Transformer 与 LLM 学习手册

覆盖：Transformer 架构、语言建模目标、预训练与对齐、推理与部署、长上下文、Agent、评测与可解释性。每个知识点包含要点与例题（含答案）。

生成日期：2026-01-18

目录（概览）

学习 Checklist（第一页用于快速打勾）

[ ]	模块	完成日期	[ ]	模块	完成日期
[ ]	0 预备基础		[ ]	1 NLP/LLM 表示与 Tokenization	
[ ]	2 Transformer 核心结构		[ ]	3 语言建模目标与损失	
[ ]	4 数据与语料		[ ]	5 预训练工程与技巧	
[ ]	6 指令微调与后训练		[ ]	7 对齐：RLHF / DPO / RLVR 等	
[ ]	8 推理与解码		[ ]	9 长上下文与记忆机制	
[ ]	10 工具调用与 Agent		[ ]	11 模型压缩与部署	
[ ]	12 评测与可靠性		[ ]	13 可解释性与分析	

使用方式建议：按模块顺序学习，每完成一个模块即在上表打勾，并完成该模块所有例题。

详细目录

0 预备基础

学习目标：理解本模块关键概念，能用自己的话解释，并能完成后续例题。

0.1 数学基础

概述：Transformer/LLM 的核心计算是高维线性变换与概率建模。你需要能熟练做形状推导、理解交叉熵/KL、以及梯度下降为何能工作。

你需要掌握：

- 线性代数：向量/矩阵乘法、广播规则、SVD/特征分解直觉（用于低秩与压缩）。
- 概率统计：条件概率、最大似然、对数似然、KL 散度与交叉熵关系。
- 信息论：熵、困惑度（Perplexity）与 bits-per-token 的含义。

工程提示：

- 训练中常见数值问题（softmax 溢出、梯度爆炸）本质是数学稳定性问题。
- 写任何 attention/并行代码前先写出张量形状（B,S,H）。

常见误区：

- 把 Perplexity 当作跨 tokenizer 的绝对可比指标（它受 token 划分影响）。

例题与答案：

Q1：形状推导：若 X 形状为 (B,S,H)，W_Q 形状为 (H, d_k)，则 Q 的形状是什么？

A1：Q = XW_Q，形状为 (B,S,d_k)。

Q2：交叉熵与 KL：给定真实分布 p、模型分布 q，交叉熵 H(p,q) 与 KL(p||q) 的关系？

A2：H(p,q)=H(p)+KL(p||q)。在 p 固定时，最小化交叉熵等价于最小化 KL。

0.2 深度学习基础

概述：LLM
训练本质上是大规模序列建模的梯度优化。需要理解反向传播、归一化、正则化对稳定性的影响。

你需要掌握：

- 反向传播与计算图：理解每层输出如何影响 loss。
- 归一化：LayerNorm/RMSNorm 的作用与差异；Pre-LN 更易训练深层。
- 正则化：dropout、weight decay、label smoothing 的使用边界。

工程提示：

- 大模型常见：dropout 变小或关闭；weight decay 依数据与规模调参。
- 训练不稳定优先检查：学习率、warmup、混合精度与溢出。

常见误区：

- 误以为“ 加大 batch 一定更好 ”；实际需配合学习率与 schedule。

例题与答案：

Q1：为什么 Pre-LN 往往比 Post-LN 更稳？

A1：Pre-LN 让残差支路的梯度更直接传播，缓解深层梯度消失/爆炸，训练更稳定。

Q2：Weight Decay 与 L2 正则完全等价吗？

A2：对 AdamW 而言不等价：AdamW 将权重衰减与梯度更新解耦，更符合经典 weight decay 的含义。

0.3 工程与系统基础

概述：训练/推理效率通常受显存与通信限制。理解并行、混合精度与性能指标，才能把模型跑起来且跑得快。

你需要掌握：

- GPU 指标：算力(Flops)、显存容量、带宽；通信：NVLink/IB。
- 并行：DP/TP/PP；状态切分：ZeRO/FSDP。
- 性能：训练吞吐 tokens/s；推理分 prefill 与 decode。

工程提示：

- 做性能分析时区分：计算瓶颈 vs 内存带宽瓶颈 vs 通信瓶颈。
- 混合精度：BF16 通常比 FP16 更抗溢出；FP16 需 loss scaling。

常见误区：

- 把“显存占用”只归因于参数；实际激活与 KV cache 往往更大。

例题与答案：

Q1：为什么 attention 的显存/计算会随序列长度近似二次增长？

A1：标准全注意力要计算 QK^T，复杂度与显存约为 O(S^2)。

Q2：推理为什么要区分 prefill 与 decode？

A2：prefill 处理整段输入，计算量大；decode 每步只生成 1 个 token，依赖 KV cache，延迟更敏感。

0.4 软件栈常识

概述：LLM
工程依赖大量成熟组件：训练加速、推理引擎、量化/编译工具链。理解各自定位可减少踩坑。

你需要掌握：

- 训练：PyTorch DDP/FSDP、DeepSpeed、Megatron-LM；算子：FlashAttention。
- 推理：vLLM/TGI/TensorRT-LLM；轻量：llama.cpp。
- 评测/数据：常用脚本与可复现流水线。

工程提示：

- 同一模型在不同引擎上速度差异很大，关键在 KV 管理、批处理与 kernel 融合。
- 上线前做端到端压测：吞吐、P95 延迟、OOM 与稳定性。

常见误区：

- 把实验代码直接搬到线上；缺少监控、限流、回滚机制。

例题与答案：

Q1：FlashAttention 的核心收益是什么？

A1：通过更好的 IO/分块策略减少显存读写与中间张量，显著降低 attention 的内存占用并提高速度。

Q2：为什么 vLLM 的 paged attention 对服务很重要？

A2：它把 KV cache 做分页管理，提升并发与动态批处理下的显存利用率，减少碎片与 OOM。

1 NLP/LLM 表示与 Tokenization

学习目标：理解本模块关键概念，能用自己的话解释，并能完成后续例题。

1.1 文本规范化与预处理

概述：输入文本的“ 细节 ”会显著影响
tokenization、训练分布与安全边界。规范化的目标是降低无意义差异、控制噪声与风险。

你需要掌握：

- Unicode 规范化（NFC/NFKC）、空白与不可见字符处理。
- 多语言与混合脚本：语言识别、编码一致性。
- 特殊内容：代码、公式、表格、URL 的保留与清洗策略。

工程提示：

- 做安全与对齐时必须考虑：不可见字符可用于 prompt 注入与绕过过滤。
- 对话数据要统一模板，否则模型学习到“ 格式偏差 ”。

常见误区：

- 过度清洗导致丢失关键分布（例如代码缩进、标点、日志格式）。

例题与答案：

Q1：为什么需要处理不可见字符（如零宽空格）？

A1：它们可能改变 token 序列或绕过规则/过滤器，造成注入与安全风险。

Q2：NFKC 规范化的潜在风险是什么？

A2：会把某些字符折叠为“ 等价形式 ”，可能改变代码/公式的精确语义，需要按场景选择。

1.2 Tokenization（子词/字节级）

概述：Tokenizer 把文本映射为离散符号序列，决定了训练目标的粒度、序列长度与跨语言鲁棒性。

你需要掌握：

- 子词方法：BPE、WordPiece、Unigram LM（SentencePiece）。
- Byte-level：以字节为基础，减少 OOV，适应噪声与多语言。
- 词表大小：压缩率、速度、泛化之间的权衡；特殊 token 设计。

工程提示：

- Tokenizer 训练要有代表性语料采样；否则会对特定域极度不友好。
- 对中文：子词往往不是“ 字 ”为单位，需要关注 token 长度分布。

常见误区：

- 用 Perplexity 比较不同 tokenizer 的模型优劣（不公平）；建议补充 bits-per-byte 或任务指标。

例题与答案：

Q1：BPE 的核心迭代步骤是什么？

A1：统计相邻符号对的频次，反复合并最频繁的对，直到达到词表大小或停止条件。

Q2：为什么 byte-level tokenization 更鲁棒？

A2：任何文本都可表示为字节序列，避免 OOV；对噪声、emoji、混合语言更稳定。

1.3 表示学习（Embedding 与位置编码）

概述：LLM 通过嵌入把离散 token
转为连续向量，再叠加位置/段落信息。位置编码决定模型如何理解顺序与距离。

你需要掌握：

- Token embedding 与 weight tying（输入 embedding 与输出 softmax 权重共享）。
- 绝对位置（learned/sinusoidal）与相对位置（bias）。
- RoPE/ALiBi：现代 decoder-only LLM 常用的位置方案与外推性质。

