

第一章 语言模型

一、统计方法的语言模型

1. n-grams 语言模型

包含 N 个元素的语言符号 $W_{1:N} = \{W_1, W_2, W_3, \dots, W_N\}$
(文本) (词)

n-gram: 长度为 n 的词序列

文本 $W_{1:N}$ 出现的概率为

$$P_{n\text{-grams}}(W_{1:N}) = \prod_{i=n}^N \frac{C(W_{i-n+1:i})}{C(W_{i-n+1:i-1})}$$

其中 $C(W_{i-n+1:i-1})$ 为 $\{W_{i-n+1}, \dots, W_{i-1}\}$ 在语料库中出现的次数

n 为变量: 当 $n=1$ 时为 unigram, 不考虑上下文关系, 分子 $C(W_i)$, 分母 $C(W_{i:i-1}) = C(\text{total})$ 为语料库为包含词的总数;

当 $n=2$ 时为 bigram, 仅考虑前 1 词;

当 $n=3$ 时为 trigram, 仅考虑前 2 词;

.....

$$\begin{aligned} \text{如: } P_{\text{bigrams}}(\text{长颈鹿}, \text{脖子}, \text{长}) &= \frac{C(\text{长颈鹿}, \text{脖子})}{C(\text{长颈鹿})} \cdot \frac{C(\text{脖子}, \text{长})}{C(\text{脖子})} \\ &= \frac{2}{5} \cdot \frac{2}{6} = \frac{2}{15} \end{aligned}$$

语料库
脖子长是长颈鹿最醒目的特征之一。
脖子长使得长颈鹿看起来非常优雅, 并为其在获取食物上带来便利。
有了长脖子的加持, 长颈鹿可以观察到动物园里那些隐蔽的角落里发生的事情。
长颈鹿脖子和人类脖子一样, 只有七节颈椎, 也容易患颈椎病。
如同长颈鹿脖子由短变长的进化历程一样, 语言模型也在不断进化。

该句未出现在 corpus 中, 但 $P = \frac{2}{15}$, 证明了 n-grams 对未知文本的泛化能力

当 n 太大, 泛化能力减弱, 可能出现大量零概率 $\rightarrow$ 拉普拉斯平滑改善 $\left(\frac{\sum I(z=j)}{m} \Rightarrow \frac{\sum I(z=j)+1}{m+z \text{ 的可能值数量}} \right)$;

当 n 太小, 特征捕获不足

2. 基于RNN的语言模型



$$\begin{aligned} \text{FNN: } h &= g(W_I x_t) & \text{RNN: } h_t &= g(W_I x_t + \underline{W_H h_{t-1}}) \\ O_t &= f(W_O h_t) & O_t &= f(W_O h_t) \quad \text{引入历史信息} \end{aligned}$$

RNN训练涉及大量矩阵联乘, 易梯度衰减/爆炸



GRU/LSTM引入门控结构

对词序列 $\{w_1, w_2, \dots, w_N\}$, RNN语言模型通过 w_i 和 h_{i-1}

$$P(w_{i+1} | w_{1:i}) = P(w_{i+1} | w_i, h_{i-1})$$

来预测 w_{i+1} 。因此, 句子的出现概率为

$$P(w_{1:N}) = \prod_{i=1}^{N-1} P(w_{i+1} | w_i, h_{i-1})$$

在训练时, 词典 D 为 $\{\hat{w}_1, \hat{w}_2, \dots, \hat{w}_{|D|}\}$, 模型输出

$$O_i = \{O_i[\hat{w}_d]\}_{d=1}^{|D|}$$

其中 $O_i[\hat{w}_d]$ 代表词典中 $\hat{w}_d$ 出现概率。则训练时,

$$P(w_{i+1} | w_i, h_{i-1}) = O_i[w_{i+1}]$$

输入第 S 个样本的第 i 个词时, 模型输出 $O_{i,S}$

$$L = \frac{1}{N \cdot |S|} \sum_{s=1}^{|S|} \sum_{i=1}^N -\log O_{i,s}[w_{i+1}]$$

文本生成任务, 采用自回归 $\Rightarrow$ $\left\langle \begin{array}{l} \text{错误级联放大} \\ \text{串行计算效率低} \end{array} \right.$



Teacher Forcing

Teacher Forcing 训练时, 每次都将输出与 GT 拼接, 而非自回归训练,



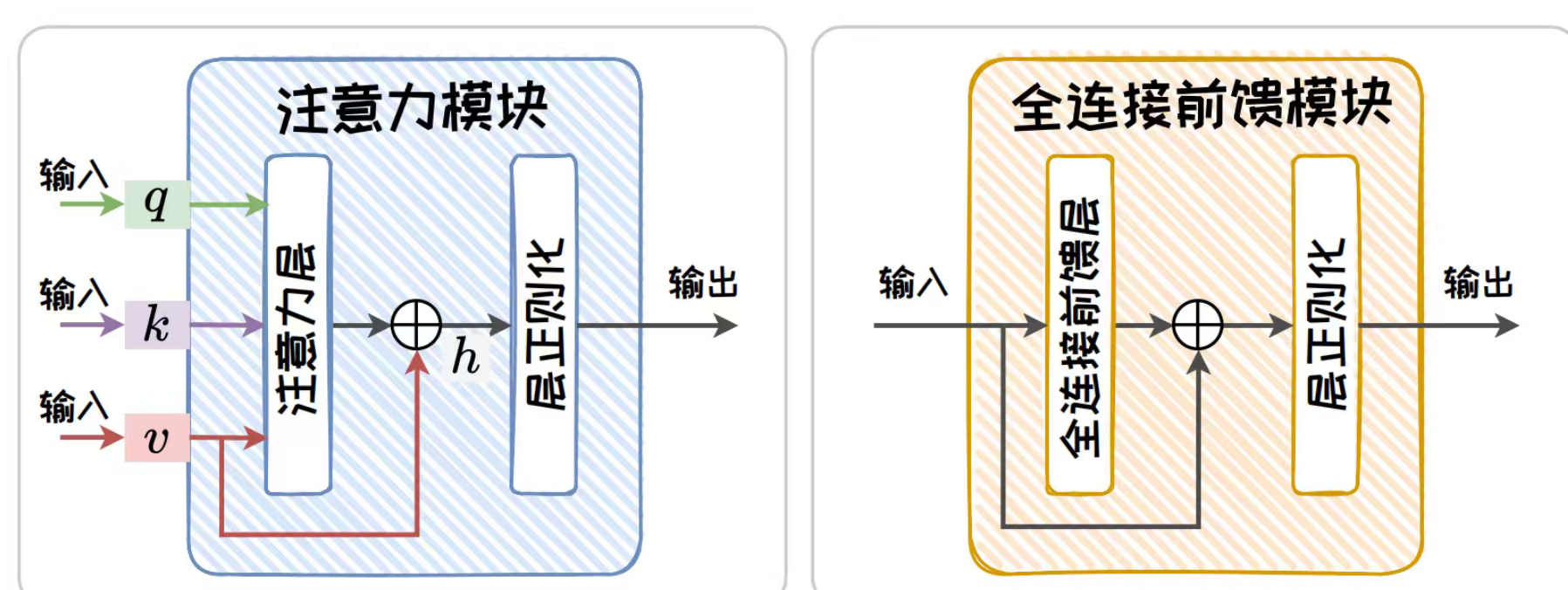
导致曝光偏差: 训练非自回归, 推理自回归, 存在差异



Scheduled Sampling: 训练中逐渐用部分模型生成词代替 GT

3. 基于 Transformer 的语言模型

Transformer $\left\{ \begin{array}{l} \text{Fully-connected Feedforward} \\ \text{Attention} \left\{ \begin{array}{l} \text{Self-Attention} \\ \text{Residual} \\ \text{Layer Norm} \end{array} \right. \end{array} \right.$



自注意力

将输入 $\{x_1, x_2, \dots, x_t\}$ 编码为 $\{(q_1, k_1, v_1), \dots, (q_t, k_t, v_t)\}$
query, key 计算自注意力权重 α , value 对输入编码

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

$$Q = X \cdot W^Q, K = X \cdot W^K, V = X \cdot W^V$$

其中 d_k 代表特征的维度

层正则化

输入向量为 $V = \{v_i\}_{i=1}^n$, 则

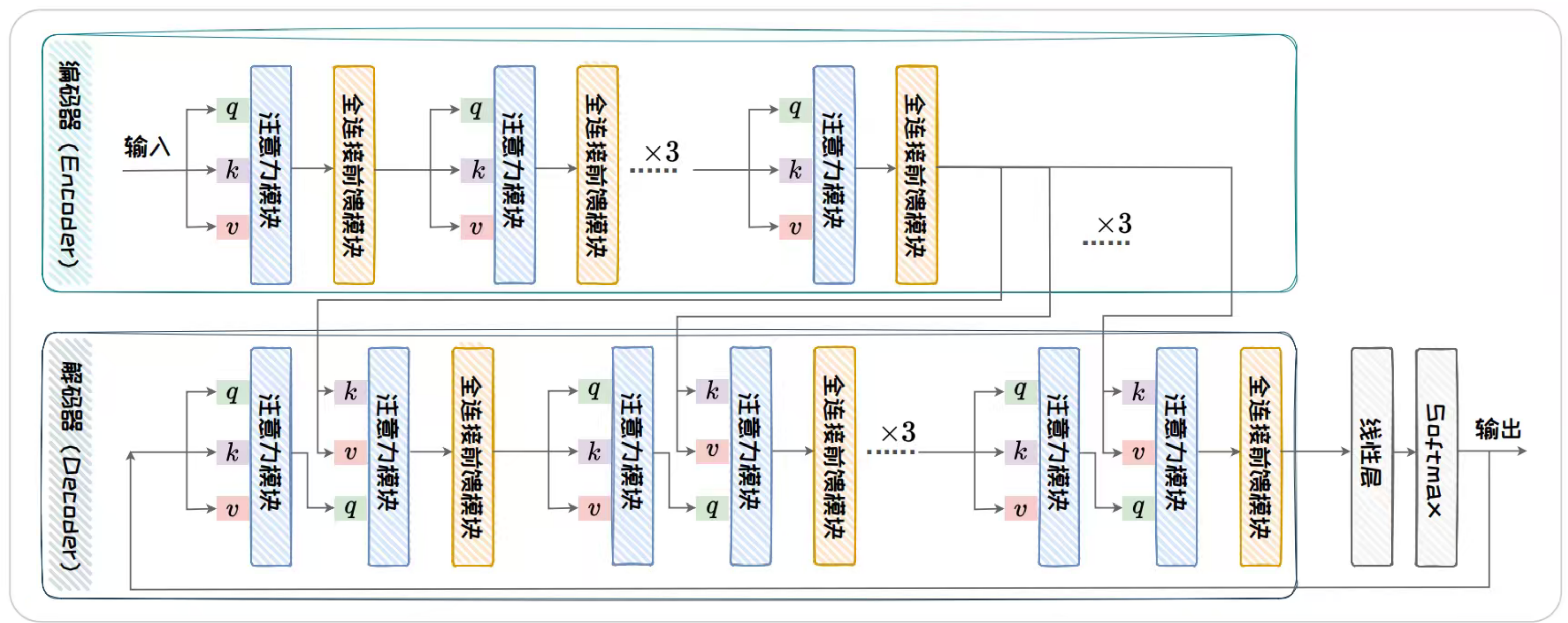
$$\text{LN}(v_i) = \frac{\alpha(v_i - \mu)}{\sigma} + \beta$$

其中 α, β 为可学习参数, μ, σ 为隐藏状态的均值和方差

Post-LN: (在残差后 LN) 梯度消失

Pre-LN: 表征坍塌

Transformer 结构



Encoder 部分由 6 个级联的 Self-Attention + FFN

Decoder 部分由 6 个级联的 Self-Attention $\times 2$ + FFN

Decoder 的 K、V 是 Encoder 的输出

训练时, 可 Encoder-Only 掩词补全 (如 BERT)

可 Encoder + Decoder 截断补全、顺序恢复 (T5)

可 Decoder-Only 预测下一词 (如 GPT-3)

4. 采样方法

概率最大化方法

词典 D , 输入文本 $\{w_1, w_2, \dots, w_N\}$, 第 i 轮输出 $O_i = \{O_i[w_d]\}_{d \in D}$,
 M 轮后生成文本 $\{w_{N+1}, w_{N+2}, \dots, w_{N+M}\}$

需最大化 $P(w_{N+1:N+M}) = \prod_{i=N}^{N+M-1} O_i[w_{i+1}] \Rightarrow \text{NP-Hard}$

1) 贪心搜索

每轮均选概率最大的词 $w_{i+1} = \arg\max_{x \in D} O_i[x]$

容易陷入局部最优

2) 波束搜索

每轮保留 b 个可能性最高的词 $B_i = \{w_{i+1}^1, w_{i+1}^2, \dots, w_{i+1}^b\}$

最后遍历 $\{B_i\}_{i=N}^M$ 中所有组合, 使概率最大

缺乏多样性

随机采样方法

1) Top-K 采样

每轮选前 K 个概率最高的词, 再对概率 softmax 归一化,

得到分布函数

$$p(w_{i+1}^1, \dots, w_{i+1}^K) = \left\{ \frac{\exp(o_i[w_{i+1}^1])}{\sum_{j=1}^K \exp(o_i[w_{i+1}^j])}, \dots, \frac{\exp(o_i[w_{i+1}^K])}{\sum_{j=1}^K \exp(o_i[w_{i+1}^j])} \right\}$$

从分布中采样出本轮的预测结果

$$w_{i+1} \sim p(w_{i+1}^1, w_{i+1}^2, \dots, w_{i+1}^K)$$

候选集为K导致词分布方差较大时,可能选到不合常理的词,
词分布方差较小时,丰富程度低,生成结果不新颖

2) Top-P 采样 (Nucleus 采样)

设定阈值 p 选择候选集 S_p , $\sum_{w \in S_p} O_i[w] \geq p$

3) Temperature 机制

temperature 参数在 softmax 之前缩放概率的尺度

引入 Temperature 后, Top-K 采样的候选集的分布如下所示:

$$p(w_{i+1}^1, \dots, w_{i+1}^K) = \left\{ \frac{\exp(\frac{o_i[w_{i+1}^1]}{T})}{\sum_{j=1}^K \exp(\frac{o_i[w_{i+1}^j]}{T})}, \dots, \frac{\exp(\frac{o_i[w_{i+1}^K]}{T})}{\sum_{j=1}^K \exp(\frac{o_i[w_{i+1}^j]}{T})} \right\}.$$

引入 Temperature 后, Top-P 采样的候选集的分布如下所示:

$$p(w_{i+1}^1, \dots, w_{i+1}^{|S_p|}) = \left\{ \frac{\exp(\frac{o_i[w_{i+1}^1]}{T})}{\sum_{j=1}^{|S_p|} \exp(\frac{o_i[w_{i+1}^j]}{T})}, \dots, \frac{\exp(\frac{o_i[w_{i+1}^{|S_p|}]}{T})}{\sum_{j=1}^{|S_p|} \exp(\frac{o_i[w_{i+1}^j]}{T})} \right\}.$$

$T > 1$ 时, 概率差异变小, 分布平坦, 随机性高

$0 < T < 1$ 时, 概率差异变大, 更易选到高概率词, 随机性小

5. 语言模型测评

内在测评

不依赖具体任务, 常为困惑度 Perplexity

测试文本为 $S_{test} = w_{1:N}$

$$PPL(S_{test}) = P(w_{1:N})^{-\frac{1}{N}} = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i | w_{<i})}}$$

生成文本的概率越大, 模型越肯定, PPL 越小

$$\text{可改写为 } PPL(S_{test}) = \exp\left(-\frac{1}{N} \sum_{i=1}^N \log P(w_i | w_{<i})\right)$$

模型生成词的分布与真实分布的交叉熵

即 $-\frac{1}{N} \sum_{i=1}^N \sum_{d=1}^{|D|} I(\hat{w}_d = w_i) \log O_{d1}[w_i]$

由于 $P(w_i | w_{<i}) \leq 1$, 因此交叉熵是生成词分布的信息熵的上界

外在测评

1) 基于统计指标

BLEU: Bilingual Evaluation Understudy

用于机器翻译(MT), 计算多次 n -grams 精度的几何平均

生成的翻译文本 $S_{gen} = \{S_{gen}^i\}_{i=1}^{|S_{gen}|}$, 参考翻译 $S_{ref} = \{S_{ref}^i\}_{i=1}^{|S_{ref}|}$

原始 n -gram 精度

$$Pr(g_n) = \frac{\sum_{i=1}^{|S_{gen}|} \sum_{g_n \in S_{gen}^i} Count_{match}(g_n, S_{ref}^i)}{\sum_{i=1}^{|S_{gen}|} \sum_{g_n \in S_{gen}^i} Count(g_n)}$$

如 large language models, 生成的是 big language models

当 $n=1$ 时, $Pr(g_1) = \frac{2}{3}$ (只有 large - big 不匹配)

当 $n=2$ 时, $Pr(g_2) = \frac{1}{2}$ (big language - large language)

BLEU 为 N 个 n -gram ($n=1 \sim N$) 精度几何平均

$$BLEU = \sqrt[N]{\prod_{n=1}^N Pr(g_n)} = \exp\left(\sum_{n=1}^N \log Pr(g_n)\right)$$

ROUGE: Recall-Oriented Understudy for Gisting Evaluation

用于摘要生成

ROUGE-N 是基于 n -gram 的召回指标

$$ROUGE-N = \frac{\sum_{s \in S_{ref}} \sum_{g_n \in s} Count_{match}(g_n, S_{gen})}{\sum_{s \in S_{ref}} \sum_{g_n \in s} Count(g_n)}$$

ROUGE-L 是基于最长公共子序列(LCS)的召回指标

$$ROUGE-L = \frac{(1 + \beta^2) R_r^g R_g^r}{R_r^g + \beta^2 R_g^r}$$

其中 $R_r^g = \frac{LCS(s_{ref}, s_{gen})}{|s_{ref}|}$, $R_g^r = \frac{LCS(s_{ref}, s_{gen})}{|s_{gen}|}$, $\beta = \frac{R_g^r}{R_r^g}$, $LCS(s_{ref}, s_{gen})$ 是生成/参考摘要间的最长公共子序列

缺点: 无法适应任务中表达的多样性、与人类相差甚远

2) 基于语言模型

BERTScore 是基于上下文词嵌入的, 用BERT得到每个词的上
下文嵌入向量

$$V_g^i = \text{BERT}(w_g^i | S_{gen}), \quad V_r^i = \text{BERT}(w_r^i | S_{ref}),$$

其中生成文本 $S_{gen} = \{w_g^i\}_{i=1}^{|S_{gen}|}$, 参考文本 $S_{ref} = \{w_r^i\}_{i=1}^{|S_{ref}|}$

BERTScore 从 Precision, Recall, F1 Score 三方面测评

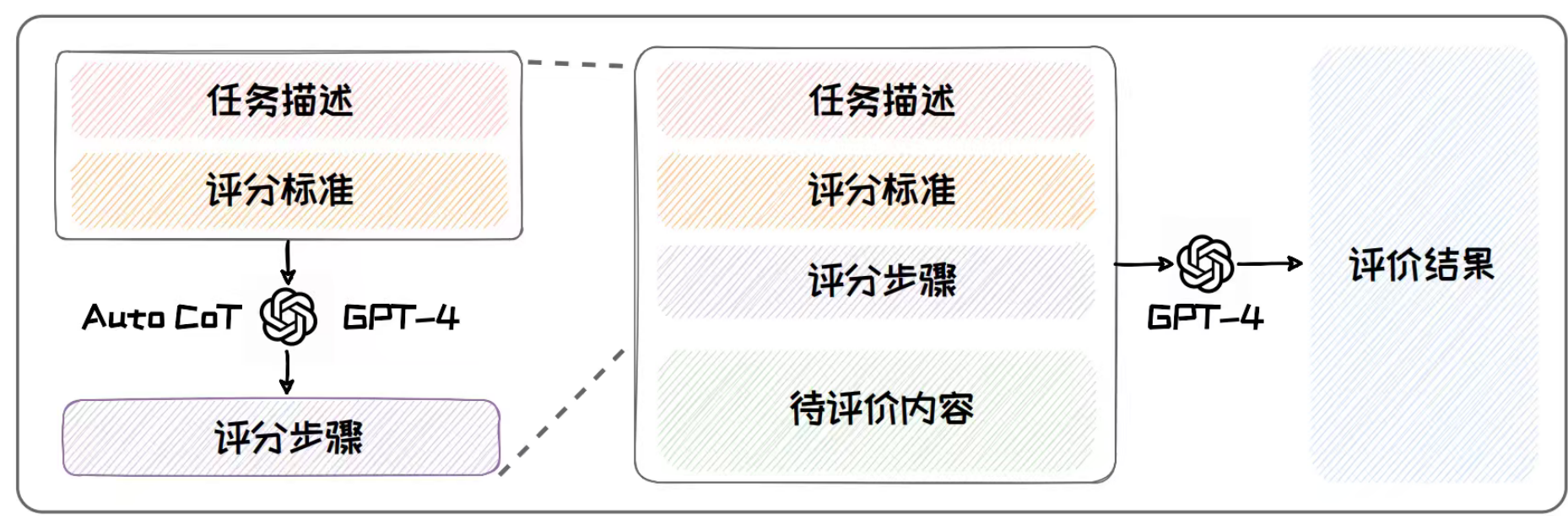
$$P_{BERT} = \frac{1}{|v_{gen}|} \sum_{i=1}^{|v_{gen}|} \max_{v_r \in v_{ref}} v_g^{i \top} v_r^i,$$

$$R_{BERT} = \frac{1}{|v_{ref}|} \sum_{i=1}^{|v_{ref}|} \max_{v_g \in v_{gen}} v_r^{i \top} v_g^i,$$

$$F_{BERT} = \frac{2P_{BERT} \cdot R_{BERT}}{P_{BERT} + R_{BERT}}.$$

G-EVAL

无需人类标注, 通过 Prompt Engineering 引导 GPT-4 输出评分



InstructScore

给出了数值得分以及对得分的解释

第二章 大语言模型架构

1. Scaling Law

① Kaplan - McCandlish 扩展法则

$$L(D) = \left(\frac{D}{D_c}\right)^{\alpha_D}, \alpha_D \sim -0.095, D_c \sim 5.4 \times 10^{13},$$

$$L(N) = \left(\frac{N}{N_c}\right)^{\alpha_N}, \alpha_N \sim -0.076, N_c \sim 8.8 \times 10^{13}.$$

数据规模 D , 模型规模 N , $L()$ 为交叉熵损失

此外, 为达到最优性能, 总计算量 $C = 6ND$

$N \propto C^{0.73}$, $D \propto C^{0.27}$ 模型规模上投入应略高于数据规模上

② Chinchilla 扩展法则

$$L(N, D) = E + \frac{A}{N^\alpha} + \frac{B}{D^\beta},$$

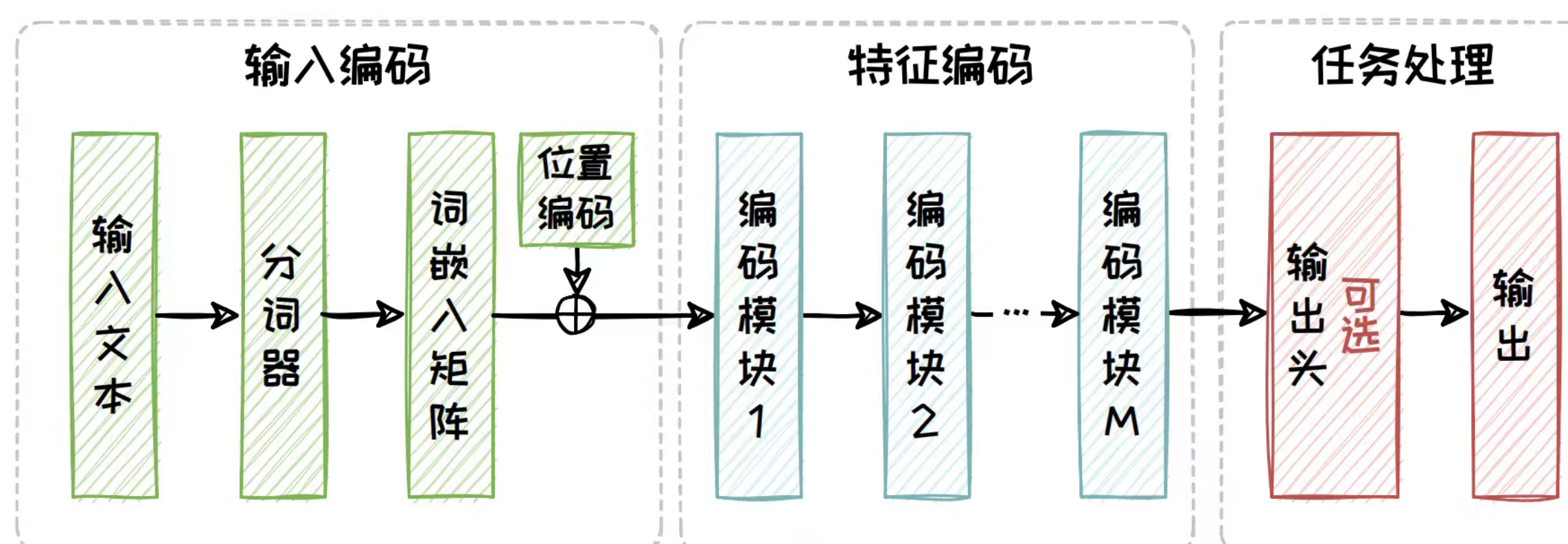
$$E = 1.69, A = 406.4, B = 410.7, \alpha = 0.34, \beta = 0.28.$$

计算预算分配 $N \propto C^{0.46}$, $D \propto C^{0.54}$

数据量和模型规模几乎同等重要

2. 主流模型架构

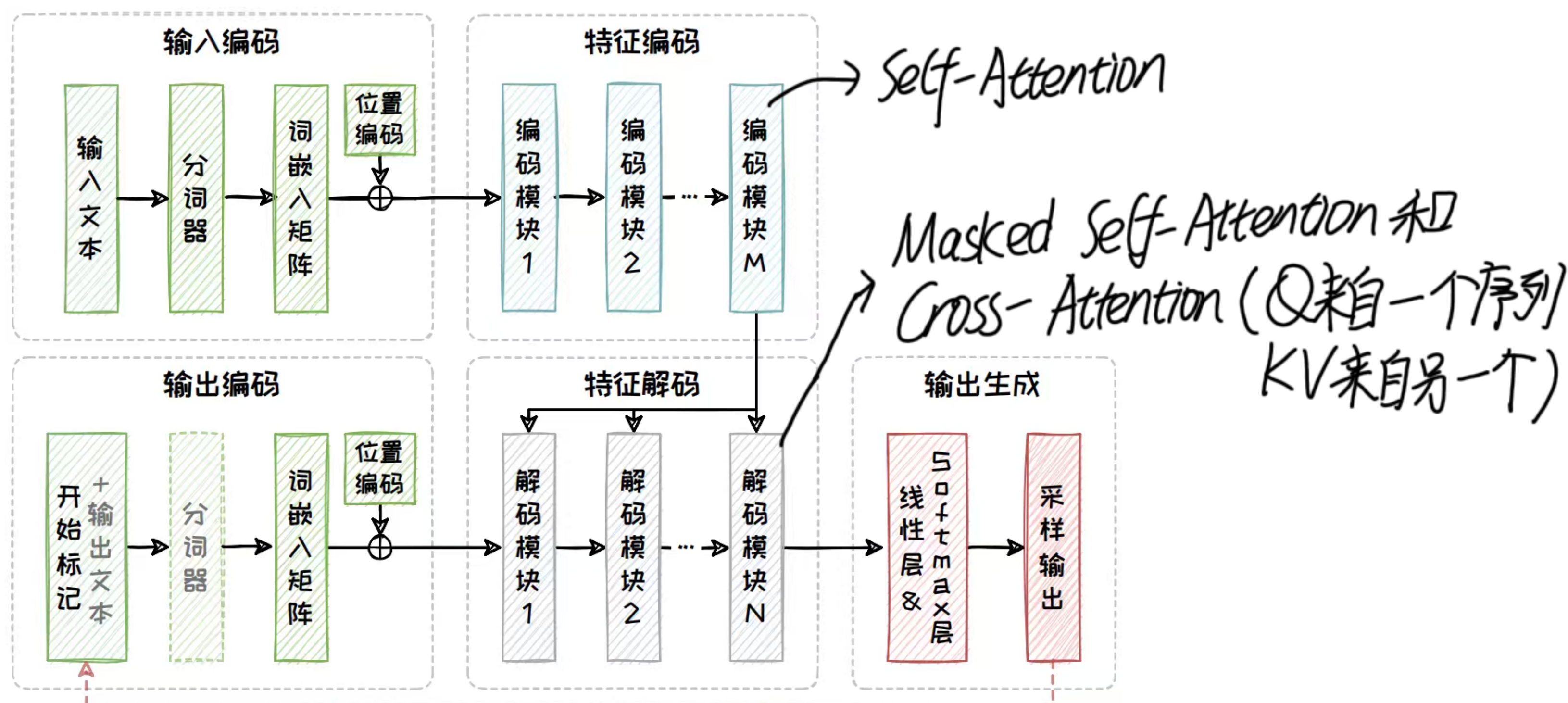
Encoder-Only



如 BERT

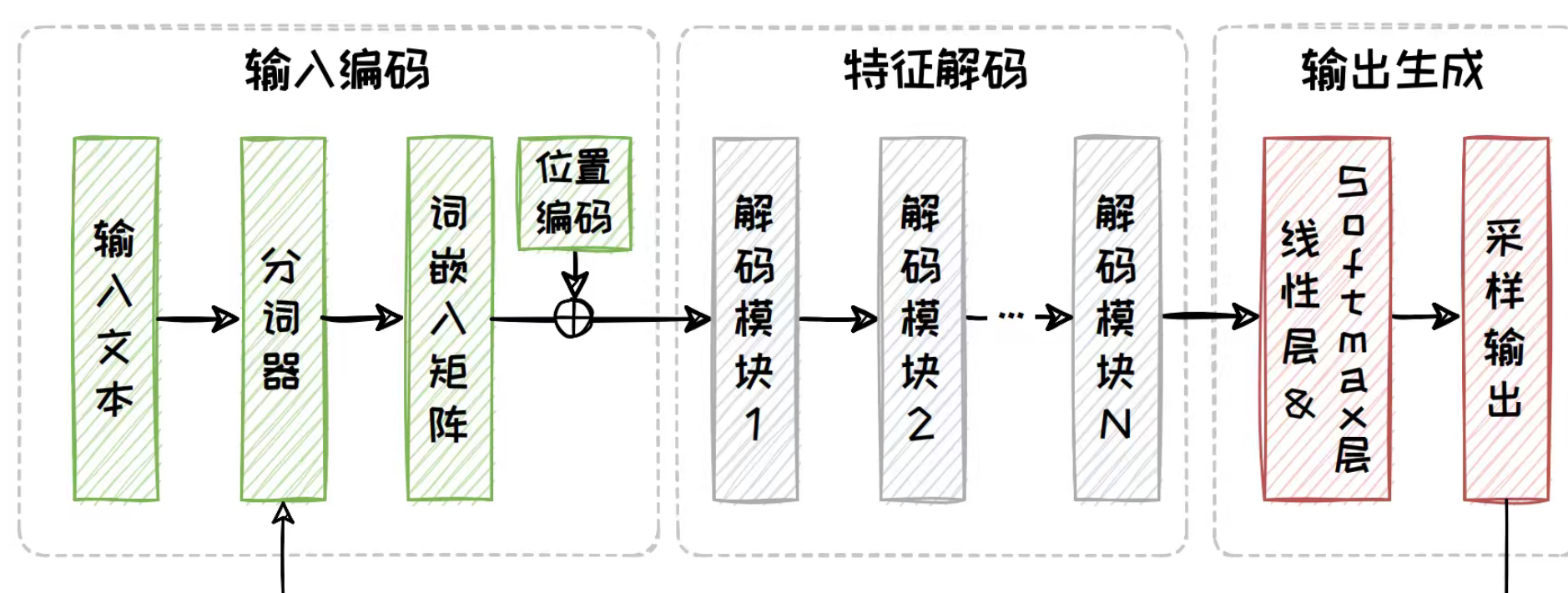
文本生成任务上, 每次都重新计算输入序列的 embedding, 生成的文本缺乏连贯性

