补充新题

Transformer 基础

- MHA、多头注意力

Attention 运算公式

为什么要除以 √□□

什么是多头注意力（MHA）

- 位置编码

位置编码的作用及主要实现方式

绝对位置编码 vs 相对位置编码：结构与异同

RoPE（Rotary Position Embedding）结构与原理

RoPE 相对于绝对编码的优势：为什么外推性更好？

RoPE 如何兼顾绝对与相对位置信息？

答案：

> “我们对 QKTQK^TQKT 除以 dk\sqrt{d_k}dk，是因为 dkd_kdk 越大，点积值越大，softmax 输出接近 one-hot，梯度几乎为 0。
>
> 缩放后能保持打分在合理区间，避免训练不稳定。”
>
> Multi-Head Attention 的核心思路是并行计算多组注意力，每组关注输入的不同关系模式。
>
> 具体做法是把 Q,K,VQ,K,VQ,K,V 投影到 hhh 个子空间，各自做 scaled dot-product attention，再把各头输出 concat 并线性映射作为最终表示。
>
> 这样能让模型同时关注多种关联特征，提升表达能力。”
>
> - **Injecting Position Info**
> - “Transformer 自注意力本身对序列位置不敏感，因此需要注入位置信息。主流做法包括：绝对位置编码（sinusoidal 或 learnable）、相对位置编码（distance bias 或 mapping）、以及如 Alibi、RoPE 等其他方法。”
> - **绝对 vs 相对位置编码**
> - “绝对编码直接把位置信息加到 token embedding，结构简单；相对编码则将距离信息加到 attention 打分或映射矩阵中，能灵活捕捉任意相对距离，对长序列泛化性更好，但设计与计算更复杂。”
> - **RoPE 结构**

- “RoPE 不是把编码向量相加，而是将 embedding 的每对维度按位置 θ 做 2D 旋转，点积时隐式产生相对距离差分，从而原生兼顾绝对和相对位置。”
- **RoPE 的外推性优势**
- “RoPE 的外推性来自位置→θ 的线性映射，不需要预存固定长度编码；对任意 pos 均可计算旋转矩阵，天然支持更长序列。”
- **RoPE 如何兼顾绝对与相对**
- “一次旋转既编码了绝对位置（θ 与 pos 一一对应），又在 Q·K 运算里体现了相对差分（θ_j–θ_i），无需额外偏置或矩阵。”

检索增强与 RAG

- RAG 的完整工作流与关键组件（对 RAG 的理解；关键词检索与重排细节；为什么用 RAG；召回效果；RAG vs SFT 优势）
- 相关模型结构（BGE 模型；如何做召回；BGE 架构要点）

答案：

对 RAG 的理解及作用

> “RAG 架构由两部分组成：第一步是用检索器（如 BM25 或 dense retriever）从知识库中选出 k 条相关文档；第二步将这些文档与原始查询拼接送入生成模型（如 BART 或 T5）进行回答。这样做能显著提升模型在开放领域问答或知识密集型任务中的准确性，并减轻模型参数存储压力。”

RAG 的完整工作流2

> “RAG 的完整流程包含：
>
> 252. 用 Encoder 将用户查询编码；
> 253. 在知识库中检索 top-k 相关文档；
> 254. 可选地对这些文档做一次交叉编码器重排；
> 255. 将检索到的文档与原始查询拼接，送入 Seq2Seq 模型生成回答。
>
> 该流程实现了‘先检索后生成’，提升了生成质量和知识覆盖率。”

