

Stop Overthinking

面试官：思维链长返回慢、思维链短结果不准，你会如何平衡？



试答：

思维链长确实能提升复杂问题的准确率，但也容易“想太多”，带来高延迟和高成本；思维链太短虽快，却在复杂任务上掉点明显。我的做法是把它做成动态可调的问题，从三层去平衡：

第一，模型层面：做渐进式微调，让模型在长/短 CoT 混合数据上删除冗余保留要点 (SFT/LoRA)，配偏好式目标（如 DPO/GRPO：更短但同样正确的样本记为“更优”），再用长度奖励的 RL（答对才奖更短、超长惩罚）收紧推理习惯；工程上可结合蒸馏/模型融合把长链模型的能力迁到更简洁的学生上。实践里，像 DeepSeek-R1、QwQ-32B 都用过长度奖励思路。

第二，输出层面：做动态早停与压缩。当多路候选已高度一致（自一致性/奖励分过阈），就提前收敛；不一致才加预算。压缩方面，既有后处理（先长 CoT，再用规则或强模型精简，典型如 TokenSkip），也有在线压缩（推理时控长：LearnSkip 跳过非关键步骤、Token-Budget 限定 token、Self-Training 选最短正确路径回流训练）。如果场景允许，还可用潜在空间压缩，把部分中间思考放进隐向量（如 Coconut 的“连续思维 token”），减少可见文本长度。

第三，任务层面：自适应路由与提示控长。简单题走短链/小模型，复杂或不确定再触发长链/强模型；或者在提示里直接给规则，如：`Let's think step by step, but within 30 tokens.` / `Be concise.`

总结一下，我会通过**模型优化、推理过程控制和任务自适应**三方面结合，把思维链从固定长短变成动态可调，既保证用户不会等太久，又能维持复杂问题的解题能力。

为了更全面地回答这个面试题，同时能扛得住追问，今天来读一篇综述《Stop Overthinking: A Survey on Efficient Reasoning for Large Language Models》。

本文目录如下：

- 大模型也会“想太多”？
- 三大解法
 - 模型层面的方法：教模型少绕弯路（渐进微调 / 蒸馏 / 模型融合）
 - 输出层面方法：让推理自动调整（早停 / 压缩推理 / 奖励引导）
 - 输入层面方法：难题深想，易题快答（自适应控制 / 动态切换）

- 其他的讨论

- 如何用更少的数据训练推理模型？
- 如何提升小模型（SLM）的推理能力？

大模型也会“想太多”？

自从 DeepSeek-R1 引入长思维链（Chain of Thought, CoT）后，逐步推理已经被证明能显著提升模型在数学、逻辑、编程等复杂任务中的正确率。模型像人一样，把思考过程铺展开来，往往能更可靠地抵达正确答案。

然而，这种深度思考并非没有代价：即便是“ $2+3=?$ ”这样的简单问题，模型也可能写上百甚至上千个 token 才给出答案：



这种过度思考（Overthinking）现象带来了三大副作用：

- **响应变慢：** 用户等待时间过长，体验感差；
- **成本变高：** 冗余推理步骤增加了算力和费用；
- **场景受限：** 对于实时对话、自动驾驶、搜索引擎等对延迟高度敏感的场景，大模型显得不够实用。

