提示工程（Prompt Engineering）

原文地址：Prompt Engineering 本文为其中文（修改）翻译版本

关键概念：

提示工程（Prompt Engineering） 上下文提示（In-Context Prompting） 对齐（Alignment）
零样本提示（Zero-shot Prompting） 少样本提示（Few-shot Prompting）
指令提示（Instruction Prompting） 自一致采样（Self-Consistency Sampling）
思维链提示（Chain-of-Thought Prompting, CoT） 自动提示设计（Automatic Prompt Design）
检索生成增强（Retrieval-Augmented Generation, RAG） 外部工具调用（External Tool Use）

内容目录：

提示工程（Prompt Engineering），亦称为 上下文提示（In-Context Prompting），指的是在不更新大语言模型（LLM）权重的前提下，通过设计输入内容来引导模型生成特定行为的技术。这是一种**无需微调**即可影响模型输出的交互方式。

提示工程本质上是一门**经验科学**，其有效性因模型架构和训练数据的不同而显著变化。因此，实际应用中往往依赖大量试验与启发式探索。本文将重点关注**自回归语言模型（autoregressive language models）**上的提示工程方法，不涉及完形填空（cloze-style tasks）、图像生成或多模态模型等其他场景。

从根本目的上看，提示工程追求的核心是：**对齐（Alignment）**与 **可控性（Steerability）**。

> 💬 **个人观点**：坦率地说，我认为一些关于提示工程的论文并不值得写满 8 页——不少"技巧"其实一句话就能讲清楚，其余篇幅往往只是围绕基准测试展开。与其写满篇幅，不如构建一个**可复用、便于社区共享的基准测试框架**，这可能更具实际价值。比如，迭代式提示（iterative prompting）和 外部工具调用（external tool use）虽然思路不错，也具有实用潜力，但其配置复杂、难以标准化，更难的是推动整个社区达成共识并共同采纳。

基础提示方式（Basic Prompting）

零样本提示（Zero-shot Prompting） 与 少样本提示（Few-shot Prompting） 是最基础的提示工程方式，由多篇开创性 LLM 论文提出，并广泛应用于大语言模型的性能评估。

零样本提示（Zero-shot Prompting）

Zero-shot 提示是指**直接提供任务指令文本，让模型给出答案**，过程中不包含任何示例样本。

```
Text: i'll bet the video game is a lot more fun than the film.
Sentiment:
```

少样本提示（Few-shot Prompting）

Few-shot 提示在输入中提供若干示例（每个包含输入及期望输出），使模型更好地理解任务意图和输出风格。通常比 zero-shot 表现更优，但也带来更高的 token 消耗，尤其在长文本任务中容易触达上下文长度限制。

```
Text: (lawrence bounces) all over the stage, dancing, running, sweating, mopping his face and generally displayi
ng the wacky talent that brought him fame in the first place.
Sentiment: positive

Text: despite all evidence to the contrary, this clunker has somehow managed to pose as an actual feature movie,
the kind that charges full admission and gets hyped on tv and purports to amuse small children and ostensible ad
ults.
Sentiment: negative

Text: for the first time in years, de niro digs deep emotionally, perhaps because he's been stirred by the power
ful work of his co-stars.
Sentiment: positive

Text: i'll bet the video game is a lot more fun than the film.
Sentiment:
```

许多研究关注**如何构造高质量示例上下文**以提升模型性能。已有观察表明：**提示格式、示例选择、顺序等细节**会导致性能在“近似随机猜测”与“接近 SOTA”之间剧烈波动。

Zhao et al. (2021) 针对 few-shot 分类任务进行了系统性的分析，指出大语言模型（如 GPT-3）中的性能波动主要来自以下偏差：

1. **主标签偏差（Majority Label Bias）**：当上下文中的示例标签分布不均时，模型更倾向预测多数类别；
2. **近因偏差（Recency Bias）**：模型倾向于复用最后一个示例的标签；
3. **常见词偏差（Common Token Bias）**：模型偏好生成更常见的词汇。

