

# Chain of Draft

## 6.7.4 Chain of Draft

### 论文总结

想象你在做数学题时，只会记录简洁而高信息量的草稿，而不会为每一步写下的详细说明。Chain of Draft模仿人类解决问题时的思维，通过prompt鼓励LLM只生成简洁和高信息量的输出，只需要7.6%的token的情况下，准确率就能跟CoT相匹配甚至超越，显著了降低推理的延迟和成本。

参考论文：[Chain of Draft: Thinking Faster by Writing Less](#)

- **Standard（直接给答案）**：使用few-shot提示，要求大模型直接生成回复，缺少透明度，在解决复杂问题时，缺少推理步骤很容易产生幻觉。
- **CoT（面面俱到）**：提供few-shot，要求逐步思考，包含了很多不必要细节（Jason、Denny），对解决问题无用。
- **CoD（一针见血）**：使用few-shot，在prompt中要求每个推理步骤最多输出5个单词，这样大模型就只保留关键信息。

#### Standard

Answer the question directly. Do not return any preamble, explanation, or reasoning.

#### Chain-of-Thought

Think step by step to answer the following question. Return the answer at the end of the response after a separator #####.

#### Chain-of-Draft

Think step by step, but only keep a minimum draft for each thinking step, with 5 words at most. Return the answer at the end of the response after a separator #####.

#### Standard

Q: Jason had 20 lollipops. He gave Denny some lollipops. Now Jason has 12 lollipops. How many lollipops did Jason give to Denny?  
A: 8

#### Chain-of-Thought

Q: Jason had 20 lollipops. He gave Denny some lollipops. Now Jason has 12 lollipops. How many lollipops did Jason give to Denny?  
A: Let's think through this step by step:  
1. Initially, Jason had 20 lollipops.  
2. After giving some to Denny, Jason now has 12 lollipops.  
3. To find out how many lollipops Jason gave to Denny, we need to calculate the difference between the initial number of lollipops and the remaining number.  
4. We can set up a simple subtraction problem: Initial number of lollipops - Remaining number of lollipops = Lollipops given to Denny  
5. Putting in the numbers:  $20 - 12 = \text{Lollipops given to Denny}$   
6. Solving the subtraction:  $20 - 12 = 8$   
Therefore, Jason gave 8 lollipops to Denny.  
####  
8 lollipops

#### Chain-of-Draft

Q: Jason had 20 lollipops. He gave Denny some lollipops. Now Jason has 12 lollipops. How many lollipops did Jason give to Denny?  
A:  $20 - x = 12$ ;  $x = 20 - 12 = 8$ . #### 8



作者论文中提到，CoD中没有强制大模型推理步骤只能生成5个token，只是通过prompt以及样例引导模型这样做。

- 实验结果：
  - 是否使用few-shot

Model	Prompt	Accuracy	Token #	Latency
GPT-4o	Standard	53.3%	1.1	0.6 s
	CoT	95.4%	205.1	4.2 s
	CoD	91.1%	43.9	1.0 s
Claude 3.5 Sonnet	Standard	64.6%	1.1	0.9 s
	CoT	95.8%	190.0	3.1 s
	CoD	91.4%	39.8	1.6 s

使用few-shot，COD相比于CoT准确率有一定损失，加速效果明显，生成token量显著降低。此结论在算数、常识推理、符号推理任务上都成立。

Model	Prompt	Accuracy	Token #	Latency
GPT-4o	Standard	56.9%	2.2	0.5 s
	CoT	94.8%	278.4	8.1 s
	CoD	84.4%	76.4	2.6 s
Claude 3.5 Sonnet	Standard	61.9%	5.2	0.9 s
	CoT	90.4%	248.8	3.5 s
	CoD	65.5%	73.7	1.6 s

不使用few-shot，CoD准确率大幅下降，加速效果也减弱，原因在于训练数据缺少CoD风格的推理样本，在缺少few-shot样例就很难生成高质量的draft。

- 在小模型上的表现：小模型上CoD的表现显著降低，也是因为训练数据中缺少CoD样本，再加上小模型表达能力较弱，难以兼顾简洁和信息完整。如果用CoD数据对模型进行微调，可以提升性能。

Model	Prompt	Accuracy	Token #
Qwen2.5-1.5B-Instruct	Standard	5.7%	6.6
	CoT	32.5%	141.4
	CoD	24.2%	75.1
Qwen2.5-3B-Instruct	Standard	7.2%	3.4
	CoT	59.1%	236.4
	CoD	43.1%	41.2
Llama3.2-3B-Instruct	Standard	3.9%	16.6
	CoT	70.7%	195.3
	CoD	52.5%	98.1
Zoom-SLM-2.3B	Standard	5.9%	3.8
	CoT	77.7%	129.0
	CoD	50.9%	55.6



- 关于这篇论文的一些思考：

第一次关注到这篇论文，是之前参加蚂蚁比赛的时候，目标是压缩推理链，它提供的baseline方法就是CoD，实测这种方法在比较简单的任务上（比如GSM8K），效果还是不错的，因为推理本身很简单，即使每一步都压缩到几个token，也能实现较好的推理效果。但一旦任务难度提升，特别是深入到一些具体领域时，几个token连一些关键概念的意义都解释不清楚，更遑论推理出正确的结果了。特别是在使用小模型时，不做专门的训练而是只寄希望于调prompt，很难实现简短而高质量的推理。

之所以要介绍这篇论文，一是大家项目中如果涉及推理加速或者面试官问道如何加速推理，可以提到这篇论文的做法，并且CoD的方法也可以与其他推理加速的方法一起使用。二是学习这篇论文的作者是如何把调prompt这件简单的事，写出这么多内容的，特别是对于项目缺乏深度的同学，调prompt做消融实验可以作为你工作的很大一部分。

- **使用CoD可能面临的问题**

1. **可解释性下降**：用户难以通过浓缩的推理步骤理解推理过程，CoD更适合于实时性强、且不需要用户深入理解过程的场景。
2. **few-shot设计偏差**：针对一个场景，few-shot样例应该尽可能覆盖场景下所有可能的问題，否则由于模型没有进行过专门的训练，prompt里也没有合适的参考，很容易回答质量下降。
3. **缺乏自我验证**：研究过大模型CoT内容的同学可能会发现，大模型推理的最后往往会进行自我验证，生成类似“让我验证下我说的有没有问题”。自我验证不一定是好事，有时候大模型会陷入不断自我否定的死循环中，但验证也确实能发现一些之前推理过程中的错误，我印象比较深刻的是25年年初时，大模型训练语料还比较缺GRPO相关的内容，我问大模型GRPO相关的问题，它把GRPO理解成了另一个非强化学习的算法分析了一堆，最后验证的时候根据之前聊天对话中的强化学习内容才反应过来，GRPO应该也是个强化学习的算法，推翻了之前的推理并生成了正确的内容。