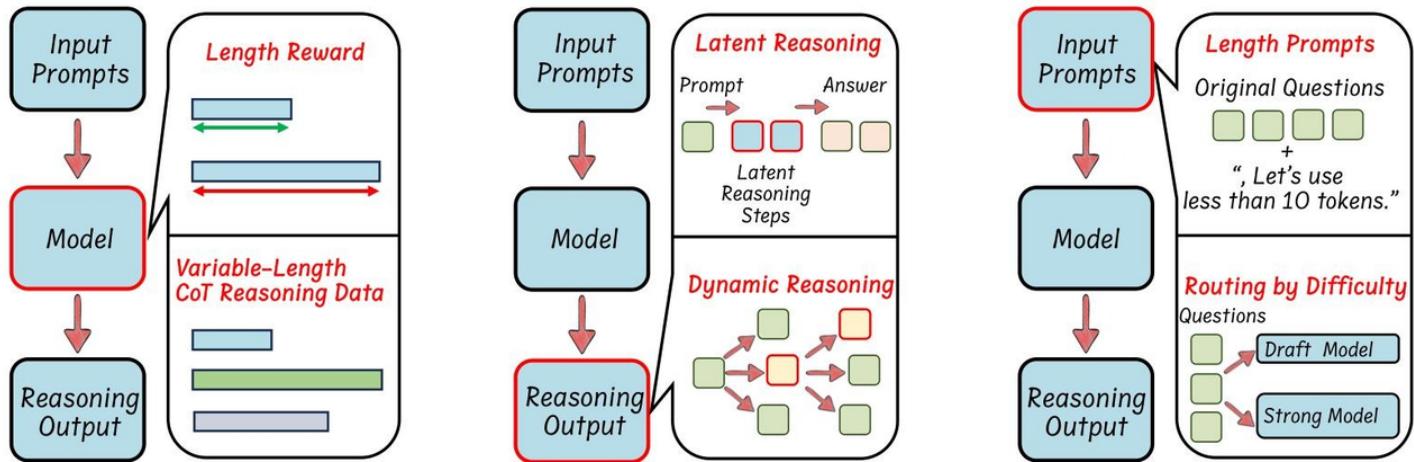
于是，一个看似悖论的矛盾出现了——思维链太长，结果更准却更慢；思维链太短，响应更快却容易出错。

为了缓解这一矛盾，很多研究者提出了多种高效推理（Efficient Reasoning）方法，核心目标是在尽量保持推理能力的前提下，减少冗余步骤、压缩思维链长度。主要可以分为三类：

1. **模型层面：** 通过奖励设计和变长思维链数据的SFT，让模型在训练层面学会更短、更高效的推理。
2. **输出层面：** 通过压缩潜在推理步骤或动态控制推理范式，在推理过程中直接减少冗余步骤。
3. **输入层面：** 在推理开始前，就通过prompt设计进行约束或引导，例如长度提示或按问题难度路由到不同模型。

此外，还有一些相关方向，例如利用高效数据训练、探索小模型的推理潜力、以及模型压缩方法，都是为了在“速度—准确率—成本”之间寻求更优解。

下面这张总览图把现有高效推理方法按三大类与六条代表性线路对应起来：



图中对应的六条路线分别是：

- I **长度奖励的RL (RL with Length Reward)**：在强化学习中加入长度相关奖励，鼓励短且对的推理序列，抑制过度思考。
- II **变长CoT的SFT (SFT with Variable-Length CoT)**：构造长/短思维链数据做监督微调，让模型学会更紧凑的链条。
- III **潜在思维压缩 (Latent Reasoning)**：把显式文字推理压到隐空间或更少的思考token，减少显式输出长度。
- IV **动态推理范式 (Dynamic Reasoning at Inference)**：在推理时按置信度/一致性等准则早停、剪枝或重采样，以较小开销获得稳定答案。
- V **基于提示的精简 (Prompt-Guided)**：用长度/步骤预算等提示直接约束推理输出（如“少于10个token”）。
- VI **按难度路由 (Routing by Difficulty)**：根据题目难度或不确定性，把请求路由到快模型（Draft Model）或强模型（Strong Model），兼顾时延与正确率。

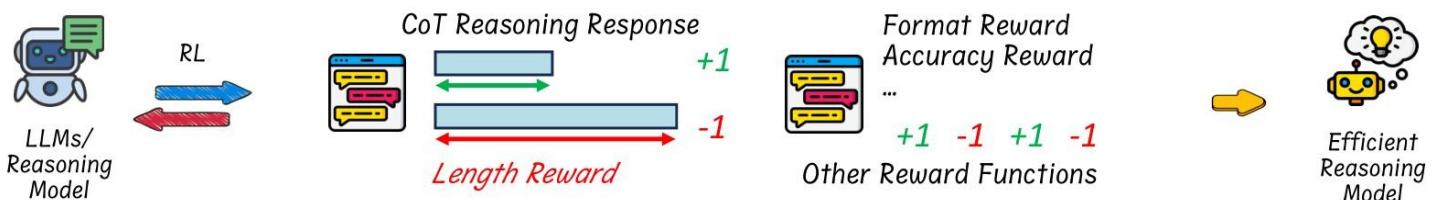
模型层面的方法

模型导向的方法目标是把“高效”写进模型本身：在训练阶段就让模型偏好**更短且稳定的思维链**。主线有两条：(1) 强化学习加入长度奖励；(2) 用变长CoT数据做SFT。