为什么要用 RAG

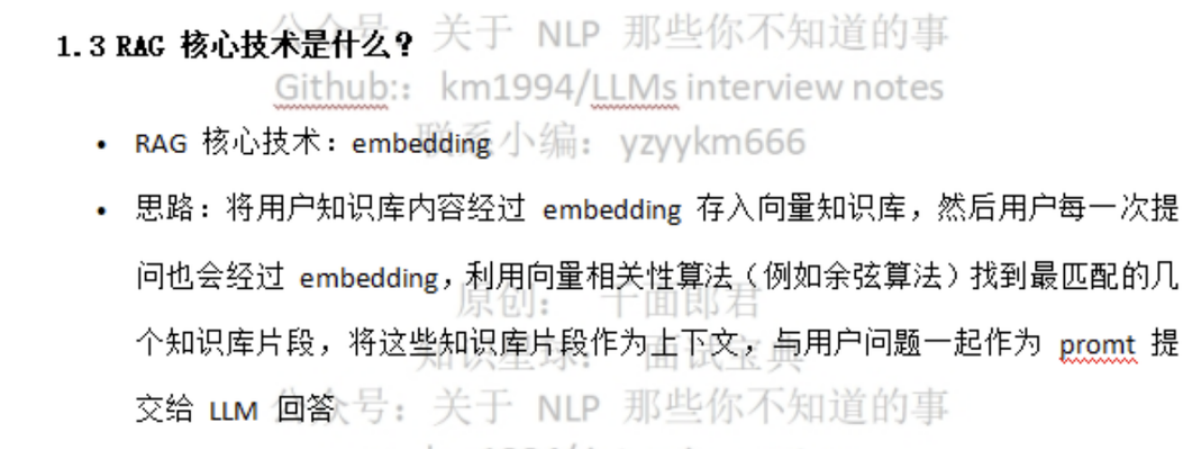
> “使用 RAG 可以显著减少生成模型的幻觉现象，因为回答会基于真实检索到的文档。同时，知识库和模型解耦，更新知识无需重新训练整个模型，提高效率与可维护性。”

RAG vs SFT 的优势

> “与 SFT 将知识固化在模型参数不同，RAG 将知识放在外部库中，模型只需保持检索与生成能力。这意味着新知识只需更新库，无需再训练大模型，并且能持续支持海量不断变化的知识。”

召回效果评估

> “召回效果主要用 Recall@k 和 MRR 衡量：Recall@k 看是否能在 top-k 中检到相关文档，MRR 则关注首个相关文档的排名。我们通常先优化召回器的高召回，再用重排器提升精度。”



高效微调 & PEFT

- LoRA（原理；参数量计算示例；微调哪些层；矩阵初始化；为什么不全初始化为 0）
- QLoRA、Adapter、Prompt Tuning、Prefix Cache（实现方式与优劣；Deepseek 如何开启 Prefix Cache；Prompt 系统化优化方法）

答案：

LoRA

> “LoRA 通过对大矩阵做低秩分解 W′=W+αBA，只微调 A
>
> A∈Rd×r 和 B B∈Rr×d，显存与存储成本显著降低。
> 例如 d=10 000,r=100时，仅需 2M 参数，相比原 100M 参数节省 50倍。通常插入注意力层的 Query/Value 投影，初始化时令 B=0，随机初始化 A 打破对称性，确保梯度有效更新。”

QLoRA

> “QLoRA 在 Int4 量化的基础模型上再加 LoRA，带来少许精度损失，但总体显存占用可下降 80–90%，并可通过调参基本恢复性能。”

Adapter

> “Adapter 在每层的 Feed-Forward 后插入小型瓶颈模块（Down→ReLU→Up），只训练这两层线性层。它支持多任务插拔、参数高效，但会带来少量推理开销，需要框架支持。”

Prompt Tuning & P-Tuning v2

> “Prompt Tuning 只更新一段连续的可训练 prompt 向量，参数量极少；P-Tuning v2 在此基础上加入小型 LSTM/MLP，对 prompt 做非线性映射，提升表达力，适合更复杂的任务。”

略Prefix Cache

> “Prefix Cache 在推理时缓存前向计算的 Key/Value，避免每步都重算注意力，能提升 20–40% 推理吞吐、降低峰值显存。以 Deepseek 为例，只需在 config 中设置

> `enable_prefix_cache=true`，框架自动管理缓存。”

略解码、缓存与部署

- KV Cache（为什么用；优化方法）
- 文生图/图生文、CLIP、BLIP（架构与解码流程；核心代码思路；多模态拼接 vs 异构图方式）
- 解码策略（贪心 vs Beam；top-k/top-p/温度等参数；如何确保 JSON 输出）
- LLM 部署与加速（vLLM 如何提速；常见部署问题；分布式架构经验；多机多卡训练）

模型架构演进

- 经典模型对比（BERT/XLNet/ALBERT/Ernie；LSTM vs GRU；MLP/CNN/RNN/Transformer 优缺点；为什么 BERT 是双向、GPT 不是）
- 多模态 & Mixture-of-Experts（Deepseek MOE 路由机制；GQA 原理；MQA/GQA/MHA 区别）