Encoder-Decoder



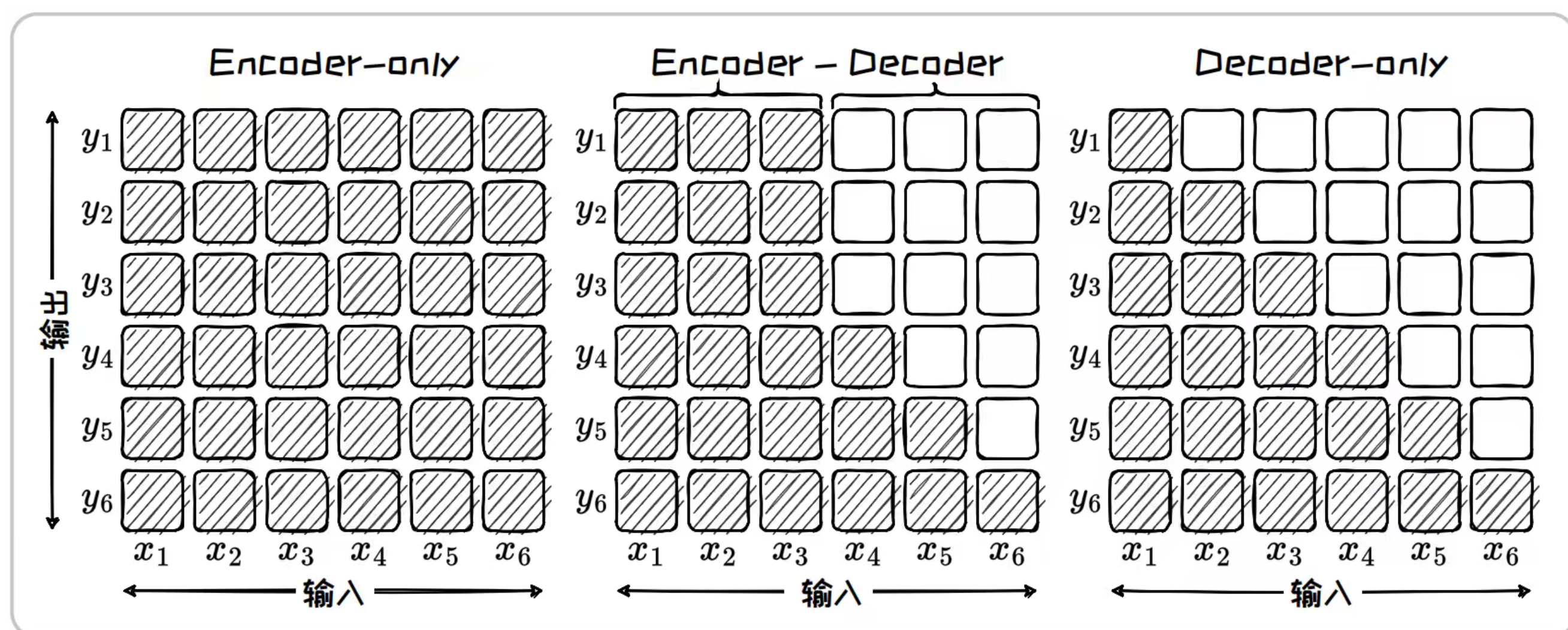
生成时,只能一步一步串行输出

Decoder-Only



如 GPT-1, GPT-2, GPT-3

注意力矩阵



Encoder-Only 来自注意力模块, 每个 token 都依赖于所有其他 token, 即双向注意力机制 $\Rightarrow$ 捕捉语义和依赖关系, 适合自然语言理解 (NLU), 如情感分析、文本分类

Encoder-Decoder 结合了编码器自注意力、解码器 Masked 自注意力和 Cross 注意力 $\Rightarrow$ 完全注意力

$\Rightarrow$ 适合条件生成任务, 如机器翻译、文本摘要、问答系统
但模型规模、计算量大

下三角注意力: 在当前 token 时, 只关注之前生成的 token

Decoder-Only 来自 Masked 自注意力, 是单向注意力机制, 掩码使其只能访问先前的 token (训练时) $\Rightarrow$ 适合无条件生成

3. Encoder-Only

① BERT

Bidirectional Encoder Representations from Transformers

分为 BERT-base 包含 12 个 encoder, 隐藏层 768 维, 注意力头数 12

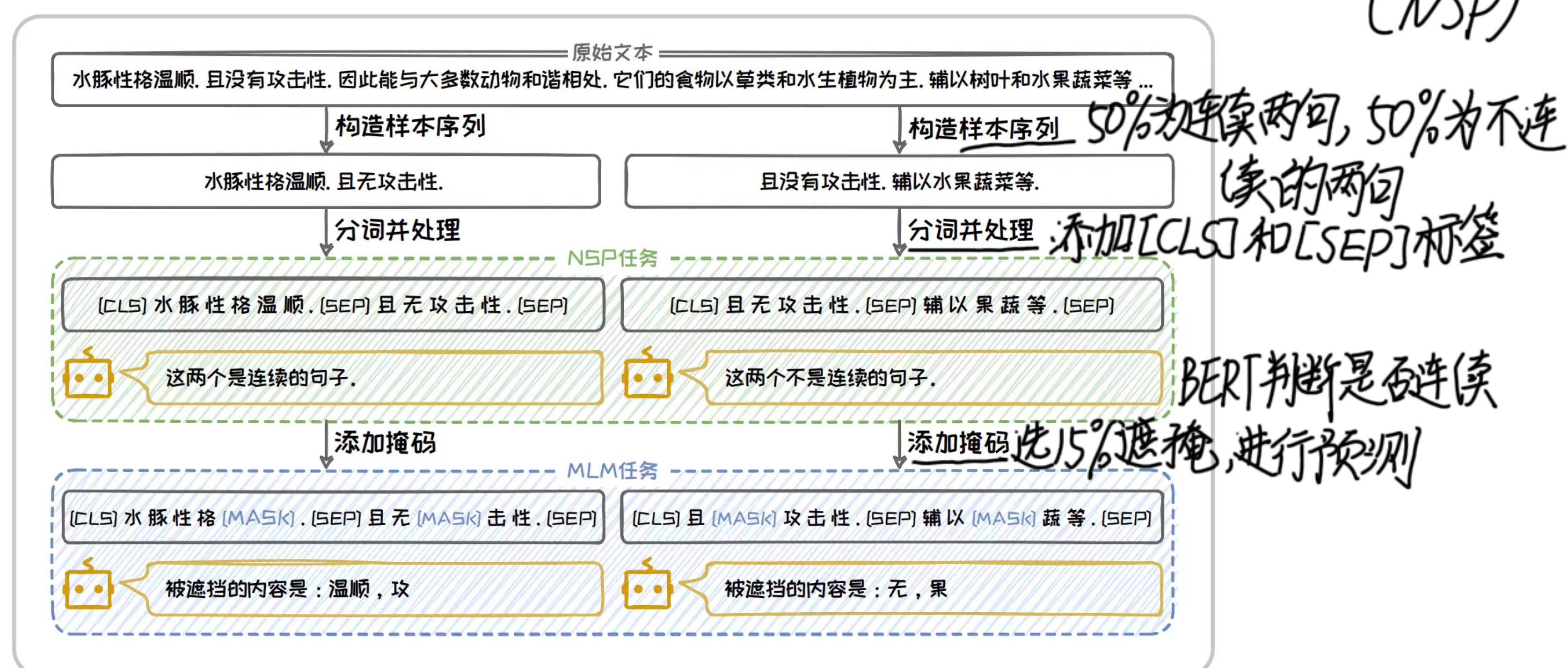
BERT-large 24

1024

16

数据集为小说 BookCorpus 和 Wikitext, 共 33 亿 token

任务为 Masked Language Model (MLM) 和 Next Sentence Prediction (NSP)



下游任务:

[CLS] 标签: 汇总整个序列信息, 是固定长度的向量, 用于分类和汇总

1) 文本分类中, 将 [CLS] 传向全连接层

2) 问答系统中, 输入 "[CLS] 问题 [SEP] 文本 [SEP]", 将输出的 [CLS] 给 2 个全连接层, 分别输出答案在文本中的起、止位置

3) 语义相似度中, 输入 "[CLS] 文本 - [SEP] 文本 = [SEP]", 将输出的 [CLS] 给全连接层

② RoBERTa

Robustly Optimized BERT Pretraining Approach

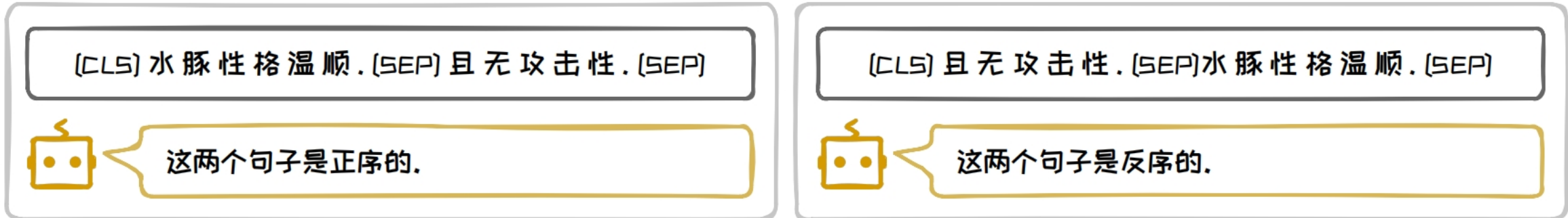
新增新闻 CC-News、网页 OpenWebText、故事 Stories

预训练任务只有动态掩码语言建模, 将数据复制 10 个副本, 给予不同的掩码

③ ALBERT
A Lite BERT

BERT词表V约为30000,隐藏层维度H=768,故Embedding层参数为 $30000 \times 768 \Rightarrow$ ALBERT先下投影至E=128,再上投影回H,故参数为 $V \times E + E \times H = 30000 \times 128 + 128 \times 768$

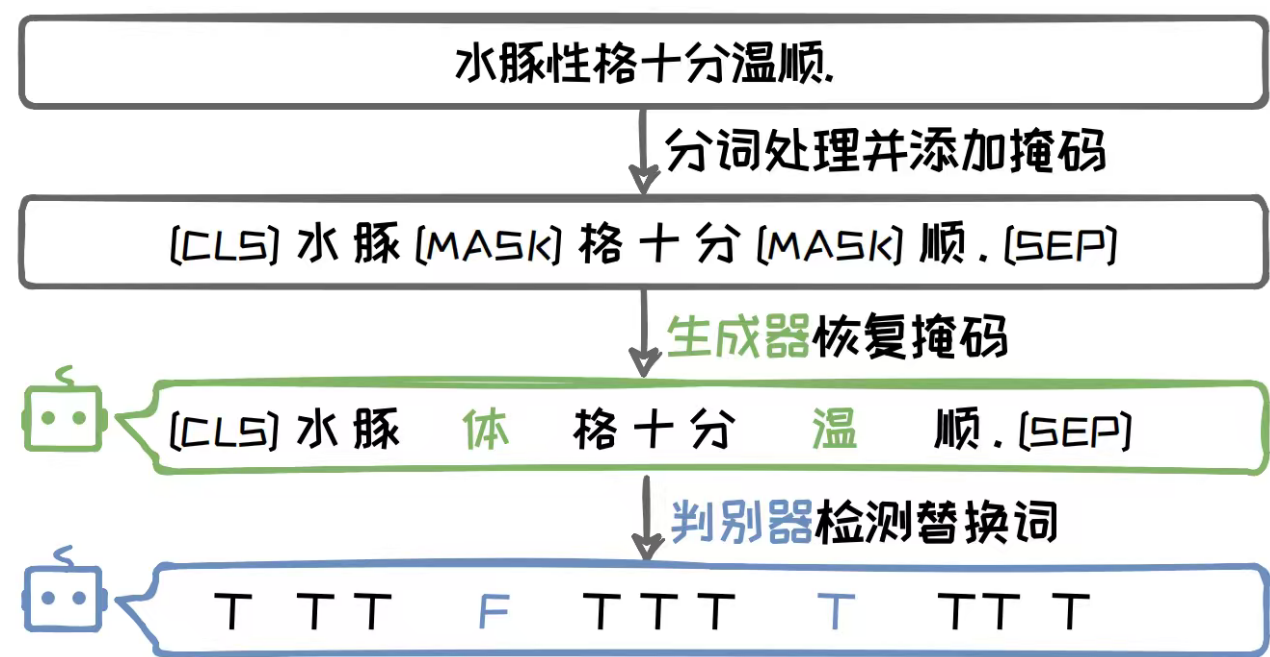
多层Encoder,只训练首层,其他层参数共享
预训练时,将NSP改为Sentence Order Prediction (SOP)



④ ELECTRA

Efficiently Learning on Encoder that Classifies Token Replacements Accurately

类似GAN训练方式,生成器为BERT,预测掩码;判别器使用 Replaced Token Detection (RTD)任务预训练



⑤ 对比

模型	发布时间	参数量 (亿)	语料规模	预训练任务
BERT	2018.10	1.1, 3.4	约 15GB	MLM+NSP
RoBERTa	2019.07	1.2, 3.5	160GB	Dynamic MLM
ALBERT	2019.09	0.12, 0.18, 0.6, 2.2	约 15GB	MLM+SOP
ELECTRA	2020.03	0.28, 2.2, 6.6	约 20-200GB	RTD

4. Encoder-Decoder

① T5

Text-to-Text Transfer Transformer

构造输入前缀, T5 执行不同的任务 (仅需一个模型)



构建 C4 数据集, 任务为 Span Corruption: 选 15% 的 token, 每次选连续的 3 个 token 作为 1 个 span, 整体掩码为 [MASK], 模型预测整个连续片段

下游任务:

Zero-Shot, 使用 Prompt 直接适配
也可微调 Fine-Tuning 适配

变体:

mT5 扩展了 100+ 语言支持

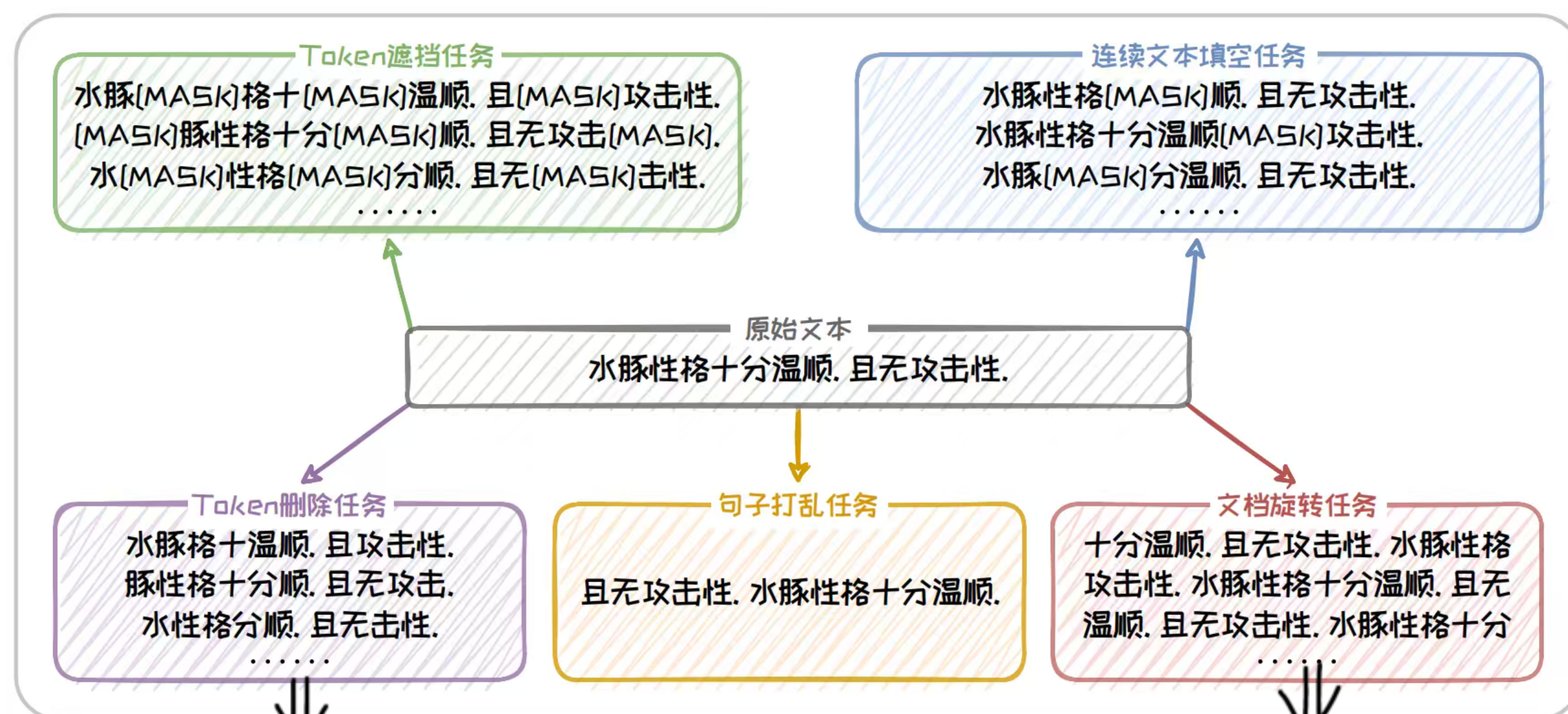
T0 多任务训练增强了 Zero-Shot Learning 能力

FlanT5 专注于指令微调

② BART

Bidirectional and Auto-Regressive Transformers

预训练任务: Token Masking, 连续文本填空 Text Infilling, 句子打乱 Sentence Permutation, Document Rotation, Token Deletion



模型推断删了哪些

模型找到文本合理起点

变体: 跨语言 mBART

③ 对比

模型	发布时间	参数量 (亿)	语料规模
T5	2019.10	0.6-110 亿	750GB
mT5	2020.10	3-130 亿	9.7TB
T0	2021.10	30-110 亿	约 400GB
BART	2019.10	1.4-4 亿	约 20GB
mBART	2020.06	0.4-6.1 亿	约 1TB

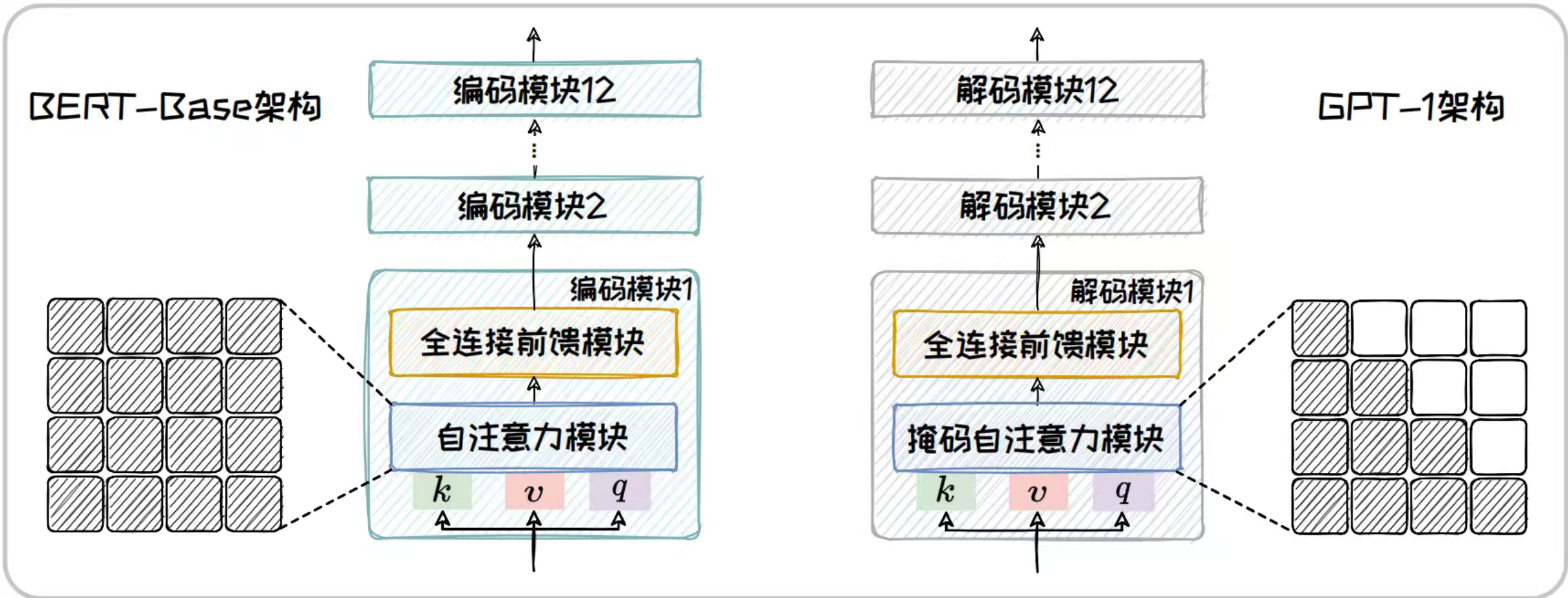
5. Decoder-Only

① GPT系列

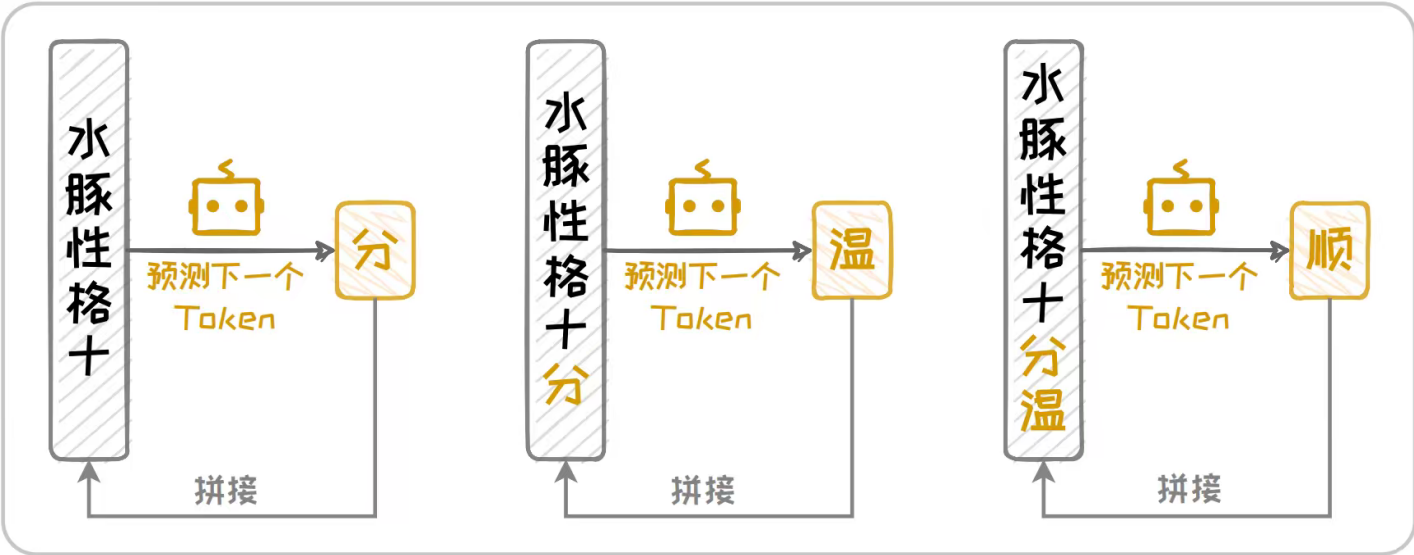
Generative Pre-trained Transformer

模型	发布时间	参数量 (亿)	语料规模
GPT-1	2018.06	1.17	约 5GB
GPT-2	2019.02	1.24 / 3.55 / 7.74 / 15	40GB
GPT-3	2020.05	1.25 / 3.5 / 7.62 / 13 / 27 / 67 / 130 / 1750	1TB
ChatGPT	2022.11	未知	未知
GPT-4	2023.03	未知	未知
GPT-4o	2024.05	未知	未知

(1) GPT-1



预训练任务: BookCorpus上的Next Token Prediction, 无需人为构造标签



下游任务: 有监督微调

文本分类、文本相似度评估、多项选择题解答

(2) GPT-2

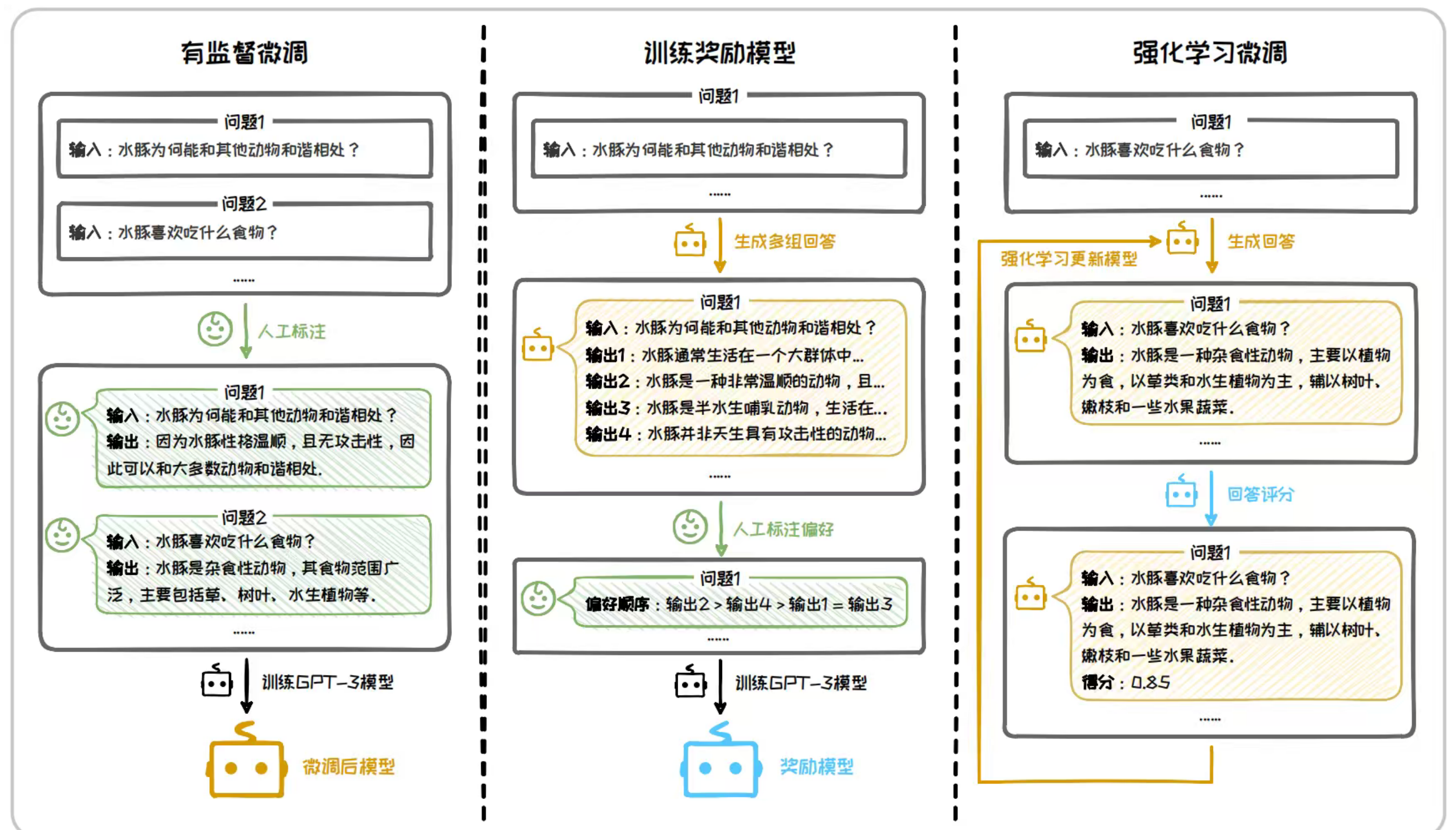
增大模型、增大数据 WebText

(3) GPT-3

出现了上下文学习 In-Context Learning (ICL) 能力

变体: Continual Pre-Training 的 CodeX → 代码生成
User Intent Alignment, 引入人类反馈强化学习 RLHF
的 Instruct GPT → 指令跟随

RLHF 步骤: 有监督微调 → 训练奖励模型 → RL 微调



RLHF 计算成本高昂 → Direct Preference Optimization (DPO)
DPO 无奖励模型, 微调模型使提高最优响应概率, 降低次优响应概率