强化学习加入长度奖励

核心问题：怎样把“既要对、也要短”写进奖励函数？

做法是在RL的回报里同时编码**正确性与长度**：鼓励“短且对”，惩罚“长且错/冗余”，以抑制过度思考（下图示意）。实践中常搭配格式/过程奖励，避免模型靠删字取巧。



四种常见的长度奖励设计（配合 PPO/PG/GRPO/SimPO 等优化框架）：

- **条件化长度**：只有在答对时才按“更短→更高分”计分；答错不因为短就得分。这样能把“先对再短”的优先级固定下来。
- **超限惩罚**：当推理长度达到上限 L_{\max} （比如设置的 token 上限）时，直接追加惩罚项 r_e ，防止拖长。
- **相对长度**：把当前推理长度与一个可度量的参考做对比，再据此奖励/惩罚。常见参考包括：
 - 参考模型的 CoT 长度 L_{ref} （例如一条强模型或教师模型产生的解），目标是比它更短而不掉准确率；
 - 目标长度/预算 B （来自业务约束或外部估算器）；
 - 标注推理链的长度 $L(y^*)$ （若数据集自带标准解的 CoT）。

这些参考都能把“短到什么程度算合适”量化出来。

- **形状函数**：把“长度→得分”的映射做成平滑曲线（如线性或余弦衰减），让梯度更稳，避免训练时出现“要么极短要么极长”的震荡。直观地说：越长扣分越多，但扣分是渐进加重而非突然断崖。

下表1 对比了长度奖励的一些代表性做法，思路分别如下：

- **O1-Pruner (PPO)**：使用长度协调奖励：比较当前 CoT 与参考模型 CoT 的长度比值 $L_{\text{ref}}/L_{\text{pred}} - 1$ ，更短得分更高；再加与参考答案一致性的约束，确保“缩短不降准”。
- **Demystifying (PPO)**：对正确/错误两种情形分别用余弦形状的长度曲线评分；到达 L_{\max} 给超限罚分，既稳住训练又抑制长链。
- **L1 (GRPO)**：在输入里显式加入“Think for N tokens.”，并用与目标长度（来自标注 CoT 或设定）之间的绝对偏差作为惩罚项，鼓励在保证正确前提下对齐到期望的 token 数。
- **DAST (SimPO)**：不依赖参考模型，先构造长度偏好对（短而对优于长而对/错），再用 SimPO 直接进行偏好优化。
- **Training (PG)**：在策略梯度的回报里乘上 $\text{Correct} \times (1 - \alpha f(L))$ 这类长度惩罚因子，把“短一点”直接写进回报，形式最简。