指标与训练流程

- 评估指标（Precision/Recall/F1/AUC；ROUGE 计算；SFT 与 RAG 的指标；有没有比简单指标更好的方法）
- 预训练 vs SFT（流程差异；限制上下文长度的因素；长上下文解决方案）

答案：

1 经典模型对比

模型	特点	优劣
BERT	双向 Transformer 编码器	表达能力强，适合分类/匹配等理解任务
GPT	单向解码器，仅看左侧上下文	擅长生成，适合续写、问答
XLNet	使用排列语言建模（permutation LM）	捕捉双向依赖，无预设遮蔽模式
ALBERT	BERT 的轻量版（参数共享 + 矩阵分解）	参数少，效果仍佳
Ernie（百度）	引入知识图谱与句法信息	增强语义建模能力

> **LSTM vs GRU：**
> GRU 更轻量，收敛快；LSTM 有更强记忆能力。多数实际工程中 GRU 更常用。

> **MLP / CNN / RNN / Transformer 对比简述：**

- **MLP**：无序输入，结构简单，不能建模序列关系；
- **CNN**：可提取局部模式，适用于文本分类等；
- **RNN**：按序处理，适合时间序列但难并行；
- **Transformer**：全局建模 + 并行计算，是当前主流架构。

> **为什么 BERT 是双向，GPT 不是?**

- BERT 使用 Masked LM，允许看到左右两侧；
- GPT 为自回归语言建模，必须只看左边，确保生成合理。

---

2 多模态与 Mixture-of-Experts（MoE）机制

> **DeepSeek MoE 路由机制：**

- 每层选择部分专家参与计算（如 top-2 gating），其余专家不激活；
- 降低计算量，提升模型容量（稀疏激活）；
- 可用于多任务或多模态融合。

> **GQA（Grouped Query Attention）原理：**

- 将多个 Query 共享 Key/Value 权重，减少冗余计算；
- 常见于 LLaMA、Qwen 等高效推理模型中。

> **分类任务：**

- **Precision**（精确率）：预测为正的有多少是真正；
- **Recall**（召回率）：所有真正中被找出的比例；
- **F1**：精确率与召回率的调和平均；

> **排序任务（如点击预测）：**

- **AUC**：二分类排序性能，越接近 1 越好。

> **略）文本生成任务：**

- **ROUGE-1/2/L**：n-gram、最长公共子序列重叠
- **BLEU**：多句翻译质量；
- **BERTScore/MoverScore**：基于语义的相似度评估（解决表面词差异问题）

> **RAG 特有指标：**

- **Recall@k、MRR**：检索质量
- **生成结果 F1/ROUGE**：最终输出的质量

Agents 与应用 ← 新增

- Agent 框架（ReAct、Toolformer、AutoGPT 等）
- Chain-of-Thought 与思考过程建模
- Agent 的决策与规划策略
- Agent 与外部工具／API 的集成方式
- 安全与对齐（如 LM-Guard）

答案：

Q1. 什么是大模型（LLMs）Agent?

✅ **回答模板：**
大模型 Agent 是一种将语言模型（LLM）作为核心推理单元，同时集成**外部工具**、**记忆机制**、**规划模块**等组件的系统，使其能够在复杂任务中完成“感知—思考—行动—反馈”闭环。
它区别于传统的“纯生成模型”，能够**调用工具**、**维护状态**、**分解任务**，解决多步骤复杂问题。

Q2. 大模型 Agent 有哪些组成部分?