② LLaMA

Large Language Model Meta AI

模型	发布时间	参数量 (亿)	语料规模
LLAMA-1	2023.02	67 / 130 / 325 / 652	约 5TB
LLAMA-2	2023.07	70 / 130 / 340 / 700	约 7TB
LLAMA-3	2024.04	80 / 700	约 50TB

(1) LLaMA1

小模型 + 大数据

模型架构: Rotary Positional Embeddings (RoPE) 代替原
位置编码 → 增加序列顺序理解
绝对位置编码:

$$p_{i,2t} = \sin\left(\frac{i}{10000^{\frac{2t}{d}}}\right)$$
$$p_{i,2t+1} = \cos\left(\frac{i}{10000^{\frac{2t}{d}}}\right)$$

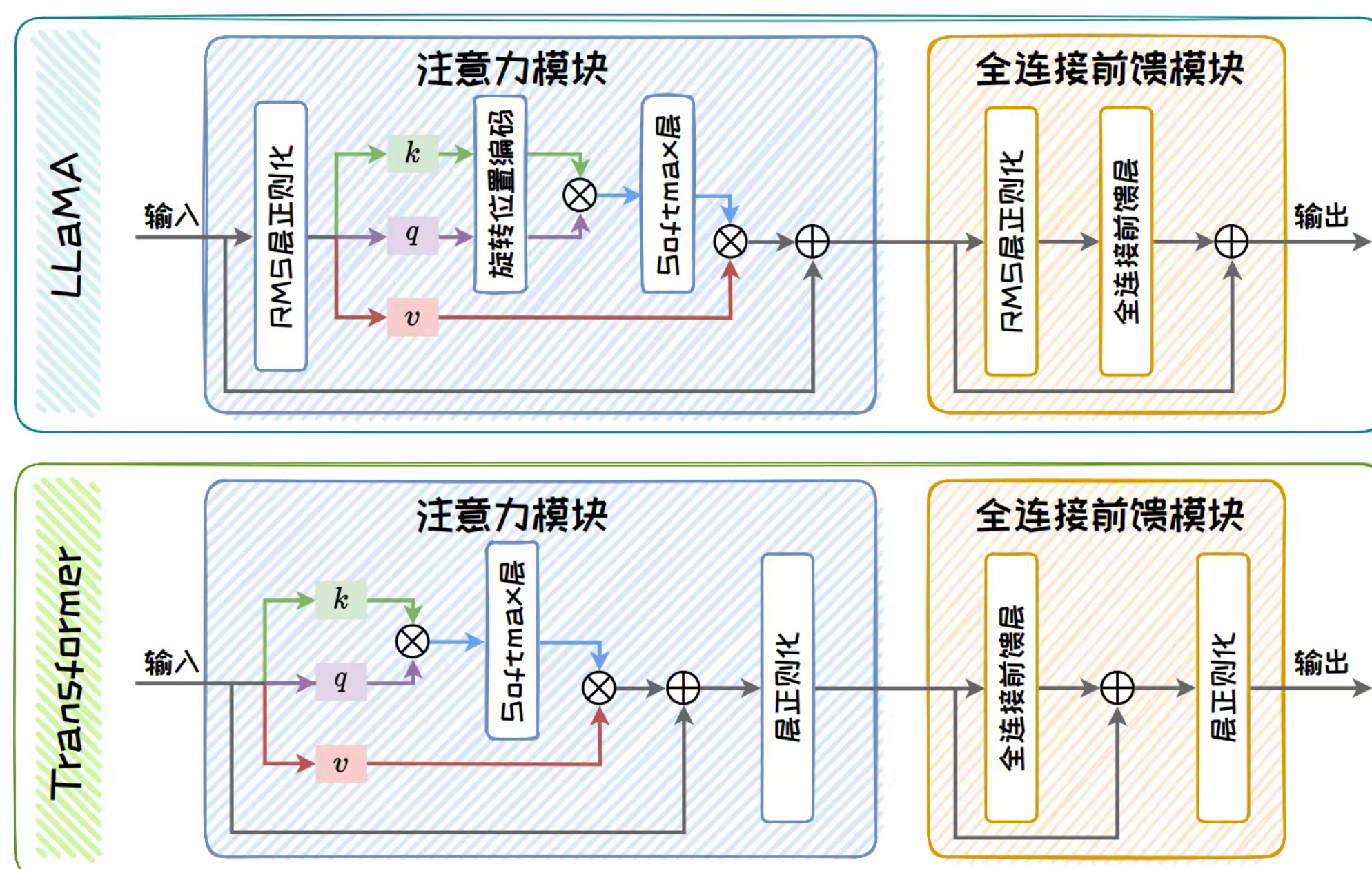
旋转位置编码

$$\begin{aligned}f_q(x_m, m) &= (W_q x_m) e^{im\theta} \\f_k(x_n, n) &= (W_k x_n) e^{in\theta} \\g(x_m, x_n, m - n) &= \text{Re}[(W_q x_m)(W_k x_n)^* e^{i(m-n)\theta}]\end{aligned}$$

考虑了相对位置关系

SwiGLU激活函数代替原有ReLU

Pre-Norm层正则化策略, 用于处理输入



(2) LLaMA2

采用了RLHF, 强化学习微调时采用了Proximal Policy Optimization (PPO)和拒绝采样

LLaMA2-34B和70B增加了 Grouped Query Attention (GQA), 对于不同的Q均共享一组KV

(3) LLaMA3

50TB数据集

Tokenizer阶段字典长度扩大了3倍, 解决中文字符拆为多个token

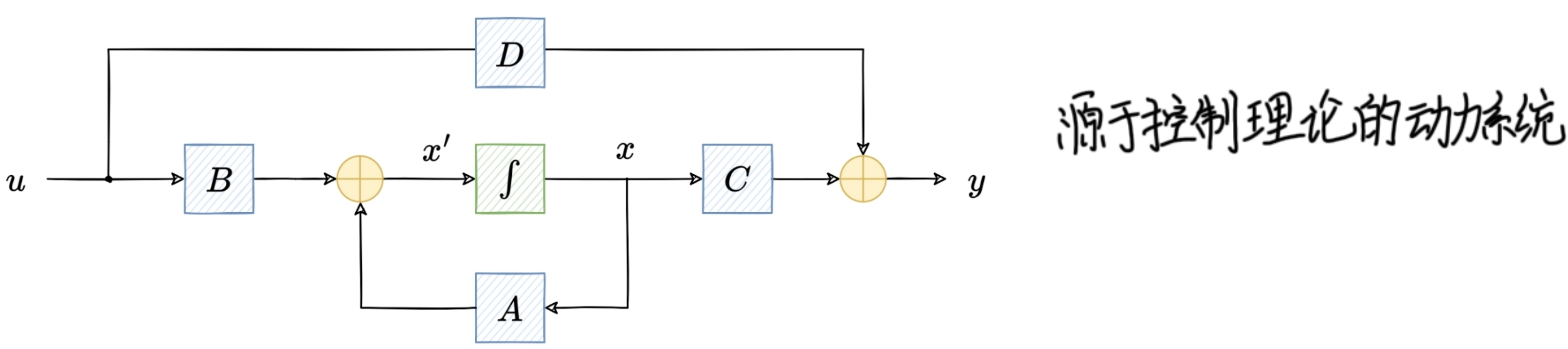
LLaMA3-80B和700B也采用了GQA

6. RNN变体架构

Transformer规模随输入序列长度平方增长 $\Rightarrow$ RNN线性增长

① State Space Model (SSM)

SSM可有效处理 Long-Range Dependencies (LRD)



$u(t)$ 为状态输入, $x(t)$ 为状态变量, $y(t)$ 为状态输出
A状态 B控制 C输出 D命令矩阵

$x'(t) = Ax(t) + Bu(t) \quad \int x'(t) dt = x(t)$

$y(t) = Cx(t) + Du(t)$, 残差连接



离散化: 递归形式 $\Rightarrow$ 推理

$x_k = \bar{A}x_{k-1} + \bar{B}u_k$

$y_k = \bar{C}x_k$

其中 $\bar{A}, \bar{B}, \bar{C}$ 为离散形式下矩阵 A, B, C

$\bar{A} = (I - \frac{\Delta t}{2}A)^{-1} (I + \frac{\Delta t}{2}A), \bar{B} = (I - \frac{\Delta t}{2}A)^{-1} \Delta B, \bar{C} = C, \Delta t = t_{n+1} - t_n$

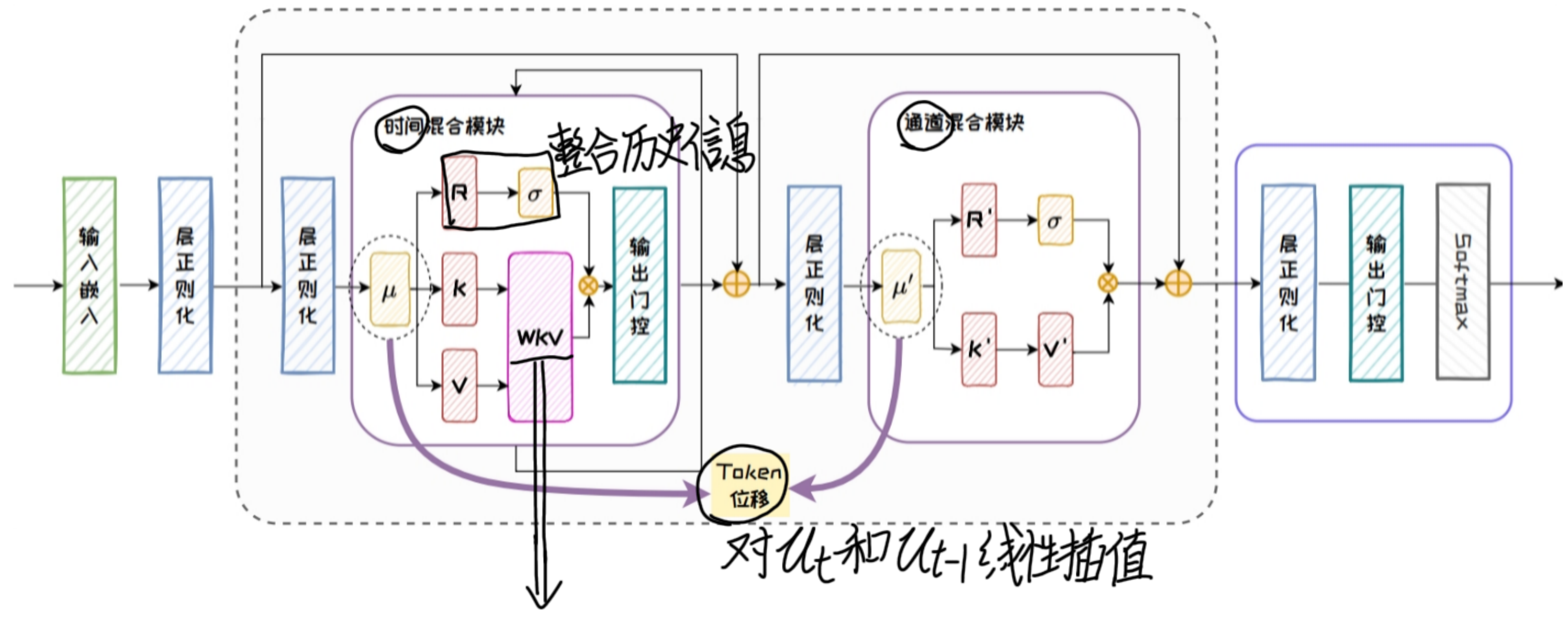
卷积形式 $\Rightarrow$ 训练

$y_k = \bar{K}_k * u_k, \bar{K}_k = (\bar{C}\bar{B}, \bar{C}\bar{A}\bar{B}, \dots, \bar{C}\bar{A}^k\bar{B})$

卷积核 $\bar{K}_k$ 是时不变性的, 因此可高效并行

(1) RWKV

Receptance Weighted Key Value



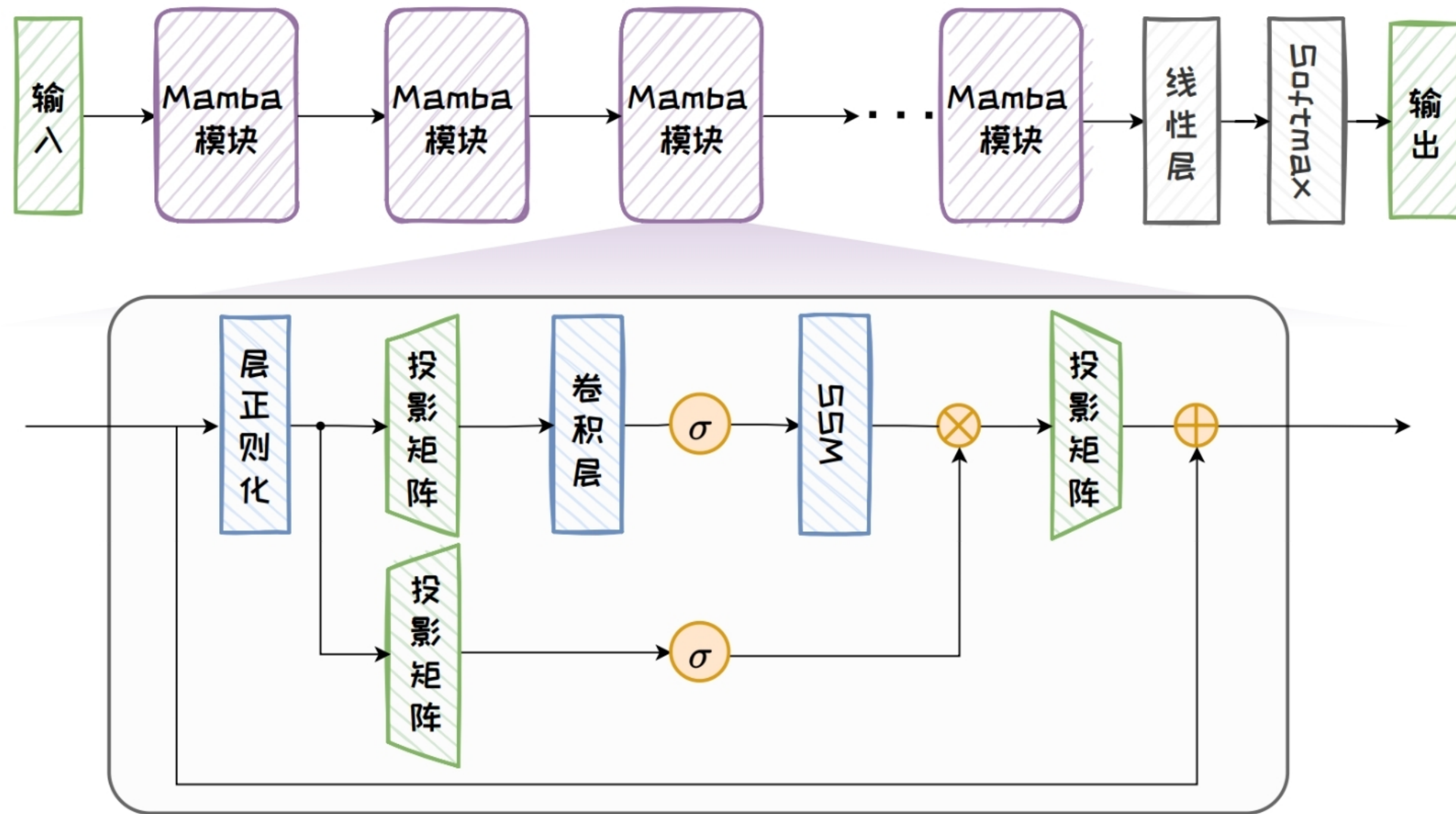
WKV 机制, 使各通道的权重均随时间向后逐步衰减

$wkv_t = \frac{\sum_{i=1}^{t-1} e^{-(t-1-i)w+k_i} \odot v_i + e^{u+k_t} \odot v_t}{\sum_{i=1}^{t-1} e^{-(t-1-i)w+k_i} + e^{u+k_t}}$ 可看作为 2 个 SSM 之比

结合了Transformer和RNN优势,但长序列建模能力较弱

(2) Mamba

选择机制: SSM参数根据输入数据动态变



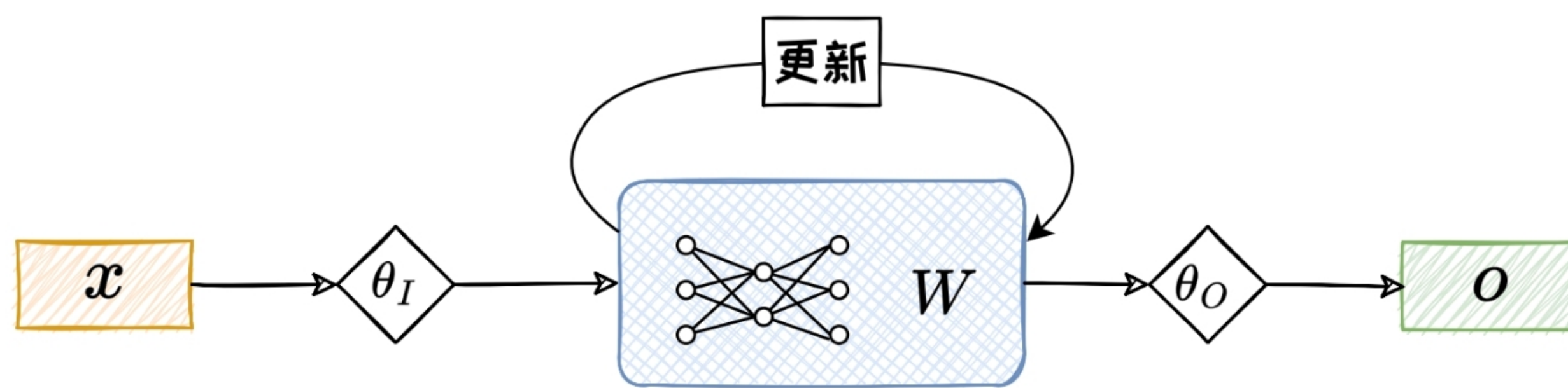
硬件感知算法: 内核融合、并行扫描、重计算

固定长度隐藏状态实现线性复杂度,但上下文 $> 16k$ 时, PPL 不再下降,模型表达能力有限,遗忘关键信息

② TTT

Test-Time Training

每一步推理均更新参数,实现上下文的



外部循环: 词预测,更新模型权重

内部循环: 更新隐藏状态,相当于训练输入信息



模型参数保存上下文信息,可捕捉超长上下文

第二章 Prompt 工程

① Prompt 工程

Prompt 包含四部分：任务说明、上下文、问题、输出格式

Prompt 分词向量化

构建词表，依赖于分词算法 BBPE / BPE / WordPiece

BBPE 算法：初始化，单字编码作为初始 token

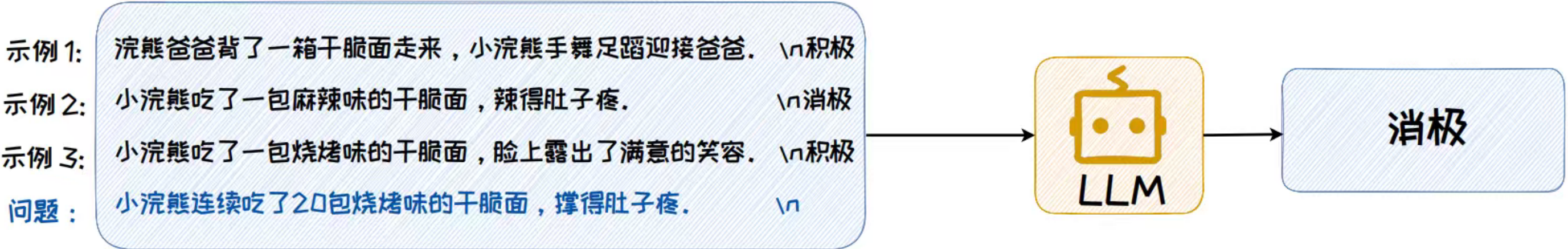
统计所有 token 对 (相邻 token 组合) 出现频率

合并高频 token 对，迭代统计、合并

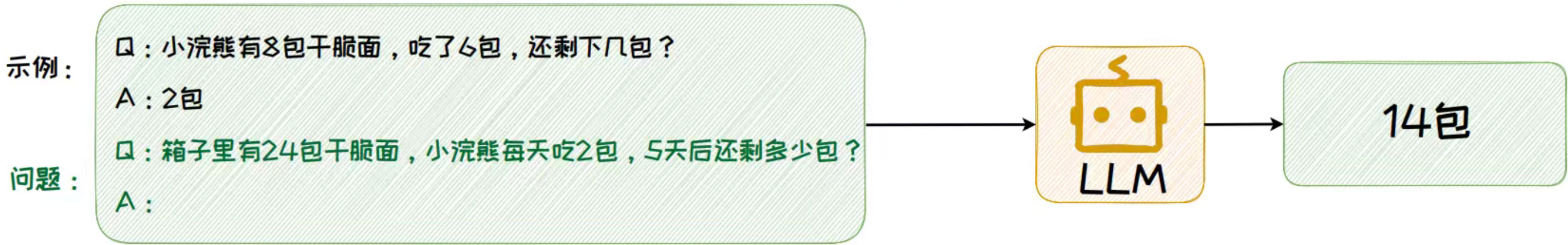
模型	词表大小	中文分词效率 (字 / Token)	英文分词效率 (词 / Token)
LLaMA1	32000	0.6588	0.6891
LLaMA2	32000	0.6588	0.6891
LLaMA3	128256	1.0996	0.7870
DeepSeek-V1	100016	1.2915	0.7625
DeepSeek-V2	100002	1.2915	0.7625
GPT-3.5 & GPT-4	100256	0.7723	0.7867
GPT-3	50,257	0.4858	0.7522
Qwen-1.5	151646	1.2989	0.7865
StarCoder	49152	0.9344	0.6513

分词后，经 Embedding Matrix 处理，转化为表征向量，用于输入

② 上下文学习



(a) 上下文学习做分类任务。



(b) 上下文学习做生成任务。

Zero-Shot: 仅任务说明

One-Shot: 仅提供一个示例

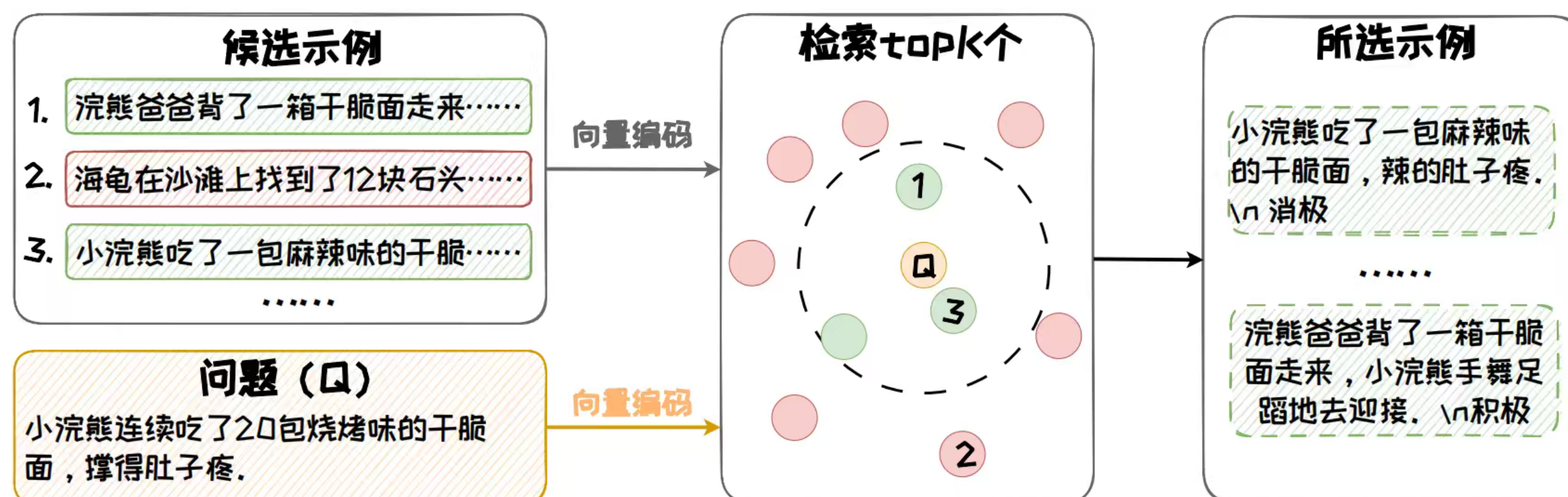
Few-Shot: 提供少量示例

上下文学习可视为隐式贝叶斯推理

演示示例选择

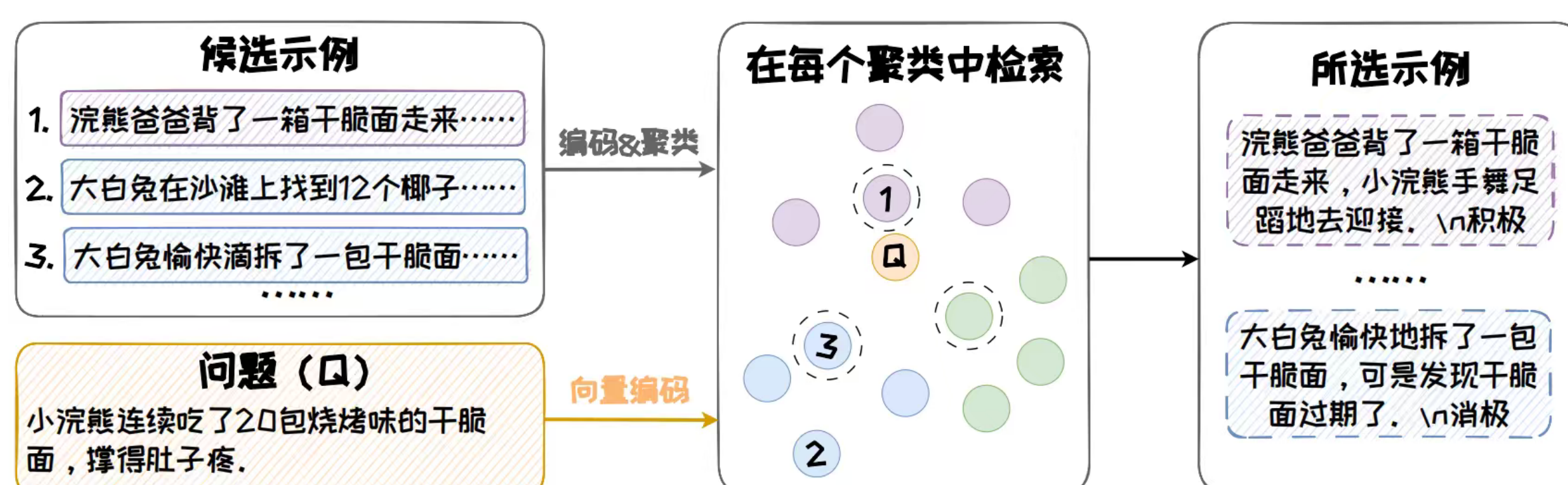
1) 直接选择

KATE用RoBERTa对问题、候选示例编码，计算问题与各候选示例的余弦相似度 $\Rightarrow$ 同质化、未考虑多样性



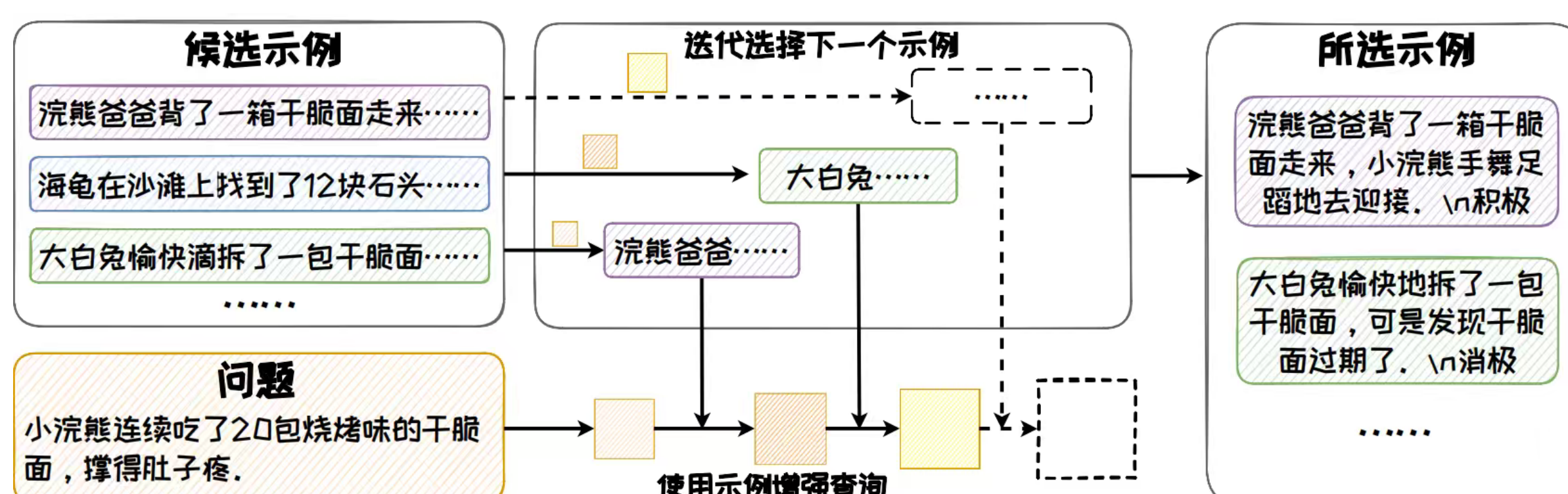
2) 聚类检索

Self-Prompting在每一类中，使用直接选择 $\Rightarrow$ 有些示例簇与问题并不相似，相似度不高



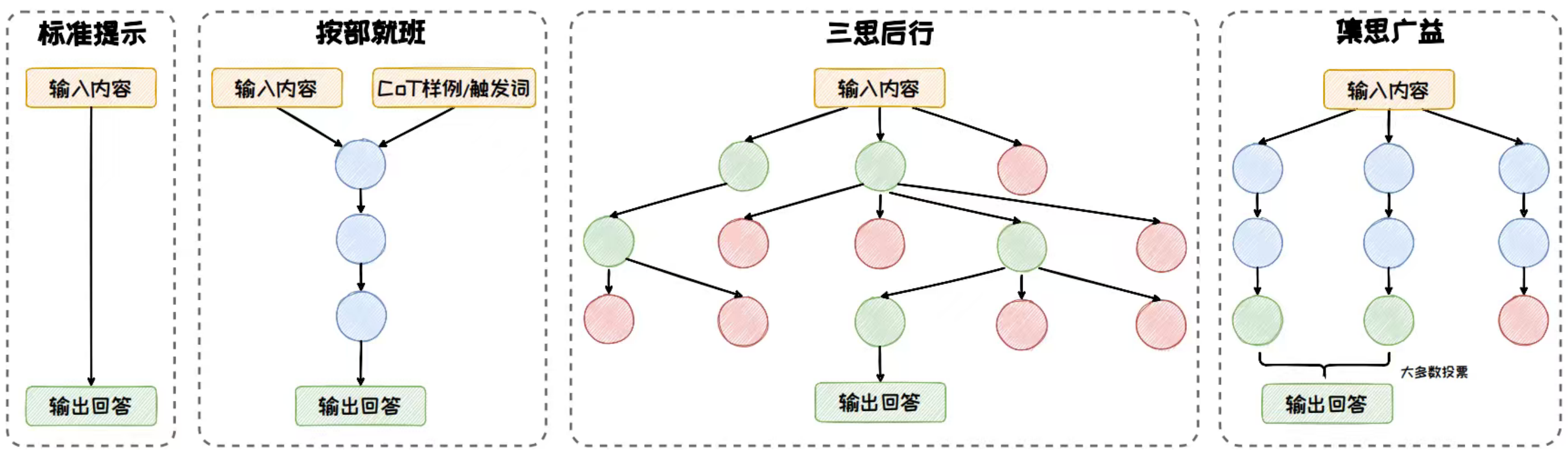
3) 迭代检索

RetICL使用LSTM检索器，每选择一个示例就更新LSTM参数



③ 思维链

面对算术求解、常识判断、符号推理，模型参数增长但性能未突破，叫做Flat Scaling Curves $\Rightarrow$ Chain-of-Thought (CoT) 引导模型生成时引入中间推理步骤

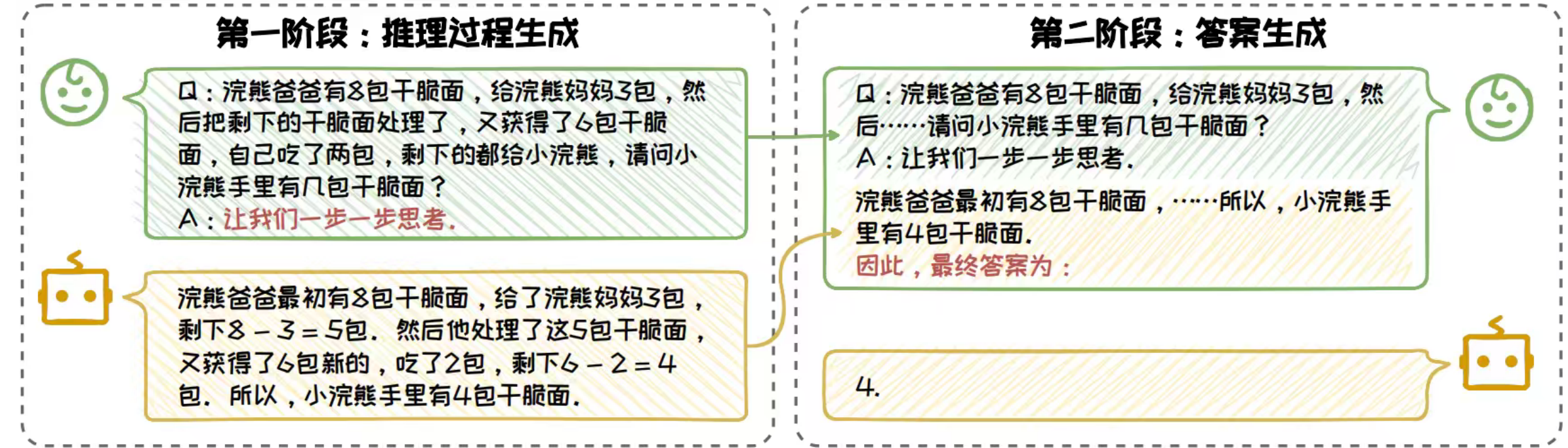


1) 按部就班

Zero-Shot CoT

第一阶段: Let's think step by step

第二阶段: 原始问题 + 第一阶段生成推理步骤 + Therefore, the answer is ?