Method	RL	Length Constraint Reward	Data	Model
O1-Pruner [127]	PPO	$\mathbb{E}_{x \sim D} \left[\mathbb{E}_{\pi_\theta, \pi_{ref}} \left[\frac{L(y_{pred})}{L(y_{gt})} \right] - 1 \right]$	GSM8K GaoKao MATH-500	Marco-o1-7B QwQ-32B-Preview
Demystifying [217]	PPO	$\begin{cases} r_0^c + 0.5 \times (r_L^c - r_0^c)(1 + \cos(\frac{\pi L(y_{pred})}{L_{max}})), & \text{if correct,} \\ r_0^w + 0.5 \times (r_L^w - r_0^w)(1 + \cos(\frac{\pi L(y_{pred})}{L_{max}})), & \text{if wrong} \\ r_e, & \text{if } L(y_{pred}) = L_{max}, \end{cases}$	MATH-500 AIME-2024 TheoremQA MMLU-Pro-1k	LLaMA-3.1-8B Qwen2.5-7B-Math
L1 [2]	GRPO	$\begin{cases} x_{new} = \text{CONCAT}(x, "Think for N tokens."), \\ r(y, y_{GT}, L(y_{GT})) = \mathbb{I}(y_{pred} = y_{GT}) - \alpha \cdot L(y_{GT}) - L(y_{pred}) \end{cases}$	AMC GPQA LAST MMLU MATH-500 AIME-2024 Olympiad-Bench	DeepSeek-R1-Distill-Qwen-1.5B
DAST [154]	SimPO	Trained with constructed length preference data	MATH-500 AIME-2024	DeepSeek-R1-Distill-Qwen-7B DeepSeek-R1-Distill-Qwen-32B
Training [5]	PG	$\mathbb{E}_{x \sim D} [1\{y_{pred} = y_{GT}\}(1 - \alpha f(L(y_{pred})))]$	GSM8K MATH-500 AIME-2024	DeepSeek-R1-Distill-Qwen-1.5B DeepSeek-R1-Distill-Qwen-7B

表1：长度奖励的代表性做法

Method	Optimization Objective
Policy Gradient (PG)	$\mathbb{E}_{\pi_\theta} [\nabla_\theta \log \pi_\theta(y_t x_t) \hat{R}_t]$
PPO [150]	$\mathbb{E} \left[\min \left(\frac{\pi_\theta(y_t x_t)}{\pi_{\theta_{ref}}(y_t x_t)} \hat{R}_t, \text{clip} \left(\frac{\pi_\theta(y_t x_t)}{\pi_{\theta_{ref}}(y_t x_t)}, 1 - \epsilon, 1 + \epsilon \right) \hat{R}_t \right) \right]$
SimPO [131]	$\mathbb{E} \left[\log \sigma \left(\frac{\beta}{ y_t^w } \log \pi_\theta(y_t^w x_t) - \frac{\beta}{ y_t^l } \log \pi_\theta(y_t^l x_t) - \gamma \right) \right]$
GRPO [151]	$\mathbb{E} \left[\min \left(\frac{\pi_\theta(y_t^G x_t)}{\pi_{\theta_{ref}}(y_t^G x_t)} \hat{R}_t^G, \text{clip} \left(\frac{\pi_\theta(y_t^G x_t)}{\pi_{\theta_{ref}}(y_t^G x_t)}, 1 - \epsilon, 1 + \epsilon \right) \hat{R}_t^G \right) - \lambda \mathbb{D}_{KL}[\pi_\theta \pi_{ref}] \right]$

