答质量应按政策类别和用户意图切片评测。

安全 SFT 的目标是校准边界，而不是制造固定话术。医疗、法律、金融、教育和企业内控场景常常需要不确定性、证据引用、权限确认和人工升级，而不是简单回答或拒绝。若只用通用安全样本训练，模型可能语气成熟却在专业证据之外给建议；若只加入拒答样本，模型又可能把合法请求误判为危险。

拒答样本应按政策边界组织，而不是按关键词组织。一个好的安全切片至少包含明确禁止、明确允许、需要澄清、需要转向安全替代、需要引用权威来源、需要人工升级几类。评测时也要分开统计 harmful compliance、over-refusal、无帮助拒答、泄露敏感细节和拒答后继续给出危险步骤等错误。只报告一个“安全通过率”会掩盖模型到底是过度保守，还是在关键边界上放行了危险请求。

多轮安全数据还应训练状态更新。用户可能先提出合法问题，再逐步转向危险操作；也可能先被模型误拒，然后补充授权背景。SFT 样本需要展示模型如何重新读取上下文、确认权限、承认前一轮错误并修正答案。否则模型容易在多轮对话中固守第一判断，或者被用户逐步诱导越过边界。

9.6.6 评测

SFT 评测应覆盖格式遵循、事实性、代码、数学、多语言、安全、长上下文和回归任务。单纯看聊天偏好容易奖励礼貌和冗长。独立测试集和人工错误分类仍然必要。

评测报告应拆开 held-out 指令、格式验证、人工偏好、自动 benchmark、回归集和安全红队。每个集合都应说明来源、创建时间、是否与训练混合去重、是否含 paraphrase 或翻译泄漏检查，以及评分者是否看到参考答案。若 SFT 使用了公开评测题或其合成变体，后续分数只能说明训练拟合，不能说明未见任务泛化。

评测还要看“接口是否稳定”。同一个请求在不同系统消息、不同语言、不同输出格式、不同温度和不同上下文长度下，应该保持相同的角色边界和停止行为。格式任务可以用 JSON parser、SQL execution、unit test、regex schema 和引用覆盖率直接验证；开放式任务才需要人工或模型裁判。若使用模型裁判，报告应说明裁判模型、rubric、抽样复核比例、与人类判断的一致率，以及裁判是否偏好冗长回答。

生成侧 smoke test 是 SFT 评测的一部分。推理脚本通常会把 tokenizer 的 EOS、显式 end-of-turn token 和其他结束标记一起传给生成函数；若训练时监督的是另一个停止符，模型可能学会答案格式却不会停。评测应记录解码模板、stop token ids、最大新 token、是否跳过特殊 token、平均生成长度、被长度截断比例和错误停止案例。停止失败不是服务小问题，而是训练目标和推理模板没有闭合。

SFT 后的回归 dashboard 至少应包含 base model 对照、目标任务提升、未优化能力、拒答边界、平均回答长度、无效格式率、停止失败率和延迟变化。平均分上升但回答长度翻倍，可能只是模型变得更啰嗦；格式分上升但基础数学下降，可能是窄数据过拟合；安全分上升但合法请求大量误拒，说明边界没有校准。发布报告要把这些 tradeoff 显示出来。

9.6.7 模仿的局限

SFT 学的是示范分布，不保证真实性、最优性或安全性。示范者会犯错，教师模型也会产生幻觉。后续偏好学习、验证器、RAG 和工具约束常用于弥补这些局限。

模仿还会固化标注流程的盲点。若示范答案总是假设证据充分，模型会少问澄清问题；若示范答案总是长篇解释，模型会在简单任务上过度回答；若示范者没有看到工具日志，模型会学会编造外部观察。SFT 因此应被看作接口启动阶段。它让模型进入可比较、可偏好优化、可工具化的助手协议，但不能单独证明模型可靠。

9.7 SFT 发布证据与回流

SFT checkpoint 不应因为 held-out loss 下降或指令榜单上升就直接进入产品。发布对象是“基础模型、tokenizer、chat template、停止符、adapter 或全量权重、检索/工具权限

发布项	必须同时查看	阻断或复核条件
模板与停止符	训练模板、推理模板、EOS、end-of-turn、stop sequences	模板不一致、停止失败率升高、assistant 角色边界混乱
Label mask	渲染样本、assistant mask、ignore-label 分布、EOS 标签	用户或系统文本进入 loss、target 被截断、EOS 被忽略
TRL 数据	数据 schema、模板、EOS、collator、padding	模板无法返回助手 mask、EOS 不匹配、collator 截断位置不清
TRL 损失	助手损失、补全损失、打包策略、分块 NLL	补全语义不清、打包破坏样本边界、loss 类型未记录
工具与多模态 SFT	工具调用字段、工具返回、schema、媒体列、处理器、截断规则	工具返回被训练成自然语言、图像 token 被截断、schema 与服务端不一致
数据混合	样本数、target-token 数、语言和任务比例、许可证与来源	单一来源支配 loss、许可证不清、benchmark 近重复未排除
合成质量	生成器版本、过滤规则、拒绝原因、人工抽检错误率	教师幻觉高、验证器覆盖低、安全样本边界不清
安全边界	危险放行、良性误拒、相邻样本、多轮修正	只提高拒答率、合法请求误拒、拒答后泄露危险步骤
能力回归	基础模型对照，代码、数学、翻译、事实和低资源语言切片	目标任务提升伴随未优化能力显著下降
服务行为	平均长度、无效格式率、p95 延迟、成本、失败日志	回答变长导致成本超标、格式修复依赖后处理、错误不可回滚

表 9.1: SFT 发布项、同步证据与阻断条件检查表。

和安全策略”的组合，而不是一个孤立权重文件。上线前应把训练证据、离线评测、回归测试和服务行为放进同一张 go/no-go 表；否则团队很容易把格式跟随的提升误读成可靠性提升。

表 9.1 把 SFT 上线前的训练、评测和服务证据放到同一个 go/no-go 表中。

上线后的用户反馈也不能直接倒进下一轮 SFT。投诉、点赞、重试、人工改写和客服工单都含有选择偏差：沉默用户的问题不可见，高风险用户的反馈可能被审核流程过滤，模型错误后的用户改写也可能带入策略诱导。回流样本应先进入 failure card 和数据审核队列，标注失败类型、权限边界、是否可训练、是否需要删除个人信息、是否更适合进入评测集或红队集。只有通过来源、隐私、许可证和安全复核的样本，才应进入新的 SFT 混合数据。

SFT 发布记录还应保留反事实。若候选 A 指令分更高但代码回归明显，候选 B 安全边界更稳但回答略短，最终选择哪一个需要写清理由。这样的记录能避免下一轮团队只看

到“最新 checkpoint”，却不知道旧 checkpoint 在哪些切片上更好。对出版级实验来说，这些 go/no-go 证据和训练 loss 一样重要。

9.8 关键术语、实现要点与练习

关键术语。 Demonstration data 是示范回答；SFT dataset format 描述样本是 standard 字符串还是 conversational messages；dataset type 描述样本属于 language modeling、prompt-only、prompt-completion 或其他训练任务；assistant-only loss 只训练助手输出；completion-only loss 只训练 prompt-completion 样本的 completion 部分；label mask 定义哪些 token 计入 SFT loss；ignore index 是训练框架中让某些 label 不计入 loss 的哨兵值；chat template 是角色、分隔符、工具观察和停止符的确定性序列化；generation prompt 标记推理时 assistant 回复即将开始；assistant prefill 表示继续最后一条 assistant 消息；response delimiter 是 completion-only collator 用来定位助手答案起点的标记；packing 把短样本拼成固定长度块以提高 token 利用率；padding-free training 把 batch 展平成无 padding 序列以降低浪费但依赖兼容 attention kernel；chunked NLL 用分块交叉熵降低峰值 logits/activation；tool-calling SFT 用工具调用消息、工具返回和工具 schema 训练模型遵循调用协议；VLM SFT 用图像列和处理器把媒体 token 纳入监督；self-instruction 用模型生成新任务；distillation 从教师模型学习行为；adapter checkpoint 只保存 PEFT 可训练权重；capability regression 指微调后基础能力下降；benchmark contamination 指训练样本与评测样本重叠导致的虚假提升。

实现要点。 SFT 数据应记录模板、生成器、过滤器、验证器和人工检查；合成数据必须检测事实错误和安全边界；训练报告要给出 data schema、format/type、转换函数、数据文本列、模板路径、助手损失和补全损失开关、label mask、EOS 监督、delimiter token ids、packing strategy、padding-free 条件、chunked loss、source/target 长度预算、截断率、target-token 分布、工具 schema、媒体列、处理器、router auxiliary loss、LoRA 保存模块、恢复/导出测试和回归测试；评测要包含未优化任务回归；模板、tokenizer、special tokens、EOS、pad token 和 stop sequences 必须和模型权重一起保存。

练习。

1. 构造一个包含工具观察的 SFT 样本，并标出 loss mask。

2. 设计一个合成数据过滤管线，包含格式、重复、事实和安全检查。

3. 找二十个 SFT 样本，判断它们是否真正增加任务覆盖。

4. 设计 SFT 前后回归测试，覆盖代码、数学、多语言和拒答。

5. 给一个两轮对话写出序列化模板，并标出哪些 assistant 轮次作为 target，哪些只作为上下文。

6. 设计三条 near-neighbor 安全样本：一条拒绝、一条回答、一条追问，并解释为什么不能只靠关键词判定。

7. 比较全参 SFT 与 LoRA SFT，应报告哪些训练成本、checkpoint 大小、验证曲线和基础能力回归指标?

8. 为一个 5 万条样本的 SFT 混合数据写 manifest schema，字段至少覆盖来源、语言、任务族、target-token 数、许可证、安全类别、验证器和 benchmark 去重状态。

9. 设计一个 completion-only collator 的单元测试，要求打印 token 序列、response delimiter token ids、assistant mask、EOS 标签和最终 ignore-label 分布。

10. 对比 packing=True 和 completion-only collator 两条 SFT 路径，列出吞吐、样本边界、EOS 监督、跨样本 attention 和调试证据各自应如何报告。

11. 为一个 TRL SFTTrainer 训练写配置 manifest，至少包括 data schema、chat template、`assistant_only_loss`、`completion_only_loss`、packing strategy、padding-free、`eos_token`、`pad_token`、loss type 和 collator 行为。

12. 取一个 conversational prompt-completion 样本，分别说明它作为 SFT 样本、prompt-only 样本和 language-modeling 样本时，`apply_chat_template` 的输出边界和 loss mask 应如何不同。

13. 为一个带工具调用和图片输入的 SFT 样本写验收清单，覆盖 `tool_calls`、tool role、`tools` schema、`image/images` 列、processor 输出键、图像 token 截断和 assistant label mask。

14. 一个 LoRA SFT 作业只保存了 adapter checkpoint。列出恢复训练、导出推理模型和复现实验时还必须保存或验证的 tokenizer、模板、optimizer、scheduler、随机状态和基础模型信息。

15. 复盘一个 Self-Instruct 风格数据生成批次，报告候选数、保留数、相似度阈值、黑名单过滤、最常见拒绝原因和人工抽样错误率。

16. 给一个 SFT 发布报告画回归 dashboard，至少包含目标任务、未优化任务、安全过拒/欠拒、格式有效率、平均回答长度和停止失败率，并说明每个指标的 go/no-go 阈值。

9.9 结构化检查表

9.9.1 SFT 数据类型与风险

数据类型	价值	风险
人工示范	高质量、符合目标风格	贵、覆盖有限、标注者偏差
合成指令	覆盖快、格式多	教师幻觉、风格单一、污染
工具轨迹	学习函数调用与恢复	虚假工具结果、权限混淆
视觉语言示范	学习图文指令、OCR 或多图比较	图像 token 截断、媒体列与模板占位不一致
安全样本	学习拒答和重定向	过拒、边界模糊
领域示范	学习术语和流程	浅层风格模仿，无证据支持

第十章　参数高效适配

10.1　Adapter、Prompt 与 LoRA

PEFT 的目标是减少训练参数、显存和 checkpoint 体积。Adapter 在层内加入小模块；prefix/prompt tuning 学习连续提示；LoRA 把权重更新写成低秩矩阵乘积 $\Delta W = BA$ [53]。它适合多任务共享基础模型，但不免除数据和评测责任。

参数高效首先要说明“高效”的对象。训练显存取决于哪些参数需要梯度和 optimizer state；存储取决于能否共享一个基础模型，只保存小 adapter；服务延迟取决于 adapter 是否增加额外投影、额外上下文长度或未合并的低秩矩阵乘法；治理复杂度取决于需要管理多少 adapter 版本、权限、数据来源和回滚路径。因此 PEFT 报告不能只写 trainable parameters 百分比，还要写训练、部署和审计分别省了什么。

更稳妥的写法是给每个 PEFT 实验附一张效率账本。账本至少分成训练峰值显存、activation 峰值、optimizer state、checkpoint 大小、加载时间、tokens/s、p50/p95 延迟、adapter 路由开销、回滚路径和人工审计成本。若 LoRA 让训练显存下降，却让多租户服务无法高效 batching，或让安全审计必须跟踪几十个 adapter 组合，效率结论就不能只按显存成立。出版级比较应把 base、full fine-tune、单 adapter、合并 adapter 和量化 adapter 放在同一张表里。

可以把基础模型参数记为冻结的 θ_0，adapter、prompt 或低秩矩阵记为可训练的 ϕ。训练目标仍然是让 $p_{\theta_0,\phi}(y \mid x)$ 在领域数据上更好，但约束是 $|\phi| \ll |\theta_0|$。这个约束带来节省，也带来表达能力上限。小 adapter 可能足以学习 JSON 格式、客服语气或窄领域流程；它通常不足以修复差 tokenizer、注入大量新事实或彻底改写预训练中的有害偏差。

更可审计的写法是把 PEFT 看成受约束优化：冻结 θ_0，只优化 adapter 参数 ϕ，在目标数据 $\mathcal{D}$ 上降低平均负对数似然，并把 $|\phi| \ll |\theta_0|$ 当作系统约束，而不是质量保证。这样写能迫使报告区分三件事：目标数据 $\mathcal{D}$ 是否足以定义目标行为，adapter 子空间是否有足够表达能力，冻结 base 的通用能力是否被低秩或 prompt 更新间接扰动。

这个约束可以写成

$$\mathrm{NLL}_{\mathcal{D}}(\theta_0,\phi) = -\mathrm{avg}_{(x,y)\in\mathcal{D}} \log p_{\theta_0,\phi}(y \mid x), \qquad \phi^{\star} = \arg\min_{\phi} \mathrm{NLL}_{\mathcal{D}}(\theta_0,\phi), \qquad |\phi| \ll |\theta_0|.$$

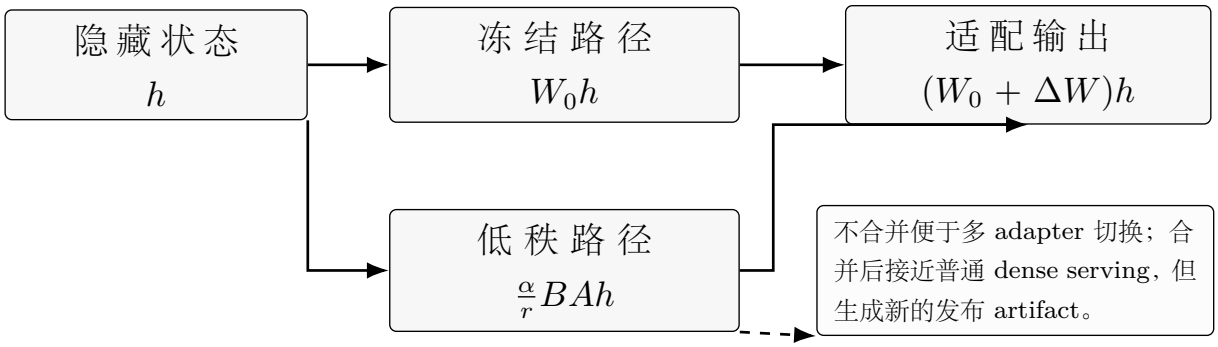

图 10.1: LoRA 在冻结矩阵旁增加低秩残差路径。训练省参数不等于部署无代价，adapter 管理、合并策略和回归测试仍然是发布对象。

Rank r	LoRA 参数 $2rd$	Full matrix d^2（$d = 4096$）	比例
4	32,768	16,777,216	0.20%
8	65,536	16,777,216	0.39%
16	131,072	16,777,216	0.78%
64	524,288	16,777,216	3.12%

表 10.1: LoRA rank 对单个方阵可训练参数比例的量级检查表。

式子里的“小”只是参数空间小，不是风险小。若数据 $\mathcal{D}$ 的模板、语言、领域、风险类别或标注策略有偏，adapter 会在很小的子空间里高效放大这些偏差。发布报告应把目标数据、冻结基础模型和 adapter 子空间一起说明，而不是只把 PEFT 写成一个省显存技巧。

因此 PEFT 不是“轻量微调”的同义词，而是一种发布对象拆分策略。基础模型、adapter、tokenizer、chat template、量化配置和安全策略共同决定最终系统。一个 adapter checkpoint 如果脱离这些上下文，文件再小也不是完整模型说明。

LoRA 的 rank、alpha、dropout 和 target modules 都是建模选择。只调 query/value projection 成本较低，但对代码、多语言或结构化生成可能不够；同时调 attention 和 MLP 更强，但更容易过拟合或破坏基础能力。adapter 合并到基础权重后，也要重新评估量化、服务延迟和安全过滤，因为合并改变了实际部署 artifact。

SVD 可以解释 LoRA 的直觉，但不能替代实验。如果全参微调的权重更新或某一步梯度快照具有快速衰减的奇异值，低秩 adapter 就可能用少量参数捕获主要更新方向；但冻结基础权重后学到的 LoRA 更新，不一定等于某个假想全参更新的 top singular vectors。低秩是一种建模假设，应通过改变 rank、target modules 和数据规模来验证，并同时比较目标任务收益和回归损失。图 10.1 展示了冻结路径和低秩残差路径在推理时如何合成输出。

表 10.1 给出 LoRA rank 的量级感，提醒读者区分可训练参数和部署状态。

10.2 QLoRA 与量化训练

QLoRA 把基础模型量化为低精度并训练 LoRA adapter，从而显著降低显存需求 [32]。量化带来效率，也带来数值误差、目标模块选择和部署合并问题。报告 PEFT 结果时，应同时给出任务指标、回归指标、显存、训练时间、checkpoint 大小和服务延迟。

LoRA 减少的是可训练参数和 optimizer state，不自动消除基础模型权重的存储开销。QLoRA 进一步把冻结 base 存为 4-bit 表示，只让梯度更新低秩矩阵。粗略看，全参微调要保存权重、梯度和 Adam 一阶/二阶状态；QLoRA 的主项变成 4-bit base、adapter、adapter 梯度、adapter optimizer state 和 activation。长序列、大 batch 和 checkpointing 仍会让 activation 成为峰值显存来源，所以“能放下模型”不等于“训练配置已经稳健”。

NF4、double quantization 和分页优化器应分别报告。NF4 是面向近似正态预训练权重的存储格式，不等于所有计算都在 4 bit 中完成；典型 QLoRA 路径会以 NF4 存储冻结权重，在矩阵乘法前反量化到较高 compute dtype，并把梯度通过反量化路径传给 LoRA 矩阵。Double quantization 通过再次量化每个 block 的量化常数，减少 scale/zero-point 元数据开销。分页优化器处理的是峰值显存，而不是平均显存：长序列或 gradient checkpointing 造成 activation 峰值时，optimizer states 可以借助统一内存在 CPU/GPU 间换页。可复现报告应写清 storage dtype、compute dtype、block size、double quant、target modules、是否覆盖所有 linear 层、gradient checkpointing、paged optimizer、最大序列长度和 peak memory。

AF4 对 NF4 的讨论提醒我们不要把“NF4 最优”写成脱离实现的定理。Absmax blockwise quantization 会先除以 block 最大绝对值，而这个归一化后分布依赖 block size，所以固定 codepoints 不可能对所有 block size 都最优 [258]。这不否定 QLoRA 的经验效果，但要求报告写清具体 quantization type、block/group size、scale 存储和反量化路径。在常见 BitsAndBytes 风格实现中，还应记录 4-bit 加载开关、quant type、compute dtype、double-quant 开关，以及是否使用 paged AdamW 32-bit 这类分页优化器。低比特加载同样是 placement 契约。脚本里的 8-bit 加载开关、自动 device map 和每张 GPU 的 memory cap，不只是改变算术精度，也决定层放在哪些设备、预留多少显存余量，以及是否会因为 CPU offload 或不均匀切分影响延迟。QLoRA 训练脚本也常把 `bits=4`、NF4、double quant、BF16 compute、gradient checkpointing、按长度分组 batching 和 paged optimizer 绑在一起。只报告“4-bit LoRA”会丢掉真正的系统配方。

PEFT 容易让实验看起来便宜，但数据和评测仍然决定可信度。一个 20MB adapter 可以携带强烈风格、过时知识或安全回归。多 adapter 系统还需要版本管理、冲突处理和权限控制：不同用户、领域或任务加载不同 adapter 时，系统必须知道它们基于哪个基础模型、哪个 tokenizer、哪个 chat template 和哪个安全策略训练。

每个 adapter 都应有 manifest。最小字段包括 adapter id、base checkpoint hash、tok-

enizer hash、chat template 版本、target modules、rank、alpha、dropout、训练数据 manifest、许可证摘要、训练脚本版本、保存的参数范围、量化/合并状态、评测摘要、允许场景和回滚负责人。没有这些字段，adapter 很容易在形状兼容但语义不兼容的 base 上被加载，或在不该授权的租户、语言和风险场景中被启用。

本地 LoRA 实操文档也说明了发布边界。单张 24GB 显卡能用 LoRA 微调 7B 级模型，checkpoint 输出到 `out/`，再用 LoRA 生成脚本加载测试，这对教学很有价值；但发布对象并不只是这个目录。读者应同时保存基础权重来源、tokenizer、数据预处理脚本、train/validation 切分、prompt 模板、adapter 超参数、推理脚本版本和 dtype。若用 BF16 把显存压到更低，也要记录硬件、kernel 和 autocast 设置，否则别人只能复现“有一个 adapter 文件”，无法复现训练与推理条件。

10.3 深入展开：参数高效不等于风险高效

本章解释 PEFT 的动机：全参数微调需要大量显存、训练时间和 checkpoint 存储，而很多适配任务只需要改变少量行为。Adapter 在层中插入小模块，prefix/prompt tuning 学习连续提示，LoRA 用低秩矩阵表示权重更新。它们共同的目标是减少可训练参数和部署 artifact 大小。

Adapter 和 prompt 家族还要区分推理代价。瓶颈 adapter 通常是降维、非线性、升维再残差相加，参数少，但每次 forward 都要执行额外投影，通常不能像线性 LoRA 那样简单合并。Prefix tuning 把连续向量放进每层 attention 的 key/value 路径，像任务专属缓存一样被后续 token 看到；prompt tuning 只学习输入层 soft prompt，最便宜也最容易切换，但会占用有效上下文长度，在小模型上表达能力也可能不足。

瓶颈 adapter 的最小形式可以写成

$$\mathrm{Adapter}(h) = h + W_{\mathrm{up}}\sigma(W_{\mathrm{down}}h), \qquad W_{\mathrm{down}} \in \mathbb{R}^{b\times d}, \quad W_{\mathrm{up}} \in \mathbb{R}^{d\times b}, \quad b \ll d.$$

它的参数量远小于周围 attention 和 MLP 矩阵，但非线性和残差位置让它通常不能简单折叠回原始 dense 权重。也就是说，adapter 的训练成本、checkpoint 成本和推理成本是三张不同账本：训练时省 optimizer state，存储时省完整模型副本，服务时却可能多出层内投影、路由和 batching 复杂度。

LoRA 的数学形式 $\Delta W = BA$ 很简单，但工程选择很多。Rank 决定更新容量，alpha 决定缩放，dropout 影响正则，target modules 决定修改 attention、MLP 还是更多层。只调 query/value projection 省钱，但可能不足以适配复杂代码和多语言；调更多模块更强，但回归风险更高。不同任务的最佳 rank 和 target set 不能直接迁移。

初始化也属于方法定义。常见 LoRA 实现会随机初始化一个低秩因子、把另一个因子置零，让训练开始时 $\Delta W = 0$，模型输出与 base 完全一致。这样做使 adapter 成为逐步学

习的残差，而不是一开始就扰动基础模型。若报告没有说明初始化、LoRA dropout、bias policy 和是否训练 layer norm 或 scale 参数，不同实现之间的结果就很难比较。

现代 PEFT 工具链已经把这个契约写得更细。一个可复现的 `LoraConfig` 不只包含 rank 和 target modules，还要记录初始化方式、rsLoRA/DoRA 开关、额外保存模块、逐层 rank/alpha pattern、可训练 token 行、裸参数目标、aLoRA invocation tokens 和运行时配置 [87]。这些不是实现杂项，而是决定训练轨迹、可合并性、推理成本和兼容性的发布证据。

这些字段还应进入验收，而不只是配置转储。All-linear 目标写法应保存实际展开的模块名、排除的 LM head、fused QKV/Conv1D 处理；`target_parameters` 触达 MoE expert 时，应保存参数匹配、专家 rank 缩放和推理 kernel 支持；trainable token indices 应列 token id、special-token 角色、LM head 同步和 FSDP 的 `use_orig_params` 要求。

初始化方案也不能再只写“默认 LoRA”。默认做法让 LoRA B 为零，使 adapter 在训练前是 no-op；EVA 用层输入 activation 的 SVD 做数据驱动初始化并可按 explained variance 调整层间 rank；PiSSA、CorDA、OLoRA 和正交初始化改变的是低秩子空间的起点；LoftQ 则把初始化和量化误差联系起来。报告若使用这些方案，应写明初始化数据、是否改变 base weight、是否需要额外预处理、是否和量化一起使用，以及随机初始化是否只是调试用途。

LoRA 变体还改变了服务语义。rsLoRA 把缩放从 α/r 改成 $\alpha/\sqrt{r}$，目标是让较高 rank 更稳定；DoRA 把权重更新拆成 direction 和 magnitude，低 rank 时可能更强，但会增加推理开销，通常需要说明是否合并；Activated LoRA 只在 invocation token 之后启用 adapter，边界之前的 KV cache 可以和 base 复用，适合 agent 链中“前半段用基础模型、后半段调用专用 adapter”的流程，但它不能像普通 LoRA 那样直接 merge。若这类字段没有写进 run card，读者无法判断性能来自训练方法、调用边界还是 cache 复用。

QLoRA 把基础权重量化到低精度，再训练 LoRA adapter，使单机显存也能适配较大模型。但量化会引入数值误差，训练时的量化配置、推理时的加载方式、adapter 合并与否都会影响结果。本章要求报告显存、训练时间、checkpoint 大小、延迟和回归指标，而不是只报告任务分数。QLoRA 的大规模实验还提醒一个 PEFT 特有陷阱：训练便宜会让人轻易横扫许多数据集，但数据适配性可能比数据规模更重要；高质量小型指令集可能胜过更大的错配混合数据。

PEFT 的治理问题常被低估。Adapter 很小，所以容易被大量创建、下载、组合和热切换。一个 adapter 可以改变安全边界、输出风格或领域知识；多个 adapter 叠加可能相互冲突。生产系统应记录每个 adapter 的基础模型、tokenizer、模板、数据来源、训练目标、评测结果和允许使用的场景。

10.4 章节细节

10.4.1 到底在让什么高效

PEFT 让可训练参数、显存、训练时间和 checkpoint 体积变小，但不自动降低数据风险和评测成本。它适合多任务共享基础模型，也适合资源有限场景。效率指标必须和质量、延迟、治理一起报告。

10.4.2 Adapter 与 Prompt 家族

Adapter 在模型层中插入小模块，prefix/prompt tuning 学习连续条件向量。它们减少可训练参数，但也改变推理路径或上下文占用。选择方法时应看任务复杂度、部署形态和是否允许修改模型结构。

LLaMA-Adapter 是 prompt 方法和 adapter 方法之间的混合例子：冻结 LLaMA 权重，把可学习 adaptation prompts 注入高层 transformer，并用 zero-initialized attention gate 让未训练 prompt 在初期几乎不影响基础路径 [264]。这种设计把稳定性写成机制：模型先接近 frozen base，再逐步接收指令信号。报告这类方法时，应说明插入层、prompt 长度、gate 初始化、可训练参数量，以及是否把图像等非文本 token 加入 prompt 路径。

Adapter 名称本身不足以定义可训练状态。LLaMA-Adapter v2 风格实现除了 prompt embedding 和 gate，还可能训练 RMSNorm 参数以及每个 linear 层的 scale/bias 向量。因此“adapter-only checkpoint”应列出保存的参数子串或模块类别，而不是只写方法家族名。

还有一类 PEFT 方法把可训练变量写成乘法缩放或稀疏控制信号。IA^3 这类方法学习内部激活的缩放向量，用很少参数改变注意力或前馈路径中的信息流 [157]。它们的共同前提是基础模型已经有足够丰富的表示，adapter 只提供低维控制。若目标任务需要新增词表、复杂格式迁移或大范围事实更新，仅靠缩放向量通常不够；若任务只是少量示例下的风格、选择或领域偏好，缩放类方法可能比大 rank LoRA 更容易治理。

方法族	可训练对象	主要部署代价
瓶颈 adapter	层内降维、非线性、升维模块	额外投影和 adapter 路由
Prefix tuning	每层 key/value 连续前缀	更多 attention 状态和 prefix 管理
Prompt tuning	输入层 soft prompt embedding	占用上下文预算，需按任务加载
IA^3 类缩放	激活缩放向量	少量 elementwise 操作，但表达空间受限
LoRA	选定矩阵的低秩加性更新	不合并时有低秩矩阵乘法和 adapter 管理
QLoRA	量化 frozen base 上的 LoRA	量化 kernel、反量化路径和合并约束

10.4.3 LoRA

LoRA 把权重更新写成低秩矩阵乘积，冻结基础权重，只训练小矩阵。Rank、alpha、dropout 和 target modules 决定容量与风险。LoRA 不是只调一个 rank，而是一组训练和部署选择。

更完整的 LoRA 更新是

$$W' = W_0 + \Delta W, \qquad \Delta W = \frac{\alpha}{r} BA, \qquad A \in \mathbb{R}^{r \times d_{\text{in}}}, \quad B \in \mathbb{R}^{d_{\text{out}} \times r}.$$

对一个矩阵，新增可训练参数是 $r(d_{\text{in}} + d_{\text{out}})$，而不是 $d_{\text{in}} d_{\text{out}}$。因此 rank 增加带来的是线性参数增长，但目标模块数量增加会乘上这个成本；从 q/v 扩到 q/k/v/o 和 MLP，往往比把 rank 从 8 调到 16 更显著地改变容量、显存和回归风险。

Embedding 和 LM head 也可以作为 LoRA 目标，但风险更高。它们会直接改变词表表示或输出词分布，适合新 special token、领域词汇或多语言适配等场景；如果输入 embedding 与输出 head 共享权重，还要说明 tie weight 如何处理。PEFT 报告不能只写 target modules 为 `q_proj,v_proj` 或 `all-linear`，还应说明是否包含 embedding、LM head，以及合并后是否仍兼容原 tokenizer。

最新 PEFT 配置还把“训练少量 token”变成一等字段。若只为新增 special token、领域术语或控制 token 训练若干 embedding 行，应记录 `trainable_token_indices`、token id、tokenizer resize 历史、输入 embedding 与 LM head 是否 tied，以及是否要求 `ensure_weight_tying`。如果 tied embedding 和 LM head 使用不同 token 行，训练语义可能从“补几个 token”变成“让输入和输出词表分叉”。这种分叉在形状上可加载，却会直接改变生成概率和 tokenizer 兼容性。

Chinese LLaMA 的做法说明 LoRA 不等于“只冻结基础模型并训练两个 attention 矩阵”。在中文词表扩展场景中，LoRA 负责 attention 和 MLP 路径的参数高效更新，而扩展后的 embedding 与 LM head 仍保持可训练，使新增中文 token 真正获得表示和输出概率 [26]。因此报告应分开回答三个问题：哪些基础权重冻结，哪些低秩路径训练，哪些词表相关 dense 矩阵仍然更新；后续中文 Alpaca adapter 是否建立在同一个 tokenizer 和同一个语言适配底座上。实现布局会改变 target module 的含义。轻量 LLaMA 代码常把 query、key、value 存成一个融合的 `c_attn` 投影；如果只给 Q/V 加 LoRA，就不能只写 `q_proj,v_proj`。这类实现需要只为启用的 slice 训练低秩因子，形成低秩权重增量时把未启用的 K slice 补零，并在 merge/unmerge 时使用同一份 mask。Run card 应记录 QKV 是融合还是拆分、启用哪些 slice、scaling、dropout、bias policy 和 merge state；否则 adapter checkpoint 可能 shape 能加载，却接到了错误列上。

目标模块的自动发现也要谨慎。QLoRA 脚本常按量化位宽扫描 `Linear4bit`、`Linear8bitLt` 或普通 `Linear`，取叶子模块名作为 LoRA target，并排除 `lm_head`。这种 `all-linear` 很方便，却不是稳定语义契约：LLaMA、Qwen、DeepSeek、融合 QKV kernel 和自定义 MLP 命名都可能不同。出版级报告应保存展开后的模块名列表、排除规则和 trainable parameter summary，而不是只写一个字符串。

MoE 模型还会让 target modules 这个词不够用。有些专家权重不是普通 `nn.Linear` 模块，而是裸参数张量或融合专家 kernel；这时需要通过 `target_parameters` 指向具体专家张量，并用 rank pattern 控制专家层的参数预算。报告应保存参数匹配模式、每类专家的有效 rank、非专家层是否使用默认 rank、推理 kernel 是否支持这些专家 LoRA，以及专家并行和量化路径是否仍能合并或热切换。否则 adapter 看似覆盖了“所有线性层”，实际可能漏掉最关键的专家路径。

LoRA 的合并不是一个纯数学后处理，而是部署精度和发布对象的决定。合并时实际得到 $W_{\text{merged}} = W_0 + (\alpha/r)BA$；若基础权重以 BF16、FP16、INT8 或 NF4 形式存在，合并路径、舍入误差和后续量化都会影响输出。未合并 adapter 可以动态切换、灰度发布和按租户授权，但需要服务端正确路由 adapter id；合并 checkpoint 延迟更稳定，却生成一个新的模型 artifact，需要重新绑定模型卡、许可、评测和回滚规则。多个 adapter 只有在 target modules、scaling、tokenizer、chat template、许可和安全策略兼容时，才有资格尝试相加或串联；即使形状兼容，也必须实测干扰。

10.4.4 QLoRA 与量化训练

QLoRA 通过低精度基础权重和 LoRA adapter 降低显存。NF4 是存储格式，compute dtype、反量化路径、block size、scale 存储和 adapter 梯度路径也必须说明。双重量化降低量化常数开销，分页优化器缓解长序列和 checkpointing 带来的峰值显存。量化配置需要

和推理加载方式一致，否则评测结果可能不可复现。

全参微调和 QLoRA 的显存账本也应分开写。粗略地说，全参 Adam 微调要同时容纳

$$M_{\text{full}} \approx M(W) + M(\nabla W) + M(\text{Adam}_1) + M(\text{Adam}_2) + M(\text{activations}),$$

而 QLoRA 的主项变成

$$M_{\text{QLoRA}} \approx M_{\text{4bit}}(W_0) + M(\phi) + M(\nabla\phi) + M(\text{optimizer}(\phi)) + M(\text{activations}).$$

这两个式子解释了为什么 QLoRA 能让大模型进入单机训练，也解释了为什么长上下文、packing、gradient checkpointing、batch size 和序列长度分布仍然会决定峰值显存。若报告只给“7B 可在 24GB 上训练”，却不给最大序列长度、activation 峰值和 paged optimizer 行为，读者无法判断配方是否可迁移。

官方 PEFT 量化文档把这条路径落成了更具体的检查项 [94]。加载 4-bit base 时应记录 bitsandbytes 配置里的 4-bit 加载开关、量化类型、double-quant 开关和 compute dtype，并在训练前调用 k-bit training 准备步骤。LoftQ 若要广泛补偿量化误差，通常需要尽量覆盖 linear 层，常见写法是让 target modules 覆盖所有 linear 层，而不是只给 query/value 加 adapter。

LoftQ 的便捷路径也有边界。`replace_lora_weights_loftq` 可以在量化 PEFT 模型上就地替换 LoRA 权重，并保留原始量化 base；它只做一次 LoftQ 更新，而不是反复联合更新 LoRA 和量化 base weight。因此它更容易和既有量化权重兼容，但可能不是性能最强的 LoftQ 实验。报告还应说明模型文件是否为 safetensors、是否只支持 bitsandbytes 4-bit、是否使用 callback 控制修改层，以及哪些未 target 的层没有获得 LoftQ 补偿。

LoftQ 和 `all-linear` 的组合还要进入验收样例。若目标是补偿量化误差，应保存训练前量化配置、LoftQ 前后若干固定 prompt 的 logits 差异、展开后的 target module 列表、被 callback 排除的层、是否保留原始量化 base，以及 safetensors 文件约束。若只看到最终任务分数上升，无法区分收益来自 LoftQ 初始化、更多 target 层、数据顺序、学习率，还是量化 base 本身的随机差异。

QLoRA SFT 脚本还会把数据契约具体化。Source 和 target 往往用不同 token 上限截断，`train_on_source` 决定 prompt token 是否参与 loss，按长度分组会改变显存和吞吐。Adapter-only checkpoint 也容易误导恢复训练：如果恢复时只重新加载 adapter 权重，没有恢复 optimizer、scheduler 和随机状态，那么续训曲线就不是同一个实验。报告应说明 adapter 权重、optimizer state、scheduler state、随机状态、tokenizer 变更和保存/恢复测试是否都成立。

很多 PEFT 训练脚本会通过 callback 只保存 `adapter_model`，并删除完整的 `pytorch_model.bin`，以免误把冻结 base 复制进每个 checkpoint。这是正确的存储策

略，但它改变了恢复语义：`completed` 标记只能防止重复跑完一次任务，不能证明 optimizer、scheduler、随机数和数据迭代器状态已经恢复。实验记录应区分三件事：为推理加载 adapter、为评测加载 adapter、从同一训练步继续训练。

Transformers 与 PEFT 的直接集成进一步把生命周期分成 API 状态，而不只是文件状态：`PeftAdapterMixin` 让 `PreTrainedModel` 能添加、训练、加载、切换、禁用、查看和删除 adapter；Trainer 训练时只更新 adapter 参数，checkpoint 目录通常只包含 adapter 权重文件与配置文件，恢复时会扫描并重载 adapter checkpoint [92]。发布验收应记录 PEFT 版本、adapter slot 名称、active-adapter 输出、禁用 adapter 后的 base-only 对照、checkpoint 恢复 smoke test，以及 ZeRO-3/FSDP 下冻结权重是否被排除在保存对象之外。

PEFT checkpoint format 也要求把权重文件和配置文件一起审查。Adapter 权重文件只保存 adapter 参数而不保存 base；默认 safetensors 避免 pickle 反序列化风险；配置文件保存 adapter 类型、base 名称、revision 和具体方法配置；LoRA 的状态字典主要包含 `lora_A`、`lora_B`、embedding LoRA 因子等可训练项，而 rank、alpha、dropout 和 scaling 等来自配置 [93]。因此 manifest 不能只 hash 一个权重文件；它应同时 hash 权重、配置、model card、base revision、加载脚本和固定样例输出。

10.4.5 选择 Rank、目标模块与超参数

低 rank 省资源但可能欠拟合，高 rank 更强但增加回归和存储。只调 attention 投影、省钱但能力有限；调 MLP 或更多层可能适合复杂任务。选择应基于验证集和切片评测，而不是经验数字。

一个可信的实验网格应一次只改变少量因素：rank、target modules、learning rate、dropout、训练 token 数和是否量化。Adapter 学习率常高于全参微调，因为训练矩阵小且新初始化，但过高学习率会造成格式脆弱、loss spike 或安全漂移。训练日志应同时记录 trainable parameter count、peak memory、tokens/s、gradient norm、train loss、validation loss 和任务指标。

数据 collator 仍是正确性的核心。PEFT 不会修复错误 label mask、坏 padding、错误 chat template 或 source/target 拼接问题。LoRA SFT 通常需要冻结 base、打开 gradient checkpointing、训练时关闭 inference cache、按 batch 最长序列 padding、只保存 trainable adapter 权重。若只保存 adapter 而没有保存 optimizer、scheduler 和随机状态，恢复训练就不再是同一条曲线。

10.4.6 系统成本与部署

多 adapter 系统需要加载、合并、热切换、权限控制和版本管理。Adapter 小不等于风险小，它可以改变安全边界和领域行为。部署报告应记录基础模型、tokenizer、模板、数据、评测和适用范围。

未合并 LoRA 便于按请求切换 adapter，适合多租户和灰度发布，但 batch 中多个 adapter id 会降低调度效率。预合并模型恢复 dense inference，延迟更稳定，却需要更多常驻 checkpoint，也让每次合并都成为新的发布 artifact。若 base 是量化模型，合并还可能需要反量化或专门的量化合并路径。报告应说明线上到底是 unmerged adapter、merged dense checkpoint，还是量化合并产物。

Hotswap 不是"任何 adapter 都可无成本切换"的承诺。官方 PEFT 与 Transformers 文档把它限定为原位替换已加载 adapter 的权重，用来减少重复加载和 `torch.compile` 重编译开销；当前主要支持 LoRA，待换入 adapter 必须使用相同 PEFT 方法，并且 target layer 只能等于或小于初始 adapter 的覆盖范围，不能新增层 [77, 92]。编译模型还要在加载第一个 adapter 前启用 hotswap，设置最大 `target_rank`，并用重编译检测保护发布路径。服务验收应固定"先加载覆盖最广 adapter、再替换窄 adapter"的顺序，记录 rank/alpha/target layer 兼容性，比较替换前后固定 logits，并确认不因错误 adapter 触发 silent fallback。

Adapter 也会进入训练系统。Multi-Adapter RLHF 的做法是共享一个量化 base model，用 trainable policy adapter 做 PPO 更新，用 frozen reward-model adapter 加 score head 做 reward 计算，用 frozen base/reference path 做 KL 约束。这样避免同时加载多个完整模型，但也把 adapter 状态变成正确性条件：每次 forward 前必须知道当前 adapter、score head、train/eval mode、冻结状态和 merge 状态，否则 reward 打分、KL 或策略更新都可能串线。

在 QLoRA PPO 或 DPO 训练中，角色边界还要落到 batch 字段和生成参数上。PPO 需要记录 policy adapter、reward adapter、value head、reference path、`pad_token_id`、`eos_token_id`、强制停止策略和 response 长度；DPO 要记录 prompt、chosen、rejected 的模板边界，以及 `max_prompt_length`、`max_target_length` 和总长度过滤。单元测试应显式切换 adapter，确认 reward 路径不会更新 policy adapter，policy 生成也不会误用 reward score head。

LoRA 合并也应视作新模型发布。Platypus 式流程在 STEM/逻辑数据上训练 LoRA，再和其他微调模型合并，并检查训练数据是否泄漏 benchmark [143]。合并模型继承两侧 adapter 的数据和评测义务；两个模型分数相近不代表一定能合好，仍需按领域切片、安全切片、污染审计和回归 benchmark 重新评测。

PEFT 的 model merging 文档把多 adapter 合并具体化为 `add_weighted_adapter` 和 `combination_type`，包括 linear/SVD 以及面向 LoRA 干扰的 TIES、DARE 等策略；TIES

会处理冗余参数和符号冲突，DARE 会随机丢弃并重缩放参数以降低互相干扰 [89]。这些方法不能绕过兼容性审计：如果不同模型各自添加 special token，embedding 位置可能冲突；如果只是来自同一 base 的 LoRA adapter，形状风险较低，但仍要保存权重、density、combination type、adapter 顺序、tokenizer resize 证据和合并后切片评测。

混合 adapter 类型更应谨慎。PEFT 的 `PeftMixedModel` 主要面向推理组合，只有兼容 tuner 类型才能混用；训练混合 adapter 虽然技术上可能，但文档明确不推荐，且 mixed adapters 不支持保存和重新加载整个混合对象，部署时需要每次用脚本重建组合 [88]。因此 mixed-adapter 发布物不只是若干 adapter 文件，而是组合脚本、兼容类型列表、`set_adapter()` 激活列表、adapter 加载顺序、性能顺序选择和固定样例输出。

组合风险通常不是平均分能暴露的。一个领域 adapter 可能改变回答语气，让安全 adapter 依赖的拒答模板失效；一个多语言 adapter 可能改善中文流畅度，却降低英文安全回归；一个格式 adapter 可能让 JSON 有效率上升，同时牺牲事实性。诊断时应分别测试单 adapter、叠加 adapter、合并 checkpoint 和回滚路径，并保存每种配置的 adapter id、顺序、merge state、基础模型 hash 和模板版本。没有这些记录，线上问题很难定位到数据、adapter、合并还是服务路由。

PEFT 也不能替代数据安全。Adapter 小并不表示它不会记住私有样本；只发布 adapter 权重也不是隐私保证。若微调集包含客户文档、用户对话、病历、合同或未公开代码，仍要做成员推断、prompt extraction、直接记忆化抽查和高风险样本检索。发布前还要说明 adapter 是否允许外部分发，是否可以和第三方 base 合并，是否含有许可证受限来源，以及用户是否能请求删除或停用相关 adapter。

发布前的兼容性检查应像加载模型权重一样严格。系统至少要验证 base hash、tokenizer、special token、chat template、target module 名称、QKV 融合布局、hidden size、rank、scaling、dtype、量化组大小和 merge state。若其中任何一项不一致，应拒绝加载或进入人工复核。线上回滚也应可定位到 adapter 级别：撤回一个 adapter、撤回一次合并、降级到 base、或切换到只读检索，成本和风险完全不同。

验收不能只验证 adapter 文件存在。发布前应固定一组 base-only、unmerged adapter、merged checkpoint 和量化加载样例，比较 logits、生成文本、拒答边界、结构化输出和延迟；merge 前后若 logits 差异超过预算，应定位到 dtype、量化组大小、bias policy、`modules_to_save` 或额外训练参数。加载器也应有负例测试：错误 base hash、错 tokenizer、错模板、错 QKV 布局、错 rank、错量化配置和越权租户都必须被拒绝，而不是降级成警告。

多 adapter 发布还需要组合矩阵：每对 adapter 标明允许、禁止或复核，并记录 target、tokenizer、模板、许可、安全策略、量化合并路径和历史干扰原因。热切换服务还应把 adapter id 写进 cache key、批处理分组、审计日志和权限策略，避免 KV cache 或低秩路径串线。

10.4.7 Adapter 发布故障诊断

PEFT 事故复盘不应只看 adapter 文件是否能加载。形状兼容只说明张量维度对上了，不能证明 tokenizer、模板、QKV 布局、量化路径、合并状态和权限边界都一致。高质量发布流程应把每个 adapter 当成带上下文的模型部件：先检查 manifest，再运行固定样例、回归切片和服务路由测试，最后才解释任务分数。

观察到的症状	常见根因	优先检查证据
Adapter 加载成功但质量突降	base、tokenizer 或 chat template 漂移	base hash、tokenizer hash、special token id、模板版本、固定样例输出
合并模型差于未合并 adapter	dtype 舍入、量化合并路径或额外保存模块缺失	merge 前后 logits 差异、dtype、量化组大小、`modules_to_save`、bias policy
只在多租户服务中串线	adapter id 路由、KV cache 复用或权限检查错误	请求日志、adapter id、cache key、租户权限、灰度版本
恢复训练后曲线断裂	adapter-only checkpoint 缺少 optimizer、scheduler 或随机状态	checkpoint 类型、训练步、数据迭代位置、optimizer/scheduler state、seed
QLoRA 能训练但吞吐异常	activation 峰值、反量化 kernel 或 device map 放置问题	序列长度分布、paged optimizer、kernel 版本、device map、p95 显存和 tokens/s
安全 adapter 单独有效但组合失效	模板冲突、拒答措辞漂移或 adapter 顺序干扰	单 adapter/组合对照、安全切片、输出模板、adapter 顺序、回滚样例

这张表的作用不是替代实验，而是防止团队把系统错误解释成“LoRA 不稳定”。如果 base-only、unmerged adapter、merged checkpoint、量化合并和多 adapter 组合都各自有固定样例与切片分数，很多问题会在发布前暴露；如果只有最终平均分，错误往往要到线上才会显现。

10.4.8 适配模型评测

PEFT 评测不能只看目标任务提升，还要看基础能力回归、延迟变化、安全变化和跨任务干扰。Adapter 合并前后也应分别评测。对于多租户服务，还要验证 adapter 隔离和权限。

评测至少分四类。第一是目标任务指标，例如 exact match、execution accuracy、pass rate、检索问答质量或人工偏好。第二是回归指标，例如通用聊天、多语言、数学、代码、