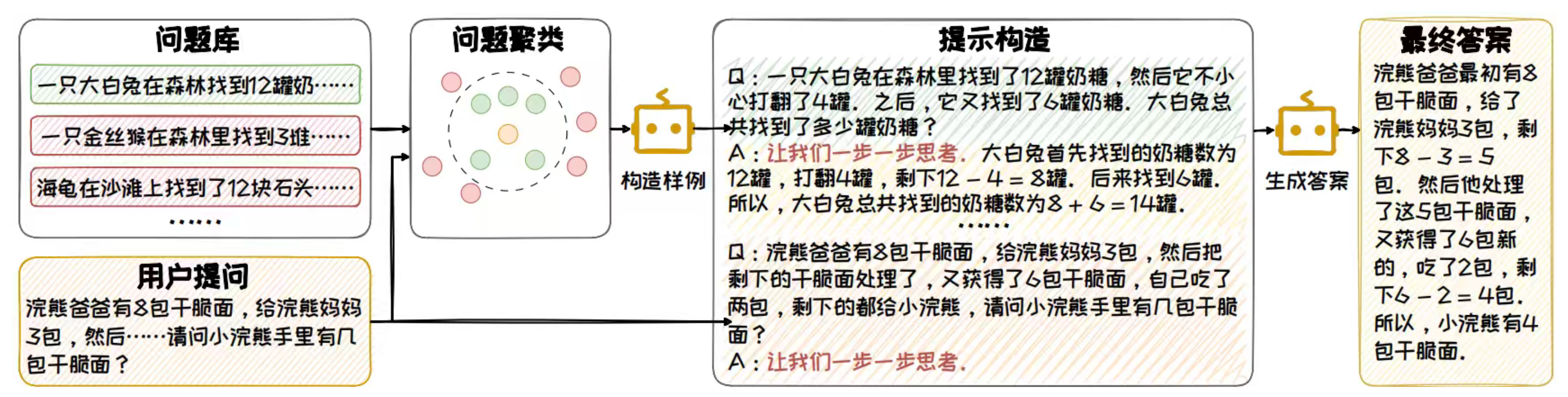


Auto-CoT

第一步: 聚类, 选出同簇问题

第二步: Zero-Shot CoT, 为每个问题生成推理链条, 形成示例, 为模型提供不同题解思路

第三步: 引导生成用户问题的推理链和答案



2) 三思后行

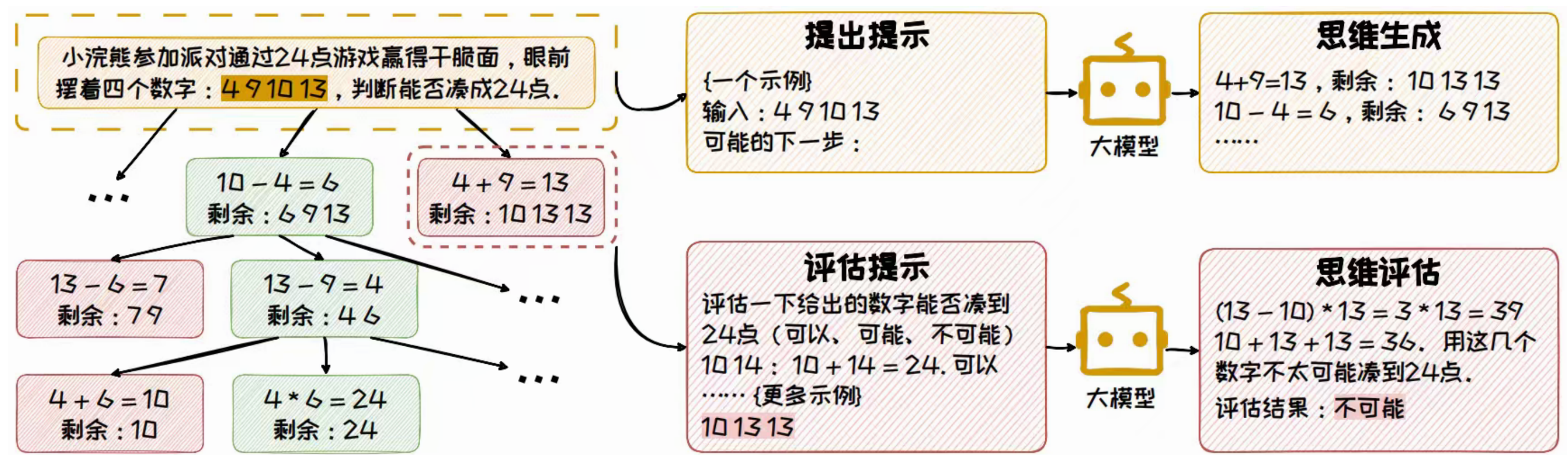
Tree-of-Thoughts (ToT)

第一步: 拆解。复杂问题 → 多个子问题

第二步: 衍生。样本启发, 用于创造性、思维宽泛的任务
命令提示, 用于受限任务, 如24点游戏

第三步: 评估。投票模式, 依票数保留结点,
打分模式, 依分数保留结点,

第四步：搜索。DFS/BFS/A*/蒙特卡洛树搜索



Graph-of-Thoughts (GOT)

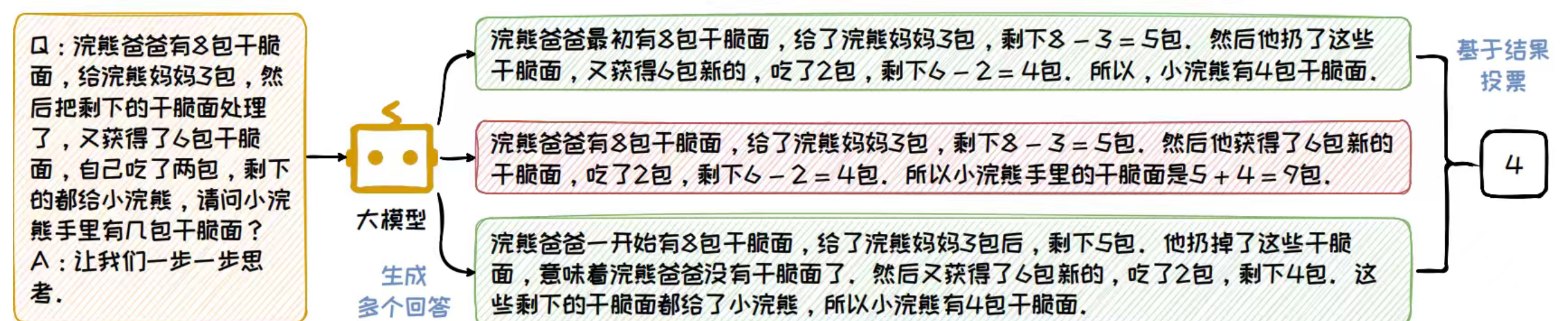
结点代表某初始/子问题及其解决方案
优势：自我反思、思维聚合

3) 集思广益

Self-Consistency

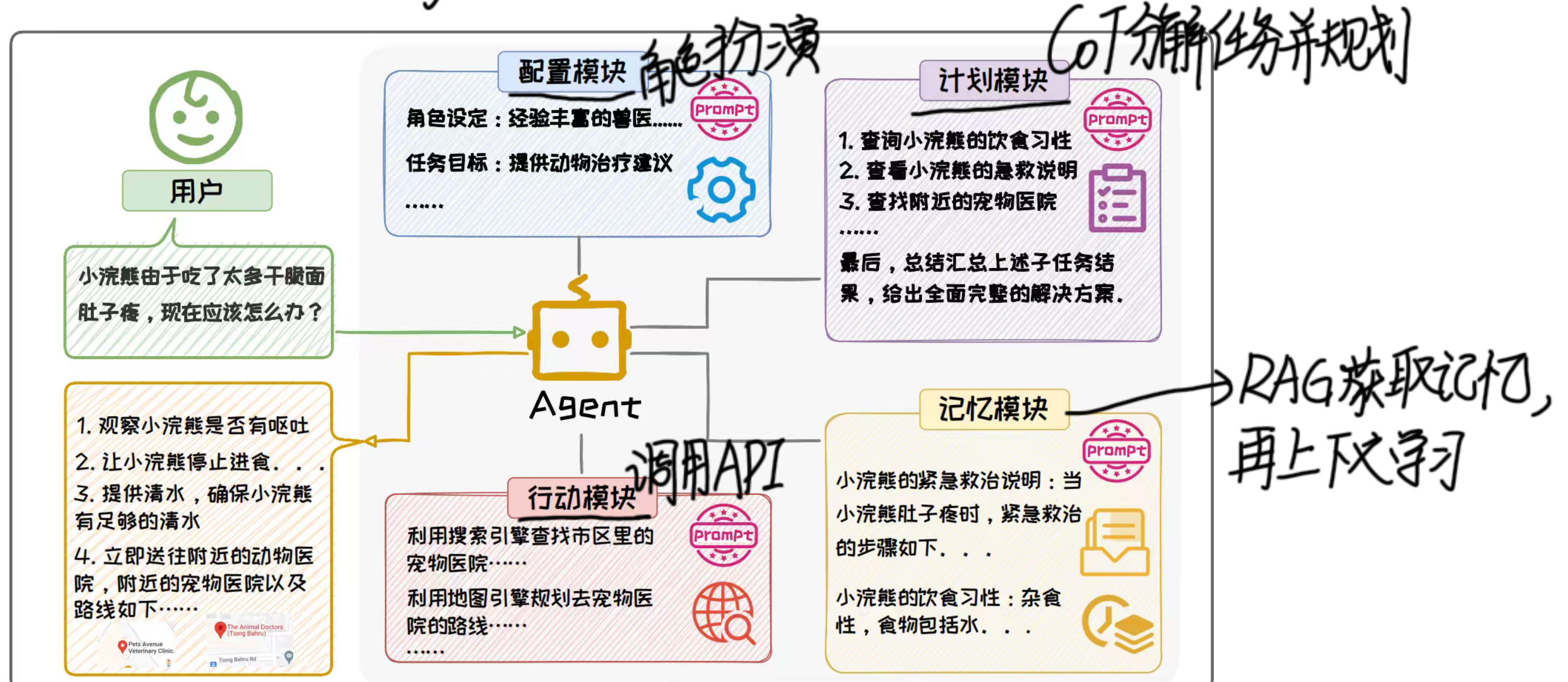
第一步：CoT/Zero-CoT生成一组多样化推理路径

第二步：每条路径都生成最终答案，统计其在所有路径中出现频率，选择频率最高的答案



④ Prompt 工程应用

1) 基于LLM的Agent

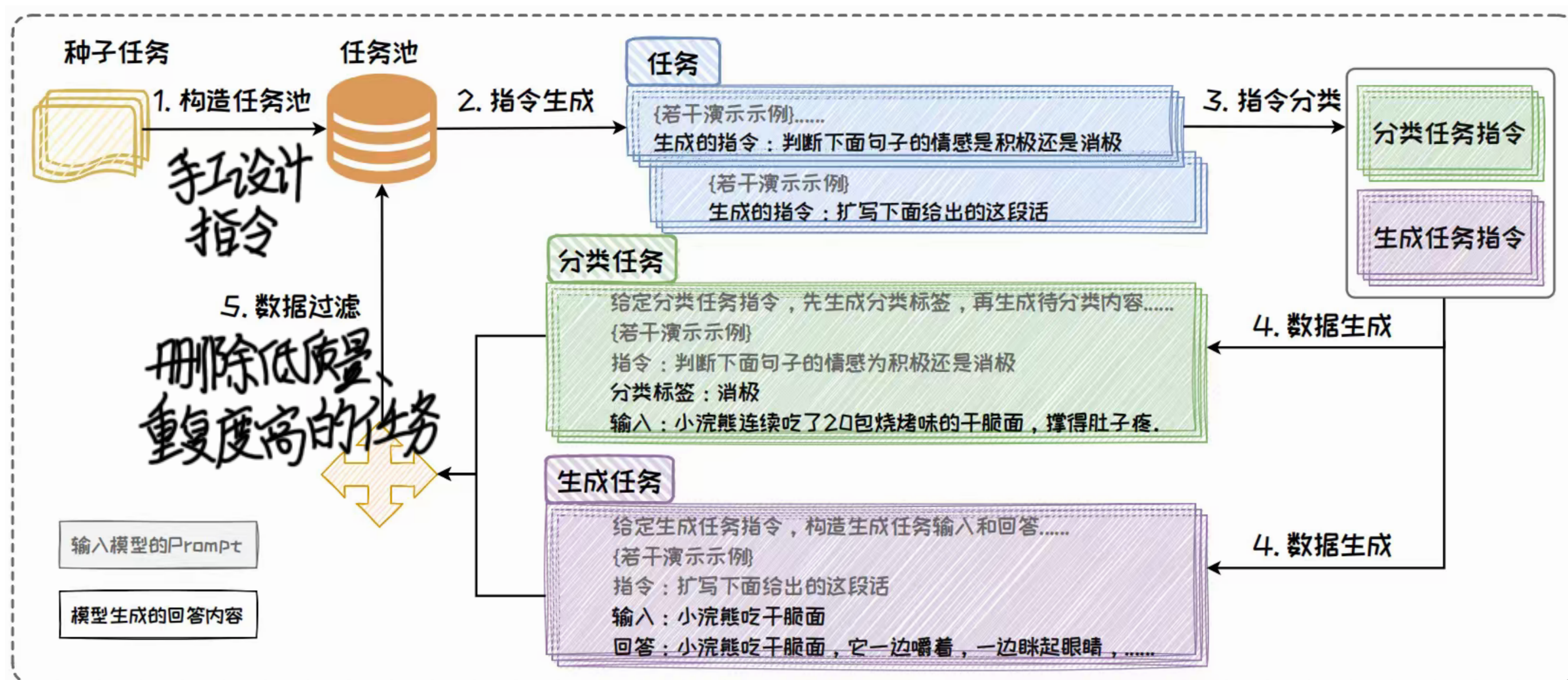


eg. HuggingGPT

以ChatGPT为控制器, 先拆分为多个子任务, 再调用Huggingface上的不同模型来解决子任务

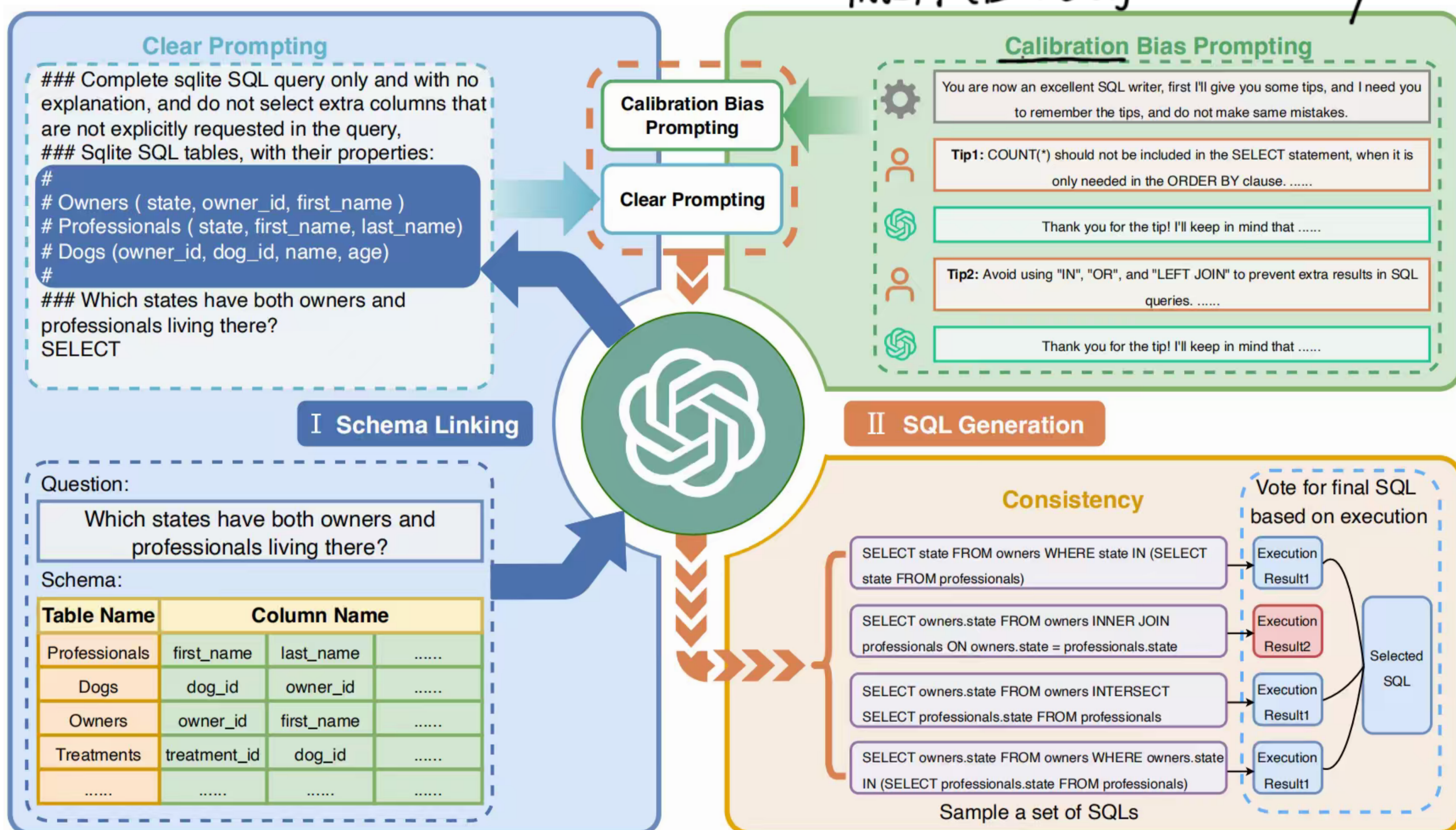
2) 数据合成

Self-Instruct: 构建任务池 → 指令生成 → 指令分类 → 数据生成 → 数据分类



Text-to-SQL: C3

提示校准: 角色扮演 + 先验知识
输出校准: Self-Consistency



第四章 参数高效微调

① 下游任务适配

(1) 上下文学习

Prompt设计耗费人力成本,效果较差

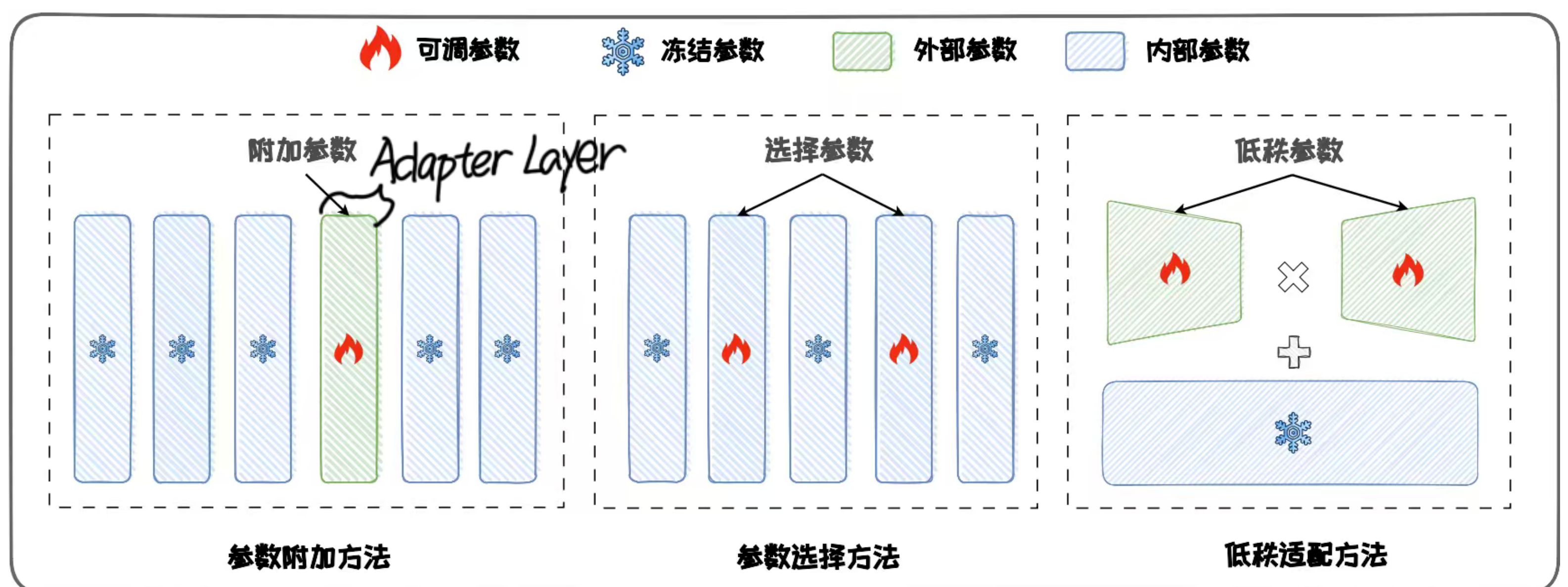
Prompt中样例增多导致推理代价提升

(2) 指令微调

构建指令数据<输入,输出>对 $\rightarrow$ 监督微调

耗费计算资源

(3) Parameter-Efficient Fine-Tuning (PEFT)



② 参数附加方法

(1) 加在输入

Prompt-tuning

包含 n 个 token 的文本序列 $\{w_1, \dots, w_n\}$, 转化为输入嵌入矩阵

$X \in \mathbb{R}^{n \times d}$ (d 为嵌入空间维度)

软提示 soft prompt 的嵌入矩阵 $P \in \mathbb{R}^{m \times d}$ (m 为软提示长度)

$\Rightarrow$ 拼接 $[P; X] \in \mathbb{R}^{(m+n) \times d}$ 输入模型

训练时仅更新 P , 不改变模型参数

(2) 加在模型

Prefix-Tuning

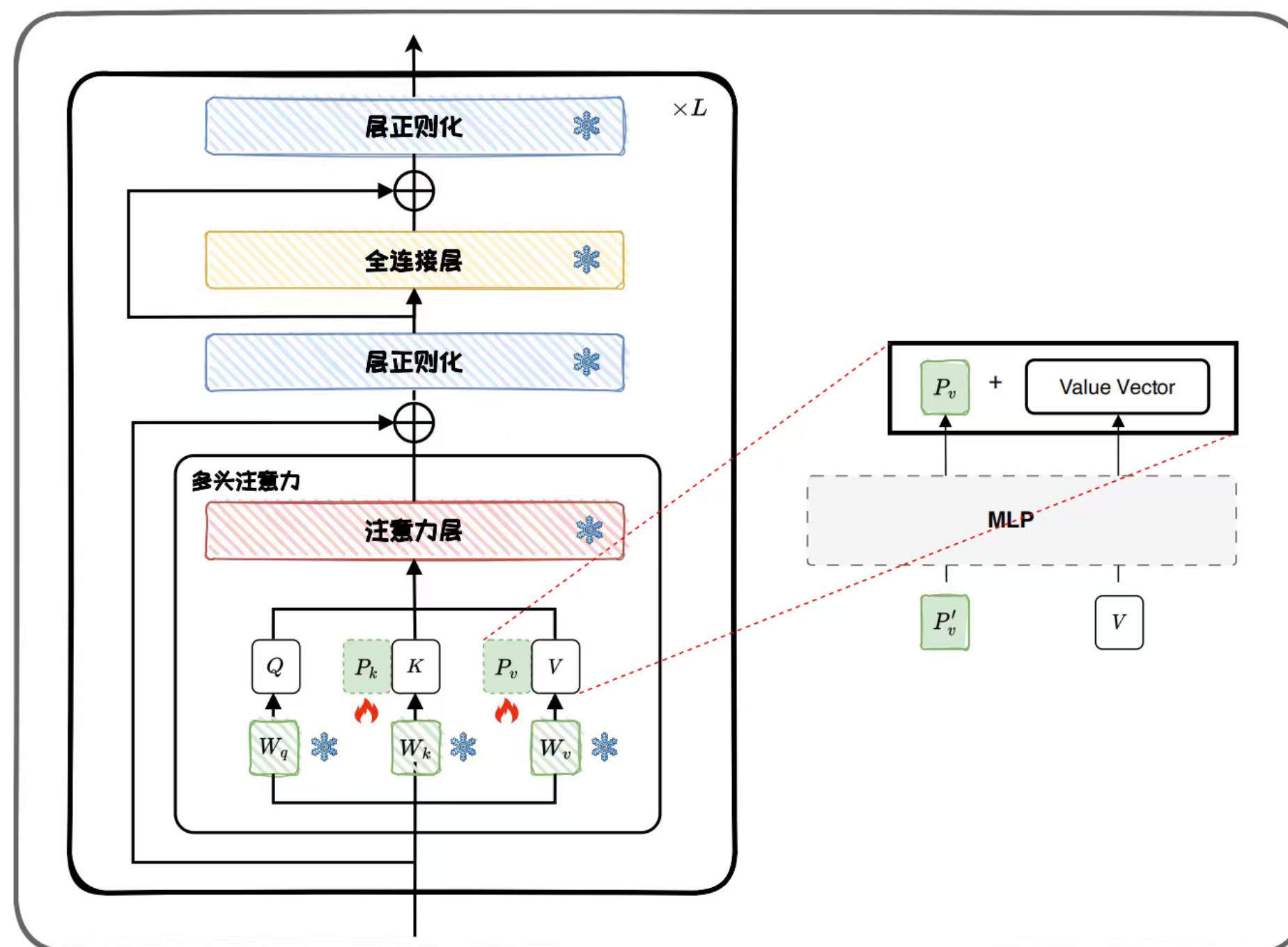
将一系列可训练前缀插入到 Transformer 中

$$K, V \rightarrow [P_K; K], [P_V; V]$$

为了解决 Prefix 更新不稳定, 优化过程难收敛的问题

⇒ MLP 重参数化

训练时更新 P_K, P_V 前缀矩阵; 推理时丢弃 MLP



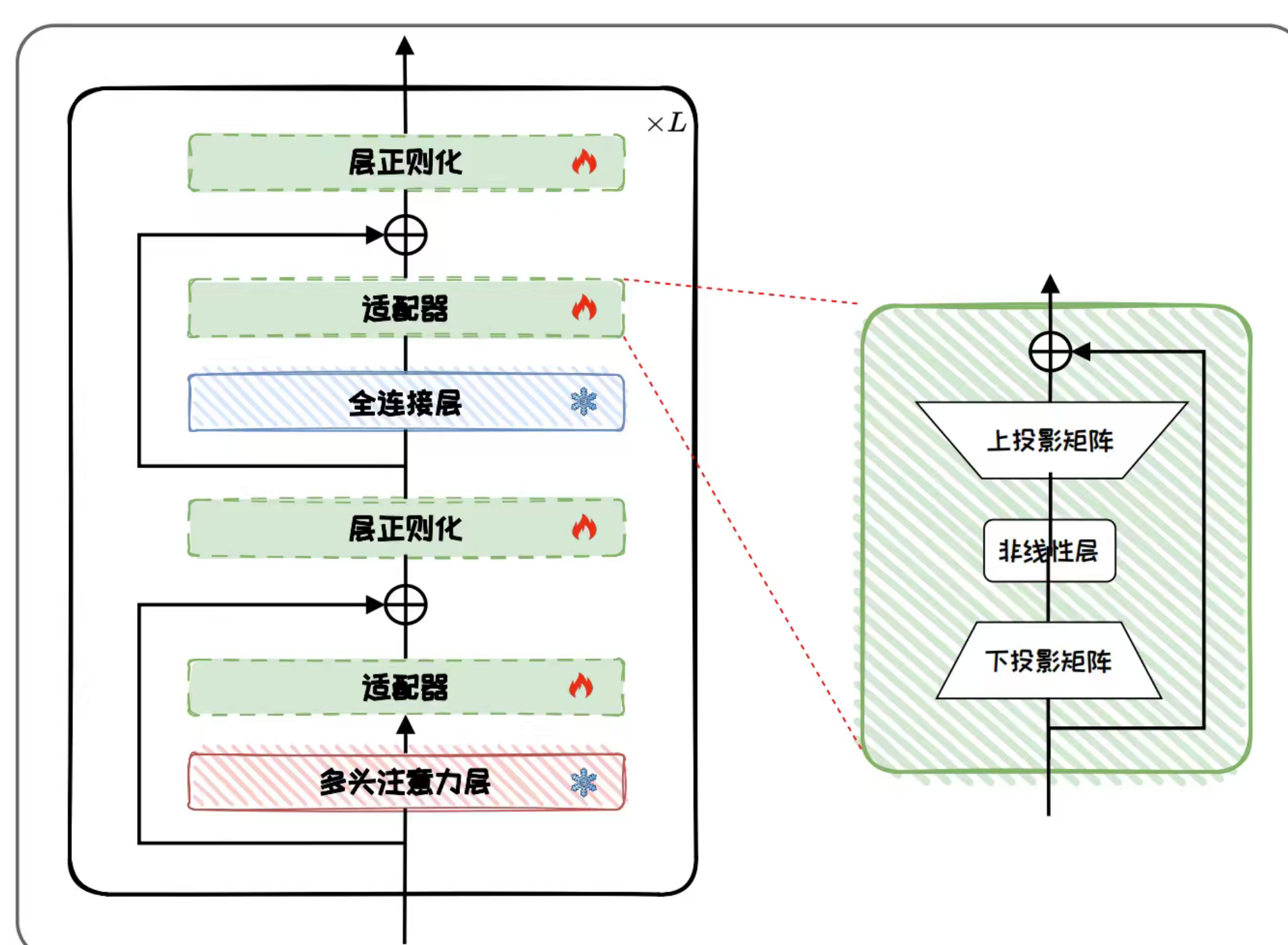
Adapter-Tuning

向模型中插入可学习的神经网络模块 (Adapter)