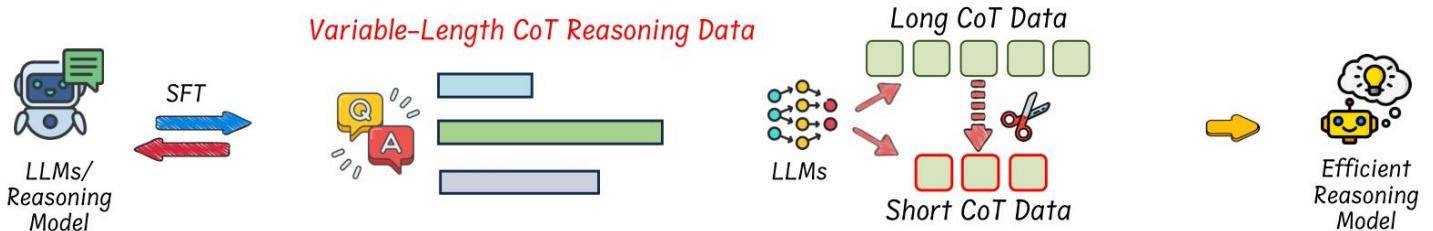
表2：CoT 长度约束下的策略优化目标一览

总体而言，通过设计RL中的长度奖励，能在不显著牺牲准确度的前提下有效缩短CoT长度。这种方法的难点在于确定不同的任务的合理推理长度，进而设计出合理的长度奖励。

用变长 CoT 数据做SFT

即先获得长+短并存的 CoT 数据，再做 SFT 让模型学会“该长则长、该短则短”的表达。

这个路线既可直接提升推理效率，也可配合 RL 使用。



第一个问题是，如何收集包含长度不同的CoT推理数据，特别是比较短的CoT数据呢？

长 CoT 数据可以通过提示预训练模型收集，基于这些长 CoT 数据，可以通过以下方式收集短 CoT 数据。

- **后处理 (Post-Reasoning) 压缩**

- 利用规则或更强模型（如 GPT-4）对已生成的长 CoT 进行压缩，删去冗余推理步骤，仅保留关键信息（例如 C3oT、TokenSkip 等方法）。优点是压缩率高、可控。

- **在线 (During-Reasoning) 压缩**

- 在生成阶段就产出**更短**的链条：

- **LearnSkip**：人工/随机跳过部分步骤，并用在 n 步内完成的提示构造训练样本。
- **Self-Training**：一次采样多条推理路径，选最短的正确解回流为训练数据。
- **Token-Budget**：为每道题估计最优 token 预算（可用二分搜索寻找），按该预算生成短链。
- **参数混合 (CoT-Valve)**：将“无思维”与“长思维”两套 LoRA 参数线性混合，调节混合系数 α 来控制生成长度并产出变长样本。

这类方法的优点是：**短链源自模型内生分布**，更贴近模型的真实表达。

第二个问题是，怎么通过微调实现高效推理？

- **标准 SFT**：用 LoRA 或全量微调，在变长 CoT 数据上最小化困惑度或偏好损失。成本低、适配广。
- **渐进式微调 (Progressive Fine-tuning)**：训练过程中逐步缩短样本链条（如 LearnSkip），或采用 CoT-Valve 的参数混合策略：先用“无思维”权重 (α 大) 开始，再逐步减小 α 向“长思维”过渡，从而平滑地学会不同长度的推理表达并整体压短。

Method	Source Data	Reasoning Pruning	SFT	LLMs
Self-Training [59]	GSM8K MATH	Sampling N reasoning then select the shortest one	Standard	Llama-3.2-{1B,3B} Llama-3.1-8B
TokenSkip [89]	GSM8K MATH	Skip tokens according to semantic importance	Standard	LLaMA-3.1-8B-Instruct Qwen2.5-Instruct
C3oT [37]	GSM8K MathQA ECQA StrategyQA	GPT-4 as compressor to make concise reasoning	Standard	Llama-2-chat-{7B,13B}
Distilling2-1 [99]	OASST2	Removing reasoning	Standard	Llama-2-70B-chat
Token-Budget [32]	GSM8K GSM8K-Z MathBench	Pursuing an optimal token budget for LLMs to complete the reasoning	Standard	Llama-3.1-8B-Instruct
CoT-Value [56]	GSM8K PRM800k	Merging parameters of non-reasoning and long reasoning LLMs	Progressive	QwQ-32B-Preview DeepSeek-R1-Distill-Llama-8B LLaMA-3.1-8B LLaMA-3.2-1B Qwen32B-Instruct
LearnSkip [50]	Analog of Algebra Multi-digit Addition Directional Reasoning	Stage 1: Manually skipping Stage 2: Prompting LLMs for shorter reasoning	Standard & Progressive	Llama-2-7B Phi-3-mini (3.8B)

