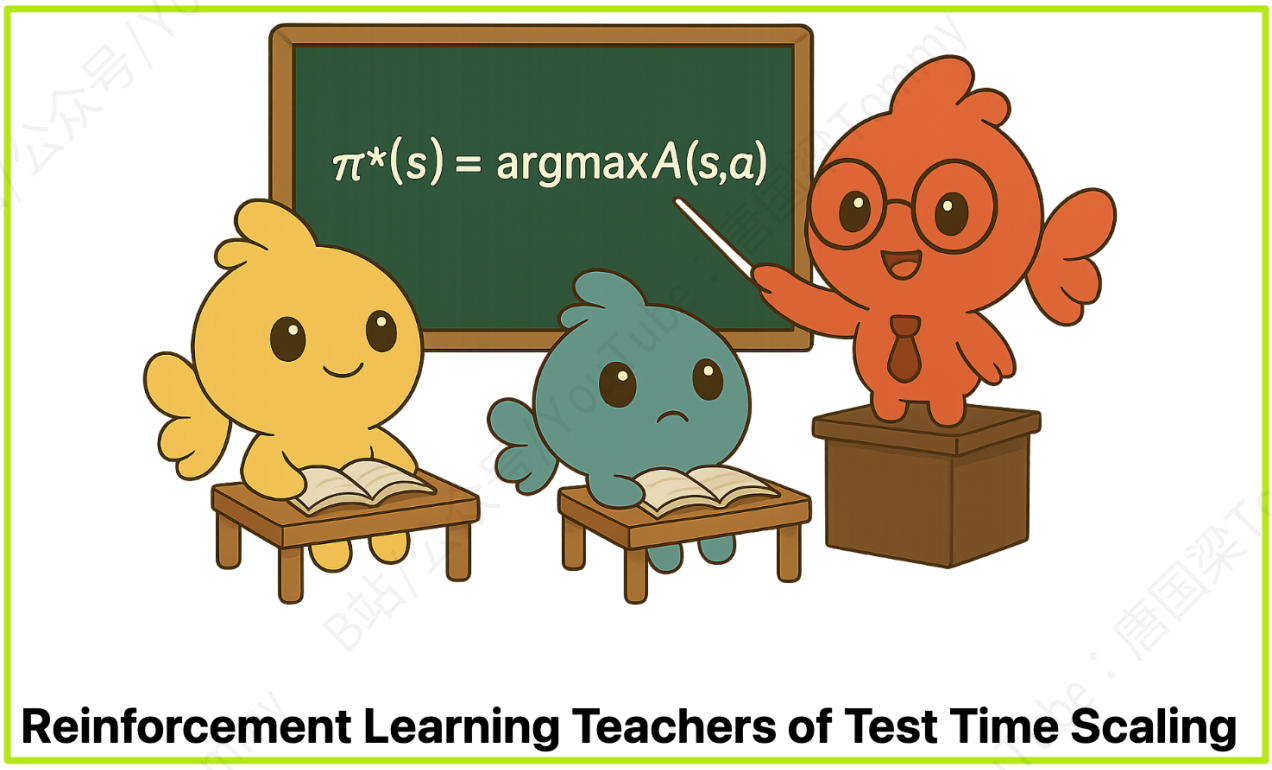
RLT (Reinforcement Learning Teachers) — 算法原理

1. 论文名称：Reinforcement Learning Teachers of Test Time Scaling
2. 第一作者：Sakana AI
3. 论文链接：https://arxiv.org/abs/2506.08388v2
4. 最新日期：2025年6月22日
5. github：https://github.com/SakanaAI/RLT.git

RLT (Reinforcement Learning Teachers) 是一个专门用于**训练测试时扩展强化学习（Test Time Scaling RL）教师模型**的高性能训练框架。该项目通过创新的两阶段训练管道（**SFT预热 + RL强化学习**），训练出能够在推理时产生高质量思维过程的教师模型，进而指导学生模型学习复杂推理能力。



1. RLT 概览与技术选型

1.1 问题背景与解决方案

- **问题**：传统的语言模型在复杂推理任务中缺乏系统性的思维过程，难以产生高质量的中间推理步骤。
- **解决方案**：通过强化学习训练**教师模型**，使其能够生成结构化的思维过程，然后用这些高质量的推理轨迹来训练更小的**学生模型**。
- **独特价值**：实现了测试时扩展（test-time scaling），即模型可以在推理时投入更多计算资源来提升答案质量。

Table 1: RLTs and prior distillation pipelines across model (7B and 32B) and data size (1K and 17K).