工程提示：

- RoPE 外推需要配合长序列训练或缩放；只改公式不训往往不够。
- 注意 mask：causal/padding/prefix-LM 在训练与推理必须一致。

常见误区：

- 把位置编码当作“可随意替换的插件”；不同方案会影响已学到的分布与能力。

例题与答案：

Q1：什么是 weight tying？常见收益是什么？

A1：输入 embedding 与输出层权重共享，减少参数并常带来轻微泛化收益。

Q2：causal mask 的作用是什么？

A2：保证位置 t 的预测只依赖 <t 的 token，防止“偷看未来”。

2 Transformer 核心结构

学习目标：理解本模块关键概念，能用自己的话解释，并能完成后续例题。

2.1 Scaled Dot-Product Attention

概述：注意力是 Transformer 的核心算子：用 Query 去“查询” Key，并对 Value 做加权汇总。缩放与 mask 主要为稳定性与因果约束服务。

你需要掌握：

- 基本公式：Attn(Q,K,V)=softmax(QK^T/sqrt(d_k)+mask) V。
- 缩放 sqrt(d_k)：避免点积随维度增大导致 softmax 饱和。
- mask：padding mask 与 causal mask；实现中注意广播与 dtype。

工程提示：

- 数值稳定：softmax 计算时减去行最大值；FP16 下更关键。
- 训练/推理注意 mask 一致，否则会出现“训练时能看未来”的泄露。

常见误区：

- 把 attention 权重当作解释因果关系的唯一证据（注意力不是解释）。

例题与答案：

Q1：为什么要除以 sqrt(d_k)？

A1：不缩放时点积方差随 d_k 增大，softmax 更容易饱和，梯度变小，训练不稳定。

Q2：若使用 causal mask，位置 t 可以关注哪些位置？

A2：只能关注 0..t 的历史位置（含自身），不能关注 >t 的未来 token。

2.2 Multi-Head Attention（多头）

概述：多头把表示投影到多个子空间并行建模不同关系（语法、指代、主题等），再拼接回主空间。

你需要掌握：

- hidden size H，head 数 h，每头维度 d=H/h（常见）。
- 每头独立 W_Q/W_K/W_V；输出再经 W_O 融合。
- 头冗余与剪枝：部分头可被移除而损失不大。

工程提示：

- 现代 LLM 常用 GQA/MQA：减少 K/V 头数以省显存与带宽。
- head 数影响 kernel 实现效率（对齐到 warp/向量化）。

常见误区：

- 误以为 head 越多越好；过多会增加开销且收益递减。

例题与答案：

Q1：H=4096，h=32，则每头维度 d？

A1：d=4096/32=128。

Q2：GQA 相比标准 MHA 的主要节省来自哪里？

A2：减少 K/V 的头数（组共享），从而降低 KV cache 与带宽开销。

2.3 前馈网络（MLP/FFN）

概述：Attention 负责信息路由，MLP 提供逐位置的非线性变换与容量。现代 LLM 常用门控 FFN 提升表达效率。

你需要掌握：

- 经典 FFN：Linear(H->4H) + 激活 + Linear(4H->H)。
- 门控变体：SwiGLU/GeGLU（两支路相乘或门控）。
- 激活：GELU/SiLU 的经验优势。

工程提示：

- FFN 通常占参数大头；kernel 融合（fused MLP）对速度影响大。
- 门控 FFN 维度系数需与实现对齐（如 2/3*4H 等常见配置）。

常见误区：

- 忽视 FFN 的带宽瓶颈；很多场景 attention 已被 FlashAttention 优化后，FFN 反而成瓶颈。

例题与答案：

Q1：FFN 的扩展维度为什么常取 4H？

A1：经验上在计算/参数预算下提供较好容量；门控 FFN 会使用不同系数但同样遵循“先扩再压”的思路。

Q2：SwiGLU 的直觉优势？

A2：门控让网络在不同输入下选择性通过信息，提升表达与训练稳定性。

2.4 残差、归一化与结构变体

概述：残差连接保证梯度流动，归一化控制数值尺度。Pre-LN 结构是现代深层 LLM 的默认选择之一。

你需要掌握：

- Residual：y = x + f(LN(x))（Pre-LN 典型形式）。
- LayerNorm vs RMSNorm：RMSNorm 省去均值项，常更快。
- Dropout：大模型常降低甚至关闭，取决于数据与过拟合风险。

工程提示：

- 深层训练不稳优先检查：是否 Pre-LN、是否有足够 warmup、norm/softmax 的数值稳定。
- RMSNorm 与权重初始化/学习率有耦合，迁移配置需小心。

常见误区：

- 把 Post-LN 直接套在深层 decoder-only 上，导致训练发散。

例题与答案：

Q1：Residual 的直接收益是什么？

A1：让梯度能绕过子层直接传播，减轻深层网络优化困难。

Q2：RMSNorm 与 LayerNorm 的主要差异？

A2：RMSNorm 只按均方根缩放，不减去均值；LayerNorm 既减均值又按方差缩放。

2.5 Transformer 形态：Encoder/Decoder/Seq2Seq

概述：不同结构对应不同任务：理解类任务偏 encoder，生成类任务偏 decoder；Seq2Seq 用于条件生成。

你需要掌握：

- Encoder-only：双向注意力（如 BERT），适合分类/抽取。
- Decoder-only：因果注意力（GPT），LLM 主流。
- Encoder-Decoder：输入编码+输出解码（T5），翻译/摘要强。

工程提示：

- 选择结构时看任务：是否需要条件输入、是否需要开放式生成。
- 同一数据格式在不同结构下的 mask 设计不同。

常见误区：

- 把 BERT 目标直接用于对话生成而不改结构/目标，导致生成能力差。

例题与答案：

Q1：为什么 decoder-only 更适合开放式生成？

A1：它直接优化 p(x_t|x_{<t}) 的自回归目标，训练与推理一致。

Q2：Seq2Seq 的优势场景？

A2：输入与输出分离的条件生成，如翻译、摘要、信息抽取到结构化输出。

2.6 参数效率与结构演进（KV/GQA/MoE/FlashAttention）

概述：现代 LLM 在不改变基本 Transformer 思路下，通过注意力实现与稀疏/共享策略获得更高效率。

你需要掌握：

- KV cache：推理阶段缓存历史 K/V，避免重复计算。
- GQA/MQA：减少 K/V 头数，降低 KV cache 成本。
- FlashAttention：分块计算减少中间张量与 IO。
- MoE：稀疏激活专家网络，提升参数规模而控制计算。

工程提示：

- MoE 关键：router 负载均衡、通信（all-to-all）、专家并行。
- FlashAttention/GQA 会改变性能瓶颈分布，需要整体 profiling。

常见误区：

- 把 MoE 当作“无成本变大模型”；实际上训练复杂、路由不稳会明显降质。

例题与答案：

Q1：KV cache 为什么能显著降低 decode 阶段复杂度？

A1：decode 时只需为新 token 计算 Q，并与缓存 K/V 做注意力；无需重复生成历史 K/V。

Q2：MoE 的主要风险是什么？

A2：路由崩塌（只用少数专家）、负载不均导致通信与显存问题，以及训练不稳定。

3 语言建模目标与损失

学习目标：理解本模块关键概念，能用自己的话解释，并能完成后续例题。

3.1 自回归语言建模（Causal LM）

概述：LLM 的主流目标：最大化序列概率 $p(x)=\prod p(x_t|x_{<t})$。训练用 teacher forcing，推理用逐步生成。

你需要掌握：

- 损失：token-level 交叉熵；常用忽略 padding 的 mask。
- 指标：Perplexity；但跨 tokenizer 比较要谨慎。
- Exposure bias：训练见到真实前缀，推理见到模型前缀。

工程提示：

- 训练时常做 sequence packing 减少 padding，提高吞吐。
- 对话训练要注意系统/用户/助手的 loss mask（是否对用户部分计算 loss）。

常见误区：

- 把困惑度当作对话质量的充分指标；对话更依赖指令遵循与偏好。

例题与答案：

Q1：给定 logits z，softmax 概率 p，真实 token y 的交叉熵是什么？

A1：$L = -\log p_y$（对每个位置求和/平均）。

Q2：为什么推理时错误会“滚雪球”？

A2：推理输入包含模型先前输出，若前面偏离，后续条件分布改变，误差逐步累积。

3.2 Masked LM / Denoising 目标

概述：MLM 与去噪目标用遮盖/扰动学习双向上下文，常用于理解或 Seq2Seq 预训练。

你需要掌握：

- MLM：随机 mask token 预测；需要 special [MASK] 或替换策略。
- Span corruption：遮盖连续片段并生成（T5）。
- 去噪自编码：删除/打乱/替换等噪声。