✅ **回答模板：**
一个典型的 Agent 系统通常包括以下几个模块：

语言模型（LLM）：核心的生成与推理模块；

规划（Planning）：任务拆解与决策引导；

工具调用（Tool Use）：与外部环境或 API 交互；

记忆系统（Memory）：短期记忆与长期知识存储；

执行引擎（Executor）：调度组件，负责按步骤执行；

Q2.1 什么是规划（Planning）？

✅ **回答模板：**
规划模块的作用是将复杂任务**拆解为子目标和子任务**，并为每一步提供清晰的指令或工具调用顺序，常见形式有：

Chain of thought：模型被要求“think step by step”利用更多的时间进行计算，将艰难的任务分解成更小，更简单的步骤。

Tree of Thoughts：进一步扩展CoT。它首先将问题分解为多个思考步骤，并

在每个步骤中生成多个思考，从而创造一个树形结构。搜索过程可以是BFS(广度优先搜索）或DFS（深度优先搜

索），每个状态由分类器（通过一个prompt）或少数服从多数的投票原则来决定。

Graph of Thoughts：同时支持多链、树形以及任意图形结构的Prompt方案，支持各种基于图形的思考转换,如聚

合、回溯、循环等。将复杂问题建模为操作图（Graph of Operations，GoO),以LLM作为引擎自动执行，从而提供解决复杂问题的能力。

ReAct 式“思考—行动—观察”循环；

明确的 Planner–Executor 架构（如 BabyAGI）。

Q2.1.1 如何进行子目标拆解与任务分解？有哪些方法？

✅ **回答模板：**
任务分解常用的方法包括：

Zero-shot CoT：直接让 LLM 生成思考步骤；

Prompt 编写引导任务分解：如“你现在是一个专业的流程设计师”；

Tree of Thoughts / DFS Planning：模型在多个方案间搜索最优路径；

调用专门的 LLM Planner（如 GPT-4），从指令中提取明确子任务。

Q2.1.2 模型自我反省（Self-Reflection）是指什么？有哪些方法？

✅ **回答模板：**
模型自我反省指的是 LLM 在做出某一步判断或调用后，能“停下来”评估结果是否合理，可行方法包括：

让模型生成 `Critique:` 段，分析自身行为是否错误；

使用 ReAct-style Prompt 插入 `Thought:` → `Self-Evaluation:`；

借助另一个 LLM 或外部函数评估当前行动结果（如 Toolformer 结构）。

Q2.2 什么是记忆（Memory）？

✅ **回答模板：**
记忆模块用于支持 LLM 处理**上下文超出窗口长度**的问题，分为：

短期记忆：当前对话上下文，通常缓存在 prompt 里；

长期记忆：外部存储（如 FAISS、Chroma、Redis）保存历史知识或事件；

通过检索相关内容插入 prompt 实现 “Context Injection”。
这使得 Agent 能跨回合、跨任务调用历史知识或经验。

Q2.3 什么是工具调用（Tool Use）？

✅ **回答模板：**
工具调用指的是模型在推理过程中，借助外部 API、函数或插件来完成任务，如：

检索（RAG）；

计算器／Python 代码执行；

数据库查询、HTTP 请求；

系统函数（Shell、浏览器等）。

工具通常通过 `Function Calling`、`Tool Spec + JSON`、`ReAct` prompt 实现与模型的对话式集成。

Q3. LLM Agent 主要利用了大模型哪些能力？

✅ **回答模板：**
Agent 能力依赖于 LLM 的以下基础能力：

语言理解与生成能力：处理任务描述，生成多步推理内容；

工具调用理解能力：理解函数名、参数含义，构造有效调用；

任务拆解能力：根据描述分解步骤；

上下文理解与对话记忆：对历史信息的关联和复用。

这些能力使得 LLM 不只是生成器，更是智能“决策执行者”。

Q4. 能否结合代码讲解一下 LLM Agent 的实现思路？

✅ **回答模板：**
当然。以 LangChain 中的 ReAct Agent 为例：

python

Copy code

```
from langchain.agents import initialize_agent, load_tools
from langchain.chat_models import ChatOpenAI
llm = ChatOpenAI()
tools = load_tools(["serpapi", "llm-math"], llm=llm)
agent = initialize_agent(tools, llm, agent="react-description", verbose=True)
agent.run("Who is the president of the USA and what's their age plus 10?")
```

这个例子中，Agent 具备：

工具加载（搜索 + 数学）；

ReAct prompt 的思考／行动切换；

内部上下文维护；

自主调度多个子任务。

Q5. 如何给 LLM 注入领域知识？

✅ **回答模板：**
方法包括：

RAG（检索增强）：用向量数据库存放结构化文档，模型每次提问时查询相关资料；

Prompt Engineering：在 prompt 中硬编码知识或提示；

微调 / SFT：在领域专属语料上进行监督微调；

Tool Use + API：在需要专业知识时查询外部数据库或调用专属模型。

Q6. 常见的 Agent 框架或应用有哪些？

✅ **回答模板：**
当前主流 Agent 框架包括：

LangChain：模块丰富，生态活跃；

AutoGPT / BabyAGI：强调任务规划、自主执行；

ReAct Prompting：Google 提出，经典的语言–行动交替框架；

Transformers Agent（HuggingFace）：集成工具 + 模型 + UI；

OpenAI Function Calling：通过 function schema 实现工具调用。

典型应用：

多轮问答机器人；

文档问答系统；

自动表格分析、代码生成；

电商智能客服；

医疗／法律知识问答。

开源工具 & 基础技能

- Transformers 库结构与参数规模
- Optimizer（Adam/RMSProp/SGD 等区别）
- Linux & Python 基础（如 `head` 命令等）
- 其他（模型压缩；prompt 构造经验；SQL/C++ 基础；项目经验相关问答）