Adapter 采用 Bottomneck 结构: 上投影、非线性、下投影

Adapter 加在每个 FCN 和 Multi-Head Attention 层后

训练时, 仅训练 Adapter、Layer Norm、最终的分层



Adapter Fusion

第一阶段: 知识提取

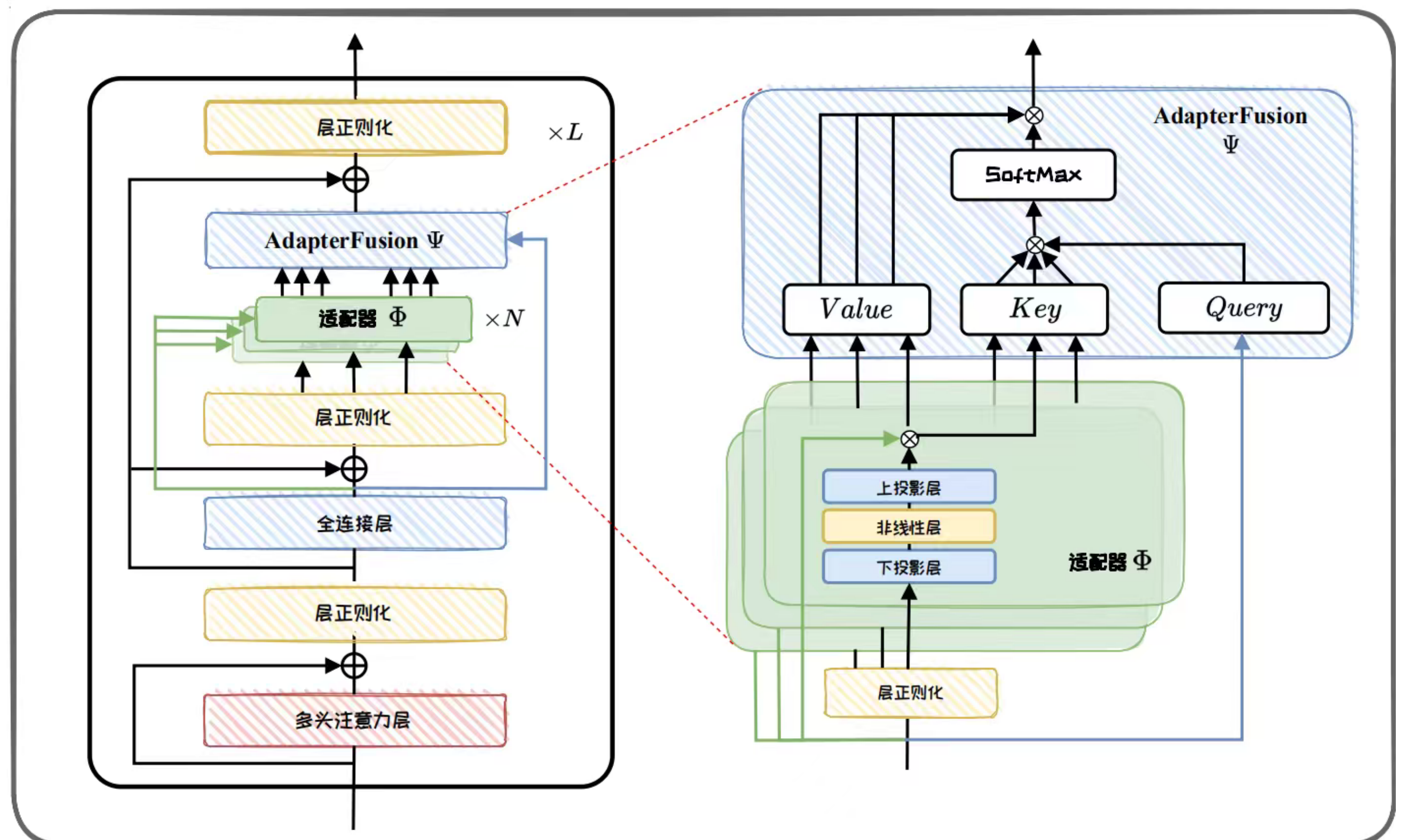
Single-Task Adapters (ST-A): 各任务间模型独立优化

Multi-Task Adapters (MT-A): 多任务学习, 联合优化各模型

第二阶段：知识组合

引入融合模块，仅更新其参数，固定 N 个Adapter参数
对第 $n \in [1, N]$ 个任务数据 D_n ，损失 L_n ，预训练模型 Θ ，
第 $i \in [1, N]$ 个任务适配器 Φ_i ，融合模块 Ψ ：

$$\bar{\Psi} \leftarrow \underset{\bar{\Psi}}{\operatorname{argmin}} L_n(D_n; \Theta, \Phi_1, \Phi_2, \dots, \Phi_N, \bar{\Psi})$$



Adapter Fusion的Query为原模型FCN输出，Key和Value为Adapter的输出

(3) 加在输出

Proxy-Tuning

仅访问模型的输出词汇表的生成的概率分布，无需访问权重
待微调的代理模型 M ，小的反专家模型 M^-

第一步，微调 M^- ，得到专家模型 M^+

推理时，计算 M^+ 和 M^- 之间的logits分布差异，将其加至 M 下一词预测的logits分布中。即对于输出分数 S ，

$$\tilde{S} = S_M + S_{M^+} - S_{M^-}$$

归一化后得到概率分布

$$P_M(X_t | X_{<t}) = \operatorname{softmax}(\tilde{S})$$

最后， X_t 从分布中采样即可

③ 参数选择方法

(1) 基于规则的方法

BitFit 仅优化每层的 Bias 项和最后的分类头

PaFi 只训练绝对值最小的模型参数

(2) 基于学习的方法

Child-Tuning: 第 t 轮迭代的参数矩阵 W_t , 引入同尺寸掩码 M_t 用于选择子网络 C_t , 仅训练子网络 C_t

$$M_t^{(i)} = \begin{cases} 1, & \text{if } W_t^{(i)} \in C_t \\ 0, & \text{if } W_t^{(i)} \notin C_t \end{cases}$$

梯度更新的公式为

$$W_{t+1} = W_t - \eta \left(\frac{\partial \mathcal{L}(W_t)}{\partial W_t} \odot M_t \right)$$

生成子网络掩码的方式

任务无关 (Child-Tuning_F)

从伯努利分布中采样

$M_t \sim \text{Bernoulli}(p_F)$ p_F 表示子网络比例

同时引入噪声正则化, 防止过拟合

任务驱动 (Child-Tuning_D)

用费舍尔信息矩阵估计参数重要性

$$F^{(i)}(W) = \frac{1}{|\mathcal{D}|} \sum_{j=1}^{|\mathcal{D}|} \left(\frac{\partial \log p(Y_j | X_j; W)}{\partial W^{(i)}} \right)^2$$

其中 X_j, Y_j 表示第 j 个样本的输入和输出

$F^{(i)}(W)$ 越高, 第 i 个参数在任务下越重要

④ 低秩适配方法

(1) LoRA

Low-Rank Adaptation

网络参数 $W_0 \in \mathbb{R}^{d \times k}$, 待学习的参数更新矩阵 $\Delta W \in \mathbb{R}^{d \times k}$,

$$W = W_0 + \Delta W$$

对 ΔW 进行矩阵分解, 得到低秩矩阵 $B \in \mathbb{R}^{d \times r}$ 和 $A \in \mathbb{R}^{r \times k}$, 更新过程为

$$W = W_0 + \alpha \underline{BA}$$

仅微调 A, B 两个矩阵

其中 α 为缩放因子, B 由高斯分布初始化, A 初始化为零矩阵, 秩 $r \ll \min\{d, k\}$

训练参数量由 $d \times k$ 变为 $d \times r + r \times k \Rightarrow$ Weight & Activation 内存少量增加, Gradient 和 Optimization 内存显著降低

(2) 变体

打破低秩瓶颈: ReLoRA

低秩更新 $\Rightarrow$ 记忆新知识、适应下游任务能力受限

微调时, 周期性地将 LoRA 模块合并到大模型中, 并重新初始化 LoRA $\Rightarrow$ 多次低秩更新累积成高秩状态

动态秩分配: AdaLoRA

冗余的 LoRA 秩 $\Rightarrow$ 性能和效率的退化 $\Rightarrow$ AdaLoRA
模型每层权重的重要性不同

AdaLoRA 使用奇异值分解表示 ΔW

$$W = W_0 + \Delta W = W_0 + P \Lambda Q$$

其中 $P \in \mathbb{R}^{d \times r}$, $Q \in \mathbb{R}^{r \times k}$ 是正交的, Λ 是对角阵 (对角元 λ_i)

根据梯度 $\times$ 权重的移动平均值判断奇异值 λ_i 重要性, 剪掉不重要的 λ_i

引入惩罚项确保 P, Q 之间的正交性

$$R(P, Q) = \|P^T P - I\|_F^2 + \|Q Q^T - I\|_F^2$$

其中 $\|\cdot\|_F$ 为 Frobenius 范式

训练过程优化: DoRA

LoRA 收敛速度慢、对超参数敏感、易过拟合 $\Rightarrow$ 约束梯度更新, 更新参数方向

将 $W_0 \in \mathbb{R}^{d \times k}$ 表示为

$$W_0 = m \frac{V}{\|V\|_c} = \|W_0\|_c \frac{W_0}{\|W_0\|_c}$$

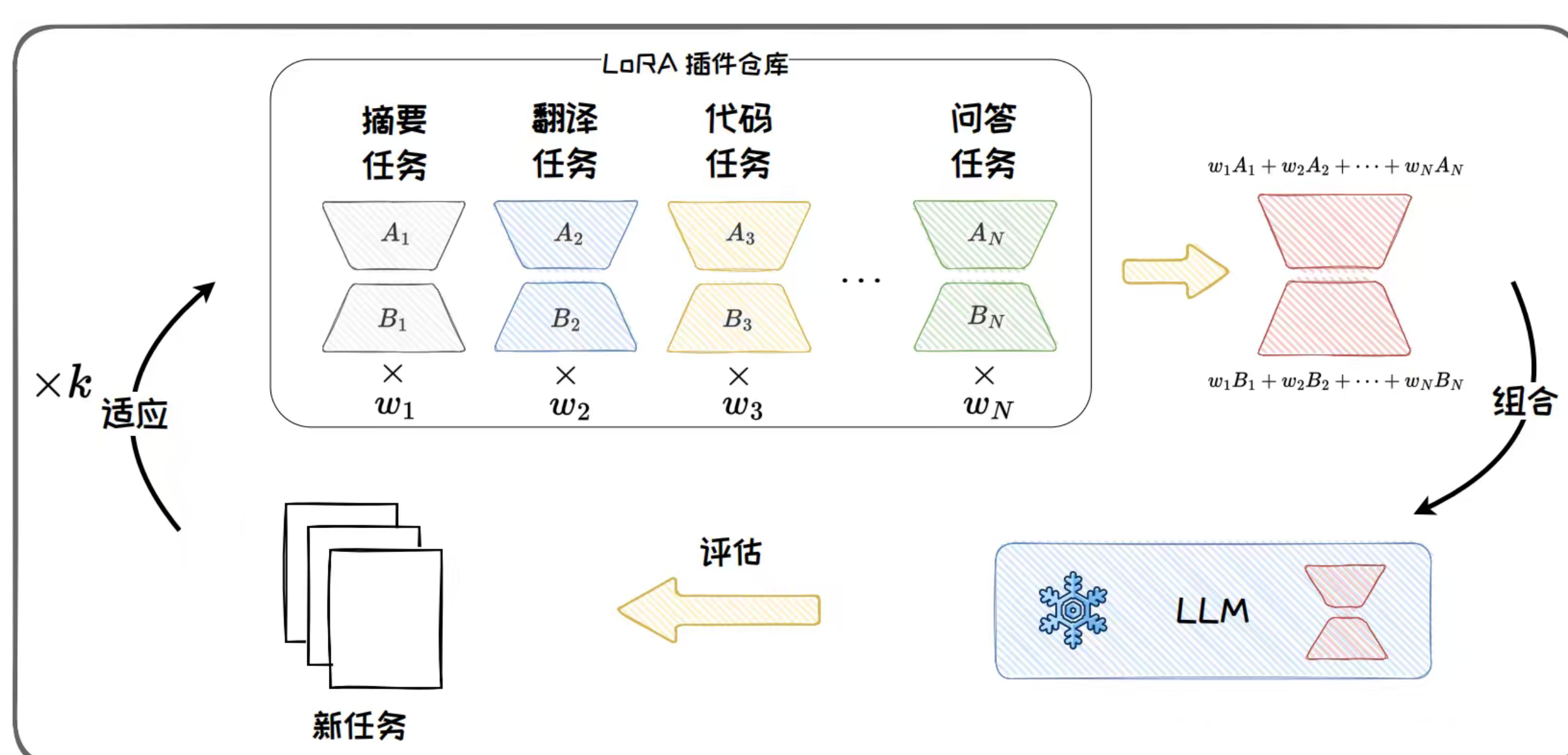
其中, $m \in \mathbb{R}^{1 \times k}$ 为大小向量, $V \in \mathbb{R}^{d \times k}$ 为方向矩阵, $\|\cdot\|_c$ 为每一列上的范数

仅对方向阵 V 使用 LoRA

$$W' = m \frac{V + \Delta V}{\|V + \Delta V\|_c} = m \frac{W_0 + BA}{\|W_0 + BA\|_c}$$

其中, m, B, A 为可训练参数

⑤ LoRA 插件的任务泛化



LoRA Hub

第一阶段: 组合

将已微调的 LoRA 模块逐元素加权和

$$\hat{m} = (w_1 A_1 + w_2 A_2 + \dots + w_N A_N) (w_1 B_1 + w_2 B_2 + \dots + w_N B_N)$$

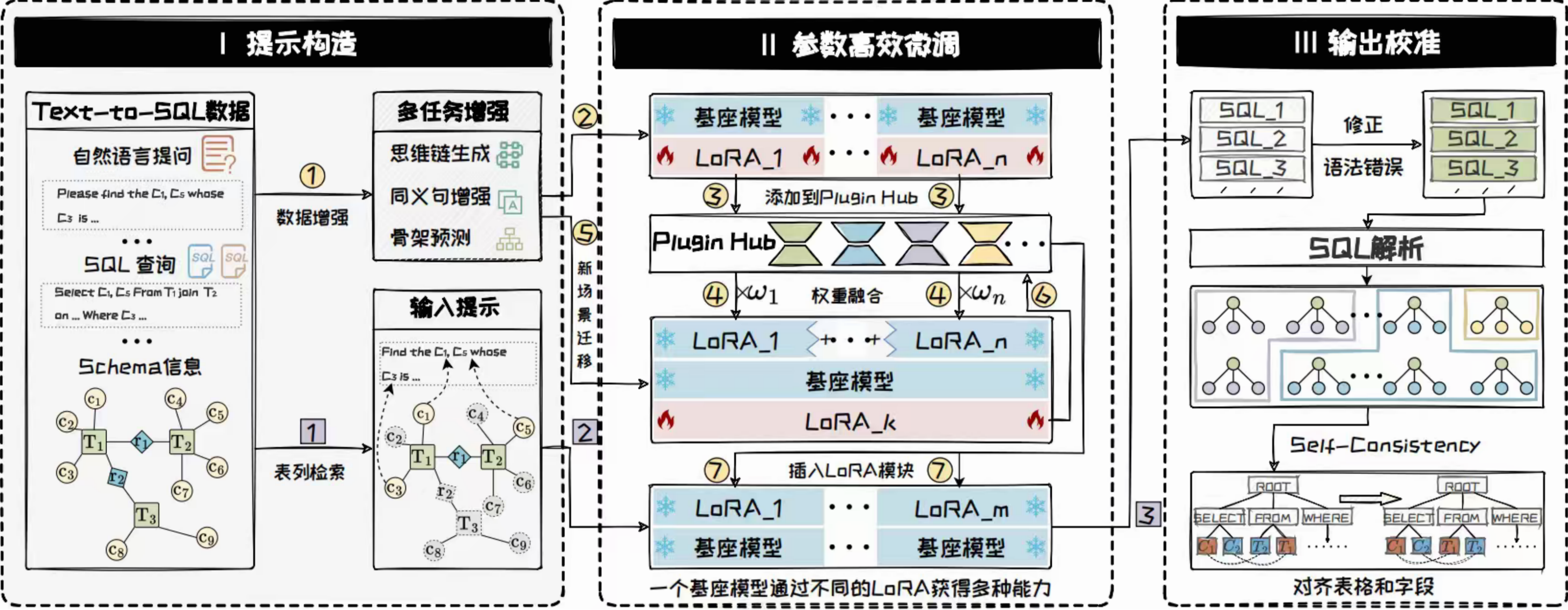
其中 $A_{i=1}^N, B_{i=1}^N$ 是第 i 个 LoRA 分解矩阵

第二阶段: 适应

无梯度的 Shiva 学习权重组合: 协方差矩阵自适应进化 (CMA-ES)

⑥ 应用

FinSQL



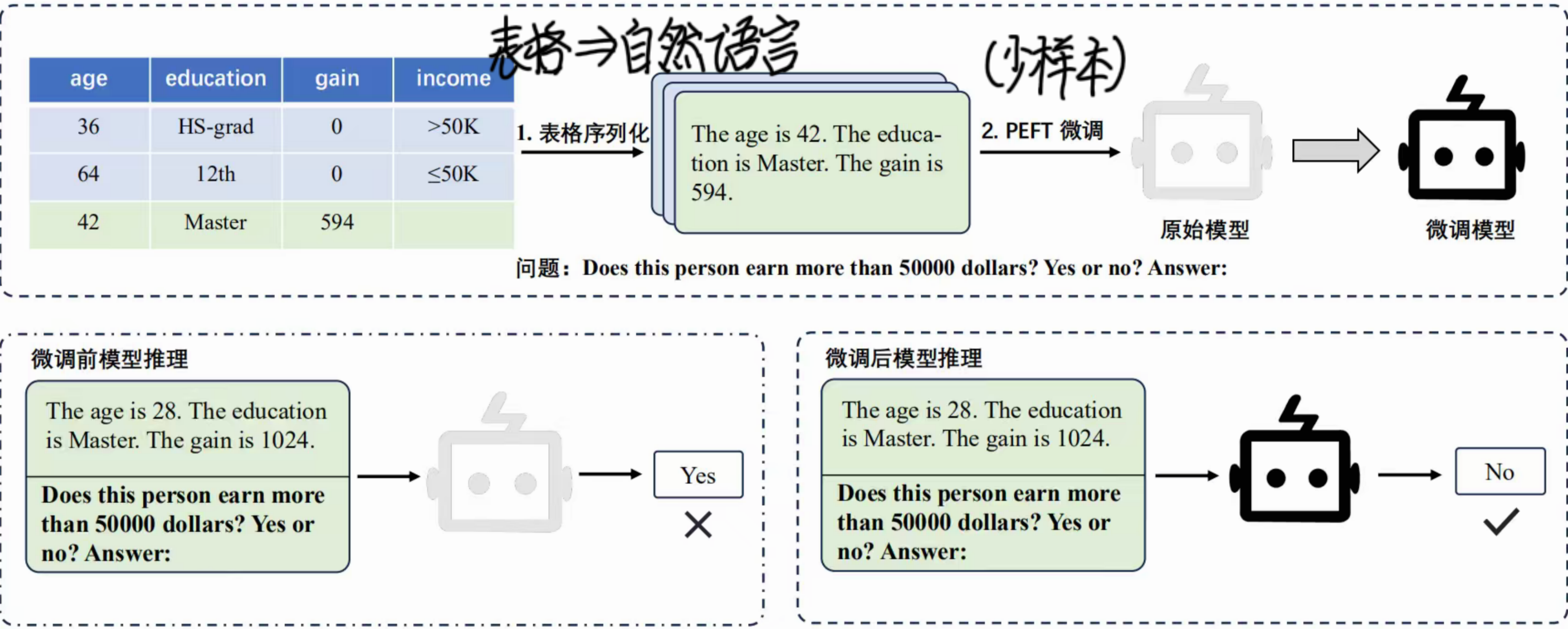
各种策略增强原数据

微调基座模型
LoRA Hub融合多模型

Self-Consistency

TabLLM

少量样本上全量微调易过拟合 $\Rightarrow$ PEFT



第五章 模型编辑

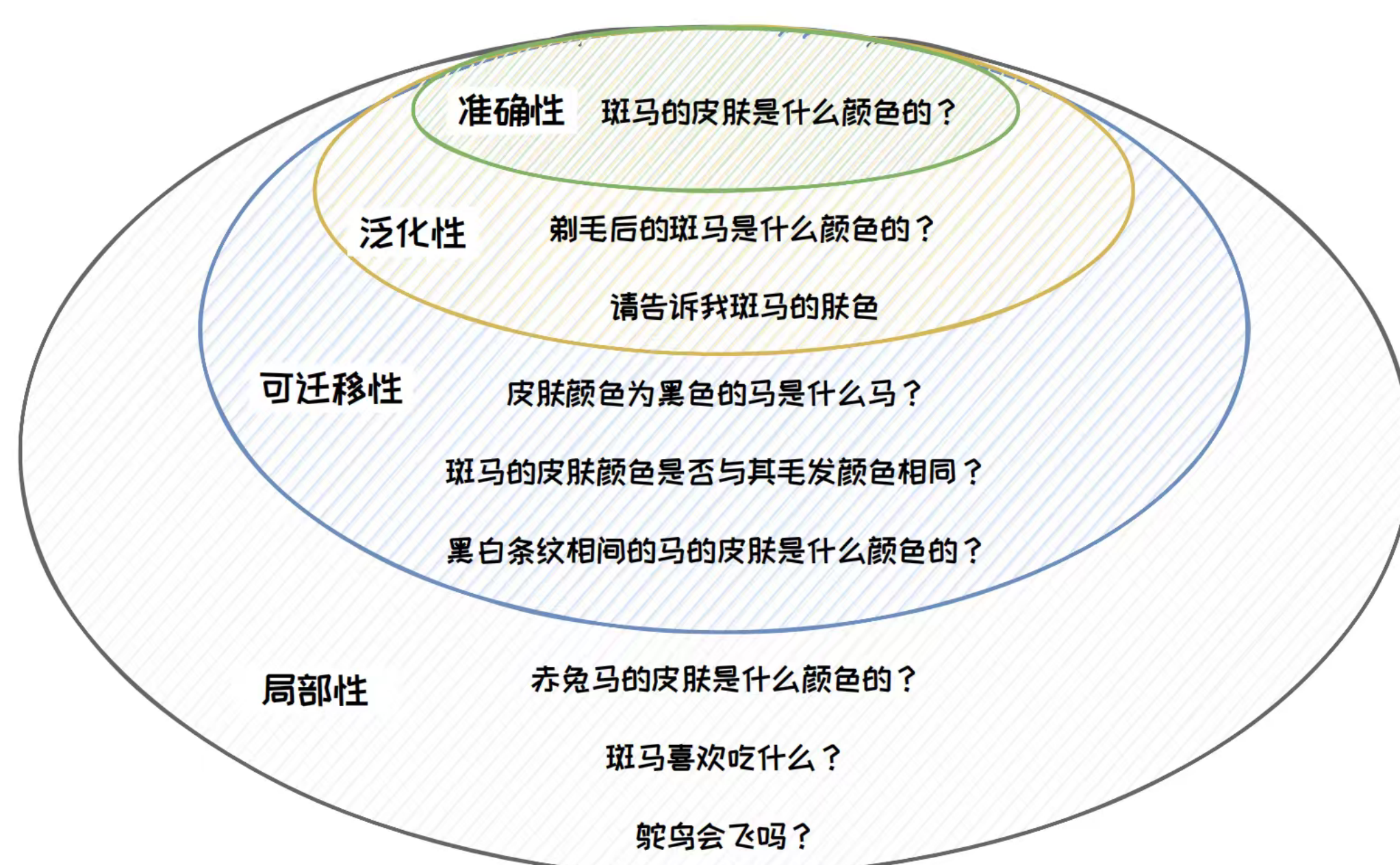
① 模型编辑

大模型有时会生成 偏见 (刻板印象、社会偏见)
毒性 (内容有害)
知识错误 (信息与事实不符)