Model	Data size	AIME 2024	MATH 500	GPQA Diamond	Overall
QwQ-32B	N.A.	50.00	90.60	54.50	65.03
DeepSeek-R1	800K+	**79.80**	**97.30**	**71.50**	**82.87**
Qwen2.5-7B-Instruct	N.A.	10.00	74.20	33.30	39.17
Bespoke-7B-1K	1K	13.30	80.00	33.80	42.37
RLT-7B-1K (Ours)	1K	**20.00**	**80.40**	**41.90**	**47.43**
Bespoke-7B	17K	20.00	82.00	37.80	46.60
RLT-7B (Ours)	17K	**23.30**	**82.80**	**42.40**	**49.50**
Qwen2.5-32B-Instruct	N.A.	26.70	84.00	49.00	53.23
s1-32B	1K	50.00	92.60	56.60	66.40
s1-32B + budget forcing	1K	56.70	93.00	59.60	69.77
Bespoke-32B-1K	1K	46.70	92.60	57.50	65.60
RLT-32B-1K (Ours)	1K	**60.00**	**94.00**	**60.10**	**71.37**
Sky-T1-32B	17K	43.30	82.40	56.80	60.83
Bespoke-32B	17K	63.30	93.00	58.10	71.47
RLT-32B (Ours)	17K	**66.70**	**93.40**	**59.60**	**73.23**

1.2 与主流强化学习训练的主要区别

简单来说，传统方法试图训练一个**学生模型**，让它从零开始学会如何解决难题；而这篇论文的方法则是训练一个**教师模型**，这个教师的任务不是自己解决问题，而是基于一个已知的正确答案，生成最清晰、最有效的教学解释，从而更高效地教会学生模型。

1️⃣ 任务目标：从 “解决问题” 到 “解释答案”

- **主流RL方法**：学生模型接收一个问题（例如一个数学题），它的任务是自行探索，生成思考过程和最终答案。
- **RLT方法**：教师模型会同时接收到问题和**正确的解决方案**。它的任务不再是从头解决，而是**连接起点和终点**，即生成一个高质量的、循序渐进的解释，说明这个答案是如何得出的。

  论文的图2非常清晰地展示了这种差异。

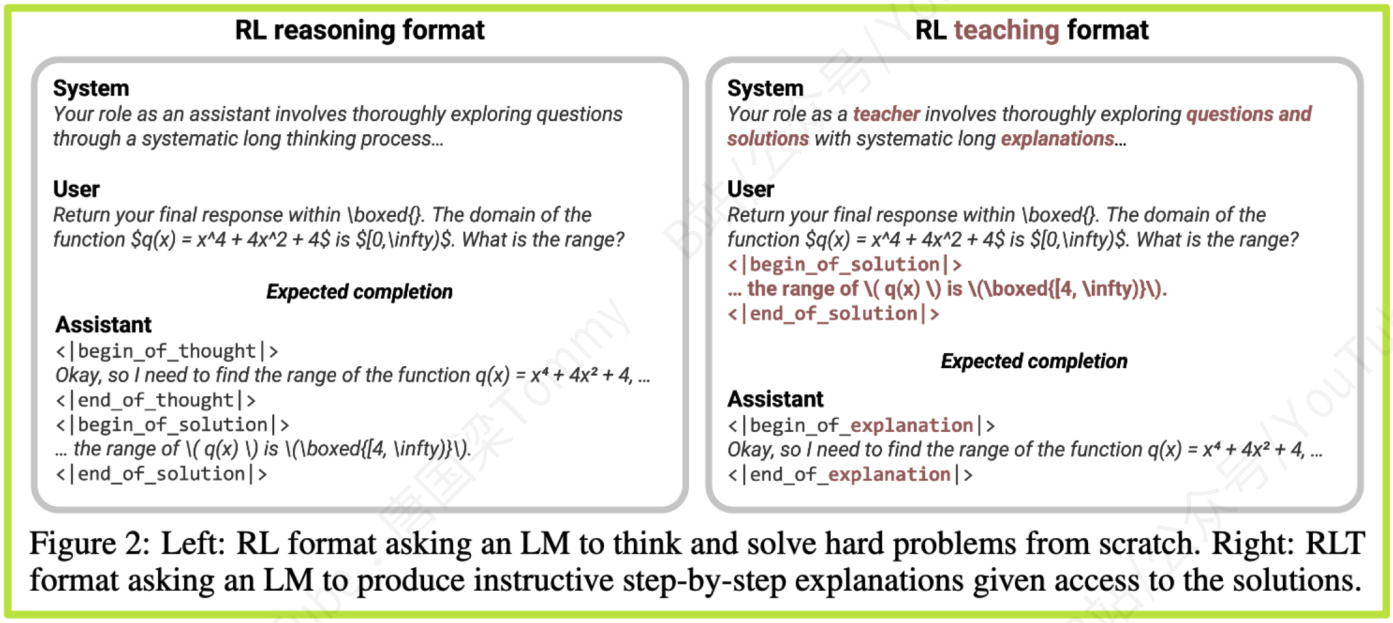


Figure 2: Left: RL format asking an LM to think and solve hard problems from scratch. Right: RLT format asking an LM to produce instructive step-by-step explanations given access to the solutions.