[语言模型应用]

```
|
├── NLP基础任务（分类 / 生成 / QA / NER / 翻译）
├── 增强型任务（RAG / Agent / Tool Use / Code）
├── 多模态任务（图文 / 音频 / 视频）
└── 行业垂直场景（教育 / 医疗 / 法律 / 金融 / 电商）
```

以下是针对高频NLP算法岗面试问题整理的模块化标准答题模板，适合用于快速准备各公司技术面，你先学习一下

一、大模型架构与原理

1. BERT 与 GPT 的区别

答题模板：

BERT 是一个基于 Transformer Encoder 的预训练模型，采用 Masked Language Model（MLM）与 Next Sentence Prediction（NSP）进行训练，能够利用双向上下文。GPT 是基于 Transformer Decoder 的自回归语言模型，仅利用左侧上下文，训练任务为语言建模（LM）。两者在结构、预训练目标及应用方向上有所不同。

2. Decoder-only 架构与 Mask 实现

答题模板：

Decoder-only 结构（如 GPT）中的 Attention Mask 使用下三角矩阵或 causal mask，实现只能看前面 token 的效果，防止信息泄漏。

3. LLaMA / LLaMA2 的结构特点与优化

答题模板：

LLaMA 使用 Pre-LN Transformer 架构，使用 RMSNorm 替代 LayerNorm 来减少归一化计算开销，采用 SwiGLU 激活函数提升非线性表达能力，引入 Rotary PE（RoPE）作为相对位置编码方法，且去除了 Dropout 层以优化推理稳定性与效率。LLaMA2 在此基础上增加了 GQA（Grouped Query Attention）以减少多头计算的冗余，引入 FlashAttention 提升注意力计算的显存效率，并通过更多长序列训练来增强对长文本的建模能力，整体提升了训练速度和推理吞吐。

4. 前置 LayerNorm / RMSNorm 的好处

答题模板：

前置 LayerNorm 有助于稳定梯度，提升训练收敛速度；RMSNorm 仅归一化均方，计算开销更小，实测性能更优。

5. Rotary Position Embedding（RoPE）优势

答题模板：

RoPE 能将位置编码与 Attention 权重自然结合，保留 token 之间相对位置信息，且不需要额外参数，支持长度外推。

6. BERT 与 LLaMA 的位置编码对比

答题模板：

BERT 使用的是可学习的绝对位置编码（position embedding），模型训练后固定；LLaMA 使用的是旋转位置编码 RoPE，更适合处理长文本，并具有相对位置信息优势。

二、训练与微调方法

1. LoRA / QLoRA 原理与区别

答题模板：

LoRA 在不更新全模型参数的前提下，对权重增量进行低秩矩阵分解（A @ B），仅训练这两个小矩阵。QLoRA 结合量化与 LoRA，将基础模型量化为 Int4，在量化模型上训练 LoRA 层，显著减少显存开销。

2. Prompt Tuning / P-Tuning v2 / Adapter

答题模板：

Prompt Tuning 通过添加可训练的 prompt 向量进行微调；P-Tuning v2 引入 LSTM + MLP 提升表达力；Adapter 在原始 Transformer 中插入小模块，仅训练插入模块。

3. RLHF 三阶段流程（SFT → RM → PPO）

答题模板：

- 第一阶段：SFT，使用人类指令数据进行有监督微调；
- 第二阶段：训练 Reward Model，用于对输出质量打分；
- 第三阶段：使用 PPO 优化语言模型以最大化 RM 打分，完成人类偏好对齐。

4. DPO 与 PPO 区别

答题模板：

DPO（Direct Preference Optimization）是近年提出的一种更简单高效的大模型对齐方法。

DPO 不用奖励模型，

直接比对人类偏好，

算 loss 更高效，

不走 RL，易实现。

KV Cache（Key-Value Cache） 是在大语言模型生成文本时，为了避免重复计算已生成内容的注意力表示而缓存下来的 Key 和 Value 张量。它大幅提升推理速度、降低计算冗余，是加速长文本生成的核心技术之一。