⇒ 重新预训练, 计算资源耗费太高

微调, 易过拟合、灾难性遗忘

⇒ 模型编辑: 准确、泛化、可迁移、局部、高效



准确性: 数据集 (x_k, y_k) , 编辑前 M , 编辑后 M^*

$$\text{Acc} = I(M^*(x_k) = y_k)$$

其中 $I(\dots)$ 为指示函数

泛化性: 泛化性数据集 $D_G = \{(x_i, y_k)\}_{i=1}^{|D_G|}$, 其中 x_i 是与 x_k 语义相似的问题

$$\text{Gen} = \frac{1}{|D_G|} \sum_{i=1}^{|D_G|} I(M^*(x_i) = y_k)$$

Gen 越接近 1, 泛化性越好, 过拟合越少

可迁移性: 可迁移数据集 $D_p = \{(x_i, y_i)\}_{i=1}^{|D_p|}$, x_i 是与 x_k 相关

但答案不同的问题， y_i 为 x_i 回答， $y_i \neq y_k$

$$\text{Port} = \frac{1}{|D_P|} \sum_{i=1}^{|D_P|} I(M^*(x_i) = y_i)$$

局部性：局部性数据 $D_L = \{(x_i, M(x_i))\}_{i=1}^{|D_L|}$ ， x_i 与 x_k 无关

$$\text{Loc} = \frac{1}{|D_L|} \sum_{i=1}^{|D_L|} I(M^*(x_i) = M(x_i))$$

$\text{Loc} \rightarrow 1$ ，局部性越好，模型编辑副作用越小

常用数据集

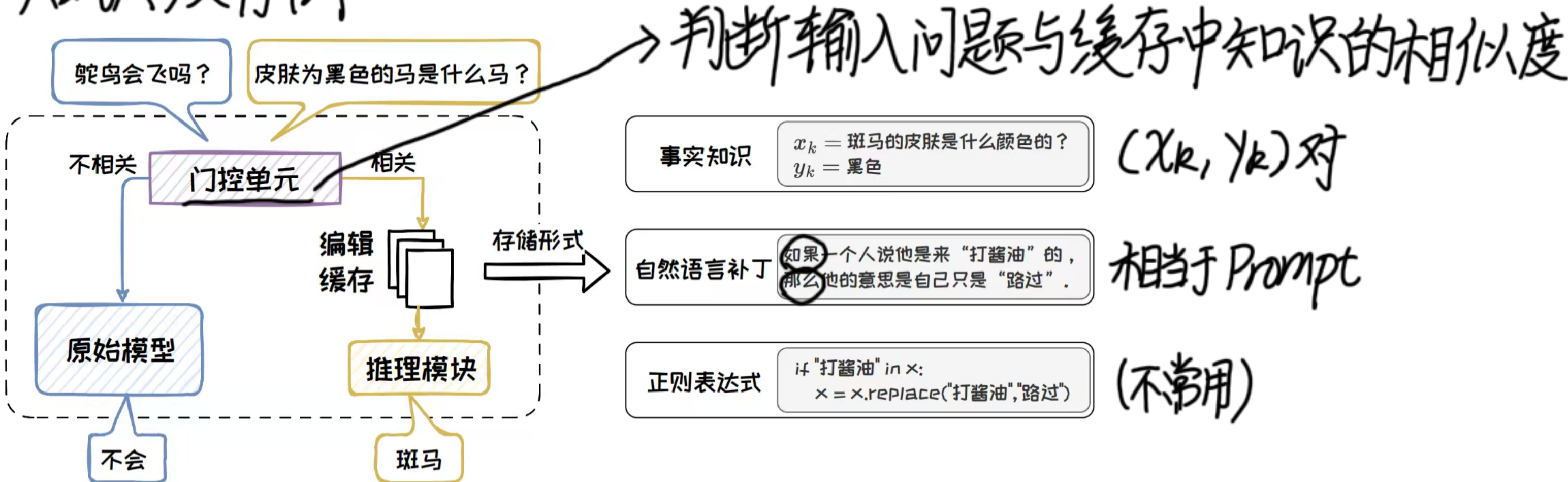
数据集	类型	训练集数量	测试集数量	输入	输出
<u>zsRE</u> [14]	知识关联	244,173	244,173	事实陈述	对象
<u>COUNTERFACT</u> [17]	知识关联	N/A	21,919	事实问题	对象
<u>WikiGen</u> [19]	文本生成	N/A	68,000	Wiki 段落	续写
<u>T-REx-100/-1000</u> [6]	知识关联	N/A	100/1,000	事实陈述	对象
<u>ParaRel</u> [4]	知识关联	N/A	253,448	事实问题	对象
<u>NQ-SituatedQA</u> [5]	问答	N/A	67,000	用户查询	答案
<u>MQuAKE-CF/-T</u> [30]	知识关联	N/A	9,218/1,825	多跳问题	对象
<u>Hallucination</u> [10]	幻觉	N/A	1,392	传记	传记
<u>MMEdit-E-VQA</u> [3]	多模态	6,346	2,093	图像问题	答案
<u>MMEdit-E-IC</u> [3]	多模态	2,849	1,000	图像描述	描述
<u>ECBD</u> [23]	知识关联	N/A	1,000	实体完成	引用实体
<u>FEVER</u> [2]	事实检查	104,966	10,444	事实描述	二进制标签
<u>ConvSent</u> [20]	情感分析	287,802	15,989	主题意见	情感
<u>Bias in Bio</u> [11]	人物传记	5,000	5,000	传记句子	职业
<u>VitaminC-FC</u> [20]	事实检查	370,653	55,197	事实描述	二进制标签
<u>SCOTUS</u> [10]	分类	7,400	931	法庭文件	争议主题

→ 准确性
→ 泛化性、局部性

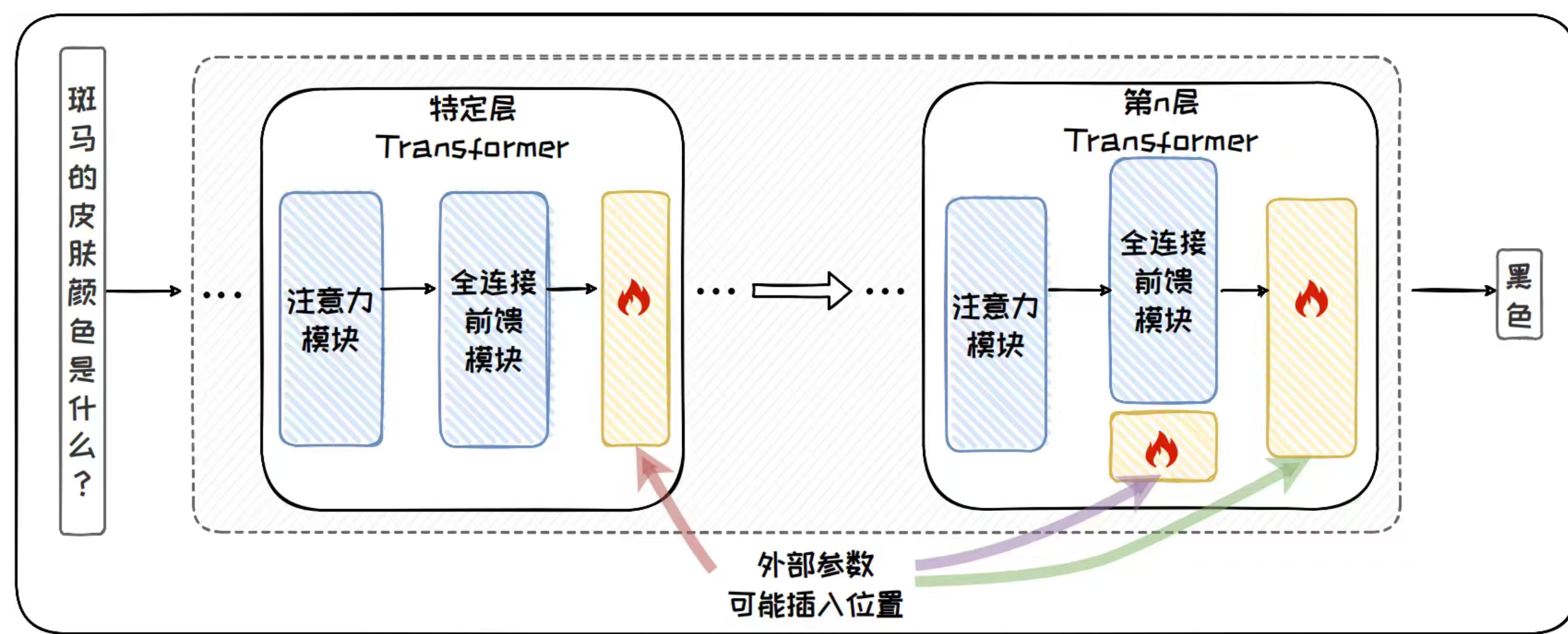
编辑方法

外部拓展法：新知识存于外部参数、外部知识库

(1) 知识缓存法



(2) 附加参数法



CALINET和T-Patcher修改最后一层transformer的全连接前馈模块

GRACE对特定层插入 codebook (适配器), 存储错误知识 Keys 和修正值 Values, 以及用于匹配输入—错误相似度的延迟半径

内部修改法: 直接修改模型参数

(1) 元学习法

元训练是 Bilevel-Optimization, 内层为不同编辑任务上的优化, 外层是元知识 ω 在验证集的编辑任务中的优化

$$\omega^* = \arg \min_{\omega} \sum_{i=1}^n L_{\text{meta}}(\theta^{*(i)}(\omega), \omega, \underbrace{D_k^{\text{val}(i)}}_{\text{验证集}})$$

$$\text{s.t. } \theta^{*(i)}(\omega) = \arg \min_{\theta} L_{\text{edit}}(\theta, \omega, \underbrace{D_k^{\text{train}(i)}}_{\text{训练集}}).$$

最小化模型 $\theta^{*(i)}$ 在 $D_k^{\text{train}(i)}$ 上的预测误差

ENN将元知识视为优化器参数, 更新优化器参数使编辑中的模型参数更新更有效 $\Rightarrow$ 仅用于小型网络, 成本高

KE将元知识视为超网络, 训练超网络, 使其(根据输入问题)来生成模型参数更新值

MEND 低秩分解来优化超网络辅助模型更新, 超网络不直接输入/输出梯度, 而是其低秩分解

(2) 定位编辑法

Transformer中的全连接前馈模块(FFN),

输入 $\rightarrow$ MLP(升维) $\rightarrow$ 激活 $\rightarrow$ MLP(降维) $\rightarrow$ 输出

$\Downarrow$ query $\Downarrow$ key $\Downarrow$ value

⇒ FFN模块可视为键值存储体，对知识进行存储

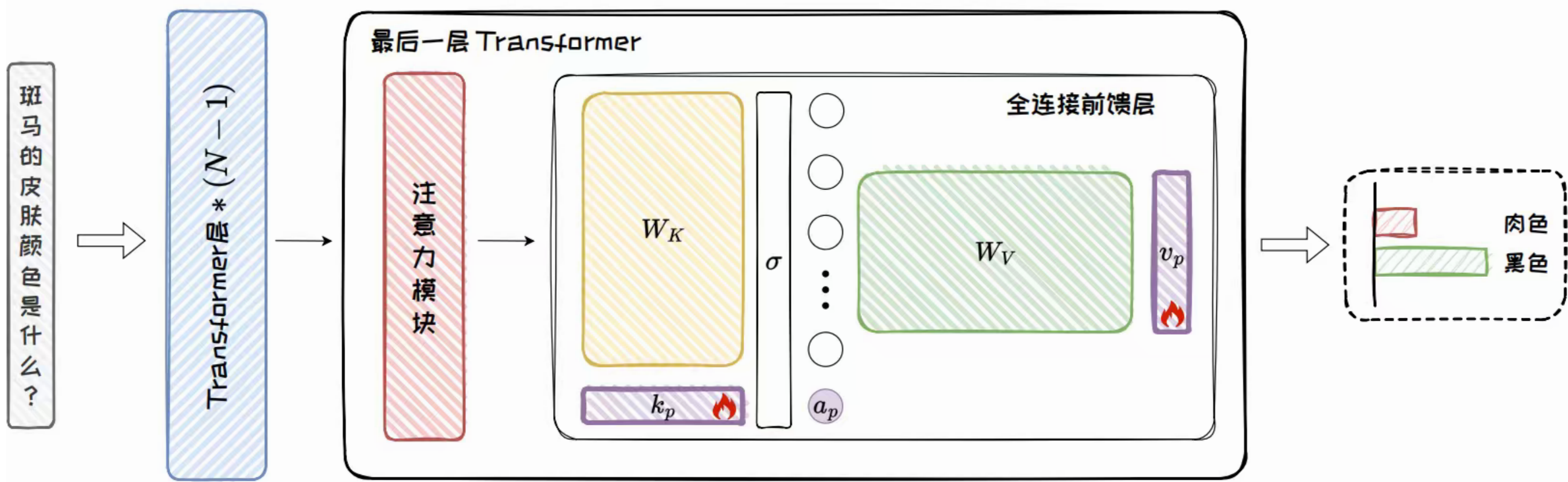
KN: FFN中间激活值定义为知识神经元
通过梯度变化，确定各神经元对某知识的贡献
仅修改特定的知识神经元

ROME: 更新整个FFN模块

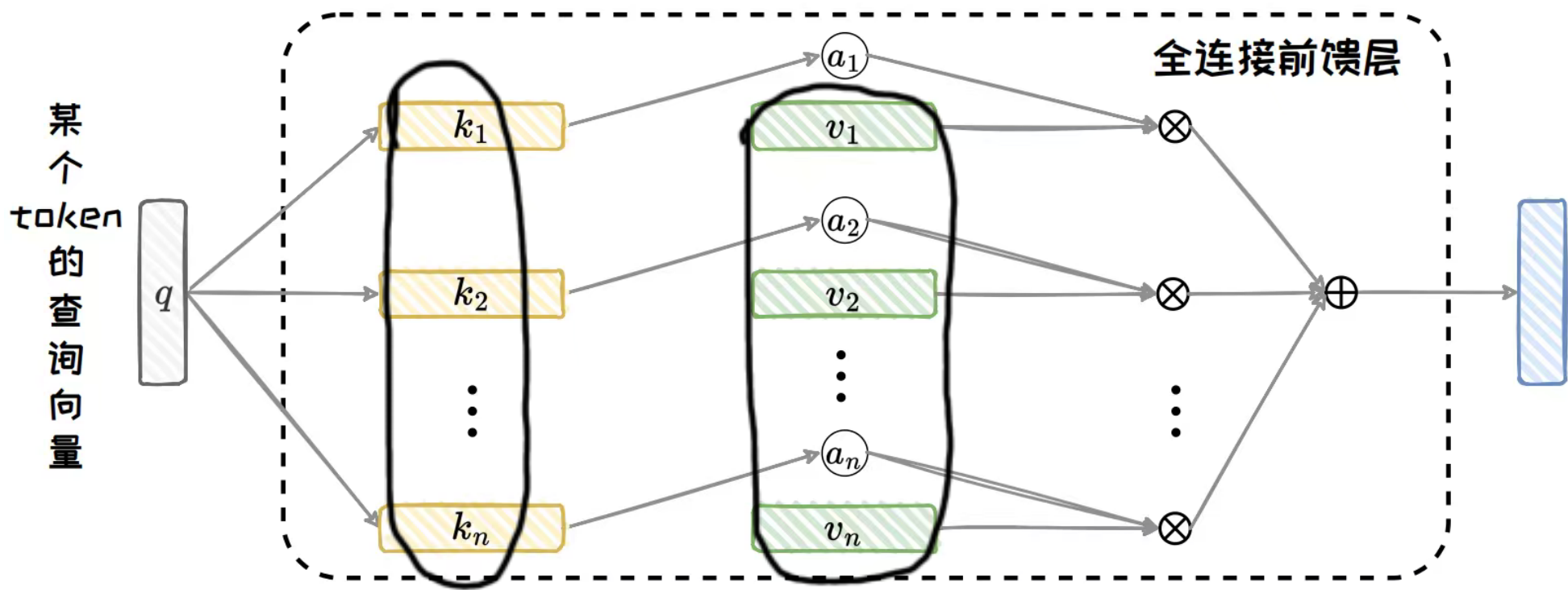
		方法	准确性	泛化性	可迁移性	局部性	高效性
外部拓展法	知识缓存法	SERAC	高	高	低	高	✓
	附加参数法	CaliNET	低	低	-	中	✓
		T-Patcher	高	高	高	中	✗
内部修改法	元学习法	KE	低	低	-	高	✓
		MEND	中	高	中	高	✓
	定位编辑法	KN	中	低	-	中	✗
		ROME	高	高	高	高	✗
		MEMIT	高	高	高	高	✓

② T-Patcher

附加参数：最后一个Transformer的FFN模块中



类似于向键值存储体中增加记忆单元



键 值 $W_{proj} = [v_1, v_2, \dots, v_n]$, b_v
 $W_{fc} = [k_1, k_2, \dots, k_n]$, b_k (偏置) $\Downarrow$
 $\Downarrow$ 模型输出概率分布
输入文本的特定模式

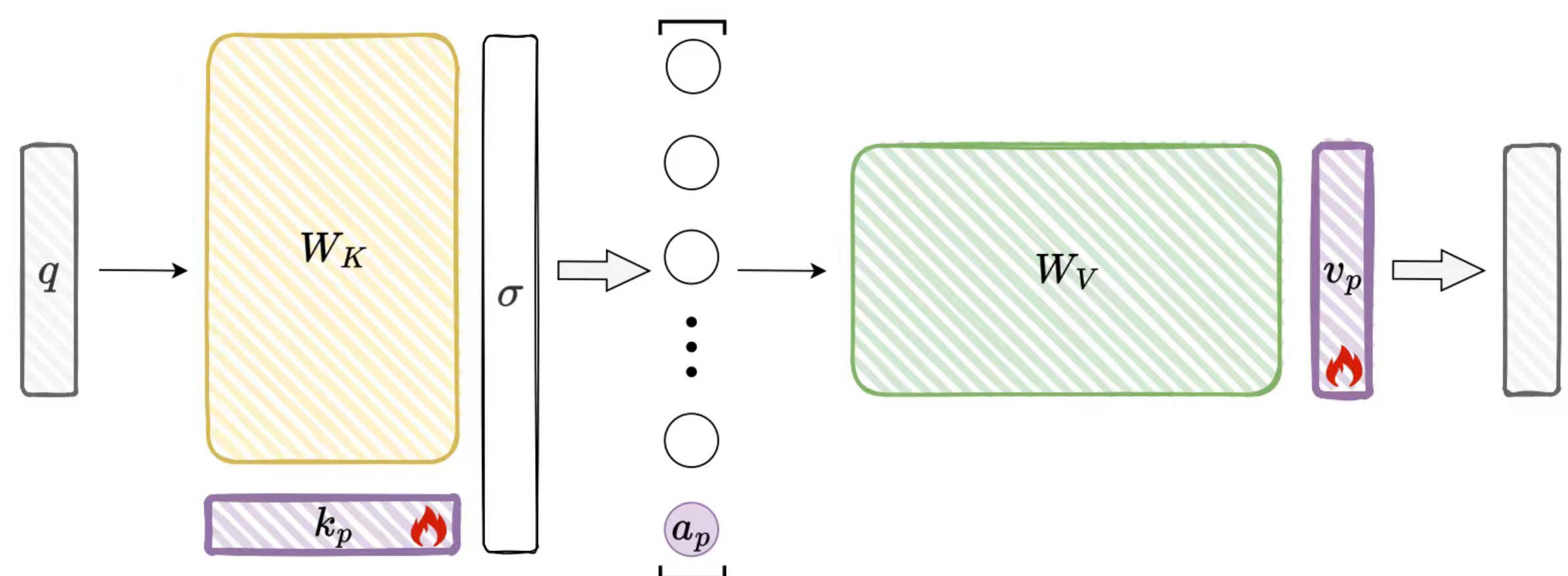
查询存储体时,

$$a = \sigma(q \cdot W_{fc} + b_k)$$

$$FFN(q) = \underbrace{a \cdot W_{proj}}_{\text{权重}} + b_v \rightarrow W_{proj} \text{中所有值加权求和}$$

⇒ FFN 隐藏层维度可视为记忆的文本模式的数量
添加补丁, 即增大维度, 添加知识

添加补丁: 键 k_p , 值 v_p , 偏置 b_p



添加后

$$[a \quad a_p] = \sigma(q \cdot [W_{fc} \quad k_p] + [b_k \quad b_p])$$

$$FFN_p(q) = [a \quad a_p] \cdot [W_{proj} \quad v_p]^T + b_v = FFN(q) + a_p \cdot v_p,$$

训练补丁参数

为每个待编辑的 token 均添加一个 patch

损失函数:

准确性 L_{acc}

激活损失 l_a : 补丁在目标输入下能被激活

编辑损失 l_e : 激活后, 能准确输出

$$L_{Acc} = l_a(k_p, b_p) + \underbrace{\alpha l_e(k_p, v_p, b_p)}_{\text{权重}}$$

$$l_a(k_p, b_p) = \exp(-\underbrace{q_e}_{\text{query}} \cdot k_p - b_p)$$

$$l_e(k_p, v_p, b_p) = CE(\underbrace{y_e}_{\text{模型预测输出}}, \underbrace{p_e}_{\text{理想输出 token}})$$

局部性 L_m

随机保留已处理 query 且与当前无关的, 组成记忆数据集 $D_M = \{q_i\}_{i=1}^{|D_M|}$

记忆损失包含 L_{m1} 和 L_{m2}

↓ 放大补丁对目标 q_e 和无关 q_i 的

不会对无关输入激活补正

激活值差异

⇒

$$L_m = l_{m1}(\mathbf{k}_p, b_p) + l_{m2}(\mathbf{k}_p, b_p, \mathbf{q}_e)$$

$$l_{m1}(\mathbf{k}_p, b_p) = \frac{1}{|D_M|} \sum_{i=1}^{|D_M|} (\mathbf{q}_i \cdot \mathbf{k}_p + b_p - \beta)$$

激活阈值

$$l_{m2}(\mathbf{k}_p, b_p) = \frac{1}{|D_M|} \sum_{i=1}^{|D_M|} ((\mathbf{q}_i - \mathbf{q}_e) \cdot \mathbf{k}_p + b_p - \gamma),$$

激活值差异的阈值

⇒ 损失函数 L_p

$$L_p = L_{Acc} + \beta \cdot L_m = l_e + \alpha \cdot l_a + \beta \cdot (l_{m1} + l_{m2}),$$

局限性：不同架构上性能波动
批量编辑时消耗内存高

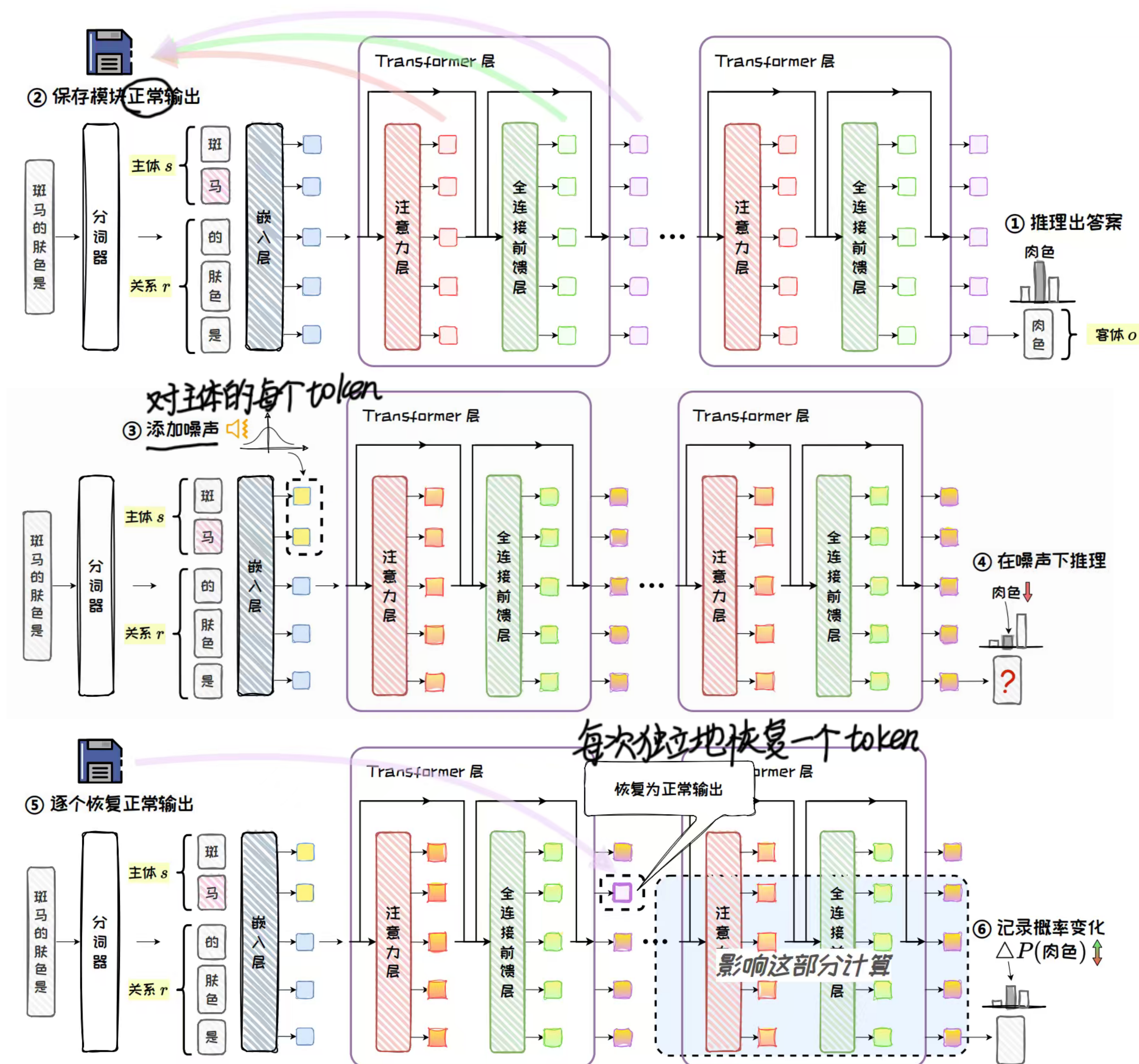
③ ROME (Rank-One Model Editing)

因果跟踪实验

针对知识元组 $t = (s, r, o)$, s, r, o 分别为主体、关系、客体

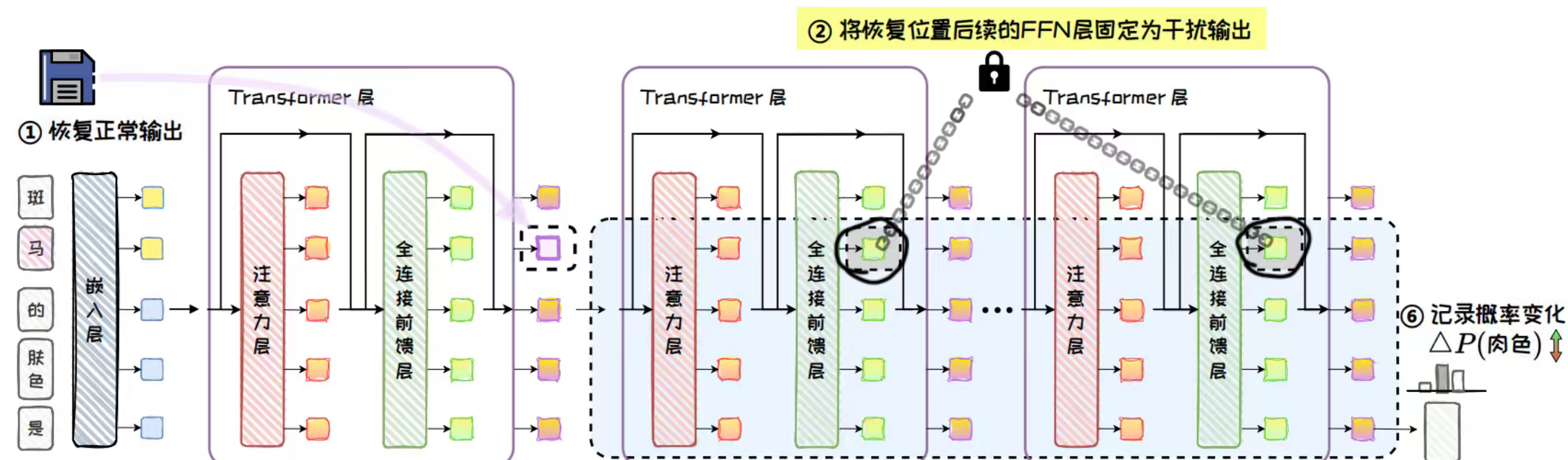
输入 $q = (s, r)$, 期望输出 o

正常推理 → 干扰推理 → 恢复推理



⇒ 中间层 Transformer 处理 s 最后一个 token ($s^{(-1)}$) 时的因果效应主要来自于 FFN, 因此其可能是模型存储知识的关键

阻断实验



⇒ 阻断 FFN, 中间层处理 $s^{(-1)}$ 失去因果效应, 末尾层无影响
阻断注意力层, 各层因果效应仅小幅下降

实验结论

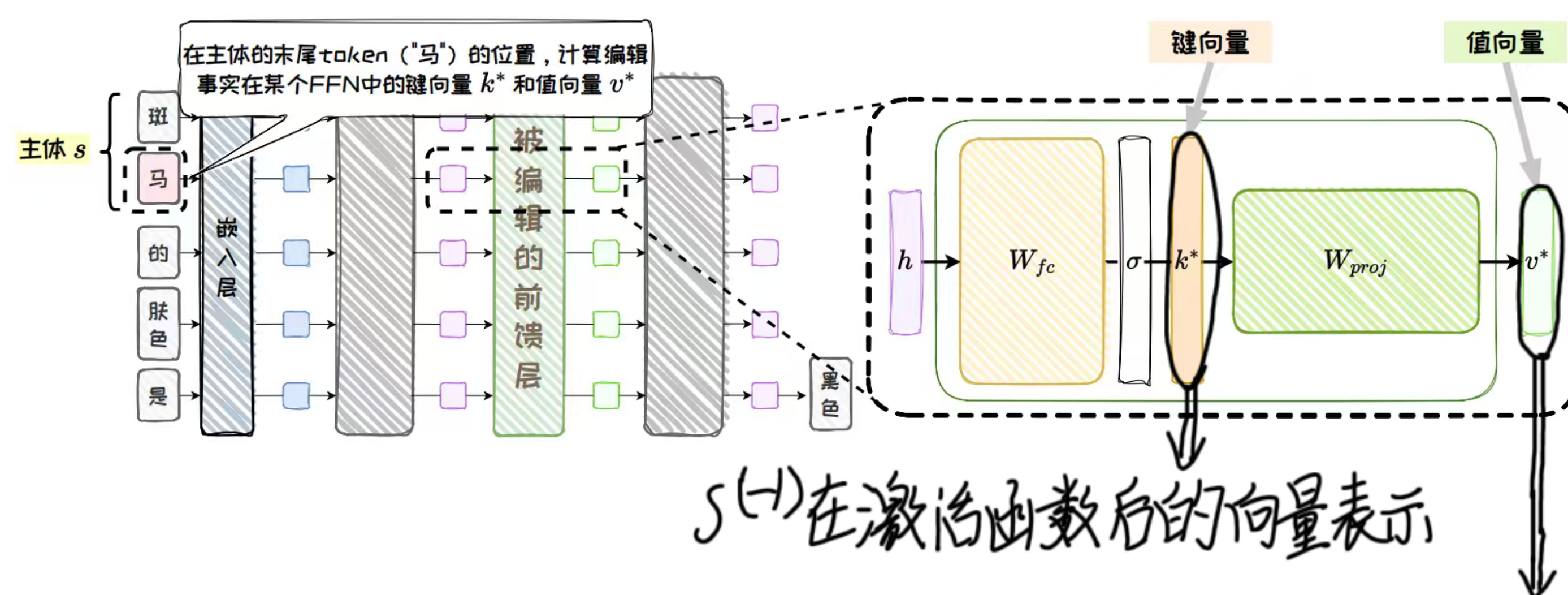
知识存于中间层, 其关键参数位于 FFN
特定中间层的 FFN 在处理主体的末尾 Token 时发挥作用

编辑方法

将 FFN 视为键值存储体, 下投影矩阵的输入向量 → Key
输出向量 → Value

上投影矩阵 W_{fc} 和激活函数 σ ⇒ 键 k^*

下投影矩阵 W_{proj} 和键向量 k^* ⇒ 值 v^*



优化 v^* (可训练参数)