工程提示：

- MLM 与生成任务存在目标差异；要生成通常需要改结构或再训练。
- Span corruption 更接近生成，迁移到生成任务更自然。

常见误区：

- 认为 MLM 预训练的模型直接就能像 GPT 那样顺畅长文本生成。

例题与答案：

Q1：MLM 与 Causal LM 的最大差异？

A1：MLM 允许双向上下文预测被 mask 的位置；Causal LM 只能用左侧上下文自回归预测。

Q2：为什么 T5 的 span corruption 更适合 Seq2Seq？

A2：输入是被遮盖的序列，输出是被遮盖片段的生成，天然是条件生成形式。

3.3 Seq2Seq 条件生成损失

概述：在给定输入 x 的条件下生成输出 y，优化 p(y|x)。常见于翻译、摘要、结构化抽取。

你需要掌握：

- Teacher forcing：用真实 y_<t 作为解码器输入。
- 可加长度惩罚/coverage 机制（经典 NMT）。
- 任务指标：BLEU/ROUGE 等，但要结合人工评审。

工程提示：

- 注意训练/推理差异：beam search 与 sampling 的选择要匹配产品目标。
- 结构化任务建议使用约束解码或 schema 校验。

常见误区：

- 用 ROUGE 作为唯一指标优化，导致模型学会堆砌关键词。

例题与答案：

Q1：为什么 Seq2Seq 常用 cross-attention？

A1：解码器需要在生成时读取输入编码表示，用 cross-attention 实现条件依赖。

Q2：Teacher forcing 的副作用？

A2：推理时无法用真实历史 token，造成 exposure bias；可用 scheduled sampling 等缓解但不常用于 LLM。

3.4 稳定性与正则化技巧（z-loss/label smoothing 等）

概述：大规模训练容易出现 logits 失控、过拟合或不稳定。适当正则与稳定项可显著降低发散风险。

你需要掌握：

- Label smoothing：把 one-hot 变为平滑分布，降低过拟合。
- z-loss/logit regularization：惩罚 logits 过大，帮助数值稳定。
- MoE 的负载均衡损失：避免路由塌缩。

工程提示：

- 正则不是越多越好：对齐/对话任务中过强会损害可控性。
- 观察信号：logit 范围、梯度范数、NaN/Inf 频率。

常见误区：

- 把 NaN 归因于“模型太大”；很多时候是学习率、混合精度或坏数据。

例题与答案：

Q1：Label smoothing 可能带来什么副作用？

A1：可能降低概率校准或影响“尖锐分布”任务（如某些分类/抽取），需评估。

Q2：什么是 logit overflow？如何缓解？

A2：logits 过大导致 softmax 溢出/饱和；可用更稳定实现、z-loss、降低 LR、使用 BF16 等。

4 数据与语料

学习目标：理解本模块关键概念，能用自己的话解释，并能完成后续例题。

4.1 数据来源谱系与覆盖

概述：预训练能力上限很大程度由数据决定：覆盖的领域越多、质量越高，模型越“通用”。

你需要掌握：

- Web、书籍、论文、代码、对话、百科等来源的差异与偏置。
- 私域数据：企业知识、工单、FAQ；必须考虑权限与合规。
- 多语种：语言比例与采样策略会影响跨语能力。

工程提示：

- 建立数据台账（来源、许可、时间、处理流程），便于审计audit 与回溯。
- 把数据按域分桶，便于做配比与质量诊断。

常见误区：

- 只追求“量”，忽视高质量数据对收敛与能力的边际收益。

例题与答案：

Q1：为什么代码数据会显著影响模型的推理/结构化能力？

A1：代码包含严格语法、长程依赖与可验证反馈，能强化结构化生成与部分逻辑能力。

Q2：私域数据加入预训练前必须先做什么？

A2：权限与合规评估（许可/PII/敏感信息）、脱敏处理与可追溯记录。

4.2 清洗、去重与质量过滤

概述：去重与清洗的目标是提高“有效 token”，减少训练把算力花在重复/垃圾样本上。

你需要掌握：

- 去重：exact 与 near-duplicate（MinHash/SimHash 等）。
- 质量过滤：语言识别、困惑度过滤、启发式规则（重复度/标点/长度）。
- 有害内容与 PII：检测、掩码、删除与审计。

工程提示：

- 去重粒度：文档级/段落级/行级；代码与自然语言策略不同。
- 过滤阈值需做 A/B：过强会丢分布、过弱会引入噪声。

常见误区：

- 把困惑度过滤当作万能质量标准；它会偏好“像训练分布”的文本，可能丢掉新域。

例题与答案：

Q1：near-duplicate 去重为什么重要？

A1：重复样本会让模型过拟合常见片段，浪费计算，并增加记忆与版权风险。

Q2：PII 处理的基本原则？

A2：最小化收集与保留；能删除就删除，必要时脱敏，并记录处理链路以便审计。

4.3 数据配比、采样温度与课程学习

概述：不同域的 token 贡献不同。合理配比与采样能在有限预算下提升综合能力与特定能力。

你需要掌握：

- 域混合：通用/代码/数学/多语种比例。
- 温度采样：对小域过采样，避免被大域淹没。
- 课程学习：从干净到复杂，或先学结构化（代码/数学）再学开放域。

工程提示：

- 配比要用可解释的目标：例如提升代码 pass@1，或提升特定语言表现。
- 避免“过度偏域”：会导致通用能力回退。

常见误区：

- 把“训练集占比”直接等同于“能力占比”；不同域的学习曲线不同。

例题与答案：

Q1：温度采样的直觉是什么？

A1：把大域的权重压低、小域抬高，使模型在训练中看到更多稀缺但重要的样本。

Q2：课程学习可能的风险？

A2：若后期加入大量噪声或风格差异太大，可能造成能力波动或遗忘，需要混入锚定数据。

4.4 数据格式、packing 与对话模板

概述：数据如何被拼成序列决定了 mask、loss 与训练效率。对话模型尤其依赖一致模板。

你需要掌握：

- EOD（文档结束）token 的使用与文档边界。
- sequence packing：把多个短样本拼到同一序列减少 padding。
- 对话格式：system/user/assistant 角色 token、轮次边界、是否对用户 token 计算 loss。

工程提示：

- packing 要确保 attention mask 正确，不同样本之间不可互相注意（除非设计为连续文档）。
- 对话训练常见：只对 assistant 回答计算 loss，提升指令遵循一致性。

常见误区：

- packing 时忘了插入分隔符或 mask，导致样本串扰与训练泄露。

例题与答案：

Q1：为什么 packing 能显著提升训练吞吐？

A1：减少 padding token 的计算与通信开销，提高有效 token 比例。

Q2：对话训练为什么常只对 assistant 部分算 loss？

A2：目标是学习“如何回答”，若对用户部分也算 loss，可能学到复述用户输入等不良行为。

5 预训练工程与技巧

学习目标：理解本模块关键概念，能用自己的话解释，并能完成后续例题。

5.1 规模法则与预算规划

概述：预训练通常受计算预算约束。规模法则用于估算在给定算力下参数量/数据量的合理配比。

你需要掌握：

- 参数 N、数据 D、计算 C 的经验关系（用于规划，不是定律）。
- 数据不足会早早过拟合；参数不足会欠拟合。
- 小规模试跑：验证稳定性与数据管线，再上大规模。

工程提示：

- 预算拆分：超参探索/消融/正式训练/续训与对齐。
- 建立自动化报表：loss、吞吐、显存、评测集趋势。

常见误区：

- 直接用他人规模配置照搬到不同数据/架构上。

例题与答案：

Q1：为什么需要小规模试跑？

A1：提前发现数据管线、数值稳定与并行配置问题，避免大规模浪费算力。

Q2：数据不足的典型信号？

A2：训练 loss 继续下降但验证困惑度不再改善或变差，且出现过拟合迹象。

5.2 优化器与学习率策略

概述：LLM 训练对学习率非常敏感。warmup + 衰减是标准做法，配合梯度裁剪与累积保证稳定。

你需要掌握：

- AdamW：beta、epsilon、weight decay 的作用。
- LR schedule：warmup + cosine/linear decay。
- 梯度裁剪、梯度累积、batch size 与有效学习率。