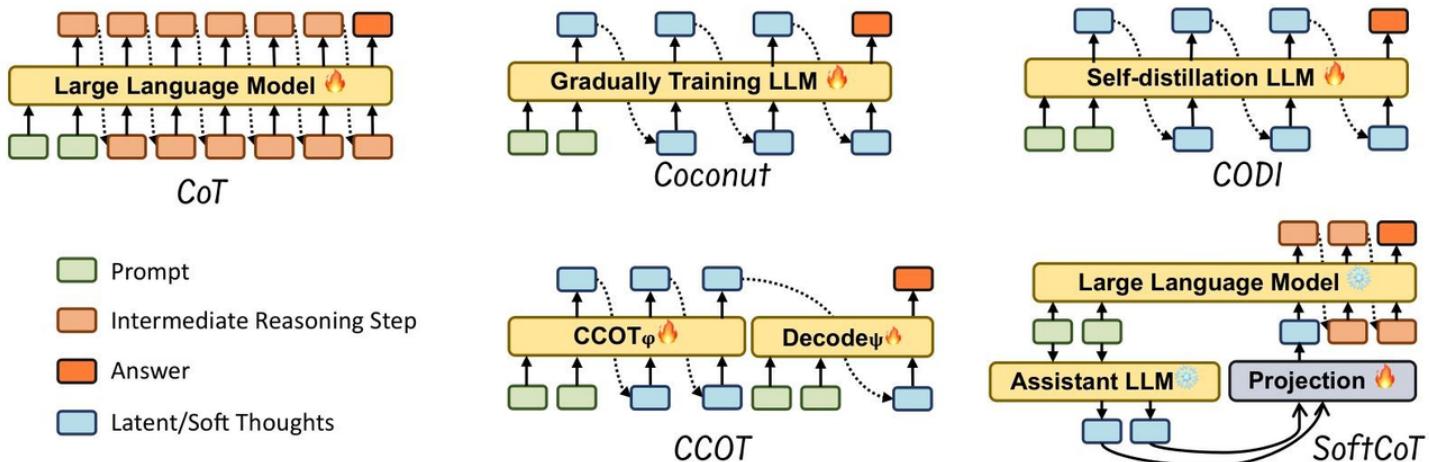
表3: 按“数据来源—压缩方式—是否在线/离线—SFT 方式—基座模型”汇总了一些 SFT 的典型做法

输出层面的方法

将显性推理步骤压缩到“潜在表示”中

近来的工作强调：提升推理力并不一定依赖长篇的人类可读 CoT。适当增加“思考用的计算”（哪怕是无语义的“占位思考”）也能带来收益，说明关键在于隐空间中的额外计算而非文本长度本身。由此催生了一类方法：将显性 CoT 替换或压缩为更紧凑的潜在表征，在保持或提升准确率的同时大幅减少输出 token 数。

那么，怎么将显式的推理步骤压缩到潜在空间呢？



如上图所示，方法大体可以分为两类：

- 直接在模型内部学习“潜在思维”：
 - **Coconut (Chain of Continuous Thought):** 把最后一层隐藏状态当作连续思维 token，循环回喂做下一步输入，等价于在隐空间多想几步，用少量连续向量替代长篇 CoT，最后再输出答案/短 CoT——用隐状态而非离散 token 展开思考，在减少显性思考长度的同时提升准确率与效率。
 - **CODI (Continuous Distillation of Implicit CoT):** 用自蒸馏同时学习显式与隐式 CoT，同一训练样本走两条路：一条写出 CoT（老师），一条不写 CoT（学生）。通过让两条路在若干位置的隐藏态对齐，学生学会把中间推理放到隐空间做（隐式 CoT），最终只需输出答案或很短的 CoT。
 - **CCOT (Compressed CoT with LoRA):** 先用强模型生成完整 CoT，从中挑出关键信息对应的位置作为“压缩目标”。训练两套 LoRA：CCoT 模块从问题直接预测少量压缩/沉思向量；DECODE 模块在“问题 + 这些压缩向量”的条件下解码出精简 CoT/答案。推理时：先预测压缩向量，再据此生成简洁推理。
 - **Heima (多模态) :** 把每个推理阶段的长文本说明替换为一个思考 token，并据此重写训练数据，实现高效的多模态隐式推理。
 - **Token Assorted:** 结合 VQ-VAE，将部分 CoT 切换成离散的潜在 token，与文本 token 混合训练，以紧凑的潜在形式表示某些推理步骤，兼顾效率与可解释性。
 - **Looped Transformer:** 对同一层重复应用多次（不增加参数），等价于更深的迭代推理，让模型在隐空间完成多步思考。
- 冻结主模型，增加轻量“隐思维”模块