略5. SFT 的数据集规模与配比

答题模板：

SFT 通常包含千亿 token 级别的数据；配比上需均衡 general instruction 与 domain instruction 的比例（如 7:3 或 8:2），防止过拟合特定领域。

6. PEFT 方法概览

答题模板：

PEFT（参数高效微调）包括 LoRA、Adapter、Prefix Tuning、Prompt Tuning、BitFit 等方法，目标是在冻结主干参数的同时仅微调小规模参数模块，实现资源友好型微调。

三、分布式训练与部署

1. ZeRO 优化器 / DeepSpeed

答题模板：

ZeRO 将模型参数、梯度、优化器状态切分存储在多个 GPU 上，分为 Stage 1/2/3，显著降低显存占用；DeepSpeed 是高性能训练框架，支持 ZeRO、Offload、混合精度训练等。

2. 三类并行方法对比

答题模板：

- 数据并行：样本维度切分，参数全量共享；
- 张量并行：模型层内参数切分，跨卡通信；
- 流水线并行：模型按层切分，按顺序串行处理。

3. 显存估算与优化技巧

答题模板：

训练显存估算 ≈ 参数 + 优化器状态（×2\~3）+ 激活值缓存；推理显存估算 ≈ 参数 + KV cache；优化手段包括量化、混合精度、LoRA、KV 分页、activation checkpointing。

4. Transformer 推理加速技术（vLLM 等）

答题模板：

vLLM 通过 PagedAttention 管理 KV cache，实现 KV 动态分页复用；结合 FlashAttention、动态 batch 合并，显著提升推理吞吐与显存效率。

四、NLP 任务细节与模型原理

1. Attention 与 QKV、softmax

答题模板：

Self-Attention 使用 Q/K/V 三个矩阵计算注意力分布，softmax 后做加权求和；除以　稳定梯度；多头机制可并行获取不同子空间特征。

2. 激活函数：ReLU 与 SiLU

答题模板：

ReLU 是分段线性函数，计算高效但存在神经元死亡问题；SiLU（Swish）为 $x \cdot \text{sigmoid}(x)$，在训练深层模型时表现更优，LLaMA 系列常采用其变体 SwiGLU。

3. 模型部署性能瓶颈与解决方案

答题模板：

常见瓶颈包括显存占用高、推理速度慢、batch 不可变等。解决方法包括：KV cache 动态分页（如 vLLM）、模型量化（GPTQ）、tensorRT 加速、请求打包与 Prefill KV 重用。

4. 常见 LLM 对比

答题模板：

主流开源 LLM 包括 LLaMA、Baichuan、ChatGLM、Qwen、Deepseek；它们在位置编码、激活函数、归一化方式、指令集构建等方面有所差异，例如 Deepseek 增强了代码能力，采用多语言预训练。

5. DeepSeek 的创新点

答题模板：

DeepSeek 使用中英双语预训练，并强调代码能力，在 tokenizer、训练策略与指令微调集上做了优化。其 DeepSeek-Coder 模型在代码任务上表现优异。

五、评估指标与优化目标

1. Precision / Recall / F1 / AUC

答题模板：

- Precision：预测为正中真阳比例；
- Recall：所有正样本中预测正确的比例；
- F1 是二者调和平均，权衡精确与召回；
- AUC 衡量排序能力，表示正例排在负例前的概率。

2. KL 散度与交叉熵

答题模板：

交叉熵是分类问题常用损失，本质是固定标签分布的 KL 散度；二者联系紧密，交叉熵更常用于训练目标，KL 用于分布偏移评估。

则交叉熵定义为：

$$\mathrm{H}(P, Q) = -\sum_{x} P(x) \log Q(x)$$

> 直观理解：交叉熵衡量 **用 Q 分布去表达 P 分布时需要付出的代价（信息量）**。

---

2. KL 散度（Kullback-Leibler Divergence）

$$\mathrm{KL}(P \| Q) = \sum_{x} P(x) \log \frac{P(x)}{Q(x)} = \mathrm{H}(P, Q) - \mathrm{H}(P)$$

> 它衡量的是从 Q 分布"偏离" P 分布的程度，是一个**非对称距离**，永远非负。