工程提示：

- warmup 不足常导致早期发散；过长会浪费训练。
- 大 batch 常需要更大的 LR，但需验证；不要机械套用线性缩放。

常见误区：

- 把 loss spike 当作正常波动；很多 spike 是坏 batch 或溢出前兆。

例题与答案：

Q1：为什么要 warmup？

A1：初期参数未稳定，梯度大且方差高，warmup 逐步提高 LR 可避免发散。

Q2：梯度裁剪有什么作用？

A2：限制梯度范数，防止个别 batch 触发梯度爆炸导致数值崩溃。

5.3 混合精度与数值稳定

概述：混合精度提升吞吐，但会带来溢出/舍入误差。需要掌握 BF16/FP16 的差异与稳定手段。

你需要掌握：

- BF16：指数位多，抗溢出强；FP16：精度更高但易溢出。
- 动态 loss scaling（FP16 常用）。
- 关键算子：softmax、layernorm 的稳定实现。

工程提示：

- 优先 BF16（若硬件支持）；FP16 必须监控 overflow/NaN。
- 日志中记录：grad norm、loss scale、inf/nan 计数。

常见误区：

- 认为使用 AMP 就自动安全；实际仍需正确的 loss scaling 与稳定算子。

例题与答案：

Q1：FP16 为什么更容易溢出？

A1：指数范围更小，表示不了大数；训练中 logits/梯度可能超范围。

Q2：softmax 为什么要减 max？

A2：避免 exp(大数) 溢出，同时不改变 softmax 结果（等价变换）。

5.4 分布式训练并行（DP/TP/PP/ZeRO/FSDP）

概述：单卡无法容纳与训练大模型，需要并行。理解各并行维度与通信模式是工程落地的关键。

你需要掌握：

- 数据并行（DP）：复制模型，分 batch；all-reduce 梯度。
- 张量并行（TP）：切分矩阵乘法；频繁 all-reduce/all-gather。
- 流水并行（PP）：按层切分；需处理 pipeline bubble。
- ZeRO/FSDP：切分参数/梯度/优化器状态。

工程提示：

- 并行策略取决于：模型大小、序列长度、网络带宽、batch 配置。
- 性能优化：计算与通信重叠、减少小通信、梯度累积。

常见误区：

- 只追求可跑通，忽视通信瓶颈，导致扩卡后吞吐不增反降。

例题与答案：

Q1：为什么 DP 需要 all-reduce？

A1：各卡计算不同样本的梯度，需要同步求平均以保持等效于大 batch 训练。

Q2：PP 的 bubble 是什么？如何缓解？

A2：流水线开始/结束阶段部分设备空闲；可用 micro-batch 切分、1F1B 调度等减小空闲。

5.5 内存与算子优化（checkpoint/FlashAttention/fused）

概述：大模型训练往往首先 OOM。通过激活重计算、内核融合与高效注意力可以在不降 batch 的情况下训练更大模型。

你需要掌握：

- Activation checkpointing：省显存但增加算力。
- FlashAttention：attention 省显存并提速。
- fused kernels：融合 LN/MLP 等减少内存读写。

工程提示：

- 先做 profiling 再优化：确定瓶颈是 attention、FFN 还是通信。
- 注意 checkpoint 粒度：过细 overhead 大，过粗省不够。

常见误区：

- 认为“加 checkpoint 一定更快”；它通常是拿算力换显存。

例题与答案：

Q1：Activation checkpointing 的代价是什么？

A1：反向传播时需要重算前向激活，增加计算时间。

Q2：为什么 fused kernel 会更快？

A2：减少中间张量的读写与 kernel launch 开销，提高算子利用率。

5.6 训练监控、调试与故障处理

概述：LLM 训练失败成本极高。必须建立从数据到数值到系统的监控与快速定位机制。

你需要掌握：

- 监控：loss、学习率、grad norm、吞吐、显存、通信时间。
- 异常：loss spike、NaN/Inf、吞吐突降、OOM。
- 定位：坏 batch、数据解码错误、并行死锁、硬件 ECC 错误。

工程提示：

- 为每次实验保存：代码版本、配置、数据版本、随机种子、环境信息。
- checkpoint 与断点续训策略：频率、保留窗口、验证一致性。

常见误区：

- 只看训练 loss 不看验证与任务评测，导致训练方向偏离目标。

例题与答案：

Q1：出现 NaN 的第一批排查项？

A1：学习率/梯度裁剪、混合精度溢出、坏数据（极长样本/异常字符）、softmax/LN 稳定实现。

Q2：吞吐突然下降常见原因？

A2：数据加载瓶颈、通信拥塞、频繁 checkpoint、GPU 频率降档或作业抢占等。

5.7 续训、领域继续预训练与长上下文扩展

概述：预训练不是一次性工程。常见需求包括继续预训练、迁移到新域、扩展上下文长度。

你需要掌握：

- DAPT/TAPT：领域/任务继续预训练。
- 长上下文：位置编码缩放 + 长序列数据继续训练。
- 能力回归：续训后需要回归评测（通用能力、对齐表现）。

工程提示：

- 续训数据要有台账与可解释目标；避免把模型带偏。
- 长上下文扩展必须做“ 长序列任务 ”训练与评测。

常见误区：

- 只增加最大长度配置而不训，期望模型自动具备长上下文能力。

例题与答案：

Q1：DAPT 与 TAPT 的差异？

A1：DAPT 面向领域数据（如金融/医疗），TAPT 面向特定任务数据分布（如客服对话）。

Q2：为什么长上下文扩展要回归评测？

A2：位置与训练分布改变可能导致短上下文与对齐能力回退。

6 指令微调（SFT）与后训练（Post-training）

学习目标：理解本模块关键概念，能用自己的话解释，并能完成后续例题。

6.1 SFT 数据：质量、覆盖与模板

概述：SFT 的目标是让模型更好地遵循指令与对话规范。数据质量与模板一致性通常比规模更重要。

你需要掌握：

- 数据形态：单轮指令-回答、多轮对话、工具调用格式示例。
- 模板：system/user/assistant 角色分隔；统一分隔符与停止 token。
- 质量：明确目标、可验证答案、避免含糊套话；覆盖边界条件。

工程提示：

- 建立数据审查：随机抽样人工复核 + 自动规则（重复、长度、敏感）。
- 避免“ 模板泄露 ”：把评测提示混入训练会导致过拟合。

常见误区：

- 把低质量合成数据大量堆入 SFT，导致模型“ 会说话但不做事 ”。

例题与答案：

Q1：多轮对话数据为什么需要严格模板？

A1：模型会学习格式作为行为线索；模板不一致会导致角色混淆与输出不稳定。

Q2：SFT 中常见的 loss mask 策略？

A2：只对 assistant 输出部分计算 loss，用户输入不计入，以强化回答生成而非复述。

6.2 SFT 训练策略：全参 vs PEFT（LoRA/QLoRA）

概述：SFT
资源成本低于预训练，但仍需控制过拟合与遗忘。参数高效微调可在有限算力下快速迭代。

你需要掌握：

- 全参微调：效果强但成本高、易遗忘；需小 LR 与稳定 schedule。
- LoRA：在部分线性层插入低秩适配；可合并权重部署。
- QLoRA：4-bit 权重量化 + LoRA，显著降显存。
- 层选择：注意力投影/MLP 等模块的取舍。

工程提示：

- PEFT 适合快速试验与多租户定制；最终可视需求做全参收敛。
- 避免过拟合：早停、混入少量通用数据、正则与数据增广。

常见误区：

- 把 LoRA rank 设得很大当作“ 更强 ”；会增加训练不稳与部署成本。

例题与答案：

Q1：LoRA 的核心思想？

A1：冻结原权重 W，仅训练低秩更新 $\Delta W=AB$（rank r），以较小参数实现适配。

Q2：QLoRA 的关键工程收益？

A2：把基座权重以 4-bit 存储并在计算中解量化，显著降低显存，使单卡可微调更大模型。

6.3 后训练方向：安全风格、工具格式与领域适配

概述：Post-training 通常指在 SFT
之后，进一步强化安全、工具调用、长上下文与领域一致性等产品属性。

你需要掌握：

- 安全风格：拒答策略、风险提示、合规边界。
- 工具调用：JSON/schema 严格性、参数填充、错误处理。
- 长上下文：长序列指令与检索任务；领域术语与格式。