为缓解上述偏差，作者提出使用**校准（Calibration）**方法：使用无意义的输入（如 `N/A`）来重设标签分布，使其输出更接近均匀分布。

🎯 示例选择技巧（Tips for Example Selection）

- **语义相似示例**：选择与测试样本语义相近的样本，常用 *k*-近邻方法在嵌入空间中进行筛选 (Liu et al., 2021)。
- **多样性选择**：采用图算法优化选择结果多样性 (Su et al., 2022)：(1) 首先，基于样本之间的嵌入（如 SBERT 或者其他 embedding models）余弦相似度，构建一个有向图 $G = (V, E)$，每个节点指向其最近的 k 个邻居；(2) 接着，维护两个集合：已选样本集 $\mathcal{L} = \phi$ 和剩余样本集 $\mathcal{U}$。对每个样本 $u \in \mathcal{U}$，计算打分函数：

$$\text{score}(u) = \sum_{v \in \{v \mid (u,v) \in E, v \in \mathcal{U}\}} s(v), \text{ where } s(v) = \rho^{-|\{\ell \in \mathcal{L} \mid (v,\ell) \in E\}|}, \rho > 1,$$

  如果 u 的邻居中有很多已被选择样本，那么其得分就会高，从而被降低被选中的概率。这个评分机制鼓励选出多样化的样本。
- **对比学习选择**：Rubin et al. (2022) 提出使用对比学习（contrastive learning）来针对某个训练数据集学习专用的嵌入，用于选择 in-context learning 示例。对于每个训练样本对(x, y)，可以通过语言模型对格式化的输入输出对 e_i 分配的条件概率 $\text{score}(e_i) = P_{LM}(y|e_i, x)$ 来评估其质量。选出条件概率得分最高和最低的示例，分别作为正负样本对用于对比学习。
- **强化学习选择**：(Zhang et al. 2022) 尝试使用 Q-learning 强化学习方法进行示例选择。
- **不确定性驱动**：受基于不确定性的主动学习方法启发，选择在多次采样中模型输出分歧较大的样本用于 few-shot 提示 (Diao et al., 2023)。

📏 **示例排序技巧（Tips for Example Ordering）**

- 一个通用建议是：**选择的示例应当多样化、与测试样本相关，并采用随机顺序**，以避免 **主标签偏差（majority label bias）** 和 **新近性偏差（recency bias）**。
- 此外，**增大模型规模或增加训练样本数量并不能减少不同示例排列所引起的性能波动**。相同的示例顺序在某个模型上可能效果很好，但在另一个模型上可能表现很差。当验证集规模有限时，建议选择一种示例排列方式，使得模型的预测不会出现**极端的类别不平衡**或**过度自信**的情况。 (Lu et al., 2022)。

指令提示（Instruction Prompting）

在 Few-shot 提示中，提供示例的目的是帮助模型理解我们的意图——本质上，是通过演示来“传达”任务指令。然而，Few-shot 方法存在明显局限：它在 token 消耗上代价较高，且受限于上下文长度。那么问题来了：**为何不直接给模型下达任务指令？**这正是指令提示的核心理念。指令微调语言模型（Instructed LM）（如 InstructGPT, natural instruction）会使用高质量的三元组（任务指令、输入、期望输出）来微调预训练模型，从而使模型更好地理解用户意图，并遵循指令。常用的方法是基于人类反馈的强化学习（RLHF）。指令微调的最大优势在于：**显著提高了模型对人类意图的对齐程度，降低了人机交互的成本**。

与指令模型交互时，应当明确具体地描述任务要求，尽量避免“不要做某事”的表述，而是要直接指定 “应该做什么”。

```
Please label the sentiment towards the movie of the given movie review. The sentiment label should be "positive"
or "negative".
Text: i'll bet the video game is a lot more fun than the film.
Sentiment:
```

通过设定**目标受众**，可以进一步引导模型生成更符合预期风格的内容：

- 适用于儿童的教育材料：

```
Describe what is quantum physics to a 6-year-old.
```

- 确保输出内容符合安全规范：

```
... in language that is safe for work.
```

上下文指令学习（In-context Instruction Learning）(Ye et al. 2023) 提出将 few-shot 学习与指令提示结合起来。他们在提示中引入了多个不同任务的示例，每个示例都包含任务定义、输入和输出。他们的研究主要针对分类任务，要求提示中明确列出所有可能的标签类别。

```
Definition: Determine the speaker of the dialogue, "agent" or "customer".
Input: I have successfully booked your tickets.
Ouput: agent

Definition: Determine which category the question asks for, "Quantity" or "Location".
Input: What's the oldest building in US?
Ouput: Location

Definition: Classify the sentiment of the given movie review, "positive" or "negative".
Input: i'll bet the video game is a lot more fun than the film.
Output:
```