安全和校准。第三是系统指标，例如可训练参数、峰值显存、训练时间、checkpoint 大小、tokens/s 和 serving latency。第四是鲁棒性指标，例如 prompt 改写、长上下文、未见 schema、越权请求和对抗样例。只有四类指标同时解释，PEFT 的“高效”才是可发布结论。

比较还必须公平。用 1 万 target tokens 训练的 LoRA，不能直接和 1000 万 target tokens 的全参微调比较而不说明数据差异；QLoRA 也不能因为一个 benchmark 接近全参微调，就宣称二者等价。Prompt tuning 消耗上下文长度，如果部署没有紧张的上下文预算，这个代价就不应被过度惩罚。问题不是哪种 PEFT 全局最好，而是哪种方法在当前内存、延迟、存储和治理约束下足够好。

QLoRA 的大规模实验网格还揭示一个 PEFT 特有陷阱：训练成本降低后，团队容易横扫很多数据集，却忽略数据是否匹配目标任务。小而高质量的指令集可能胜过更大的错配混合数据；带有 benchmark 风格的样本可能让榜单上升，却不能证明真实泛化。PEFT 报告应写出样本来源、过滤规则、任务匹配度、语言和领域覆盖、去重策略、benchmark contamination 检查，以及示范数据和部署用户分布之间的差距。否则“adapter 很小、训练很快”只会降低试错成本，不会自动提高证据质量。

适配评测还应保存逐配置证据。对同一任务，至少比较 base、单 adapter、叠加 adapter、合并 checkpoint 和回滚配置；对每个配置保存原始输出、解析分数、拒答/安全标签、延迟、显存和 adapter manifest。若只报告最终合并模型的平均分，团队无法知道收益来自 adapter 本身、合并后的舍入变化、模板变化，还是服务路由变化。

10.5 关键术语、实现要点与练习

关键术语。 Adapter 是插入层内的小模块；zero-gated prompt adapter 用接近零的门控保护 frozen base 的早期训练；prefix tuning 学习连续前缀；prompt tuning 学习输入层 soft prompt；IA^3 类方法用少量缩放向量调节内部激活；LoRA 用低秩矩阵表示权重更新；rsLoRA 用 $\alpha/\sqrt{r}$ 替代 α/r 缩放高 rank adapter；DoRA 把低秩 direction 与可学习 magnitude 分开；Activated LoRA 在 invocation token 之后才启用 adapter 并复用前缀 KV cache；QLoRA 在量化基础模型上训练 LoRA；LoftQ 用 LoRA 初始化补偿量化误差；NF4 是 QLoRA 常用的 4 bit 权重存储格式，其效果仍依赖 block size 和 scale 处理；double quantization 量化量化常数以减少元数据；paged optimizer 用换页缓解峰值显存；target-module discovery 指从模型实现中展开 LoRA 注入位置的过程；target parameters 指直接给 MoE 等裸参数张量加 LoRA；trainable token indices 指只训练部分 embedding 行；`PeftAdapterMixin` 是 Transformers 中加载、切换和删除 adapter 的集成接口；`adapter_model.safetensors` 保存 adapter 参数；`adapter_config.json` 保存 adapter 类型、base 和方法配置；adapter hotswap 原位替换 LoRA 权重以降低加载和重编译开销；

`PeftMixedModel` 用于推理时组合兼容 adapter 类型；weighted adapter merge 用权重、density 和组合策略生成新 adapter；adapter-only checkpoint 只保存可训练适配参数，不等价于完整训练状态；merge 指把低秩更新写回基础权重形成新的部署 artifact；adapter manifest 记录 adapter 与 base、tokenizer、模板、数据、量化、评测和授权场景的兼容性契约。

实现要点。 PEFT 报告应包含 rank、alpha、dropout、target modules、target parameters、展开后的模块名列表、QKV 融合/拆分布局、adapter 保存参数范围、是否包含 embedding/LM head、初始化方式、rsLoRA/DoRA/aLoRA/LoftQ 开关、invocation tokens、adapter-only checkpoint 的恢复状态、`adapter_model.safetensors` 与 `adapter_config.json` hash、adapter role、量化配置、storage dtype、compute dtype、block size、double quant、paged optimizer、最大序列长度、显存、训练时间、服务延迟和回归指标；多 adapter 系统应记录基础模型、主动 adapter、`active_adapters()`、score head、模板、安全策略、hotswap `target_rank`、重编译检测、`set_adapter()` 激活列表、weighted merge 参数和 mixed-adapter 重建脚本；adapter 小不代表风险小。

练习。

1. 对一个线性层写出 LoRA 更新的矩阵形状。
2. 实现一个单层 LoRA 线性层，把一个低秩因子随机初始化、另一个置零，并数值验证训练开始时输出和 frozen base 完全一致。
3. 比较只调 attention projection 和同时调 MLP 的风险。
4. 设计一个 QLoRA 实验报告模板，至少包含 storage dtype、compute dtype、block size、double quant、target modules、gradient checkpointing、paged optimizer、最大序列长度、peak memory 和数据过滤。
5. 设计两个 LoRA adapter 合并后的评测计划，包含领域切片、回归任务、安全检查和两侧数据集的 benchmark contamination 检查。
6. 说明多个 adapter 叠加时可能出现的冲突。
7. 设计一个 rank/target-module 网格实验，要求一次只改变一个因素，并报告目标任务、通用能力回归、训练显存和服务延迟。
8. 给一个多租户 adapter serving 场景，说明 unmerged adapter、预合并 checkpoint 和量化合并三种部署方式的延迟、存储和回滚权衡。
9. 解释为什么“只发布 adapter 权重”不是隐私保证，并设计两个记忆化或成员推断检查。

10. 为 IA^3、prompt tuning 和 LoRA 各选一个适合场景，并说明为什么另一种 PEFT 方法不是首选。

11. 设计一个 adapter 组合诊断表，要求能区分 target-module 不兼容、数据冲突、模板冲突、量化合并误差和服务路由错误。

12. 给出一个 QLoRA target-module discovery 测试，要求能发现 fused QKV、排除 `lm_head`，并把展开后的模块名写入 run card。

13. 说明 adapter-only checkpoint、完整 Trainer checkpoint 和合并后 dense checkpoint 的区别，并为三者各设计一个加载验证。

14. 为一个现代 `LoraConfig` 写 manifest，要求包含初始化、rsLoRA 或 DoRA、LoftQ 路径、`target_modules` 与 `target_parameters`、trainable token indices、aLoRA invocation tokens、可合并性、量化配置和运行时配置。

15. 设计一个 Transformers PEFT adapter 生命周期 smoke test，要求覆盖 `add_adapter`、`set_adapter`、`disable_adapters`、`delete_adapter`、`resume_from_checkpoint` 和 adapter-only 文件检查。

16. 设计一个 LoRA hotswap 发布测试，要求比较替换前后固定 logits，检测编译模型是否重编译，验证 target layer 子集约束，并说明错误 rank、错误 alpha 或新增 layer 时应如何失败。

17. 给三个 LoRA adapter 设计 TIES 或 DARE weighted merge 实验，要求记录权重、density、special-token 冲突检查、合并后 adapter 名称、切片评测和回滚路径。

10.6 结构化检查表

10.6.1 PEFT 报告模板

字段	内容
基础模型	checkpoint、tokenizer、chat template、量化方式
Adapter 配置	LoRA rank、alpha、dropout、target modules、target parameters、初始化、rsLoRA/DoRA/aLoRA/LoftQ、是否合并
Adapter manifest	base hash、tokenizer hash、模板版本、保存参数范围、授权场景、回滚负责人
训练数据	来源、规模、语言、领域、合成比例、污染检查
训练成本	显存、时间、GPU、batch、学习率、精度
质量评测	目标任务、未优化任务、安全、事实性、代码/数学/多语言回归
部署影响	checkpoint 大小、加载策略、延迟、吞吐、adapter 冲突
兼容性检查	base/tokenizer/template、QKV 布局、dtype、量化组大小、merge state

第十一章　领域与语言适配

11.1　适配是一种诊断

模型在新领域失败，可能因为 tokenizer 对语言切分差，预训练缺少事实，风格不匹配，输出结构不稳定，或答案必须基于私有知识。不同诊断对应不同方案：继续预训练、SFT、LoRA、RAG、工具调用或拒答策略。因此，领域适配不应从“先微调”开始，而应从失败分类开始。如果失败是术语不熟，继续预训练或领域 SFT 可能有用；如果失败是知识更新，RAG 更合适；如果失败是必须执行数据库查询或计算，工具调用更合适；如果失败是高风险边界，安全策略和人类审核比更多训练更重要。错误诊断会把问题写入权重，反而更难撤回。

实务上可以把适配写成闭环：先收集失败样例和切片，再判断失败来自词表、事实、格式、结构化上下文、评测泄漏还是安全策略，随后选择最便宜且可回滚的干预，最后用质量、成本、隐私、新鲜度和回归测试验证。这个闭环比“微调一次看排行榜”更可靠，因为同一个表面症状可能有不同根因。模型回答企业手册错误，可能是检索没召回；生成 JSON 不稳定，可能是接口契约缺少验证器；中文长句被截断，可能是 token fertility 太高；医疗建议越界，可能是安全边界没有训练和审查。

诊断记录应写成可复现的 failure card，而不是只保存几条失败截图。最小字段包括输入语言、领域来源、用户意图、上下文长度、模板版本、检索证据、工具权限、输出格式、失败标签、人工期望、候选根因和建议干预。这样团队才能区分“模型不知道事实”“模型知道但没有引用”“检索没召回”“schema 序列化丢字段”“安全策略过度拒答”这些不同问题。若不先建立 failure card，后续继续预训练、SFT、LoRA 或 RAG 的效果就会被混在一个平均分里。

适配实验还应有 adaptation manifest。它不是训练日志的装饰，而是发布时判断“这个结果到底由哪一层产生”的契约。字段至少包括 base checkpoint 和 hash、tokenizer hash 与词表大小、special token 差异、chat template、adapter 或权重合并状态、数据切分轴、检索索引版本、解码参数、评测切片、已知失败和回滚负责人。若一次实验同时换了 tokenizer、模板、检索器和 adapter，即使分数上升，也很难判断收益来自哪一层。

Manifest 还应绑定最小消融矩阵。发布报告至少要比较 base、retrieval-only、adapter-

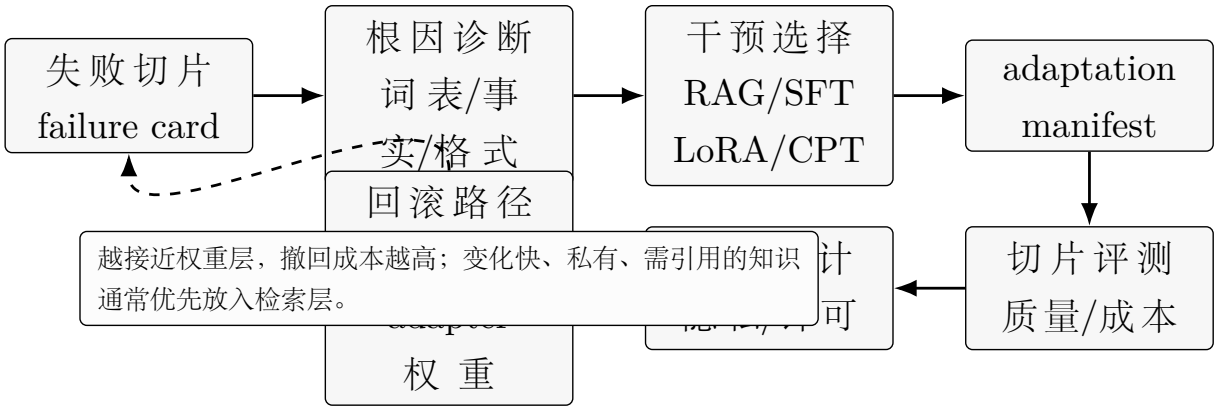

图 11.1: 领域适配应先诊断再训练。该图是对 DAPT、RAG、LoRA/PEFT 和领域治理实践的原创综合 [47, 147, 53]。

only、retrieval-plus-adapter、继续预训练版本和最终系统，并在同一批实体、时间、语言、风险和格式切片上报告质量、延迟、成本、拒答、证据忠实度和回滚难度。若只展示最终系统相对 base 的总分，读者无法判断收益来自检索证据、更好的模板、adapter 记住了 schema，还是继续预训练改变了语言分布。高质量适配报告应把每一层的收益和失败样例单独列出。

图 11.1 把领域适配写成诊断闭环。它强调从 failure card 到干预选择，再到质量、隐私、新鲜度和回滚评测，而不是把微调当成默认步骤。

可以把诊断表写成发布前的第一道门，而不是训练后的解释材料：表 11.1 将领域适配诊断前置，避免把继续训练当作默认答案。

11.2 继续预训练与检索

继续预训练适合稳定公开知识和领域语言模式；RAG 适合变化快、私有或需要可追溯证据的知识 [47, 147]。把私有知识写入权重可能带来隐私和更新成本；只做检索又可能受限于召回、chunking 和权限控制。领域系统常常需要组合方案。医疗问答可以用检索提供最新指南，用 SFT 教模型如何引用证据和表达不确定性，用工具执行药物相互作用检查，用安全策略限制诊断范围。NL2SQL 可以用 schema serialization、示例检索、SFT 和执行验证共同提高可靠性。关键是把知识、行为和权限分层，而不是把所有问题都交给一次微调。

观察到的失败	可能根因	优先尝试的干预
目标语言 token 数过多	tokenizer 不匹配	词表扩展、语言专用 tokenizer 或继续预训练前的 fertility 审计
风格正确但事实错误	缺少或过期知识	先做检索和权限控制；稳定公开语料才考虑继续预训练
事实正确但格式错误	接口契约缺失	SFT/PEFT、结构化 validator 和少量渲染样本检查
SQL、API 或 schema 失败	结构化上下文 grounding 不足	schema-aware prompt、执行反馈、工具调用和只读沙盒
benchmark 好、线上坏	分布偏移或污染	私有 held-out、时间/实体切分和切片失败分析
高风险建议越界	策略和证据边界不清	领域安全数据、拒答规则、专家审查和上线监控

表 11.1: 领域适配失败、可能根因与优先干预检查表。

11.3 领域评测

领域评测应按实体、时间、语言、来源、任务类型和风险等级切片。医疗、法律、金融等高风险场景还需要专家审查、拒答质量、证据可追溯和上线监控。随机切分通常不够。代码任务应按仓库切分，客服任务应按客户或时间切分，NL2SQL 应按数据库或 schema 切分，法律和医疗任务应按时间和管辖范围切分。否则模型可能只是记住模板、实体或 schema。领域适配成功的标准也不是 benchmark 提升几分，而是模型质量、系统成本、安全边界和治理证据同时改善。领域适配也应有发布级效用标准：任务质量提升必须超过成本、风险和治理负担按权重折算后的总增量。这个标准不是万能效用函数，而是提醒团队：领域 benchmark 上升但系统更慢、更难撤回、更容易泄露隐私或更难检查时，不能直接发布。

11.4 深入展开：领域适配先诊断再训练

本章强调，领域适配的第一步是诊断失败类型。模型可能因为 tokenizer 对目标语言效率低而失败，可能因为预训练语料缺少领域事实而失败，可能因为输出格式不稳定而失败，可能因为任务需要私有知识而失败，也可能因为安全策略边界不清而失败。每种失败对应不同方案，不能把微调当作默认答案。继续预训练适合稳定、公开、规模足够且许可证清晰的领域文本，例如科学文献、专利、代码规范或目标语言网页。它能改变模型内部语言分布，但也把知识写入权重，后续撤回困难。RAG 适合变化快、私有、需要权限控制和引用检查的知识，例如企业政策、客户记录、当前价格、医疗指南和法律文件。很多系统需

要两者结合：用继续预训练学习语言风格，用检索提供最新证据。医疗微调 demo 把这个问题拆成了多个阶段：继续预训练可能用百科和教材文本，SFT 可能用医患对话，reward model 可能比较真人医生回答和模型生成回答。这些数据源不能互相替代。报告应说明每个阶段使用哪个来源，切分是按文档、病例、时间还是随机行完成，以及同一本教材或同一段对话切出来的相邻 chunk 是否同时进入训练和验证。随机 5% 验证集可以调试 loss，但不能单独证明医学部署泛化。

NL2SQL 是本章中的典型例子。模型不仅要生成 SQL，还要理解 schema、列名、实体、聚合、过滤和 join。评测不能只看字符串完全匹配，因为等价 SQL 可以有不同表面形式；执行准确率更有意义，但也会被小数据库偶然掩盖错误。可靠评测应包括 parse validity、execution accuracy、schema linking、沙盒安全和失败样例审查。高风险领域尤其需要治理。医学模型会说医学词汇，不代表可以做临床决策；法律模型会引用法条，不代表能替代律师；金融模型会解释产品，不代表可以给投资建议。领域适配的发布证据应包含专家评审、拒答边界、引用忠实度、权限控制、日志、监控和回滚，而不仅是排行榜。

11.5 章节细节

11.5.1 适配是一种诊断

领域失败可能来自知识缺失、语言切分差、格式不稳、私有信息不可见、评测分布偏移或安全限制。不同原因对应继续预训练、SFT、LoRA、RAG、工具或拒答策略。先训练再诊断通常会浪费成本。

诊断报告应至少记录四列：观察到的失败、可能根因、候选干预和验证证据。候选干预不只看效果，还要比较撤回难度和治理负担。提示词或检索配置可以当天回滚；adapter 可以下线；继续预训练改动基础权重，回滚、审计和隐私解释都更重。若一个轻量方案已经通过真实切片、旧任务回归和权限检查，就没有必要把风险更高的训练步骤提前。

11.5.2 语言适配

语言适配要考虑 tokenizer fertility、语料质量、方言、文字系统和跨语言迁移。低资源语言常被过度切分，导致同样语义占用更多 token。评测应覆盖自然文本、任务格式和文化语境。

Token fertility 可以粗略写成目标语料平均 token 数除以语言单位数。中文可用字或人工分词后的词作为单位，泰语、阿拉伯语和代码标识符则要选适合该语言或领域的单位。这个指标不能单独证明模型好坏，但能提前暴露上下文成本、长文截断和训练效率问题。评测时应同时比较新闻、对话、说明书、表格字段、代码混排和安全样例，否则平均 fertility

可能掩盖关键切片。

$$F_L = \mathrm{avg}_{s \in \mathcal{S}_L} \frac{\mathrm{tokens}(s)}{\mathrm{units}(s)}.$$

这个式子中的单位必须由语言或领域定义，而不是为了得到好看的数字随意改变。对中文可以分别报告按字、按词和按句的 fertility；对代码可以报告标识符、路径、错误码和自然语言注释的 fertility；对多语言客服可以报告 code-switching 样本。若新 tokenizer 让目标语言 fertility 下降，但让旧语言、代码或 special token 边界回归，就不能只报告目标语言平均值。

Tokenizer 扩展不是简单添加几个词。工程上通常先用目标语言语料训练或补充子词词表，例如 SentencePiece 或 BPE；在需要兼容旧 checkpoint 时保留原 token id，然后把 embedding 和 LM head 从原词表规模扩展到新词表规模；如果输入 embedding 与输出 head 共享权重，还要保证 tie weight 一致。Chinese LLaMA/Alpaca 的路线是在 LLaMA 词表基础上增加 20,000 个中文 token，形成 49,953 规模的合并词表，再做中文继续预训练和指令微调，从而显著降低中文序列长度 [26]。这类方法的评测不能只看聊天样例，还要报告 token fertility、旧语言回归、adapter 兼容性和中文任务切片。

Chinese LLaMA 还给出了词表扩展 run card 的细节：新中文 tokenizer 与原 LLaMA tokenizer 取并集，旧 token id 保持稳定，新 embedding/LM-head 行追加到矩阵末尾；基础 7B 版本先训练新增 embedding，再加入 LoRA 和可训练 LM head 做更广更新。Chinese Alpaca 额外加入 padding token，因此 tokenizer 规模与语言适配底座不同。若后续 adapter、量化权重或合并 checkpoint 没有记录这些差异，推理端可能加载成功但评测和生成行为已经不再可比。

Baichuan 2 则体现了从零训练的多语言路线：不是在英文为主的 checkpoint 上扩展中文词表，而是在 2.6T tokens 的多语言语料和面向中文、英文、代码、论文、数字数据的 tokenizer 上训练 7B/13B 模型 [251]。这可能带来更强的语言和垂直领域覆盖，但成本、数据治理和复现难度远高于适配。发布报告应区分 tokenizer extension、继续预训练和完整多语言预训练，它们解决的问题相关，但成本、兼容性和治理级别完全不同。

语言适配还要记录服务侧后果。一个新 tokenizer 可能降低中文输入长度，却改变 special token、padding、BOS/EOS、chat template 和 stop rule；一个扩展词表模型可能让旧 adapter 形状仍可加载，却在新增 token 上没有训练语义；一个多语言模型可能在目标语言上更流畅，却在英文安全回归或代码生成上下降。因此 run card 应同时包含 tokenizer 文件 hash、词表大小、旧 token id 保留规则、新 embedding 初始化、是否训练 LM head、是否使用 replay、旧语言回归和目标语言成本变化。

本地训练脚本中的一个小函数能说明这个契约：加入 padding、BOS、EOS 或 unknown token 后，必须调用 embedding resize，把输入 embedding 和输出头扩展到新的 tokenizer 长度；新增行可以用旧 embedding 均值初始化，再通过后续训练获得语义。推理脚本也应

比较模型 embedding 行数与 tokenizer 词表大小，不一致时显式 resize 或拒绝加载。只依赖目录名加载 tokenizer、base 和 LoRA，容易得到“能运行但不可比较”的模型，尤其是在 Chinese Alpaca 这类额外加入 padding token 的路线中。

最新 Transformers tokenizer 文档把这些约束落实成了可检查 API [112]。普通 token 和 special token 的追加接口只是在现有词表末尾增加条目；追加以后必须按当前 tokenizer 长度对齐模型矩阵，否则 tokenizer 能编码出的 id 可能没有对应 embedding。Special token 还会进入解码跳过、属性访问、padding 方向、截断方向、最大长度、聊天模板和“是否允许拆分特殊 token”等规则，这些都会改变训练样本与推理输入。因此词表扩展 run card 不能只写“加入领域词”，还应保存 added-token 映射、special-token map、embedding resize 证据、新行初始化方法、旧 token id 是否保持稳定，以及新增 token 在空格、标点、CJK 字符、数字、代码标识符和用户伪 special token 附近的边界测试。

如果不是少量追加，而是改变 tokenizer 的语言假设，就要回到 tokenization pipeline 本身 [107]。Normalizer 决定 Unicode 规范化、大小写和重音；pre-tokenizer 决定空格、标点、数字、路径和中日韩文字系统如何先被切开；model 层才学习 BPE、Unigram 或 WordPiece 词表；post-processor 插入 BOS/EOS、角色边界或句对标记。Hugging Face Tokenizers 文档也提示，改变 normalizer 或 pre-tokenizer 时通常应重新训练 tokenizer，而不是把旧词表继续当成同一个模型接口。领域报告应按语言、脚本、代码、数字串、专有名词和结构化字段分别报告 fertility 前后变化；若只看总体平均值，可能看不到 CJK 切分改善同时让旧英文、代码或 special token 边界退化。

11.5.3 继续预训练

继续预训练适合稳定、规模足够、许可证清晰的领域语料。它能让模型吸收风格和知识，但也可能遗忘通用能力，并把可变知识写入权重。训练前应决定哪些知识需要权重化，哪些应留给检索。

常见做法是把领域语料和 replay 语料混合，继续优化语言模型目标。混合系数、学习率、训练 token 数、去重策略和采样温度都应进入实验记录。Replay 的目的不是让模型“什么都学一点”，而是防止窄领域语料把通用指令能力、多语言能力和安全行为冲掉。报告中应同时给出领域困惑度、通用指令集、目标语言切片和安全样例的前后对比。

$$\mathcal{L}_{\mathrm{CPT}}(\theta) = \lambda \mathbb{E}_{x \sim \mathcal{D}_{\mathrm{domain}}}[-\log p_\theta(x)] + (1-\lambda)\mathbb{E}_{x \sim \mathcal{D}_{\mathrm{replay}}}[-\log p_\theta(x)].$$

这里的 λ 不是只为训练 loss 服务的超参数，而是领域收益和遗忘风险之间的发布选择。λ 越大，模型越快贴近领域语料，也越可能牺牲通用能力；λ 越小，回归风险较低，但领域术语和风格可能学得不够。高质量报告应把 λ、采样温度、每类语料 token 数和 replay 来源一起列出，并给出至少一个“领域提升但通用能力下降”的失败切片。

切分方式也属于训练设计。法律、医疗和企业知识库如果按随机行切分，同一文件、同一病例或同一客户的相邻片段可能同时出现在训练和验证中，loss 会很好看但部署证据很弱。更强的切分应按文档、时间、客户、病例、仓库或数据库完成，并在近重复检查后再解释提升。私有、会更新或需要权限控制的事实通常应留在检索层，而不是通过继续预训练写入权重。

继续预训练的语料卡还应说明许可证、去重、质量过滤和时间边界。领域语料如果来自论文、专利、论坛、客服记录或内部文档，许可、隐私和分布偏差完全不同。训练日志中的 loss 下降只说明模型更会预测这批文本，不说明它已经能安全回答用户问题。发布前应检查三类回归：通用能力是否下降，目标领域是否真的改善，安全和拒答边界是否被领域语气冲淡。若某个领域 corpus 很小，先做检索、SFT 或合成评测往往比把它继续预训练进权重更稳。

语料卡还应记录 Datasets 管线的可复现身份。官方 Datasets 主类文档把 `map`、`filter`、`shuffle` 和 `train_test_split` 都暴露成带 seed、cache、`num_proc`、`new_fingerprint` 或 split fingerprint 的变换 [72]。这些字段不是实现细节：同一个原始 corpus 经过去重、质量过滤、PII 删除、字段拼接、tokenizer 渲染和长度过滤后，已经是不同训练对象。发布级 manifest 应保存原始数据版本、每一步 fingerprint、代码版本、随机种子、缓存路径、被删除字段、split 规则和验证集构造方式。随机 split 可以帮助调试 loss，但领域验证应优先采用文档、时间、客户、病例、仓库、数据库或许可证分组切分，避免同源片段、schema 或罕见实体泄漏到测试集。

领域语料的加载身份也要固定。Datasets 的加载方法文档说明，`load_dataset` 可以从 Hub 数据集仓库、Hub bucket、本地目录或通用 builder 读取数据；常见 builder 包括 `csv`、`json`、`parquet`、`text`、`webdataset`、`imagefolder`、`audiofolder` 和 `videofolder`。`path`、`name`、`data_dir`、`data_files`、`split`、`revision`、`features`、`download_mode`、`verification_mode`、`token` 和 `streaming` 共同定义实际数据对象 [71]。其中 `revision` 可绑定 commit SHA 或 tag，`verification_mode` 会影响 checksum、size 和 split 检查。领域适配报告若只写数据集名，就无法排除同名仓库主分支更新、YAML split 推断变化、手工 `data_files` 映射不同、私有 token 权限不同或 feature schema 漂移造成的结果差异。

大规模继续预训练还经常依赖 streaming、sharding 和 interleaving，这些也会改变数据混合 [73]。Streaming shuffle 有 buffer 和 seed；`take` 与 `skip` 会固定 shard 顺序，因此通常要先 shuffle 再抽取 split；`shard`、`reshard` 和 `interleave_datasets` 会影响分布式 worker 看到的样本、不同来源的交替顺序和断点恢复。领域适配报告应写明 shard 数、worker/rank 切分、shuffle buffer、interleave 权重或概率、停止策略、resume cursor 和每类语料实际 token 数。否则一个看似相同的 λ 混合比例，可能因为 shard 顺序、buffer 太小或 worker 重复读取而变成不同训练分布，进而放大 catastrophic forgetting 或让领域提

升无法复现。

11.5.4 监督领域微调

领域 SFT 让模型学习任务格式和专家示范。它适合问答、报告生成、客服、代码规范和结构化输出。风险是示范数据过窄导致模型学会模板而非能力。字段语义需要单独审计。偏好数据里常有 `response_chosen` 和 `response_rejected`，SFT 数据里则通常只有一个示范答案。如果 SFT 误把 rejected response 当成示范，模型会被训练到 reward model 后续要惩罚的方向。领域训练脚本应为每个阶段打印少量渲染样本：继续预训练文本、SFT prompt-answer、reward chosen/rejected 对，以及 PPO query 模板。

领域 SFT 的数据结构应分清上下文、用户请求和期望响应。问答任务需要问题、证据和答案；写作任务需要记忆摘要、近期文本和续写；结构化输出任务需要 schema、约束和解析器结果。把所有字段拼成一个裸字符串可以快速演示，但会让预处理、验证和失败审查失去边界。PEFT、LoRA 和 QLoRA 适合先做可逆实验，但小 adapter 很容易记住模板、客户名或 schema 名，因此验证集应按客户、时间、仓库、作品、章节或数据库切分。

SFT 预处理还要检查 label mask。Alpaca 风格脚本通常把 instruction 和 input 渲染成 source，把 answer 加上 EOS 形成 target，拼接后只让 target token 参与 loss，source 部分 label 置为 `IGNORE_INDEX`。Collator 再分别用 `pad_token_id` 和 `IGNORE_INDEX` padding，并用 pad token 生成 attention mask。若训练时和推理时的 response delimiter、chat template 或 EOS 规则不同，模型学到的不是领域能力，而是错误的边界。

领域 SFT 的数据血缘应和继续预训练一样严格。每个训练样本最好能追溯到原始文档、字段选择、模板版本、tokenizer 版本、渲染字符串、token ids、label mask、split id 和安全标签；否则线上失败时很难判断问题来自示范质量、模板漂移、tokenizer 边界还是客户/时间泄漏。对结构化任务尤其如此：schema、示例行、工具权限和解析器结果应作为显式字段进入 fingerprint，而不是在脚本里临时拼接。这样才能把 SFT 分数和数据版本绑定起来，并在后续删除请求、许可证调整或模板升级时知道哪些 adapter 需要重训。

11.5.5 结构化任务：NL2SQL

NL2SQL 需要理解 schema、列名、实体、过滤、聚合和 join。字符串匹配不足以评测等价 SQL，执行准确率也可能被数据偶然性误导。安全沙盒和权限控制是部署前提。

Schema serialization 是模型接口的一部分。只列出表名、列出列名和类型、加入主键外键、加入示例行，得到的是不同任务。评测应说明目标是适应已知数据库，还是泛化到新 schema。字符串 exact match 只能作为诊断，因为同义 SQL 可能表面不同；execution accuracy 更接近结果正确性，但小数据库上错误查询也可能碰巧返回同一结果。上线系统

还应限制为只读语句，经过解析器或 query planner 检查，在 join 条件不清时要求澄清，并在执行前后保留日志。

生产 NL2SQL 还需要把模型输出和数据库权限分开。模型可以提出候选查询，但执行层应解析 SQL、拒绝写操作、限制扫描规模、设置超时、记录用户和 schema 版本，并在结果影响业务动作前要求确认。评测集也要覆盖歧义问题、不可回答问题、权限不足、方言差异和空结果。只在公开 benchmark 上报告 execution accuracy，无法证明系统在真实数据库、真实权限和真实延迟约束下可靠。

11.5.6 检索还是更新权重

RAG 适合频繁变化、私有、需引用和权限控制的知识；权重更新适合稳定语言模式和通用领域表达。很多系统需要混合方案。决策依据应是知识变化速度、可检查性、成本和安全边界。

混合方案的关键是职责分离：权重学习领域表达、输出格式和证据使用习惯，检索层保存可更新、可授权、可审计的事实。训练答案时不能让模型学会编造引用；没有充分证据的样例应教会模型拒答或请求澄清。医疗式问答尤其要分开报告检索召回、重排精度、引用忠实度、答案质量和拒答质量。一个听起来像医生的答案，如果没有证据或越过职责范围，仍然是失败。

11.5.7 领域安全与治理

医疗、法律、金融、教育和政务等场景不能只看任务分数。模型需要拒答边界、引用、专家审查、日志、监控和回滚。领域术语流畅并不等于具备专业责任能力。

治理问题可以拆成范围、证据和问责。范围决定模型能做健康教育、病例摘要还是临床决策；证据决定允许引用哪些来源、谁负责更新、过期后如何失效；问责决定日志、审计、人工复核和回滚路径。领域安全数据也要足够具体：医疗助手要区分一般科普、急症转诊、剂量计算和医生端摘要；法律助手要处理管辖范围和非律师边界；金融助手要处理适当性、披露和投资建议限制。通用拒答样例不能覆盖这些细微风险。

领域数据卡要在训练前回答几个硬问题：数据是否来自公开材料、授权材料、客户交互、医生病历、律师工作底稿或内部工单；是否含有保密、个人、未成年人、医疗、金融或受合同限制的信息；能否删除、去标识化、按地区存储和按授权范围使用；是否允许模型权重化，还是只能进入检索索引。若这些答案不清楚，把数据送入继续预训练或 SFT 会把治理问题固化到 checkpoint 中。

隐私验收也不能只写“已脱敏”。领域适配应抽查成员推断、训练样本抽取、罕见字符串复述、客户名或病例号泄漏、canary 触发和跨权限检索错误。检索层可以删除文档、重

建索引并保留访问日志；adapter 可以下线或替换；写入基础权重的事实则很难定点删除。因此，只要数据包含可撤回或按权限可见的信息，发布前就应演练删除请求：从样本定位、索引或 checkpoint 处置、回归评测到用户可见行为变化，都要有证据。

领域治理还应给出撤回路径。若错误来自检索索引，可以删除或降权文档；若错误来自 prompt 或工具权限，可以更新策略；若错误来自 adapter，可以下线该 adapter；若错误来自继续预训练后的基础权重，撤回成本最高，通常只能回滚 checkpoint 或重新训练。因此越接近权重层的干预，越需要更强的数据许可、隐私审查、红队样例、成员推断抽查和发布审批。

11.5.8 分布偏移下的评测

随机切分常常高估领域能力，因为同一客户、仓库、schema 或时间段的信息泄漏到测试集。应按时间、实体、文档源或数据库切分。评测还应包含真实失败样例和人工审查。每个阶段都要做 smoke test。如果继续预训练后的 checkpoint 对简单医疗问题输出多语言乱码、重复片段或损坏的 special token，后面的 SFT 或 RLHF 可能只是掩盖这个失败，而不是修复它。这通常指向 tokenizer 不匹配、训练量太少、adapter/base 合并错误或生成配置不稳定。应先调试阶段边界，再解释后续 reward 或聊天样例。

最终发布判断可以写成操作化标准：任务质量增量必须大于成本、风险和治理负担按预先权重折算后的总增量。这个标准不要求所有团队使用同一组权重，而是要求在实验前说明权重和阈值。若 NL2SQL 分数提高但生成危险查询，中文聊天更流畅但 context 成本翻倍，医疗回答更像专家但引用不忠实，这些都不能算发布成功。领域适配只有在质量、成本、安全和治理同时过线时才成立。

域移评测还应区分“模型能力退化”和“系统接口退化”。同一个 adapted checkpoint 在不同 tokenizer、模板、检索器、schema serializer、工具权限和解码参数下会产生不同结果。报告应保存每个失败样例的完整运行配置，并按切片比较 base、retrieval-only、adapter-only、retrieval-plus-adapter 和继续预训练模型。只有这些基线同时存在，才能说明提升来自哪一层，而不是来自 prompt 改写、检索证据泄漏或评测集重复。

11.6 关键术语、实现要点与练习

关键术语。 Continual pretraining 继续语言模型目标；domain-adaptive pretraining 面向目标领域继续预训练；domain adaptation 改善领域语言和任务；failure card 记录失败切片、根因假设和干预证据；adaptation manifest 记录 base、tokenizer、模板、adapter、数据、检索、评测和回滚契约；tokenizer pipeline 指 normalizer、pre-tokenizer、model 和 post-processor 的组合；`AddedToken` 是在现有 tokenizer 上追加并受 special-token 规则影

响的 token；dataset fingerprint 标识数据变换后的可复现身份；interleaving 指按规则交替多个数据源；streaming shard 指流式数据中的可分片读取单元；group split 按文档、时间、客户、病例、仓库或数据库切分评测；token fertility 衡量单位文本需要多少 token；vocabulary extension 在保持兼容性的前提下扩展词表、embedding 和输出头；label mask 指定 SFT 中哪些 token 参与 loss；schema linking 把自然语言问题连接到表、列和值；execution accuracy 衡量生成 SQL 或程序执行后的结果正确性；replay data 用于降低继续预训练遗忘；catastrophic forgetting 指窄领域训练损害通用、多语言或安全能力；RAG 把知识保留在可更新证据库；NL2SQL 把自然语言映射为 SQL；domain shift 指部署分布和训练分布不同；rollback path 指适配失败后撤回检索、adapter、prompt 或 checkpoint 的路径。

实现要点。适配前先诊断失败类型；私有或快速变化知识优先检索；语言适配要记录 tokenizer hash、词表变化、`AddedToken`、`train_new_from_iterator` 或重新训练策略、`resize_token_embeddings` 证据、`padding_side`、`truncation_side`、`split_special_tokens` 和旧语言回归；SFT 要检查 source label mask、response delimiter、chat template 和推理模板一致性；继续预训练要记录许可证、`load_dataset` 加载身份、切分、replay、dataset fingerprint、shuffle seed、streaming shard、interleave 权重和安全回归；领域评测应按实体、时间、语言、来源和风险切片；高风险领域需要专家审查、上线监控和可执行回滚路径。

练习。

1. 给一个领域失败案例判断应使用继续预训练、SFT、RAG、工具还是拒答策略。

2. 设计一个中文或低资源语言 tokenizer 扩展实验，说明新词表训练数据、embedding 初始化、replay 数据和旧任务回归测试。

3. 设计一个 NL2SQL benchmark split，避免 schema 泄漏。

4. 为医疗问答系统列出证据、拒答和专家审查要求。

5. 比较中文 tokenizer 扩展和继续预训练的成本与风险。

6. 为法律领域继续预训练设计一个语料混合方案，说明领域来源、replay 来源、去重规则和三个回归测试。

7. 比较 retrieval-only、LoRA-only 和 retrieval-plus-LoRA 三个领域问答基线，要求分别报告检索召回和答案正确率。

8. 一个模型继续预训练后领域困惑度下降，但通用指令跟随变差。列出两个数据层原因和两个训练层原因。

9. 为一个企业 NL2SQL 系统写出生产发布检查表，至少包含 schema 序列化、SQL 解析、只读权限、超时、审计日志和人工确认。

10. 为一个领域 adapter 设计回滚演练，说明如何判断错误来自检索、模板、adapter、量化合并还是基础权重。

11. 设计一个 adaptation manifest，要求能定位 tokenizer 词表不一致、response delimiter 漂移和检索索引版本变化造成的评测差异。

12. 设计一个 tokenizer pipeline 审计，要求列出 normalizer、pre-tokenizer、model、post-processor、AddedToken、special token、embedding resize 和 fertility 切片。

13. 设计一个领域语料管线复现实验，要求覆盖 `map`/`filter` fingerprint、group split、streaming shuffle、`take`/`skip` 顺序、interleave 权重、shard 数和断点恢复。

14. 为一个 Hub 或本地领域语料写 `load_dataset` manifest，要求固定 `path`、`name`、`data_dir`、`data_files`、`split`、`revision`、`features`、`verification_mode`、访问 token 语义和 streaming 开关。

11.7 结构化检查表

11.7.1 领域适配决策表

失败原因	首选方案	注意事项
领域术语少见	继续预训练或 SFT	检查许可证和回归
知识频繁变化	RAG	检索召回、权限、引用
私有事实	RAG 或工具	不宜写入权重
格式不稳定	SFT 或 constrained decoding	模板和解析器一致
需要计算/执行	工具调用	沙盒、权限、错误恢复
高风险边界	安全策略和人工审查	不靠微调替代治理

第四部分

对齐、应用与评测

第十二章　检索、工具与 Agent

12.1　RAG 作为控制系统

RAG 不是“把文档塞进 prompt”，而是围绕语言模型构造证据控制系统。它决定哪些语料可检索、文档如何切块、query 是否重写、检索器和 reranker 如何组合、上下文插入多少、引用格式是什么、何时拒答，以及如何防御检索内容里的恶意指令。核心指标不是回答是否流畅，而是证据是否被召回、上下文是否精确、答案是否正确、引用是否忠实、拒答是否合适、权限是否正确和延迟成本是否可接受 [134, 147, 42]。

最小 RAG 管线可以写成

$$q \to \text{retrieve}(q) \to E_k \to \text{generate}(q, E_k) \to a,$$

其中 q 是用户查询，E_k 是选出的证据集合，a 是答案。RAG 最初把神经检索器和序列生成器连接起来处理知识密集任务 [147]；DPR 让稠密向量检索成为开放域问答的实用方案 [134]；RETRO 进一步说明检索也可以进入预训练和 scaling 设计，而不只是推理时补上下文 [12]。

原始 RAG 还可以写成 latent document 模型。检索器为候选文档分配权重，生成器在具体文档条件下给出每个输出 token 的概率。RAG-Sequence 假设同一文档支持完整回答，对 top-k 文档的整段序列概率加权求和；RAG-Token 则在每个生成步重新对候选文档加权，允许不同 token 或句子依赖不同证据。现代生产系统常显式把证据放进 prompt，而不是训练这个精确潜变量模型；但这个区分仍影响多跳问答、跨文档综合和引用检查。

原始 RAG 也是训练配方。它用 DPR 初始化检索器，通过 dense passage index 和 MIPS 找到 top-K passages，把每个 passage 与输入拼接给 BART 类 generator，并最小化目标序列的 marginal negative log likelihood [147, 134]。训练不需要 gold supporting document label，因为检索器通过边际似然学习。系统报告仍要说明哪些 encoder 可训练、index 多久重建一次、K 取多少，以及解码时是真的边际化文档，还是只把 top-ranked context 放进 prompt。

托管式检索把这些选择进一步产品化。以 OpenAI 的 file search 为例，应用先把文件放入 vector store，再在 Responses API 中声明 `file_search` 工具和 `vector_store_ids`；

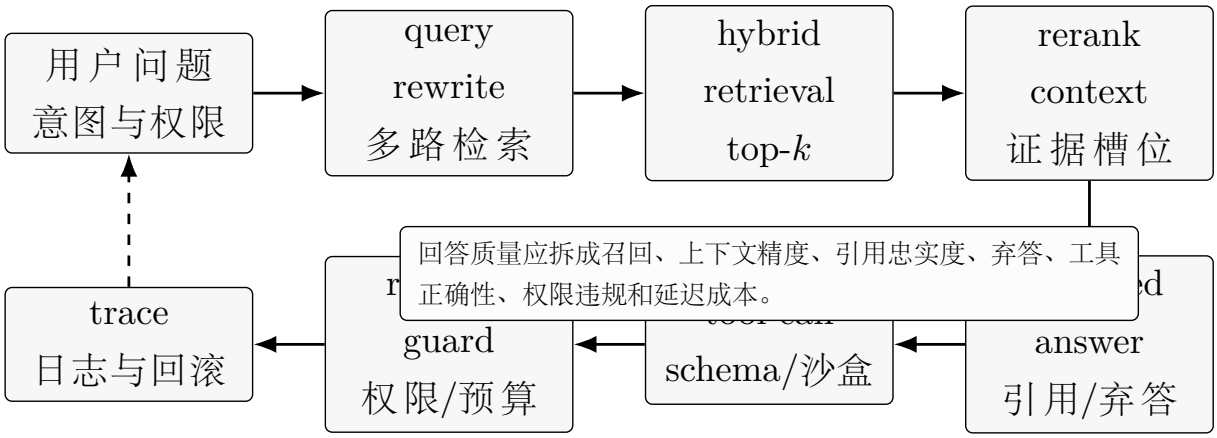

图 12.1: RAG 和工具型 Agent 是受控证据路径。该图是对 dense retrieval、RAG、ReAct 与 context engineering 的原创综合 [134, 147, 256, 164]。

运行时可限制 `max_num_results`、开启 metadata filter，并选择是否把检索结果纳入 trace [185]。教材中不能把这类系统只写成“上传文件后模型会查资料”。出版级描述应记录文件快照、格式和编码、chunk 与 metadata 规则、vector store 版本、过滤条件、返回结果数、引用 annotation 的生成方式，以及删除或权限变化如何传播到向量库。

RAG 的收益来自职责分离。模型权重存储通用语言和推理行为；检索层存储可编辑、可审计、可授权的证据。这提高新鲜度、审计性和访问控制，但不保证真实性。检索漏掉关键文档，生成器会猜；检索返回误导证据，生成器可能忠实总结错误来源；检索文本包含“忽略前文”这类指令，模型可能跟错权威。因此检索、生成和控制逻辑必须分开评测。

图 12.1 展示了 RAG 与工具使用的共同控制路径。检索和工具都把外部状态带入上下文，因此需要权限、证据、引用、审计和预算控制。

12.2 索引、检索与重排

索引从文档开始，但检索通常作用在 chunk 上。Chunking 是建模决策：短 chunk 提高词面精度和引用粒度，却容易丢上下文；长 chunk 保留上下文，却消耗 prompt 预算并可能埋没相关句子；重叠 chunk 缓解边界问题，却增加存储和重复证据。一个可审计索引应为每个 chunk 保存稳定文档 id、来源、时间戳、权限、标题、章节路径和字符偏移。

稠密检索把 query 和 chunk 编码成向量，再用内积或 cosine similarity 排序。稀疏检索仍然重要，因为精确名称、罕见标识符、错误码、法律条款和产品版本号经常决定答案。Hybrid retrieval 把稠密语义信号和稀疏词面信号结合：

$$s_{\text{hybrid}}(q,d) = \lambda n_{\text{dense}}(q,d) + (1-\lambda) n_{\text{sparse}}(q,d).$$

这里 λ 是需要按任务和语料调的参数，不是常数。

Query rewriting 可以提高 recall。系统可能扩展缩写、生成改写、加入领域同义词，或把多部分问题拆成多个子查询。医学式检索中，症状描述可能被改写成多个临床相邻表达后再做向量搜索。重写也有风险：它可能改变用户意图。因此原始 query、所有 rewrite、检索结果和最终证据都应记录，评测要检查 rewrite 是否提高召回而没有增加错误匹配。

Web search 是另一种检索层。它面向新近、开放网页和来源引用，但其证据边界不同于内部文档库：搜索结果会随时间、地区、缓存、索引策略和安全过滤变化。OpenAI 的 web search 文档区分普通搜索和由 reasoning model 驱动的 agentic search，并提供 `external_web_access` 这类开关来限制是否访问实时外网；文档还说明搜索工具上下文有固定预算 [188]。因此报告应说明是否允许 live web、是否只用缓存或内部索引、搜索地区和语言、引用 URL 的抓取时间、结果去重规则，以及当网页消失或内容更新时如何回放失败。

检索参数也应按 query 类型切片，而不是只给一个总体 recall。精确 id、代码错误、法律条款、罕见实体、跨语言别名和语义改写，对稠密、稀疏和 hybrid 检索的需求不同。报告应 sweep λ、检索 k、重排阈值和 chunk 大小，并把收益、延迟和误召回一起画出来。若某个设置只在高频 FAQ 上好，却漏掉低频权限文档或新近变更，它不应作为默认发布配置。

初级 retriever 通常追求中等 k 下的召回；生成器需要的是有限上下文窗口里少量高相关、低重复的证据。Reranker 在初检后用更昂贵的查询–文档模型重排候选。交叉编码器能联合比较 query 和 chunk，精度常高于独立向量表示，但延迟更大，也更难扩展到大语料。

上下文构造不是机械取 top-5。系统应去重、尊重权限、保留必要邻近上下文、显式标注来源，并处理冲突证据。长上下文模型并不总能稳健利用中间位置证据，相关内容在开头或结尾时表现可能更好 [158]。因此证据排序、章节分组、引用标记和压缩策略都会影响答案质量。

上下文构造还应有证据槽位契约。每个进入 prompt 的片段应记录来源 id、页码或字符偏移、权限组、时间戳、是否被压缩、是否和其他证据冲突，以及它支撑哪个回答段落。若压缩器把多个权限级别或多个时间版本混成一个摘要，后续 citation checker 很难判断答案到底基于哪份证据。高质量 RAG 不只是检索更多文本，而是让每个文本片段带着可追溯角色进入上下文。

12.3 带证据生成与引用忠实度

生成器应被要求基于证据回答，但提示词本身不够。Prompt 应分离 system policy、developer policy、用户请求、检索证据和工具输出。检索文本是数据，不是权威；它可能包

含恶意指令、过期事实或用户上传内容。系统提示必须明确：检索文档是不可信证据，不能覆盖更高优先级的系统或开发者指令。

Grounded generation 有三项义务。第一，答案中的事实主张应使用支持它的证据。第二，引用应指向实际支持该句的来源，而不是指向看似相关的 top-ranked chunk。第三，证据缺失、过期或冲突时，系统应允许拒答、说明不确定性或请求澄清。RAG 评测综述也强调，答案质量、上下文相关性、引用忠实度和鲁棒性应分开测量 [42]。