工程提示：

- 分阶段：先指令能力、再偏好/安全、再专项（工具/长上下文/领域）。
- 每阶段都做回归评测，避免能力漂移。

常见误区：

- 把安全/拒答数据与普通指令混在一起无区分地训练，导致过度拒答。

例题与答案：

Q1：为什么工具调用需要专项数据？

A1：它要求严格结构化输出与状态机式行为，普通对话数据难以覆盖格式与错误处理细节。

Q2：后训练为什么要分阶段？

A2：不同目标（遵循、偏好、安全、工具）可能冲突；分阶段更易控制权衡并定位回归。

7 对齐（Alignment）：RLHF / DPO / RLVR 等

学习目标：理解本模块关键概念，能用自己的话解释，并能完成后续例题。

7.1 对齐的目标与约束（Helpful/Harmless/Honest）

概述：对齐的本质是在多目标之间做权衡：有用、无害、诚实，并遵守指令层级与合规要求。

你需要掌握：

- 指令层级：system > developer > user；冲突时遵从高优先级。
- 诚实：不确定时表达不确定、避免编造；可引用依据。
- 无害：拒绝违法/危险请求，提供安全替代方案。

工程提示：

- 对齐评测要覆盖：正常任务、边界任务、对抗越狱、提示注入。
- 注意 over-refusal：过度拒答会损害可用性。

常见误区：

- 把“对齐=拒答”，导致模型在安全与可用性上失衡。

例题与答案：

Q1：什么是 instruction hierarchy？

A1：系统指令优先级最高，其次开发者，再次用户；模型应在冲突时遵守高优先级。

Q2：诚实对齐的一个可操作准则？

A2：对无法确定的事实明确说明不确定性，并建议可验证途径（检索、咨询专家等）。

7.2 RLHF：三阶段与 PPO 关键点

概述：经典 RLHF：SFT 初始化策略，训练奖励模型，再用 PPO 在 KL 约束下优化策略以符合人类偏好。

你需要掌握：

- 偏好数据：同一提示下输出 A/B 的比较标注。
- Reward Model：学习打分；需防止过拟合与分布外失真。
- PPO：clip 目标、优势函数；KL penalty 控制偏离参考模型。

工程提示：

- 工程难点：on-policy 采样成本、训练不稳定、reward hacking。
- 要监控：平均奖励、KL、长度分布、拒答率与安全指标。

常见误区：

- 把 RM 当作“真实价值函数”；RM 有偏差，会被策略钻空子。

例题与答案：

Q1：RLHF 中 KL penalty 的作用？

A1：限制新策略与参考策略差异，提升稳定性并防止过度优化导致退化。

Q2：什么是 reward hacking？

A2：模型学会利用奖励模型漏洞取得高分，而非真实符合人类偏好（例如堆砌安全套话）。

7.3 DPO：直接偏好优化

概述：DPO 用成对偏好直接更新策略，绕开显式 RM 与 PPO，训练更简单且常更稳定。

你需要掌握：

- 数据：chosen 与 rejected 成对样本。
- 超参 β：控制偏好强度，相当于 KL 权衡。
- 常见变体：IPO、KTO、ORPO 等（不同正则与目标形式）。

工程提示：

- 偏好数据质量极关键：错标会强烈写入模型行为。
- 对拒答偏好要细分场景，避免把“谨慎”训练成“全拒”。

常见误区：

- 误以为 DPO 一定优于 RLHF；两者取决于数据与目标，且可组合。

例题与答案：

Q1：β 过大可能导致什么现象？

A1：模型过度追随偏好，输出变得极端或多样性下降，并可能牺牲通用能力。

Q2：为什么 DPO 更易落地？

A2：无需单独训练 RM 与在线 PPO 回路，训练管线更短、稳定性通常更好。

7.4 RLVR：可验证奖励的强化学习

概述：RLVR 把奖励建立在可自动验证的信号上（如单元测试、数学答案检查），特别适合代码/数学等可验证任务。

你需要掌握：

- 奖励来自：测试通过率、编译成功、答案匹配、约束满足等。
- 优势：减少主观标注、降低奖励噪声。
- 难点：指标覆盖面有限，可能“教会应试”。

工程提示：

- 用多样化测试与对抗样例减少投机取巧。
- 结合偏好/安全约束，避免只追求可测指标导致行为退化。

常见误区：

- 只优化可测指标，导致输出可读性差或泛化差。

例题与答案：

Q1：为什么 RLVR 在代码领域有效？

A1：代码可以通过编译/测试自动验证，奖励信号相对客观且可规模化。

Q2：如何缓解“应试”风险？

A2：增加隐藏测试、多分布测试、对抗生成，并评估分布外任务表现。

7.5 RLAIF / Constitutional AI（模型辅助对齐）

概述：通过 AI 生成偏好数据、批注或基于“宪法规则”自我批评，降低人力标注成本并提高一致性。

你需要掌握：

- RLAIF：用强模型或规则产生偏好/评分。
- Constitutional AI：基于规则集进行自我批评与修订。
- 风险：偏差自举、回音室效应、规则覆盖不足。

工程提示：

- 对 AI 标注必须做抽样人工校验与对抗测试。
- 规则要可操作、可审计，并与产品政策一致。

常见误区：

- 把 AI 生成的偏好当作“真理”，忽视其系统性偏差。

例题与答案：

Q1：RLAIF 的主要优点？

A1：可规模化生成偏好数据，降低成本，并可能保持标注一致性。

Q2：Constitutional 方法的关键前提？

A2：规则（宪法）要清晰、可执行，并能覆盖关键风险场景。

7.6 对齐评测与红队（Jailbreak/Injection/Over-refusal）

概述：没有系统评测的对齐不可控。红队测试用于发现越狱、注入与信息泄露等高风险问题。

你需要掌握：

- 越狱：角色扮演、分步诱导、编码混淆等。
- 提示注入：把恶意指令藏在网页/文档/工具返回中。
- 指标：拒答率、误拒率、成功越狱率、敏感信息泄露率。

工程提示：

- 建立分层防线：输入净化、工具权限最小化、输出安全分类器与审计日志。
- 持续回归：模型/工具/检索升级都要重新红队。

常见误区：

- 只做一次性红队；上线后分布变化会让风险快速回归。

例题与答案：

Q1：什么是 prompt injection？

A1：外部内容（网页/文档/工具输出）携带指令试图覆盖系统/开发者约束，诱导模型泄露或违规行动。

Q2：over-refusal 的典型表现？

A2：对正常、无害请求也频繁拒答或给出过多安全套话，导致可用性下降。

8 推理（Inference）与解码（Decoding）

学习目标：理解本模块关键概念，能用自己的话解释，并能完成后续例题。

8.1 Prefill vs Decode：推理阶段拆分

概述：LLM 推理分为 prefill（处理输入）与 decode（逐 token 生成）。两者瓶颈不同，优化策略也不同。

你需要掌握：

- Prefill：对整段 prompt 做前向，attention 仍是 O(S^2)。
- Decode：每步生成 1 token，依赖 KV cache，关键是延迟与并发。
- 指标：TTFT（首 token 时间）、TPOT（每 token 时间）、吞吐 tokens/s。

工程提示：

- 产品侧常关心 TTFT 与 P95 延迟；批处理会提升吞吐但增加 TTFT。
- 对长 prompt，prefill 成本可能主导总时延。

常见误区：

- 只优化 decode（如 KV），忽视长 prompt 的 prefill 成本。

例题与答案：

Q1：为什么 TTFT 往往由 prefill 主导？

A1：首 token 生成前必须完成对 prompt 的计算；prompt 越长，prefill 越慢。

Q2：如何降低长 prompt 的 prefill 成本？

A2：压缩/裁剪 prompt、检索后精简上下文、使用高效 attention 或分块策略。

8.2 KV Cache：机制、显存与优化

概述：KV cache 缓存每层历史 K/V，使 decode 阶段避免重复计算。它也是推理显存的主要来源之一。

你需要掌握：

- 缓存维度约为：layers × (K,V) × batch × heads × seq_len × head_dim。
- GQA/MQA：减少 KV 头数；KV quantization：降低 cache 精度。
- Paged KV 管理：减少碎片，支持连续批处理。