自一致采样（Self-Consistency Sampling）

Wang et al. 2022a 提出自一致采样（Self-consistency sampling）：使用温度参数 $T > 0$的采样生成多个输出，然后从中选出最佳结果。

“最佳答案”的判定标准依任务而异。最常用的是**多数投票（majority voting）**，而在可验证性较强的任务（如编程题）中，可以通过执行代码并运行测试用例来判断哪个输出是正确的。

思维链提示（Chain-of-Thought Prompting, CoT）

Wei et al. 2022 提出的思维链提示（Chain-of-Thought Prompting），其核心思想是引导模型逐步生成一系列中间推理步骤，最终得出答案。这种方式在**复杂推理任务**（如数学题、多跳问答）中表现尤为出色，尤其是在参数规模较大的模型（50B 以上）中优势明显。而对于较简单任务，思维链的性能提升则相对有限。

两类 CoT 提示方式：Few-shot / Zero-shot CoT

1. **Few-shot CoT：** 向模型提供包含完整推理链的高质量示例（可由人工或模型生成）。

```
Question: Tom and Elizabeth have a competition to climb a hill. Elizabeth takes 30 minutes to climb the hill. Tom takes four times as long as Elizabeth does to climb the hill. How many hours does it take Tom to climb up the hill?
Answer: It takes Tom 30*4 = <<30*4=120>>120 minutes to climb the hill.
It takes Tom 120/60 = <<120/60=2>>2 hours to climb the hill.
So the answer is 2.
===
Question: Jack is a soccer player. He needs to buy two pairs of socks and a pair of soccer shoes. Each pair of socks cost $9.50, and the shoes cost $92. Jack has $40. How much more money does Jack need?
Answer: The total cost of two pairs of socks is $9.50 x 2 = $<<9.5*2=19>>19.
The total cost of the socks and the shoes is $19 + $92 = $<<19+92=111>>111.
Jack need $111 - $40 = $<<111-40=71>>71 more.
So the answer is 71.
===
Question: Marty has 100 centimeters of ribbon that he must cut into 4 equal parts. Each of the cut parts must be divided into 5 equal parts. How long will each final cut be?
Answer:
```

2. **Zero-shot CoT：** 通过自然语言提示模型进行推理，例如添加引导语：
   - `"Let's think step by step."`（Kojima et al. 2022）

     ```
     Question: Marty has 100 centimeters of ribbon that he must cut into 4 equal parts. Each of the cut parts must be divided into 5 equal parts. How long will each final cut be?
     Answer: Let's think step by step.
     ```

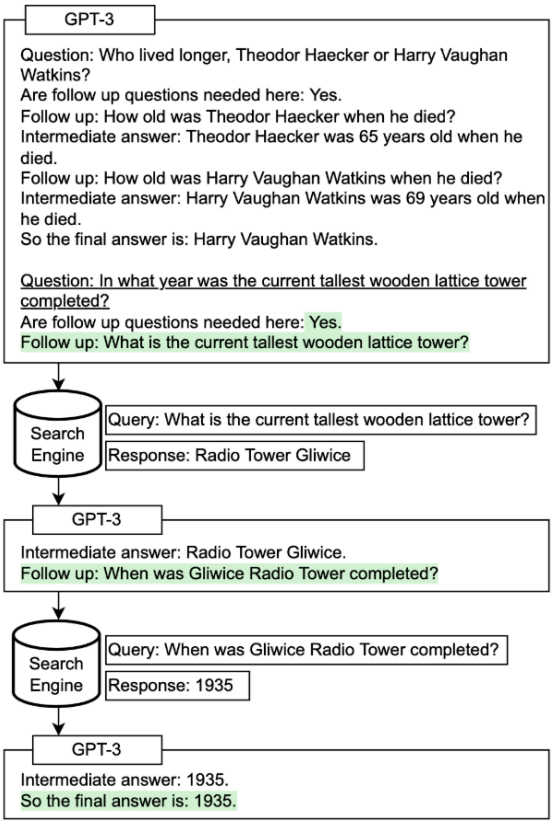
   - `"Let's work this out in a step-by-step way to be sure we have the right answer."`（Zhou et al. 2022）