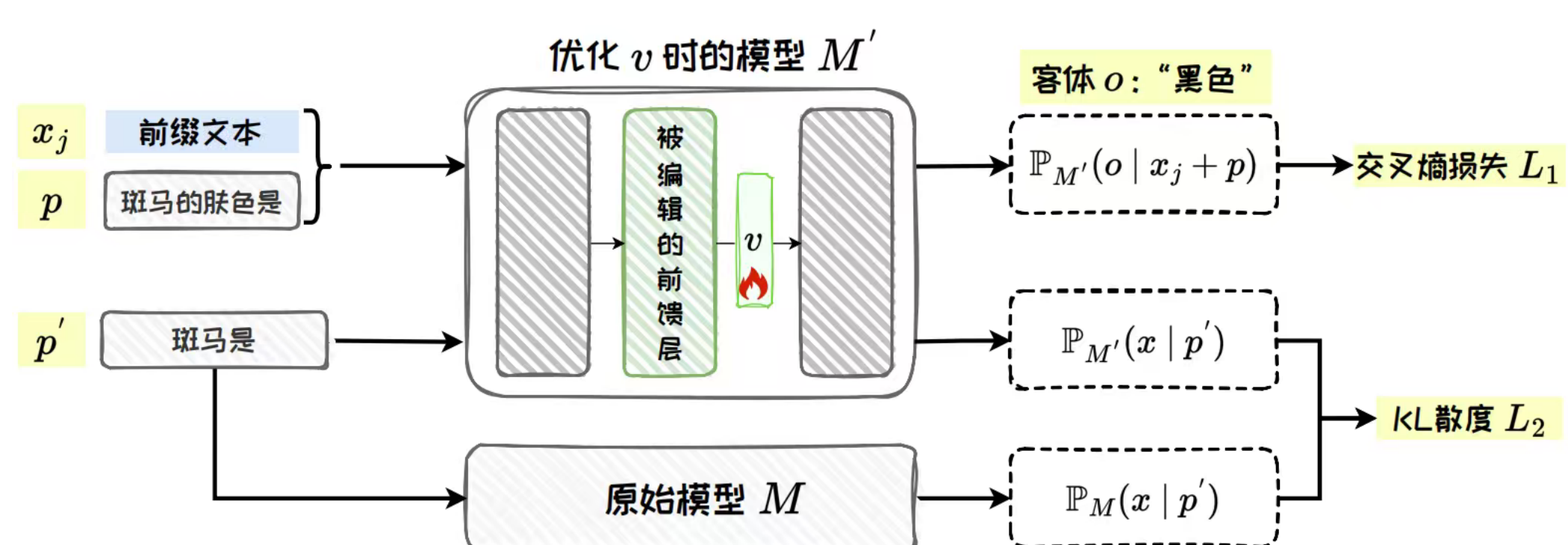
损失函数 $\mathcal{L}(v) = \mathcal{L}_1(v) + \mathcal{L}_2(v)$

$$\mathcal{L}_1(v) = \frac{1}{N} \sum_{j=1}^N -\log \mathbb{P}_{M'}(o | x_j + p)$$

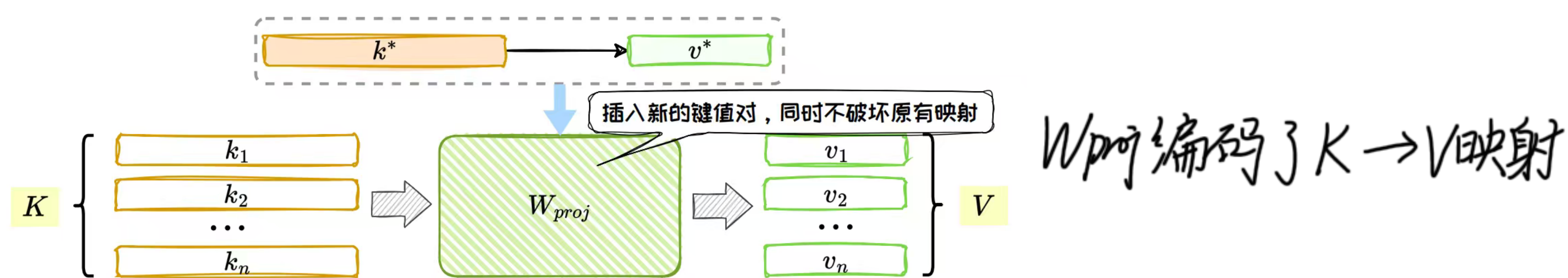
编辑目标问题是 prompt
从 s 含义的 prompt

$$\mathcal{L}_2(v) = D_{KL}(\mathbb{P}_{M'}(x | p') || \mathbb{P}_M(x | p'))$$

KL 散度



调整下投影矩阵 W_{proj} , 使得 $W_{proj} k^* = v^*$, 从而插入新知识



$\Rightarrow$ 带约束的最小二乘问题

$$\begin{cases} \min ||\hat{W}K - V|| \\ \text{s.t. } \hat{W}k^* = v^* \end{cases} \Rightarrow \text{闭式解: } \begin{cases} \hat{W} = W + \Lambda(C^{-1}k^*)^T, \\ \Lambda = (v^* - Wk^*)/(C^{-1}k^*)^T k^* \end{cases}$$

其中 $C = KK^T$ 为常数

④ 应用

被遗忘权 RTBF

如 DPEN, 检测并编辑隐私神经元

去除毒性

如识别并增加积极神经元的权重 $\Rightarrow$ 处理敏感术语时受限

减弱偏见

LSDM 采用类似 ROME 方法减少性别偏见

第六章 检索增强生成

① Retrieval-Augmented Generation (RAG)

幻觉现象

训练数据

知识过时、知识边界、知识偏差

模型自身

知识长尾 (训练数据中部分信息出现频率低)

曝光偏差 (训练与推理时的任务存在差异)

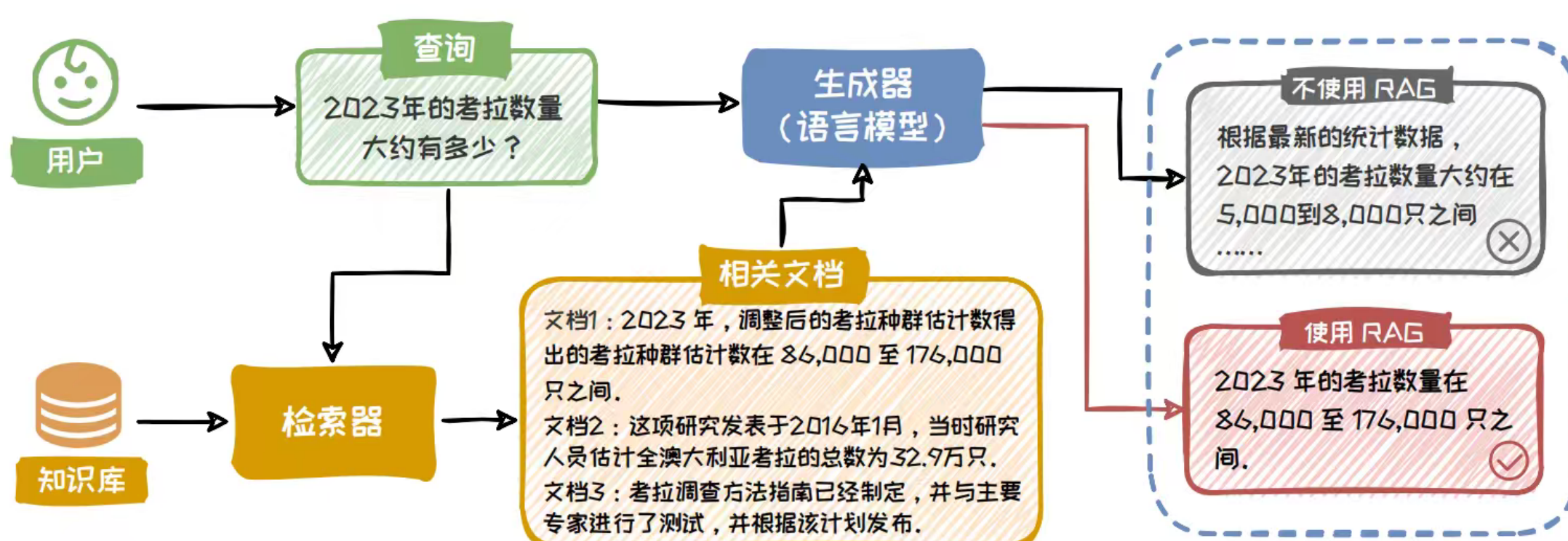
对齐不当 (人类偏好对齐的标注不当, 引入不良偏好)

解码偏差 (解码策略中的随机因素)

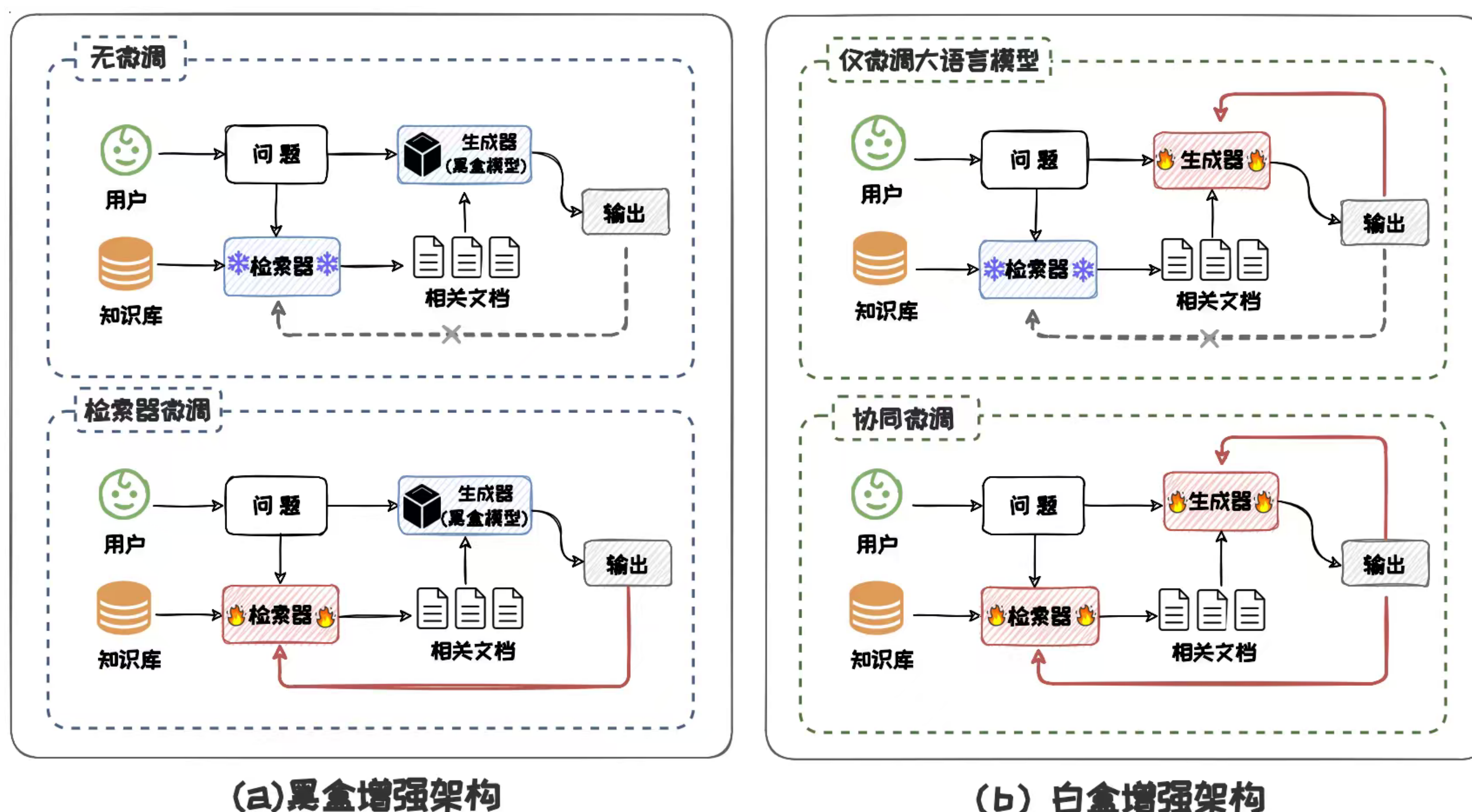
⇒ RAG: 无需更新模型, 避免 Catastrophic Forgetting, 解决幻觉

RAG的组成

Corpus
Retriever
Generator



② RAG架构



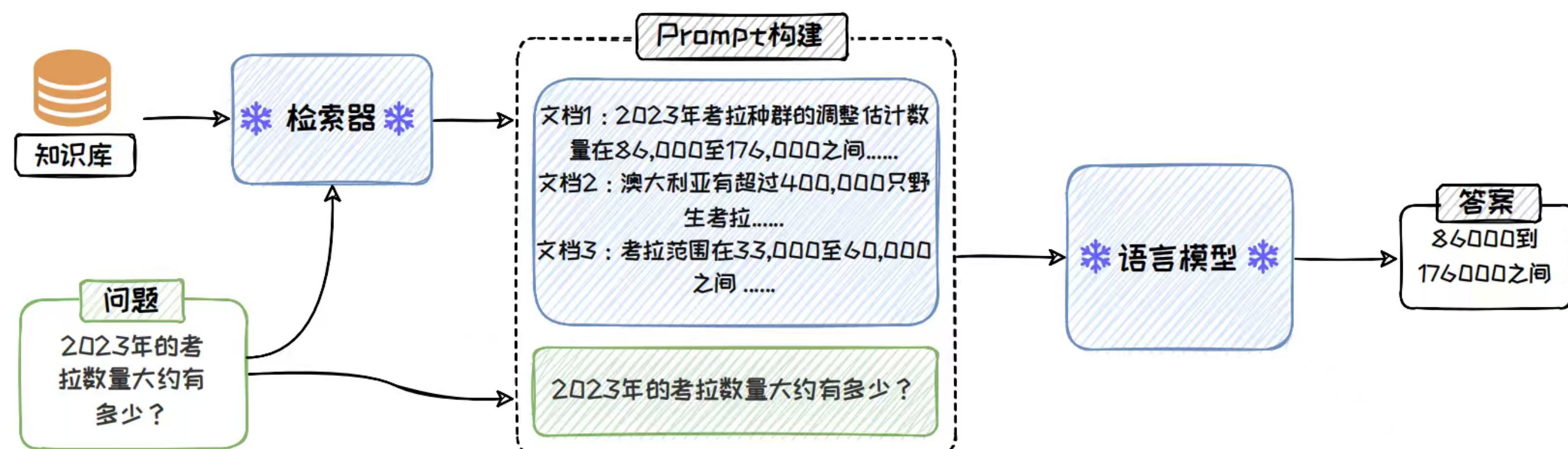
(1) 黑盒 — 无微调

In-Context RALM 将检索到的文档放于 Prompt 的问题之前
一个 RAG 任务可能涉及多次检索和生成

关键参数：检索步长、检索查询长度

↓
响应速度、信息及时性

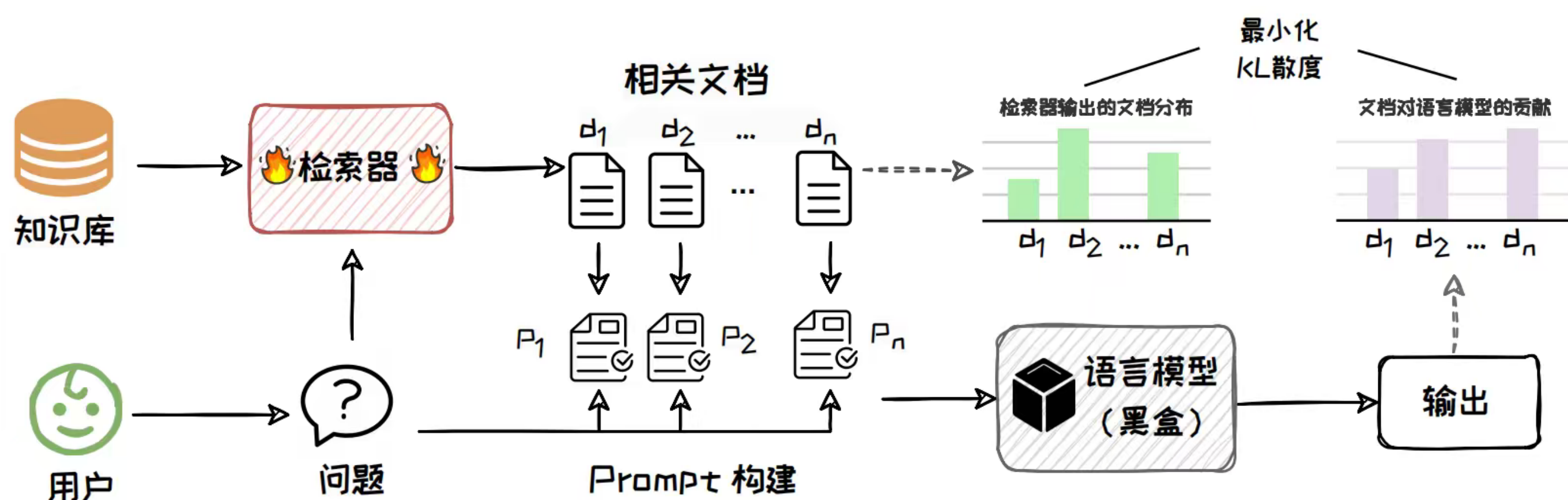
↓
检索信息与当前任务相关程度



(2) 黑盒 — 检索器微调

REPLUG LSR 以 PPL 为监督信号微调检索器，使用 KL 散度损失对齐检索文档的相关性分布与其对模型性能提升的贡献分布

上下文与文档间的相似度 文档+上文的模型预测结果

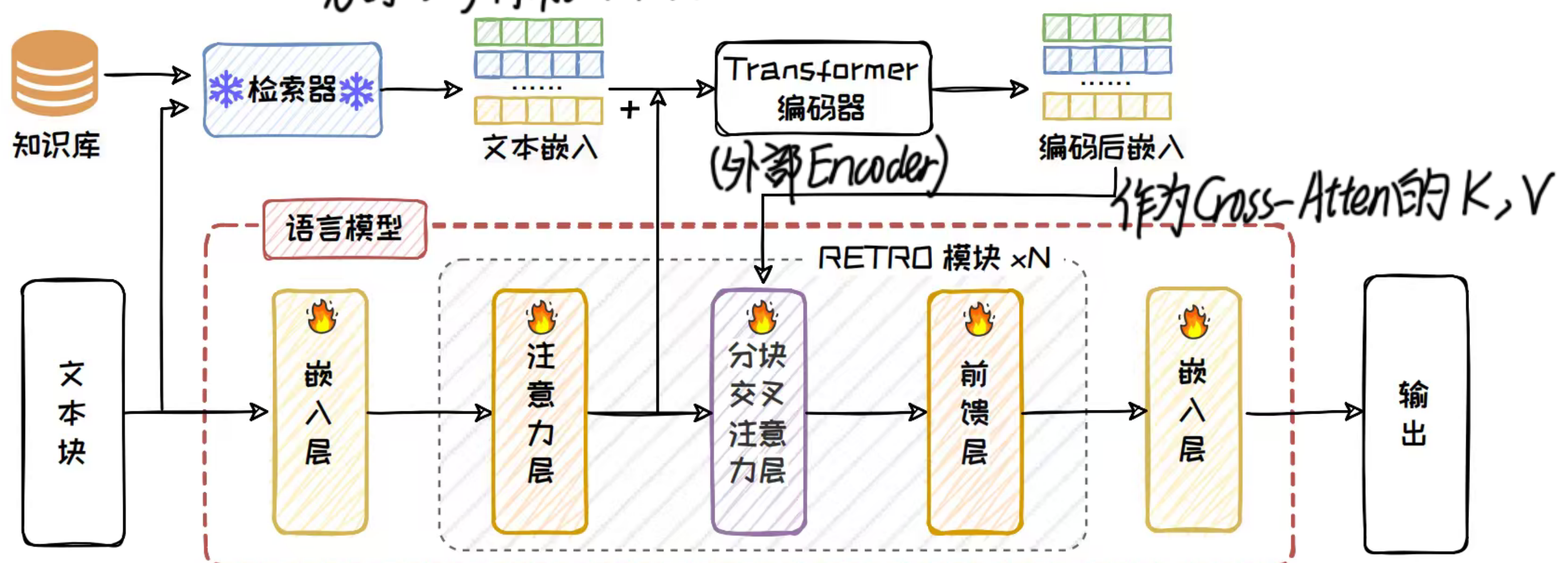


ARR 引入小语言模型的交叉注意力得分来标注文档 $\Rightarrow$ 微调检索器
 $\Rightarrow$ 增强跨任务表现

(3) 白盒 — 微调 LLM

RETRO

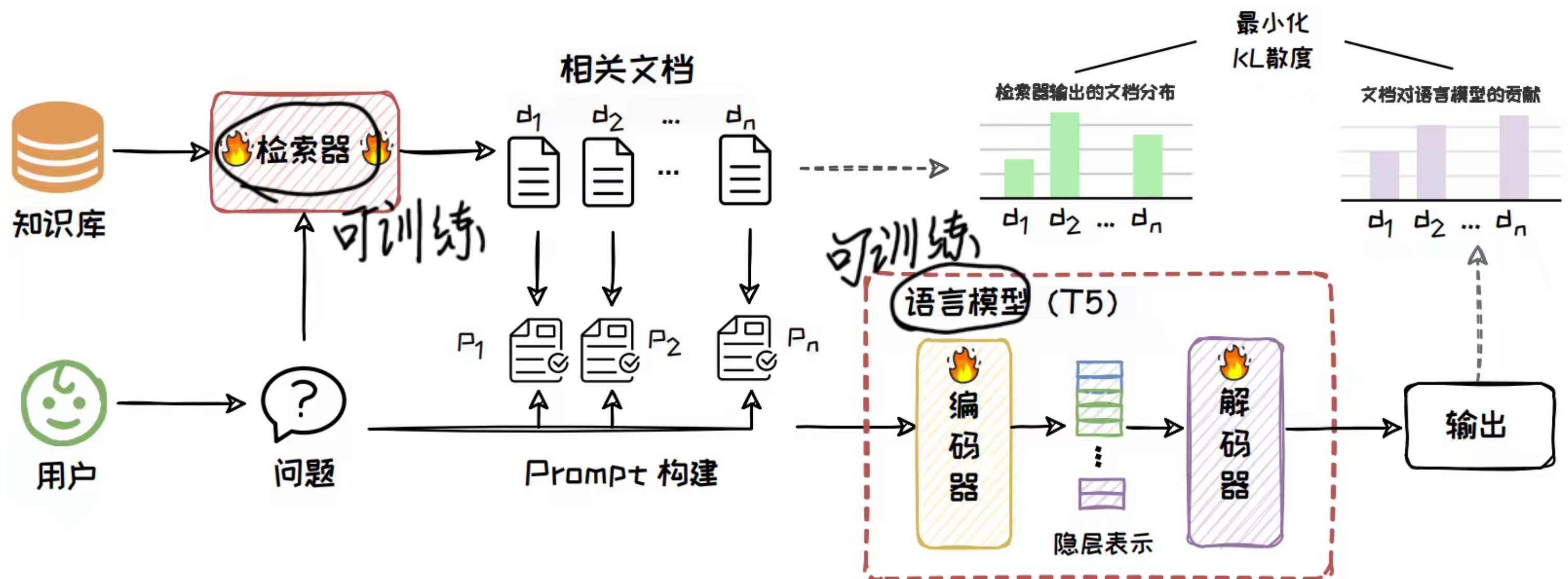
先切块，再用 BERT 嵌入



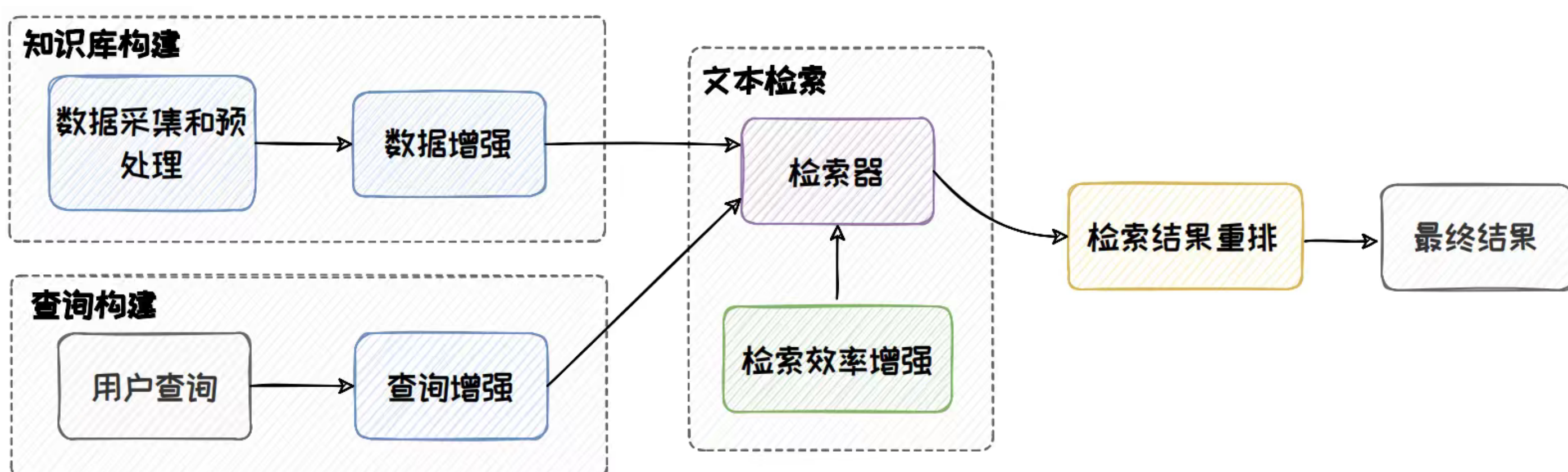
自回归生成时,每生成一段文本,就检索最相似的嵌入向量
将嵌入向量与Self-Attention层输出 $\Rightarrow$ 外部 Encoder 编码 $\Rightarrow$
作为 Cross-Attention 的 K, V (结合自身与外部信息)

(4) 白盒 —— 检索器、模型协同微调

Atlas 用 KL 散度联合训练两者



③ 知识检索



(1) 知识库构建

数据采集和预处理

文本元信息 $\Rightarrow$ 预处理 $\left\{ \begin{array}{l} \text{数据清洗 (特殊字符、异常编码、无用HTML)} \\ \text{文本分块 (适应上下文窗口长度、减少长文本不相关内容)} \\ \downarrow \\ \text{先拆为句子,再组合为一定大小的块,相邻块会 overlap 以保持语义连贯} \end{array} \right.$

知识库增强

查询生成: 生成与文档内容相关的伪查询,作为“键”用于检索时匹配

标题生成: 为无标题文段生成标题

(2) 查询增强

查询语义增强

同义改写: 将原始查询改写为不同表达形式

多视角分解: 将复杂查询分解为不同视角的子查询

查询内容增强

生成背景文档: 原始根据查询生成背景, 补充查询的上下文

(3) 检索器

判别式检索器

稀疏检索器

TF-IDF

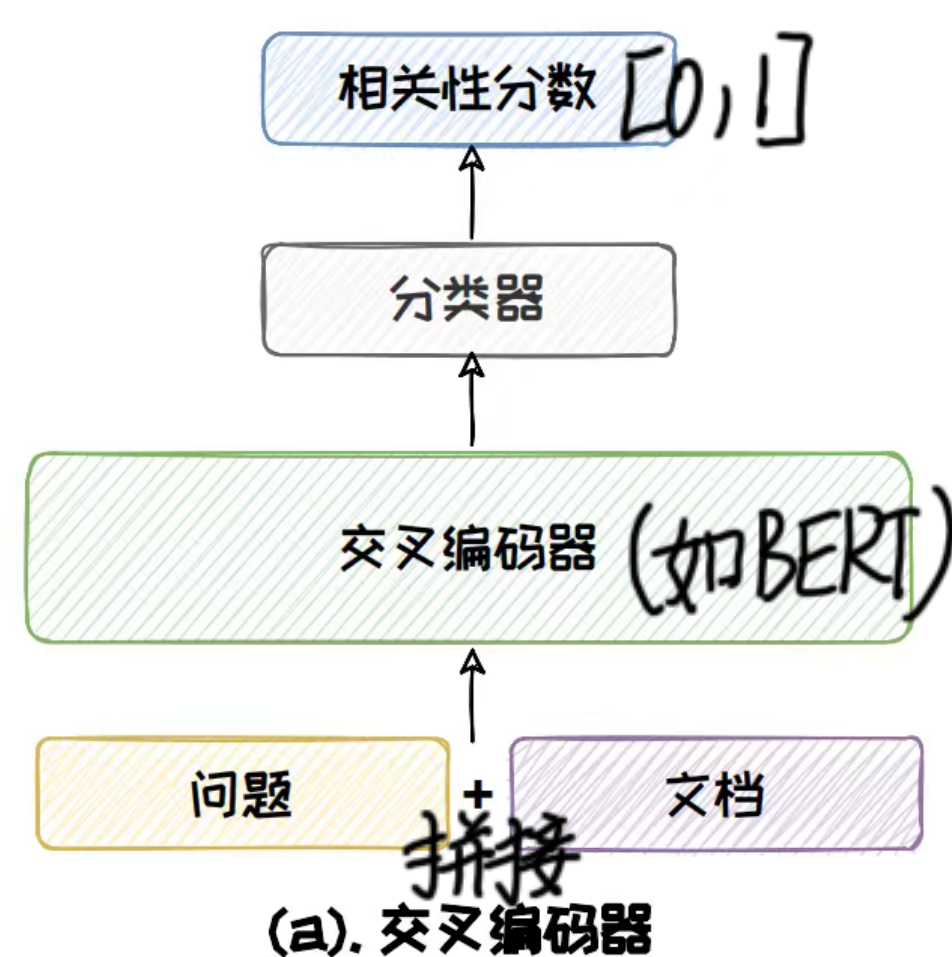
词频 TF $tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}$, $n_{i,j}$ 为词 t_i 在文档 d_j 出现的次数

逆文档频率 IDF $idf_i = \log \frac{|D| \text{ (总文档数)}}{|\{j : t_i \in d_j\}| \text{ (含 } t_i \text{ 的文档数)}}$