工程提示：

- 在高并发服务中，KV 管理策略决定能否稳定不 OOM。
- 多轮对话可复用前缀 KV（prompt cache），但要考虑个性化与安全隔离。

常见误区：

- 忽视 KV cache 的增长，导致长上下文或高并发下频繁 OOM。

例题与答案：

Q1：KV cache 为什么随 seq_len 线性增长？

A1：每增加 1 个 token，就要为每层存一份 K/V 向量，因此是 O(S)。

Q2：GQA/MQA 对质量可能的影响？

A2：减少 KV 头数可能降低表示能力；需要在速度/显存与质量之间做实验权衡。

8.3 解码策略：Greedy/Beam/Sampling

概述：解码是把概率分布变成具体文本的策略选择。不同策略在确定性、创造性与重复性上差异显著。

你需要掌握：

- Greedy：每步取最大概率；稳定但可能保守。
- Beam search：探索多条路径；对开放式对话可能变得刻板。
- Sampling：temperature、top-k、top-p（nucleus）、typical sampling。

工程提示：

- 对话系统常用 top-p + 适度 temperature；严谨任务（抽取/格式）更偏低温或贪心。
- 加入重复惩罚与 stop tokens 控制啰嗦与跑题。

常见误区：

- 把 temperature 调高当作“更聪明”；过高会增加胡言乱语与幻觉。

例题与答案：

Q1：top-p 的定义？

A1：选择累计概率达到 p 的最小 token 集合并在其中采样。

Q2：beam search 为什么可能降低对话自然度？

A2：它偏向高概率、通用、安全的表达，容易产生模板化与冗长重复。

8.4 约束解码与结构化输出（JSON/schema/grammar）

概述：当输出必须满足格式（JSON、函数参数、DSL），需要约束解码或事后校验与纠错。

你需要掌握：

- JSON schema：字段类型、必填项、枚举；输出后做验证。
- Grammar-constrained decoding：在解码过程中限制 token 合法性。
- logit bias：对特定 token 加偏置；stop sequence 管理。

工程提示：

- 严格约束会提高失败率：建议“约束 + 自动修复（repair）+ 重试”。
- 工具调用应当设计幂等与超时，避免模型重复触发副作用。

常见误区：

- 只做事后正则修复，忽视模型分布；导致大量不可修复输出。

例题与答案：

Q1：为什么 grammar-constrained decoding 更可靠？

A1：它在生成过程中就排除了非法路径，避免生成后再修补的不可控复杂度。

Q2：结构化输出常见失败模式？

A2：漏字段、类型错误、尾逗号、未闭合括号；需验证+修复+重试策略。

8.5 推理服务工程：批处理、流式与推测解码

概述：真实系统需要在吞吐、延迟与成本之间做取舍，并通过批处理与新型解码加速提升性价比。

你需要掌握：

- 连续批处理（continuous batching）：动态合并请求。
- 流式输出：改善体验但增加系统复杂度。
- 推测解码（speculative decoding）：小模型草稿 + 大模型校验。

工程提示：

- 高并发下要做限流与队列；对关键用户可做优先级。
- 推测解码收益取决于 draft 模型质量与验证成本。

常见误区：

- 只用离线 tokens/s 指标评估服务，忽视 P95 延迟与尾部抖动。

例题与答案：

Q1：连续批处理的核心收益？

A1：提高 GPU 利用率，把多个请求合并计算，提升吞吐并降低单位 token 成本。

Q2：推测解码何时收益最大？

A2：当 draft 模型能较准确预测大模型输出，且验证开销较小、并发足够时。

9 长上下文与记忆机制

学习目标：理解本模块关键概念，能用自己的话解释，并能完成后续例题。

9.1 长上下文瓶颈：O(S^2) 与 lost-in-the-middle

概述：长上下文不仅是算力问题，也是使用能力问题：模型可能“看得见但用不好”。

你需要掌握：

- 标准 attention 计算/显存 O(S^2)，KV cache O(S)。
- 中间遗忘（lost-in-the-middle）：信息在中部更难被利用。
- 长 prompt 导致 prefill 成本主导。

工程提示：

- 先从系统设计减负：检索后精简、摘要、分块读取。
- 用针对性评测（needle、multi-hop）验证“可用性”。

常见误区：

- 把 max_length 拉长就以为模型会更会用长文。

例题与答案：

Q1：什么是 lost-in-the-middle？

A1：模型更容易利用开头/结尾信息，中间信息利用率下降，导致检索失败或引用错误。

Q2：长上下文性能下降的两个来源？

A2：计算/显存瓶颈，以及模型在训练中缺少长序列任务导致的能力不足。

9.2 位置编码外推与扩展（RoPE/ALiBi/缩放）

概述：要让模型在训练长度之外工作，需要位置编码外推策略并配合长序列继续训练。

你需要掌握：

- RoPE：旋转位置编码；外推常用缩放/插值策略。
- ALiBi：用线性偏置实现距离衰减，外推较自然。
- 关键：只改位置编码不续训，效果通常有限。

工程提示：

- 扩展后做回归：短序列能力、对齐、安全都可能漂移。
- 长序列数据要包含：跨段推理、检索、引用与一致性任务。

常见误区：

- 在不变训练数据的情况下强行外推到很长，导致输出幻觉与引用混乱。

例题与答案：

Q1：为什么 RoPE 外推会出问题？

A1：训练只见过有限长度的相位分布，超出范围的相位组合在模型参数中未充分学习。

Q2：位置插值/缩放的目标？

A2：把更长的实际位置映射到训练见过的相位范围内，减少分布漂移。

9.3 高效注意力：局部/稀疏/分块与线性注意力

概述：为了突破 O(S^2)，可用局部窗口、块稀疏或分层结构；但通常需要训练适配。

你需要掌握：

- Sliding window：只关注邻近窗口。
- Block sparse：块级稀疏 pattern。
- 层次注意力：先段内聚合再段间交互。
- 线性注意力：用核技巧近似（质量与稳定性需验证）。

工程提示：

- 高效 attention 通常改变归纳偏置；要用匹配的训练任务与评测。
- 对检索任务，常结合全局 token 或跨块 summary token。

常见误区：

- 直接替换 attention 实现而不续训，期望质量不变。

例题与答案：

Q1：局部注意力的主要代价？

A1：削弱长距离依赖建模；需要额外机制（全局 token/跨块汇总）补偿。

Q2：为什么稀疏 pattern 要慎选？

A2：pattern 决定模型可见信息路径；不合适会造成信息断裂与能力退化。

9.4 记忆机制：RAG、摘要记忆与 KV 压缩

概述：超出上下文窗口的“ 记忆 ”通常靠外部存储（检索）或压缩（摘要/结构化）。

你需要掌握：

- RAG：chunking、embedding、向量检索、重排序、引用。
- 摘要记忆：把对话历史压缩为要点，降低 token 成本。
- 结构化记忆：事件/偏好/事实的表结构存储，便于检索与更新。
- KV 压缩：合并 token、重要性采样等。

工程提示：

- RAG 关键在：切分粒度、召回率与精确率、rerank、引用一致性。
- 记忆要有权限与隐私策略：可撤回、可审计、最小化。

常见误区：

- 把 RAG 当作“自动消除幻觉”；检索质量差会放大错误。

例题与答案：

Q1：RAG 的两阶段检索常见组成？

A1：向量召回（高召回）+ rerank（高精确），再把最相关片段拼入上下文。

Q2：摘要记忆的风险？

A2：摘要可能丢关键信息或引入错误，且错误会被反复放大；需可追溯与必要时回看原文。

9.5 长上下文评测：needle、多跳与引用准确性

概述：没有评测就无法判断长上下文是否真正可用。评测应覆盖检索、推理与一致性。

你需要掌握：

- Needle-in-a-haystack：把关键信息埋入长文测试定位能力。
- 多跳检索：跨段组合信息回答。
- 引用准确性：回答是否能对应到给定片段，减少编造。

工程提示：

- 评测集要接近真实业务：文档格式、噪声、长度分布。
- 区分：模型没检索到 vs 检索到了但没用好。

常见误区：

- 只用一个 needle 测试就宣称“支持 128k”；缺少多场景验证。

例题与答案：

Q1：needle 测试的关键控制变量？

A1：needle 位置（开头/中间/结尾）、上下文长度、干扰文本强度。

Q2：引用准确性为什么重要？

A2：它能约束模型把答案绑定到证据，降低幻觉与责任风险。

10 工具调用与 Agent

学习目标：理解本模块关键概念，能用自己的话解释，并能完成后续例题。

10.1 工具/函数调用基础（schema 与选择策略）

概述：工具调用让模型把语言能力扩展为可执行能力。关键是清晰接口、严格 schema 与可靠的调用策略。

你需要掌握：

- 工具定义：函数名、参数 schema、返回结构、错误码。
- 何时调用工具：信息不足、需要实时数据、需要执行/计算。
- 输出格式：严格 JSON；字段校验与容错。