   这些语句能显著提高模型生成推理链的概率和质量。

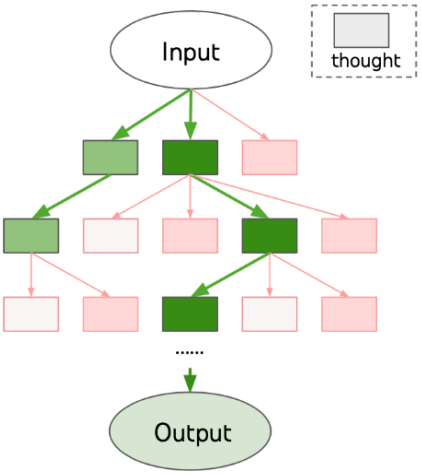
技巧与扩展（Tips and Extensions）

- **Self-consistency sampling**：对同一个提示多次采样，使用多数投票提升准确率（Wang et al. 2022a）。
- **CoT 示例集成策略**：通过打乱示例顺序、替换推理链引入随机性，进行多轮预测后再投票合并输出（Wang et al. 2022b）。

- **STaR（Self-Taught Reasoner）方法**：若训练数据只有正确答案没有推理链，可采用 STaR（Zelikman et al. 2022）方法：（1）让模型生成推理链，只保留能导出正确答案的；（2）用这些推理链微调模型，重复直到收敛。注意：高温度更可能生成错误推理但正确答案的情况。如果训练集中没有正确答案，可考虑用多数投票结果作为"正确答案"。
- **示例复杂性影响性能**：示例推理链越复杂、步骤越多，模型性能通常越好。换行 `\n` 比 `step i.`、`;` 更适合分隔推理步骤（Fu et al. 2023）。
- **复杂度一致性（Complexity-based Consistency）**：仅选用最复杂的若干条推理链参与多数投票，有助于提升准确率（Fu et al. 2023）。
- **复杂 vs 简单示例的影响差异**：使用复杂示例对复杂问题效果好，但可能削弱模型在简单任务上的表现（Shum et al. 2023）。
- **格式微调技巧**：将 `Q:` 改为 `Question:` 可以在多个任务中显著提升表现（Fu et al. 2023）。
- **解释质量问题**：在文本推理任务中（如 QA 和 NLI），模型生成的解释往往包含事实性错误而非逻辑错误。错误解释往往是预测出错的根源（Ye & Durrett, 2022）。
- **多轮推理与外部检索的组合：** Press et al. 2022 提出 Self-Ask 方法，模型可反复提出子问题构建思考过程。子问题也可以借助搜索引擎获取答案。类似方法还有：交错检索思维链（Interleaving Retrieval CoT, IRCoT）（Trivedi et al. 2022）与 ReAct（Reason + Act）（Yao et al. 2023）。它们都结合了检索思维链与对 Wikipedia 等外部 API 的检索结果，对应内容添加回上下文使用。一个示意图（Press et al. 2022）如下：



- Tree of Thoughts （ToT）（Yao et al. 2023）将 CoT 扩展为搜索树结构：（1）每个推理阶段生成多个"思维分支"；（2）通过 BFS 或 DFS 搜索完整路径；（3）使用分类器或多数投票评估每个状态，选出最佳答案。ToT 能在数学、逻辑和策略推理任务中显著提升表现。



(d) Tree of Thoughts (ToT)

自动提示设计（Automatic Prompt Design）

在大语言模型中，Prompt（提示词）是一组作为输入前缀的 token，其目的是在给定输入的条件下提高生成目标输出的概率。因此，**提示词本质上可以被视为可优化的参数**，并通过梯度下降在嵌入空间中直接训练。

典型方法包括：AutoPrompt（Shin et al., 2020）；Prefix-Tuning（Li & Liang, 2021）；P-tuning（Liu et al., 2021）；Prompt-Tuning（Lester et al., 2021）。这些方法的发展趋势是：提示结构越来越简洁，训练方式更高效，适配更大模型。

APE：自动提示工程师（Automatic Prompt Engineer）Zhou et al. (2022)，是一种在候选指令集合中进行搜索并筛选评分最高提示的策略。其流程如下：