2 奖励信号：从 “稀疏奖励” 到 “密集奖励”

- **主流RL方法**：奖励通常是**稀疏的**（Sparse Reward）。只有当学生模型最终给出的答案完全正确时，它才能得到一个正向奖励（比如+1），否则就是0分或负分。这导致了所谓的**探索困境**：如果学生模型初期能力很弱，一次都做不对，它就永远得不到学习信号，训练很难进行。
- **RLT方法**：奖励是**密集的**（Dense Reward）。教师模型生成的**每一份解释**都会得到一个精确的分数。这个分数基于**两个核心指标**：（1）这份解释对于一个学生模型理解并得出正确答案有多大帮助；（2）这份解释的逻辑本身是否清晰、连贯。

  这种密集的奖励信号让训练过程更加稳定和高效，因为教师模型总能得到反馈。

3 核心优化对象：从 “解题能力” 到 “教学质量”

- **主流RL方法**：其目标是直接最大化模型独立解决问题的能力。
- **RLT方法**：其目标是最大化模型的**教学质量**。论文认为，在实际应用中，很多强大的RL模型最终的用途就是作为“教师”，生成高质量的数据集（称为“蒸馏数据”）来训练更小、更高效的“学生”模型。RLT框架直接优化这个最终目标，而不是一个中间目标，从而避免了目标错配的问题。