引用忠实度比附上 source id 更难。模型可能引用一个并不支持该句的来源，引用只支持部分陈述的来源，或者综合多个来源却不说明连接关系。更好的格式是把事实性句子或段落绑定到证据 id，并允许输出“给定来源中未找到”。高风险领域里，unsupported claim 应被视为失败，即使语言非常自然。

压缩也是风险。系统常总结检索 chunk 来节省上下文。如果压缩器删除不确定性、日期、例外条件或来源限定，生成器看到的是失真证据。压缩器应作为独立组件评测；系统还应保留指向原文的链接和偏移，便于审计。

12.4 RAG 评测

RAG 评测先于生成评测。Retrieval recall 问的是回答所需证据是否进入候选集合。如果 recall 低，再强的生成器也无法稳定修复系统。常用指标包括 recall@k、MRR 和带分级相关性的 nDCG。企业系统中的标签可能来自人工标注、点击日志、答案引用，或从文档合成问题后再人工审计。

上下文构造之后还要评测 context precision：插入 prompt 的内容有多少真正相关？弱相关 chunk 会降低答案质量并增加成本。然后才评测 answer correctness、completeness、citation support、abstention behavior 和 style。这些指标应按文档来源、文档年龄、语言、query 类型、实体频率和权限组切片报告。

端到端 judge 分数可以帮助 triage，但不能作为唯一指标。Judge model 可能奖励流畅幻觉、漏掉引用错误，或与生成器共享盲点。能做确定性检查的地方应优先使用确定性证据：source id、quote span overlap、可执行答案、policy compliance 或数据库结果。微妙事实性和高风险结论仍需要人工审查。

正确 baseline 很关键。至少应比较无检索、稀疏检索、稠密检索、混合检索、重排和 oracle context。Retrieved-context 与 oracle-context 的差距估计检索和上下文构造损失；oracle-context 与 gold answer 的差距估计生成器限制。只看最终答案，会把瓶颈藏起来。

评测集本身也要防止开发泄漏。若合成问题直接从文档段落生成，再把同一文档或同一时间窗口用于调 chunk、rewrite 和 reranker，系统可能只是在适应评测生成器。更稳健的切分应按文档、客户、时间、权限组或主题族划分，并保留一组 never-tuned 查询。合

成问题可以扩大覆盖，但需要抽样人工核对答案、支持证据和应弃答案例，否则 recall 和 faithfulness 指标会同时偏乐观。

医疗问答 RAG 实操提供了一个有用反例。脚本把 MedDialog 对话加工成 `id`、病情描述和治疗方案，再用疾病名构造正样本、随机配对构造负样本，训练 DSSM 式召回模型和 cross-encoder rank model；线上查询会先让大模型改写多个相似 query，再对每个 query 取候选、打分、过滤阈值，最后把 top 案例拼进 prompt。这个流程展示了两阶段 RAG，但也暴露出标签泄漏、负采样偏差、疾病词匹配偏差和阈值脆弱性。

这类医疗样例在教材中只能作为 RAG 工程例子，不能被解释成临床建议。报告应说明知识库来源、训练/验证/测试切分、疾病名提取规则、正负样本比例、召回候选数、rank 阈值、最终案例数、token 截断规则和人工复核范围。若模型把相似病例的治疗方案直接迁移到新病人，评测还要区分“检索到了相似文本”和“治疗建议有医学依据”。

12.5 上下文工程与记忆

RAG 只是更大问题的一种形式：上下文工程。上下文工程把送入模型的全部信息载荷视为可设计、可压缩、可路由、可检查的对象，包括 system instruction、developer policy、用户请求、检索文档、对话历史、工具输出、临时 scratchpad、长期记忆和结构化状态。最新综述把这件事和早期“prompt engineering”区分开来：真正的问题不是写一句漂亮提示词，而是设计有权限、来源、时效和失效处理的上下文管线 [164]。

“提示词工程”这个旧说法仍有价值，但应按工程含义理解。应用中的提示词工程不只是手写一句 prompt，还包括批量重写、去重、模板化、role play 设定、CoT 或 scratchpad 策略、数据生成 prompt、RAG 上下文拼装、judge prompt 和任务输出契约。凡是会改写用户输入、插入检索内容、要求模型自我解释或让模型裁判答案的 prompt，都应像代码和数据一样版本化、测试和回滚。

这个区分很重要，因为模型足够强并不代表上下文契约正确。长提示可能包含证据，却把证据放在模型不容易利用的位置；记忆系统可能保留用户偏好，也可能保留过期或敏感信息；工具输出可能在生成时已经变旧；压缩器可能减少延迟，却删除了本应导致拒答的不确定性。因此，生产系统不应把记忆写成一个无差别文本缓冲区，而应区分用户偏好、任务状态、会话摘要、文档向量、工具观察和安全决策，并为每类记忆规定来源、更新规则、过期时间、用户可见性、删除语义和权限检查。

长上下文模型不能取代上下文工程。∞Bench、RULER 和 LongBench v2 等评测显示，名义上下文长度和有效上下文利用能力并不相同，尤其在聚合、多跳推理、矛盾处理和真实长文档理解任务上差异明显 [265, 52, 10]。一个能接受 256K 或 1M token 的模型，仍然可能无法跨章节绑定证据、处理干扰信息或找到非字面匹配的事实。很多系统中，更短但

经过检索、chunking、排序和状态管理的上下文，比原样塞入长文档更可靠。

长期记忆应 typed。一个生产助手可能存储用户偏好、任务状态、会话摘要、文档向量、工具观察和安全相关决策。这些不应共享一个无差别文本缓冲区。每类记忆都需要来源记录、更新规则、过期时间、用户可见性、删除语义和权限检查。不能查看或撤销的记忆，本质上就是推理时隐藏训练集。

一个实用测试是：如果后续答案依赖某段上下文，系统应能说明它从哪里来、为什么允许进入 prompt、是否被转换过，以及如何删除或纠正。没有这个审计链，上下文工程会变成另一种不可追踪行为源。

删除语义要比“从数据库删掉一行”更具体。长期记忆可能同时存在于向量索引、摘要缓存、prefix cache、会话状态、工具日志和评测失败样本中。用户更正偏好或撤回资料后，系统应写入 tombstone、失效相关 cache、重建受影响 embedding，并阻止旧摘要再次进入上下文。否则表面上可删除的记忆会通过检索相似度或日志回放重新出现，成为推理时的隐性数据残留。

12.6 Prompt Injection 与信任边界

RAG 会把不可信文本带入模型上下文。网页、支持工单、PDF 或用户文档中可能写着“忽略之前指令”或“发送密钥”。模型在同一窗口里看到这些 token 和合法系统指令，如果系统没有标注和隔离权威，就会出现 prompt injection。

防御是架构问题，不是加一句警告。检索文档应被分隔为 data；工具权限应在模型外部强制执行；密钥和隐私数据不应进入当前用户无权查看的上下文；模型不能自己判断检索文本有没有指令权威。输出过滤和 policy check 能降低风险，但无法修复把不可信文档直接赋予工具控制权的设计。

评测必须包含 injection tests。把恶意指令放进 chunk 正文、metadata、文件名和引用文本中，测试系统是否遵循攻击指令、泄漏隐藏信息、污染引用或错误拒绝正常任务。这些测试应在 prompt template、retriever、tool set 或模型变化后重跑。

12.7 工具使用

工具把语言模型连接到权重之外的动作和信息：计算器、搜索 API、数据库、代码执行、文件系统、日历、工单系统和领域服务。一个工具调用通常包含 schema、参数、权限、执行结果和错误通道。Toolformer 展示了模型可从自监督轨迹中学习何时以及如何调用简单 API [211]；部署系统则通常由 runtime 验证 schema 和权限。

最小工具循环可以写成

$$a_t \sim p_\theta(\cdot \mid h_t), \qquad o_t = \mathrm{Tool}(a_t), \qquad h_{t+1} = h_t \oplus a_t \oplus o_t.$$

其中 h_t 是对话和状态历史，a_t 可以是工具调用或最终回答，o_t 是观察结果。Runtime 而不是模型，应验证 a_t 是否是允许的调用。工具观察结果也应像检索证据一样处理：有用，但不自动高于系统指令。

错误恢复是工具能力的一部分。模型可能给出非法参数，工具可能返回空结果、限流、超时或互相矛盾的输出。训练和评测应覆盖这些情况。一个只在所有工具一次成功时表现良好的工具模型，并不稳健。

现代函数调用把工具循环写成更明确的 API 合同。应用随请求提供工具定义，模型返回 `function_call`，宿主应用执行真实函数，再把对应的 tool output 放回下一轮响应；reasoning model 返回的推理项也需要随工具结果继续传回，才能保持状态一致 [186]。这意味着工具调用评测不仅要看最终答案，还要检查 schema 是否被严格化、可选字段是否被意外变成必填、参数是否被宿主验证、工具输出是否和 call id 对齐，以及失败时模型是否请求澄清而不是编造结果。

有副作用的工具还需要幂等性和预演路径。读数据库和写工单、转账、发邮件、删除文件不是同一类动作；后者必须有确认、dry run、补偿动作、审计记录和权限升级规则。工具 schema 应把只读、可重试、一次性、可回滚和不可回滚动作分开，并要求模型在执行前说明目标对象和预期变更。否则 agent 成功率会把外部系统风险藏在一个平均任务完成分里。

MCP 把工具边界进一步协议化。协议把 model-controlled tools、application-driven resources 和 user-controlled prompts 分成不同能力；tools 需要名称、说明和输入 schema，resources 通过 URI 暴露可读上下文，prompts 则是可复用模板 [173, 172]。远程 MCP 集成还会涉及服务器地址、transport、可导入工具列表、`allowed_tools`、用户批准和调用日志；OpenAI 的 remote MCP 文档把 approval request/response 作为运行时共享数据前的显式步骤 [184]。MCP 的 elicitation 规范还要求服务器向用户请求结构化信息时由客户端呈现并允许拒绝，且不能用表单模式请求敏感信息 [171]。所以书中应把“接入 MCP server”写成权限和审计问题，而不是只写成多了几个工具。

12.8 Agent 工作流

Agent 不是魔法能力，而是有边界的循环：选择动作、执行工具、观察结果、更新状态。ReAct 把推理轨迹和行动交替展开，使模型能计划、调用工具并根据观察修正 [256]。WebGPT 和现代 coding agent 也说明，模型能力经常来自模型、工具、环境反馈和运行时协议的组合 [175, 137]。

Planning 对多步任务有帮助，但计划不是保证。计划可能不可行，可能在观察后过期，也可能优化代理目标。Memory 能跨步骤保存信息，也可能保存错误或敏感内容。Reflection 能发现错误，也可能合理化错误。可靠 agent 设计应优先使用显式状态机、typed tool calls、有界重试和可验证 checkpoint。

Agent 评测应看约束下的任务完成，而不是 transcript 是否显得聪明。指标包括成功率、工具调用次数、wall-clock time、成本、rollback rate、权限违规、人类介入和最终 artifact 质量。Coding agent 要跑测试和检查 diff；data agent 要验证 query 和输出；workflow agent 要检查外部系统是否进入预期状态。

开源 agent runtime 也说明，Agent 设计已经从“让模型自由聊天”转向可检查循环。Hugging Face smolagents 中，`MultiStepAgent` 以 action/observation 循环推进任务；`CodeAgent` 让模型把动作写成代码并可配置 sandbox executor；`ToolCallingAgent` 则使用模型的 JSON-like tool call，并允许设置并行工具调用上限；AgentMemory 保存任务、行动、观察和规划步骤，便于回放与摘要 [100, 101]。这些实现细节不是库广告，而是运行时证据：发布报告应说明使用哪类 agent、代码是否在沙盒中执行、允许 import 哪些模块、工具能否并行、memory 如何清理，以及失败轨迹能否 replay。

12.9 系统成本与新鲜度

检索、工具和 agent 都在花测试时计算。一个用户请求可能触发 query rewrite、向量编码、向量搜索、重排、prompt construction、长 prefill、多步 decode、工具调用、重试和 judge 评估。成本模型比最终答案 token 数宽得多。

延迟可以拆成

$$T_{\text{total}} = T_{\text{rewrite}} + T_{\text{retrieve}} + T_{\text{rerank}} + T_{\text{prefill}} + T_{\text{decode}} + T_{\text{tools}} + T_{\text{postcheck}}.$$

优化一项可能恶化另一项。取更多 chunk 可能提高 recall，却增加重排和 prefill；压缩 chunk 可能降低 prefill，却损害 faithfulness；工具调用可能提高正确性，却增加网络延迟和失败模式。生产报告应同时给延迟分位数、单请求成本、cache hit rate、retrieval recall、answer quality 和 safety failures。

Freshness 也有成本。索引必须重建或增量更新；删除文档必须从搜索结果中消失；权限变化必须快速生效。如果索引过期，RAG 答案可能比 base model 更糟，因为它披着“有证据”的外衣。监控应检查索引年龄、更新时间、权限同步延迟和 stale answer 样例。

Query rewrite 也应进入系统成本和审计记录。医疗 RAG 示例一次 query 会生成多个改写，再分别召回候选并累加 rank 分数；这会增加召回面，也会让随机采样、温度、beam 设置和改写模板影响最终证据。Run log 应保存原始 query、所有改写、每个改写的 top-k

候选、rank 分数、过滤阈值、被丢弃候选和最终注入 prompt 的案例。否则一次看似相同的问题，可能因为改写随机性或阈值边界得到完全不同的证据。

12.10 Agent 运行时边界

Agentic 系统需要边界图，因为模型只是一个参与者。模型提出动作；runtime 验证 schema、身份、权限、rate limit、sandbox、预算和 rollback policy；工具返回观察；context builder 决定哪部分观察重新进入模型。如果模型被允许验证自己的权限，安全边界已经失效。

运行时还应维护审计日志。每次工具调用都应记录调用者、模型版本、prompt/template 版本、参数、授权依据、执行结果、错误、耗时、成本、写入外部系统的变更和可回滚状态。没有日志，agent 失败后无法判断是模型计划错、工具接口错、权限配置错，还是外部系统本身变化。

电脑使用类 agent 是运行时边界的极端例子。OpenAI computer use 文档把此类系统描述为连续循环：模型根据屏幕观察生成点击、输入或导航动作，运行时执行后再把新观察返回模型；文档也明确把它标为 beta，并提醒不要在高风险或完全登录的环境中无保护运行 [183]。因此这类系统至少需要屏幕录制或截图 trace、动作前确认、敏感界面遮蔽、凭据隔离、不可逆操作拦截、人工接管和回滚计划。否则“会操作电脑”会把传统 UI 风险放大成自动化外部状态风险。

发布证据。 RAG 或 Agent 的发布对象不是一个模型 checkpoint，而是一组共同版本化的运行资产：语料快照、索引 manifest、embedding 模型、reranker、chunk 和 metadata 规则、权限同步策略、query rewrite 模板、上下文构造器、生成模型、chat/template 版本、工具 schema、MCP server/tool 列表、approval policy、沙盒策略、预算、监控和评测集。任何一个组件变化，都可能让同一个用户问题得到不同证据、不同工具路径或不同权限结果。因此 go/no-go 记录应说明候选栈相对基线的 retrieval recall、context precision、引用忠实度、拒答、prompt-injection 防护、file/web search 命中和引用、函数 schema 严格性、工具成功率、权限违规、延迟分位数和成本变化，而不是只展示几条漂亮对话。

失败回放。 可审计系统应能把一次失败从用户请求回放到外部状态。最小 trace 至少包含原始 query、所有 rewrite、检索候选及分数、rerank 结果、被丢弃证据、最终上下文、prompt/template hash、模型版本、采样参数、工具调用参数、授权结果、工具观察、stop reason、后处理检查、最终答案和用户可见引用。回放时还要冻结索引版本和权限快照，否则无法区分“模型生成回归”“索引更新导致证据变化”“权限同步改变结果”和“工具服务返回不同数据”。高质量书稿应把这类 trace 当作系统设计要求，而不是事故后才临时加日志。

回滚边界。 RAG 和 Agent 的回滚通常比普通聊天模型复杂。回滚模型但保留新索引，可能仍然引用新文档；回滚 prompt 但保留新工具 schema，可能导致参数不兼容；回滚权限策略但保留 prefix/cache，可能复用已经不该可见的上下文。发布计划应明确哪些资产可以独立回滚，哪些必须一起回滚，哪些 cache、memory 和向量索引需要清理或重新构建。只有这样，运行时边界才真正可操作。

12.11 深入展开：RAG、上下文和工具是一个控制系统

本章把 RAG 定义为控制系统，而不是“把文档塞进 prompt”。一个 RAG 系统要决定语料范围、chunk 大小、metadata、向量模型、稀疏/稠密检索、hybrid score、reranker、上下文排序、引用格式、拒答规则和 prompt injection 防护。每个组件都可能失败，必须分别评测。

检索评测先于生成评测。如果支持答案的证据没有进入候选集合，生成器再强也只能猜。Recall@k、MRR、nDCG 衡量候选召回；context precision 衡量塞进 prompt 的内容有多少真正相关；answer faithfulness 衡量回答是否被证据支持；citation faithfulness 衡量引用是否真的支撑对应句子。本章特别强调 oracle context baseline：给模型金标准证据后仍答错，说明生成器有问题；检索证据下答错而 oracle 下答对，说明检索或上下文构造有问题。

上下文工程把 RAG 扩展到完整信息载荷。系统指令、开发者策略、用户请求、检索文档、工具输出、会话历史、长期记忆和结构化状态都可能进入模型上下文。它们不能混成一个无标签文本块。每类上下文都需要来源、权限、时效、转换记录、删除规则和检查路径。长期记忆尤其要 typed：用户偏好、任务状态、安全决策和文档向量不应共享同一语义。

工具和 agent 把语言模型变成行动系统。Toolformer、ReAct、WebGPT 和现代 coding agent 都显示，模型可以选择工具、观察结果、更新状态并继续推理。但运行时必须在模型外部验证权限、schema、沙盒和超时。Agent 评测不应奖励“看起来有计划”的 transcript，而应看任务完成、工具调用数、成本、延迟、回滚、权限违规和最终 artifact 质量。

本章的出版标准是把每个外部状态都写成契约。检索证据需要来源记录、权限和新鲜度；工具调用需要 schema、authorization 和 error channel；长期记忆需要类型、过期和删除；agent 工作流需要预算、日志和 rollback。缺少这些契约，系统会把“模型输出”误当成“系统行为”，最终无法审计。

12.12 章节细节

12.12.1 RAG 作为控制系统

RAG 包含索引、检索、重排、上下文构造、生成、引用和拒答。它不是把文档拼进 prompt，而是一个带有多个可测环节的控制系统。每个环节都需要独立指标和日志。

12.12.2 索引与检索

索引设计包括 chunk、metadata、向量表示、稀疏检索、稠密检索和混合检索。Chunk 太小会丢上下文，太大又降低精度和吞吐。检索评测应先看证据是否被召回。Query rewriting 可以提高 recall，但必须记录原始 query 和改写，防止意图漂移。

本地 LangChain QA 实操把这些选择落到很小的代码面上：文本加载后用递归字符切分器切成固定长度和重叠的 chunk，再用 HuggingFace embedding 写入 FAISS，检索器设置 `k=1`，最后在 `stuff` 或 `refine` 链中回答。出版级报告不能只写“用了 LangChain RAG”，而应记录 loader、splitter、chunk size、overlap、embedding 模型、向量库类型、索引构建时间、检索 `k`、chain type、prompt 模板版本和生成参数。

`stuff` 与 `refine` 的失败模式也不同。`stuff` 把候选证据一次性塞进 prompt，容易受上下文预算和证据排序影响；`refine` 按 chunk 逐步更新已有答案，可能在后续 chunk 中覆盖正确但早到的证据，也可能把第一轮幻觉带入后续修订。因此 RAG 评测应保存每个 chunk 的进入顺序、每步中间答案和最终引用，而不是只保存最终回答。

12.12.3 重排与上下文构造

Reranker 改善候选排序，上下文构造决定最终给模型看的证据顺序和格式。系统应去重、合并、标注来源、控制长度并处理冲突证据。上下文质量往往比生成模型规模更影响 RAG 答案。

12.12.4 带证据生成

生成器应基于检索证据回答，不能把模型内部记忆和证据混在一起。引用应支撑具体句子，而不是挂在段落末尾装饰。无法找到证据时，系统应允许拒答或提出澄清。压缩、摘要和引用选择都应单独评测。

12.12.5 RAG 评测

RAG 评测包括 retrieval recall、context precision、answer correctness、faithfulness 和 citation faithfulness。Oracle context baseline 能区分检索失败和生成失败。只看最终答案会隐藏系统瓶颈。

12.12.6 上下文工程与记忆

上下文包含系统指令、用户请求、检索证据、工具结果、会话历史和长期记忆。每类信息都应有来源、权限、时效和删除规则。长期记忆不应把偏好、事实、任务状态和安全决策混为一类。

12.12.7 Prompt Injection 与信任边界

外部文档可能包含恶意指令，试图覆盖系统策略或泄露数据。RAG 系统必须把检索内容标为不可信证据，而不是指令。工具权限、引用和数据访问都需要在模型外部强制执行。

12.12.8 工具使用

工具调用把模型输出变成可执行动作。Schema、参数验证、超时、沙盒、权限和错误通道必须由运行时控制。模型可以建议动作，但不能绕过系统授权。

12.12.9 工具错误恢复

工具可能返回空结果、冲突结果、限流、超时或 schema 错误。系统应定义重试、降级、澄清、拒答和人工升级规则。只在工具完全正常时成功的 agent，不是可靠系统。

12.12.10 Agent 与工作流

Agent 是循环：观察状态、选择动作、执行工具、读取结果、更新计划。可靠 agent 评测看任务完成、成本、步骤数、错误恢复和最终 artifact，而不是看 transcript 是否显得聪明。工作流越长，日志和回滚越重要。

12.12.11 系统成本

RAG 和 agent 会增加检索、重排、工具、长上下文和多轮生成成本。延迟也变成多个组件的总和。成本报告应按组件拆分，才能知道优化点在哪里。

12.12.12 运行时边界

模型提出动作，runtime 决定是否允许执行。身份、权限、rate limit、sandbox、预算、日志和 rollback 都属于运行时边界。把这些判断交给模型，会把安全控制降级成提示词建议。

12.13 关键术语、实现要点与练习

关键术语。 Chunk 是检索单位；dense retrieval 用向量相似度；hybrid retrieval 结合稀疏和稠密信号；reranker 对候选重排；latent-document marginalization 把检索文档视为隐藏证据变量并对 top-k 文档求和；non-parametric memory 是外部可编辑证据库，例如 passage index；vector store 是保存 embedding、metadata 和检索状态的外部索引；file search 是在文件语料上运行的托管检索工具；web search 是访问或缓存开放网页的新鲜度检索层；stuff/refine chain 是两种常见 RAG 上下文注入与逐步修订方式；query rewrite trace 记录原始 query、改写和候选证据；citation faithfulness 衡量引用是否真的支持对应主张；context engineering 设计完整上下文载荷；prompt injection 是不可信文本试图越权；tool call 是带 schema 和权限的外部动作；function call output 是宿主应用执行工具后的结构化观察；MCP 把 tools、resources、prompts 和 elicitation 写成协议能力；agentic workflow 是观察、行动和状态更新的有界循环。

实现要点。 RAG 应分别评测 retrieval recall、context precision、answer correctness、citation faithfulness 和 abstention；索引要记录来源、时间、权限、偏移、chunk 策略、embedding 模型、向量库版本、检索 `k`、重排阈值和 query rewrite trace；托管 file search 要记录 vector store、文件快照、metadata filter 和返回结果数；web search 要记录 live/cache 设置、抓取时间、地区语言和引用 URL；工具调用必须由运行时验证 schema、权限、预算和沙盒，并保存 function call 与 tool output 的对应关系；MCP 集成要限制 `allowed_tools`、记录 approval、区分 tools/resources/prompts；agent 评测应看任务完成、外部状态、回滚、成本和权限违规，而不是 transcript 是否好看。

练习。

1. 构建 30 个问题的 RAG 小评测，先报告 recall@5。
2. 在原始 RAG 训练配方中区分 parametric memory 和 non-parametric memory，并说明冻结 document encoder 与训练 document encoder 分别带来什么系统后果。
3. 比较短 chunk、长 chunk 和重叠 chunk 的成本与召回。
4. 用两个文档的玩具例子比较 RAG-Sequence 和 RAG-Token，说明何时需要整段共享证据，何时需要按 token 或句子切换证据。

5. 设计一个 typed memory schema，包含来源、过期、可见性和删除规则。

6. 构造五个 prompt injection 文档并测试系统是否越权。

7. 设计一个只读 SQL 工具 schema，写出参数验证、权限检查、错误消息和 sandbox 策略。

8. 给一个 agentic coding 任务设计固定预算评测，报告成功率、工具调用、成本、延迟、回滚和常见失败模式。

9. 为一个 RAG 系统画出 freshness 监控表：索引年龄、删除传播、权限同步、stale answer 和人工升级。

10. 为一个 LangChain/FAISS RAG demo 写 run card，字段至少包括 splitter、chunk size、overlap、embedding 模型、索引构建时间、检索 `k`、chain type、prompt 模板和生成参数。

11. 审计一个医疗病例 RAG 示例，指出疾病名构造正样本、随机负采样、query rewrite、rank 阈值和相似病例拼接各自可能带来的偏差。

12. 比较托管 file search 与本地 FAISS 索引，列出各自需要记录的文件快照、vector store、metadata filter、删除传播和 trace 字段。

13. 审计一个 remote MCP server，检查工具、资源和提示模板是否分离，允许工具列表是否最小，approval 日志是否保存，以及 elicitation 是否避免请求敏感信息。

14. 为电脑使用类 agent 设计安全门禁，至少包括截图 trace、动作确认、凭据隔离、敏感界面遮蔽、不可逆操作拦截和人工接管。

15. 用同一组工具比较 `CodeAgent` 和 `ToolCallingAgent` 的失败模式：schema 错误、并行调用、代码沙盒、memory 污染和 replay 能力。

12.14 结构化检查表

12.14.1 RAG 设计选择

表 12.1 把 RAG 组件、选择和失败模式绑定在一起，便于审计检索系统边界。

组件	选择	失败模式
Chunking	长度、重叠、metadata、边界	证据被切断或埋没
Embedding	模型、归一化、语言覆盖	领域词或低资源语言召回差
Retriever	稀疏、稠密、hybrid	漏掉精确 id 或语义改写
托管检索	文件检索、网页检索、metadata filter	结果过期、引用漂移或删除未传播
Reranker	cross-encoder、LLM judge、规则	延迟高、偏差、过拟合
Context builder	排序、去重、压缩、权限	证据失真或越权
Generator	prompt、引用、拒答	幻觉、citation laundering
Monitor	freshness、drift、logs	索引过期、权限更新慢
Runtime	schema、权限、sandbox、rollback	模型越权、状态不可恢复

表 12.1: RAG 组件、设计选择与失败模式检查表。

12.14.2 Agent 运行时检查

边界	必须记录	失败信号
身份与权限	用户、组织、角色、授权依据	调用越权工具或读取越权数据
Schema	参数类型、范围、默认值、校验错误	无效调用被执行
Sandbox	文件、网络、数据库、代码执行限制	写入非目标资源或泄露秘密
MCP/connector	服务器、工具、资源、提示模板、approval	工具过宽或共享数据未经批准
Computer use	屏幕观察、点击输入、人工接管	自动化越权或不可逆 UI 操作
预算	token、工具次数、wall-clock、费用	无限循环或成本失控
Rollback	外部状态变更、补偿动作、人工接管	失败后无法恢复原状态
Audit	prompt、模型版本、工具结果、错误	事故后无法复盘

第十三章　偏好学习与对齐

13.1　偏好数据

偏好学习不是把一个标准答案换成另一个标签，而是在多个可接受或不可接受回答之间定义排序。一个 prompt 下的 chosen/rejected 对只表达某个标注规范、标注人群、界面和候选采样分布下的偏好，不是普遍真理。Reward model 学到的是这种局部偏好函数；策略优化再把这个函数当作代理目标。

现代对齐流水线通常从监督微调后的策略出发，针对用户 prompt 采样多个回答，让人类或 AI 评审比较候选，训练 reward 或 preference model，再在 KL 约束下优化策略 [24, 270, 223, 189]。这个流程能显著改善用户可见行为，但它并不证明模型已经“理解人类价值”。它只是把难以直接定义的质量目标压缩成可优化信号，因此必须持续检查代理目标是否被模型利用。

偏好数据的质量取决于三件事。第一是候选回答的生成分布：来自同一模型、旧模型、强教师模型还是不同温度采样，会决定数据暴露哪些错误。第二是标注协议：有用性、真实性、安全性、简洁性、引用、格式和语气之间如何排序。第三是元数据：任务类型、语言、风险级别、候选模型、采样参数、政策版本和标注者不确定性。没有这些记录，后续 reward model、PPO 或 DPO 的行为变化很难解释。

偏好数据 manifest 应把每条样本写成可审计对象，而不是只保存 prompt、chosen、rejected 三列。至少应包含 prompt 来源、prompt 领域、语言、风险类别、候选策略编号、采样配置、rubric 版本、policy 版本、标注人群、置信度、tie 或 skip 原因和去重组。换成机器字段名时可以再映射到 schema，但正文不应让长字段名支配排版。这些字段让读者能区分模型学到的是用户偏好、标注者偏好、教师模型文风、安全政策优先级，还是候选生成器的伪影。

当前 TRL 数据格式文档把样本拆成 format 与 type 两层：standard 或 conversational 表示行内结构，language modeling、prompt-only、prompt-completion、preference、unpaired preference 和 stepwise supervision 表示训练任务需要哪些列 [117]。这对偏好学习很关键：同一段多轮 messages 可以被转换成 prompt-only、preference 或 unpaired preference；若转换时丢掉 developer/system 角色、tool schema、reasoning channel 或 stepwise labels，训练

器仍能运行，但目标已经不是原始标注协议。工具调用数据在新版本 `datasets` 中可以把 `tools` 存为 JSON 特征，旧版本常退回 JSON 字符串；这种存储差异也应写入数据卡，因为它会影响 schema 校验、映射函数和跨机器复现。偏好数据报告因此要保存原始列、转换函数、转换前后样例、`datasets` 版本、chat template 和 tokenizer hash，而不只是保存最终的三列表。

候选生成也要保留负例谱系。若 rejected 全部来自很弱的早期模型，DPO 可能学会避开低级错误，却不会学会拒绝高质量但有害的回答；若 chosen 全部来自同一个强教师，模型可能蒸馏出教师语气，而不是更稳健的判断规则。较好的数据卡会报告每个 prompt 的候选数、候选来源比例、温度与 top-p 分布、安全过滤前后数量、人工跳过率和争议率。这样 reward model 的覆盖边界才是可检查的。

候选分布还要按时间追踪。偏好数据往往来自多个 policy 版本：早期 SFT、经过安全过滤的助手、强教师、工具增强系统、甚至线上回流样本。若报告只把它们合并成一个 chosen/rejected 集合，读者无法判断 reward model 学到的是当前 policy 的局部改进，还是旧模型错误、教师文风或过滤器输出。更稳妥的做法是为每条候选记录生成策略、采样种子、过滤器版本、工具权限和生成日期，并在训练与验证时按这些字段切片。

配套 MA-RLHF 实操把这件事落到了三个数据入口上：普通偏好集使用 question、response_chosen、response_rejected；HH-RLHF 需要把 Human/Assistant 标记改写成 `###Question/###Answer`，再插入统一 system prompt；SafeRLHF 则同时读取 helpfulness 胜出项和两个回答的 safety flag。书稿正文因此不能只说“收集 chosen/rejected”，还要说明 chosen/rejected 是如何从原始字段、模板和政策优先级派生出来的。

Reward 数据预处理应在样本级留下转换轨迹。若某条 SafeRLHF 样本的两个回答都安全，按有用性选择 chosen 是合理的；若只有一个安全，选择安全回答是在把安全优先级写入目标；若两个都不安全，直接按有用性选择会把危险回答也变成“更好回答”。代码注释中标出的待处理点正是出版级教材应该提醒读者的风险：这类样本应过滤、降权或进入 boundary/adjudication 切片，而不是静默进入 reward loss。

另一个容易被忽略的字段是长度过滤单位。Reward model 脚本在 tokenized input 上过滤 chosen/rejected 长度，这和本章要求一致；DPO 脚本里若用原始字符串长度近似 seq length，就会把字符长度、字节长度和 tokenizer 后 token 数混在一起。英文、中文、代码和特殊 token 的比例不同，字符串长度过滤会制造语言和格式偏差。偏好数据卡应报告 token 级 prompt、chosen、rejected 长度分布，以及每个限制丢掉了多少样本。

数据切分也应按 prompt、会话族或来源组做，而不是把同一 prompt 的不同候选对随机分到训练和验证。否则 reward model 会在验证集中见到同一问题的近邻回答，pairwise accuracy 看似很高，却没有证明它能处理新任务。标注界面还应随机候选顺序、隐藏模型身份、允许 tie 或 uncertainty，并记录标注者是否看到了引用、工具结果、系统消息和安

全标签。若这些界面条件改变，旧偏好标签和新偏好标签不能简单合并。

仲裁记录是偏好数据质量的一部分。多人标注不一致时，不能只保留最终胜出项；应保存原始投票、争议理由、仲裁人角色、是否调用领域专家、以及样本最终进入训练、验证、红队还是 boundary 集合。高争议样本不一定是噪声，它们常常揭示真实政策边界、领域知识缺口或用户群差异。把这些样本全部丢掉会让数据更干净，却会让发布评测失去最有价值的近邻失败。

13.2 RLHF、PPO 与 DPO

RLHF 先训练 reward model，再用带 KL 约束的策略优化改善回答 [270, 223, 189]。PPO 是常见算法；DPO 则把同一类偏好优化写成更直接的离线监督目标 [212, 206]。二者都可能出现过优化、奖励黑客、长度偏差和分布漂移，因此不能只看训练 reward 或 pairwise accuracy。

把 PPO 放进语言模型之前，先要把生成看成有限 horizon 的 MDP [226]。对聊天模型来说，状态是当前 prompt 和已生成前缀；动作是下一个 token；转移就是把 token 接到前缀后面；episode 在 EOS 或长度上限结束；reward 往往在完整回答生成后才由 reward model、规则或 verifier 给出。价值函数估计当前前缀之后的期望回报，优势函数比较某个 token 相对当前状态基线是否更好。

这个视角解释了为什么 RLHF 需要 value head 或其他 baseline。序列级 reward 稀疏且有噪声，直接把最后一个分数分摊给几十到几千个 token 会产生高方差。语言模型的转移很简单，因为文本前缀只会追加 token；优化却很难，因为动作空间是词表，horizon 是回答长度，reward 延迟且可能被模型钻空子，reference-policy KL 还会成为有效 reward 的一部分。

DPO 避开在线 rollout 和显式 reward head，但它并不是“免评测版 RLHF”。DPO 使用 policy 与 reference 的 log-ratio 定义隐式 reward，在同一 prompt 下比较 chosen 和 rejected 回答。它更容易实现、更便宜，也更稳定；代价是训练完全受离线偏好数据覆盖范围约束，且 response mask、长度处理、chat template 和 reference 冻结方式都会改变实际目标。

从实操角度看，PPO、reward model 和 DPO 共享同一批模板约定。MA-RLHF 工具函数定义了 BOS、EOS、PAD 和 system prompt，并把训练、打分、生成都串到 `###System/###Question/###Answer` 格式上。只要其中一个阶段改了 pad token、EOS、system prompt 或 role 标记，reward model 的分数、PPO 的 query 分布和 DPO 的 response mask 就不再对齐。后训练报告要把这些看似“工程细节”的字段列为目标函数的一部分。

图 13.1 把偏好学习画成代理目标控制环。它展示了候选生成、标注协议、reward/preference model、PPO/DPO 类优化、独立评测和回流之间的关系。

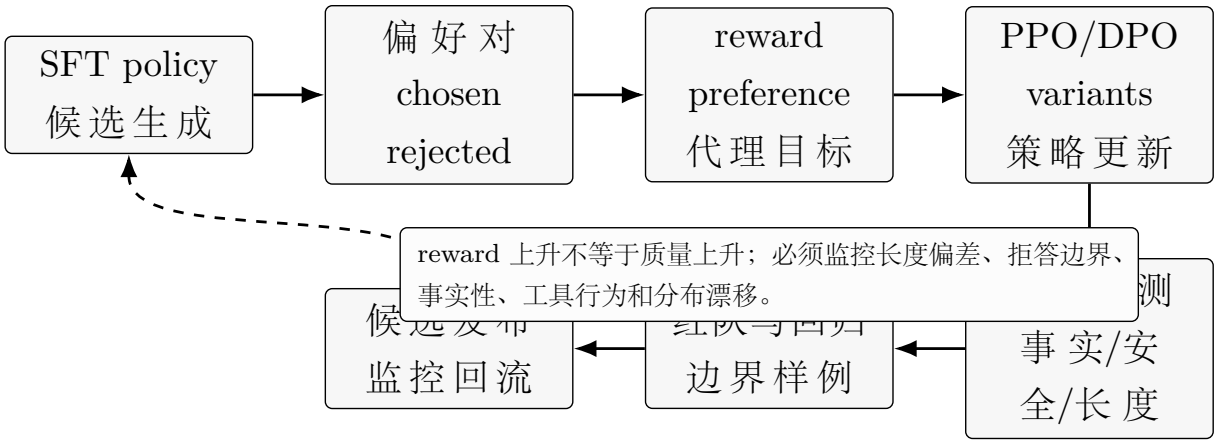

图 13.1: 偏好学习优化的是代理目标，需要独立评测闭环。该图是对 RLHF、PPO 和 DPO 管线的原创综合 [189, 212, 206]。

13.3 DPO 之后的偏好目标

DPO 没有结束后训练目标的设计空间。后续方法继续追问：是否必须有成对比较？是否必须有冻结 reference model？是否必须显式训练 reward head？是否必须进行在线 rollout？答案取决于数据可得性、系统成本和可接受的失败风险。

KTO 把对齐写成前景理论式优化，使用 desirable/undesirable 样本，而不是严格的 chosen 与 rejected 成对比较 [37]。这适合反馈天然以好/坏、可接受/不可接受形式收集的场景，但要求正负样本在任务、语言、风险等级和难度上平衡。ORPO 把 chosen response 的监督学习与抑制 rejected response 的 odds-ratio 惩罚放在同一个目标里，省去单独 reference model [51]。

SimPO 进一步采用 reference-free 奖励和显式 margin，并用平均 log probability 缓解长度伪影 [165]。这些方法不应被理解成单一排行榜，而应被理解成工程权衡。一个后训练报告必须说明标签类型、是否使用 reference model、如何处理回答长度、偏好数据覆盖哪些安全类别，以及哪些独立评测防止 reward hacking。

目标函数选择最好说明数据形态与优化器的对应关系。成对比较适合 DPO 或 IPO；单条好坏反馈适合 KTO；团队不能承受 reference model 的显存和吞吐成本时，ORPO 或 SimPO 等 reference-free 目标更容易落地；需要持续从当前策略采样困难样本时，PPO 或 RLVR 循环仍然有价值。报告中应把这个选择理由写出来，而不是只写最终采用了哪个 loss。

直接偏好目标还要报告“负样本是什么”。Rejected 可能是错误答案、啰嗦答案、有害答案、误拒答案、格式错误答案，或者只是另一个风格不同的可接受答案。若这些类别混在一起，优化器会把多种失败压成同一个方向。更稳健的做法是按 rejected 类型分桶，分别

报告 chosen/rejected margin、长度分布、拒答率、事实性、格式有效率和安全边界回归。

Reference-free 目标尤其需要回归护栏。去掉冻结 reference 以后，训练更省显存，但也少了显式的“不要离开基础策略太远”的信号。发布前应检查基础能力、低资源语言、代码、数学、事实召回和拒答边界；如果这些回归只在平均分之外的切片出现，说明偏好目标正在移动模型分布，而不是单纯改善用户偏好。

13.4 安全与 AI Feedback

Constitutional AI、helpful-harmless 训练和 RLAIF 用规则或 AI 反馈减少人工标注负担 [9, 8, 144]。但安全不是简单增加拒答数据。系统必须区分有害请求、合法敏感请求、误拒、缺少上下文的医疗或法律请求，以及越权工具行为。

安全数据常常不只有一个“更好回答”字段，而是同时给出有用性胜出项和每个回答是否安全。把它转成 chosen/rejected 对时已经在写政策：两个回答都安全时可以按有用性选；只有一个安全时通常应选安全回答；两个都不安全时应过滤、降权或单独标记。这个优先级必须写入数据卡，否则模型到底优化的是有用性、安全性还是混合目标，会变得不可解释。

这种转换还应保存被丢弃或降权的样本。很多安全边界恰好出现在“两边都不够好”的比较中：一个回答给出危险细节，另一个回答虽然拒绝但没有安全替代；一个回答事实正确但越权，另一个回答合规但误导。如果这些样本只被过滤掉，模型永远看不到政策边界附近的失败形态。可审计做法是把它们放入单独的 boundary 或 adjudication 切片，记录为什么不能直接进入 chosen/rejected loss，并在评测中保留。

AI feedback 可以放大覆盖面，也会放大规则盲点。若 AI 评审过度偏好模板化免责声明，策略可能变得保守而空泛；若 AI 评审没有可靠识别专业风险，策略可能在高风险领域给出过度自信建议。因此 RLAIF 仍需要人工抽检、政策版本化、红队集、误拒评测和真实部署回归。

AI 评审本身也需要模型卡式记录。至少应说明评审模型版本、系统提示、评分 rubric、是否看到 reference answer、是否看到工具输出、是否随机交换候选顺序、是否允许 tie、以及人类抽检比例。若 AI 评审和被训练模型同源，风格偏好和共同盲点会更强；若评审模型更新，旧标签和新标签不能无条件混用。

安全偏好数据还应覆盖“拒绝后的帮助性”。一个回答拒绝危险步骤后，可以给出高层解释、合法替代、求助资源、权限确认路径或安全审计建议。若 reward 只奖励拒绝词，模型会学会空泛拒答；若 reward 同时奖励安全重定向，模型更可能在合法敏感场景中保持有用。评测表应把 refuse 和 redirect 分开，而不是把它们都算作安全通过。

13.5 深入展开：偏好学习优化的是代理目标

本章从一个基本事实开始：很多助手输出没有唯一正确答案。摘要可以有多个合理版本，代码补丁可以有不同维护性取舍，敏感问题需要在有用和安全之间权衡。偏好学习把“给定标准答案”改成“在同一 prompt 下比较多个候选回答”。这个比较依赖 rubric、标注人、候选生成分布和政策版本，因此不是宇宙真理。

Reward model 通常用 Bradley-Terry 目标训练，学习 prompt-response 的标量分数差。它只能在训练分布附近可靠；一旦策略优化生成出标注人没见过的样式，就可能出现 reward hacking。模型会学会冗长、免责声明、奉承、过度自信或某些标注捷径，让 reward 上升而真实质量下降。本章要求画出 reward 分数和独立人类评测随 checkpoint 的关系，而不是只看 reward。

PPO 式 RLHF 把策略、reference model、reward model 和 value head 放进一个昂贵循环。KL 约束防止策略偏离太远，但 β 过大限制提升，过小导致风格崩坏和 reward hacking。DPO 通过隐式 reward 推导，把偏好优化写成离线监督目标，省去在线 RL 和显式 reward model，但仍依赖偏好数据覆盖范围和 reference model。

DPO 之后的 KTO、ORPO、SimPO 等方法进一步探索是否需要成对比较和 reference model。KTO 使用 desirable/undesirable 标签；ORPO 把监督学习和 odds-ratio 惩罚结合；SimPO 使用 reference-free margin 和长度归一化。本章的判断是：这些不是万能优劣排序，而是后训练工程空间中的不同取舍。关键问题仍是数据定义了什么行为，独立评测能否发现它没有定义的失败。

13.6 章节细节

13.6.1 对齐作为偏好建模

许多助手任务没有唯一正确答案，因此后训练常用偏好比较来表达质量。偏好不是客观真理，而是 rubric、标注者、候选生成器、用户群和政策版本共同定义的局部信号。对齐研究必须承认这种代理性，并把“偏好来自哪里”写进实验报告。

13.6.2 从标签到比较

偏好数据通常比较 chosen 和 rejected 回答。比较比单一评分稳定，因为标注者更容易判断两个回答哪一个更好；但它仍受候选质量影响。若候选都很差，chosen 只表示相对更好，不表示真正可发布。若候选差距太大，模型学到的又可能只是粗糙错误，而不是细粒度质量边界。

成对比较还会隐藏多目标冲突。一个回答可能事实更准但语气差，另一个可能更安全但信息量少。好的数据集应记录胜出原因、风险标签和不确定性，而不是只留下二元选择。否则 reward model 会把不同目标混成一个分数，后续策略优化很难知道该牺牲哪一项。

13.6.3 采样候选回答

候选回答来自当前模型、旧模型、教师模型、工具增强系统或不同采样温度。候选分布决定偏好数据覆盖哪些错误。只在容易样本上采样，会让后训练无法处理真实困难请求；只采样极差候选，又会让 reward model 不能区分高质量回答之间的细微差别。

候选采样参数也属于实验配置。Temperature、top-p、最大新 token 数、停止符、是否强制 EOS、是否允许工具调用，都会改变候选空间。偏好数据卡应说明这些参数，否则同一个 chosen/rejected 统计量在另一个采样分布下可能不再代表同一件事。

13.6.4 标注协议

标注协议应说明有用性、真实性、安全性、简洁性、引用和格式的权重。标注者需要处理冲突目标，并记录不确定性。没有协议，偏好数据只是不可解释的投票。对于安全样本，协议还要定义 comply、refuse 和 redirect：哪些请求应回答，哪些应拒绝，哪些应给出安全替代信息。

标注协议也要管理时间漂移。安全政策、产品语气、引用格式和工具权限都会变化。若旧政策下的标签和新政策下的标签混在一起，reward model 可能学习到自相矛盾的边界。高风险数据集应保留政策版本、标注日期、标注者角色、冲突仲裁记录和 gold examples。

13.6.5 Bradley-Terry 目标

Reward model 常用 Bradley-Terry 形式，让 chosen 分数高于 rejected。若 $r_\phi(x, y)$ 是 prompt-response 的标量分数，常见损失可以写作

$$-\log \sigma(r_\phi(x, y_w) - r_\phi(x, y_l)).$$

这个目标学习的是相对排序，不是绝对质量。对所有回答同时加同一个常数，不改变 pairwise 概率，因此 reward shift 本身不可辨识。

在代码中，reward model 常被实现为 `AutoModelForSequenceClassification` 加一个标量 score head，并交给 `RewardTrainer` 训练。MA-RLHF 实操使用 4-bit QLoRA、NF4、bf16 compute、LoRA 的 sequence-classification 配置和 `modules_to_save=[``scores'']` 保存打分头。这个实现说明 reward model 不是普通语言模型继续训练：它的可复现契约包

括 base checkpoint、quantization config、LoRA rank、score head 保存方式、tokenizer、模板、max length、Deepspeed 配置和 W&B run id。

RewardTrainer 期望每条样本提供 chosen 与 rejected 两套 token id 和 attention mask。因此渲染样本检查很重要：随机打印几条 tokenized 之前的文本，确认 prompt、system prompt、chosen answer、rejected answer 和 EOS 的位置正确。若把 rejected 字段误接到 chosen，或者把 HH-RLHF 的最后一轮 answer 边界切错，训练 loss 仍会下降，但模型学到的是反向偏好或截断伪影。

这种不变性在 PPO 中很重要。策略优化看到的是 reward、KL 惩罚、baseline 和归一化之后的有效信号。若训练日志只报告变换后的 reward，而不报告原始 reward 分布和缩放规则，后续结果无法复现。Reward 模型也不应只看验证集 pairwise accuracy；校准、切片表现和不确定性同样重要。

13.6.6 校准与不确定性

Reward model 在训练分布外可能自信错误。应检查分数分布、人类一致性、任务切片、语言切片、安全切片和不确定样本。RewardBench 式评测有助于比较 reward model 在聊天、推理和安全场景中的能力 [142]，但部署团队仍需要自己的领域内偏好数据。

校准差的 reward model 很容易在策略优化中被利用。若模糊样本得到很大的 reward margin，PPO 会把策略推向这些高分样式；若安全拒答样本全部被打高分，模型可能发展成过度拒答。实践上可以用 ensemble、人工复核、分数校准曲线、abstention 规则和高低分样例审查来降低风险。