- **生成候选指令**：利用语言模型生成指令候选，这些候选是基于一小组输入-输出对形式的示例。例如：`{{Given desired input-output pairs}}\n\nThe instruction is`。
- **目标函数定义**：给定一个数据集 $\mathcal{D}_{\text{train}} = \{(x, y)\}$, 希望找到一个指令 ρ 使得 $\rho^* = \arg\max_\rho \mathbb{E}_{(x,y)\in\mathcal{D}_{\text{train}}}[f(\rho, x, y]$, 其中 $f(\cdot)$ 是样本级评分函数，如准确率 $\mathbb{1}[\text{LM}(\cdot|\rho, x) = y]$ 或者概率得分 $p_{\text{LM}}(y|\rho, x)$。
- **迭代搜索优化**：使用**蒙特卡洛搜索**迭代改进候选指令，并通过如下提示生成语义等价的变体：`Generate a variation of the following instruction while keeping the semantic meaning.\n\nInput: ...\n\nOutput:...`

为自动生成 CoT 提示，Shum et al. (2023) 提出了三阶段流程：

- **增强（Augment）**：使用 Few-shot 或 Zero-shot CoT 提示生成大量伪推理链；
- **剪枝（Prune）**：剔除那些生成答案与真实标签不一致的链条；
- **选择（Select）**：采用策略梯度方法，将样本分布视为策略，并以验证集准确率作为奖励信号进行优化。

Zhang et al. (2023) 提出了聚类引导的 CoT 示例选择（Clustering for CoT Sampling），然后生成推理链。他们观察到大语言模型倾向于重复某些类型的错误，而这些错误类型在嵌入空间中往往可以被聚类。为避免样本分布偏倚，他们提出了如下构建方法：

- **问题聚类（Question Clustering）**：将问题编码为嵌入表示，应用 k-均值聚类以识别常见错误类型或主题；
- **示例选择（Demonstration Selection）**：从每个聚类中选择代表性的问题，即选择离聚类中心最近的样本；
- **推理链生成（Rationale Generation）**：利用 Zero-shot CoT 为这些代表性问题生成推理过程，构建具有多样性和覆盖性的 Few-shot 提示。

增强型语言模型（Augmented Language Models）

Mialon et al. (2023) 对 增强语言模型（Augmented Language Models） 进行了系统综述，涵盖了多种具备推理能力和外部工具使用能力的模型范式，推荐阅读。

检索增强（Retrieval-Augmented Generation）

对于需要**访问模型预训练截止时间之后的知识**，或**集成私有/内部知识库**的任务，如果不通过 prompt 显式提供上下文，语言模型将难以做出准确回答。许多开放域问答 Open Domain Question Answering 方法采用**"检索-生成两阶段架构"**：先在知识库中检索相关文段，再将其作为上下文输入语言模型。

Lazaridou et al. (2022) 研究了如何使用 Google 搜索进行文档检索以增强语言模型。给定一个问题 q，（1）从 Google 返回的 20 个 URL 中提取干净文本，形成一个文档集合。（2）由于这些文档很长，每个文档会被分割成长度为 6 个句子的段落 $\{p\}$。（3）基于 TF-IDF 计算每个段落与问题 q 的相似度，保留得分最高的段落作为 prompt 上下文；（4）最终大模型只将最相关的段落用于 prompt 以生成答案 a。

对于闭卷问答，每个示例的格式如下以构建 few-shot prompt。

```
Evidence: ...
Question: ...
Answer: ...
```

实验发现，将 prompt 中的问题与 evidence 顺序调换（即延长问题与答案之间的 token 距离）会在所有数据集上显著降低模型性能。

答案评分有三种方式：

- RAG （Retrieval-Augmented Generation ）形式，即$p(a_i \mid q) = \sum_{i=1}^{n} p_{\text{tf-idf}}(p_i, q) \cdot p_{\text{LM}}(a_i|q, p_i)$, 其中 $p_{\text{tf-idf}}(p_i, q)$ 是 基于TF-IDF 计算的段落与问题的相似度。；
- 噪声通道推理（Noisy channel inference）：$p(a_i|q) = \frac{p_{\text{LM}}(q|a_i,p_i) \cdot p_{\text{LM}}(a_i|p_i)}{p_{\text{LM}}(q|p_i)}$
- 专家乘积（Product-of-Experts, PoE）：结合所有上述概率以及 $p_{\text{LM}}(p_i|q)$