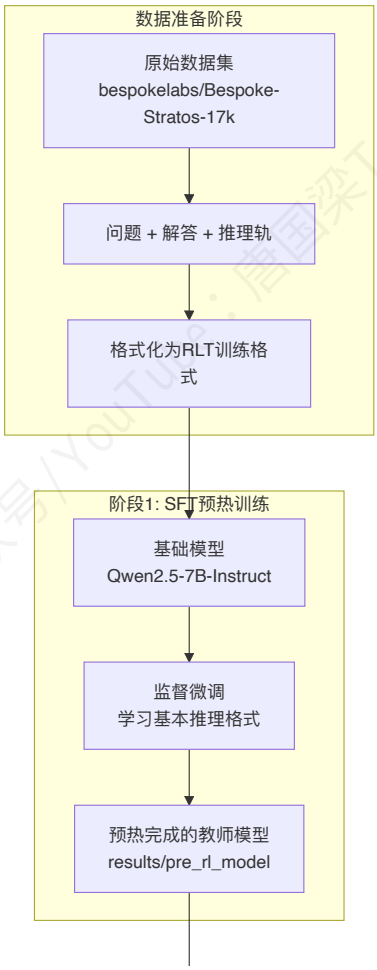
1.3 技术栈

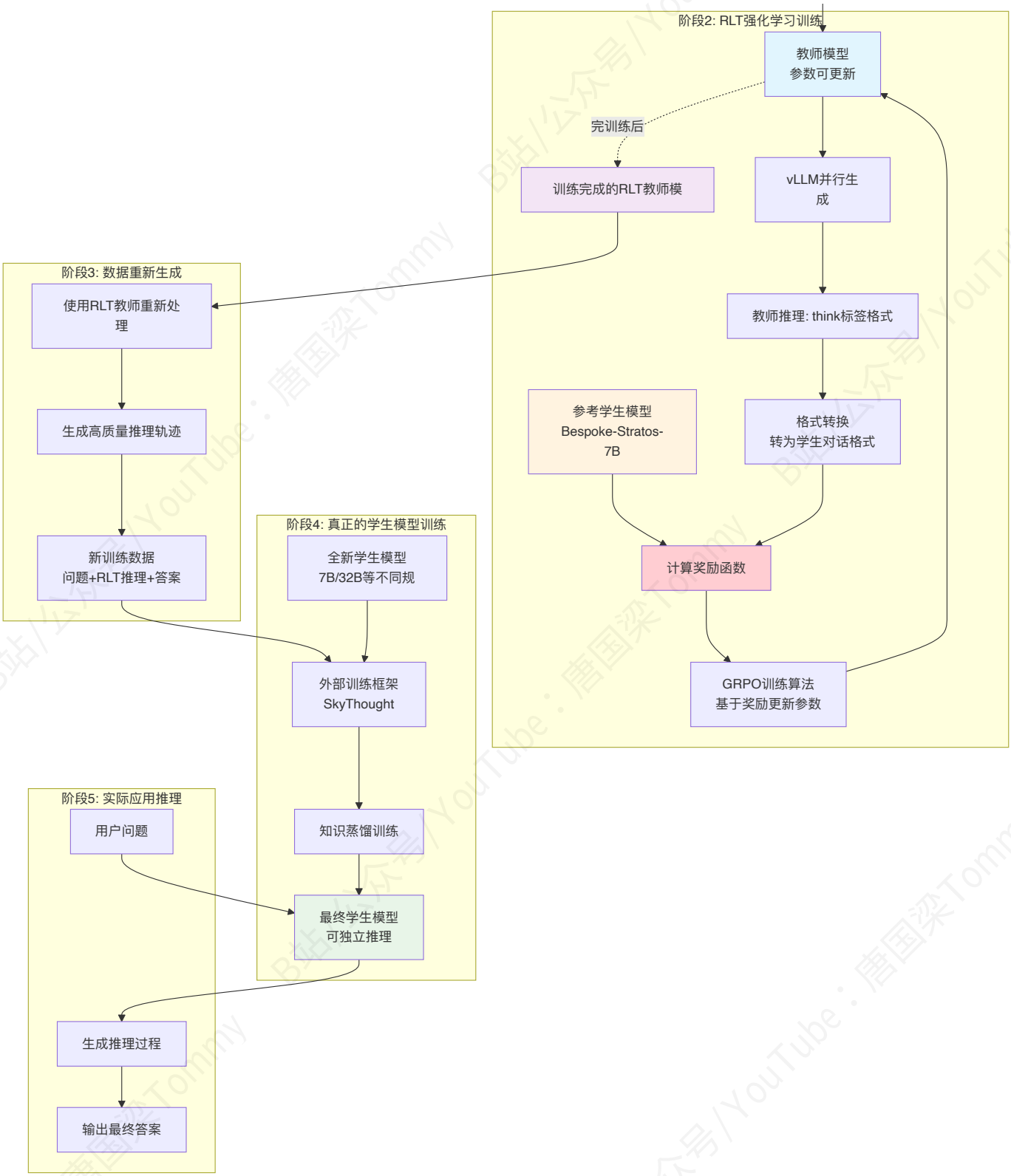
技术层次	技术组件	版本	作用
深度学习框架	PyTorch 2.6.0	最新稳定版	强大的分布式训练支持
大模型训练	Transformers 4.51.1	企业级稳定版	HuggingFace生态完整
强化学习	TRL 0.14.0	专业RL库	专门针对语言模型优化
分布式训练	DeepSpeed 0.15.4	微软开源方案	ZeRO优化内存效率高
高效推理	vLLM 0.8.3	业界领先	推理速度优化显著
分布式计算	Ray[serve]	分布式框架	易用的分布式抽象
配置管理	Hydra 1.3.2	Facebook开源	配置层次化管理优秀

2. 训练与推理逻辑

RLT（Reinforcement Learning Teachers）通过多阶段训练pipeline，先得到一个能生成高质量推理过程的教师模型，再用教师的输出来训练实用的学生模型，实现推理能力的有效传递。

整个系统的运作机制：





阶段1：SFT预热训练

目标：让基础模型学会基本的推理格式和结构

具体流程：

- **输入**：原始数据集（`bespokelabs/Bespoke-Stratos-17k`）
- **模型**：`Qwen/Qwen2.5-7B-Instruct`
- **训练方式**：标准监督微调（SFT）
- **学习内容**：`<|begin_of_thought|>...<|end_of_thought|>`格式的推理结构
- **输出**：具备基础推理格式能力的预热模型

阶段2：RLT强化学习训练（核心阶段）

目标：通过强化学习优化教师模型的推理质量

关键角色定义：

- **教师模型**：正在被训练的目标模型（参数更新）
- **参考学生模型**：质量评判标准（参数冻结，`Bespoke-Stratos-7B`）

训练循环机制：

第1步：并行推理生成

```
# 教师模型通过vLLM生成多个候选推理过程
num_generations: 64  # 每个问题生成64个不同的推理轨迹
temperature: 0.7     # 控制生成多样性
```

第2步：格式转换与对齐

```
# 教师格式："<|begin_of_thought|>推理内容<|end_of_thought|>解答"

# 转换为学生对话格式：
student_chat = [
    {"role": "system", "content": student_system_prompt},
    {"role": "user", "content": question},
    {"role": "assistant", "content": "推理+解答"}
]
```

第3步：多维度奖励计算

```
总奖励 = 解答质量奖励 + KL散度惩罚 + 格式匹配奖励

# 解答质量奖励：参考学生模型对解答部分的对数概率
answer_reward = student_model.compute_log_prob(solution_part)

# KL散度惩罚：教师与学生模型在推理过程上的差异
kl_penalty = KL_divergence(teacher_reasoning, student_reasoning)

# 格式匹配：确保生成内容符合预期标签格式
format_reward = match_think_tags(generated_text)
```

第4步：GRPO策略优化

```
# 基于奖励使用GRPO算法更新教师模型参数
loss = GRPO_loss(advantages, log_probs, ref_log_probs)
teacher_model.update_parameters(loss)
```