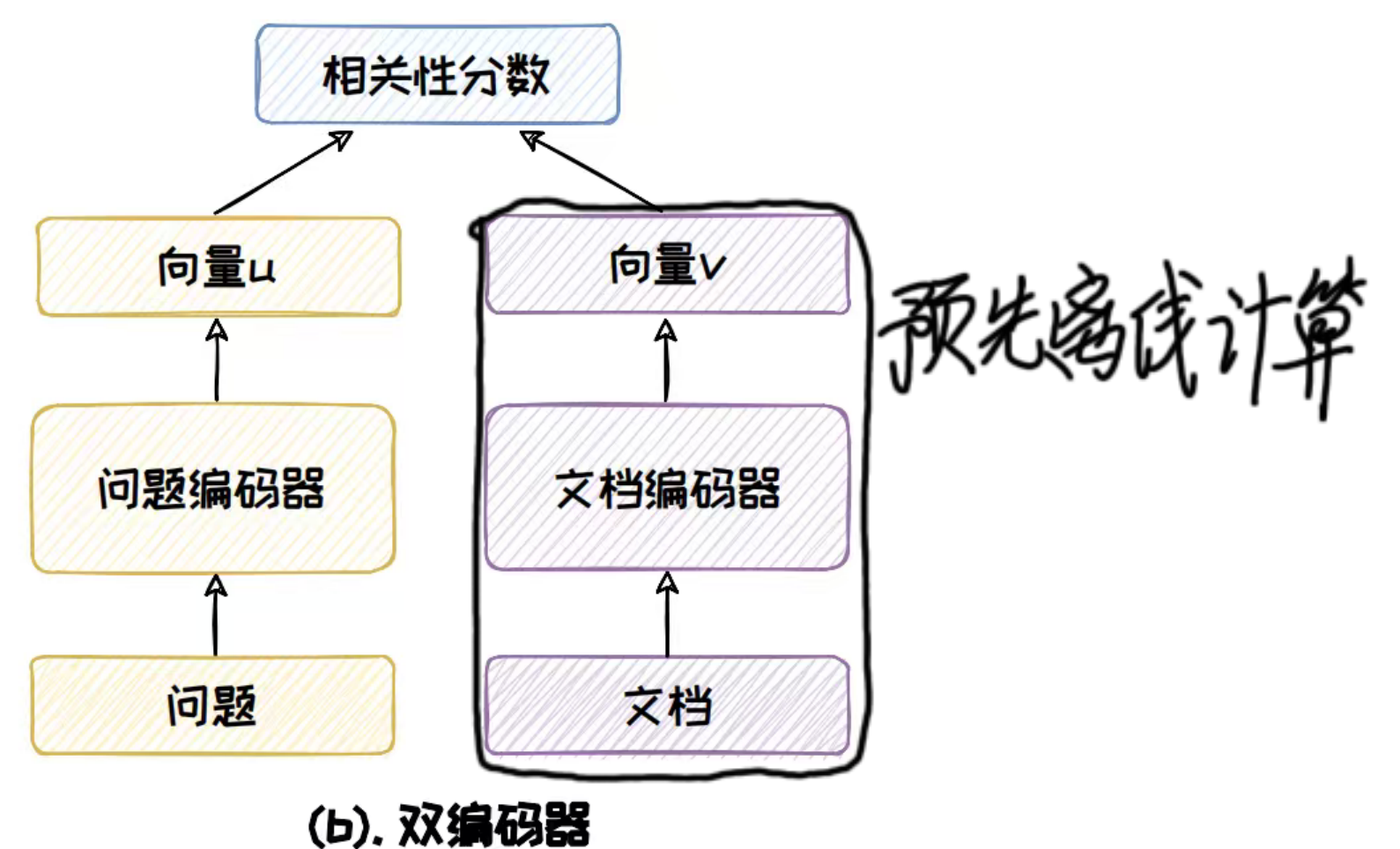
TF-IDF 值 $tfidf_{i,j} = tf_{i,j} \times idf_i$

高词频、低文档频率 $\Rightarrow$ 高权重
过滤常见词语, 保留重要词语

稠密检索器



Cross-Encoder
计算量大、更精确



Bi-Encoder
效率高、查询和文档缺乏交互

$\Downarrow$
ColBERT 根据两者 Token 相似度, 以对比学习微调两 Encoders

生成式检索器

将知识库文档记忆在模型参数中, 直接生成相关文段的标识符 (DocID)

第一阶段: Seq-to-Seq 学习将查询映射为相关文档的 DocID

最大似然估计 MLE 优化 Encoder-Decoder 模型

第二阶段: 数据增强 (伪查询、文档片段作为查询)

排名优化 (对比损失、排名损失)

DocID 设计

数字 (大量文档 $\rightarrow$ 标识符数量激增)
词 (直接从标题、URL、N-gram 中提取)

难处理大规模文档、难以动态新增文档

(4) 检索效率增强

高效向量存储、查询 $\Rightarrow$ 向量数据库 $\xrightarrow{\text{核心}}$ 相似度索引算法

基于空间划分的方法

基于数的索引: KD 树, Ball 树

基于哈希的索引: 局部感知哈希 (LSH, 桶方法)

基于图的方法

邻近图, 向量检索即为图的遍历

稀疏 $\Rightarrow$ 计算代价低, 稠密 $\Rightarrow$ 缩短搜索路径

如 NSW, IPNSW, HNSW

基于乘积量化的方法

对文本: 划分为多个子空间, 各自聚类 $\Rightarrow$ 聚类中心为码字
所有码字集合为码本

对查询: 划分为子向量, 计算每个子向量到各自距离最近的码字的距离, 形成距离表, 得到最近邻结果

(5) 检索结果重排

基于双编码的方法

MiniLM-L、Cohere (API)

基于上下文学习的重排方法

RankGPT

RankGPT Prompt 模板

这是RankGPT，一款智能助手，专门用于根据查询的相关性对段落进行排序。

以下是{{num}}段文字，每段都有一个数字标识符[]。我将根据查询内容对它们进行排序：{{query}}

[1] {{passage_1}}

[2] {{passage_2}}

(更多段落) ...

查询是：{{query}}

我将根据查询对上述{{num}}段文字进行排序。这些段落将按照相关性降序列出，使用标识符表示，最相关的段落将列在最前面，输出格式应该是[] > [] > 等，例如，[1] > [2] > 等。

对{{num}}段文字的排序结果（仅限标识符）是：

对长文本,使用滑动窗口逐片段排序

④ 生成增强

(1) 何时增强

判断LLM是否具有内部知识

外部观测法：通过Prompt直接询问 or
反复询问同一问题观察一致性



你需要额外的信息帮助来回答这个问题吗？
问题：2023年的考拉数量大约有多少？

可能的回答

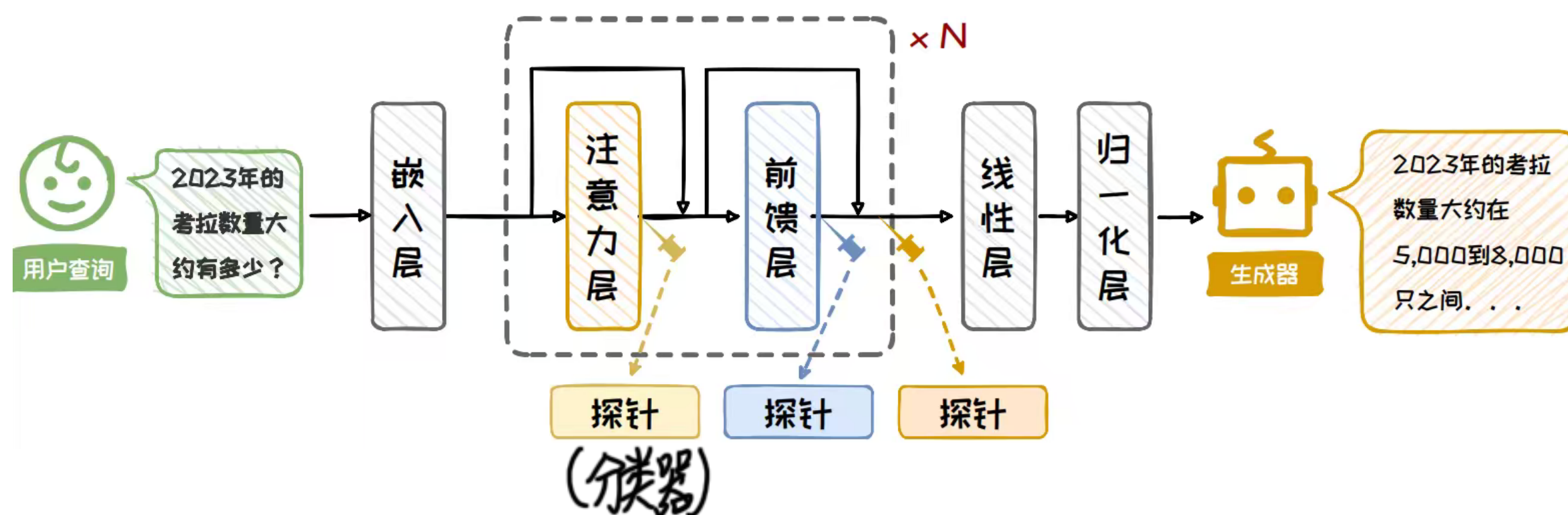
不，我不需要额外的信息帮助来回答这个问题。 / 是的，我需要额外的信息帮助来回答这个问题。



观察训练数据

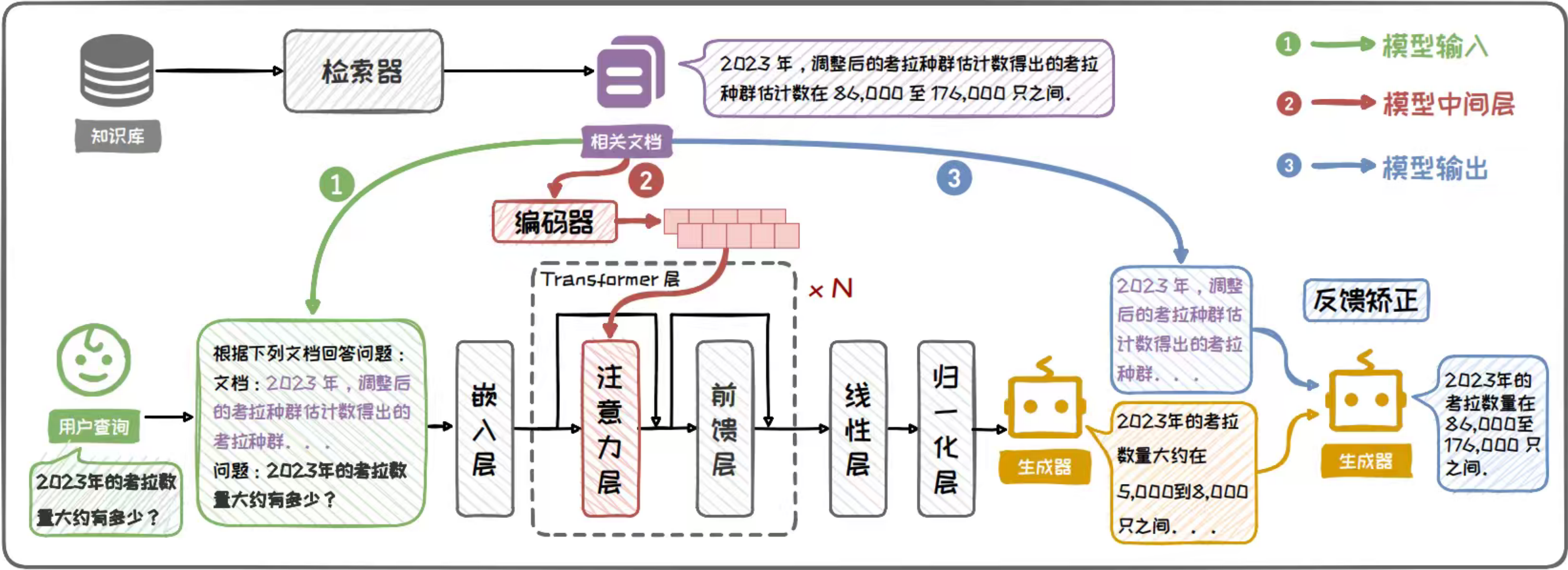
or
伪训练数据统计量 (实体的流行程度)

内部观测法：探针



中间层探针分类准确率高 $\Rightarrow$ 中间层隐藏状态反映相关知识储备

(2) 何处增强



输入端

直观、简单

文本过长时,可能超出模型最大序列长度,增加推理成本

中间层 (如 Retro)

更深入影响模型内部表示

无法用于黑盒模型

输出端 (如 REFEEED)

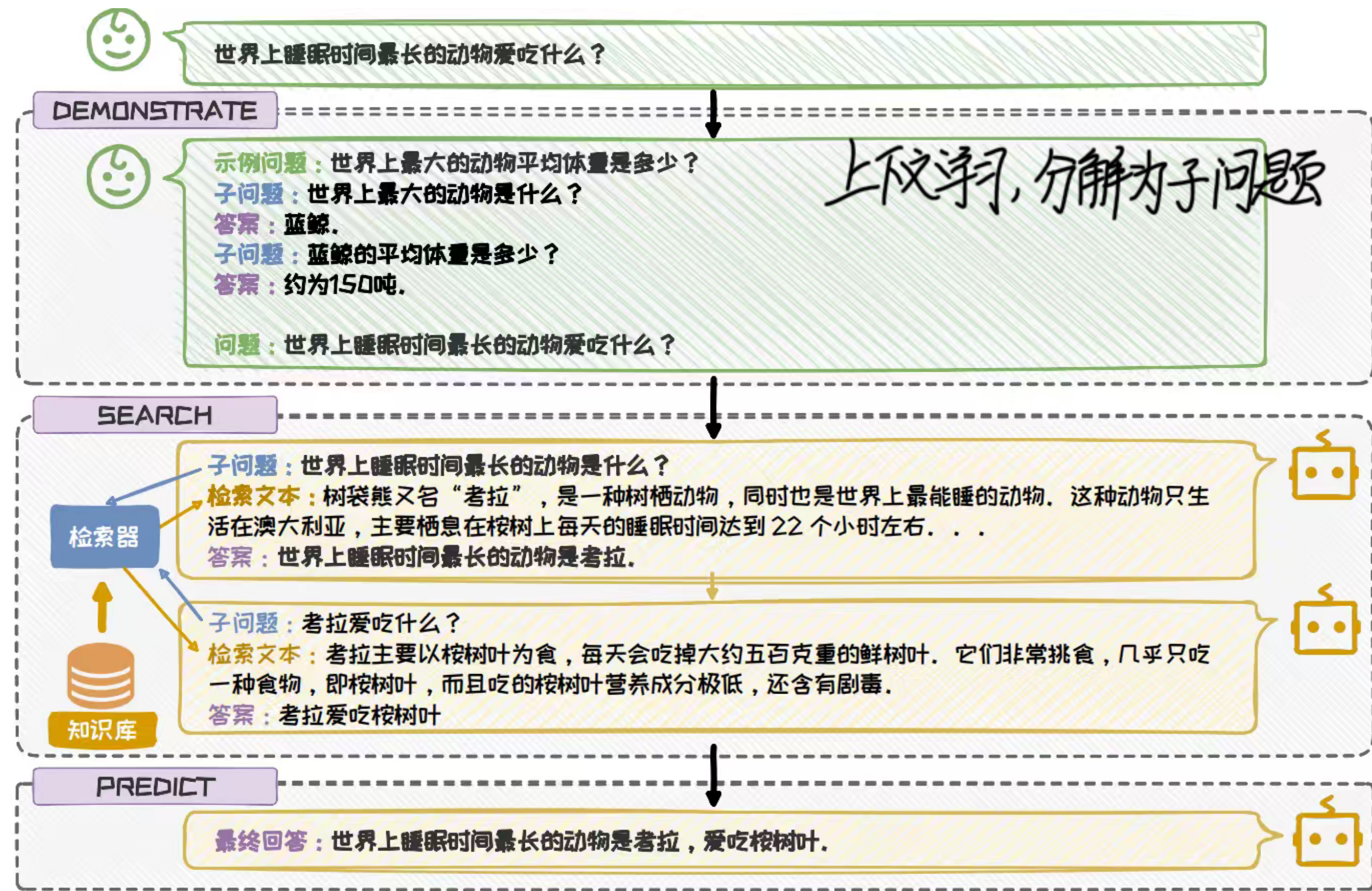
校准, 提高可靠性

依赖外部知识质量和相关性

(3) 多次增强

复杂问题 $\Rightarrow$ 分解式增强

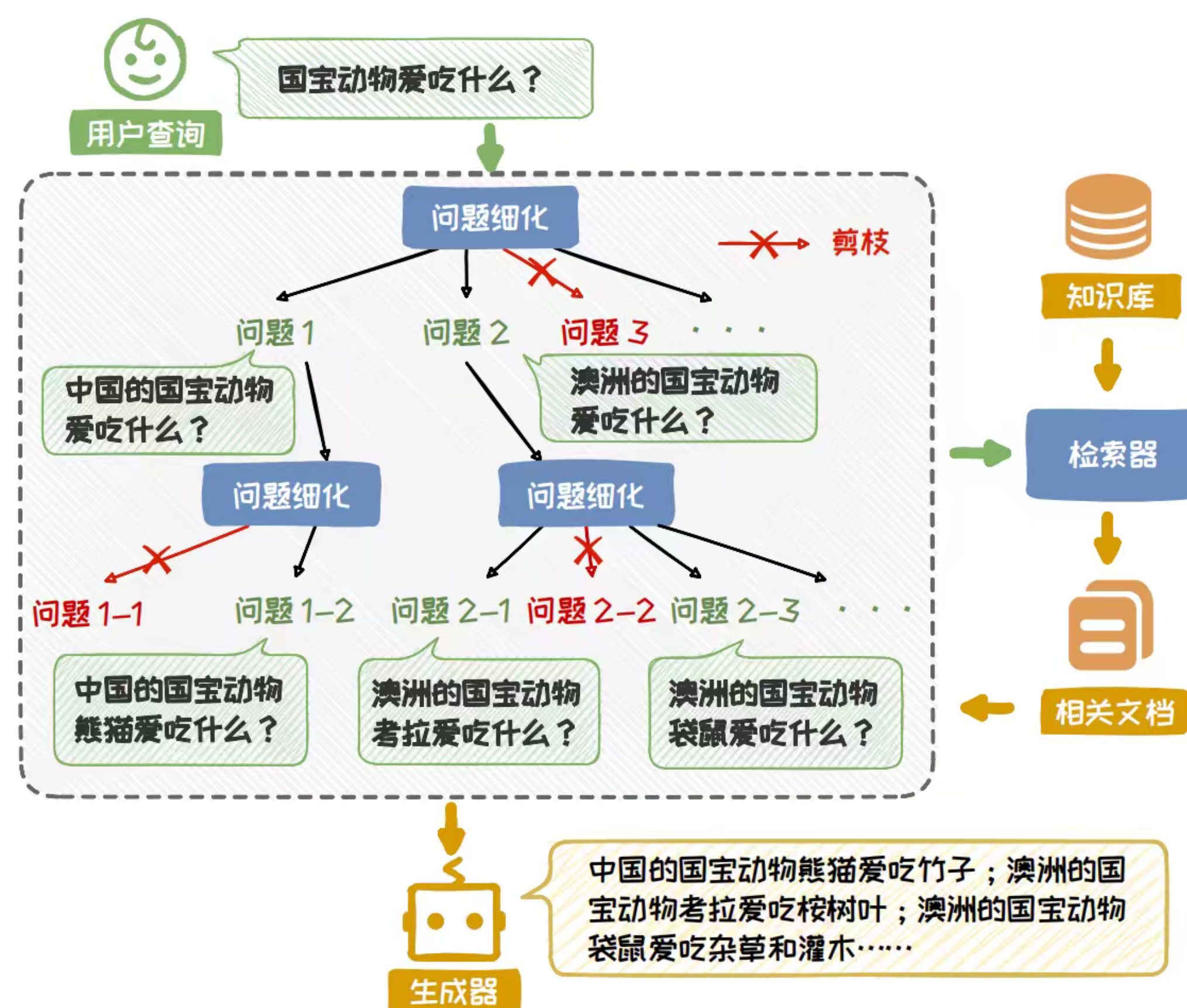
Demonstrate-Search-Predict (DSP)



性能取决于子问题分解质量

模糊问题 $\Rightarrow$ 渐进式增强

Tree-Of-Classification (TOC)



计算量指数级增长、多轮时间延迟、推理路径不稳定性

(4) 降本增效

去除冗余文本

Token级: 低PPL的Token $\rightarrow$ 更易预测到, 更普遍 $\rightarrow$ 冗余
LongLM LINGUA利用小模型评估PPL, 删除冗余文本

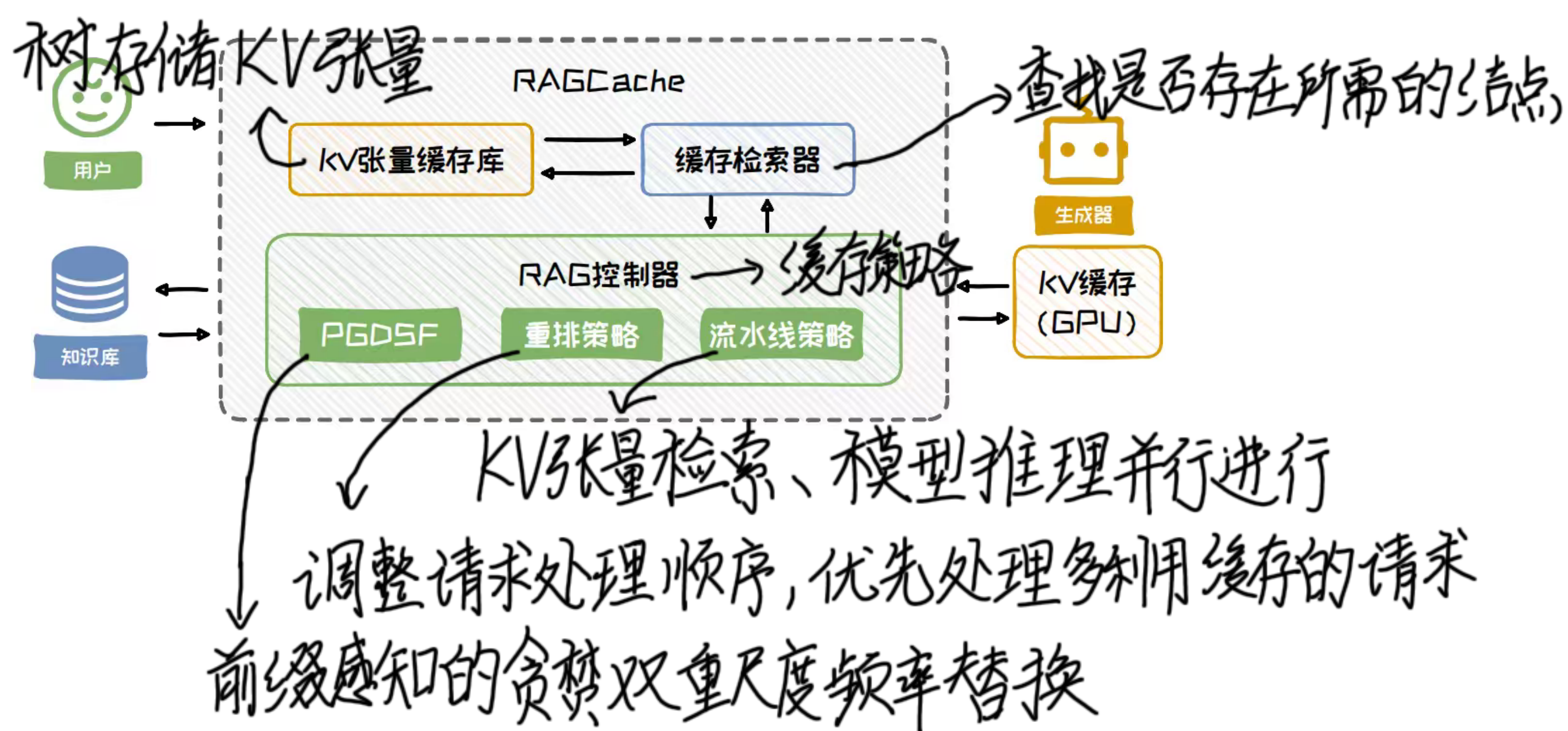
子文本级: FIT-RAG使用预训练双标签子文档对防器 (文档事实性、模型对其偏好程度)
利用滑动窗口评分, 最后删除

全文本级: PRCA先上下文提取, 训练使压缩文本与原文档差异尽量小
再以LLM作为奖励, 强化学习优化信息提取器

复用计算结果

生成每个Token都需要之前Token在注意力的Key & Value
RAG使输入文本长度增加 $\Rightarrow$ KV-cache显存占用增加
 $\Rightarrow$ 复用常用文本的KV-cache

RAGCache: 多级动态缓存



⑤ 应用

(1) LangChain

六模块: ModelIO、Retrieval、Chains、Memory、Agents、Callbacks

↓

文档加载、文本分割、
向量构建、索引生成、向量检索

↓

LLM自动决策操作

(2) LlamaIndex

更关注索引与检索 → 查询效率更高

(3) 实例

```
# pip install langchain langchain_community langchain-chroma langchain_openai
```

```
# 加载文档/网页内容
```

```
from langchain_community.document_loaders import WebBaseLoader
loader = WebBaseLoader("https://example.com/page")
docs = loader.load()
```

```
# 分割
```

```
from langchain_text_splitters import RecursiveCharacterTextSplitter
text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=200)
splits = text_splitter.split_documents(documents=docs)
```

```
# 向量化编码、存储
```

```
from langchain_chroma import Chroma
from langchain_openai import OpenAIEmbeddings
vector_store = Chroma.from_documents(documents=splits, embedding=OpenAIEmbeddings())
```

```
# 创建检索器和生成器
```

```
retriever = vector_store.as_retriever(search_kwargs={"k": 4})
import os
os.environ["OPENAI_API_KEY"] = "your_openai_api_key"
from langchain_openai import ChatOpenAI
llm = ChatOpenAI("gpt-3.5-turbo", temperature=0.7)
```



```
# LangChain Execution Language (LCEL) 构建链
from langchain import hub
from langchain_core.runnables import RunnablePassthrough
from langchain_core.output_parsers import StrOutputParser
def format_docs(docs):
    return "\n\n".join(doc.page_content for doc in docs)
prompt = hub.pull("rlm/rag-prompt")
RAG_chain = (
    {
        "context": retriever | format_docs,
        "question": RunnablePassthrough()
    }
    | prompt
    | llm
    | StrOutputParser()
)

# 使用RAG链回答问题
question = "What is the main topic of the webpage?"
response = RAG_chain.invoke(question)
print(response)
```

(4) 智能体

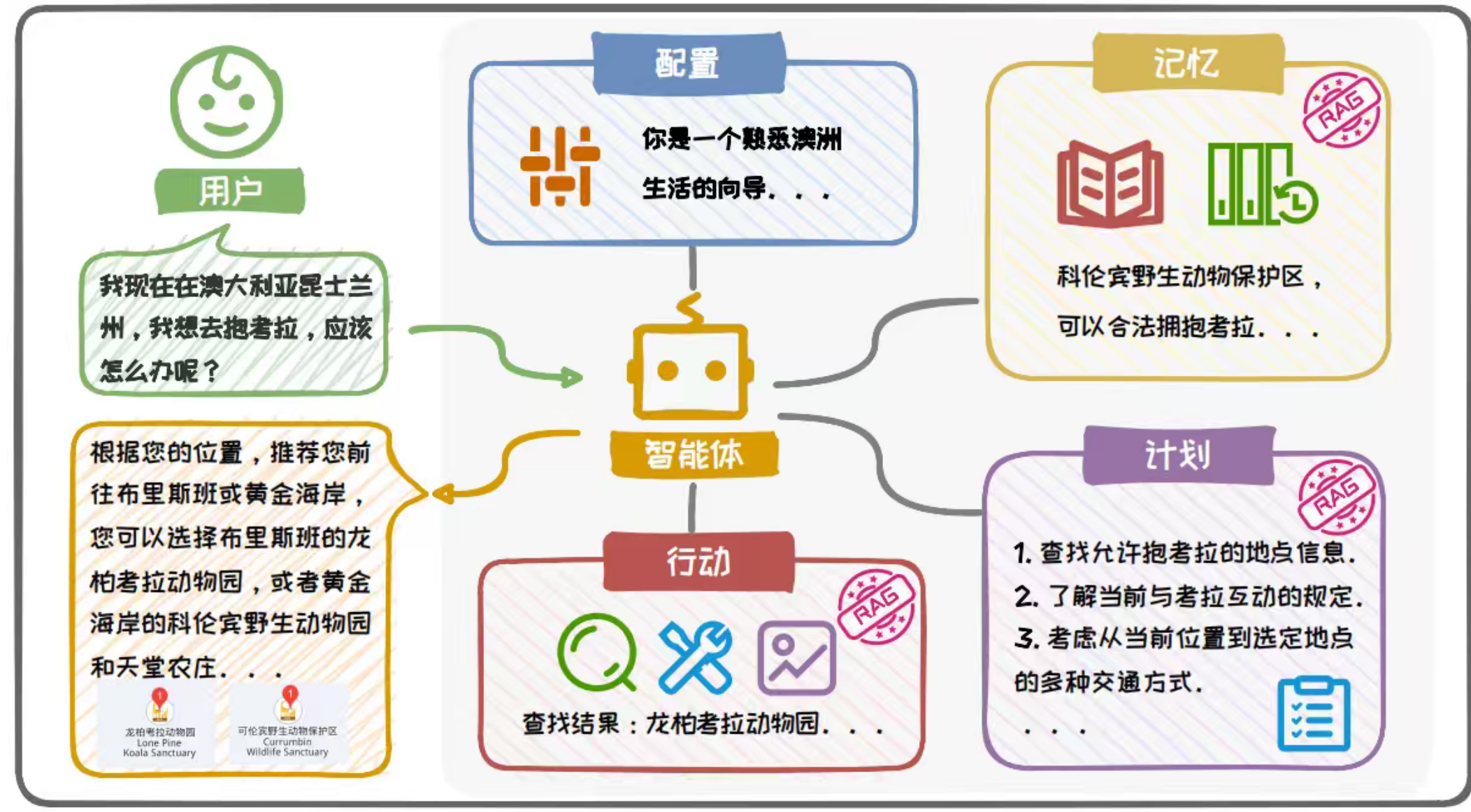
配置 Profile：定义 Agent 角色

记忆 Memory：存储学习到的知识和历史信息

计划 Planning：将复杂任务分解为简单子任务

行动 Action：网页检索、工具调用、多模态输出等

→ RAG



(5) 多模态垂直领域应用