在生成和分类任务中的实验表明，重排序得分 PoE > Noisy channel inference > RAG；在各个概率项中，$p_{\text{LM}}(a_i|q, p_i)$ 和 $p_{\text{LM}}(q|a_i, p_i)$ 最具信息性。 $p_{\text{LM}}(q|a_i, p_i)$ 衡量在已知 evidence 和答案的条件下，模型对问题的解释能力，并且在答案重排序中非常可靠。

在 SituatedQA 数据集上观察到：即使允许语言模型访问互联网，对于 2020 年之后的问题，其表现仍明显落后于 2020 年前的问题，说明模型"内在知识"与检索信息之间可能存在冲突。

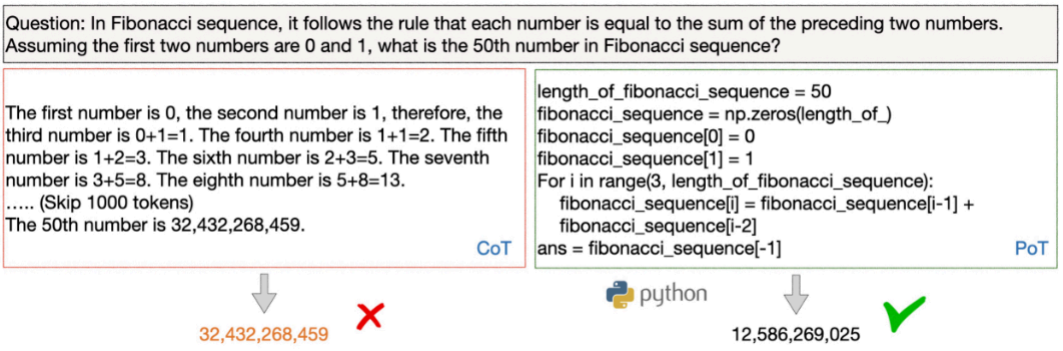
此外，即使不使用外部信息，仅通过**"自我检索"**，也能显著提升表现（Liu et al. 2022）。可使用如下 prompt 模板生成知识：

```
Generate some knowledge about the input. Examples:

Input: What type of water formation is formed by clouds?
Knowledge: Clouds are made of water vapor.

Input: {question}
Knowledge:
```

然后结合模型生成的知识，再进一步提示语言模型以获得答案。

编程辅助（Program-Aided Language Models）

PAL（Program-Aided LM）（Gao et al., 2022）和 PoT（Program of Thoughts Prompting）（Chen et al., 2022）是两种依赖程序执行的推理方法。核心思想是：语言模型生成代码（如 Python）来解决复杂任务，通过调用外部解释器完成逻辑计算与推导。这种设置有效解耦了语言建模与计算复杂度，提高了准确性和可控性。下面是 PoT 的示意图：



外部 API (External APIs)

在增强语言模型（Augmented Language Models）中，调用外部 API 是提升模型能力的关键手段之一。它允许模型通过工具来弥补其推理、计算或事实性知识的不足。

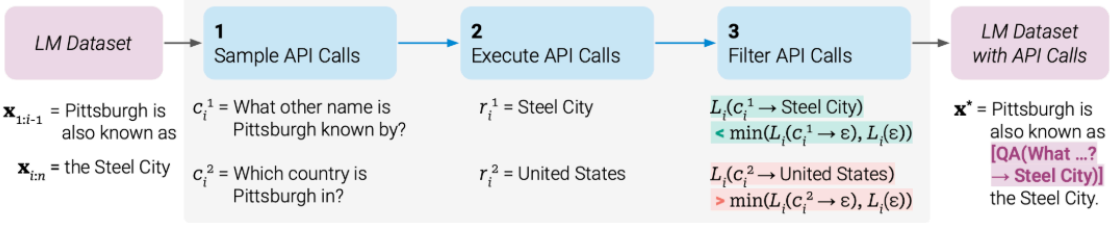
TALM（Tool-Augmented Language Models）（Parisi et al. 2022）通过文本形式实现与外部工具的交互。模型会在输出中自动插入 `|tool-call` 来发起 API 调用，随后附上调用输入文本。当生成 `|result` 标记时，实际调用指定 API，并将返回结果插入到上下文中，模型再在 `|output` 标记后继续生成最终答案。

An abstract task:
task input text *|tool-call* tool input text *|result* tool output text *|output* task output text

A weather task:
how hot will it get in NYC today? *|weather* lookup region=NYC *|result* precipitation chance: 10, high temp: 20c, low-temp: 12c *|output* today's high will be 20C