阶段3：数据重新生成

目标：使用训练完成的RLT教师为整个数据集生成高质量推理轨迹

过程：

1. **输入**：原始问题集合
2. **处理**：训练完成的RLT教师模型逐一生成推理过程
3. **输出**：包含高质量推理轨迹的新训练数据

```
每条数据：{
  "question"："原始问题",
  "reasoning_trace"："RLT生成的推理过程",
  "solution"："标准答案"
}
```

阶段4：真正的学生模型训练

目标：训练能独立进行推理的实用模型

实现方式：

- **训练框架**：外部SkyThought库（非本项目代码）
- **训练数据**：阶段3生成的高质量推理轨迹
- **训练方法**：知识蒸馏
- **学生模型**：全新的7B/32B等不同规模模型

- **最终产品**：能够独立推理的学生模型

阶段5：实际推理应用

使用场景：

```
# 用户输入问题
user_question = "解这个数学题..."

# 学生模型生成推理过程
response = student_model.generate(user_question)
# 输出："首先分析题目...[详细推理]...因此答案是x"
```

---

3. 稠密奖励函数（Dense Reward Function）

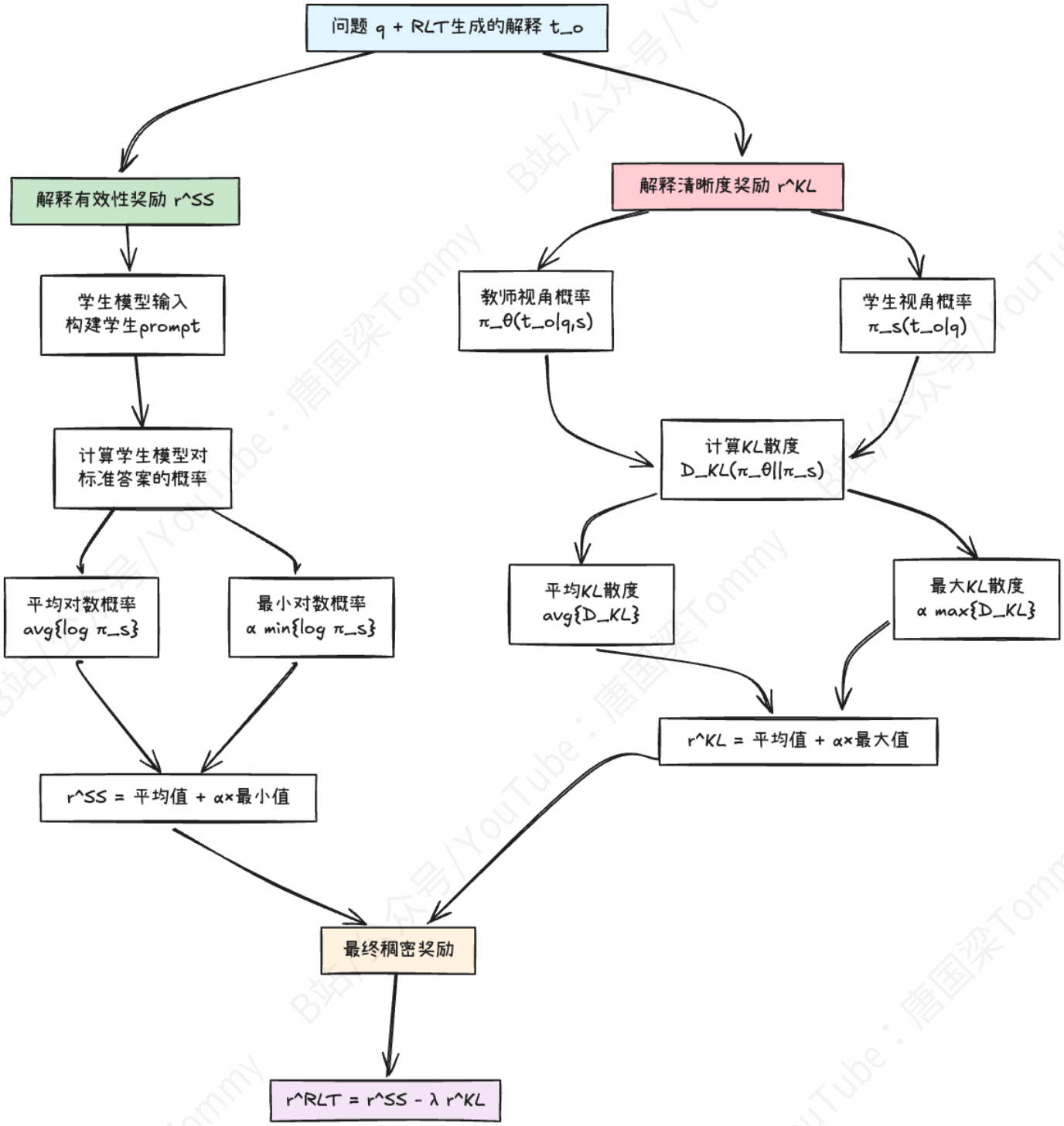
3.1 核心设计理念

稠密奖励函数的目标是为RLT（强化学习教师）模型在每个token级别提供**细粒度的反馈信号**，使其能够学会生成既有效又自然的教学解释。

与简单的0/1二元奖励不同，稠密奖励为每个生成的解释提供连续的分数，让模型在训练过程中获得更丰富的学习信号。

3.2 奖励函数构成

流程图：



3.2.1 解释有效性奖励 r^{SS}（Student Success）

(1) 核心思想

- 评估**学生在听完老师解释后，能否独立解出问题**的能力。这个奖励衡量的是解释的**最终效果**。

(2) 算法原理：

该项奖励量化了学生模型在阅读了RLT的解释 t_o 后，对标准答案 s_i 的理解程度。

它通过计算学生模型在给定问题 q_i 和解释 t_o 的条件下，生成答案 s_i 的对数概率（log probabilities）来衡量。

公式（3）定义为：

$$r^{\mathrm{SS}}(o_i, s_i, q_i) = \operatorname{avg}\{\log \pi_s^{s_i}\} + \alpha \min\{\log \pi_s^{s_i}\}, \quad \text{where} \quad \pi_s^{s_i} = \pi_s(s_i \mid t_{o_i}.q_i). \quad (3)$$

这里的 avg 和 $\min$ 操作确保了解释对答案的每个部分都有贡献，防止模型只关注答案的个别关键词。

r^{SS} 的分数由两部分组成：

- **平均对数概率 ($\operatorname{avg}\{\log \pi_s\}$)**: 这是所有正确答案token的对数概率的平均值。它代表了学生对整个答案的整体把握程度。分数越高，说明学生生成完整答案的信心越强。
- **最小对数概率 ($\alpha \min\{\log \pi_s\}$)**: 这是所有正确答案token的对数概率中的最小值，再乘以一个权重 α。这个项非常关键，它扮演了“短板效应”的角色。如果解释中有一个逻辑点没讲清楚，导致学生在生成某个关键token（比如 `+` 号）时犹豫不决（概率很低），那么即使其他部分概率很高，这个$\min$项也会拉低总分。它迫使RLT确保解释的每一步都清晰无误。

(3) 实现流程：

举例说明：

问题：求解方程 2x + 6 = 14

数据集中的正确答案：x = 4

1️⃣ 教师模型生成解释：

教师模型接收到问题+正确答案，生成详细解释：