- **SoftCoT**: 不改动主 LLM，单独训练一个小网络生成若干连续思维向量，再通过线性变换投影到主模型的嵌入维度，把它们当作前缀软 token 拼到输入前；主模型据此进行更多内部计算，而无需输出冗长 CoT。

以上方法能有效减少显性推理 token 数量并降低响应延迟，并且在潜在空间的操作具有灵活性和可拓展性，缺点则是降低了可解释性（难以逐步审计），并且通常需要额外训练或模块。

推理过程中动态推理

另一条思路是不改模型参数，而是在推理时动态调整思考策略：对容易题早停，难题多想；对差样本尽早丢弃，对好样本加算力。这类方法通常是无需额外训练的或只需极少附加模块。

两个问题：选什么标准来指导推理？什么范式更高效？

实践上有四条主线（互补可叠加）：

1. 测试时扩展（Test-Time Scaling）：在不改参数的前提下，通过搜索/采样扩展思考的广度或深度。
 - **Best-of-N**：对同一输入生成 N 条候选，用多数投票或奖励模型选优，能显著提升复杂任务表现。
 - **Beam Search**：把生成拆成多步，配过程奖励模型（PRM）逐步保留最优分支，细粒度提升中间质量。
 - **MCTS**：并行探索、回溯与回传累积得分，最终选最优分支，适用于需要回溯/分支搜索的任务。
2. 基于奖励的推理优化（在 TTS 基础上省算力）：
 - **Speculative Rejection**：BoN 过程中动态丢弃低分样本，仅保留高潜力序列，更省显存和时间。
 - **RSD (Reward-Guided Speculative Decoding)**：用草稿模型先跑，PRM 打分：高分直接接受，低分再交给大模型精炼，兼顾速度与准确。
3. 基于置信度的推理优化（难题多想、易题少想）：
 - **DPTS**：将 Tree-of-Thought 并行化，按置信度早剪枝，减少无效探索并动态平衡探索/利用。
 - **Certainindex / Dynasor-CoT**：用语义熵+PRM 分数度量“继续思考是否还会改变答案”，高置信就提前终止并把资源让给更难的查询。
 - **FastMCTS**：优先扩展高置信轨迹并按题目复杂度调节树扩展，效率与多样性兼顾。
 - **Length-filtered Vote**：理论上存在最优 CoT 长度；据此对不同长度分组投票，剔除过短/过长的答案以稳定正确率。
4. 基于一致性的推理优化：
 - **ST-BoN (Self-Truncation BoN)**：比较早期生成的隐向量一致性，不等全序列生成就提前淘汰不一致路径，降低评估成本。

补充：基于摘要的动态推理

- **LightThinker**: 学习何时&如何压缩中间思路，把冗长 CoT 汇成gist tokens，配稀疏注意力聚焦关键信息；属于“边想边压缩”的范式。
- **InftyThink**: 交替思考→摘要→丢旧思路，只保留最新摘要，突破上下文窗口而保持高精度。

小结：输出侧优化要么把思维“藏进”隐空间以缩短可见文本，要么在推理时按价值/置信/一致性来早停、剪枝或重采样。两类方法可叠加：先用潜在表征降长度，再配合动态范式把算力花在刀刃上。

输入层面的方法

提示词引导

思路是在prompt里直接约束思考的长度/形式，让模型在不牺牲正确性的前提下减少冗余步骤。典型做法与可复用模板如下（均可zero shot使用）：

- **Token-Budget**：通过设置token预算减少不必要的推理token，引入TALE-EP，首先通过提示LLM本身来估算合理的 token 预算，然后将预算嵌入到提示词中，引导模型生成更高效的回复。

代码块

```
1 Let's think step by step and use less than <Token-Budget> tokens.
```

- **CoD (Chain-of-Draft)**：鼓励“逐步思考”，但每一步只保留极简草稿（如≤5词），模仿人类只记要点的习惯：

代码块

```
1 Think step by step, but only keep a minimum draft for each thinking step, with at most five words.
```