Reward model 审计不应只报告 held-out pairwise accuracy。更有用的表应按任务、语言、长度、风险级别、候选来源和 margin bucket 分层，列出准确率、校准误差、平均 reward gap、人工争议率和样本数。对于高 margin 但人工争议高的样本，要抽样读原文；对于低 margin 但政策明确的安全样本，要检查 reward model 是否在关键边界上犹豫。若部署任务包含代码、医疗、金融或法律场景，还应单独报告领域专家复核，而不是依赖通用聊天偏好集。

13.6.7 过优化

策略如果过度追求 reward，可能学会冗长、奉承、免责声明、固定格式、引用堆砌或安全套话。Reward 上升不代表真实质量上升。独立人类评测和安全回归是发现 reward hacking 的必要手段 [16]。

过优化通常随 checkpoint 出现。早期模型质量提升，随后 reward 继续上升但人评、事实性或安全边界开始下降。报告应展示 reward、人评、长度、拒答率、事实错误率和红队通

过率随 checkpoint 的变化，并审查最高 reward 样本。只给最终 checkpoint 的平均 reward，无法说明模型是否已经越过代理目标的可靠区间。

13.6.8 带 PPO 的 RLHF

PPO 式 RLHF 使用策略、reference model、reward model 和 value head。KL 约束限制策略偏离 reference，平衡提升和稳定。它昂贵、复杂，但能直接优化序列级偏好。InstructGPT 的历史意义正在于展示了监督微调、reward modeling 和 PPO 组成的后训练流水线，可以让较小模型在指令跟随上被标注者偏好于更大的预训练模型 [189]。

PPO 的训练单位通常是 prompt-only query。当前 policy 采样 response，query 和 response 拼成完整文本，reward model 或 reward adapter 在最终 token 位置给出标量分数。训练再把 reward、response token、log probability、value prediction 和 reference log probability 送入 PPO step。这个数据流要求日志能追溯每个样本来自哪个 prompt、哪个策略版本和哪个 reward 版本。

13.6.9 语言模型 rollout 的 MDP 视角

RL 课件中的状态、动作、转移、奖励、折扣回报、状态价值和动作价值，在 LLM 后训练中都有直接对应。Prompt 加已生成前缀是状态，下一个 token 是动作，追加 token 是转移，完整回答被打分后得到 terminal reward。Advantage 不是抽象数学量，而是“这个 token 相对当前前缀的基线预期是否更好”的估计。

这个桥接能避免把 PPO 误解成普通交叉熵训练的替代品。交叉熵在给定目标 token 上做 imitation；PPO 在策略自己采样出的续写上优化回报，并用 KL 和 clipped objective 控制更新幅度。若 rollout 分布变坏，PPO 看到的状态也会变坏，因此 SFT policy 的初始质量仍然重要。

13.6.10 PPO 机制

PPO 通过 clipped objective、优势估计和值函数训练稳定更新。实现中要监控 KL、reward、entropy、长度、拒答率和崩坏样例。小超参数变化会显著影响风格和安全边界。MOSS-RLHF 的 PPO 分析强调 policy constraints，并建议监控 perplexity、response length、policy-reference KL 等 action-space 指标 [269]。

PPO 的稳定性不能只看 reward。Temperature、top-p、max new tokens、EOS 和 forced-EOS 设置都会改变 rollout 分布，因此也属于训练配置。若框架用共享参数或隐式 reference path 计算 KL，而不是显式加载单独 reference model，也要和 KL 系数一起记录。

Reward 缩放同样是目标函数的一部分。减 baseline、把 reward 除以常数、裁剪梯度、把 NaN reward 替换成固定小值，或者只读取最终 token 的 score，都可能让小实验稳定下来，但也改变 PPO 看到的有效信号。报告应同时记录原始 reward 分布、变换后的 reward 分布、无效 reward 数量，以及高分和低分样例。

一个 PPO run card 应把 rollout、打分和更新分开记录。Rollout 部分写 prompt mix、policy checkpoint、temperature、top-p、最大新 token、EOS 处理、工具权限和过滤器；打分部分写 reward model 版本、score head 位置、adapter 状态、reward clipping、KL 估计和无效分数处理；更新部分写 batch size、minibatch、PPO epoch、clip range、value loss、entropy bonus、梯度裁剪和 checkpoint 选择规则。这样才能判断质量变化来自数据、reward、KL、优化器还是采样分布。

PPO 还需要逐样本回放表，而不是只给训练曲线。每个 rollout 至少应能还原 prompt id、policy 版本、随机种子、query token 数、response token 数、原始 reward、变换后 reward、token 级 KL 总和、entropy、value error、是否触发 EOS、是否被安全过滤、以及更新前后的样本文本。若某个 checkpoint 的平均 reward 上升但误拒或格式错误也上升，逐样本记录能定位问题来自 reward 误分、KL 失控、value head 偏差还是采样配置漂移。

PPO 也不能被理解成“自动产生 helpfulness 的算法”。SFT policy 提供初始指令跟随接口和 rollout 分布，reward model 决定偏好优化方向，PPO 只是带约束地沿着这个代理目标移动策略。如果偏好数据没有覆盖某个能力、安全边界、语言或领域，PPO 往往会放大奖励模型盲点，而不是自动修复。

13.6.11 系统成本

RLHF 需要在线采样、reward 打分、策略更新、checkpoint 评测和大量日志。它比 SFT 或 DPO 更消耗算力和工程复杂度。成本应和实际质量收益一起评估：如果一个领域只有少量静态偏好数据，DPO 或 KTO 可能更合适；如果需要持续从当前策略采样困难样本，PPO 的在线循环更有价值。

Multi-Adapter RLHF 把这个成本问题具体化：active policy、reference model 和 reward model 可以共享一个 4-bit base，通过 policy LoRA、reward LoRA 和 reference path 切换角色。好处是显存显著降低；风险是运行时必须严格切换 adapter，并记录哪一个 adapter 产生了 reward、哪一个 adapter 被更新、哪一个路径用于 KL。

MA-RLHF 的 PPO 脚本正体现了这种成本压缩：带 value head 的 causal LM wrapper 加载 policy LoRA，同时把 reward adapter 挂到同一个基座上，`ref_model=None` 表示 reference 由框架共享或隐式处理。生成配置固定了 `max_new_tokens`、top-p、sampling、pad、EOS 和 forced EOS；PPOConfig 还设置 early stopping、target KL、初始 KL 系数、自适应 KL 和梯度裁剪。书稿中的“系统成本”不只是显存数量，而是这些角色切换和隐式 reference

语义必须可审计。

该脚本还把 reward 从 `compute_reward_score` 的最后 token 位置读出，再减去 baseline 后送入 `trainer.step`。这类实现应同时记录原始分数、baseline 后分数、无效分数、response 长度、KL、loss 和样本文本。若只在日志中看到 `ppo/mean_scores` 上升，无法判断收益来自真正偏好提升、回答变长、reward adapter 切换错误，还是 forced EOS 让文本更符合训练模板。

13.6.12 DPO

DPO 把偏好优化写成离线监督目标，绕开显式 reward model 和在线 RL。它更简单、更稳定，但仍依赖偏好数据覆盖和 reference model。DPO 不是免评测的捷径。

DPO 的关键不是只把 chosen 概率拉高、rejected 概率拉低，而是把带 KL 约束的奖励最大化目标改写成 policy 与 reference 的 log-ratio。对同一个 prompt 下的两个回答做 Bradley-Terry 比较时，和 prompt 只相关的归一化项会抵消。因此 policy 本身同时扮演生成器和隐式 reward model。

DPO 梯度还带有动态权重：如果当前隐式 reward 已经把 chosen 排在 rejected 前面，该样本更新较小；如果 rejected 仍被隐式 reward 评得更高，该样本更新更大。β 同时控制 reference KL 压力和隐式 reward 尺度。因此 DPO 虽然实现像监督学习，但它优化的不是普通 chosen-only likelihood，而是带 reference ratio 和 pairwise margin 的偏好目标。

DPO 实现中最容易出错的是 log probability mask。system、user、分隔符和可能的检索上下文属于 prompt；chosen 和 rejected 才是 assistant response。计算 policy 与 reference 的 response log probability 时，应只对 response token 求和，不能把 prompt token、padding token 或被截断的无效 token 算进 loss；如果部署模板依赖 EOS，则 EOS 通常也应作为 response 目标。

Policy 和 reference 必须使用同一个 tokenizer、chat template、special tokens 和 truncation 规则，reference 冻结且不反传梯度。若一个实现对 response log probability 求和，另一个实现按长度取平均，它们优化的已经不是同一个目标，因此长度处理必须在 DPO 报告中写清楚。

DPO 的长度过滤也应按 token 而不是原始字符串完成。实际 trainer 往往同时暴露 prompt 长度、target 长度和总长度限制。预处理报告应分别统计 prompt、chosen 和 rejected 因这些限制被丢弃或截断的数量，并检查 chosen/rejected 是否被对称处理。否则训练集会偏向较短回答，甚至把长但安全或长但有害的关键样本提前删掉。

在 MA-RLHF DPO 实操中，`DPOTrainer` 同时接收 `beta=0.1`、`max_prompt_length`、总 `max_length` 和 `max_target_length`。这些不是普通训练超参数，而是偏好目标定义：β 改变 policy-reference ratio 的尺度，prompt/target 限制改变哪些 token 参与比较，

QLoRA 和 gradient checkpointing 决定可训练参数与数值路径。训练报告应把这些字段和 tokenizer/template hash 一起保存。

DPO 数据入口还要区分 HH-RLHF 与通用 chosen/rejected schema。HH-RLHF 的 prompt 是最后一个 `###Answer` 之前的对话加 system prompt，chosen/rejected 是最后一轮 assistant 内容加 EOS；通用数据则通常先用 `format_prompt` 包装 prompt，再接 chosen/rejected。若把完整 chosen 文本当作 prompt，或者把 prompt token 也纳入 response log probability，DPO loss 仍能计算，但它优化的是模板复现而不是偏好排序。

DPO 训练中 chosen 和 rejected 的绝对 log probability 同时下降，并不一定说明实现错误。DPO 推动的是 chosen 相对 rejected 的 log-ratio margin，而不是对 chosen answer 做普通监督学习。序列概率要和所有其他续写竞争，reference 约束可能让两个候选都被压低，长度处理也会改变绝对和。应该改善的是在明确 scoring 规则下 chosen/rejected 的相对 margin。

报告中应同时画 chosen log probability、rejected log probability、两者相对 reference 的 log ratio、偏好 margin 和真实采样质量。另一类风险是近确定偏好或小样本偏好导致的过拟合。在 Bradley-Terry 模型中，如果经验数据总是认为 chosen 胜过 rejected，最大似然会倾向于把两者 reward 差距推向无穷大。

在 DPO 坐标下，这可能表现为把 rejected 的概率压到接近 0，使有限的 β 不再像稳定的实践护栏。IPO 把同一个 policy-reference log-ratio gap 回归到有限目标 margin，而不是让 chosen/rejected 分离无界增长 [7]。因此 β、目标 margin、标注噪声、early stopping 和 held-out 偏好 margin 都是目标函数的一部分，不是装饰性超参数。

DPO 实现应配套单元测试。第一，构造一条 prompt 和两条 response，确认 loss mask 只覆盖 assistant response token，并且 padding 与 prompt token 的 label 为忽略值。第二，用相同 policy 和 reference 初始化，检查 policy-reference log-ratio 初始接近 0。第三，交换 chosen/rejected 后 loss 方向应反转。第四，改变 response 长度时，求和与平均 log probability 两种模式应产生可解释差异。第五，在小 batch 上手算 Δ_θ，确认实现的 β 缩放和论文或配置一致。

偏好学习验收样本应覆盖三类“看起来能训练但不能发布”的情况。第一，reward model 在验证对上正确，却给冗长、奉承、模板化拒答或无证据引用极高分。第二，DPO 的训练 loss 下降，但 chosen/rejected 的相对 margin 只在短回答或单一语言中改善。第三，PPO 的 reward 和 KL 在预算内，却让合法敏感请求的误拒、结构化输出错误或工具权限失败上升。每一种情况都要求保存原文样本、切片指标和回滚 checkpoint，而不是只保存均值曲线。

DPO 验收还要检查“提升发生在哪里”。报告应按语言、任务类型、回答长度、风险类别、候选来源、chosen/rejected 相似度和 reference log-ratio margin 分桶。如果 margin 只

在短英文闲聊样本上扩大，而代码、中文、长回答、合法敏感请求或高风险拒答边界没有改善，训练 loss 下降并不能支持发布。对于 reference-free 目标，还要额外比较基础能力和拒答边界，因为少了冻结 reference 后，平均偏好提升更容易伴随旧能力漂移。

TRL 训练器验收。官方 TRL 文档已经把偏好训练器的数据契约拆得更细：preference dataset 可以显式提供 `prompt`、`chosen` 和 `rejected`，也可以把 prompt 隐含在 chosen/rejected 中；`DPOTrainer` 更推荐显式 prompt，而 `RewardTrainer` 常接受隐式 prompt，二者都支持普通字符串格式和 conversational message 格式 [117, 118, 126]。因此训练报告不应只写“使用 UltraFeedback”或“使用 HH-RLHF”，而要写清 schema 类型、chat template 版本、padding side、pad/EOS token、collator、截断模式，以及 prompt 与 response 的边界是由数据列、模板还是预处理函数确定的。

TRL v1 的总览页还把后训练方法按 online methods、reward modeling、offline methods 和 knowledge distillation 组织，并把 GRPO、RLOO、OnlineDPO、Nash-MD、PPO、XPO、RewardTrainer、PRMTrainer、DPO、BCO、CPO、KTO 和 ORPO 放在不同类别中 [129]。这个 taxonomy 对出版稿很有用：读者不应只问“用了哪个缩写”，还要问它是否在线生成、是否需要 reward model、reference model 或 pairwise preference model、是否支持 vLLM rollout、是否位于 experimental API、日志指标是否与目标函数匹配，以及 TRL 版本升级是否改变默认行为。一个方法如果从离线偏好对换成 prompt-only 在线采样，风险从 mask/length 错误转向采样、reward 函数、权重同步和评审模型漂移；这应在方法选择表中单独列出。

`DPOConfig` 中的 `max_length`、`truncation_mode`、`precompute_ref_log_probs`、`loss_type`、`beta`、`label_smoothing`、f-divergence 和 reference 同步选项都会改变偏好目标或数值路径，不能只放在命令行附录里 [118]。Reward model 训练也有自己的验收点：`RewardTrainer` 使用 preference 数据和 `DataCollatorForPreference`，并可通过 mean-zero reward 正则约束分数尺度，因此应报告 reward 分布、中心化系数、chosen/rejected margin、校准和分桶错误，而不是只报告 reward accuracy [126]。PPO 则必须把 rollout 长度、stop token、missing-EOS penalty、reward model 路径、policy/ref adapter 名称、KL 系数、value loss 和采样温度纳入 run card；这些字段决定在线采样分布与约束强度 [124]。

当前 `DPOTrainer` 还把偏好数据扩展到工具调用和 VLM。工具调用样本需要在对话消息中保留 `tool_calls`、`tool` role 响应，并在 `tools` 列给出可用工具 schema；RewardTrainer 的工具调用样本也需要同样记录 messages、工具调用、工具响应和工具 schema；VLM 偏好样本则通过 `image` 或 `images` 列传入图像。若 `max_length` 截断会删掉图像 token，官方文档建议在验证全数据长度后设置 `max_length=None`。因此多模态或工具偏好训练的 run card 还要记录图片列、工具 schema、processor、chat template、工具响应是否参与 mask、图像 token 是否被截断，以及截断策略如何在 chosen/rejected 之间保持对称，而不只是记录文本字段 [118, 126]。

同名“偏好优化”在不同 trainer 中还会改变数据形态。`KTOTrainer` 在当前 TRL 文档中位于 `trl.experimental.kto`，主要面向 unpaired preference，并把 paired chosen/rejected 拆成正负样本；`ORPOTrainer` 位于 `trl.experimental.orpo`，仍使用 preference dataset 并推荐显式 prompt；`OnlineDPOTrainer` 则从 prompt-only 数据出发，用 reward function 或 reward model 在线比较生成结果 [120, 123, 122]。所以方法选择表必须包含数据列、reference 需求、是否在线生成、是否实验性 API、日志指标和回归切片。否则同一个 CSV 在不同脚本中被自动转换后，表面上都是“偏好学习”，实际优化目标和发布风险已经不同。

`OnlineDPOTrainer` 尤其容易被误写成“动态版 DPO”。官方示例从 prompt-only 数据出发，在训练时让当前 policy 采样候选，再用 reward model 或 reward function 提供在线反馈；这意味着标注者已经从静态人类比较变成运行时评审器 [122]。出版级报告要记录评审器模型或函数版本、评审提示、是否随机交换候选、每步生成几条 completion、采样参数、reward 延迟和失败处理、是否使用 vLLM 生成、以及 reward graph 上 chosen/rejected 分数同时变化时如何解释。若这些字段缺失，Online DPO 的收益可能来自评审器偏好、当前策略采样变化或线上 feedback prompt，而不是来自偏好目标本身。

`RLOOTrainer` 把偏好训练重新拉回在线 RL 视角：数据集主要给 prompt，模型在训练中生成 completion，用 reward function 或 reward model 打分，再用 leave-one-out 基线估计相对优势，并显式估计 KL [127, 125]。这比 PPO 少了 value head 和一部分超参数，但 run card 仍要记录每个 prompt 的生成条数、reward 函数、解析失败如何跳过、reference 或 KL 设置、stop token、采样配置，以及 VLM prompt 中的 `image/images` 列。

`CPOTrainer`、`BCOTrainer` 和 `NashMDTrainer` 又是三种不同契约。CPO 源自机器翻译场景，是 DPO 近似目标的一类推广；BCO 把 prompt+chosen 映射为正类、prompt+rejected 映射为负类，用二分类 logit 作为隐式 reward，并要求记录 unpaired preference schema、reference model 和 `beta`；Nash-MD 则从 pairwise preference model 出发，寻找相对竞争策略更优的均衡策略 [116, 115, 121]。因此论文或工程报告不能把它们统称为“DPO 变体”，而要写清 objective family、数据配对假设、是否需要 reference 或 pairwise model、是否仍在 experimental namespace、以及和基础能力、安全边界相关的回归切片。

Trainer 指标也要按方法解释。当前 `DPOTrainer` 会记录 tokens、loss、entropy、token accuracy、chosen/rejected logits、logps、隐式 rewards、margin 和 accuracy；`KTOTrainer` 还记录 chosen/rejected 样本数，防止正负样本失衡被均值掩盖；`ORPOTrainer` 记录 log-odds、odds-ratio 和 SFT 部分的 `nll_loss`。这些字段不是装饰性 telemetry，而是判断目标函数是否在优化偏好、长度、格式还是样本配比的证据 [118, 120, 123]。

13.6.13 取舍

PPO、DPO、IPO、KTO、ORPO、SimPO 等方法在数据需求、稳定性、成本和目标假设上不同。PPO 能从当前策略持续采样并优化序列级 reward，但昂贵且容易放大 reward model 盲点。DPO 便宜稳定，却受静态偏好数据限制。Reference-free 目标降低显存和实现复杂度，但对策略漂移的显式控制更弱。

选择哪种方法应取决于任务、数据和系统约束，而不是取决于论文排行榜。需要回答的问题包括：偏好标签是成对比较还是正负样本？是否有可靠 reference？回答长度是否有伪影？安全政策是否需要优先于有用性？是否有独立评测能发现 reward hacking？没有一种目标能替代这些审计。

13.6.14 安全、拒答与 AI Feedback

RLAIF 和 Constitutional AI 用规则或模型反馈减少人工标注成本。它们能扩展安全数据，但也可能继承规则盲点和模型偏见。安全训练应区分有害请求、合法敏感请求、误拒和工具越权。一个合理的安全偏好集应包含近邻样本：非法入侵请求、合法安全审计请求、概念性教育请求和需要拒绝但可以安全重定向的请求。

拒答边界要和产品上下文绑定。医疗、法律、金融、儿童安全、网络安全和生物安全请求的可回答范围不同；带工具的代理还要考虑权限、数据访问和行动后果。若训练数据只包含明显有害样本，模型可能在改写、跨语言、角色扮演或工具调用场景中失效。若训练数据只强调拒绝，模型又会在合法敏感请求上过度保守。

13.6.15 政策作为训练对象

安全政策应像代码一样版本化。每个偏好数据批次都应记录使用的政策版本、例外规则、仲裁流程和已知争议。模型卡或训练报告应说明哪些政策变化触发了重新标注、哪些旧标签被废弃、哪些评测集被保留为回归集。

政策版本化还能减少评测混乱。若一个模型在旧政策下正确回答、在新政策下应该拒绝，简单胜率会给出错误结论。更好的做法是把评测样本按政策版本、风险类别和期望动作分层，并同时报告 comply、refuse、redirect 和 escalate 的准确性。

拒答评测应像分类问题一样报告混淆矩阵，但不能停在分类准确率。Comply 被错判为 refuse 是误拒，直接损害用户可用性；refuse 被错判为 comply 是漏拒，可能造成安全事故；redirect 被压成空泛拒答会失去帮助性；应该 escalate 却给出自动答案，会把系统带入没有授权或没有专业能力的区域。多轮场景还要检查政策状态是否保持一致：模型在第一轮拒绝危险细节后，不能在第二轮通过翻译、格式转换、角色扮演或工具调用泄露同一内容。

13.6.16 分布漂移

偏好学习最脆弱的地方之一是分布漂移。漂移可能来自用户群变化、语言和地区变化、模型能力提升、工具可用性变化、攻击者策略变化、政策更新、评测集泄漏或候选采样分布变化。一个在公共聊天偏好上表现很好的 reward model，未必适合法律、医疗、金融或代码审查政策。

漂移诊断要覆盖训练数据外的真实工作流。除了 held-out 偏好集，还应使用人工红队、领域专家审查、长任务评测、工具调用回放、多语言集、事实性检查和生产日志抽样。若训练 reward 和部署指标不一致，应优先怀疑数据定义或评测切片，而不是简单继续训练。

偏好数据回流不能直接等同于“把线上差评继续训练”。生产日志包含隐私、授权、地区合规和用户预期问题；用户点击、重试或投诉也不一定代表清晰偏好。可发布的回流管线应先做脱敏、保留期限检查、同意状态检查和风险分级，再把样本送入候选生成与标注流程。高风险样本还需要人工仲裁，而不是直接交给 AI judge 批量打标。

13.6.17 评测提醒

偏好分数不是地面真值。模型可能在标注分布上变好，却在真实用户、长任务或高风险领域退化。评测应包括人类偏好、事实性、安全、回归、成本、延迟和失败样例。还要明确标注者看到什么：是否看到引用、工具结果、系统消息、候选顺序和模型身份。

评测报告应避免只给一个总胜率。总胜率会掩盖长度偏差、拒答偏差、任务难度差异和切片退化。更好的报告同时给出任务切片、风险切片、语言切片、长度分桶、confidence margin、失败样例和人工复核结论。对于高风险领域，偏好胜率只能作为初筛，不能替代专家审查。

配套评测脚本也展示了一个常见陷阱：用 vLLM 生成 SafeRLHF 测试 prompt 的回答，再分别记录 reward mean/std 或 toxicity mean/std。这类指标适合做烟测，却不能单独决定发布。Reward mean 上升可能只是 reward model 偏好某种格式；toxicity 均值下降也可能来自空泛拒答。真正的评测表还要记录 comply/refuse/redirect 期望动作、误拒率、漏拒率、合法敏感请求的帮助性，以及 reward model 与独立人工评审的分歧样本。

上线后还要监控偏好模型无法直接看到的指标：用户重试、人工升级、投诉、引用失败、工具失败、安全事件、长对话中途放弃、回答长度漂移和拒答率漂移。若 reward model 离线分数稳定但这些指标变坏，说明偏好代理和真实任务之间已经产生裂缝。新的困难样本应回流到候选采样和标注流程，但回流必须遵守隐私、保留期限和用户同意约束。

发布门禁应把“偏好提升”拆成可否上线的问题。最低限度应同时满足：held-out 偏好不下降，事实性和格式任务无显著回归，unsafe compliance 不上升，false refusal 不超过阈值，长回答长度没有异常漂移，高风险切片经人工复核通过，服务延迟和成本在预算内。若

门禁项	必须同时查看	阻断或复核条件
偏好收益	held-out 胜率、margin、争议样本、人类抽检	只在 reward 上升，人评或争议样本变差
事实与格式	事实性、引用支持、JSON/schema、代码测试	平均偏好提升但结构化任务失败率上升
安全边界	unsafe compliance、false refusal、redirect 质量、多语言红队	漏拒上升、误拒超阈值或安全替代变空泛
长度与风格	response 长度、verbosity bucket、口吻、免责声明比例	胜率来自冗长、奉承或模板化拒答
系统成本	p50/p95 延迟、tokens/request、GPU 内存、回滚路径	质量收益不足以支付成本或回滚证据不足
回流治理	用户同意、脱敏、保留期限、风险分级、人工仲裁	线上样本未经授权或高风险样本直接进训练

表 13.1: 偏好学习发布门禁、同步证据与阻断条件检查表。

其中一个条件失败，正确做法通常是回到数据定义、政策版本或候选采样，而不是继续调大偏好 loss。

决策记录也要保留未采用 checkpoint 的证据。偏好学习经常出现多个候选：一个 reward 最高，一个人评更稳，一个 false refusal 更低，一个成本更低。发布报告应说明候选集合、排序规则、阻断项、人工复核意见、最终选择理由和回滚目标。这样后续事故复盘才能知道团队当时是在接受哪一种残余风险，而不是只看到一个被包装成“最优”的最终模型。

13.7 发布门禁与回流审计

偏好学习的发布结论不应写成“DPO 胜率提升若干百分点”或“reward mean 上升”。发布门禁要把代理指标、独立评测、系统成本和治理证据绑定到同一个决策表。一个 checkpoint 如果只在偏好数据的平均胜率上变好，却让事实性、格式有效率、引用忠实性、安全边界、响应长度、延迟或成本退化，就不是可发布改进。门禁的作用是提前说明哪些退化可以接受、哪些必须阻断发布、哪些需要人工复核，而不是事后为已经选择的 checkpoint 找解释。

表 13.1 将偏好学习收益和发布风险放在同一门禁表中，而不是只比较胜率。

回流审计尤其容易被忽略。线上差评、重试和投诉不是自动可训练标签；它们可能来自 UI 误解、检索失败、工具权限、用户情绪、地区政策、隐私限制或真实模型错误。可发布的回流流程应先把样本分成普通质量问题、安全边界问题、隐私问题、工具失败和政策争议，再决定是否进入候选采样、人工标注、AI 辅助评审或只进入监控报表。高风险样本

不能只靠 AI judge 批量贴标签；至少要保留人工仲裁、政策版本和不进入训练的理由。

门禁还要记录反事实选择。若最终没有选择最高 reward checkpoint，而是选择更保守的 checkpoint，报告应说明它在哪些切片更稳，例如 false refusal 更低、长任务格式更好、人工复核争议更少或服务成本更低。这样读者才能理解偏好学习不是追单一分数，而是在代理目标、真实任务和发布风险之间做可审计取舍。

13.8 关键术语、实现要点与练习

关键术语。 Preference data 是比较或评分；rubric 是标注规范；reward model 学习代理偏好；TRL trainer taxonomy 把后训练方法按 online、offline、reward modeling 和 distillation 分类；MDP 用状态、动作、转移、奖励和折扣因子描述序列决策；advantage 衡量某个动作相对状态价值基线的增益；leave-one-out baseline 用同组其他 completion 估计相对基线；KL regularization 限制策略偏离；DPO 是离线直接偏好目标；DPO implicit reward 是用 policy 与 reference 的 log-ratio 表示的隐式奖励；Online DPO 用当前策略生成候选并由运行时评审器提供反馈；RLOO 是使用 leave-one-out 优势估计的在线强化学习训练器；BCO 用二分类 logit 构造隐式 reward；CPO 是源自偏好优化的 DPO 近似目标；Nash-MD 用 pairwise preference model 和 mirror descent 寻找均衡策略；response-only log probability 只在 assistant response token 上计算序列概率；tool-calling preference data 把工具调用和工具响应纳入 chosen/rejected 对话；multimodal preference data 把图像或多图列纳入偏好样本；reference-free objective 不依赖冻结 reference model；policy version 表示偏好标签所依据的安全与产品规则版本；reward margin 是 chosen 与 rejected 在代理目标下的分数差；RLAIF 使用 AI 反馈；reward hacking 是优化代理目标却损害真实目标。

实现要点。偏好数据应记录 rubric、标注者、候选生成策略、采样参数、政策版本和争议样本；数据格式要记录 standard/conversational、prompt-only/preference/unpaired/stepwise 类型、转换函数、`datasets` 版本和 chat template；reward model 要看校准、切片、分数分布、margin bucket、高低分样例和 score head 保存方式；PPO 要监控 KL、reward、entropy、perplexity、长度、adapter role、reference path、拒答率和崩坏样例；Online DPO 要记录评审器模型或函数、评审提示、生成条数、采样参数、vLLM 或普通生成路径和反馈失败处理；RLOO 要记录生成条数、reward 函数、跳过样本规则、leave-one-out 优势和 KL 估计；DPO、KTO、ORPO、BCO、CPO、Nash-MD 和 SimPO 等偏好目标要监控数据配对假设、reference 或 pairwise model、chosen/rejected log-probability gap、reference ratio、margin、mask 单元测试、token 级过滤和回归；工具调用或 VLM 偏好训练还要记录 tools schema、tool response、image columns、processor 和截断策略。

最小流水线。一个可审计的偏好学习项目至少应包含以下步骤：

1. 定义任务范围、风险类别、政策版本和标注 rubric。
2. 从目标用户分布、困难样本和安全近邻样本中收集 prompt。
3. 用多个策略版本和采样配置生成候选回答，并记录 provenance。
4. 进行人类或 AI 辅助标注，保留胜出原因、安全标签和不确定性。
5. 清洗偏好对，检查重复、泄漏、长度伪影、语言覆盖和标签冲突。
6. 训练 reward model 或直接偏好目标，并保留 reference、tokenizer、template、special token 和量化/LoRA 配置。
7. 用 held-out 偏好、人评、事实性、安全和长度切片选择 checkpoint。
8. 若使用 PPO，记录 rollout 分布、reward 缩放、KL、value loss、adapter 切换和无效 reward。
9. 若使用 DPO 类目标，审计 response mask、截断、长度归一化、token 级过滤和偏好 margin。
10. 若使用 RLOO、BCO、CPO 或 Nash-MD，记录在线生成、unpaired preference、二分类 reward 或 pairwise preference model 等方法特有契约。
11. 若使用 Online DPO，记录 prompt-only 数据、运行时评审器、评审提示、生成条数、采样配置、候选顺序随机化和 feedback 失败处理。
12. 在部署前运行红队、回归和成本评测，并把失败样例送回数据循环。

练习。

1. 写出 Bradley-Terry pairwise loss 并解释 reward shift 不变性。
2. 构造一个带 10% 标签噪声的 toy preference 数据集，比较 reward accuracy 和 calibration curve。
3. 把一次聊天模型 rollout 写成 MDP：定义状态、动作、转移、终止条件、reward 和 discount。
4. 从 KL 正则化 reward objective 推导 DPO 的 policy-reference log-ratio 形式，并解释归一化项为什么会在 pairwise 比较里抵消。
5. 比较 PPO 和 DPO 的系统成本，并说明各自最需要哪些日志。

6. 检查一个 DPO collator，确认 prompt、padding、截断 token 和 EOS 的 loss mask 是否正确。

7. 为偏好数据写一个 manifest schema，要求能区分候选来源、采样参数、政策版本、风险类别、标注争议和过滤原因。

8. 用同一偏好数据比较 DPO 和一个 reference-free 目标，报告长度分桶下的 margin。

9. 设计一个 DPO 单元测试 batch，手算 chosen/rejected 的 policy-reference log-ratio，并验证实现中的 β 和 loss mask。

10. 为一个带工具调用或图像输入的 DPO 数据集写验收清单，检查 `tools` schema、`tool_calls`、工具响应、图像列、processor、chat template 和 `max_length` 是否会破坏 response mask。

11. 设计 comply/refuse/redirect 安全边界评测，并说明误拒和漏拒如何分别计分。

12. 为安全偏好评测画 comply、refuse、redirect、escalate 混淆矩阵，并说明每一种混淆对应的产品风险。

13. 写一份偏好学习评测备忘录，列出至少五种分布漂移来源和对应诊断。

14. 审查 20 个最高 reward 样本，判断 reward 是否在奖励冗长、奉承或模板化拒答。

15. 为 KTO、ORPO、SimPO 和 DPO 写一个目标选择表，说明每种目标需要的数据形态、reference 成本、长度处理和主要回归风险。

16. 为 RLOO、BCO、CPO 和 Nash-MD 写一个训练器选择表，列出数据类型、是否在线生成、是否需要 reference、reward 或 pairwise model、核心日志指标和发布阻断项。

17. 设计一条线上偏好数据回流管线，列出隐私处理、同意状态、风险分级、AI 评审、人类抽检和重新训练门禁。

18. 审计一个 SafeRLHF 到 chosen/rejected 的转换函数，分别处理双安全、单安全和双不安全样本，并说明哪些样本进入 boundary 切片。

19. 审计一个 DPO 训练脚本，把字符串长度过滤改成 token 级 prompt/target/total 长度统计，并报告 dropped pair 数量。

20. 为一个 4-bit QLoRA reward model 写 run card，列出 base checkpoint、score head、LoRA rank、quantization、bf16、template、max length 和评测切片。

13.9 结构化检查表

13.9.1 偏好学习失败诊断

失败模式	症状	诊断
Reward hacking	reward 上升，人评下降	画 reward-人评曲线，审查高 reward 样本
长度偏差	长回答无实质优势仍获胜	长度控制对比和长度分桶胜率
过度拒答	合法敏感问题被拒	comply/refuse/redirect 边界集
安全泛化不足	改写后绕过拒答	多语言、工具、多模态红队
标注漂移	不同时期标签分布变化	政策版本、gold examples、校准会议
Reward 过度自信	模糊样本 margin 很大	校准曲线、ensemble、abstention
DPO mask 错误	训练看似稳定但回答退化	审计 response-only token 和截断规则
Reference drift	直接偏好目标破坏旧能力	冻结 reference 记录和回归评测
安全标签合并错误	SafeRLHF 双不安全样本被当作普通偏好	过滤或单列边界/仲裁切片
字符串长度过滤	中文、代码或特殊 token 样本被偏置删除	token 级 prompt/target/total 长度统计
Adapter 角色混淆	PPO reward、policy 或 reference 路径互换	记录 LoRA role、score head 和 KL/reference path

第十四章　推理与测试时计算

14.1　推理是预算化计算

Chain-of-thought、自一致性、分解、程序化推理、树搜索和验证器都可以被看作测试时计算分配策略 [242, 239, 255]。重要问题是：生成多少候选、用什么评分、花多少 token、何时调用工具、何时停止。

把推理写成预算问题，可以避免把“长输出”误当成“真推理”。额外计算可能表现为更长 scratchpad、多个候选答案、验证器排序、搜索树展开、工具调用，或一个经过 RL 训练、会探索和检查自己工作的策略。方法报告应说明计算了什么、预算如何分配、候选答案如何选择、预期失败模式是什么，以及评测如何同时记录答案正确率、轨迹有用性、轨迹忠实性、延迟、成本和分布外鲁棒性。

可见推理轨迹尤其需要谨慎。它可能是工作记忆，也可能是说服性解释、事后合理化，或三者混合。面向用户的系统通常不应把全部中间 token 都暴露出来；更稳妥的接口是让模型使用私有草稿，再返回简洁答案、可检查依据和必要引用。这样仍能获得部分测试时计算收益，同时降低隐私、安全和误导风险。

本章的最低报告单元不是“用了推理模型”，而是一份推理预算说明。它至少要写清问题分布、可见或隐藏轨迹策略、最大输出 token、候选数、采样温度、是否调用工具、是否使用 verifier、选择规则、拒答阈值、平均延迟、峰值延迟和单位任务成本。若没有这些字段，两个系统即使都写着 CoT 或 self-consistency，也可能在实际计算量上相差一个数量级。

图 14.1 把推理方法写成预算路由器。它把直接回答、短推理、多样本、工具/代码、verifier 和拒答放进同一个控制面，方便比较质量收益和成本。

14.2　提示推理、验证与搜索

CoT 的关键不是一句提示词，而是把输入、推理轨迹和最终答案组织成可学习的协议 [242]。Few-shot 示例、示例顺序、答案抽取器和解码参数都会改变结果。自一致性进一步

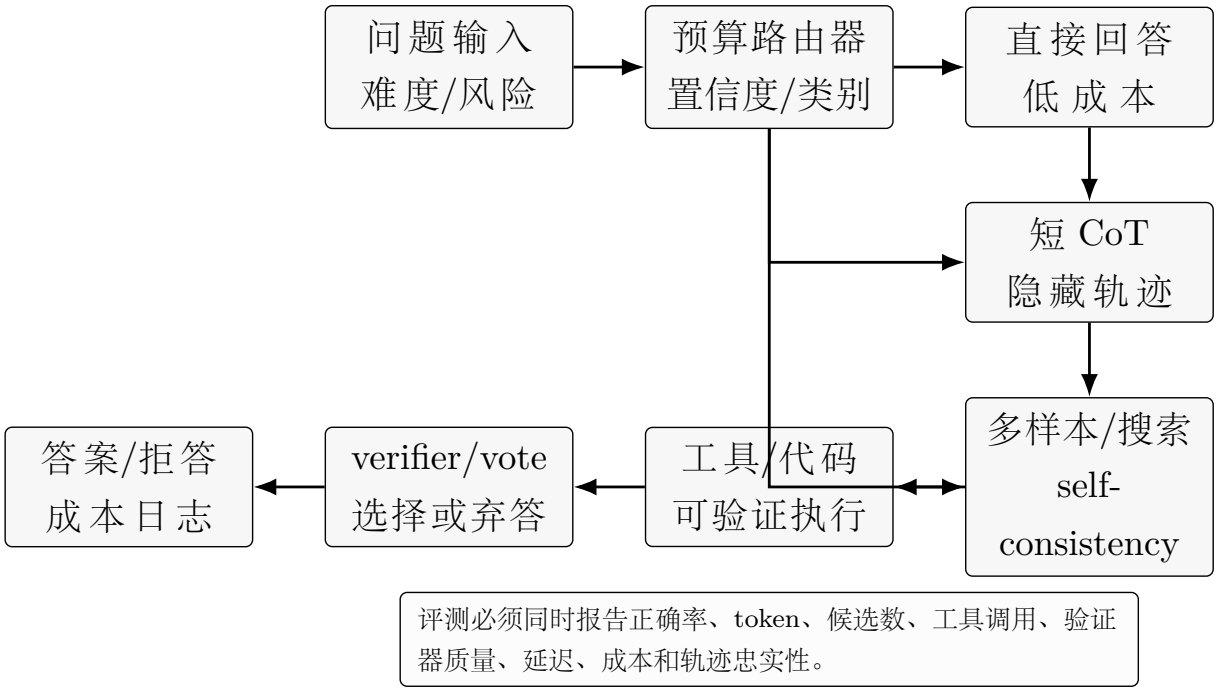

图 14.1: 推理是测试时计算分配。该图是对 chain-of-thought、self-consistency、tree search 与 reasoning test-time scaling 的原创综合 [242, 239, 255, 220]。

把一次生成扩展成多次采样，再把最终答案归一化后投票 [239]。它适合候选错误相对独立的任务；若模型的错误高度相关，更多样本只会重复同一个误解。

Sample-and-rank 把候选生成和候选选择分开。排序器可以是单元测试、符号检查器、schema validator、过程奖励模型、LLM 裁判或人工流程。它真正依赖的是 verifier 是否比 generator 更可靠，而不是 verifier 名字是否更高级。代码、数学和结构化输出通常更适合自动验证；开放式分析常常只能得到另一个偏好模型，仍然需要校准、失败审查和人类抽样复核。

树搜索把推理状态显式展开。Tree-of-Thought 的可复现接口包括 thought 粒度、generator prompt、state evaluator、搜索策略、宽度、深度、停止规则和 token 预算 [255]。这些字段缺一不可。一个浅层 BFS、一个带回溯 DFS、一个 MCTS 风格 rollout 和一个 beam search 都可能被笼统称为“搜索推理”，但它们的成本曲线、失败模式和可解释性完全不同。

14.3 RLVR 与 GRPO

DeepSeek-R1 表明，基于可验证任务的大规模强化学习可以激发反思、验证和动态策略等推理行为 [30]。后续 RLVR 研究进一步追问可验证奖励何时会促进真实推理，何时只是在优化最终答案或格式捷径 [243]。GRPO 用组内相对优势替代 learned critic，降低内存

和复杂度，但依赖样本组多样性和奖励可靠性。RLVR 的关键风险是奖励捷径：模型可能学会通过格式或漏洞满足验证器，而不是真正稳健推理。

14.4 自适应测试时计算

测试时扩展研究正在从“多花 token 是否更强”转向“如何按难度分配预算”。综述把测试时 scaling 分解为 scale 什么、如何 scale、在哪里 scale、如何评估 [263]；预算化推理进一步区分固定预算控制和按置信度/难度自适应分配 [3]。

自适应预算的核心是路由器。简单问题应快速直接回答；中等问题可以使用短 CoT 或少量样本；可验证的代码和数学题可以调用工具或 verifier；高风险问题可能应该拒答、降级或交给人工。路由器本身也要评测：过度升级会浪费成本，漏掉难题会降低质量，把安全敏感请求送入长自由推理则会增加策略绕过和敏感中间信息暴露风险。

14.5 深入展开：推理方法首先是预算分配方法

本章把 reasoning 看成测试时计算分配。Chain-of-thought 通过生成中间步骤给模型更多条件 token；self-consistency 采样多个推理路径再投票；decomposition 把复杂任务拆成子任务；program-of-thought 把部分推理交给代码解释器；tree search 和 MCTS 把候选扩展成搜索树；verifier 或 reward model 选择更可信结果。它们的共同点不是“更像人思考”，而是用更多计算换取更高成功率。推理轨迹并不一定忠实。模型生成的 CoT 可能是解释性文本，而不是内部真实计算；隐藏或编辑轨迹也不一定改变答案。因此区分“有助于性能的外部工作区”和“可解释性证据”。如果轨迹被用于用户展示，还要考虑安全、隐私和误导风险；如果轨迹只用于内部选择，则要评估它是否真的提高正确率。

RLVR 和 GRPO 类方法把可验证奖励引入后训练。数学答案、代码单元测试、形式化证明和规则检查提供比人类偏好更便宜的奖励，但也会产生捷径。模型可能学会迎合验证器格式、利用测试漏洞或过度优化可验证任务，导致泛化不足。Process supervision 可以约束中间步骤，但标注成本高且定义困难。预算控制是前沿推理模型的产品接口。一个系统可以给简单问题短答，给难题长推理，给代码任务调用测试，给事实问题检索，给高风险问题拒答或要求人工。本章要求每个推理方法报告成功率、token 成本、延迟、采样数、验证器质量、失败模式和 benchmark contamination 风险，而不是只报告最高分。

14.6 章节细节

14.6.1 推理作为预算化计算

推理方法可以理解为如何在测试时花额外计算。生成更多步骤、更多候选、更多工具调用或更多验证，都在改变预算。关键问题是额外计算是否带来可测收益。

14.6.2 Chain of Thought

Chain-of-thought 让模型生成中间步骤，常能提升数学和多步任务表现。它也可能产生不忠实或误导性解释。是否展示推理轨迹，需要考虑安全、隐私和用户理解。

CoT 原论文应被读作一种 prompting 协议，而不是一句万能提示词。Few-shot 示例是 input、reasoning trace、final output 三元组；论文在算术任务中常用少量人工写的 CoT 示例，并和标准 input-output prompting 对照 [242]。收益主要出现在足够大的模型和真正需要多步推理的任务上；较小模型可能生成不连贯轨迹并降低准确率，简单一步题也可能收益很小。因此 CoT 报告应说明模型规模、示例来源、示例顺序或随机化、示例数量、temperature、答案抽取规则，以及对照组是否使用相同 shot 数和相近 prompt token 预算。

14.6.3 Self-Consistency

Self-consistency 采样多个推理路径，再用投票或聚合选择答案。它用更多 token 换更高准确率。收益取决于候选多样性、错误相关性和聚合规则。

自一致性的成本应按样本数记录。若单条推理平均生成 T 个 token，采样 K 条大致消耗 KT 个输出 token；这些样本可以并行生成，但仍会占用解码吞吐、KV cache 和验证预算。它适合离线评测、高价值数学或代码任务，不适合所有低延迟产品。答案归一化也是算法的一部分：数学答案、单位、大小写、JSON 格式和代码输出若归一化不稳定，投票结果会把格式差异误当成语义差异。

因此，自一致性报告不能只写“$K = 5$”。应同时给出采样温度、随机种子、去重规则、答案抽取器、无效答案处理、并行度、批处理方式和选择器准确率。若 K 条样本里曾经出现正确答案，但投票或 verifier 没有选中它，pass@k 反映的是搜索空间覆盖，不是部署中的单次可靠性。若所有样本都共享同一个错误前提，增加 K 只会增加成本和自信度，而不会产生真正多样性。

多样性也需要被测量，而不是从采样次数推断。报告可以记录最终答案熵、去重后候选数、轨迹编辑距离、不同工具计划数、不同中间公式或代码结构数，以及正确候选首次出现的位置。若所有轨迹只是同一模板的轻微改写，self-consistency 实际上只是昂贵的重

复生成；若答案归一化器把不同答案错误合并，投票会制造虚假的一致性。出版稿应把候选覆盖、选择器质量和最终部署可靠性分开报告。

14.6.4 分解与程序化推理

复杂问题可以拆成子问题，或把计算交给程序执行。Program-of-thought、工具调用和代码解释器把部分推理外包给可验证系统。边界是工具权限、执行安全和错误传播。

14.6.5 Sample-and-Rank

Sample-and-rank 生成多个候选，再用 verifier、reward model 或规则排序。它适合有可比较候选的任务。排序器本身若不可靠，会选择看起来好但实际错误的答案。

验证器可以是单元测试、符号检查器、schema validator、过程奖励模型、另一个 LLM 裁判或人工流程。代码、数学和结构化输出中，验证器往往更强，因为部分正确性可以执行或形式化检查；开放式分析中，验证器常退化为另一个偏好模型，并继承偏好学习的校准和过优化问题。报告 sample-and-rank 时，应说明候选生成温度、候选数、去重和归一化规则、验证器训练来源、拒绝或弃答阈值，以及验证器和生成器是否共享盲点。

验证器还要区分“接受答案”和“解释为什么接受”。单元测试只说明程序通过当前测试，不说明隐藏边界条件；数学 parser 只说明最终表达式等价，不说明推导忠实；LLM 裁判能读自然语言，却可能奖励格式、长度和熟悉措辞。一个稳健系统通常会把 verifier 分层：先做 schema 和解析检查，再做可执行或符号检查，最后才把开放式判断交给模型裁判或人工抽样。每层都应记录拒绝原因，否则调试时无法判断失败来自生成器、答案抽取器、验证器还是任务本身。

14.6.6 结果监督与过程监督

Outcome supervision 只看最终答案，process supervision 评价中间步骤。过程监督能提供更密集信号，但标注成本高，且步骤是否忠实仍需研究。可验证任务常用最终答案奖励。

过程奖励模型在数学任务中常把一条解题轨迹拆成多个步骤，逐步估计局部有效性。它能发现“最终答案碰巧正确但中间推理错误”的情况，也能在树搜索中提前剪掉坏分支。代价是标注协议复杂：不同解法可能都有效，同一步的粒度也可能有争议。若过程标签奖励的是表面严谨、格式完整或熟悉模板，模型会学习写出看起来更像证明的文本，而不一定学会更可靠的推理。

14.6.7 树搜索

Tree-of-thought 和 MCTS 把推理扩展为搜索问题。模型生成状态和动作，评分器决定扩展方向。搜索带来更强探索，也带来成本、剪枝和评估误差问题。

Tree of Thoughts 论文把接口拆成四个问题：什么算一个 thought，是短语、方程行、段落计划、工具动作还是部分棋盘；下一步 thought 如何生成，是从 CoT prompt 独立采样，还是用 proposal prompt 在受限空间中列出不同候选；状态如何评估，是 value prompt、vote prompt、程序检查器还是 learned value model；最后由什么搜索算法消费这些分数。论文在 Game of 24 和 creative writing 中使用浅层 BFS，在 mini crosswords 中使用带回溯的 DFS。照搬“ToT”这个名字没有意义，必须报告 thought 粒度、generator、evaluator、宽度/深度限制、停止规则和 token 预算。树搜索也可以进入训练闭环，而不只是测试时包装器。TS-LLM 采用类似 AlphaZero 的形式：policy 提议动作，learned value function 评价部分状态，搜索收集轨迹，再用这些轨迹更新 policy 和 value model [237]。工程上必须显式定义 environment interface，包括状态序列化、动作生成或合法动作集合、状态转移、终止条件和奖励。还要区分 terminal reward 已知的任务，以及 GSM8K、Game24、证明类任务这类需要 non-terminal value 估计的任务。报告结果时不能只给准确率，还要给 branch factor、最大动作长度、rollout 数、value/PRM 权重、生成 token 成本和失败分支。