工程提示：

- 工具返回应视为不可信输入：防注入、脱敏、权限控制。
- 对关键动作要二次确认或使用安全策略（例如只读工具）。

常见误区：

- 让模型直接拼 SQL/命令在生产环境执行而无沙箱与审计。

例题与答案：

Q1：为什么需要 schema 校验？

A1：防止模型输出半结构化文本导致解析失败，提升工具调用成功率与可观测性。

Q2：何时不应调用工具？

A2：当用户仅需解释/写作且不依赖外部实时信息时；无意义调用会增加延迟与成本。

10.2 Agent 范式：ReAct 与 Plan-and-Execute

概述：Agent 是“多步策略 + 工具”组合。常见范式包括 ReAct（边想边做）与计划-执行分离。

你需要掌握：

- ReAct：思考-行动-观察循环；适合探索性任务。
- Plan-and-Execute：先规划子任务，再逐步执行；更可控。
- 多 Agent：角色分工（检索/规划/审阅/执行）。

工程提示：

- 为 Agent 设计停止条件与失败回退：避免无限循环。
- 把工具成本纳入策略（例如检索次数上限）。

常见误区：

- 只看单次回答质量，不记录轨迹；导致无法调试与回归。

例题与答案：

Q1：ReAct 的关键优势？

A1：把中间观察（工具结果）纳入后续决策，提升复杂任务完成率。

Q2：Plan-and-Execute 的主要风险？

A2：计划可能在信息不足时不准确；需要执行中动态修正与重新规划。

10.3 Agent 关键组件：Planner/Memory/Executor/Critic

概述：可用的 Agent 需要模块化：规划、记忆、执行与审阅。每个模块都要可观测、可测试。

你需要掌握：

- Planner：任务分解、依赖、优先级、停止条件。
- Memory：短期上下文、长期记忆（向量库/结构化）。
- Executor：工具调用、重试、超时、幂等。
- Critic/Verifier：自检、事实核验、约束验证。

工程提示：

- 引入 verifier 可显著降低幻觉：例如用检索核对引用，用单元测试核对代码。
- 日志要记录：输入、工具调用参数、返回、决策与耗时。

常见误区：

- 把所有逻辑塞进 prompt；可维护性差且难以评测。

例题与答案：

Q1：幂等性对工具调用为何重要？

A1：模型可能重试同一调用；幂等可避免重复扣费/重复写入等副作用。

Q2：Verifier 的一个简单实现？

A2：对关键事实做检索核对；对代码运行单元测试；对 JSON 做 schema 验证。

10.4 安全：Prompt Injection、最小权限与沙箱

概述：Agent 的风险主要来自工具链：模型可能被外部内容注入指令，或在高权限工具上误操作。

你需要掌握：

- Prompt injection：外部文档/网页携带恶意指令。
- 最小权限：只给必要工具与必要范围的数据。
- 沙箱：代码执行隔离、网络/文件权限控制。
- 审计：全链路日志、可回放。

工程提示：

- 对外部文本做“指令去激活”：把工具输出当数据而非指令。
- 对高风险动作做策略门控（policy gate）。

常见误区：

- 让检索到的网页内容直接进入 system 指令区域，导致越权。

例题与答案：

Q1：如何降低 prompt injection 风险？

A1：隔离指令与数据、对工具输出做净化、限制工具权限、对关键动作二次确认与审计。

Q2：为什么要最小权限？

A2：即使模型被诱导，也能把潜在伤害限制在最小范围。

10.5 Agent 评测与可观测性（trace/回归）

概述：Agent 的质量不能只看最终答案，需要评估轨迹、工具调用成本与失败模式。

你需要掌握：

- 指标：任务完成率、平均步数、工具调用次数/成本、P95 延迟。
- 可观测性：trace（每一步输入/输出/工具结果）、错误分类。
- 回归：工具升级、模型升级、知识库更新都要跑评测。

工程提示：

- 为关键任务建立“金标准轨迹”或可验证结果（tests）。
- 把失败样本沉淀为训练数据或规则修复。

常见误区：

- 只做人工 spot-check；上线后问题难复现且无法定位。

例题与答案：

Q1：为什么需要 trace？

A1：它能定位失败发生在哪一步（检索、解析、调用、推理），并支持复盘与改进。

Q2：Agent 的线上指标为何重要？

A2：离线评测无法覆盖真实分布与工具故障；线上指标能及时发现回归与风险。

11 模型压缩与部署

学习目标：理解本模块关键概念，能用自己的话解释，并能完成后续例题。

11.1 量化（PTQ/QAT/KV 量化）

概述：量化通过降低数值精度减少显存与带宽，提升推理吞吐并降低成本，是部署的常见手段。

你需要掌握：

- PTQ：后训练量化（INT8/INT4，GPTQ/AWQ 等）。
- QAT：量化感知训练，质量更好但成本更高。
- KV 量化：降低 KV cache 精度，特别利于长上下文。

工程提示：

- 量化需要校准数据（代表性很重要）。
- 对工具/JSON 任务要重点测试：量化可能更易产生格式错误。

常见误区：

- 只看困惑度变化不看任务；量化对结构化输出的影响可能更大。

例题与答案：

Q1：PTQ 与 QAT 的差异？

A1：PTQ 不再训练模型，靠校准直接量化；QAT 在训练中模拟量化误差以适配。

Q2：KV 量化的收益主要体现在哪？

A2：降低长序列与高并发下 KV cache 的显存与带宽成本。

11.2 剪枝与稀疏化（结构化/非结构化）

概述：剪枝通过移除冗余参数降低计算，但只有在结构与内核支持下才能转化为真实加速。

你需要掌握：

- 非结构化稀疏：随机权重置零，需专用内核/硬件支持。
- 结构化剪枝：剪通道/剪头/剪层，更易获得实际速度提升。
- 注意力头剪枝：常用于分析与轻量化。

工程提示：

- 剪枝后通常需要再训练/蒸馏恢复质量。
- 先从结构化剪枝入手更容易落地。

常见误区：

- 把稀疏率当作速度提升的线性指标；实际受内核与访存限制。

例题与答案：

Q1：为什么非结构化稀疏不一定加速？

A1：如果硬件/内核不能利用稀疏结构，仍会按密集矩阵计算。

Q2：结构化剪枝的优势？

A2：改变张量形状，能直接减少矩阵乘法规模，更容易获得真实加速。

11.3 蒸馏（logits/序列级/偏好蒸馏）

概述：蒸馏用大模型指导小模型学习，在成本与质量之间取得更优折中，适合边缘部署或高 QPS 场景。

你需要掌握：

- Logits 蒸馏：匹配教师 soft targets。
- 序列级蒸馏：用教师生成的输出作为训练目标。
- 偏好蒸馏：把对齐后的偏好行为迁移给小模型。

工程提示：

- 蒸馏数据要覆盖真实使用分布；否则小模型只会“背模板”。
- 对工具/结构化任务，蒸馏可显著提升格式稳定性。

常见误区：

- 只蒸馏开放式对话，忽略安全与拒答策略，导致线上风险。

例题与答案：

Q1：为什么 soft targets 有用？

A1：教师分布包含类间相似性信息，给学生提供更丰富梯度信号。

Q2：序列级蒸馏的风险？

A2：学生可能过拟合教师的特定表达方式，降低多样性与鲁棒性。

11.4 部署与服务化：并发、监控、灰度与安全

概述：部署不仅是把模型跑起来，更要保证可用性、稳定性、成本与合规。

你需要掌握：

- 加载与分片：权重 sharding、并行推理。
- 服务指标：QPS、P50/P95、TTFT、tokens/s、OOM、错误率。
- 灰度发布：AB、回滚；审计与日志脱敏。

工程提示：

- 把评测纳入 CI：每次模型/引擎升级都自动跑回归。
- 生产必须有限流与配额，防止雪崩。

常见误区：

- 没有监控就上线；问题发生时无法定位是模型、数据还是系统。

例题与答案：

Q1：为什么需要灰度发布？

A1：新版本可能引入质量或安全回归；灰度可限制影响范围并支持快速回滚。

Q2：日志为何必须脱敏？

A2：日志可能包含用户隐私与敏感信息；脱敏与最小化存储是合规与安全要求。

12 评测与可靠性

学习目标：理解本模块关键概念，能用自己的话解释，并能完成后续例题。

12.1 能力评测维度与基准设计

概述：评测应覆盖语言、推理、代码、工具、长上下文与安全，并与业务目标对齐。

你需要掌握：

- 理解/生成：摘要、改写、抽取。
- 推理：数学、逻辑、多跳问答。
- 代码：pass@k、单测通过率。
- 工具：schema 正确率、任务完成率。
- 长上下文：needle、多文档引用。