```
# 输入到教师模型：

问题：求解方程 2x + 6 = 14
答案：x = 4

# 教师模型生成：
<|begin_of_solution|>x = 4<|end_of_solution|>

<|begin_of_explanation|>
我需要求解方程 2x + 6 = 14。

首先，我将常数项移到等号右边：
2x = 14 - 6
2x = 8

然后，两边同时除以2：
x = 8 ÷ 2
x = 4

让我验证一下：2(4) + 6 = 8 + 6 = 14 ✓
<|end_of_explanation|>
```

2️⃣ 构造学生评估对话：

系统构造学生模型的完整对话（包含标准答案）：

```
student_chat = [
    {
        "role": "system",
        "content": "你是一个数学学习助手..."
    },
    {
        "role": "user",
        "content": "求解方程 2x + 6 = 14"
    },
    {
        "role": "assistant",
        "content": """<think>
                    我需要求解方程 2x + 6 = 14。

                    首先，我将常数项移到等号右边：
                    2x = 14 - 6
                    2x = 8

                    然后，两边同时除以2：
                    x = 8 ÷ 2
                    x = 4

                    让我验证一下：2(4) + 6 = 8 + 6 = 14 ✓
                    </think>

                    <solution>x = 4</solution>
                """
    }
]
```

3️⃣ Token级别概率计算：

```
# 1. Tokenization
student_text = tokenizer.apply_chat_template(student_chat)
tokens = tokenizer.encode(student_text)
# 假设得到：[1, 45, 234, 567, ..., 2341, 894, 567, 23, 445]
#           ["求", "解", "方", "程", ..., "x", "=", "4", "</solution>"]

# 2. 学生模型前向传播
student_outputs = student_model(input_ids=tokens)
student_logits = student_outputs.logits

# 3. 计算每个token的对数概率
student_log_probs = []
```

```
for i in range(len(tokens)-1):
    next_token = tokens[i+1]
    logits_at_i = student_logits[i]  # 位置i的logits
    probs_at_i = F.softmax(logits_at_i, dim=-1)
    log_prob = torch.log(probs_at_i[next_token])
    student_log_probs.append(log_prob)

# 结果示例：
# student_log_probs = [-0.1, -0.3, -0.2, ..., -1.2, -0.8, -0.5, ...]
```