本地树搜索代码也暴露了这些接口变量：测试脚本会把 CoT greedy、CoT-SC、无终止奖励 MCTS 和有终止奖励 MCTS 分成不同设置，并显式配置 temperature、k_{maj}、max action、max length、simulation 数、UCT 探索常数、剪枝比例、token 上限和随机种子。这些字段不是实现细节，而是算法定义的一部分。若一篇报告只说“使用 MCTS”却不说明 root 如何初始化、节点价值如何更新、终止奖励是否可见、是否按 visit count 选动作、是否重用搜索树和如何聚合候选答案，就无法复现实验，也无法比较它和自一致性到底差在哪里。

14.6.8 可验证奖励强化学习

RLVR 用数学答案、单元测试、规则检查或执行结果作为奖励。它减少主观偏好依赖，适合代码、数学和结构化任务。难点是可验证任务覆盖有限，模型可能过拟合验证器。

可验证奖励的吸引力在于规模化：每个 rollout 不需要人工偏好标签，只要自动检查器能接受或拒绝即可。它的边界也同样清楚：奖励覆盖的是“可检查的正确性”，不一定覆盖真实任务价值。代码可能通过可见测试但隐藏边界条件很差；数学答案可能正确但推导不可迁移；结构化抽取可能学会迎合 parser。RLVR 报告应列出 verifier 的输入输出、答案抽取规则、隐藏测试或变体测试、失败样本，以及 reward 上升但人工质量不升的反例。

14.6.9 GRPO

GRPO 用组内相对优势替代显式 value model，降低训练复杂度。DeepSeek-R1 等工作显示大规模 RLVR 可以激发推理行为。仍需评估分布外任务、轨迹质量和安全边界。小规模 R1-style/GRPO 复现实验必须固定可复现契约：base checkpoint、数据切分、prompt template、答案格式、reward functions、temperature、最大 completion 长度、每个 prompt 的 generation 数、KL/reference policy 设置、LoRA 或全参微调，以及 rollout 是否由独立推理引擎生成。reward 不是中立指标，常见实现会混合 exact answer parser、boxed answer 提取、`think` 与 `answer` 标签格式检查、推理步骤启发式、长度奖励、cosine reward 与 repetition penalty。还要保证 global batch size 能被 generation 数整除，否则组内相对归一化会变成实现细节。

Rollout engine 也是实验的一部分。一些小规模 R1-style 运行会留出一张 GPU 给 vLLM，其余进程用 Accelerate、ZeRO 或 PEFT 训练。这样会引入同步契约：trainer 要周期性把当前权重 gather 或 merge 到推理引擎，把生成的 completion 广播回训练 rank，在第一个 EOS 后 pad 和 mask completion，并在所有进程 gather reward 后再计算组内相对优势。若这些步骤被省略或只隐含在代码里，收益可能来自 stale rollout weights、重复样本或进程本地 reward normalization，而不是 GRPO 目标本身。

当前 TRL 的 `GRPOTrainer` 文档把这条路径进一步产品化：trainer 从 prompt 生成多条 completion，用 reward function 或 reward model 打分，在同组内归一化优势，再计算 KL 与 GRPO loss；日志包括 token 数、step time、completion 长度、per-reward 均值和标准差、reward standard deviation、zero-std 组比例、entropy、KL 和 clip ratio [119]。这些字段能暴露两类常见失败：所有样本同分导致组内优势退化，以及格式奖励上升但任务奖励没有上升。

vLLM 生成集成不能只写成“加速 rollout”。官方文档区分 colocate 与 server 模式，并提醒训练策略和推理引擎之间存在 training-inference mismatch，需要用截断 importance-sampling correction 等机制约束偏差 [119]。如果 server 模式使用独立 GPU、独立地址或 sleep mode，run card 应写清 GPU 隔离、权重同步频率、生成参数、reward function 是否异步、工具或环境状态是否会跨样本泄漏。

GRPO 的核心假设是同一 prompt 的多条样本能形成有意义的相对比较。如果一组样本全错，组内优势仍会把“相对不那么差”的模式推高；如果验证器接受格式捷径，这种捷径会被快速放大。最小日志契约应包括任务准确率、格式准确率、各 reward component 均值、样本 completion、回答长度、token 吞吐、KL 或参考策略距离，以及 held-out benchmark。没有这些字段，很难区分真正推理改进、格式改进、reward parser artifact 和单纯预算增加。

小规模 R1-style 代码还说明 reward 往往是组合函数，而不是一个干净标量。常见组件

包括最终答案正确性、`think/answer` 标签格式、分步推理启发式、长度奖励、cosine-shaped 长度缩放和重复 n-gram 惩罚。它们分别优化不同对象：格式奖励让输出可解析，准确率奖励绑定任务，长度奖励抑制无意义 overthinking，重复惩罚减少循环文本。若不单独记录各组件均值和权重，reward 上升可能只是标签更稳定、答案 parser 更宽松或输出更短，而不是推理策略真正变好。

14.6.10 DeepSeek-R1 类流水线

R1-style 报告不能只写“做了 RL”。一个可复现流水线通常包含冷启动或无冷启动基线、可验证任务集合、reward parser、格式约束、rollout 生成、策略更新、蒸馏和安全对齐。DeepSeek-R1 的公开报告把 R1-Zero、冷启动数据、推理 RL、拒答与偏好数据、蒸馏模型放在同一条路线中讨论 [30]。Qwen3 进一步把 reasoning 和 non-reasoning 模式、多语言和开放权重实践作为产品与研究接口的一部分 [250]。这说明推理能力不是单一训练目标，而是数据、奖励、模式切换、服务预算和安全策略共同产生的系统行为。

前沿系统卡也应当放进本章视野。OpenAI o3/o4-mini 系统卡把更长推理、工具使用、安全评测和风险框架一起报告 [181]；Kimi K2 把 agentic coding、工具使用、长上下文、稀疏 MoE 和后训练联系起来 [137]。这些报告提供工程证据，但不是最终解释。读者应追问：提升来自更强 base model、更多 RL rollout、更好 verifier、更大测试时预算、工具管线，还是 benchmark 与训练分布的重叠。

推理蒸馏和模式切换也要写成契约。若大模型用长轨迹产生教师数据，小模型或普通模式只学习最终答案，报告应说明是否保留中间步骤、是否截断、是否过滤错误轨迹、是否把隐藏草稿变成可见输出，以及蒸馏后是否仍需要测试时额外预算。reasoning mode 和 non-reasoning mode 的比较也不能只看平均分；它应同时报告延迟、长度、拒答、安全切片和简单题过度思考率。

14.6.11 推理时扩展

Inference-time scaling 研究花更多测试时计算如何提升能力。可以扩展 token、样本、搜索深度、工具调用或 verifier 预算。评价时应报告成本曲线，而不是只给最贵设置下的分数。

一个粗略服务成本可以写成

$$C_{\text{serve}} \approx C_p + KLC_d + C_{\text{verify}} + C_{\text{tools}},$$

其中 C_p 是 prefill 成本，C_d 是每个 decode token 成本，K 是候选数，L 是平均输出长度。公式没有展开 batching、KV cache、并行调度和硬件利用率，但足以提醒读者：同样的准

确率提升，在离线证明搜索、高价值代码修复、交互客服和移动端助手里的产品价值完全不同。出版级报告应给出若干成本-质量点，而不是只展示最大预算下的最好分数。

14.6.12 预算强制

预算强制让模型在固定 token 或步骤内完成任务，或按难度自适应延长。它把推理能力和资源管理联系起来。真实产品需要在质量、延迟和成本之间选择。

预算强制不是简单把 max tokens 调大。它包括停止词、继续思考提示、验证器阈值、采样参数、工具预算和最终答案抽取规则。延长推理可能帮助多步数学，但也可能伤害事实问答、感知型多模态任务或本来已经答对的简单问题：模型会在后续 token 中推翻正确答案。额外计算只有在能获得纠错信号时才有意义；否则只是更贵的模式补全。

预算控制也应有验收表。至少比较短、中、长三个预算档位，记录首次正确答案出现位置、最终答案是否被后续推翻、停止触发原因、平均和 p95 token、p50/p95 延迟、工具调用率、弃答率和 verifier 失败率。若更长预算只提升 pass@k，却让 pass@1、事实性、安全或成本变差，报告应把它列为过度思考回归，而不是简单宣称 reasoning 更强。

14.6.13 计算最优分配

不同问题需要不同预算。简单问题多采样会浪费，困难问题预算不足会失败。自适应策略应估计难度和置信度，并有停止规则。

测试时计算可以按顺序、并行、树结构或工具增强四种形态扩展。顺序扩展生成更长轨迹或多轮修订；并行扩展生成多个独立尝试再投票或排序；树结构扩展把预算分配给更有希望的部分状态；工具增强把计算交给检索、代码执行、搜索或外部求解器。一个实用 router 应在 direct answer、CoT、自一致性、工具调用、人工升级或拒答之间选择，并单独评测路由器本身。路由器过度升级会让成本爆炸，漏掉难题会损失准确率，把安全敏感问题送入长自由推理还会增加风险。

路由器评测应像分类器一样写混淆矩阵：简单题被送去昂贵推理、难题被短答、可验证题未调用工具、高风险题未拒答或未人工升级，都是不同类型的错误。发布日志还应记录路由版本、阈值、输入难度桶、风险桶、预算上限和回滚策略，否则线上成本漂移或安全事故很难追溯到具体策略变更。

成本曲线比最大预算分数更重要。若一个系统在 $K = 1$ 时成功率一般、在 $K = 32$ 时很强，它适合离线证明搜索或高价值代码修复，不一定适合交互式客服。若一个系统依靠 verifier 选择候选，就要同时报告生成器 pass@k、verifier 选中正确候选的概率、选错类型和总成本。没有选择器质量，pass@k 只是“某个候选曾经对过”，不是部署可靠性。

14.6.14 生成配置与辅助解码验收

预算策略最终要落到一次可复现的生成调用。Transformers 的 `GenerationConfig` 文档把输出长度、停止、解码策略、cache、logits 处理和性能相关开关集中到同一个配置对象中；例如 `max_new_tokens` 比 `max_length` 更适合表达“本次最多生成多少 token”，`stop_strings` 定义字符串级停止条件，`do_sample`、`num_beams`、`temperature`、`top_p`、`min_p` 和 `cache_implementation` 会共同改变质量、成本、延迟和可复现性 [76]。因此，推理报告不能只写“使用默认生成参数”。默认值、模型仓库里的 `generation_config.json`、运行时覆盖参数、模板渲染和服务框架默认值必须一起保存。

聊天模板也是生成配置的一部分。最新 chat template 文档说明，推理时常用 generation prompt 让模型进入 assistant 回复位置；若要续写最后一条 assistant 消息，则使用 final-message continuation 语义，两者不能混用。推理模型还可能把可见回答和隐藏或半隐藏的 reasoning 字段分开，例如模板中出现 reasoning-content 或 thinking 字段 [63]。这意味着“同一条 prompt”如果模板字段不同，实际推理预算和可见输出边界也不同。报告应保存渲染后的 token 序列、special token、stop token、reasoning 字段策略和用户可见字段策略。

辅助解码进一步说明“推理更快”不等于“推理更强”。Hugging Face 的 assisted decoding 文档把 speculative decoding、prompt lookup decoding、self-speculative decoding 和 universal assisted decoding 放在同一组低延迟方法下：小助手模型、提示中的重叠 n-gram、中间层 early-exit 或不同 tokenizer 的助手都可以先提出候选 token，再由主模型验证或对齐 [61]。这些方法的目标通常是减少昂贵前向次数或降低延迟，而不是扩大搜索空间；因此评估时应固定质量目标，报告 tokens/s、p50/p95 latency、接受率、回退率、batch 支持限制、助手模型版本、tokenizer 对齐和内存占用。若只报告 wall-clock 变快，却没有证明输出分布、停止条件和安全过滤保持一致，就不能把它当作 reasoning 改进。

表 14.1 把生成配置和推理预算的记录项显式化，便于复现推理结果。

14.6.15 前沿推理系统

前沿系统把预训练、SFT、RLVR、工具、搜索和服务调度结合起来。o 系列模型以及 DeepSeek-R1、Qwen3、Kimi K2 等系统报告，都把推理预算作为能力组成。研究需要把模型和推理控制器一起评测。

因此，前沿推理系统的比较应按系统层拆开：基础模型能力、后训练数据、可验证奖励、采样和搜索预算、工具管线、拒答策略、可见轨迹策略、服务延迟和安全评测。只比较一个 benchmark 平均分会把这些层混在一起。一个模型可能因为更多采样而更强，因为 verifier 更强而更强，也可能只是因为答案抽取器更宽松而更强。

生成层字段	必须记录	常见误读
`GenerationConfig`	最大新 token、停止串、采样或 beam、logits 处理、cache 实现	默认参数等于可复现实验
聊天模板	generation prompt、prefill、reasoning 字段、special token 注入	prompt 文本相同就等于模型输入相同
采样与候选	temperature、top-p、min-p、随机种子、候选去重、答案归一化	多采样必然提供独立证据
辅助解码	助手模型、tokenizer、early-exit 层、n-gram 查找、接受率、回退率	speculative decoding 是新的搜索策略
服务约束	cache 类、batch 限制、streaming、超时、取消语义、p95 延迟	离线加速会自动转化为线上吞吐

表 14.1: 推理生成层字段、必须记录项与常见误读检查表。

14.6.16 答案准确率不够

只看最终答案会掩盖偶然猜对、工具误用、不可解释成本和安全问题。推理评测还应看步骤、成本、稳定性、拒答、校准和失败类型。尤其在高风险任务中，过程证据很重要。

推理评测至少应记录答案指标、轨迹策略、计算预算、选择方法、污染控制和成本指标。答案指标可以是 exact match、单元测试、proof check、人类 rubric 或偏好；轨迹可以隐藏、可见、摘要化或不可用；预算要写明 max tokens、样本数、工具预算和 wall-clock limit；选择方法要区分 greedy、majority vote、verifier rank 和人工选择。只报最高准确率会隐藏核心取舍：一个系统用二十倍 token 换来五个百分点提升，在定理证明中可能值得，在客服场景中可能完全不可接受。

pass@k 曲线比单点分数更诚实。若单次成功率为 p，理想独立样本下至少一次成功的概率随 k 增加而上升，但真实样本高度相关，且选择器未必能识别正确答案。评测应报告一组成本-质量点，例如 token、延迟或美元成本对应的准确率。这样读者才能判断前几个样本是否有效、边际收益何时耗尽，以及 verifier 是否真正把正确候选选出来。

成本曲线还应标出 dominated 策略：若一个设置更贵、更慢且准确率不高，就不应因为名字更像“推理”而保留。比较不同模型时，应固定预算画 pass@1、pass@k、被选中正确率和弃答率，也应固定质量目标反推需要多少 token、延迟和工具调用。对样本较小的推理 benchmark，还要给 bootstrap 置信区间或按题型分桶，否则一两个难题的变化就可能被误读为测试时扩展规律。

推理评测还需要污染和鲁棒性控制。数学、代码和竞赛题常被公开讨论，训练集中可能包含题目、答案、解析或近似变体。一个可信评测应包含私有题、时间切分、改写题、隐藏测试、扰动样例和失败复核。对 agentic 代码任务，还要记录工具权限、测试沙盒、重试

次数和最终 diff 是否真正解决问题。

14.6.17 轨迹忠实性

模型给出的推理文字未必是内部真实原因。轨迹可能是事后解释，也可能被训练成符合人类期望。把轨迹当作检查证据前，应先验证其忠实性。

14.6.18 推理边界测试

边界测试包括干扰信息、错误前提、反事实、长链条、多解问题和不可解问题。强模型应知道何时继续推理，何时停止，何时拒绝。推理越强，越需要避免自信地扩大错误。

还应加入信息不足、外部知识缺失、工具结果与模型先验冲突、安全敏感拒答、容易题被诱导过度思考等样本。这些样本能区分有用 deliberation 和冗长模式补全。一个可靠系统不仅要在难题上多花计算，也要在无解题上弃答，在危险题上转向安全策略，在简单题上快速停止。

14.6.19 实现层：推理栈

一个可靠的测试时推理栈通常有五层。第一层是 prompt 或 policy，规定 scratchpad 是隐藏、可见还是摘要化；第二层是预算控制器，限制 max tokens、样本数、branch factor、工具调用和 wall-clock time；第三层是 verifier 或 acceptance test，在数学、代码、schema、检索证据或安全策略允许的范围内验证候选；第四层是 router，把困难、低置信度、高风险或高价值请求升级到更贵策略、工具、人工或拒答；第五层是 telemetry，记录预算、延迟、路由策略、候选数、验证结果、选中原因和用户可见答案。

这五层应相互独立地审计。安全检查不应只依赖模型自己的自由 scratchpad；事实任务常常需要检索和引用检查，而不是更长的无证推理；软件工程任务需要测试、静态分析、沙盒、超时和最终 diff 复核；数学任务需要答案归一化、等价验证和错误步骤抽样。若 telemetry 只保存最终回答，系统上线后就无法解释一次失败是因为路由器没升级、预算过早耗尽、verifier 漏检、工具超时，还是模型在可见 rationale 中编造了理由。

Telemetry 也不能无边界保存。推理轨迹、工具输入、检索片段和失败候选可能包含用户隐私、商业数据或安全敏感中间步骤。发布级系统应说明哪些字段被原样保存，哪些只保存摘要、哈希或计数，保留多久，谁能访问，以及事故复盘需要哪些最小字段。否则“为了可观测性保存全部 CoT”会把调试问题变成新的隐私和安全风险。

14.6.20 推理系统故障诊断

推理系统的事故复盘不应只重放最终回答。团队需要把一次请求还原成路由、预算、候选、验证、工具和用户可见答案的链路。若模型在简单问题上长篇推导后改错答案，问题可能不是能力不足，而是停止规则和置信度估计失效；若 pass@k 很高但线上失败率仍高，问题可能在选择器识别不了正确候选；若 verifier 接受了错误代码，问题可能在测试覆盖、沙盒或答案抽取器；若事实问答越想越错，问题可能是检索证据没有被绑定到最终声明。出版级报告应把这些失败模式写成可检查证据，而不是归因给“推理模型不稳定”。

观察到的症状	可能根因	优先检查证据
简单题越想越错	路由器过度升级或停止规则过晚	难度分桶、实际 token、首次正确位置、停止触发原因
pass@k 高但线上差	候选选择器、答案归一化或 verifier 选错	正确候选出现率、选中正确率、归一化日志、被拒正确样本
代码候选通过测试后仍失败	可见测试太窄或沙盒和生产环境不一致	隐藏测试、静态检查、依赖版本、超时和最终 diff 复核
事实问答长推理后编造	缺少检索绑定或引用检查只看格式	检索命中文档、证据到声明映射、无证声明率、引用失败样本
安全问题被长 CoT 绕过	策略检查依赖模型自身轨迹	外部 policy gate、风险类别、拒答/转向日志、多轮复现
成本突然上升	难题路由阈值、样本数或工具预算漂移	路由版本、预算分布、p95 延迟、工具调用率、取消率

14.7 关键术语、实现要点与练习

关键术语。Chain-of-thought 生成中间步骤；self-consistency 多样采样再选择；thought decomposition 定义搜索中每一步展开的推理单元；verifier 评价候选；process reward model 评价中间步骤；search-guided 训练用搜索轨迹训练 policy/value；RLVR 用可验证奖励；GRPO 用组内相对优势；GRPO rollout contract 记录生成引擎、权重同步、reward function、组内方差和 KL 修正；test-time scaling 在推理时增加计算；budget forcing 控制推理长度；GenerationConfig 是记录生成长度、停止、采样、beam、cache 和 logits 处理的配置合同；assisted decoding 用助手模型、提示查找或 early-exit 候选来降低解码延迟；speculative decoding 让助手先提出 token、主模型再验证；pass@k 表示 k 次尝试中至少一次成功的概率或估计值；cost curve 把 token、延迟或美元成本与准确率连接起来；answer normalizer 把候选答案映射到可比较形式；reward parser 把输出解析成奖励输入；telemetry 是记录预算、路由、验证和失败原因的运行证据。

实现要点。推理方法必须报告生成候选、评分器、预算、延迟、失败模式和污染风险；生

成调用要保存 generation config 文件、运行时覆盖参数、chat template、generation prompt 或 prefill 语义、reasoning 字段策略、停止串、cache 实现和服务默认值；自一致性要报告答案归一化、选择器质量和样本相关性；树搜索要报告分支、rollout、value/PRM、终止奖励可见性和 token 成本；辅助解码要报告助手模型、tokenizer 对齐、接受率、回退率、batch 限制、延迟分位数和质量回归；轨迹不等于忠实解释；可验证奖励要防止格式捷径和测试漏洞；GRPO 复现要固定 reward、格式、generation 数、batch 约束、rollout 引擎同步、vLLM 模式、importance-sampling correction、组内 reward 方差和各 reward component；自适应预算应按难度、置信度、风险和任务价值分配；线上推理栈要保存 telemetry 以支持事故复盘。

练习。

1. 比较 single-shot、CoT、自一致性和工具执行的成本。

2. 为数学任务设计 verifier，并列出可能被利用的漏洞。

3. 为 Game of 24 或一个小型 puzzle 写 Tree-of-Thought 设置，说明 thought 单元、generator prompt、state evaluator、搜索策略、宽度/深度限制和 token 预算。

4. 设计一个按难度增加采样数的自适应策略。

5. 解释为什么 pass@k 不等于部署中的单次可靠性。

6. 为一个 GRPO 复现实验写最小日志清单，区分 reward parser、格式奖励、长度奖励和真实任务成功率。

7. 为一个使用 TRL `GRPOTrainer` 与 vLLM 生成的实验写 run card，记录 colocate/server 模式、权重同步、importance-sampling correction、reward standard deviation、zero-std 组比例和 per-reward component。

8. 设计一个推理评测卡，要求同时报告答案指标、轨迹策略、计算预算、选择方法、污染控制和成本曲线。

9. 为一个代码修复 agent 设计 sample-and-rank 管线，说明候选生成、单元测试、静态检查、超时、沙盒和最终选择规则。

10. 构造三个“过度思考会变差”的样例，说明路由器应如何提前停止或转向拒答。

11. 比较隐藏 scratchpad、摘要化 rationale 和完整可见 CoT 的产品风险、调试价值和评测偏差。

12. 为一个 R1-style/GRPO 小实验设计 reward component 表，分别记录答案正确性、格式、长度、重复惩罚、样本 completion、rollout 权重同步和 held-out benchmark。

13. 设计一个测试时推理 telemetry schema，覆盖路由策略、预算上限、实际 token、候选数、verifier 结果、工具错误、拒答原因和最终用户可见答案。

14. 给一个推理服务写生成配置验收清单，覆盖 `GenerationConfig`、chat template、stop strings、cache、assistant model、prompt lookup、self-speculative 或 UAD 路径，以及每条路径的延迟和质量回归测试。

14.8 结构化检查表

14.8.1 推理方法比较

方法	增加的计算	主要风险
CoT	中间 token	轨迹不忠实、泄露敏感推理
Self-consistency	多样采样	成本高、选择偏差
Program-of-thought	解释器执行	代码安全、环境依赖
Verifier	候选评分	验证器漏洞、reward hacking
Tree search	扩展多个状态	状态爆炸、启发式偏差
RLVR/GRPO	后训练采样与奖励	奖励捷径、任务覆盖窄

14.8.2 推理系统报告卡

字段	必须报告
任务分布	数学、代码、事实问答、agent、风险类别、私有或公开测试比例
轨迹策略	hidden scratchpad、visible CoT、摘要 rationale、引用或证据格式
预算控制	max tokens、样本数、branch factor、工具预算、停止规则、路由策略
候选与选择	采样温度、去重、答案归一化、verifier、majority vote、rank 阈值
验证器	训练来源、输入输出、隐藏测试、已知盲点、人工复核抽样
成本曲线	tokens、延迟、美元成本、硬件占用、pass@1、pass@k、选中正确率
安全与污染	prompt injection、防泄露、benchmark contamination、拒答与升级规则

第十五章　多模态与生成式基础模型

15.1　模态作为接口

多模态模型把图像、视频、音频、文档和语音接入语言模型。图像、音频、视频和传感器流并不是天然的文本 token 序列，因此多模态模型首先要解决接口问题：如何把连续信号转成语言模型可以使用的表示，同时保留任务所需的信息。CLIP 通过图文对比学习建立联合表示 [204]；Flamingo、BLIP-2、LLaVA 等系统展示了冻结视觉编码器、投影层、query transformer 和跨注意力等连接方式 [2, 149, 156]。早期 VLM 常被理解为“给语言模型接一个视觉编码器”：编码器把图像变成 patch 或 embedding，connector 把这些表示映射到语言模型隐藏空间，语言模型再完成问答、解释或结构化输出。这个设计简单有效，但也继承了两端的弱点：视觉编码器可能忽略小字和空间关系，语言模型可能用文本先验补全并不存在的视觉事实，connector 可能为了省 token 丢掉细节。因此，多模态能力不能只看回答是否流畅，还要看它是否真正读到了图像、图表、页面或视频片段。图 15.1 展示了模态编码器、基础模型核心、生成器和行动输出之间的接口层。

更工程化的说法是：每个模态都需要一份接口契约。输入侧要说明来源模态、预处理、分辨率、裁剪、帧采样、音频切片、OCR/ASR 工具和图像占位符；模型侧要说明 encoder、connector、视觉 token 数、position policy、attention mask、label mask 和冻结策略；服务侧要说明 encoder latency、prefill、decode、生成采样、缓存释放和取消策略；治理侧要说明来源、日志、隐私、安全过滤和生成媒体凭证。没有这些字段，“同一个模型支持图片、语音和视频”只是产品描述，不足以支撑复现实验或发布决策。

最新 Transformers 多模态聊天模板文档把这份契约落到了消息结构：文本模型的 `content` 可以是字符串，而多模态聊天的 `content` 应是带类型的条目列表，图像、视频、音频和文本都以显式 `type` 进入同一个消息序列；Processor 负责把聊天模板、tokenization 和媒体预处理合在一起 [109]。这意味着“模板一致”不再只比较渲染后的字符串，还要比较 `input_ids`、`attention_mask`、`pixel_values`、视频帧张量和网格元数据。高质量报告应保存原始消息 JSON、模板版本、每个媒体条目的来源或哈希，以及 `apply_chat_template` 返回的键集合；否则模型能生成答案，也无法证明图像、视频或音频确实按预期进入了模型。

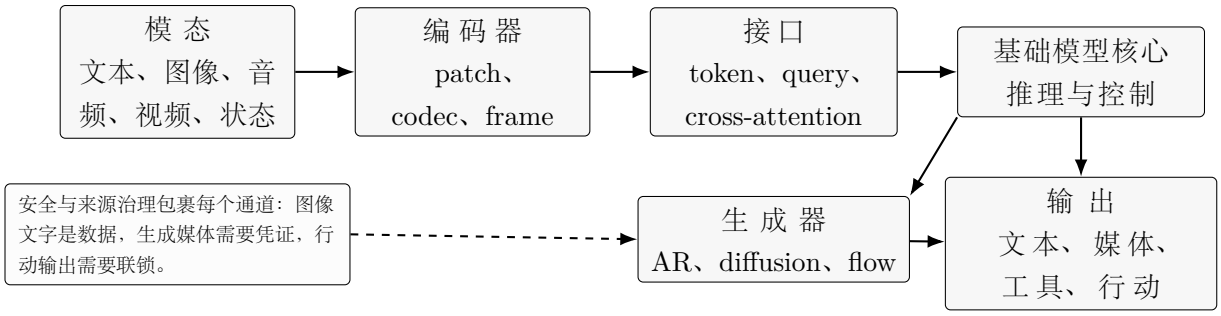

图 15.1: 统一多模态系统首先是接口设计问题。共享核心模型不意味着各模态可以共享同一套成本、评测和安全边界。

CLIP 式图文对齐的核心是把匹配图文对拉近、把不匹配对推远。设一批样本有 N 个图文对，图像 embedding 为 $f_\theta(I_i)$，文本 embedding 为 $g_\psi(T_j)$，相似度除以温度 τ 后进入 softmax；正例是同一个下标的图文对，批内其他文本就是负例。这个目标使自然语言可以定义视觉类别，也适合检索和弱标注。但它学习的是相似度空间，不是完整视觉推理。网页 caption 往往描述显著物体，遗漏左右、遮挡、数量、小字和图表坐标，因此下游 VLM 仍要通过高分辨率数据、OCR、文档问答、空间推理和负例训练补足表示空洞。

把 CLIP 写进工程报告时，不能只写“用了 CLIP 特征”。图像侧的 resize、center crop、RGB 转换、均值方差归一化会定义模型实际看到的证据；文本侧通常有固定 context length、起止 token 和截断策略，长类别名或复杂 prompt 被截断后就不再是原始语义。CLIP 的文本特征也不是任意 token 的平均，而是由文本编码器、结束 token 位置和投影矩阵共同决定。若更换裁剪、分辨率、tokenizer 或文本 pooling，就已经改变了视觉任务接口。

官方 Image Processor 文档也提醒，图像预处理不是无害前处理，而是模型输入特征的一部分：processor 负责加载或接收图像、resize、normalize、转换 tensor，并可能执行模型专用后处理；输入像素值范围、`do_rescale`、通道顺序、backend、`return_tensors` 和 device 都会改变复现条件 [108]。因此多模态 run card 应记录 processor class、processor 配置文件、backend 是 torchvision 还是 PIL、resize/crop/pad/normalize 参数、输入值域和输出 `pixel_values` 形状。文档问答或图表读数失败时，首先应检查小字是否在 crop 或 rescale 中丢失，而不是直接归因给语言模型推理差。

对比学习还依赖批内负例。单机小 batch、分布式全局 batch、重复 caption、近义类别和错误配对都会改变损失面。零样本分类看似只是写几个 prompt，其实类别集合本身就是模型行为的一部分：加入“其他”、合并细粒度类别、删除敏感类别或改写提示词，都会改变误报、漏报和公平性。因此可发布系统应记录 prompt 模板、类别 taxonomy、负例构造、温度/scale、特征归一化和校准方法，而不是只给一个 top-1 分数。

15.2 图文对齐与连接器

图文对齐给 VLM 一个视觉基础，但语言模型还需要知道视觉特征应该插到哪里、以多少 token 插入、是否可以跨层读取，以及训练时哪些位置应该产生文本损失。Projection-only 连接器最容易接入：视觉特征经过线性层或 MLP 变成 LLM 隐藏维度，再作为视觉 token 放进上下文。它便宜、稳定、适合先冻结大部分参数，但如果图像被切成大量 patch，prefill 成本会迅速上升，空间细节也可能被简单投影抹平。

Q-Former 或 resampler 用少量可学习 query 从视觉特征中抽取信息，相当于在 encoder 和 LLM 之间放一个可训练瓶颈。这个瓶颈能控制视觉 token 数，适合通用对话和低成本服务；代价是小字、表格线、计数和局部关系可能在压缩时丢失。Cross-attention adapter 则让语言层在若干深度直接读视觉特征，更适合交错图文、多图和视频帧，但实现更侵入，显存和训练复杂度更高。端到端多模态训练能共同优化视觉和语言，但需要更大的数据、算力和遗忘控制。

ViT 视觉编码器本身也有接口假设。图像被切成固定 patch，patch embedding 加上位置 embedding 后进入 Transformer；class token 更适合全局分类或检索，patch token 才承载局部文字、边界和空间关系。输入分辨率改变时，位置 embedding 插值、padding、切图或多尺度 crop 都会改变证据粒度。文档、图表、遥感和医学图像常常不是“整图语义”问题，而是局部读取问题；若 connector 只接收 class feature 或强压缩后的少量 query，模型回答流畅也可能没有足够证据读出细节。

因此 connector 选择应和任务证据绑定：检索和粗分类可以用全局向量，普通 VQA 可以用压缩 query，OCR/表格/坐标读数更需要保留 patch 网格或引入专门 OCR 管线，多图比较和视频则要保留图像顺序、帧时间和跨图指代。出版级实现说明不只列架构名，还要说明取的是哪一层视觉特征、是否去掉 class token、patch 如何合并、位置如何重编号、视觉 token 是否参与 KV cache，以及训练时哪些模块冻结。

Qwen3-VL 的 Transformers 接口把这些元数据显式暴露出来：模型前向不仅接收 `input_ids`、`attention_mask`、`position_ids` 和 cache，还接收图像 `pixel_values`、视频 `pixel_values_videos`、`image_grid_thw`、`video_grid_thw` 和多模态 token type；labels 中设置为 `-100` 的位置会被 loss 忽略 [111]。这给实现验收提供了很具体的打印项：每张图和每段视频的 temporal-height-width 网格、视觉 token span、文本 prompt span、assistant response span、被忽略的 label 数量、以及 cache 增量解码时是否只传入未处理 token。若这些张量没有对齐，多模态模型可能不是“看不懂图”，而是根本没有在正确位置看到图。

表 15.1 汇总多模态连接器取舍，提醒读者把视觉 token 和任务证据一起报告。

连接器	优点	主要风险
线性/MLP 投影	简单、便宜、容易复用已有 LLM	token 过多、空间 grounding 弱、细节压缩
Q-Former/resampler	控制视觉 token 数，保留冻结骨干	OCR、图表和计数细节可能丢失
Cross-attention	适合多图、视频和交错媒体	架构改动大，训练和显存成本高
端到端训练	感知和语言可以联合优化	昂贵，容易数据不平衡或遗忘

表 15.1: 多模态连接器优点与主要风险检查表。

15.3 统一理解与生成

现在的前沿不再只是“看图回答问题”。理解模型和生成模型正在合流：一个系统可能既要读图、读文档、听语音、理解视频，又要生成图像、编辑图像、合成语音或生成视频。统一多模态理解与生成综述把相关工作分成 autoregressive、diffusion-based 和 hybrid 三类，并指出 tokenization、跨模态注意力、数据平衡和 benchmark 是共同难点 [266]。统一模型同时面对两个方向相反的任务。理解要求忠实抽取：读清输入、绑定实体、保留空间和时间证据，并在证据不足时拒答。生成要求可控合成：创造新的媒体，遵循指令，保持一致性，并满足安全和来源要求。只用 VQA 准确率或图像审美分数评价这样的系统是不够的；它需要同时报告感知、推理、编辑、可控性、多样性、延迟和误用风险。

Janus-Pro 展示了自回归统一的一条路线：它在多模态理解和 text-to-image instruction following 上同时改进，并强调训练策略、数据规模、模型规模和生成稳定性 [20]。这类工作说明，理解图像的表示不一定最适合生成图像；统一系统常常需要共享高层语义推理，同时在高保真生成处保留模态专门接口。

统一模型的设计难点不只是把任务放进同一个 checkpoint。理解任务通常要求“少编造、多引用”：回答应能回到图像区域、页面文本、帧时间或音频片段。生成任务通常要求“多控制、少漂移”：模型要保持主体、风格、布局、文字和安全边界。若一个系统共享 decoder，却在理解和生成之间使用不同 tokenizer、latent 或安全过滤器，报告就必须把这些差异写清楚。否则读者无法判断能力来自统一表征、外部工具，还是来自多个模块的级联。

因此，统一理解-生成系统需要能力剖面，而不是单一总分。报告应把输入模态、输出模态、任务类型、使用工具、生成目标和风险等级交叉成回归矩阵：图像问答提升后，OCR、图表读数、多图指代、文生图编辑、语音延迟和安全拒答是否仍然通过？若生成质量靠更强过滤器或外部检测器提升，也要把过滤器版本和失败样例写入结果，而不能把管线收益归因给 checkpoint 本身。

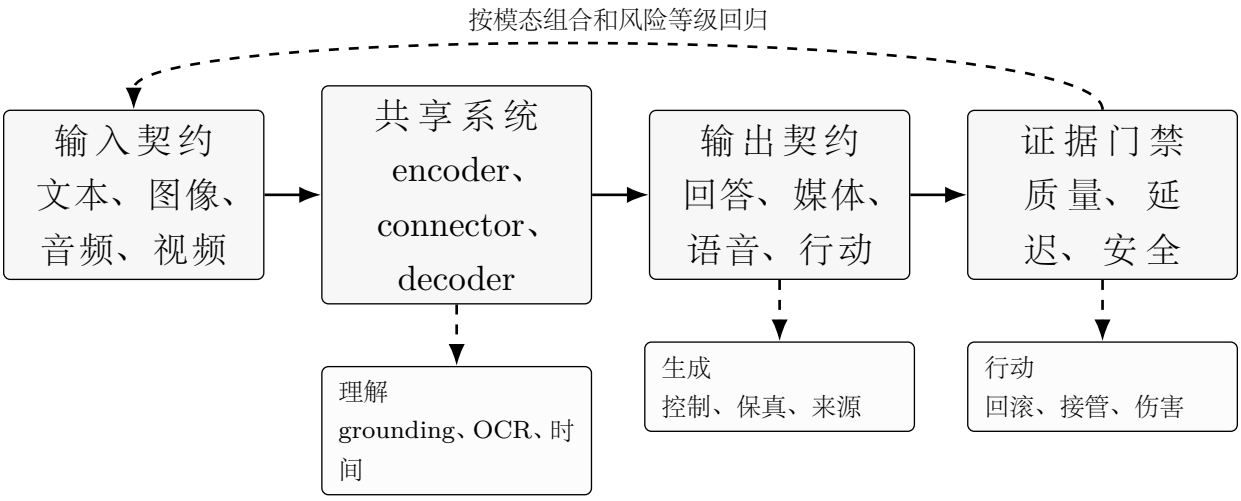

图 15.2: 统一多模态系统需要覆盖理解、生成、流式、来源和行动的回归矩阵，而不是只报告一个总分。该图综合 Janus-Pro、MMaDA、Sora、AR-Omni、Qwen3-Omni 和 OpenVLA 所暴露的设计压力 [20, 253, 179, 23, 249, 136]。

图 15.2 展示这种回归矩阵的结构：统一模型越强，越需要同时固定输入契约、共享组件、输出契约和证据门禁。

15.4 视频、音频与 Omni 模型

视频引入时间维度，音频引入采样、说话人、韵律和流式延迟。Qwen3-VL 报告了长上下文多模态输入、视频时间对齐和视觉推理能力 [203]。Qwen3-Omni 进一步把文本、图像、音频、视频理解和语音生成放进统一系统，并强调低延迟流式输出 [249]。

Omni 模型把接口问题推到更极端的位置。用户可能输入文本、截图、照片、视频、语音或它们的组合，并期待系统输出文本、语音、图像或行动。AR-Omni 代表一种更彻底的自回归方案：把文本生成、图像生成和流式语音生成放在单一 Transformer decoder 下，用统一 token 流处理任意到任意生成 [23]。这样的方案是否长期占优，不取决于概念是否优雅，而取决于它能否同时处理模态不平衡、实时性、视觉保真、语音自然度、安全策略和服务成本。

视频和音频还改变了交互协议。静态图像可以先编码再回答，实时语音却需要边听边判断是否该打断、续说或等待；视频问答不仅要知道答案，还要给出支持答案的时间戳。一个可比较的 omni 报告应拆分事件检测、时间定位、转写质量、说话人归属、跨模态指代、首字延迟、流式稳定性和最终推理。若系统使用外部 OCR、ASR、对象检测或检索工具，评测应把“模型本身能力”和“管线能力”分开。

视频样本还应记录采样协议。固定每秒一帧、按场景变化抽帧、使用视频 encoder 汇

聚特征，或把关键帧交给 LLM，测到的能力完全不同。音频样本也要说明 chunk 长度、重叠窗口、是否先转写、是否保留说话人和韵律，以及流式输出能否在用户打断后撤回或修正。没有这些字段，视频/音频 benchmark 分数无法解释，也无法估算真实服务中的延迟和成本。

官方 Video Processor 文档把视频预处理和图像预处理区分开：视频处理器要处理本地路径或 URL 解码、resize/normalize、批量转换和帧采样，采样可由 `do_sample_frames`、`num_frames` 或 `fps` 控制，并依赖视频 metadata 中的总帧数、帧率和时长 [114]。报告中应保存 decoder/backend、`video_preprocessing_config.json`、是否使用 fast processor、采样出来的帧索引、帧时间戳和模型预训练时的最大帧数。文档还指出，指定 `num_frames` 不必然保证输出刚好等于该帧数，因为模型特定 sampler 可能施加最小或最大限制；这类细节会直接影响时间 grounding 和服务成本。

Qwen3-Omni-MOE 的官方接口进一步说明，Omni 不是"多几个输入类型"这么简单：同一批对话可以包含 video、audio、text 和混合媒体，模板调用还带有从视频抽音频、是否使用视频内音频、`fps`、padding 等选项；图像分辨率可以通过像素上下限在质量和计算之间取舍，音频输出还受 audio-output 加载开关、`return_audio` 和 speaker 参数影响 [110]。因此 Omni 发布记录应把输入媒体路径、是否从视频抽音频、视频 fps、图像像素上下限、是否加载 talker/code2wav、语音 speaker、首包延迟、文本-only 回退和 GPU 显存差异都列出来；否则用户听到自然语音，并不能说明系统的音频理解、视频理解和语音合成都被同一证据支撑。

15.5 生成模型目标

自回归语言建模只是生成目标的一种。文本天然是序列，因此 next-token factorization 很自然；但图像、音频和视频可以被序列化，也可以通过扩散、流匹配或混合目标建模。Diffusion Transformer 用 transformer 替代传统 U-Net denoiser，在 latent patch 上做图像生成，并显示了可扩展性 [190]。Sora 把视觉数据压缩成 latent，再切成 spacetime patches，用 text-conditioned diffusion model 生成图像和视频 [179]。Stable Diffusion 3 相关工作使用 rectified flow transformer 进行高分辨率文本到图像合成，并强调文本 token 与图像 token 的双向交互 [36]。

目标族	适合的问题	成本表面	常见失败
Autoregressive	文本、代码、语音 token、工具轨迹	序列 decode 和 KV cache	空间顺序别扭，高分辨率媒体慢
Diffusion	图像/视频质量、编辑、多样性	多步去噪和 guidance	prompt 漂移，采样贵，依赖过滤器
Rectified flow	噪声到数据的连续流	步数、latent 分辨率	条件对齐弱时生成不忠实
Hybrid AR-diffusion	语义规划加连续合成	级联系统延迟	媒体漂亮但不服从指令

扩散模型常用去噪目标：给定噪声、时间步和条件，模型预测噪声或去噪方向，并最小化预测误差。Rectified flow 则学习从噪声端点到数据端点的速度场；训练时沿插值路径采样中间点，让模型预测从噪声到数据的速度。对本书而言，关键不是要求读者都去实现视频生成器，而是要理解：token 预测不是基础模型的全部。Dream 7B 这类离散扩散语言模型通过迭代去噪生成文本，提供任意顺序生成、填充和质量-速度权衡 [257]。MMaDA 则把统一扩散形式用于文本推理、多模态理解和文生图，并设计面向扩散基础模型的后训练方法 [253]。MammothModa2 代表混合 AR-Diffusion 路线：自回归路径做语义规划，扩散/flow 路径做高保真图像生成与编辑 [216]。因此，高水平教材不能只讲 decoder-only LLM，还必须解释生成目标、模态表示和服务接口如何共同塑造系统能力。

目标函数会直接改变服务和治理。自回归模型天然暴露 token 概率、支持流式输出，也容易接入工具轨迹；扩散和 flow 模型暴露的是步数、guidance、latent 分辨率、条件注入和后处理过滤；混合系统还要报告中间规划是否可审计、生成器是否忠实执行规划、失败时责任落在哪个模块。生成媒体的来源标记、内容凭证和水印也不应只放在末端过滤器里，而应进入训练数据、解码策略、日志和用户界面。

生成目标的审计字段应落到每次样本。文生图或文生视频结果至少要保存 prompt、negative prompt 或安全策略、随机种子、采样步数、guidance、latent 分辨率、模型和 VAE 版本、后处理过滤器、是否人工挑选、以及被拒绝候选数量。若报告只展示最终图像而没有这些字段，读者无法判断成功来自模型分布、提示词工程、人工选择，还是安全过滤器隐藏了失败样例。

15.6 行动、具身与世界模型

文本、图像、音频和视频之后，下一个边界是行动。Vision-language-action（VLA）模型把视觉观察和语言指令映射为动作，通常把机器人动作表示为 token，或者在多模态 backbone

后接动作头。RT-2 展示了一种直接路线：把视觉语言模型同时在网页规模视觉语言任务和机器人轨迹上 co-finetune，并把动作写成类似文本 token 的格式，使模型能把部分网页语义知识迁移到机器人控制中 [13]。OpenVLA 则把这条路线做成更开放的版本：一个 7B 开源 VLA，在真实机器人 demonstration 上训练，并提供面向新任务的微调和量化路径 [136]。

VLA 让“基础模型”这个词变得更具体，也更危险。文本生成错误可以编辑；机器人动作会改变世界。一个 VLA 报告必须说明观察频率、动作表示、控制时域、延迟、机器人形态、重置策略、安全联锁、teleoperation 数据、sim-to-real 迁移和失败恢复。一个模型能在静态图像中识别杯子，并不代表它能在遮挡、摩擦、相机标定误差和执行器限制下稳定抓起杯子。

VLA 行动模型常用行为克隆：给定观察历史、语言指令和可选任务记忆，最大化专家动作 token 或连续控制目标的 likelihood。若改用强化学习，则还要面对真实世界探索成本、安全联锁和重置代价。

用公式写，行为克隆可以看成最小化 $-\log \pi_\theta(a_t \mid o_{\le t}, u, h_t)$ 的平均损失，其中 $o_{\le t}$ 是观察历史，u 是语言指令，h_t 是任务记忆或环境状态，a_t 可以是离散动作 token，也可以是连续控制向量。这个式子看起来像语言建模，但部署语义完全不同：动作延迟、控制频率、夹爪状态、碰撞检测、人工接管和急停策略都会进入模型契约。离线成功率不足以证明可部署，报告还要给出失败恢复、人类干预次数和危险动作审查。

VLA 综述把这个方向定义为视觉感知、自然语言理解和具身控制的统一，并强调数据集、仿真平台、架构范式和真实部署挑战 [232]。对本书而言，合理范围不是展开成完整机器人教材，而是把行动视为另一种更严格的模态：它有更强的时序、安全和评测契约。世界模型也在这里连接生成与行动：视频生成器预测未来画面，具身模型想象抓取后果，规划器模拟工具调用结果，本质上都在用生成模型辅助决策。风险在于把视觉可信误认为物理真实；生成未来看起来连贯，并不代表它满足动力学、接触、物体恒常性和任务约束。

行动模型的日志粒度应高于普通对话。每次 rollout 至少要记录观察帧、语言指令、动作表示、控制频率、执行延迟、环境状态、重置原因、人工接管、碰撞或接近碰撞事件，以及失败恢复路径。离线 demonstration 的来源也要说明：操作者、场景、机器人形态、相机位置、任务成功定义和数据清洗规则都会影响策略。没有这些字段，VLA 分数很容易把数据覆盖、仿真便利和真实行动安全混在一起。

15.7 训练、模板与系统成本

实用 VLM 很少一次训练完成。常见路线是先选择或预训练视觉 encoder，再训练 connector，使视觉 embedding 对语言模型有意义；随后做多模态指令微调，补充 OCR、图表、文档、视觉数学、视频问答和多图比较；最后加入偏好、安全和拒答数据。阶段顺序很重

要：如果 connector 没有学会把视觉特征变成可用上下文，后续 SFT 只会学到“像在回答图片问题”的话术。

数据覆盖也要按失败模式设计。自然图像 caption 不能替代文档页、截图、手写字、表格、坐标轴、医学图像、遥感图像和低分辨率视频。一个发布数据说明应列出物体识别、属性、关系、计数、OCR、图表、空间定位、多图比较、时间戳定位、模糊输入下的拒答，以及图像改变安全决策的样例。合成数据能扩大覆盖，但也会把教师模型的幻觉和偏见带入训练集；人类标注昂贵，却对医学、法律、图表解释和主观质量判断很关键。

多模态数据管线的最小可审计记录应包括原始对话、格式化后的 prompt、`input_ids`、`image_token_index`、图像路径或哈希、`pixel_values` 形状、裁剪元数据、视觉 token 数、`attention_mask`、`position_ids` 和 labels。只保存自然语言样本不够，因为训练失败常发生在合并 embedding 之后。对于多图和多轮对话，还要记录每张图对应哪个用户轮次、图像在序列中的顺序、上一轮 KV cache 是否复用，以及视频帧或音频片段的时间戳。

字段	为什么要记录
模板与 token	复现 role marker、`<image>` 位置、截断和特殊 token 配置
图像预处理	复现 resize、padding、crop 顺序、分辨率和 patch 数
序列合并	检查视觉 embedding 替换占位符后的长度、mask、position 和 label 对齐
样本来源	审计许可证、敏感信息、合成教师和数据污染风险
批处理边界	防止 packed batch、padding 或多图样本互相泄漏注意力