- **CCoT**：直接在 CoT 提示中加入简洁性约束：`Think step by step, and be concise.`
- **Token Complexity**：使用硬约束模板（如 `BulletPoints` / `WordLimit(k)` / `StepLimit(k)` / `TokenLimit(k)` 等）限定表达形式或上限，研究显示各类压缩提示大体落在相似的“准确率-压缩率”曲线之上，说明每个任务存在内在 token 复杂度（解决该任务所需的最小 token 数）。这意味着当前提示仍远未到达理论最优压缩，有进一步优化空间。

推理路由

思路是根据题目难度/不确定性，把输入路由到快模型或强模型，或路由到不同的推理范式（直答 vs. 深思）。这能在总体算力固定时提升性价比。

如何判断“该走哪条路”？三类主流做法：

1. 模型内置模式切换

- **Claude 3.7 Sonnet**：业内早期的混合推理形态，支持快速答复与分步思考两种模式，复杂题留更多思考预算，简单题快速返回。

2. 训练一个路由器

- **RouteLLM**: 用 Chatbot Arena 偏好数据训练路由器，简单题→轻量模型，复杂题→强模型，在不牺牲准确率的前提下显著降时延与成本。
- **Sketch-of-Thought (SoT)** : 用轻量编码器（如 DistilBERT）先判题型，再在三种“思维风格”中选一种：

Conceptual Chaining (口头链式) / Chunked Symbolism (紧凑数学记号) / Expert Lexicons (领域速记)，以最短表达覆盖核心推理。

3. 不确定性驱动的自路由 (不一定要外部分类器)

- **Self-Ref**: 在 LLM 内部微调专用置信度 token，让模型自己估计是否不确定，仅对高不确定样本触发“更强/更长链”的路径。
- **Confident or Seek Stronger**: 构造校准数据，让模型在不读取用户输入内容的前提下，预测自身对即将产生输出的信心，用作路由信号（自信则直答，不自信则求助更强）。

其他讨论

如何用更少的数据训练推理模型？

- **最小但高影响力数据筛选**
 - **LIMO**: 仅用 817 个精心筛选的示例（按难度、通用性、多样性挑选问题，按最优结构组织、有效认知、验证质量挑选答案），即可超越此前 10 万+示例训练的模型。
 - **s1**: 构建了包含 1,000 个高质量问题-推理对的 s1K 数据集，并在推理阶段引入“预算强制”(budget forcing)，在推理过程中调节推理时间。
- **自我验证作为信号**
 - **S2R**: 先用小量示例做 SFT，让模型学会自我验证与自我纠错；再通过 RL 在过程层面与结果层面强化这一能力。

如何提升小模型 (SLM) 的推理能力？

1. **知识蒸馏 (Distillation)**
 - **混合蒸馏**: 结合长/短 CoT 示例或 CoT+程序化思路 (PoT)，提升蒸馏效果。
 - **反事实蒸馏**: 通过遮蔽问题中因果特征来扩充训练集，促使LLM完成被屏蔽的文本。为每个数据生成正负样本，从多视角蒸馏。
 - **反馈驱动蒸馏**: 从现有问题中创建多样和复杂的问题来迭代扩充蒸馏集，生成多样化、更复杂问题用于训练小模型。
 - **探测与检索蒸馏**: 训练两个互补小模型，一个“探测模型”检索相关知识，另一个“推理模型”据此生成推理链，分工协作提升推理质量。
 - **自适应思考蒸馏**: 让蒸馏过程中的小模型，基于任务复杂度动态调整推理策略，提升对长短不同问题的适应能力。

- **符号知识内化：**将符号化知识注入 SLM，帮助其在 CoT 推理中引入结构化、可解释的符号操作，提升效率与准确度。
- **自我纠错管道：**生成小模型的自我纠错数据，并据此微调，使其具备先推理→再校正的能力，有效减少逻辑漏洞。

2. 模型压缩

- **量化：**将权重降低到低精度，能在大幅减小模型参数量和加速推理的同时，几乎无损保留推理性能；
- **剪枝：**移除低重要性权重或神经元，会严重损害多步逻辑能力，显示出剪枝对推理链路的破坏性。

参考论文：《Stop Overthinking: A Survey on Efficient Reasoning for Large Language Models》