4️⃣ **掩码提取：**

```
# 创建答案部分的掩码（只关注"x = 4"）
solution_start = find_position("<solution>")
solution_end = find_position("</solution>")
solution_mask = create_mask(student_log_probs, solution_start, solution_end)
# solution_mask = [False, False, ..., True, True, True, False, ...]
#                                    对应 "x", "=", "4"
```

5️⃣ **计算 解释有效性奖励 r^{SS}**

```
# 提取答案部分的学生概率
solution_log_probs = student_log_probs[solution_mask]
# solution_log_probs = [-1.2, -0.8, -0.5]  # 对应 "x", "=", "4"

# 应用处理函数（这里使用恒等函数）
processed_log_probs = solution_log_probs  # normalize_log_prob_fn = null

# 应用约简函数（平均值）
mean_log_prob = torch.mean(processed_log_probs)  # (-1.2 + -0.8 + -0.5) / 3 = -0.83

# 应用系数得到最终奖励
r_SS = mean_log_prob * 5.0  # answer_log_prob_coeff = 5
# r_SS = -0.83 * 5 = -4.15
```

解释：学生模型对答案"x = 4"给出的平均对数概率是-0.83，说明学生对这个答案有一定信心，但不是很高。

3.2.2 解释清晰度奖励 r^{KL}（KL Divergence）

(1) 核心思想：

- 防止教师模型生成只有自己能理解的**天书式**解释，确保解释符合**普通学习者的认知规律**。

- 这个奖励衡量的是解释的“过程合理性”，用来防止RLT利用学生模型的弱点“作弊”。

(2) 算法原理：

该项奖励旨在确保RLT生成的解释对于学生模型来说是逻辑自然且易于理解的，而不是一些无法学习的捷径。它通过计算教师模型（在知晓答案的条件下）和学生模型（在不知晓答案的条件下）生成相同解释的概率分布之间的 D_{KL}**散度** 来衡量。D_{KL} 散度越小，说明解释越符合学生模型的认知逻辑。

公式（4）定义为：

$$r^{\mathrm{KL}}(o_i, s_i, q_i) = \operatorname{avg}\left\{\mathbb{D}_{\mathrm{KL}}\left(\pi_\theta^{t_{o_i}} \| \pi_s^{t_{o_i}}\right)\right\} + \alpha \max\left\{\mathbb{D}_{\mathrm{KL}}\left(\pi_\theta^{t_{o_i}} \| \pi_s^{t_{o_i}}\right)\right\}, \quad (4)$$
$$\text{where} \quad \pi_s^{t_{o_i}} = \pi_s(t_{o_i} \mid q_i), \quad \pi_\theta^{t_{o_i}} = \pi_\theta(t_{o_i} \mid s_i, q_i).$$

其中 $\pi_\theta^{t_{oi}}$ 代表教师在给定问题和答案下生成解释的分布，而 $\pi_s^{t_{oi}}$ 代表学生仅在给定问题下生成解释的分布。

r^{KL} 的分数同样由两部分组成，但这次它是作为**惩罚项**（所以最终奖励是减去它）：

- **平均KL散度 (**$\operatorname{avg}\{D_{\mathrm{KL}}\}$**)**: 衡量了解释的整体自然度。
- **最大KL散度 (**$\alpha \max\{D_{\mathrm{KL}}\}$**)**: 这是对解释中最突兀、最不自然的那一步（KL散度最大的token）的“严厉惩罚”。它防止RLT在解释中夹带任何让学生无法理解的“私货”。

(3) 实现流程：

举例说明：

假设我们有一个数学推理问题：

```
问题：Solve for x: 3x - 9 = 12
```

根据论文，教师模型被同时给定问题和正确答案，任务是生成一个详细的解释来"连接"问题和答案：

1️⃣ **教师模型生成的解释**

```
<|begin_of_explanation|>
我需要求解方程 3x - 9 = 12。

首先，我将常数项移到等号右边：
3x = 12 + 9
3x = 21

然后，我将两边都除以3：
x = 21 ÷ 3
x = 7

让我验证一下：3(7) - 9 = 21 - 9 = 12 ✓
<|end_of_explanation|>
```

2️⃣ 构造学生对话

学生模型需要评估包含教师思维过程和正确答案的完整对话：

```
student_completion = """
<think>
我需要求解方程 3x - 9 = 12。

首先，我将常数项移到等号右边：
3x = 12 + 9
3x = 21

然后，我将两边都除以3：
x = 21 ÷ 3
x = 7

让我验证一下：3(7) - 9 = 21 - 9 = 12 ✓
</think>

<solution>x = 7</solution>
"""
```