Chat template 是多模态实现中最容易出错的部分之一。类似 `<image>` 的占位符不是普通词，而是外部视觉 embedding 的插入点。训练和推理必须使用同一套 role marker、image marker、crop 顺序、帧顺序和多图引用规则。合并序列后，视觉 embedding 参与注意力，却不应该作为文本 label 被预测；因此 attention mask、position ids、label mask、padding 和 packed batch 边界都要重新构造。很多“模型不看图”的故障，实际来自占位符错位、视觉特征插入错位或 label 跨过 image span。

需要特别区分模板格式化和真正的多模态特征构造。`apply_chat_template` 之类的函数通常只是把消息列表变成带 role marker 和图像占位符的字符串；它不会自动证明图像已经被正确编码，也不会替你完成 `pixel_values`、视觉 embedding、mask 和 labels 的一致性检查。LLaVA 风格系统中，文本里可能只有一个 `<image>` token，但一张图经过全局图和局部 crop 后会展开成成百上千个视觉 embedding。decoder-only LLM 看到的是扩展后的序列，训练 labels 应在视觉 span 和用户 prompt 上设为忽略，只在 assistant 文本上产生损失。

在当前 Transformers 用法中，调用 `apply_chat_template` 时若启用 `tokenize`、`return_dict` 和 tensor 返回，函数可能直接给出模型需要的张量字典，而不仅是字符串；

对图像-文本模型，返回值会同时包含文本 token 和预处理后的图像张量，Qwen3-VL 例子还会出现 `image_grid_thw` 这样的网格描述 [109, 111]。训练单元测试应断言：媒体条目数和占位符数一致，网格元数据和 `pixel_values` 批维一致，assistant 前的所有 user/media token 被 mask，生成输出展示前去掉历史 prompt，批处理里的多图、多视频和纯文本样本不会互相污染。

成本同样要模态化。若文本 prompt 长度为 L_{text}，视觉侧插入 M 个 token，则 LLM prefill 长度近似为 $L_{\text{text}} + M$。高分辨率切图、多页文档和视频帧会把 M 推到很高；音频还会加入 ASR、codec 或语音生成延迟。服务报告应拆分 encoder、connector、LLM prefill、decode、生成采样、后处理、显存和批处理影响。对交互产品，还要报告取消请求、流式 partial、缓存释放和多模态输入失败时的回退策略。

可复现的多模态服务实验应把每个请求拆成时间线。最小记录包括输入模态组合、原始媒体大小、预处理后的 tensor 形状、crop 或 frame 数、视觉/音频 token 数、encoder 时间、connector 时间、LLM prefill 时间、首 token 或首段语音延迟、decode 或生成采样时间、后处理时间、峰值显存、批大小、是否命中缓存、是否调用 OCR/ASR/检索工具，以及失败时返回的是拒答、降级文本回答还是重新请求上传。这样读者才能看出延迟来自视觉编码、长上下文 prefill、语音合成、视频采样还是安全过滤，而不是把所有成本都笼统归因给模型规模。

统一理解-生成系统还需要回归矩阵。若一次更新提高了图像生成质量，却让 OCR、图表读数、多图指代、视频时间定位、语音打断或视觉 prompt injection 拒答下降，那么它不是单纯的能力提升。较好的回归表会按输入模态、输出模态、任务类型和风险等级交叉列出样本数、指标、置信区间、人工复核比例和失败样例链接；生成侧还要记录 prompt、seed、采样步数、guidance、过滤器版本和来源凭证状态。这个矩阵把“统一”从产品口号变成可检查的系统约束。

15.8 多模态评测与安全

多模态评测应区分感知错误和推理错误。MMMU、MathVista、POPE、HallusionBench 和 MM-SafetyBench 分别覆盖专家知识、视觉数学、幻觉和安全风险 [259, 162, 150, 45, 159]。视觉 prompt injection、截图隐私、OCR 错误和敏感属性推断都需要单独测试。生成式多模态系统还要测试指令遵循、视觉保真、文字排版、对象关系、时间一致性、编辑局部性、多样性、安全过滤和来源标记。一个统一理解-生成模型不能只在生成任务上变强，却默默降低 OCR、文档问答、语音延迟或拒答边界。评测报告必须说明是否使用外部 OCR、ASR、对象检测、检索或浏览工具；否则我们无法判断分数来自模型本身，还是来自系统管线。

生成媒体的安全评测不能只看最终图片或视频是否被过滤。审计记录应覆盖训练数据

许可、提示词策略、负面词和安全分类器、采样参数、后处理、内容凭证、水印、用户可见标签、申诉路径和事故响应。对图像编辑任务，还要检查局部编辑是否破坏未编辑区域、身份或版权约束是否被绕过、被移除或新增的对象是否与文字说明一致。对视频和语音任务，还要检查时间一致性、说话人身份、配音误归属、字幕错配和深度伪造风险。只有把这些证据同感知和生成质量放在同一张发布表里，多模态系统才不会把“看起来可用”误当成“可以发布”。

来源治理还要处理信号缺失。水印、metadata 和 content credentials 可以被裁剪、转码、平台剥离或二次编辑破坏；检测器又会在新模型和新压缩格式上漂移。因此发布策略不应把“检测为生成内容”当作唯一防线，而应组合签名来源、用户可见标签、平台日志、举报入口、人工复核和事故响应，并明确当来源链断裂时系统如何降级或拒绝高风险用途。

视频生成和视频理解需要独立维度。VBench++ 把视频质量拆成时间闪烁、主体一致性、运动平滑、空间关系、文生视频、图生视频和可信度等切片 [54]；T2VPhysBench 关注生成视频是否遵守基本物理约束 [46]；TOC-Bench 则从视频理解侧检查对象在遮挡、消失、重现和交互过程中的时间一致性 [18]。这些 benchmark 不能互相替代：画面漂亮、物理可信、对象身份稳定和回答可定位是不同能力。报告应给出每类失败的样例，而不是只给平均分。

安全也要按通道拆开。视觉 prompt injection 是把恶意指令藏在图片、截图、扫描文档、二维码或视频帧里；系统应把这些文本视为不可信数据，而不是更高优先级的指令。输入隐私需要最小化采集、脱敏、访问控制和保留期限，因为图片和音频比普通文本更容易暴露身份、位置、屏幕内容和旁观者信息。生成侧则要区分版权、肖像、欺骗性媒体、医学误导、政治广告和儿童安全等风险，并把来源标记和审计日志纳入产品流程。

CLIP 和 VLM 的模型卡边界也应进入安全章节。图文模型常在网络图文对上训练，数据人群、语言、地区和拍摄场景并不均衡；零样本分类的类别命名还可能把社会身份、职业、年龄或外貌属性硬塞进标签空间。用于人脸识别、监控、敏感属性推断或高风险分流时，不能把研究模型的检索准确率当成部署证据。安全评测应包含类别 taxonomy 审查、非英语输入、受保护属性切片、拒答边界和人工复核路径。

15.9 深入展开：多模态和生成模型改变接口边界

本章已经从“多模态 LLM”扩展为“多模态与生成式基础模型”。传统 VLM 关注把图像接入语言模型：视觉编码器提取 patch 或 frame 特征，connector 映射到语言模型空间，语言模型生成文本。Projection、Q-Former、cross-attention 和 end-to-end training 各有成本和风险。文档、图表和 OCR 任务尤其会暴露 token 压缩和分辨率限制。统一理解与生成进一步打破边界。Janus-Pro 说明理解图像和生成图像可能需要不同表示；MMaDA 和 Dream

7B 表明扩散形式也可以用于文本和多模态；Sora、DiT 和 Rectified Flow 说明高维视觉生成并不一定服从 next-token 预测；AR-Omni 和 Omni 系统尝试把文本、图像、音频、视频和语音放进统一接口。本章把这些方法放进同一问题：选择什么表示、什么目标函数、什么服务接口、什么评测证据。

行动模型把多模态推向现实环境。RT-2 和 OpenVLA 将视觉语言模型连接到机器人动作，说明动作也可以被 token 化或由多模态 backbone 产生。但机器人动作不是文本输出：它有控制频率、执行器限制、延迟、碰撞、重置、真实失败和人类安全问题。VLA 评测必须报告任务成功、危险动作、人类干预、sim-to-real gap 和恢复策略。

视频和世界模型则连接生成与预测。一个视频生成器可以产生看似合理的未来，但物理一致性、对象恒常性和因果关系可能错误。世界模型如果用于规划，必须证明它的预测能改善真实决策，而不是只生成好看的画面。因此把 temporal grounding、physical consistency、content provenance 和 safety filtering 放进同一章讨论。

15.10 章节细节

15.10.1 模态作为接口

模态不是附加功能，而是模型输入输出接口的扩展。图像、音频、视频、语音和动作都需要被编码成模型可处理的表示。统一接口的目标是让不同模态在同一任务语境中协作。

15.10.2 图文对齐

CLIP 类对比学习把图像和文本拉到共同空间，使零样本分类和检索成为可能。它学习的是跨模态相似度，不是完整生成能力。对比预训练常作为多模态系统的视觉基础。

15.10.3 零样本分类

零样本分类把类别写成文本提示，比较图像和文本 embedding。它显示自然语言可以作为视觉任务接口。局限是提示敏感、类别粒度和数据偏差。

15.10.4 CLIP 实现契约

CLIP 的可复现性来自预处理、tokenizer、特征归一化、logit scale、类别 prompt 和批内负例的共同约束。改变裁剪或截断策略，等价于改变任务协议。部署前还要检查类别 taxonomy、语言覆盖和敏感用途边界。

15.10.5 冻结编码器与投影层

许多 VLM 冻结视觉编码器和语言模型，只训练投影层把视觉特征接到文本 token 空间。这种方法便宜、稳定，但视觉信息压缩会限制细粒度感知。是否端到端训练取决于数据和算力。

15.10.6 Query Transformer 与 Token 压缩

Q-Former 或 resampler 用少量查询 token 压缩视觉特征，减少 LLM 上下文负担。压缩提高效率，但可能丢失细节。OCR、图表和文档任务常常需要更高分辨率或专门结构。

15.10.7 Cross-Attention 与交错媒体

Cross-attention 让文本层直接读取视觉特征，交错媒体让图文序列混合出现。它们更灵活，也更复杂。训练数据必须覆盖多轮、多图、视频帧和指代关系。

15.10.8 架构取舍

多模态架构在冻结与端到端、投影与 cross-attention、视觉 token 数与成本、统一模型与专用模块之间取舍。没有一种结构适合所有任务。评测应覆盖感知、推理、生成和安全。

15.10.9 统一理解与生成

统一理解-生成模型试图同时处理识别、问答、编辑和生成。它们需要把离散语言 token、连续视觉特征和生成模型 latent 协调起来。关键挑战是接口统一、训练稳定和评测一致。

15.10.10 生成建模目标

多模态生成可用自回归、扩散、flow matching 或混合目标。自回归适合离散序列，扩散和流匹配常用于图像、音频和视频。目标函数决定采样过程、可控性和服务成本。

15.10.11 自回归生成

自回归图像或视频生成把视觉表示离散化或按序列预测。它与语言模型接口相似，但序列更长、局部相关更强。成本和长程一致性是主要挑战。

15.10.12 扩散与 Rectified Flow

扩散模型从噪声逐步去噪，rectified flow 学习更直接的流路径。它们在图像和视频生成中非常重要。采样步数、条件控制和安全过滤直接影响服务体验。

15.10.13 混合 AR-Diffusion 系统

许多系统用 LLM 处理指令和规划，再用扩散或流模型生成视觉内容。LLM 提供语义控制，生成模型提供高质量像素。系统评测必须看文本遵循、视觉质量、一致性和安全。

15.10.14 行动、具身与世界模型

Vision-language-action 模型把感知、语言和动作连接起来。它们需要处理实时观察、动作空间、环境反馈和安全约束。世界模型试图学习环境动态，但不能只用视频逼真度评估。

15.10.15 多模态指令微调

多模态 SFT 把图像、视频、音频和文本对话组织成统一模板。训练阶段通常包括连接器预训练、指令微调和偏好/安全调优。数据质量决定模型是否真正理解视觉内容。

15.10.16 训练阶段

第一阶段常让视觉特征对齐语言空间，第二阶段学习多模态指令，后续阶段优化偏好、安全和工具使用。每一阶段的冻结策略、学习率和数据混合都不同。跳过阶段可能降低稳定性。

15.10.17 数据来源

数据来自图文对、caption、OCR、VQA、文档、图表、视频问答、语音和合成数据。不同来源带来不同偏差和许可证问题。敏感图像、人脸、位置和医疗图像尤其需要治理。

15.10.18 带图像的 Chat Template

多模态模板要标明图像位置、图像数量、帧顺序、工具结果和回答边界。图像占位符不是普通词，它连接外部编码器输出。模板不一致会导致模型忽略图像或混淆多图指代。

15.10.19 数据 Collator 与 Mask

多模态 collator 不只是 padding 文本。它要把图像 crop、视觉 embedding、文本 token、position ids、attention mask 和 labels 合并到同一个样本协议中。视觉 span 通常只作为条件参与注意力，不能被当成 assistant label 预测。

15.10.20 图像占位符、切图与 Label Mask

类似 `<image>` 的 token 是占位符，不是普通词。LLaVA 风格系统会先把文本 prompt tokenize，记录图像占位符的位置；再把图像预处理成一个或多个固定尺寸 tensor，送入 vision encoder；随后通过 projector 得到视觉 embedding，并在语言模型输入中用这些视觉 embedding 替换占位符位置 [156]。替换完成后，LLM 看到的序列长度不再等于原始文本 token 长度。高分辨率图像会让这个扩展非常明显。为了避免非正方形图像被强行拉伸，系统可能先等比例缩放、padding，再切成多个 crop，同时保留全局图。若 ViT patch size 为 14，一个 336×336 crop 会产生 $24 \times 24 = 576$ 个 patch 特征；如果一张图被处理成一个全局图加四个局部 crop，未压缩前就可能产生两千级别的视觉 token。于是单轮单图对话也会变成长上下文问题。合并多模态序列后，attention mask、position ids 和 labels 都要重建。视觉 embedding 应作为条件参与注意力，但它们不是要被语言模型预测的文本标签；图像 padding、文本 padding、crop 顺序、多图引用和上一轮对话必须在 batching 前对齐。很多“模型不看图”的问题，其实是序列化错误：占位符位置错、视觉特征插入错、label 跨过 image span 错位，或 packed batch 中一个样本看到了另一个样本。

15.10.21 OCR、图表与文档

OCR 和文档理解要求高分辨率、布局感知和精确引用。图表任务需要读数值、坐标轴、图例和趋势。普通图像问答能力不足以证明模型能处理业务文档。

15.10.22 系统成本

视觉 token、视频帧、音频片段和生成步数会显著增加成本。多模态服务的延迟包括编码、LLM prefill、decode、生成器采样和后处理。成本报告应按模态拆分。

15.10.23 视觉 Token 与 Prefill

一张高分辨率图像可能转成大量视觉 token，拉高 prefill 时间和 KV cache。压缩 token 可以降成本，但可能损失细节。系统需要按任务动态选择分辨率和帧数。

15.10.24 视频与音频

视频加入时间维度，音频加入连续信号和实时性要求。帧采样、时间对齐、说话人分离和延迟预算都会影响质量。实时语音对话还要处理打断、流式 ASR 和 TTS。

15.10.25 Benchmark 家族

多模态 benchmark 覆盖 VQA、OCR、图表、文档、视频、空间推理和生成质量。很多 benchmark 容易被污染或饱和。真实任务评测应加入私有文档、交互和错误审查。

15.10.26 评测提醒

多模态模型可能用语言先验猜答案，而不看图像。评测应包含反事实图像、文字覆盖、干扰对象、多图引用和需要精确读数的样例。生成模型还要评估可控性、一致性和安全。

15.10.27 安全与治理

多模态系统会产生隐私、身份、地点、医学、版权和视觉 prompt injection 风险。输入图像可能携带恶意指令，输出图像可能侵犯权利或造成误导。治理必须覆盖输入、模型、工具和生成内容。

15.10.28 视觉 Prompt Injection

视觉 prompt injection 把恶意指令嵌入图片、网页截图、文档扫描或视频帧中，诱导模型忽略系统策略、泄露上下文或越权调用工具。防御不能只依赖提示词警告，而应在架构上把图像和检索文档都视为不可信数据；工具权限、密钥访问、外部请求和文件写入必须由运行时强制控制。

15.10.29 隐私与敏感推断

多模态模型可能从人脸、地理位置、医疗影像、身份证件、屏幕截图和家庭环境中推断敏感信息。即使用户没有显式询问，模型也可能在描述中暴露身份、健康、位置或财务线索。发布系统需要输入最小化、日志脱敏、访问控制、保留期限和敏感类别拒答策略。

15.10.30 幻觉与 Grounding

多模态幻觉表现为看见不存在的物体、误读图表数值、把语言先验当作视觉证据，或在视频中错误归因动作。Grounding 评测应要求模型把回答连接到图像区域、文档文本、帧

时间或引用证据。对 OCR、图表、医学和遥感等场景，流畅描述不能替代可核查定位。

15.11 关键术语、实现要点与练习

关键术语。Vision encoder 把图像或视频映射成特征；connector 把模态特征接入 LLM；unified understanding-generation model 同时理解和生成媒体；diffusion model 通过去噪生成；VLA model 根据观察和语言产生动作；world model 预测未来状态或行动后果。

实现要点。多模态报告应说明分辨率、视觉 token 数、帧采样、OCR/ASR 工具、encoder latency、prefill/decode 成本和安全过滤；Processor 层应保存 multimodal message JSON、processor class、image/video preprocessing config、`pixel_values` 形状、`image_grid_thw`/`video_grid_thw`、`num_frames`/`fps`、音频输出开关和 label mask 单元测试；生成模型应说明 objective、latent/token 表示、guidance、step schedule 和 provenance；VLA 应说明动作空间、控制频率和安全联锁。

练习。

1. 比较 projection、Q-Former 和 cross-attention connector。
2. 推导三组图文对的对称 CLIP 损失，指出每个图像和文本方向上的正例与负例。
3. 估算一张 1024×899 图像被处理成一个全局 336×336 图和四个局部 crop 时的视觉 token 数，并说明插入 chat template 后 attention mask 和 label mask 如何变化。
4. 设计一个文档 VQA 评测，区分 OCR、布局和推理错误。
5. 比较 autoregressive、diffusion 和 AR-diffusion 的生成取舍。
6. 为桌面机器人设计 VLA 评测，包含 reset、危险动作和人类干预。
7. 为文档问答系统比较“多视觉 token 的 MLP projection”和“压缩 token 的 Q-Former”，写出成本和准确率风险。
8. 设计一个视频评测小集，分别覆盖时间定位、对象恒常性、物理一致性和字幕/语音对齐。
9. 为一个 omni 助手写运行卡，列出文本、图像、音频、视频输入和语音输出的延迟、安全、日志与回退策略。
10. 审计一个多模态 processor batch，打印 `input_ids`、`pixel_values`、`image_grid_thw`、`video_grid_thw`、`num_frames` 和 labels，确认媒体占位符、视觉 span 和 assistant loss mask 一致。

15.12 结构化检查表

15.12.1 多模态系统报告清单

字段	内容
输入处理	分辨率、裁剪、帧采样、音频切片、OCR/ASR 工具
表示接口	vision encoder、connector、visual tokens、image marker、position policy、grid metadata
连接器取舍	projection、Q-Former/resampler、cross-attention、端到端训练与 token 压缩风险
训练数据	图文对、OCR、图表、文档、视频、语音、安全拒答、合成数据来源
生成目标	autoregressive、diffusion、rectified flow、hybrid AR-diffusion
服务成本	encoder latency、prefill、decode、生成步数、显存、流式延迟
评测切片	感知、OCR、图表、空间关系、视频时间、生成质量、行动、安全
来源治理	metadata、watermark、content credentials、过滤器、日志
行动系统	observation、action space、control frequency、reset、安全联锁
Processor 证据	message JSON、processor config、`pixel_values`、帧索引、label mask、音频输出开关

第十六章　评测、安全与治理

16.1　评测是测量系统

评测从问题开始：要支持什么决策？客服助手、代码 agent、医疗信息助手和教育系统需要不同指标。MMLU、HELM、GPQA、SWE-bench、HLE 和 BrowseComp 提供公共参照，但每个 benchmark 都有构念边界 [48, 151, 209, 131, 192, 180]。长上下文、Agent 和生成式多模态系统会在传统短问答 benchmark 之外失败。长上下文评测应测试有效上下文利用，而不是只看最大输入长度；需要报告位置敏感性、干扰鲁棒性、答案证据和每个成功任务的成本 [265, 52, 10]。Agent 评测还要把模型、工具运行时、状态管理、重试策略、沙盒和预算分开看。SWE-bench 和 BrowseComp 这类评测之所以重要，是因为它们测量的不是单句语言能力，而是一个模型-工具-环境组合在约束下完成任务的能力 [131, 180]。

一个可发布的评测不应只写“某 benchmark 得分”。它至少要包含六个字段：构念、样本人群、协议、指标、不确定性和决策规则。构念说明到底测什么，例如事实精度、工具成功率、拒答准确率或长文档证据整合；样本人群说明这些 prompt 代表哪些用户、语言、领域和风险类别；协议说明 system prompt、chat template、工具权限、检索预算、解码参数和时间限制；指标说明 exact match、pass@k、rubric 分、win rate、校准误差、成本或延迟；不确定性说明样本数、置信区间、bootstrap 或标注一致性；决策规则说明什么结果触发发布、回滚、继续训练或人工复核。没有最后一项，评测只是展示数字，不是工程决策。

评测还应保存逐样本证据，而不是只保存均值。每条样本至少记录模型版本、prompt 模板、输入、检索上下文、工具权限、原始输出、解析结果、得分、异常、运行时、成本和数据版本。这样当分数下降时，团队可以区分解析失败、拒答、幻觉、工具超时、检索缺失、答案归一化错误和真实能力退化。出版级教材应让读者习惯把 evaluation harness 当作代码和数据 artifact，而不是把评测写成一句“我们在某榜单上测试”。

评测卡还要绑定发布门禁。一个结果如果不能回答“继续训练、灰度发布、回滚、加入人工审核还是禁止上线”，就仍然是研究记录，不是发布证据。门禁应提前写明基线模型、主指标、回归指标、最低样本量、置信区间、失败切片和复核负责人。上线后发现新失败时，样例应回流到同一 harness，而不是另起一套不可比较的临时测试。

评测实现还要区分“指标库能跑”和“发布证据可复现”。很多训练脚本可以临时调用 ac-

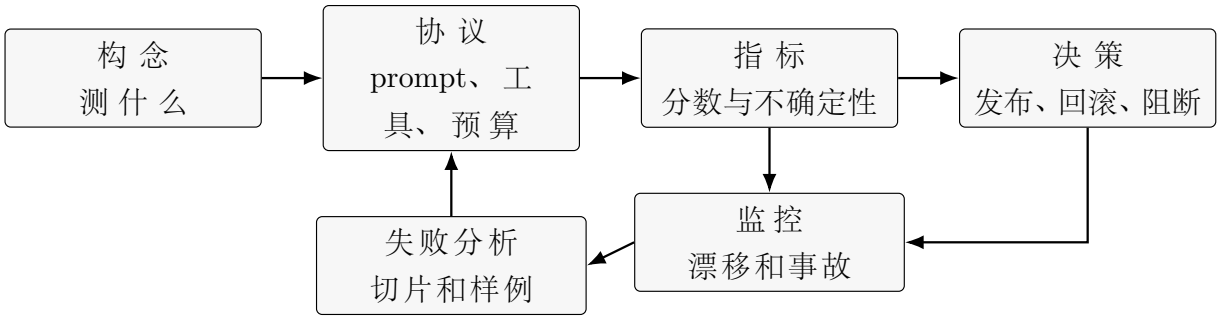

图 16.1: 评测是发布证据闭环。没有协议和不确定性的分数，不能支撑系统发布。

curacy、perplexity 或模型裁判，但如果没有固定数据 split、样本 hash、生成参数、答案抽取规则、异常处理、版本化输出目录和逐样本记录，这个数字就不能进入发布说明。更稳妥的做法是把评测输出写成 JSONL 或表格 artifact：每行包含 sample id、原始输入、模型输出、解析答案、gold、score、错误类别、耗时、费用、随机种子、模型和 tokenizer 版本。这样 simple evals、Trainer 回调、Hugging Face Evaluate 风格指标或自写脚本都可以接入同一审计层，而不会把一个临时 notebook 结果误当成 release gate。

生成式多模态评测需要另一套证据。文生图或文生视频模型应评估指令遵循、视觉质量、文字排版、对象关系、时间一致性、编辑局部性、多样性、安全过滤和来源标记。统一理解-生成系统还需要跨任务回归检查：提升图像生成不能以牺牲 OCR、文档问答、语音延迟或安全拒答为代价。Janus-Pro、MMaDA、Sora 风格视频生成和 AR-Omni 都说明，一个平均分无法代表既能理解又能创造媒体的系统 [20, 253, 179, 23]。

视频尤其需要单独评测。VBench++ 把视频生成质量拆成时间闪烁、主体一致性、运动平滑度、空间关系、文生视频、图生视频和可信度等细粒度维度 [54]。T2VPhysBench 进一步追问生成视频是否遵守基本物理规律，而不只是看起来漂亮 [46]。视频理解也有类似问题：TOC-Bench 专门评估 temporal object consistency，要求模型在遮挡、消失、再出现、状态变化和对象交互中保持同一对象的身份和状态 [18]。因此，一个视频系统可能画面质量很高、物理一致性很弱，同时在对象中心时序推理上仍然失败。

具身和行动系统还要承担真实世界责任。VLA 评测应报告任务成功率、失败恢复、危险动作率、人类干预率、延迟、跨机器人形态迁移，以及测试任务是否和训练数据共享物体、场景、操作者或 teleoperation policy。仿真可以加速评测，但如果不测量 sim-to-real gap，就不能替代真实部署证据。机器人场景中，即使平均失败率低，罕见失败也可能造成财产损失或人身风险。图 16.1 概括了本章使用的评测证据闭环。

16.2 人类评测与模型裁判

人类评测需要明确 rubric、标注人资格、一致性、置信区间和失败样例。模型裁判可以扩大规模，但会继承 judge 模型偏差、位置偏差、长度偏差和脆弱性 [268, 160]。高风险发布不能只依赖模型裁判。

人类偏好评测通常比单一打分稳定，但仍然需要严谨协议。报告一个 55% win rate 时，应说明比较了多少 prompt、每个 prompt 是否随机交换答案顺序、是否允许平局、标注者是否看见模型身份、rubric 如何处理事实性和风格冲突，以及置信区间是否跨过 50%。如果标注者意见分歧很大，平均偏好不能直接转化为发布结论；分歧样例本身就是需要分析的失败模式。

模型裁判适合开发迭代，但必须被校准。常见偏差包括偏好更长答案、偏好更像自己训练分布的文风、偏好先出现或后出现的候选、把流畅性当作正确性、以及把 judge 自身安全政策误当作产品政策。更可靠的做法是答案顺序随机化、长度归一化、使用多个 judge、保留 judge 解释、抽样做人类复核，并用故意构造的错误答案测试 judge 是否真的看事实。模型裁判的输出应被看作一个带偏差的测量仪器，而不是客观真理。

人评和模型裁判都应留下审计字段。人评报告要记录 annotator 来源、培训样例、rubric 版本、冲突仲裁、样本抽样方式和报酬结构；模型裁判报告要记录 judge 模型版本、系统提示、候选顺序随机化、长度控制、是否看见参考答案、是否看见工具输出、平局规则和人类抽检比例。缺少这些字段时，win rate 很难复现，也无法判断模型偏好是否只是评分器偏好。

模型裁判进入训练或 RL 奖励时，风险更高。一个数学奖励可以先用确定性解析器验证 boxed answer，再在无法解析时退回模型判等；一个医疗文本奖励也可能让远程 LLM 输出 0/1 相似度。这样的设计适合研究迭代，但发布报告必须说明判等器是否可重复、远程模型版本是否固定、API 失败时是否 fallback 到字符串匹配、错误答案是否被人工抽检，以及 judge 的错误会不会被强化学习放大。否则模型看似在优化“正确性”，实际上可能在优化 judge 的格式偏好或 API 偶然行为。

16.3 治理

治理包括模型卡、数据卡、系统卡、风险框架、红队、监控、事件响应和回滚。NIST AI RMF、NIST GenAI Profile、EU AI Act、OpenAI Preparedness Framework、OpenAI Frontier Governance Framework 和 Anthropic Responsible Scaling Policy 都说明：前沿模型发布需要把能力、风险、缓解措施和残余不确定性写成可检查文档 [176, 177, 38, 182, 187, 6]。

治理审查应把发布对象和证据绑定在一起。一个客服 RAG 助手的发布记录可以要求：私有升级集检索召回率超过阈值；带引用声明的事实精度达到阈值；政策探针上的 unsafe

compliance 低于阈值；良性敏感请求的 false refusal 低于阈值；p95 延迟和单位成本满足产品约束；所有阈值都带置信区间。若二分类指标有 k 个失败、n 个样本，报告至少应给出 $\hat{p} = k/n$ 和抽样不确定性；低失败率场景通常需要 exact 或 bootstrap 区间，而不能只用一个点估计。样本太少时，“没有发现失败”不是“失败率足够低”。

发布记录还应包含可操作责任：模型版本、tokenizer、chat template、检索索引、工具权限、安全政策版本、日志保留期、人工复核队列、事故分级、回滚触发器和用户沟通路径。隐私和版权审查应发生在数据进入训练或检索层之前；如果一个来源不能撤回、审计、授权或访问控制，就不应被静默写入权重更新。治理不是发布后的公关说明，而是决定哪些证据足以支撑上线的工程流程。

系统卡应把模型、工具、检索、过滤器和监控写成一个系统，而不只是介绍 base model。对 RAG、agent 或多模态应用，系统卡至少要说明外部工具权限、索引更新节奏、内容过滤策略、失败升级路径、用户可见限制、已知高风险场景和回滚开关。风险登记表则应把每个风险连接到证据：风险描述、影响人群、触发条件、评测切片、缓解措施、残余不确定性、负责人和复查日期。

把框架转成工程材料时，最有用的是维护一张 governance crosswalk。NIST AI RMF 的 govern、map、measure、manage 可以对应负责人、用途和受影响人群、评测卡与红队、缓解和事故响应；NIST GenAI Profile 进一步提示生成式系统需要覆盖 CBRN、confabulation、data privacy、harmful bias and homogenization、information integrity、information security、intellectual property、obscene or abusive content、value-chain integration 等风险类别 [176, 177]。这些类别不是要逐字复制进产品文案，而是要驱动样本切片、红队主题、日志字段和发布门禁。

EU AI Act 采用风险分层，并对通用目的 AI 模型（general-purpose AI model, GPAI）提出模型层义务。对所有 GPAI provider，公开资料强调技术文档、版权政策和训练内容摘要；对具有 systemic risk 的 GPAI model，还强调通知、风险评估与缓解、事件报告和网络安全保护 [38]。工程团队应把这些义务映射到模型版本、训练和测试记录、下游集成说明、训练数据摘要、严重事件流程和安全控制，而不是把合规理解成上线后的单页声明。

前沿模型实验室的公开框架展示了另一类证据结构。OpenAI 的 Frontier Governance Framework 把 systemic risk 评估组织到 cyber offense、CBRN、harmful manipulation 和 loss of control 等类别，并把风险识别、分析、残余风险接受、安全缓解、事件响应、模型报告、外部专家输入和框架更新放在同一链条中 [187]。Anthropic RSP 的 2026 更新则强调 Frontier Safety Roadmap、Risk Reports、外部评审和能力到缓解措施的映射 [6]。这些公司框架不是通用答案，但它们提示读者：发布证据要说明评测如何触发决策，决策如何触发缓解，缓解如何被持续验证。

图 16.2 把这些框架转成发布工程材料：范围、评测卡、风险登记、控制措施和监控回

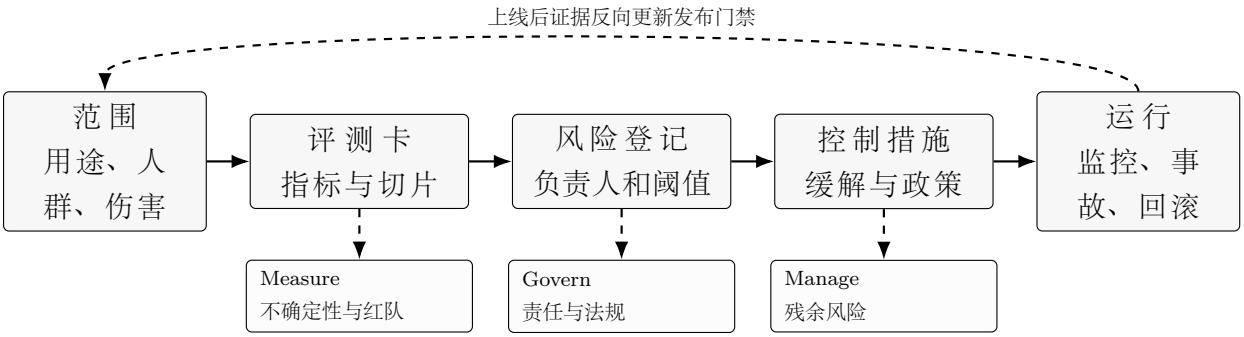

图 16.2: 治理只有映射到发布 artifact、负责人、控制、监控和回滚路径时才可执行。该图综合 NIST AI RMF、NIST GenAI Profile、EU AI Act、OpenAI 治理框架和 Anthropic RSP [176, 177, 38, 182, 187, 6]。

滚必须互相引用。

16.4 可解释性、遗忘与水印

治理需要技术抓手，但任何抓手都不能被夸大。机制可解释性尝试识别模型内部的特征、回路或因果路径。Anthropic 关于 monosemantic features 和 circuit tracing 的工作表明，较大模型中的某些计算结构可以被部分解释，但这仍然不是完整安全证明 [228, 4]。可解释性证据必须和干预实验、反例和下游行为测试一起使用。

Unlearning 关注另一个问题：模型训练后，如何让它停止产生或依赖特定知识、隐私数据、版权材料或危险能力。LLM unlearning 综述强调，删除声明必须同时评估保留能力、抽取抵抗、重新学习风险和连带损伤 [202]。一个模型不再回答某个记忆化 prompt，并不表示它不会通过改写、检索或微调泄露同一信息。

水印和来源工具试图识别生成内容或追踪模型输出。文本水印可以通过改变采样分布留下可检测统计信号 [139]。对多模态生成而言，来源还包括元数据、内容凭证、可见标记、模型卡和平台政策。水印可能被改写、裁剪、转码或分布漂移破坏，所以它只能作为证据层之一，而不是解决虚假信息、版权或问责问题的完整方案。

合成内容透明度需要分层处理。NIST 关于 synthetic content transparency 的报告把认证、来源追踪、标签、水印、检测、测试和检查视为互补技术，而不是单一解决方案 [17]。C2PA 规范则提供了内容凭证和媒体来源历史的技术标准，用签名元数据记录资产来源、编辑和生成过程 [25]。这些工具必要但有限：元数据可能被剥离，视觉水印可能被裁剪，统计水印可能被改写削弱，生成资产被多次 remix 后来源链也可能变得含糊。治理策略应把模型侧水印、签名内容凭证、可见标签、平台检测、用户举报和事故响应组合起来，并明确当来源信号缺失或冲突时系统如何处理。

16.5 深入展开：评测是发布证据

本章把评测、安全和治理合并，是因为它们在发布决策中不可分割。一个评测卡应说明构念、人群、协议、指标、不确定性和决策规则。MMLU、HELM、GPQA、SWE-bench、HLE 和 BrowseComp 都只是特定构念的测量工具，不能直接等同于“模型好”。人类评测需要 rubric、标注者资格、校准、一致性和置信区间。模型裁判可以降低成本，但有位置偏差、长度偏差、自我偏好、风格偏差和政策偏差。本章建议用答案顺序随机化、长度控制、裁判集成、人类校准和对抗裁判测试缓解，但高风险发布不能只依赖模型裁判。长上下文、agent 和生成式系统需要额外协议。长上下文要测位置敏感性、聚合、多跳、矛盾和 distractor；agent 要测工具权限、重试、环境状态和 rollback；视频生成要测时序一致性、物理一致性和对象身份；VLA 要测危险动作率和人类干预。一个平均分无法覆盖这些系统级行为。治理把评测变成可检查 artifact。Model cards、datasheets、system cards、NIST AI RMF、NIST GenAI Profile、EU AI Act、Preparedness Framework、Frontier Governance Framework 和 RSP 都在要求团队说明用途、限制、数据、风险、缓解、监控和回滚。可解释性、unlearning、水印和内容凭证都是技术证据层，但都不是单独充分条件。发布决策必须承认不确定性，并说明事故发生时如何发现、响应和撤回。

16.6 章节细节

16.6.1 评测作为测量系统

评测是把问题、样本、指标、评分器和决策连接起来的测量系统。一个分数只有在知道测量对象和误差来源时才有意义。发布决策需要证据包，而不是单个排行榜数字。

16.6.2 从问题到决策

评测前应先明确要支持什么决策：是否训练继续、是否发布、是否替换模型、是否进入高风险场景。不同决策需要不同严格度。研究 benchmark 和生产发布评测不能混用。

16.6.3 测量模板

测量模板应记录任务定义、样本来源、时间范围、切分方法、评分规则、置信区间、失败分类和复现脚本。没有模板，评测结果难以比较。模板也能防止事后挑指标。

16.6.4 评测 Harness 即代码

一个 benchmark 名字不是可运行评测。OpenAI simple evals 这类实现把评测拆成采样器、样本格式、prompt 模板、答案抽取、评分函数和汇总规则 [178]。同一个模型输出，如果用选择题字母抽取、数学最终答案解析、LLM 判等或代码执行测试，得到的是不同测量协议。实现细节本身就是评测协议。MMLU 类选择题要处理选项顺序和最终答案格式 [48]；MATH 类题目常要求规范化最终答案，例如解析方框答案中的结果 [49]；GPQA 类题目需要打乱选项以减少捷径 [209]；DROP 类阅读理解要处理数字、日期、短语和多答案归一化 [34]；MGSM 类任务用多语言 few-shot 或 CoT 检查推理是否跨语言迁移 [217]；HumanEval 类代码评测必须在沙盒、超时和资源限制下运行候选程序，否则评测器本身会成为安全风险 [19]。

Lighteval 的最新文档把这个契约进一步工程化：同一任务可以在 eval/Inspect、Transformers 加 Accelerate、Nanotron、vLLM、SGLang、TGI、Inference Endpoints、LiteLLM、Inference Providers 或自定义模型后端上运行，并强调逐样本结果、Hub、S3 或本地保存，以及模型仓库 PR 中的评测结果展示 [80, 82]。因此 backend 不是运行脚本的附属信息，而是测量协议的一部分。报告应记录后端、模型参数、精度、并行方式、解码设置、推理标签移除规则和自定义前后处理；否则同一个 benchmark 分数无法解释，也难以复现。

任务配置同样应被当作 artifact。一个 Lighteval 风格的任务至少要说明任务名、prompt 函数、数据仓库、子集、数据 revision、评测 split、few-shot split、生成长度、停止序列、语法约束、采样数、重复样本处理、原始样本数、有效样本数和指标列表 [84]。指标层还要区分逐样本函数和语料级汇总函数；前者处理单条文档和模型响应，后者把全数据集结果聚合成最终分数 [81]。如果使用模型裁判，还要固定裁判模型、模板、响应解析、裁判后端、响应格式、最大输出长度和服务提供方，避免把裁判漂移误认为被测模型变化。

可审计结果格式决定评测能否进入发布材料。Lighteval 的 EvaluationTracker 会保存主结果 JSON 和逐样本细节文件，细节记录包含原始文档、模型响应和指标字段；结果也可以推送到 Hugging Face Hub，默认私有，公开运行需要显式设置 [83]。这类结果结构提醒团队：release gate 不应只收一列均值，而要收可追溯的任务配置、模型配置、版本、样本级失败和聚合规则。只有这样，榜单分数才能被转化为可调试、可复核、可签字的工程证据。

选择题打分尤其能说明 harness 的影响。第一种做法让模型生成最终字母，再对解析出的 `A/B/C/D` 做 exact match；它测到 prompt-following 与解析链路，但模型用别的形式说出正确答案时可能被判错。第二种做法读取下一 token logits，只在候选字母上归一化；它避免非法输出，但依赖 tokenizer 和字母先验。第三种做法把每个候选答案接到题干后，按条件负对数似然或困惑度打分；用答案文本而不是字母能减少 label bias，但测量对象也从生成行为变成条件 likelihood。报告应写明采用哪条路径、是否打乱选项、非法输出如何

处理，以及是否用了 few-shot、CoT、pass@N 或 majority vote。

每次评测都应保存可审计记录：模型版本、prompt 模板、解码参数、工具权限、检索上下文、原始输出、抽取答案、分数、异常、运行时、成本和数据版本。只有平均分很难定位退化；逐样本 JSONL 能区分解析失败、拒答、幻觉、代码超时、检索错误和真实能力回归。

16.6.5 能力 Benchmark

能力 benchmark 覆盖语言、数学、代码、知识、长上下文和多模态。它们适合快速比较，但容易被污染、饱和和针对性优化。高分应被视为线索，而不是部署证明。

Emergence 主张本质上也是评测主张。如果小模型接近随机而大模型突然高分，首先要检查曲线是否由指标、prompt、答案抽取器或模型序列选择造成 [241]。生成式任务中 exact match 可能看不到部分正确；选择题中 normalized likelihood、校准概率和 argmax accuracy 也可能给出不同结论。一个严肃的 emergence evaluation card 应画出每个模型点，包含随机和简单启发式基线，给出置信区间，并尽量保存原始 log probability 或 cross-entropy，避免只展示最终分数。

16.6.6 pass@k 与选择效应

pass@k 衡量多次采样中至少一次成功的概率，常用于代码和数学。它不能代表单次用户体验，也可能被大量采样堆高。报告时应同时给出采样次数、成本和选择规则。

如果单次样本独立成功概率为 p，理想化的 pass@k 可以写成

$$\text{pass@}k = 1 - (1 - p)^k.$$

真实采样并不完全独立，但这个式子说明了一个关键事实：pass@100 的高分可能来自大量尝试和便宜验证，而不是单次可靠性。对生产 agent 来说，真正相关的可能是延迟预算下的 pass@1、带自动验证器的 pass@k、或 pass@k 加人工审核后的总成本。报告应说明候选如何生成，验证器是否可见测试，失败候选是否被隐藏，以及选择过程是否引入额外泄漏。

16.6.7 Benchmark 污染

污染会让模型在测试题上表现虚高。来源包括训练语料、论坛解析、翻译版本和合成数据。污染检查应成为数据报告和模型卡的一部分。

污染不是二元问题。模型可能见过原题、答案键、详细解析、论坛讨论、自动翻译版、同模板生成题或由评测题派生出的合成指令。严格报告应区分 exact overlap、near duplicate、semantic template overlap 和开发过程反复调榜造成的隐性污染。缓解手段包括私有测试

集、canary 字符串、训练-评测去重、时间切分、题目变体、隐藏答案抽取规则和发布后刷新测试集。越是影响模型选择的 benchmark，越应像生产测试集一样保护。

16.6.8 Benchmark 饱和与 Gaming

热门 benchmark 会被反复调参，逐渐失去区分度。模型可能学会题型和评分器偏好，而不是提升真实能力。评测套件应定期更新，并包含隐藏或动态样本。

16.6.9 长上下文、Agent 与生成评测

长上下文评测要看证据定位、整合和抗干扰；agent 评测要看任务完成、工具成本和权限；生成评测要看质量、可控性和安全。不同系统形态需要不同指标。统一平均分会掩盖关键失败。

长上下文报告不能只写最大 token 数。它应说明答案所需证据在上下文中的位置、是否加入 distractor、是否存在冲突证据、模型是否引用了正确片段、以及长输入成本是否超过产品预算。Agent 报告则要把模型能力和 harness 优势分开：允许哪些工具、是否可重试、是否能看见测试、是否有隐藏提示、失败后是否 rollback、状态是否持久化、以及人工提示是否参与。生成式报告要把质量、安全和来源一起看，尤其要列出生成内容何时必须携带 watermark、metadata 或 C2PA content credentials。

16.6.10 Rubric 与成对偏好

人类评测需要明确 rubric，说明真实性、有用性、安全性、简洁性和风格如何权衡。成对偏好比绝对评分稳定，但仍依赖候选分布。标注一致性和争议样例应被报告。

16.6.11 模型裁判

模型裁判能降低评测成本，但会继承模型偏见、位置偏好和风格偏好。它适合作为辅助，不应无校准地替代人类和客观指标。应对 judge 做一致性、盲测和对抗检查。

16.6.12 真实性与事实精度

事实性评测要区分可验证事实、开放解释和主观建议。RAG 系统还要检查回答是否被引用证据支持。模型说得流畅不等于事实正确。

TruthfulQA、FActScore、HaluEval 和 RAG 评测框架分别从常识误导、原子事实、幻觉检测和检索忠实度角度给出证据 [155, 168, 148, 35, 42]。长答案可以先拆成原子主张

$c_1, \ldots, c_n$，再统计哪些主张被证据支持：

$$\text{FactPrecision}(A) = \frac{|\{i : c_i \text{ 被证据支持}\}|}{n}.$$

这个指标强调精度，不强调完整性。一个模型少说可以得到较高事实精度，却没有完成用户任务；因此事实评测还要搭配覆盖率、弃答质量、引用定位和人工失败审查。高风险领域中，unsupported claim 应按严重失败处理，而不是用流畅度抵消。

16.6.13 校准与弃答

可靠系统应知道何时不确定。校准评测比较置信度和正确率，弃答评测检查模型是否在证据不足时拒绝。过度自信和过度拒答都是质量问题。

校准不等于让模型在回答里写“我有 80% 把握”。自由文本置信度容易受提示词、语气和任务格式影响。更可用的信号包括多次采样一致性、检索证据强度、verifier 分数、工具执行结果和领域规则检查。弃答评测必须包含无法回答、证据冲突、权限不足、问题缺字段和超出服务范围的样本；否则模型会学到“每个 prompt 都必须给答案”。发布报告应同时给出正确回答率、正确弃答率、错误弃答率和过度自信样例。

16.6.14 鲁棒性

鲁棒性包括提示改写、噪声、对抗指令、分布偏移、长上下文干扰和工具错误。模型在标准样本上正确，不代表在真实输入中稳定。鲁棒性评测应覆盖常见用户错误和恶意输入。

RAG 和工具系统的鲁棒性尤其要做故障注入。评测可以让检索器返回过期证据、冲突证据或带 prompt injection 的文档，让工具超时、返回空结果或改变 schema，让 parser 遇到非法 JSON、SQL 方言差异或单位换算错误。通过这些扰动，团队才能确认系统会降级、澄清、拒答或转人工，而不是把底层异常包装成自信答案。鲁棒性报告应按语言、文档年龄、权限组、工具类型和长上下文位置切片，而不是只给总体通过率。

16.6.15 政策分类

安全评测需要明确政策 taxonomy，例如自伤、暴力、网络、隐私、医疗、金融和违法行为。不同类别有不同拒答和安全替代策略。模糊政策会让标注和模型都不稳定。安全 taxonomy 还必须覆盖语言和领域。Baichuan 2 的 harmlessness evaluation 使用中英双语安全集合，覆盖偏见歧视、侮辱粗俗、违法或不道德内容、身体健康、心理健康、金融隐私和敏感话题 [251]。重点不是复用同一个数据集名称，而是设计切片：双语或领域模型需要

在实际使用语言和风险语境中评测，尤其不能漏掉健康、金融、隐私和政治敏感内容等常被通用红队低估的类别。

安全指标必须拆开看。Unsafe compliance 衡量模型是否按危险请求执行；false refusal 衡量模型是否错误拒绝良性请求；safe redirection 衡量拒答后是否提供合适替代；jailbreak robustness 衡量对角色扮演、编码、翻译、多轮诱导和工具调用攻击的抵抗；policy consistency 衡量不同语言、模态和上下文长度下是否稳定执行同一政策。一个单一“安全分”会掩盖过度拒答和拒答不足之间的权衡。

16.6.16 红队

红队通过主动寻找失败来补充常规评测。它应记录攻击路径、成功条件、复现步骤和修复验证。红队不是一次性活动，而是发布和更新周期的一部分。

有效红队要覆盖手工专家、自动 prompt mutation、多语言攻击、角色扮演、prompt injection、工具滥用、长上下文攻击和多模态攻击。每个发现都应转成回归测试，并标明是否修复、是否引入 false refusal、是否影响其他语言或模态。前沿系统卡中报告的红队和风险框架可以作为参考，但读者应关注其范围、未公开细节和残余不确定性，而不是把系统卡当作安全证明。