工程提示：

- 基准要做数据隔离：训练/评测严格分离，避免泄露。
- 用小而精的回归集监控版本漂移。

常见误区：

- 把公开榜单分数当作唯一目标，忽略业务分布与安全。

例题与答案：

Q1：为什么需要任务分解式评测？

A1：单一总分掩盖失败模式；分维度才能定位问题并指导数据/训练。

Q2：评测泄露的常见来源？

A2：把 benchmark 提示或答案混入训练语料；或评测集与训练集近重复未去重。

12.2 离线与在线指标（质量、成本、稳定性）

概述：离线指标适合迭代与对比，在线指标反映真实用户分布。两者都不可缺。

你需要掌握：

- 离线：准确率、pass@k、人工评审；对话可用偏好对比。
- 在线：会话成功率、升级人工率、拒答率、用户满意度。
- 系统：延迟、吞吐、成本、错误率。

工程提示：

- 用 guardrail 监控安全事件与敏感输出率。
- 把评测与监控自动化，形成闭环。

常见误区：

- 只做离线评测；上线后分布漂移导致质量与风险失控。

例题与答案：

Q1：为什么在线指标可能与离线不一致？

A1：用户分布、提示方式、工具故障与系统延迟等因素离线难以复现。

Q2：如何监控模型回归？

A2：固定回归集 + 线上 A/B + 关键指标报警（拒答率、错误率、敏感输出）。

12.3 幻觉（Hallucination）与事实性

概述：幻觉是生成模型在不确定时仍给出看似合理答案的倾向。降低幻觉需要证据约束与不确定性表达。

你需要掌握：

- 类型：编造事实、错引来源、时间性错误、过度自信。
- 缓解：检索+引用、事实核验工具、拒答阈值、logprob 校准。
- 训练：加入“引用式回答”、不确定性标注、反幻觉偏好。

工程提示：

- RAG 不等于无幻觉：检索错误/拼接错误会引入新幻觉。
- 对关键场景使用 verifier（例如二次检索核对）。

常见误区：

- 把模型输出当作事实来源，缺少引用与可追溯。

例题与答案：

Q1：为什么模型会产生幻觉？

A1：目标是预测下一个 token 的高概率序列，而非保证事实正确；在缺证据时也会生成“像真的”。

Q2：引用机制如何降低幻觉？

A2：要求答案与检索证据对应，使模型倾向于从证据中抽取/改写而非编造。

12.4 安全、偏见与隐私评测

概述：可靠系统必须评估有害内容、越狱鲁棒性、偏见与隐私泄露，并持续回归。

你需要掌握：

- 安全类别：暴力、自伤、仇恨、成人、非法行为等。
- 越狱/注入：对抗提示、编码混淆、工具链注入。
- 隐私：PII 泄露、训练数据记忆、成员推断风险。

工程提示：

- 建立红队脚本与自动化扫描；对高风险输出做人工复核。
- 隐私与合规：最小化保留、用户可撤回、日志脱敏。

常见误区：

- 只评测“明显违法”内容，忽视灰色地带与间接指导的风险。

例题与答案：

Q1：什么是训练数据记忆风险？

A1：模型可能复现训练语料中的敏感片段（如 PII 或受版权保护文本）。

Q2：偏见评测为什么难？

A2：偏见具有语境性与多维度；需要多样提示、分群体统计与人工审查结合。

13 可解释性与分析

学习目标：理解本模块关键概念，能用自己的话解释，并能完成后续例题。

13.1 黑盒分析：敏感性、校准与失效模式

概述：黑盒分析从输入-输出行为入手：测试提示扰动、稳定性与不确定性表达，快速定位问题类别。

你需要掌握：

- 提示敏感性：同义改写、格式变化、顺序置换。
- 稳定性：temperature=0 的一致性、重复回答偏差。
- 校准：logprob 与错误相关性、拒答阈值。

工程提示：

- 用 system prompt 约束风格后仍要评测：模型可能绕过或忽视。
- 记录失败样本并分类：事实、推理、工具、对齐、格式。

常见误区：

- 用少量示例得出泛化结论；需要系统化扰动与统计。

例题与答案：

Q1：什么是 prompt sensitivity？

A1：对提示微小变化输出大幅波动的现象，反映模型边界与对齐稳定性不足。

Q2：校准良好的模型有什么特征？

A2：高置信输出更可能正确；低置信时更愿意承认不确定或请求更多信息。

13.2 白盒分析：Attention/Probe/Logit lens

概述：白盒分析查看内部表示与激活，帮助理解模块贡献与潜在可剪枝结构，但需要谨慎解释。

你需要掌握：

- Attention 可视化：头关注位置分布。
- Probing：用探针模型测试表示是否包含某属性。
- Logit lens：看各层残差流对最终 logits 的贡献。

工程提示：

- 注意力权重不等于因果贡献；建议用干预方法验证。
- 探针结果受探针容量与数据集影响，需要对照实验。

常见误区：

- 把 attention heatmap 直接当作“解释”，忽视因果检验。

例题与答案：

Q1：为什么说 attention 不是解释？

A1：高注意力不必然代表对输出有因果影响；需要干预（patching/ablation）验证。

Q2：logit lens 能回答什么问题？

A2：某一层的表示在输出空间上更倾向于哪些 token，帮助理解层间形成答案的过程。

13.3 机制可解释性：因果追踪与特征分解（SAE）

概述：机制可解释性关注可重复、可干预的“电路”级结构，用因果方法验证哪些激活真正驱动能力。

你需要掌握：

- Activation patching：替换/注入激活观察输出变化。
- Causal tracing：定位关键信息从哪层/哪位置传递。
- SAE：稀疏自编码器提取可解释特征。

工程提示：

- 干预要做对照：随机 patch、层/位置对比，避免误判。
- 特征操控（steering）要评估副作用与分布外风险。

常见误区：

- 把单次干预结论泛化到所有任务；电路常是任务相关的。

例题与答案：

Q1：activation patching 的基本步骤？

A1：在干净输入与对照输入之间替换特定层/位置的激活，观察输出指标变化以推断因果贡献。

Q2：SAE 的目标是什么？

A2：用稀疏表示分解激活为可解释特征，便于定位与操控模型内部概念。

13.4 训练数据记忆、隐私与 unlearning

概述：模型可能记住训练语料中的稀有片段。需要评估记忆风险并在必要时做遗忘（unlearning）或数据治理。

你需要掌握：

- 记忆检测：重复生成测试、相似度匹配、成员推断评估。
- 隐私保护：PII 清洗、最小化日志、访问控制。
- Unlearning：针对特定数据/概念减弱记忆（实现复杂且可能伤害能力）。

工程提示：

- 优先从数据治理解决：源头减少敏感语料比事后 unlearning 更可靠。
- 对外部引用要合规，避免复现受版权保护文本。

常见误区：

- 用简单“黑名单词”当作隐私方案；无法覆盖变体与上下文。

例题与答案：

Q1：为什么 unlearning 很难？

A1：参数共享导致知识纠缠，删除某段记忆可能影响相关能力；且难以验证完全遗忘。

Q2：降低记忆风险的首要手段？

A2：训练前数据去重与 PII 清理，以及对敏感域设置严格过滤与审计。

13.5 对齐与安全的可解释性：拒答触发与对齐税

概述：对齐后模型行为变化需要可解释：什么时候拒答、为何拒答、是否牺牲了有用性（alignment tax）。

你需要掌握：

- 拒答触发：是策略层还是模型内化？不同路径的可控性不同。
- 越狱路径分析：指令冲突、角色注入、工具链注入。
- 对齐税：安全增强导致能力回退的量化与定位。

工程提示：

- 用分层评测拆解：安全合规、可用性、事实性、长上下文、工具能力。
- 对齐与能力冲突时，考虑分模型路由或分场景策略。

常见误区：

- 把能力回退全归因于“模型变笨”；很多是目标权重改变。

例题与答案：

Q1：如何量化 alignment tax？

A1：在相同基准与评测设置下对比对齐前后模型的能力指标变化，并分任务维度分析。

Q2：拒答触发定位的一个方法？

A2：对比加入/移除策略层（如安全分类器）输出差异，并结合激活分析或偏好数据溯源。