3️⃣ Token级别概率计算

教师模型概率计算：

```
# 教师模型计算自己对思维过程的概率
teacher_text = """<|begin_of_explanation|>
我需要求解方程 3x - 9 = 12。
首先，我将常数项移到等号右边：
3x = 12 + 9
3x = 21
然后，我将两边都除以3：
x = 21 ÷ 3
x = 7
让我验证一下：3(7) - 9 = 21 - 9 = 12 ✓
<|end_of_explanation|>"""

# Tokenization（简化表示）
tokens = ["我", "需要", "求解", "方程", "3x", "-", "9", "=", "12", "。",
          "首先", ",", "我", "将", "常数", "项", "移到", "等号", "右边", ":",
          "3x", "=", "12", "+", "9", "3x", "=", "21",
          "然后", ",", "我", "将", "两边", "都", "除以", "3", ":",
          "x", "=", "21", "÷", "3", "x", "=", "7",
          "让我", "验证", "一下", ":", "3", "(", "7", ")", "-", "9", "=", "21", "-", "9",
"=", "12", "✓"]

# 教师模型的对数概率（示例值）
```

```
teacher_log_probs = torch.tensor([
    -0.1, -0.2, -0.1, -0.3, -0.5, -0.1, -0.2, -0.1, -0.4, -0.1,  # "我需要求解方程 3x - 9 =
12。"
    -0.2, -0.1, -0.2, -0.3, -0.4, -0.2, -0.3, -0.2, -0.3, -0.1,  # "首先，我将常数项移到等号右
边："
    -0.3, -0.1, -0.6, -0.2, -0.4, -0.2, -0.1, -0.5,              # "3x = 12 + 9 3x = 21"
    -0.2, -0.1, -0.2, -0.3, -0.4, -0.2, -0.3, -0.4, -0.1,        # "然后，我将两边都除以3："
    -0.3, -0.1, -0.7, -0.3, -0.5, -0.2, -0.1, -0.6,              # "x = 21 ÷ 3 x = 7"
    -0.3, -0.2, -0.3, -0.1, -0.8, -0.2, -0.9, -0.2, -0.3, -0.4, -0.1, -0.7, -0.3, -0.4,
-0.1, -0.6, -0.1  # 验证部分
])
```

学生模型概率计算:

```
# 学生模型计算对相同思维过程的概率
student_log_probs = torch.tensor([
    -0.3, -0.4, -0.2, -0.5, -0.8, -0.2, -0.4, -0.2, -0.6, -0.2,  # "我需要求解方程 3x - 9 =
12。"
    -0.4, -0.2, -0.4, -0.5, -0.6, -0.4, -0.5, -0.4, -0.5, -0.2,  # "首先，我将常数项移到等号右
边："
    -0.5, -0.2, -1.0, -0.4, -0.6, -0.4, -0.2, -0.8,              # "3x = 12 + 9 3x = 21"
    -0.4, -0.2, -0.4, -0.5, -0.6, -0.4, -0.5, -0.6, -0.2,        # "然后，我将两边都除以3："
    -0.5, -0.2, -1.2, -0.5, -0.8, -0.4, -0.2, -1.0,              # "x = 21 ÷ 3 x = 7"
    -0.5, -0.4, -0.5, -0.2, -1.2, -0.4, -1.4, -0.4, -0.5, -0.6, -0.2, -1.0, -0.5, -0.6,
-0.2, -1.0, -0.2  # 验证部分
])
```

KL散度计算:

```
# 计算KL散度：KL = P_teacher * log(P_teacher / P_student)
# 在对数空间中：KL ≈ teacher_log_probs - student_log_probs

kl_divergence = teacher_log_probs - student_log_probs
```

奖励处理和计算:

```
# 1. 应用处理函数（这里为恒等函数）
processed_kl = kl_divergence  # normalize_kl_fn = null

# 2. 应用约简函数（平均值）
mean_kl = torch.mean(processed_kl)
# mean_kl = (0.2 + 0.2 + 0.1 + ... + 0.1) / 54 ≈ 0.21

# 3. 应用系数并取负值（因为我们希望KL散度小）
kl_penalty_coeff = 5.0
r_KL = -1 * mean_kl * kl_penalty_coeff
# r_KL = -1 * 0.21 * 5.0 = -1.05
```

3.2.3 最终的稠密奖励

最终的奖励函数是：

$$r^{\mathrm{RLT}} = r^{\mathrm{SS}} - \lambda r^{\mathrm{KL}}$$

这个公式完美地平衡了两个目标：

- 通过 r^{SS} **鼓励**模型生成**有效**的解释。
- 通过 r^{KL} **惩罚**模型生成**不自然、难懂或作弊**的解释。

通过这个精心设计的稠密奖励，RLT模型在训练的每一步都能得到细致的反馈，从而学会如何成为一名既能让学生听懂（高r^{SS}），又能循循善诱、符合学生认知（低r^{KL}）的优秀教师。