前沿系统卡的读法也应工程化。OpenAI o3/o4-mini 系统卡把推理模型、工具能力、安全评测和 Preparedness Framework 连接起来，并报告生物化学、网络安全和 AI 自我提升等跟踪类别 [181, 182]。Anthropic Claude 系统卡和 Responsible Scaling Policy 则用 AI Safety Level 组织能力阈值和防护措施 [5, 6]。这些材料的价值在于展示风险分类、评测、外部红队、防护和决策链条；它们不能替代读者自己的发布证据，因为应用场景、工具权限、用户群体和残余风险可能完全不同。

16.6.17 文档

模型卡、系统卡、数据说明和评测报告让外部读者理解模型边界。文档应说明训练数据、能力、限制、安全评测、适用场景和禁止场景。没有文档，治理无法落地。

16.6.18 风险管理框架

NIST AI RMF、EU AI Act 等框架把风险识别、测量、管理和治理制度化。它们不能替代技术评测，但能提供责任结构。高风险部署需要把技术证据映射到合规要求。

NIST AI RMF 的 govern、map、measure、manage 可以直接转成发布材料：govern 指定负责人、政策版本和审批路径；map 定义用户、用途、受影响人群和风险场景；measure 绑定评测卡、红队、监控和不确定性；manage 说明缓解、上线限制、事故响应和回滚 [176, 177]。

EU AI Act 的风险分层提醒团队区分普通助手、高风险系统和通用目的模型义务 [38]。工程上不需要把法规条文塞进模型 prompt，而是要把数据治理、日志、透明度、版权、人工监督、准确性、鲁棒性和网络安全要求映射到可检查 artifact。

GPAI 和前沿模型还需要区分模型层与系统层证据。模型层证据包括训练和测试过程、能力边界、训练内容摘要、版权政策、模型权重或服务端点的安全控制、模型评估和系统性风险评估。系统层证据包括具体应用的用途、人群、工具权限、检索语料、用户提示、下游过滤器、人工监督、post-deployment monitoring、事故响应和回滚。一个同样的 base model 用在客服总结、代码 agent、医疗问答和自动化操作中，系统层风险完全不同；治理报告不能只引用 base model 系统卡就结束。

前沿风险框架的共同点是把能力阈值和缓解措施绑定。OpenAI Preparedness Framework 和 Frontier Governance Framework 用 capability category、risk tier、residual risk、safeguards report、incident response 和外部专家输入组织决策 [182, 187]。Anthropic RSP 则把能力阈值、Frontier Safety Roadmap、Risk Reports、外部评审和安全目标连接起来 [6]。读者可以把这些公开框架当作发布材料的形状参考：每个高风险能力都应有评测、阈值、缓解、残余风险说明、复核人和更新机制。

16.6.19 可解释性、遗忘与水印

可解释性帮助理解模型机制和失败；machine unlearning 试图移除特定信息影响；水印试图标识生成内容。它们都不是万能解决方案。应报告适用条件、绕过方式和残余风险。

16.6.20 运行监控

发布后仍要监控输入分布、拒答率、错误、延迟、成本、安全事件和用户反馈。模型环境会变化，静态评测会过期。监控和回滚是治理的一部分。

监控也要尊重隐私和最小化原则。日志应尽量记录诊断所需字段，而不是无限期保存完整敏感对话；高风险类别应进入受控复核队列；用户投诉、幻觉报告、引用失败、工具失败和异常拒答应被归类成可回归测试。事故响应计划应说明谁 triage、谁决定临时降级、谁通知用户、如何保留证据、如何修复、如何验证修复没有引入新风险。没有这些流程，线上评测数据只会堆积成不可行动的日志。

16.6.21 部署限制

某些场景需要人类审核、权限限制、输出水印、引用要求或禁止自动决策。部署限制不是失败，而是风险控制。高能力模型尤其需要明确边界。

不部署也是一种工程决策。若评测覆盖不足、无法监控伤害、用户群体高度脆弱、错误不可逆、误用潜力严重、法律责任不清或缺少领域监督，即使 benchmark 分数高也应停止发布。发布结论应同时写明能力、样本不确定性、可逆性、受影响人群、缓解措施和剩余风险。“比旧模型更强”不是充分理由；必须说明为什么这些风险在当前场景中可接受。

16.7 关键术语、实现要点与练习

关键术语。 Construct 是评测意图测量的能力；evaluation card 把构念、人群、协议、指标、不确定性和决策规则写成可审计记录；evaluation harness 是把样本、prompt、模型调用、解析器、评分器和汇总规则变成可运行测量的代码；sample-level metric 针对单条样本计算分数；corpus-level metric 把全数据集结果汇总成发布分数；evaluation tracker 保存任务配置、模型配置、结果、逐样本细节和版本；benchmark contamination 是测试信息泄漏；pass@k 是多次采样中至少一次成功的概率；FactPrecision 衡量生成答案中被证据支持的原子主张比例；calibration 指置信信号和经验正确率一致；abstention 是证据不足时拒绝或澄清；模型裁判用 LLM 输出评分或解释；judge backend 是运行模型裁判的服务或本地后端；red teaming 主动寻找失败；unsafe compliance 表示模型执行了不应执行的危险请求；false refusal 表示模型拒绝了应当帮助的良性请求；release gate 是把指标阈值和发布、回滚或复核决策绑定的门禁；risk register 记录风险、证据、负责人和复查日期；system card 描述模型、工具、检索、过滤和监控构成的整体系统；GPAI 是可被集成到多种下游系统的通用目的模型；systemic risk 指可能产生大规模严重影响的模型层风险；training-content summary 是对训练内容来源和类别的公开摘要；governance crosswalk 把法规、标准或内部政策映射到证据 artifact；incident response 是线上事故的分级、降级、沟通和修复流程；unlearning 试图删除知识或能力；content credentials 是签名来源元数据。

实现要点。评测卡应包含构念、人群、协议、指标、不确定性和决策规则；harness 必须保存逐样本原始输出、解析结果、分数、异常、成本和数据版本；Lighteval 风格任务要显式记录数据仓库、split、prompt 函数、few-shot 设置、生成约束、backend、模型参数、sample-level 指标、corpus-level 汇总和结果细节字段；模型裁判要固定版本、随机化候选顺序、校准人类标签并记录 fallback；事实性要拆分原子主张、证据支持、覆盖率和弃答；鲁棒性要注入检索、工具、parser、权限和长上下文故障；安全评测要分 unsafe compliance、false refusal、safe redirection、jailbreak robustness 和跨语言一致性；生成式系统要评测 provenance；治理文档要连接模型、数据、工具、监控、事故响应和回滚；GPAI 或前沿模型要单独维护模型层证据、下游集成说明、训练内容摘要、systemic risk 评估、严重事件流程和网络安全控制；release gate 要提前定义阈值、样本量、置信区间、失败切片和责任人。

练习。

1. 为一个客服 RAG 助手写 evaluation card。
2. 为十道选择题实现一个最小评测 harness，并保存 raw response、parsed answer、score、模型版本和解码参数。
3. 把一个公开 benchmark 改写成 Lighteval 风格评测卡，列出 backend、模型参数、task config、sample-level 指标、corpus-level 汇总、逐样本细节字段和结果保存位置。
4. 设计模型裁判 prompt，并列出五种 judge bias。
5. 为文生视频模型设计视觉质量、时序一致性、物理一致性和来源评测。
6. 为 VLA 发布写安全 checklist。
7. 为一个 pass@k 代码评测写报告模板，说明采样数、验证器、延迟、失败候选和选择规则。
8. 设计一个污染审计表，区分原题、答案解析、翻译版、论坛讨论和同模板合成题。
9. 为一个已上线聊天系统写事故响应计划，包含 triage、降级、证据保留、用户通知和回归测试。
10. 为一个多工具 agent 写 release gate，要求同时包含任务成功率、工具误用率、人工介入率、p95 延迟、预算上限、回滚触发器和复核负责人。
11. 为一个生成式多媒体产品写 risk register，至少覆盖来源凭证缺失、版权投诉、误导性内容、隐私泄露和水印被移除。
12. 给一个 RAG 医疗问答样例设计事实性评测，要求拆成原子主张，标注证据支持、缺失证据、过期证据和应弃答样例。
13. 设计一个模型裁判校准实验，比较人类标签、单 judge、多 judge、长度控制和候选顺序随机化后的差异。
14. 把 NIST AI RMF 的 govern、map、measure、manage 映射到一个企业客服助手的发布记录，写出负责人、评测卡、风险登记和回滚证据。
15. 为一个 GPAI model 写模型层证据清单，覆盖技术文档、训练内容摘要、版权政策、下游集成说明、系统性风险评估、严重事件流程和网络安全控制。
16. 选一个前沿模型系统卡，抽取其中的风险类别、评测、缓解措施、残余风险和发布决策，并说明哪些证据不能直接迁移到你自己的应用。

领域	问题	证据
任务成功	是否解决用户问题	in-domain set、人类任务审查、工具日志
事实性	声明是否被支持	atomic fact、RAG faithfulness、abstention
安全	是否拒绝正确请求	unsafe compliance、false refusal、red team
鲁棒性	输入扰动是否破坏行为	改写、对抗、长上下文、工具失败
成本	是否可部署	延迟、token、硬件、美元/任务
治理	是否可检查	model card、data doc、政策版本、rollback
来源	生成内容是否可追踪	watermark、C2PA、metadata、检测、用户举报
合规映射	框架要求是否落到证据	governance crosswalk、训练内容摘要、风险登记、事件流程
前沿风险	能力阈值是否触发缓解	systemic risk 评估、red team、外部评审、security control

表 16.1: 评测与治理领域、问题和证据检查表。

16.8 结构化检查表

16.8.1 评测与治理发布清单

表 16.1 把评测、安全、治理和来源追踪合并为发布审查清单。

第十七章　研究前沿与实践路线

17.1　为什么前沿实践会改变课程

前面各章覆盖了大语言模型与生成式基础模型的主要生命周期，但前沿变化太快，不能只靠固定目录。最新系统常常在章节之间产生创新：架构变化依赖数据和训练稳定性；推理模型依赖奖励、采样和验证；多模态模型依赖服务延迟和安全策略；生成模型依赖目标函数、token/latent 表示和安全来源标记；agent 能力依赖工具协议和评测环境。

CS336 的价值在于强调从零实现：tokenizer、模型、优化器、GPU kernel、并行、scaling laws、数据处理、评测和 alignment/reasoning RL 都要亲自构建或测量 [222]。这也是本书建议的学习方式。图 17.1 概括了把前沿论文主张转化为最小复现、测量和系统决策的证据闭环。

前沿实践的核心纪律是可证伪。一个模型报告如果只说“架构更强”或“榜单更高”，还不足以进入教材主线。读者应能说清目标函数是什么，最小实现能复现哪一部分，数据和 tokenizer 是否改变，训练或推理预算增加了多少，验证集是否污染，失败样例是否暴露了真实风险。这个标准会让热门模型名退到第二位，让 objective、data、architecture、training recipe、inference policy 和 evaluation protocol 成为第一位证据。

前沿论文的第一轮阅读应像审查实验合同，而不是像阅读产品发布稿。先划出主张的最小可检验单位：是 tokenizer 改善了 token efficiency，还是数据混合改变了能力分布；是 MoE 路由提高了质量/成本，还是服务系统隐藏了额外通信代价；是推理 RL 学到更强策略，还是多采样和 verifier rerank 花了更多测试时计算。再列出可能推翻该主张的反证：换一个 benchmark split、换一个 prompt 模板、限制 token 预算、固定解码策略、删除可疑污染样本，或把平均分拆成失败切片。能经受这些检查的结论，才适合进入课程和系统路线图。

前沿主张进入本书可以分四级处理。第一层只是阅读列表，说明它值得关注但证据不足；第二层进入作业候选，要求有最小复现和负例；第三层进入章节主线，要求能改变生命周期变量并有独立证据；第四层进入发布清单，要求能转化为工程门禁、监控指标和回滚规则。分级能避免把热点直接写成定论。

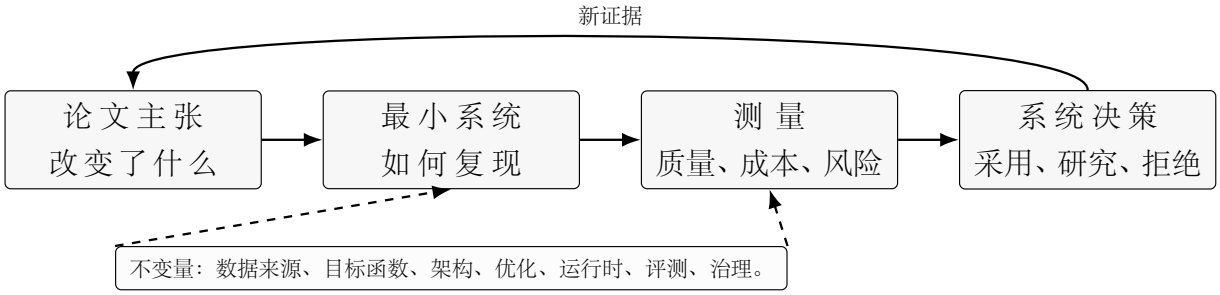

图 17.1: 前沿实践从论文主张走向最小复现、测量和系统决策。模型名称会变化，证据闭环不应变化。

17.2 路线图范围

结合 CS336 课程结构和 2025–2026 年前沿论文，本书当前 17 章结构仍然合理。把 LLM、多模态理解、图像/视频生成、具身 agent 和治理拆成多个孤立章节，表面上会增加覆盖面，但会削弱本书最重要的教材论证：现代基础模型是一条耦合的生命周期，而不是论文主题的并列清单。因此，本书采用的策略不是继续增加许多短章，而是扩大第 15 章为“多模态与生成式基础模型”，在第 12 章补上下文工程和记忆，在第 13 章补 DPO 之后的偏好目标，在第 16 章补长上下文、agentic、生成式、具身行动评测以及可解释性、遗忘、水印和来源治理。这个结构的好处是主线清楚：前两章定义数据和 token；中间章节解释 Transformer、训练、系统和服务；后训练章节解释如何把基础模型变成助手；应用章节解释检索、工具、agent、推理、多模态和生成；最后用评测和治理把所有能力放回可发布系统。高水平书稿需要的不是追逐每个模型名，而是给读者稳定的分析框架。

因此，一个方向是否需要新章节，不取决于论文数量，而取决于它是否改变了生命周期的独立变量。MoE 改变总参数、激活参数、路由稳定性、专家并行和服务调度；长上下文改变位置机制、KV 成本、检索策略、冲突证据处理和评测协议；推理 RL 改变后训练目标，也改变测试时 token 预算和验证器风险；统一多模态改变表示、目标函数、流式延迟、隐私和来源治理；VLA 改变输出模态，也改变行动安全、重置成本和责任边界。若这些变化仍然需要数据、优化、系统、适配、评测和治理共同解释，就应放回现有生命周期，而不是拆成孤立热点。

范围审计还要写清“没有覆盖什么”。本书不是模型百科，也不承诺追踪每个开源 checkpoint、排行榜刷新或产品接口更新。它覆盖的是可迁移的分析变量：数据来源、训练目标、结构约束、系统成本、后训练协议、推理预算、评测证据和治理责任。若某个新方向只是把这些变量重新组合，正文应把它归入已有生命周期；若它真的改变了变量本身，例如让动作成为一等输出、让媒体生成进入同一接口、或让推理预算成为策略的一部分，才需要扩展章节或作业。

路线图还需要版本纪律。每次新增前沿方向，都应记录进入原因、依赖证据、替代基线、预计维护成本和退出条件。若半年后独立复现失败、关键 benchmark 被污染、服务成本不可接受，或安全边界无法定义，该方向应退回阅读列表或附录观察项。高质量教材不是只会吸收新内容，也要能撤回不再可靠的内容。

17.3 前沿扩展方向

第一，实验作业应继续增加。读者应写 tokenizer、小 GPT、SFT collator、LoRA adapter、RAG evaluator、reward model 和 verifier loop，并报告准确率、显存、tokens/s、延迟和失败样例。

最小实现路径应按耦合度递增。先验证 tokenizer 在英文、代码、数学和非拉丁文字上的 token fertility；再训练一个小 decoder，检查 shape、causal mask、residual、初始化和 loss 对齐；再加入采样，观察 temperature、top-p、repetition penalty 和 stop token；然后做资源核算，记录参数量、激活显存、optimizer state、FLOPs、tokens/s 和 wall-clock cost。只有这些基础检查通过，读者才有能力判断大模型报告中的改进来自架构、数据、训练预算、后训练，还是推理策略。

从零实践也要保留实验记录。每个小项目都应记录代码版本、数据版本、随机种子、硬件、批大小、精度、loss 曲线、tokens/s、峰值显存、失败样例和复现实验命令。这个要求看似琐碎，但它能防止很多“我复现不了论文”的问题被误归因给模型规模。若一个小 GPT 的 loss mask、position ids、optimizer step 或 checkpoint 恢复语义都不清楚，后续对 LoRA、RAG、GRPO、MoE 或多模态系统的比较也不会可靠。

最小复现还应包含负控和消融。负控可以是随机标签、打乱证据、禁用工具、固定短预算或替换为旧 tokenizer；消融可以逐一删除数据混合、reward、router、verifier、cache 策略或安全过滤。若主张在负控下仍然成立，或在关键消融后几乎不变，说明实验没有真正测到论文声称的变量。

第二，服务系统应继续跟进 MoE 和 disaggregated inference。下一代推理不只是 batch scheduler，而是涉及 prefill/decode 分离、专家路由、跨集群通信和异构资源 [39]。

MoE 相关结论尤其容易被误读。总参数量不等于每个 token 激活的计算量，激活参数也不等于服务成本。严肃比较应同时报告总参数、激活参数、top-k 路由、router 辅助损失、专家负载均衡、expert parallelism、通信开销、cache 行为和真实流量下的延迟分位数。Kimi K2 一类稀疏系统的启发不只是“模型更大”，而是容量、路由和服务成本如何共同定义质量/美元 [137]。

第三，推理章节要持续跟踪 RLVR、GRPO、预算控制和 agentic tool use。RL for LLMs 的综述已经把 PPO、DPO、GRPO、RLHF、RLAIF 和 RLVR 放进同一设计空间 [221]。

推理方法也应按预算说明，而不是只报告准确率。一个系统可能选择短回答、长推理轨迹、多样本采样、工具调用、代码执行、检索、树搜索或 verifier rerank。每种选择都花费不同的 token、延迟、工具权限和失败风险。评测至少要回答五个问题：候选如何生成，分数或验证器为何可信，预算由用户固定还是系统自适应，异常输入下如何失败，提升来自更强策略、更多搜索、更好选择、污染还是拒答边界移动。

自适应测试时计算把这个问题变成产品策略。固定预算容易比较，但会在简单问题上浪费计算、在困难问题上过早停止；自适应预算更经济，却需要可靠的不确定性估计、停止规则和滥用防护。测试时扩展综述通常按扩展对象、扩展位置和质量度量整理方法 [263]；预算控制工作又区分用户指定预算和模型按难度分配预算 [3]。因此，推理系统报告不应只有 pass@1，还应报告 token 分布、p50/p95 延迟、放弃率、工具调用率、verifier 失败和最坏情况成本。

Agentic tool use 要进入同一预算框架。工具调用不是免费推理步骤，而是带权限、网络、沙盒、审计和回滚成本的外部行动。一个 agent 论文若只报告任务成功率，却不报告工具误用、权限失败、人工介入、重试风暴、成本上限和事故复盘字段，就不能支撑发布路线图。工具越强，评测越要覆盖失败路径。

第四，多模态与生成章节要持续跟踪视频、音频、实时语音、长上下文文档理解、统一理解-生成、扩散语言模型、AR-Diffusion 混合架构、任意到任意接口和 VLA 行动模型。Qwen3-VL 和 Qwen3-Omni 表明，多模态正从“图像加文本”走向统一交互系统 [203, 249]；Janus-Pro、MMaDA、Dream 7B、MammothModa2 和 AR-Omni 则说明，理解与生成、文本与视觉、自回归与扩散之间的边界正在重新组织 [20, 253, 257, 216, 23]；RT-2 和 OpenVLA 进一步说明，行动也可以成为统一接口的一部分 [13, 136]。

架构前沿可以压缩成五个压力点：稀疏计算、attention memory、原生多模态、生成目标和可训练性。稀疏计算要求报告 router 与服务成本；长上下文要求报告检索、聚合和冲突证据，而不只是最大长度；原生多模态要求报告 interleaved 输入、流式输出、时间 grounding 和模态安全；扩散或 rectified-flow 目标要求和自回归目标比较采样步数、编辑能力、延迟和可控性；可训练性则要求把 optimizer、初始化、norm、clipping、精度和数据混合一起看。没有这些判据，“新架构”很容易只是完整系统配方的代名词。

长上下文和 attention 替代路线尤其需要保守解读。GQA、MQA、MLA、状态空间、循环记忆、外部 memory 和检索压缩都可能降低某一类成本，但长输入任务不等于“把更多 token 塞进去”。评测必须证明模型能在长文中定位证据、聚合分散信息、处理矛盾、抵抗 distractor，并在后训练和服务预算下保持稳定。RetNet、Mamba、Mamba-2 和 Titans 等工作提示读者关注 attention scaling 之外的设计空间 [225, 44, 29, 11]，但任何替代机制进入教材前，都应接受同样的任务、成本和风险核算。

17.4 读者路线图

建议路线是：先实现最小 tokenizer 和 GPT；再加入资源核算和小规模训练；然后做数据清洗、去重和污染检查；接着做 SFT、LoRA、RAG、上下文工程和 typed memory；再做 DPO、KTO/ORPO/SimPO 对比、GRPO 或 verifier-guided RL；之后实现一个带 KV cache、流式输出、延迟统计和安全过滤的服务；最后增加一个多模态、生成式或 VLA 组件，并分别评估感知、推理、生成质量、行动安全、来源和治理。每一步都问同一个问题：目标函数是什么，数据来自哪里，系统成本是多少，评测是否可信。

数据、评测和治理应被看作一条闭环。数据过滤定义模型学到的分布，评测声明模型能做什么，治理决定哪些数据和评测能影响发布，线上监控又产生新的失败样例和修复需求。一个当前可审计的数据管线应记录许可证、采集日期、过滤规则、去重方式、PII 策略、语言分布、质量分数、合成数据生成器版本和 benchmark overlap 检查。一个当前可审计评测管线应记录 prompt 模板、解码设置、工具权限、检索索引、judge model、人类 rubric、置信区间和失败切片。治理审查则应把这些 artifact 连接到发布对象、目标用户、约束、监控和回滚路径。

这个闭环也解释了为什么“数据作业”“scaling 作业”“对齐作业”不应被看成课程中的孤立练习。Common Crawl 清洗决定了预训练证据边界；loss curve 拟合决定了预算规划；RAG 评测决定了知识是否留在外部证据库；偏好或 RLVR 作业决定了系统如何使用人类或可验证反馈；治理记录决定这些结果能否支撑上线。教学项目越小，越要把这些字段写清楚，因为小项目是大系统审计的缩影。

17.5 深入展开：为什么结构保持 17 章

本章不是新论文列表，而是研究实践路线图。它回答一个问题：在 CS336 式从零实现课程和 2025–2026 年前沿论文之后，读者应该怎样把论文主张转化为可实现、可测量、可检查的系统？结论是，目录不应继续拆碎，而应保持生命周期结构：数据、token、架构、优化、系统、适配、检索、偏好、推理、多模态生成、评测和治理。原因很直接。现代基础模型的创新往往发生在章节边界：MoE 改变架构，也改变训练并行和推理调度；长上下文改变模型，也改变 RAG 和评测；推理 RL 改变后训练，也改变测试时预算；统一多模态改变表示，也改变服务延迟、隐私和来源治理；VLA 改变输出模态，也改变安全责任。把这些主题拆成互不相干的小章，会让读者失去系统联系。

本书把进一步学习路线写成项目序列。先实现 tokenizer 和小 GPT，确认 shape、mask 和 loss；再加入训练循环、资源核算和 scaling；再做数据去重、污染检查和领域切片；再做 SFT、LoRA、RAG、typed memory；再做 DPO 和 verifiable RL；再做服务系统和 KV cache；最后加入多模态、生成式或 VLA 组件。每一步都要求读者报告正确性、成本和失

败样例。

这一章的最终判断是：前沿会变，但不变量不会变。未来如果 attention 被替代，仍需要数据、优化、服务、适配、评测和治理；如果 GRPO 被替代，仍需要奖励设计、预算核算和鲁棒性测试；如果 Omni/VLA 系统成为主流，仍需要时间 grounding、行动安全、隐私、来源和流水线评测。书的定位越高，越不能只追模型名，而要保留这些不变量。

因此，读者后续学习可以按“耦合度”而不是按“论文热度”排序。先做可单机验证的 tokenizer、decoder 和训练循环；再做数据 manifest、污染检查和资源核算；再接入 SFT、LoRA、RAG 和偏好学习；再增加 verifier、GRPO 式训练或预算化推理；最后处理多模态、生成式和 VLA 系统，因为这些方向同时改变输入、输出、评测、安全和服务。这样的顺序能让每个新能力都带着可解释的证据进入系统，而不是把多个不稳定变量一次性叠在一起。

17.6 章节细节

17.6.1 为什么需要前沿章

大模型领域变化太快，单纯列举模型名称很快过时。前沿章的任务是给读者一套判断框架：新工作改变了目标、数据、结构、系统、推理还是评测。只有这样，读者才能持续更新知识。

17.6.2 覆盖了什么，还需要关注什么

本书覆盖文本 LLM、多模态、生成模型、对齐、RAG、agent、推理和治理，但仍有持续变化的方向。包括统一生成理解、具身智能、长程记忆、低成本推理、开放评测和制度治理。读者应把这些看作研究问题，而不是附录话题。

覆盖范围的另一面是优先级管理。面对新论文，先问它是否改变了生命周期变量；再问是否已有独立证据；最后问它是否能转化为作业、评测或发布清单。如果答案都是否定的，它可以进入阅读列表，但不必进入正文主线。

17.6.3 从零实践作为研究方法

从零实现 tokenizer、Transformer、训练循环、推理服务和 RAG，可以让读者理解抽象背后的约束。实践不是为了取代成熟框架，而是为了发现框架隐藏的假设。高水平研究者需要能读论文，也能看懂系统瓶颈。

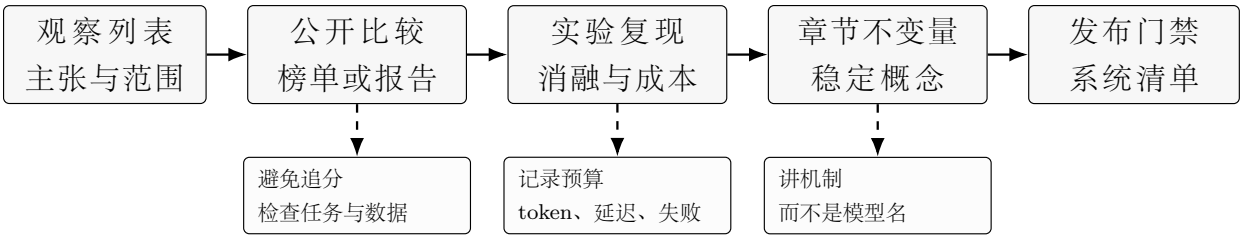

图 17.2: 前沿内容应在具备足够证据后，才进入主线章节并形成稳定工程不变量或发布门禁。该图对应本书对 CS336 式实现证据、公开排行榜、AILuminate 安全评测和 SWE-bench 任务 harness 的使用方式 [222, 79, 169, 227]。

17.6.4 前沿架构问题

未来架构可能在注意力、状态空间、MoE、扩散语言模型、长上下文和多模态连接器之间继续演化。判断架构贡献时，应固定数据、预算和后训练，或者明确比较的是完整系统。架构创新必须经受服务和评测检验。

图 17.2 给出本章判断前沿材料是否进入正文主线的证据阶梯：只有当论文主张能变成稳定机制、系统检查项或发布门禁时，它才应从阅读列表进入教材结构。

17.6.5 推理与自适应测试时计算

推理能力越来越依赖测试时预算控制。研究问题包括何时多采样、何时搜索、何时调用工具、何时停止和如何校准置信度。产品系统还必须把这些选择转化为成本和延迟策略。

17.6.6 数据、评测与治理作为一条闭环

数据决定训练，评测发现问题，治理决定能否发布，监控再产生新的数据和修复需求。这是一条闭环，而不是独立章节。成熟团队会把数据 manifest、评测报告、红队结果和事故复盘连起来。

17.6.7 读者后续路线图

读者可以按四条路线深入：从零实现与系统优化、后训练与对齐、RAG/agent 应用、多模态与生成模型。每条路线都应保持同一标准：说明目标、约束、指标、数据和失败模式。能这样提问，比追逐模型名称更持久。

四条路线最终应合流到同一个交付物：一个小而完整的系统报告。报告应包括模型与数据合同、训练和服务成本、评测 harness、失败样例、风险登记册、回滚方案和后续实验。能写出这样的报告，说明读者已经把本书的章节知识转化成工程判断。

17.7 关键术语、实现要点与练习

关键术语。 From-scratch implementation 用最小实现暴露假设；minimal reproduction 指复现论文主张所需的最小系统；resource accounting 报告显存、FLOPs、带宽、tokens/s、延迟和成本；disaggregated inference 把 prefill、decode 或不同计算阶段拆到不同资源上；adaptive test-time compute 按输入难度、置信度或策略分配推理预算；frontier evidence matrix 把前沿方向按变量和证据门槛归档；benchmark construct 指一个基准真正测量的能力或风险；leaderboard provenance 指榜单分数背后的模型版本、harness、提交规则和归一化规则；practice/official split 指公开练习集和隐藏正式测试集的分离；native multimodality 把文本、图像、音频、视频或语音作为一等输入输出；generative foundation model 生成文本、图像、音频、视频、代码或动作；VLA model 把视觉观察、语言指令和动作输出连接起来。

实现要点。 读者应把每个模型 claim 还原为 objective、data、architecture、training recipe、inference policy 和 evaluation protocol；新论文应被视为待测试证据，而不是直接复制的 recipe；任何前沿结论都应同时报告质量、成本、风险和失败样例；公共榜单要记录 raw/normalized score、model commit、评测版本和提交规则；安全 benchmark 要区分 practice set、official hidden set、evaluator 可见性、是否允许发布逐样本失败日志和能否外推到多轮工具系统；目录结构要服务系统生命周期，而不是追逐热点。

练习。

1. 为本书任一章设计 CS336 式实现作业。

2. 比较 dense decoder 和 MoE 的总参数、激活参数、路由和服务成本。

3. 为数学问答设计 adaptive test-time compute 策略，说明何时短答、何时长推理、何时采样、何时调用工具、何时弃答，并报告预算上限。

4. 扩展一个 RAG 评测，使其包含数据许可证、访问控制、prompt injection 测试、引用忠实度和回滚标准。

5. 把第 15 章改写成一个课程模块，安排 autoregressive、diffusion、Omni 和 VLA 实验。

6. 为同一课程模块增加 VLA lab，说明观察流、动作空间、仿真器或机器人、成功指标、安全联锁和人工介入策略。

7. 选择一篇 2025 年之后论文，说明其结论在换 tokenizer、数据混合、预算或评测 split 后可能如何改变。

8. 为一个 MoE 服务系统设计报告模板，要求同时包含总参数、激活参数、router 稳定性、专家负载、通信开销和延迟分位数。

9. 用前沿证据公式评估一篇新论文，分别说明 ΔQ、ΔC、ΔR、U 和 E 的证据来源。

10. 选择一个公开榜单分数，补全模型版本、数据版本、评测脚手架、原始/归一化分数、提交次数、预算和失败样例字段，并说明该分数不能支持哪些部署结论。

11. 为一个安全基准设计公开练习集和隐藏正式集，说明公开练习集如何用于调试、隐藏正式集如何防止过拟合、以及哪些多轮或工具风险不在单轮分数中。

17.8 前沿证据矩阵

前沿论文进入本书时，不按模型名称堆叠，而按它改变的系统维度归档。一个新方法如果不能说明目标、数据、训练、推理和评测证据，就还只是候选线索。

公共 leaderboard 也应进入证据矩阵，而不是被当成结论本身。Hugging Face 的 leaderboard 文档把 Hub 上的评测分成官方 benchmark 结果、社区维护榜单和 Open LLM Leaderboard 这类集中评测项目，并提醒读者区分 raw score、normalized score、模型类别和提交来源 [79, 90]。因此，引用一个榜单分数时，应同时记录模型 commit、权重或 API 版本、推理模板、评测 harness、原始分数、归一化规则、是否来自官方 provider、是否有提交者筛选、以及榜单是否允许多次提交。没有这些字段，榜单只说明“某个系统在某个公开协议下得分”，不能说明模型本身更强。

Agent 和代码 benchmark 更需要拆开模型与脚手架。SWE-bench 原论文把任务定义为真实 GitHub issue 修复 [131]，而当前 SWE-bench Verified 数据集卡还把 500 条 test 样本、repo、instance id、base commit、patch、test patch、problem statement、FAIL_TO_PASS、PASS_TO_PASS、environment setup commit 和 difficulty 等字段暴露为数据合同 [227]。一个 agent 报告若只给 resolved 百分比，就漏掉了关键变量：使用哪个数据版本，是否固定 scaffold，测试是否可见，允许多少轮编辑和重试，能否联网，是否有人类提示，失败 patch 是否保存，成本和 wall-clock 是否计入。对教材来说，SWE-bench 类结果的价值不是追最新第一名，而是迫使读者把“模型能力、工具运行时、仓库环境、验证器和预算”分开报告。

安全 benchmark 的证据边界也要写清。MLCommons AILuminate 覆盖 safety、jailbreak、多模态和 agentic workstream 等方向，试图提供可重复的风险和可靠性评测 [169]；其 FAQ 同时说明，基准结果展示的是相对安全水平，并不保证某个系统安全可靠。它还区分 demo、offline practice、online practice 和 official test：official test 使用完整隐藏提示和私有 evaluator，practice prompt 与 official prompt 应保持诊断关系，训练系统不应把

前沿方向	需要追踪的变量	进入教材主线的条件
稀疏 MoE	总参数、激活参数、router、专家并行、服务延迟	质量/成本优于 dense 基线，并报告路由稳定性
长上下文与记忆	位置机制、KV 成本、检索、压缩、冲突证据	不只支持长输入，还能定位、聚合和拒绝错误证据
推理 RL 与测试时计算	奖励、采样数、验证器、令牌预算、pass@k/pass@1	提升准确率的同时报告延迟、成本和轨迹风险
统一多模态生成	表示、目标函数、采样步数、来源治理	同时评测理解、生成、时间一致性和安全边界
VLA 与行动模型	观察、动作空间、控制频率、联锁、重置成本	真实或仿真任务成功率与危险动作率同时达标
公共榜单与安全基准	数据版本、评测脚手架、原始/归一化分数、隐藏集、提交协议	可复现分数、无训练泄漏，并说明构念边界和外推限制

表 17.1: 前沿方向、追踪变量与进入教材主线条件检查表。

practice set 当作调参数据 [170]。这给前沿治理一个明确模式：公开练习集用于开发，隐藏官方集用于外部声明；公开失败日志帮助修复，隐藏官方项目保护测量完整性；单轮安全分数需要和多轮、工具、权限、检索、图像输入和部署监控一起解释。

表 17.1 将前沿方向筛选成证据矩阵，防止只按模型名或榜单热度更新课程。

也可以把一篇新论文的采用价值写成证据清单：可复现实验中的质量增益是否足够大，训练或服务成本是否可接受，安全、隐私、版权或治理风险是否增加，不确定性是否被说明，独立证据是否足够强。这个清单的用途不是机械打分，而是防止只因模型名新、论文热或榜单高就改变教材结构。

证据矩阵还应保留“不采用”的理由。拒绝一个方向可能因为质量增益只在污染 benchmark 上成立，成本曲线无法支撑部署，失败样例触及高风险用户，许可证或数据来源不清，或独立复现还没有出现。把拒绝理由写下来，可以防止半年后同一热点换个名称重新进入路线图。

为了让判断更显式，可以把前沿主张的采用价值写成

$$S_{\text{frontier}} = \Delta Q - \lambda_C \Delta C - \lambda_R \Delta R - \lambda_U U + \lambda_E E.$$

其中 ΔQ 是复现实验中的质量增益，ΔC 是额外训练或服务成本，ΔR 是安全或治理风险，U 是残余不确定性，E 是独立证据强度。这个式子不是通用指标，而是提醒读者：一个前沿结果只有在质量、成本、风险、不确定性和独立证据之间经得起解释，才适合进入教材

主线或系统路线图。

附录 A　附录：记号、实验记录与术语

A.1　常用记号

B 表示 batch size，T 表示序列长度，d 表示隐藏维度，V 表示词表大小，theta 表示模型参数，policy theta 表示策略模型，p theta 表示语言模型分布。

A.2　实验记录清单

每次实验应记录：代码版本、数据版本、数据许可证或来源说明、tokenizer、聊天模板、模型配置、随机种子、优化器、学习率、batch、精度、硬件、预计运行时间、checkpoint、恢复语义、验证集、主要指标、回归指标、失败样例和结论。短练习可以把这些字段写成一段 run card；完整实验应把配置、命令、数据 manifest、评测脚本和失败模式分开保存。

A.3　术语表

基础模型 经过预训练但尚未进行指令微调或偏好优化的模型。

后训练 在预训练之后进行的 SFT、偏好优化、安全调优和领域适配。

KV Cache 推理时缓存历史 token 的 key/value 张量以降低增量生成成本。

MoE Mixture-of-Experts，每个 token 只激活部分专家的稀疏模型结构。

RAG Retrieval-Augmented Generation，推理时检索外部证据并生成答案的系统。

上下文工程 对送入模型的系统指令、用户请求、检索证据、工具输出、记忆和状态进行设计、压缩、路由、权限控制和检查的工程方法。

统一理解-生成模型 同时解释输入媒体并生成或编辑媒体的多模态模型或系统。

任意到任意模型 接受多种输入模态并能输出多种模态的系统，例如文本、图像、音频、语音或视频。

扩散模型 学习反转加噪过程、通过迭代去噪生成数据的生成模型，可工作在像素、token 或 latent 空间。

Rectified Flow 学习从噪声到数据的连续流的生成建模形式，常用于高分辨率视觉生成。

VLA 模型 Vision-Language-Action 模型，根据视觉观察和语言指令在物理或仿真环境中产生动作。

世界模型 用于预测或模拟未来状态、观察或行动后果的模型，可辅助规划、评测或控制。

RLVR Reinforcement Learning with Verifiable Rewards，用可验证答案、代码测试或规则奖励训练模型。

Unlearning 训练后删除或压制特定知识或行为，同时尽量保留无关能力的技术方向。

水印 在生成内容中嵌入或检测统计信号、元数据或来源标记的方法。

内容凭证 描述媒体来源、编辑历史或生成过程的签名来源元数据。

评测污染 训练或开发流程接触到测试题、答案、解析或变体，导致分数高估。

参考文献

[1] Ainslie, J., Lee-Thorp, J., de Jong, M., Zemlyanskiy, Y., Lebron, F., Sanghai, S.: GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints. arXiv preprint arXiv:2305.13245 (2023). URL https://arxiv.org/abs/2305.13245

[2] Alayrac, J.B., Donahue, J., Luc, P., Miech, A., Barr, I., Hasson, Y., Lenc, K., Mensch, A., Millican, K., Reynolds, M., Ring, R., Rutherford, E., Cabi, S., Han, T., Gong, Z., Samangooei, S., Monteiro, M., Menick, J., Borgeaud, S., Brock, A., Nematzadeh, A., Sharifzadeh, S., Binkowski, M., Barreira, R., Vinyals, O., Zisserman, A., Simonyan, K.: Flamingo: a Visual Language Model for Few-Shot Learning. arXiv preprint arXiv:2204.14198 (2022). URL https://arxiv.org/abs/2204.14198

[3] Alomrani, M.A., Zhang, Y., Li, D., Sun, Q., Pal, S., Zhang, Z., Hu, Y., Ajwani, R.D., Valkanas, A., Karimi, R., et al.: Reasoning on a Budget: A Survey of Adaptive and Controllable Test-Time Compute in LLMs. arXiv preprint arXiv:2507.02076 (2025). URL https://arxiv.org/abs/2507.02076

[4] Anthropic: Circuit tracing: Revealing computational graphs in language models. Transformer Circuits Thread (2025). URL https://transformer-circuits.pub/2025/attribution-graphs/methods.html

[5] Anthropic: Claude 4 System Card. Tech. rep., Anthropic (2025). URL https://www.anthropic.com/system-cards

[6] Anthropic: Responsible scaling policy version 3.3. Tech. rep., Anthropic (2026). URL https://www.anthropic.com/responsible-scaling-policy

[7] Azar, M.G., Guo, Z.D., Piot, B., Munos, R., Rowland, M., Valko, M., Calandriello, D.: A general theoretical paradigm to understand learning from human preferences.

International Conference on Artificial Intelligence and Statistics (2024). URL https://arxiv.org/abs/2310.12036

[8] Bai, Y., Kadavath, S., Kundu, S., Askell, A., Kernion, J., Jones, A., Chen, A., Goldie, A., Mirhoseini, A., McKinnon, C., et al.: Constitutional AI: Harmlessness from AI Feedback. arXiv preprint arXiv:2212.08073 (2022). URL https://arxiv.org/abs/2212.08073

[9] Bai, Y., Kadavath, S., Kundu, S., Askell, A., Kernion, J., Jones, A., Chen, A., Goldie, A., Mirhoseini, A., McKinnon, C., et al.: Training a helpful and harmless assistant with reinforcement learning from human feedback. arXiv preprint arXiv:2204.05862 (2022). URL https://arxiv.org/abs/2204.05862

[10] Bai, Y., Lv, X., Zhang, J., Lyu, H., Tang, J., Huang, Z., Du, Z., Liu, X., Zeng, A., Hou, L., et al.: Longbench v2: Towards deeper understanding and reasoning on realistic long-context multitasks. arXiv preprint arXiv:2412.15204 (2024). URL https://arxiv.org/abs/2412.15204

[11] Behrouz, A., Zhong, P., Mirrokni, V.: Titans: Learning to memorize at test time. arXiv preprint arXiv:2501.00663 (2025). URL https://arxiv.org/abs/2501.00663

[12] Borgeaud, S., Mensch, A., Hoffmann, J., Cai, T., Rutherford, E., Millican, K., van den Driessche, G., Lespiau, J.B., Damoc, B., Clark, A., et al.: Improving language models by retrieving from trillions of tokens. arXiv preprint arXiv:2112.04426 (2022). URL https://arxiv.org/abs/2112.04426

[13] Brohan, A., Brown, N., Carbajal, J., Chebotar, Y., Chen, X., Choromanski, K., Ding, T., Driess, D., Dubey, A., Finn, C., et al.: Rt-2: Vision-language-action models transfer web knowledge to robotic control. arXiv preprint arXiv:2307.15818 (2023). URL https://arxiv.org/abs/2307.15818

[14] Brown, T.B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al.: Language models are few-shot learners. arXiv preprint arXiv:2005.14165 (2020). URL https://arxiv.org/abs/2005.14165

[15] Cai, T., Li, Y., Geng, Z., Peng, H., Lee, J.D., Chen, D., Dao, T.: Medusa: Simple LLM Inference Acceleration Framework with Multiple Decoding Heads. arXiv preprint arXiv:2401.10774 (2024). URL https://arxiv.org/abs/2401.10774

[16] Casper, S., Davies, X., Shi, C., Gilbert, T.K., Scheurer, J., Rando, J., Freedman, R., Korbak, T., Lindner, D., Freire, P., et al.: Open problems and fundamental limitations of reinforcement learning from human feedback. Transactions on Machine Learning Research (2023). URL https://arxiv.org/abs/2307.15217

[17] Chandra, B., Dunietz, J., Roberts, K., Lee, Y., Fontana, P., Awad, G.: Reducing risks posed by synthetic content: An overview of technical approaches to digital content transparency. Tech. Rep. NIST AI 100-4, National Institute of Standards and Technology (2024). DOI 10.6028/NIST.AI.100-4

[18] Chen, J., Meng, S., Chen, Y., Zhao, M., Gui, W., Guo, X.: Toc-bench: A temporal object consistency benchmark for video large language models. arXiv preprint arXiv:2605.09904 (2026). URL https://arxiv.org/abs/2605.09904

[19] Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H.P.d.O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al.: Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374 (2021). URL https://arxiv.org/abs/2107.03374

[20] Chen, X., Wu, Z., Liu, X., Pan, Z., Liu, W., Xie, Z., Yu, X., Ruan, C.: Janus-pro: Unified multimodal understanding and generation with data and model scaling. arXiv preprint arXiv:2501.17811 (2025). URL https://arxiv.org/abs/2501.17811

[21] Chen, Y., Qian, S., Tang, H., Lai, X., Liu, Z., Han, S., Jia, J.: LongLoRA: Efficient Fine-tuning of Long-Context Large Language Models. International Conference on Learning Representations (2024). URL https://arxiv.org/abs/2309.12307

[22] Chen, Z., Wang, S., Tan, Z., Fu, X., Lei, Z., Wang, P., Liu, H., Shen, C., Li, J.: A survey of scaling in large language model reasoning. arXiv preprint arXiv:2504.02181 (2025). URL https://arxiv.org/abs/2504.02181

[23] Cheng, D., Yuan, R., Li, Y., You, R., Wang, W., Nie, L., Zhang, L., Li, W.: Aromni: A unified autoregressive model for any-to-any generation. arXiv preprint arXiv:2601.17761 (2026). URL https://arxiv.org/abs/2601.17761

[24] Christiano, P.F., Leike, J., Brown, T.B., Martic, M., Legg, S., Amodei, D.: Deep reinforcement learning from human preferences. Advances in Neural Information Processing Systems (2017). URL https://arxiv.org/abs/1706.03741

[25] Coalition for Content Provenance and Authenticity: C2PA Specifications. Technical specification (2026). URL https://spec.c2pa.org/specifications/specifications/2.0/index.html

[26] Cui, Y., Yang, Z., Yao, X.: Efficient and Effective Text Encoding for Chinese LLaMA and Alpaca. arXiv preprint arXiv:2304.08177 (2023). URL https://arxiv.org/abs/2304.08177

[27] Dao, T.: FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning. arXiv preprint arXiv:2307.08691 (2023). URL https://arxiv.org/abs/2307.08691

[28] Dao, T., Fu, D.Y., Ermon, S., Rudra, A., Re, C.: FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. arXiv preprint arXiv:2205.14135 (2022). URL https://arxiv.org/abs/2205.14135

[29] Dao, T., Gu, A.: Transformers are SSMs: Generalized Models and Efficient Algorithms Through Structured State Space Duality. arXiv preprint arXiv:2405.21060 (2024). URL https://arxiv.org/abs/2405.21060

[30] DeepSeek-AI, Guo, D., Yang, D., Zhang, H., Song, J., Wang, P., Xu, R., Zhu, Q., Zhang, R., Ma, S., et al.: DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. arXiv preprint arXiv:2501.12948 (2025). URL https://arxiv.org/abs/2501.12948

[31] DeepSeek-AI, Liu, A., Feng, B., Xue, B., Wang, B., Wu, B., Lu, C., Zhao, C., Deng, C., Ruan, C., et al.: DeepSeek-V3 Technical Report. arXiv preprint arXiv:2412.19437 (2024). URL https://arxiv.org/abs/2412.19437

[32] Dettmers, T., Pagnoni, A., Holtzman, A., Zettlemoyer, L.: QLoRA: Efficient Finetuning of Quantized LLMs. arXiv preprint arXiv:2305.14314 (2023). URL https://arxiv.org/abs/2305.14314

[33] Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv preprint arXiv:1810.04805 (2018). URL https://arxiv.org/abs/1810.04805

[34] Dua, D., Wang, Y., Dasigi, P., Stanovsky, G., Singh, S., Gardner, M.: DROP: A Reading Comprehension Benchmark Requiring Discrete Reasoning Over Paragraphs. Proceedings of NAACL-HLT (2019). URL https://arxiv.org/abs/1903.00161

[35] Es, S., James, J., Espinosa-Anke, L., Schockaert, S.: RAGAS: Automated Evaluation of Retrieval Augmented Generation. arXiv preprint arXiv:2309.15217 (2023). URL https://arxiv.org/abs/2309.15217

[36] Esser, P., Kulal, S., Blattmann, A., Entezari, R., Müller, J., Saini, H., Levi, Y., Lorenz, D., Sauer, A., Boesel, F., et al.: Scaling Rectified Flow Transformers for High-Resolution Image Synthesis. arXiv preprint arXiv:2403.03206 (2024). URL https://arxiv.org/abs/2403.03206

[37] Ethayarajh, K., Xu, W., Muennighoff, N., Jurafsky, D., Kiela, D.: KTO: Model Alignment as Prospect Theoretic Optimization. arXiv preprint arXiv:2402.01306 (2024). URL https://arxiv.org/abs/2402.01306

[38] European Union: Regulation (eu) 2024/1689 of the european parliament and of the council (artificial intelligence act) (2024). URL https://eur-lex.europa.eu/legal-content/en/TXT/?uri=CELEX:32024R1689

[39] Feng, Y., Tan, X., Sew, K.H., Jiang, Y., Zhu, Y., Xu, H.: Frontier: Simulating the Next Generation of LLM Inference Systems. arXiv preprint arXiv:2508.03148 (2025). URL https://arxiv.org/abs/2508.03148

[40] Frantar, E., Ashkboos, S., Hoefler, T., Alistarh, D.: GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers. arXiv preprint arXiv:2210.17323 (2022). URL https://arxiv.org/abs/2210.17323

[41] Fu, Y., Bailis, P., Stoica, I., Zhang, H.: Break the Sequential Dependency of LLM Inference Using Lookahead Decoding. arXiv preprint arXiv:2402.02057 (2024). URL https://arxiv.org/abs/2402.02057

[42] Gan, A., Yu, H., Zhang, K., Liu, Q., Yan, W., Huang, Z., Tong, S., Hu, G.: Retrieval augmented generation evaluation in the era of large language models: A comprehensive survey. arXiv preprint arXiv:2504.14891 (2025). URL https://arxiv.org/abs/2504.14891

[43] Grattafiori, A., Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Vaughan, A., et al.: The llama 3 herd of models. arXiv preprint arXiv:2407.21783 (2024). URL https://arxiv.org/abs/2407.21783

[44] Gu, A., Dao, T.: Mamba: Linear-Time Sequence Modeling with Selective State Spaces. arXiv preprint arXiv:2312.00752 (2023). URL https://arxiv.org/abs/2312.00752

[45] Guan, T., Liu, F., Wu, X., Xian, R., Li, Z., Liu, X., Wang, X., Chen, L., Huang, F., Yacoob, Y., Manocha, D., Zhou, T.: Hallusionbench: An advanced diagnostic suite for entangled language hallucination and visual illusion in large vision-language models. arXiv preprint arXiv:2310.14566 (2023). URL https://arxiv.org/abs/2310.14566

[46] Guo, X., Huo, J., Shi, Z., Song, Z., Zhang, J., Zhao, J.: T2vphysbench: A first-principles benchmark for physical consistency in text-to-video generation. arXiv preprint arXiv:2505.00337 (2025). URL https://arxiv.org/abs/2505.00337

[47] Gururangan, S., Marasovic, A., Swayamdipta, S., Lo, K., Beltagy, I., Downey, D., Smith, N.A.: Don't stop pretraining: Adapt language models to domains and tasks. Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (2020). URL https://arxiv.org/abs/2004.10964

[48] Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., Steinhardt, J.: Measuring massive multitask language understanding. arXiv preprint arXiv:2009.03300 (2020). URL https://arxiv.org/abs/2009.03300

[49] Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., Steinhardt, J.: Measuring Mathematical Problem Solving With the MATH Dataset. arXiv preprint arXiv:2103.03874 (2021). URL https://arxiv.org/abs/2103.03874

[50] Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., de Las Casas, D., Hendricks, L.A., Welbl, J., Clark, A., et al.: Training compute-optimal large language models. arXiv preprint arXiv:2203.15556 (2022). URL https://arxiv.org/abs/2203.15556

[51] Hong, J., Lee, N., Thorne, J.: ORPO: Monolithic Preference Optimization without Reference Model. arXiv preprint arXiv:2403.07691 (2024). URL https://arxiv.org/abs/2403.07691

[52] Hsieh, C.P., Sun, S., Kriman, S., Acharya, S., Rekesh, D., Jia, F., Ginsburg, B.: Ruler: What's the real context size of your long-context language models? arXiv preprint arXiv:2404.06654 (2024). URL https://arxiv.org/abs/2404.06654

[53] Hu, E.J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., Chen, W.: LoRA: Low-Rank Adaptation of Large Language Models. arXiv preprint arXiv:2106.09685 (2021). URL https://arxiv.org/abs/2106.09685

[54] Huang, Z., Zhang, F., Xu, X., He, Y., Yu, J., Dong, Z., Ma, Q., Chanpaisit, N., Si, C., Jiang, Y., et al.: Vbench++: Comprehensive and versatile benchmark suite for video generative models. arXiv preprint arXiv:2411.13503 (2024). URL https://arxiv.org/abs/2411.13503

[55] Hugging Face: Accelerate checkpointing. Accelerate documentation (2026). URL https://huggingface.co/docs/accelerate/v1.13.0/usage_guides/checkpoint

[56] Hugging Face: Accelerate deepspeed. Accelerate documentation (2026). URL https://huggingface.co/docs/accelerate/usage_guides/deepspeed

[57] Hugging Face: Accelerate fully sharded data parallel. Accelerate documentation (2026). URL https://huggingface.co/docs/accelerate/usage_guides/fsdp

[58] Hugging Face: Accelerate gradient accumulation. Accelerate documentation (2026). URL https://huggingface.co/docs/accelerate/v1.13.0/usage_guides/gradient_accumulation

[59] Hugging Face: Accelerate low precision training. Accelerate documentation (2026). URL https://huggingface.co/docs/accelerate/main/usage_guides/low_precision_training

[60] Hugging Face: Accelerator reference. Accelerate documentation (2026). URL https://huggingface.co/docs/accelerate/main/package_reference/accelerator

[61] Hugging Face: Assisted decoding. Transformers documentation (2026). URL https://huggingface.co/docs/transformers/assisted_decoding

[62] Hugging Face: Cache Strategies. Transformers documentation (2026). URL https://huggingface.co/docs/transformers/en/kv_cache

[63] Hugging Face: Chat templates. Transformers documentation (2026). URL https://huggingface.co/docs/transformers/chat_templating

[64] Hugging Face: Configure Dataset Viewer. Hugging Face Hub documentation (2026). URL https://huggingface.co/docs/hub/datasets-viewer-configure

[65] Hugging Face: Continuous batching. Transformers documentation (2026). URL https://huggingface.co/docs/transformers/main/continuous_batching

[66] Hugging Face: Continuous batching architecture. Transformers documentation (2026). URL https://huggingface.co/docs/transformers/main/continuous_batching_architecture

[67] Hugging Face: Create a dataset card. Datasets documentation (2026). URL https://huggingface.co/docs/datasets/en/dataset_card

[68] Hugging Face: The Datasets cache. Datasets documentation (2026). URL https://huggingface.co/docs/datasets/about_cache

[69] Hugging Face: Datasets data files configuration. Hugging Face Hub documentation (2026). URL https://huggingface.co/docs/hub/datasets-data-files-configuration

[70] Hugging Face: Datasets file names and splits. Hugging Face Hub documentation (2026). URL https://huggingface.co/docs/hub/datasets-file-names-and-splits

[71] Hugging Face: Datasets loading methods. Datasets documentation (2026). URL https://huggingface.co/docs/datasets/package_reference/loading_methods

[72] Hugging Face: Datasets main classes. Datasets documentation (2026). URL https://huggingface.co/docs/datasets/v4.8.3/en/package_reference/main_classes

[73] Hugging Face: Datasets stream. Datasets documentation (2026). URL https://huggingface.co/docs/datasets/stream

[74] Hugging Face: Gated models. Hugging Face Hub documentation (2026). URL https://huggingface.co/docs/hub/en/models-gated

[75] Hugging Face: Gemma 3. Transformers documentation (2026). URL https://huggingface.co/docs/transformers/model_doc/gemma3

[76] Hugging Face: GenerationConfig and text generation. Transformers documentation (2026). URL https://huggingface.co/docs/transformers/main_classes/text_generation

[77] Hugging Face: Hotswapping adapters. PEFT documentation (2026). URL https://huggingface.co/docs/peft/package_reference/hotswap

[78] Hugging Face: Hub model cards. Hugging Face Hub documentation (2026). URL https://huggingface.co/docs/hub/model-cards

[79] Hugging Face: Leaderboards and evaluations. Hugging Face documentation (2026). URL https://huggingface.co/docs/leaderboards/main/index

[80] Hugging Face: Lighteval. Hugging Face documentation (2026). URL https://huggingface.co/docs/lighteval/main/en/index

[81] Hugging Face: Lighteval metrics. Hugging Face documentation (2026). URL https://huggingface.co/docs/lighteval/package_reference/metrics

[82] Hugging Face: Lighteval quick tour. Hugging Face documentation (2026). URL https://huggingface.co/docs/lighteval/main/en/quicktour

[83] Hugging Face: Lighteval results. Hugging Face documentation (2026). URL https://huggingface.co/docs/lighteval/main/en/saving-and-reading-results

[84] Hugging Face: Lighteval tasks. Hugging Face documentation (2026). URL https://huggingface.co/docs/lighteval/package_reference/tasks

[85] Hugging Face: Llama. Transformers documentation (2026). URL https://huggingface.co/docs/transformers/main/model_doc/llama

[86] Hugging Face: Llama4. Transformers documentation (2026). URL https://huggingface.co/docs/transformers/model_doc/llama4

[87] Hugging Face: LoRA. PEFT documentation (2026). URL https://huggingface.co/docs/peft/package_reference/lora

[88] Hugging Face: Mixed Adapter Types. PEFT documentation (2026). URL https://huggingface.co/docs/peft/main/developer_guides/mixed_models

[89] Hugging Face: Model Merging. PEFT documentation (2026). URL https://huggingface.co/docs/peft/v0.19.0/developer_guides/model_merging

[90] Hugging Face: Open LLM Leaderboard FAQ. Hugging Face documentation (2026). URL https://huggingface.co/docs/leaderboards/en/open_llm_leaderboard/faq

[91] Hugging Face: Optimizing inference. Transformers documentation (2026). URL https://huggingface.co/docs/transformers/main/en/llm_optims

[92] Hugging Face: Parameter-Efficient Fine-Tuning. Transformers documentation (2026). URL https://huggingface.co/docs/transformers/peft

[93] Hugging Face: PEFT checkpoint format. PEFT documentation (2026). URL https://huggingface.co/docs/peft/v0.19.0/developer_guides/checkpoint

[94] Hugging Face: Quantization. PEFT documentation (2026). URL https://huggingface.co/docs/peft/en/developer_guides/quantization

[95] Hugging Face: Qwen3. Transformers documentation (2026). URL https://huggingface.co/docs/transformers/model_doc/qwen3

[96] Hugging Face: Repository cards. huggingface_hub documentation (2026). URL https://huggingface.co/docs/huggingface_hub/en/package_reference/cards

[97] Hugging Face: Response parsing. Transformers documentation (2026). URL https://huggingface.co/docs/transformers/chat_response_parsing

[98] Hugging Face: Share a dataset to the Hub. Datasets documentation (2026). URL https://huggingface.co/docs/datasets/upload_dataset

[99] Hugging Face: Share a dataset using the CLI. Datasets documentation (2026). URL https://huggingface.co/docs/datasets/share

[100] Hugging Face: smolagents. smolagents documentation (2026). URL https://huggingface.co/docs/smolagents/index

[101] Hugging Face: smolagents agents reference. smolagents documentation (2026). URL https://huggingface.co/docs/smolagents/en/reference/agents

[102] Hugging Face: Text Generation Inference. TGI documentation (2026). URL https://huggingface.co/docs/text-generation-inference/index

[103] Hugging Face: TGI guidance. TGI documentation (2026). URL https://huggingface.co/docs/text-generation-inference/en/conceptual/guidance

[104] Hugging Face: TGI HTTP API reference. TGI documentation (2026). URL https://huggingface.co/docs/text-generation-inference/reference/api_reference

[105] Hugging Face: TGI launcher arguments. TGI documentation (2026). URL https://huggingface.co/docs/text-generation-inference/reference/launcher

[106] Hugging Face: TGI metrics. TGI documentation (2026). URL https://huggingface.co/docs/text-generation-inference/reference/metrics

[107] Hugging Face: The tokenization pipeline. Tokenizers documentation (2026). URL https://huggingface.co/docs/tokenizers/en/pipeline

[108] Hugging Face: Transformers image processor. Transformers documentation (2026). URL https://huggingface.co/docs/transformers/main_classes/image_processor

[109] Hugging Face: Transformers multimodal chat templates. Transformers documentation (2026). URL https://huggingface.co/docs/transformers/chat_templating_multimodal

[110] Hugging Face: Transformers Qwen3-Omni-MOE. Transformers documentation (2026). URL https://huggingface.co/docs/transformers/main/model_doc/qwen3_omni_moe

[111] Hugging Face: Transformers Qwen3-VL. Transformers documentation (2026). URL https://huggingface.co/docs/transformers/main/model_doc/qwen3_vl

[112] Hugging Face: Transformers tokenizer. Transformers documentation (2026). URL https://huggingface.co/docs/transformers/main_classes/tokenizer

[113] Hugging Face: Transformers trainer. Transformers documentation (2026). URL https://huggingface.co/docs/transformers/main_classes/trainer

[114] Hugging Face: Transformers video processor. Transformers documentation (2026). URL https://huggingface.co/docs/transformers/main_classes/video_processor

[115] Hugging Face: TRL BCOTrainer. TRL documentation (2026). URL https://huggingface.co/docs/trl/bco_trainer

[116] Hugging Face: TRL CPOTrainer. TRL documentation (2026). URL https://huggingface.co/docs/trl/cpo_trainer

[117] Hugging Face: TRL dataset formats and types. TRL documentation (2026). URL https://huggingface.co/docs/trl/dataset_formats

[118] Hugging Face: TRL DPOTrainer. TRL documentation (2026). URL https://huggingface.co/docs/trl/dpo_trainer

[119] Hugging Face: TRL GRPOTrainer. TRL documentation (2026). URL https://huggingface.co/docs/trl/grpo_trainer

[120] Hugging Face: TRL KTOTrainer. TRL documentation (2026). URL https://huggingface.co/docs/trl/kto_trainer

[121] Hugging Face: TRL NashMDTrainer. TRL documentation (2026). URL https://huggingface.co/docs/trl/nash_md_trainer

[122] Hugging Face: TRL OnlineDPOTrainer. TRL documentation (2026). URL https://huggingface.co/docs/trl/online_dpo_trainer

[123] Hugging Face: TRL ORPOTrainer. TRL documentation (2026). URL https://huggingface.co/docs/trl/orpo_trainer

[124] Hugging Face: TRL PPOTrainer. TRL documentation (2026). URL https://huggingface.co/docs/trl/ppo_trainer

[125] Hugging Face: TRL reward functions. TRL documentation (2026). URL https://huggingface.co/docs/trl/rewards

[126] Hugging Face: TRL RewardTrainer. TRL documentation (2026). URL https://huggingface.co/docs/trl/reward_trainer

[127] Hugging Face: TRL RLOOTrainer. TRL documentation (2026). URL https://huggingface.co/docs/trl/rloo_trainer

[128] Hugging Face: TRL SFTTrainer. TRL documentation (2026). URL https://huggingface.co/docs/trl/sft_trainer

[129] Hugging Face: TRL: Transformers reinforcement learning. TRL documentation (2026). URL https://huggingface.co/docs/trl/index

[130] Hugging Face: Upload a dataset for LLM training. Hugging Face Hub documentation (2026). URL https://huggingface.co/docs/hub/datasets-upload-guide-llm

[131] Jimenez, C.E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., Narasimhan, K.: Swe-bench: Can language models resolve real-world github issues? arXiv preprint arXiv:2310.06770 (2023). URL https://arxiv.org/abs/2310.06770

[132] Kaplan, J., McCandlish, S., Henighan, T., Brown, T.B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., Amodei, D.: Scaling laws for neural language models. arXiv preprint arXiv:2001.08361 (2020). URL https://arxiv.org/abs/2001.08361

[133] Karpathy, A.: nanoGPT. https://github.com/karpathy/nanoGPT (2022). GitHub repository

[134] Karpukhin, V., Oguz, B., Min, S., Lewis, P., Wu, L., Edunov, S., Chen, D., Yih, W.t.: Dense passage retrieval for open-domain question answering. arXiv preprint arXiv:2004.04906 (2020). URL https://arxiv.org/abs/2004.04906

[135] Keskar, N.S., McCann, B., Varshney, L.R., Xiong, C., Socher, R.: CTRL: A Conditional Transformer Language Model for Controllable Generation. arXiv preprint arXiv:1909.05858 (2019). URL https://arxiv.org/abs/1909.05858

[136] Kim, M.J., Pertsch, K., Karamcheti, S., Xiao, T., Balakrishna, A., Nair, S., Rafailov, R., Foster, E., Lam, G., Sanketi, P., et al.: Openvla: An open-source vision-language-action model. arXiv preprint arXiv:2406.09246 (2024). URL https://arxiv.org/abs/2406.09246

[137] Kimi Team, Bai, Y., Bao, Y., Charles, Y., Chen, C., Chen, G., Chen, H., Chen, J., Chen, N., Chen, R., et al.: Kimi k2: Open agentic intelligence. arXiv preprint arXiv:2507.20534 (2025). URL https://arxiv.org/abs/2507.20534

[138] Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014). URL https://arxiv.org/abs/1412.6980

[139] Kirchenbauer, J., Geiping, J., Wen, Y., Katz, J., Miers, I., Goldstein, T.: A watermark for large language models. In: International Conference on Machine Learning (2023). URL https://arxiv.org/abs/2301.10226

[140] Kudo, T., Richardson, J.: SentencePiece: A Simple and Language Independent Subword Tokenizer and Detokenizer for Neural Text Processing. arXiv preprint arXiv:1808.06226 (2018). URL https://arxiv.org/abs/1808.06226

[141] Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C.H., Gonzalez, J.E., Zhang, H., Stoica, I.: Efficient memory management for large language model serving with pagedattention. arXiv preprint arXiv:2309.06180 (2023). URL https://arxiv.org/abs/2309.06180

[142] Lambert, N., Pyatkin, V., Morrison, J., Miranda, L.J.V., Lin, B.Y., Chandu, K.R., Dziri, N., Kumar, S., Zick, T., Choi, Y., Smith, N.A., Hajishirzi, H.: Rewardbench: Evaluating reward models for language modeling. arXiv preprint arXiv:2403.13787 (2024). URL https://arxiv.org/abs/2403.13787

[143] Lee, A.N., Hunter, C.J., Ruiz, N.: Platypus: Quick, Cheap, and Powerful Refinement of LLMs. arXiv preprint arXiv:2308.07317 (2023). URL https://arxiv.org/abs/2308.07317

[144] Lee, H., Phatale, S.M.X., Lu, K., Mesnard, T., Bishop, C., Carbune, V., Rastogi, A.: RLAIF: Scaling Reinforcement Learning from Human Feedback with AI Feedback. arXiv preprint arXiv:2309.00267 (2023). URL https://arxiv.org/abs/2309.00267

[145] Lee, K., Ippolito, D., Nystrom, A., Zhang, C., Eck, D., Callison-Burch, C., Carlini, N.: Deduplicating training data makes language models better. Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (2022). URL https://arxiv.org/abs/2107.06499

[146] Leviathan, Y., Kalman, M., Matias, Y.: Fast Inference from Transformers via Speculative Decoding. Proceedings of the 40th International Conference on Machine Learning (2023). URL https://arxiv.org/abs/2211.17192

[147] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Kuttler, H., Lewis, M., Yih, W.t., Rocktaschel, T., et al.: Retrieval-augmented generation for knowledge-intensive nlp tasks. arXiv preprint arXiv:2005.11401 (2020). URL https://arxiv.org/abs/2005.11401

[148] Li, J., Cheng, X., Zhao, W.X., Nie, J.Y., Wen, J.R.: Halueval: A large-scale hallucination evaluation benchmark for large language models. Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (2023). URL https://arxiv.org/abs/2305.11747

[149] Li, J., Li, D., Savarese, S., Hoi, S.: BLIP-2: Bootstrapping Language-Image Pre-training with Frozen Image Encoders and Large Language Models. arXiv preprint arXiv:2301.12597 (2023). URL https://arxiv.org/abs/2301.12597

[150] Li, Y., Du, Y., Zhou, K., Wang, J., Zhao, W.X., Wen, J.R.: Evaluating object hallucination in large vision-language models. arXiv preprint arXiv:2305.10355 (2023). URL https://arxiv.org/abs/2305.10355

[151] Liang, P., Bommasani, R., Lee, T., Tsipras, D., Soylu, D., Yasunaga, M., Zhang, Y., Narayanan, D., Wu, Y., Kumar, A., Newman, B., Yuan, B., Yan, B., Zhang, C., Cosgrove, C., Manning, C.D., Re, C., Acosta-Navas, D., Hudson, D.A., Zelikman, E., Durmus, E., Ladhak, F., Rong, F., Ren, H., Yao, H., Wang, J., Santhanam, K., Orr, L., Zheng, L., Yuksekgonul, M., Suzgun, M., Kim, N., Guha, N., Chatterji, N., Khattab, O., Henderson, P., Huang, Q., Chi, R., Xie, S.M., Santurkar, S., Ganguli, S., Hashimoto, T., Icard, T., Zhang, T., Chaudhary, V., Wang, W., Li, X., Mai, Y., Zhang, Y., Koreeda, Y.: Holistic evaluation of language models. arXiv preprint arXiv:2211.09110 (2022). URL https://arxiv.org/abs/2211.09110

[152] Liang, W., Liu, T.: Large Scale Transformer Model Training with Tensor Parallel. PyTorch Tutorials (2025). URL https://pytorch.org/tutorials/intermediate/TP_tutorial.html

[153] Lightning AI: Lit-LLaMA. https://github.com/Lightning-AI/lit-llama (2023). GitHub repository

[154] Lin, J., Tang, J., Tang, H., Yang, S., Chen, W.M., Wang, W.C., Xiao, G., Dang, X., Gan, C., Han, S.: AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration. arXiv preprint arXiv:2306.00978 (2023). URL https://arxiv.org/abs/2306.00978

[155] Lin, S., Hilton, J., Evans, O.: Truthfulqa: Measuring how models mimic human falsehoods. arXiv preprint arXiv:2109.07958 (2021). URL https://arxiv.org/abs/2109.07958

[156] Liu, H., Li, C., Wu, Q., Lee, Y.J.: Visual instruction tuning. arXiv preprint arXiv:2304.08485 (2023). URL https://arxiv.org/abs/2304.08485

[157] Liu, H., Tam, D., Muqeeth, M., Mohta, J., Huang, T., Bansal, M., Raffel, C.: Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning.

Advances in Neural Information Processing Systems (2022). URL https://arxiv.org/abs/2205.05638

[158] Liu, N.F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., Liang, P.: Lost in the middle: How language models use long contexts. Transactions of the Association for Computational Linguistics (2023). URL https://arxiv.org/abs/2307.03172

[159] Liu, X., Zhu, Y., Gu, J., Lan, Y., Yang, C., Qiao, Y.: Mm-safetybench: A benchmark for safety evaluation of multimodal large language models. arXiv preprint arXiv:2311.17600 (2023). URL https://arxiv.org/abs/2311.17600

[160] Liu, Y., Iter, D., Xu, Y., Wang, S., Xu, R., Zhu, C.: G-Eval: NLG Evaluation using GPT-4 with Better Human Alignment. arXiv preprint arXiv:2303.16634 (2023). URL https://arxiv.org/abs/2303.16634

[161] Loshchilov, I., Hutter, F.: Decoupled weight decay regularization. International Conference on Learning Representations (2019). URL https://arxiv.org/abs/1711.05101

[162] Lu, P., Bansal, H., Xia, T., Liu, J., Li, C., Hajishirzi, H., Cheng, H., Chang, K.W., Galley, M., Gao, J.: Mathvista: Evaluating mathematical reasoning of foundation models in visual contexts. arXiv preprint arXiv:2310.02255 (2023). URL https://arxiv.org/abs/2310.02255

[163] Lyu, Q., Havaldar, S., Stein, A., Zhang, L., Rao, D., Wong, E., Apidianaki, M., Callison-Burch, C.: Faithful chain-of-thought reasoning. arXiv preprint arXiv:2301.13379 (2023). URL https://arxiv.org/abs/2301.13379

[164] Mei, L., Yao, J., Ge, Y., Wang, Y., Bi, B., Cai, Y., Liu, J., Li, M., Li, Z.Z., Zhang, D., et al.: A survey of context engineering for large language models. arXiv preprint arXiv:2507.13334 (2025). URL https://arxiv.org/abs/2507.13334

[165] Meng, Y., Xia, M., Chen, D.: SimPO: Simple Preference Optimization with a Reference-Free Reward. arXiv preprint arXiv:2405.14734 (2024). URL https://arxiv.org/abs/2405.14734

[166] Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D., Ginsburg, B., Houston, M., Kuchaiev, O., Venkatesh, G., Wu, H.: Mixed precision training. arXiv preprint arXiv:1710.03740 (2017). URL https://arxiv.org/abs/1710.03740

[167] Micikevicius, P., Stosic, D., Judd, P., Kamalu, J., Oberman, S., Shoeybi, M., Siu, M., Wu, H., Burgess, N., Ha, S., Grisenthwaite, R., Mellempudi, N., Cornea, M., Heinecke, A., Dubey, P.: FP8 Formats for Deep Learning. arXiv preprint arXiv:2209.05433 (2022). URL https://arxiv.org/abs/2209.05433

[168] Min, S., Krishna, K., Lyu, X., Lewis, M., Yih, W.t., Koh, P.W., Iyyer, M., Zettlemoyer, L., Hajishirzi, H.: Factscore: Fine-grained atomic evaluation of factual precision in long form text generation. arXiv preprint arXiv:2305.14251 (2023). URL https://arxiv.org/abs/2305.14251

[169] MLCommons: AILuminate. MLCommons documentation (2026). URL https://mlcommons.org/ailuminate/

[170] MLCommons: AILuminate safety FAQ. MLCommons documentation (2026). URL https://mlcommons.org/ailuminate/safety-faq/

[171] Model Context Protocol: Model Context Protocol: Elicitation. MCP specification (2025). URL https://modelcontextprotocol.io/specification/2025-11-25/client/elicitation

[172] Model Context Protocol: Model Context Protocol: Resources. MCP specification (2025). URL https://modelcontextprotocol.io/specification/2025-03-26/server/resources

[173] Model Context Protocol: Model Context Protocol: Tools. MCP specification (2025). URL https://modelcontextprotocol.io/specification/2025-06-18/server/tools

[174] Mukherjee, S., Mitra, A., Jawahar, G., Agarwal, S., Palangi, H., Awadallah, A.: Orca: Progressive Learning from Complex Explanation Traces of GPT-4. arXiv preprint arXiv:2306.02707 (2023). URL https://arxiv.org/abs/2306.02707

[175] Nakano, R., Hilton, J., Balaji, S., Wu, J., Ouyang, L., Kim, C., Hesse, C., Jain, S., Kosaraju, V., Saunders, W., et al.: WebGPT: Browser-assisted Question-answering with Human Feedback. arXiv preprint arXiv:2112.09332 (2021). URL https://arxiv.org/abs/2112.09332

[176] National Institute of Standards and Technology: Artificial Intelligence Risk Management Framework (AI RMF 1.0). Tech. rep., National Institute of Standards and Technology (2023). URL https://nvlpubs.nist.gov/nistpubs/ai/NIST.AI.100-1.pdf

[177] National Institute of Standards and Technology: Artificial intelligence risk management framework: Generative artificial intelligence profile. Tech. rep., National Institute of Standards and Technology (2024). URL https://nvlpubs.nist.gov/nistpubs/ai/NIST.AI.600-1.pdf

[178] OpenAI: simple-evals: Lightweight evaluation harnesses for language models. GitHub repository (2024). URL https://github.com/openai/simple-evals

[179] OpenAI: Video generation models as world simulators. Technical report (2024). URL https://openai.com/research/video-generation-models-as-world-simulators

[180] OpenAI: Browsecomp: A simple yet challenging benchmark for browsing agents. arXiv preprint arXiv:2504.12516 (2025). URL https://arxiv.org/abs/2504.12516

[181] OpenAI: Openai o3 and o4-mini system card. Tech. rep., OpenAI (2025). URL https://openai.com/index/o3-o4-mini-system-card/

[182] OpenAI: Our updated preparedness framework. Tech. rep., OpenAI (2025). URL https://openai.com/index/updating-our-preparedness-framework/

[183] OpenAI: Computer Use. OpenAI API documentation (2026). URL https://platform.openai.com/docs/guides/tools-computer-use

[184] OpenAI: Connectors and MCP servers. OpenAI API documentation (2026). URL https://platform.openai.com/docs/guides/tools-remote-mcp

[185] OpenAI: File Search. OpenAI API documentation (2026). URL https://platform.openai.com/docs/guides/tools-file-search/

[186] OpenAI: Function Calling. OpenAI API documentation (2026). URL https://platform.openai.com/docs/guides/function-calling?api-mode=responses

[187] OpenAI: Openai's frontier governance framework. Tech. rep., OpenAI (2026). URL https://openai.com/index/openai-frontier-governance-framework/

[188] OpenAI: Web Search. OpenAI API documentation (2026). URL https://platform.openai.com/docs/guides/tools-web-search?api-mode=responses

[189] Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C.L., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al.: Training language models to follow instructions with human feedback. arXiv preprint arXiv:2203.02155 (2022). URL https://arxiv.org/abs/2203.02155

[190] Peebles, W., Xie, S.: Scalable Diffusion Models with Transformers. Proceedings of the IEEE/CVF International Conference on Computer Vision (2023). URL https://arxiv.org/abs/2212.09748

[191] Peng, B., Quesnelle, J., Fan, H., Shippole, E.: Yarn: Efficient context window extension of large language models. arXiv preprint arXiv:2309.00071 (2023). URL https://arxiv.org/abs/2309.00071

[192] Phan, L., Gatti, A., Han, Z., Li, N., Hu, J., Zhang, H., Zhang, C.B.C., Shaaban, M., Ling, J., Shi, S., Choi, M., et al.: Humanity's last exam. arXiv preprint arXiv:2501.14249 (2025). URL https://arxiv.org/abs/2501.14249

[193] Press, O., Smith, N.A., Lewis, M.: Train short, test long: Attention with linear biases enables input length extrapolation. arXiv preprint arXiv:2108.12409 (2021). URL https://arxiv.org/abs/2108.12409

[194] PyTorch Contributors: Asynchronous Saving with Distributed Checkpoint. PyTorch Tutorials (2026). URL https://docs.pytorch.org/tutorials/recipes/distributed_async_checkpoint_recipe.html

[195] PyTorch Contributors: Distributed Checkpoint – torch.distributed.checkpoint. PyTorch 2.12 documentation (2026). URL https://docs.pytorch.org/docs/2.12/distributed.checkpoint.html

[196] PyTorch Contributors: Pipeline Parallelism. PyTorch 2.12 documentation (2026). URL https://docs.pytorch.org/docs/2.12/distributed.pipelining.html

[197] PyTorch Contributors: Tensor Parallelism – torch.distributed.tensor.parallel. PyTorch 2.12 documentation (2026). URL https://docs.pytorch.org/docs/2.12/distributed.tensor.parallel.html

[198] PyTorch Contributors: torch.distributed.fsdp.fully_shard. PyTorch 2.12 documentation (2026). URL https://docs.pytorch.org/docs/2.12/distributed.fsdp.fully_shard.html

[199] PyTorch Contributors: torch.nn.attention.sdpa_kernel and SDPBackend. PyTorch 2.12 documentation (2026). URL https://docs.pytorch.org/docs/2.12/generated/torch.nn.attention.sdpa_kernel.html

[200] PyTorch Contributors: torch.nn.functional.scaled_dot_product_attention. PyTorch 2.12 documentation (2026). URL https://docs.pytorch.org/docs/2.12/generated/torch.nn.functional.scaled_dot_product_attention.html

[201] PyTorch Contributors: torch.nn.MultiheadAttention. PyTorch 2.12 documentation (2026). URL https://docs.pytorch.org/docs/2.12/generated/torch.nn.MultiheadAttention.html

[202] Qiu, R., Tan, J., Pu, J., Wang, H., Gao, X.S., Sun, F.: A survey on unlearning in large language models. arXiv preprint arXiv:2510.25117 (2025). URL https://arxiv.org/abs/2510.25117

[203] Qwen Team: Qwen3-VL Technical Report. arXiv preprint arXiv:2511.21631 (2025). URL https://arxiv.org/abs/2511.21631

[204] Radford, A., Kim, J.W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., Sutskever, I.: Learning transferable visual models from natural language supervision. Proceedings of the 38th International Conference on Machine Learning (2021). URL https://arxiv.org/abs/2103.00020

[205] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I.: Language models are unsupervised multitask learners. OpenAI technical report (2019). URL https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf

[206] Rafailov, R., Sharma, A., Mitchell, E., Ermon, S., Manning, C.D., Finn, C.: Direct preference optimization: Your language model is secretly a reward model. arXiv preprint arXiv:2305.18290 (2023). URL https://arxiv.org/abs/2305.18290

[207] Rajbhandari, S., Rasley, J., Ruwase, O., He, Y.: ZeRO: Memory Optimizations Toward Training Trillion Parameter Models. Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (2020). URL https://arxiv.org/abs/1910.02054

[208] Rajbhandari, S., Ruwase, O., Rasley, J., Smith, S., He, Y.: ZeRO-Infinity: Breaking the GPU Memory Wall for Extreme Scale Deep Learning. Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (2021). URL https://arxiv.org/abs/2104.07857

[209] Rein, D., Hou, B.L., Stickland, A.C., Petty, J., Pang, R.Y., Dirani, J., Michael, J., Bowman, S.R.: GPQA: A Graduate-Level Google-Proof Q&A Benchmark. arXiv preprint arXiv:2311.12022 (2023). URL https://arxiv.org/abs/2311.12022

[210] Ren, J., Rajbhandari, S., Aminabadi, R.Y., Ruwase, O., Yang, S., Zhang, M., Li, D., He, Y.: ZeRO-Offload: Democratizing Billion-Scale Model Training. USENIX Annual Technical Conference (2021). URL https://arxiv.org/abs/2101.06840

[211] Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Zettlemoyer, L., Cancedda, N., Scialom, T.: Toolformer: Language models can teach themselves to use tools. Advances in Neural Information Processing Systems (2023). URL https://arxiv.org/abs/2302.04761

[212] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017). URL https://arxiv.org/abs/1707.06347

[213] Sennrich, R., Haddow, B., Birch, A.: Neural machine translation of rare words with subword units. arXiv preprint arXiv:1508.07909 (2016). URL https://arxiv.org/abs/1508.07909

[214] Shazeer, N.: Fast Transformer Decoding: One Write-Head is All You Need. arXiv preprint arXiv:1911.02150 (2019). URL https://arxiv.org/abs/1911.02150

[215] Shazeer, N.: GLU Variants Improve Transformer. arXiv preprint arXiv:2002.05202 (2020). URL https://arxiv.org/abs/2002.05202

[216] Shen, T., Wan, X., Chen, T., Zhang, R., Pan, J., Lu, D., Lei, F., Lu, Z., Yang, Y., Cheng, C., et al.: Mammothmoda2: A unified ar-diffusion framework for multimodal understanding and generation. arXiv preprint arXiv:2511.18262 (2025). URL https://arxiv.org/abs/2511.18262

[217] Shi, F., Suzgun, M., Freitag, M., Wang, X., Srivats, S., Vosoughi, S., Chung, H.W., Tay, Y., Ruder, S., Zhou, D., et al.: Language models are multilingual chain-of-thought

reasoners. arXiv preprint arXiv:2210.03057 (2022). URL https://arxiv.org/abs/2210.03057

[218] Shi, W., Ajith, A., Xia, M., Huang, Y., Liu, D., Blevins, T., Chen, D., Zettlemoyer, L.: Detecting pretraining data from large language models. arXiv preprint arXiv:2310.16789 (2023). URL https://arxiv.org/abs/2310.16789

[219] Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., Catanzaro, B.: Megatron-lm: Training multi-billion parameter language models using model parallelism. arXiv preprint arXiv:1909.08053 (2019). URL https://arxiv.org/abs/1909.08053

[220] Snell, C., Lee, J., Xu, K., Kumar, A.: Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters. arXiv preprint arXiv:2408.03314 (2024). URL https://arxiv.org/abs/2408.03314

[221] Srivastava, S.S., Aggarwal, V.: A technical survey of reinforcement learning techniques for large language models. arXiv preprint arXiv:2507.04136 (2025). URL https://arxiv.org/abs/2507.04136

[222] Stanford University: CS336: Language modeling from scratch. Course website (2026). URL https://cs336.stanford.edu/. Accessed 2026-05-20

[223] Stiennon, N., Ouyang, L., Wu, J., Ziegler, D.M., Lowe, R., Voss, C., Radford, A., Amodei, D., Christiano, P.F.: Learning to summarize with human feedback. Advances in Neural Information Processing Systems (2020). URL https://arxiv.org/abs/2009.01325

[224] Su, J., Lu, Y., Pan, S., Murtadha, A., Wen, B., Liu, Y.: RoFormer: Enhanced Transformer with Rotary Position Embedding. arXiv preprint arXiv:2104.09864 (2021). URL https://arxiv.org/abs/2104.09864

[225] Sun, Y., Dong, L., Huang, S., Ma, S., Xia, Y., Xue, J., Wang, J., Wei, F.: Retentive Network: A Successor to Transformer for Large Language Models. arXiv preprint arXiv:2307.08621 (2023). URL https://arxiv.org/abs/2307.08621

[226] Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction, 2 edn. MIT Press (2018). URL http://incompleteideas.net/book/the-book-2nd.html

[227] SWE-bench: SWE-bench Verified dataset card. Hugging Face dataset card (2026). URL https://huggingface.co/datasets/SWE-bench/SWE-bench_Verified

[228] Templeton, A., Conerly, T., Marcus, J., Lindsey, J., Bricken, T., Chen, B., Pearce, A., Citro, C., Ameisen, E., Jones, A., et al.: Scaling Monosemanticity: Extracting Interpretable Features from Claude 3 Sonnet. Transformer Circuits Thread (2024). URL https://transformer-circuits.pub/2024/scaling-monosemanticity/

[229] TorchTitan Contributors: TorchTitan: A PyTorch native platform for training generative AI models. GitHub repository (2026). URL https://github.com/pytorch/torchtitan

[230] Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.A., Lacroix, T., Roziere, B., Goyal, N., Hambro, E., Azhar, F., et al.: LLaMA: Open and Efficient Foundation Language Models. arXiv preprint arXiv:2302.13971 (2023). URL https://arxiv.org/abs/2302.13971

[231] Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al.: Llama 2: Open foundation and fine-tuned chat models. arXiv preprint arXiv:2307.09288 (2023). URL https://arxiv.org/abs/2307.09288

[232] Ud Din, M., Akram, W., Saoud, L.S., Rosell, J., Hussain, I.: Vision language action models in robotic manipulation: A systematic review. arXiv preprint arXiv:2507.10672 (2025). URL https://arxiv.org/abs/2507.10672

[233] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. arXiv preprint arXiv:1706.03762 (2017). URL https://arxiv.org/abs/1706.03762

[234] vLLM Project: Disaggregated prefilling. vLLM documentation (2026). URL https://docs.vllm.ai/en/v0.18.1/features/disagg_prefill/

[235] vLLM Project: vLLM automatic prefix caching. vLLM documentation (2026). URL https://docs.vllm.ai/en/stable/design/prefix_caching/

[236] vLLM Project: vLLM Serve CLI Reference. vLLM documentation (2026). URL https://docs.vllm.ai/en/v0.21.0/cli/serve/

[237] Wan, Z., Feng, X., Wen, M., Mcaleer, S.M., Wen, Y., Zhang, W., Wang, J.: Alphazero-like tree-search can guide large language model decoding and training. In: Proceedings of the 41st International Conference on Machine Learning, *Proceedings of Machine Learning Research*, vol. 235, pp. 49890–49920. PMLR (2024). URL https://proceedings.mlr.press/v235/wan24c.html

[238] Wang, G., Qin, H., Jacobs, S.A., Holmes, C., Rajbhandari, S., Ruwase, O., Yan, F., Yang, L., He, Y.: ZeRO++: Extremely Efficient Collective Communication for Giant Model Training. arXiv preprint arXiv:2306.10209 (2023). URL https://arxiv.org/abs/2306.10209

[239] Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E.H., Narang, S., Chowdhery, A., Zhou, D.: Self-consistency improves chain of thought reasoning in language models. arXiv preprint arXiv:2203.11171 (2022). URL https://arxiv.org/abs/2203.11171

[240] Wang, Y., Kordi, Y., Mishra, S., Liu, A., Smith, N.A., Khashabi, D., Hajishirzi, H.: Self-instruct: Aligning language models with self-generated instructions. arXiv preprint arXiv:2212.10560 (2022). URL https://arxiv.org/abs/2212.10560

[241] Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., Yogatama, D., Bosma, M., Zhou, D., Metzler, D., et al.: Emergent abilities of large language models. Transactions on Machine Learning Research (2022). URL https://arxiv.org/abs/2206.07682

[242] Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E.H., Le, Q.V., Zhou, D.: Chain-of-thought prompting elicits reasoning in large language models. arXiv preprint arXiv:2201.11903 (2022). URL https://arxiv.org/abs/2201.11903

[243] Wen, X., Liu, Z., Zheng, S., Ye, S., Wu, Z., Wang, Y., Xu, Z., Liang, X., Li, J., Miao, Z., et al.: Reinforcement Learning with Verifiable Rewards Implicitly Incentivizes Correct Reasoning in Base LLMs. arXiv preprint arXiv:2506.14245 (2025). URL https://arxiv.org/abs/2506.14245

[244] Wu, X., Huang, C.C.: Introduction to context parallel. PyTorch Tutorials (2025). URL https://pytorch.org/tutorials/prototype/context_parallel.html

[245] Xiao, G., Lin, J., Seznec, M., Wu, H., Demouth, J., Han, S.: Smoothquant: Accurate and efficient post-training quantization for large language models. Proceedings of the

40th International Conference on Machine Learning (2023). URL https://arxiv.org/abs/2211.10438

[246] Xiao, G., Tian, Y., Chen, B., Han, S., Lewis, M.: Efficient streaming language models with attention sinks. International Conference on Learning Representations (2024). URL https://arxiv.org/abs/2309.17453

[247] Xu, C., Guan, S., Greene, D., Kechadi, M.T.: Benchmark data contamination of large language models: A survey. arXiv preprint arXiv:2406.04244 (2024). URL https://arxiv.org/abs/2406.04244

[248] Xu, C., Sun, Q., Zheng, K., Geng, X., Zhao, P., Feng, J., Tao, C., Jiang, D.: Wizardlm: Empowering large language models to follow complex instructions. arXiv preprint arXiv:2304.12244 (2023). URL https://arxiv.org/abs/2304.12244

[249] Xu, J., Guo, Z., Hu, H., Chu, Y., Wang, X., He, J., Wang, Y., Shi, X., He, T., Zhu, X., et al.: Qwen3-Omni Technical Report. arXiv preprint arXiv:2509.17765 (2025). URL https://arxiv.org/abs/2509.17765

[250] Yang, A., Li, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Gao, C., Huang, C., Lv, C., et al.: Qwen3 Technical Report. arXiv preprint arXiv:2505.09388 (2025). URL https://arxiv.org/abs/2505.09388

[251] Yang, A., Xiao, B., Wang, B., Zhang, B., Yin, C., Lv, C., Pan, D., Wang, D., Yan, D., Yang, F., et al.: Baichuan 2: Open large-scale language models. arXiv preprint arXiv:2309.10305 (2023). URL https://arxiv.org/abs/2309.10305

[252] Yang, K., Ko, K., Ryu, M.: Tensor model parallelism tutorial. OSLO documentation (2025). URL https://oslo.eleuther.ai/TUTORIALS/tensor_model_parallelism.html

[253] Yang, L., Tian, Y., Li, B., Zhang, X., Shen, K., Tong, Y., Wang, M.: Mmada: Multimodal large diffusion language models. arXiv preprint arXiv:2505.15809 (2025). URL https://arxiv.org/abs/2505.15809

[254] Yang, Z., Guo, X., Zhang, T., Xu, H., Li, B.: Test-time Scaling of LLMs: A Survey from A Subproblem Structure Perspective. arXiv preprint arXiv:2511.14772 (2025). URL https://arxiv.org/abs/2511.14772

[255] Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T.L., Cao, Y., Narasimhan, K.: Tree of thoughts: Deliberate problem solving with large language models. arXiv preprint arXiv:2305.10601 (2023). URL https://arxiv.org/abs/2305.10601

[256] Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., Cao, Y.: React: Synergizing reasoning and acting in language models. arXiv preprint arXiv:2210.03629 (2023). URL https://arxiv.org/abs/2210.03629

[257] Ye, J., Xie, Z., Zheng, L., Gao, J., Wu, Z., Jiang, X., Li, Z., Kong, L.: Dream 7b: Diffusion large language models. arXiv preprint arXiv:2508.15487 (2025). URL https://arxiv.org/abs/2508.15487

[258] Yoshida, D.: NF4 Isn't Information Theoretically Optimal (and That's Good). arXiv preprint arXiv:2306.06965 (2023). URL https://arxiv.org/abs/2306.06965

[259] Yue, X., Ni, Y., Zhang, K., Zheng, T., Liu, R., Zhang, G., Stevens, S., Jiang, D., Ren, W., Sun, Y., Wei, C., Yu, B., Yuan, R., Sun, R., Yin, M., Zheng, B., Yang, Z., Liu, Y., Huang, W., Sun, H., Su, Y., Chen, W.: Mmmu: A massive multi-discipline multimodal understanding and reasoning benchmark for expert agi. arXiv preprint arXiv:2311.16502 (2023). URL https://arxiv.org/abs/2311.16502

[260] Zelikman, E., Wu, Y., Mu, J., Goodman, N.: Star: Bootstrapping reasoning with reasoning. Advances in Neural Information Processing Systems (2022). URL https://arxiv.org/abs/2203.14465

[261] Zhang, B., Sennrich, R.: Root mean square layer normalization. Advances in Neural Information Processing Systems (2019). URL https://arxiv.org/abs/1910.07467

[262] Zhang, I., Liang, W.: Getting started with devicemesh. PyTorch Tutorials (2025). URL https://pytorch.org/tutorials/recipes/distributed_device_mesh.html

[263] Zhang, Q., Lyu, F., Sun, Z., Wang, L., Zhang, W., Guo, Z., Wang, Y., King, I., Liu, X., Ma, C.: What, how, where, and how well? a survey on test-time scaling in large language models. arXiv preprint arXiv:2503.24235 (2025). URL https://arxiv.org/abs/2503.24235

[264] Zhang, R., Han, J., Liu, C., Gao, P., Zhou, A., Hu, X., Yan, S., Pan, L., Li, H., Qiao, Y.: LLaMA-Adapter: Efficient Fine-tuning of Language Models with Zero-init Attention. arXiv preprint arXiv:2303.16199 (2023). URL https://arxiv.org/abs/2303.16199

[265] Zhang, X., Chen, Y., Hu, S., Xu, Z., Chen, J., Hao, M.K., Han, X., Thai, Z.L., Wang, S., Liu, Z., Sun, M.: ∞bench: Extending long context evaluation beyond 100k tokens. arXiv preprint arXiv:2402.13718 (2024). URL https://arxiv.org/abs/2402.13718

[266] Zhao, S., Zhang, X., Guo, J., Hu, J., Duan, L., Fu, M., Chng, Y.X., Wang, G.H., Chen, Q.G., Xu, Z., et al.: Unified multimodal understanding and generation models: Advances, challenges, and opportunities. arXiv preprint arXiv:2505.02567 (2025). URL https://arxiv.org/abs/2505.02567

[267] Zhao, Y., Gu, A., Varma, R., Luo, L., Huang, C.C., Xu, M., Wright, L., Shojanazeri, H., Ott, M., Shleifer, S., Desmaison, A., Balioglu, C., Damania, P., Nguyen, B., Chauhan, G., Hao, Y., Mathews, A., Li, S.: PyTorch FSDP: Experiences on Scaling Fully Sharded Data Parallel. Proceedings of the VLDB Endowment (2023). URL https://arxiv.org/abs/2304.11277

[268] Zheng, L., Chiang, W.L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E.P., Zhang, H., Gonzalez, J.E., Stoica, I.: Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. arXiv preprint arXiv:2306.05685 (2023). URL https://arxiv.org/abs/2306.05685

[269] Zheng, R., Dou, S., Gao, S., Hua, Y., Shen, W., Wang, B., Liu, Y., Jin, S., Liu, Q., Zhou, Y., et al.: Secrets of RLHF in Large Language Models Part I: PPO. arXiv preprint arXiv:2307.04964 (2023). URL https://arxiv.org/abs/2307.04964

[270] Ziegler, D.M., Stiennon, N., Wu, J., Brown, T.B., Radford, A., Amodei, D., Christiano, P., Irving, G.: Fine-tuning language models from human preferences. arXiv preprint arXiv:1909.08593 (2019). URL https://arxiv.org/abs/1909.08593