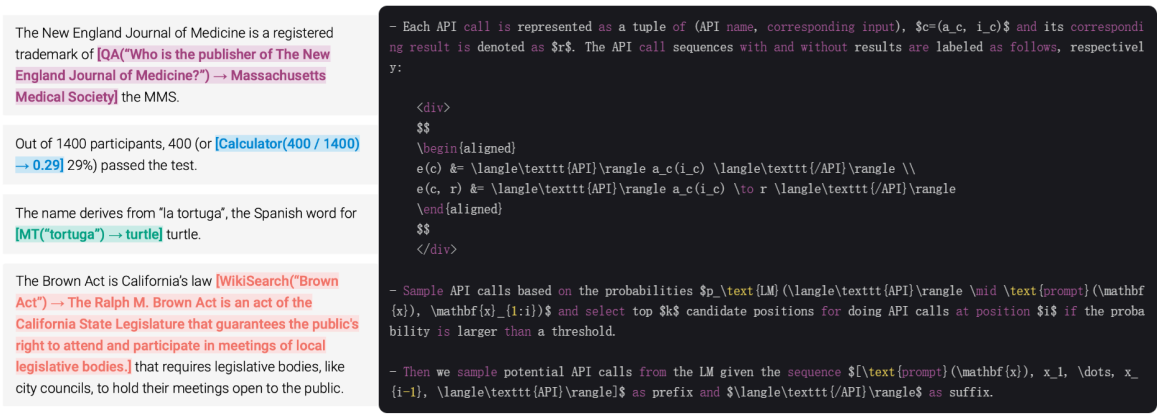
ALM 的训练采用一种**自博弈（self-play）机制**，即模型与 API 工具交互，在不断尝试中生成数据来微调自身。只有那些能显著提升模型输出质量的工具使用示例才被纳入数据集。这一过程类似强化学习：语言模型作为策略网络（policy network），通过策略梯度（policy gradient）方法进行优化。

Toolformer（Schick et al. 2023）是一种可通过 API 使用外部工具的语言模型。它无需大量人工标注，仅基于少量示例就能自监督学习调用工具。Toolformer 内置的工具包括：（1）🧮 **计算器**：用于精确数学计算。（2）❓ **问答系统**：提升事实准确率，减少幻觉。（3）🔍 **搜索引擎**：获取最新知识。（4）🌐 **翻译系统**：改善低资源语言表现。（5）📆 **日历系统**：理解时间语境。



Toolformer 的训练流程如下：

1. **标注潜在 API 调用点**：使用一个预训练语言模型，通过 few-shot 学习，在数据集上进行 API 调用使用示例的标注。示例格式如下：



2. **筛选有效的调用点**：根据 API 调用是否能帮助模型预测后续 token，通过自监督损失函数判断调用是否有效果。
    1. **执行 API 调用**：运行每个生成的 API 调用 c_i，获取对应结果 r_i。
    2. **计算加权交叉熵损失**：分别计算加入 API 结果与不加入 API 结果时（空序列ϵ）模型在 tokens $x_i, \dots, x_n$ 上的交叉熵损失。$L_i^+ = L_i(e(c_i, r_i))$，$L_i^- = \min(L_i(\epsilon), L_i(e(c_i, \epsilon)))$
    3. **筛选有效调用**：只保留带来预测性能提升的 API 调用（损失差值 $L_i^- - L_i^+$ 超过阈值），说明调用和结果确实对模型有帮助。
3. 在新标注的数据集上**微调语言模型**。新训练序列 $\mathbf{x}^* = x_{1:i-1}, e(c_i, r_i), x_{i:n}$ 由原始数据（如 CCNet 子集）与其增强版本构成。

推理阶段：在推理过程中，模型持续生成 token，直到遇到 `->`，表示它期望获得 API 响应。此时系统会自动发起调用，并将结果插入到序列中供模型继续生成答案。

Toolformer 目前**不支持工具链式调用**（即使用一个工具的输出作为另一个工具的输入），也**不支持交互式调用**（即根据人类选择采用 API 响应）。这两项能力是未来扩展该模型的有趣方向。

References

[1] Zhao et al. "Calibrate Before Use: Improving Few-shot Performance of Language Models." ICML 2021

[2] Liu et al. "What Makes Good In-Context Examples for GPT-3?" arXiv preprint arXiv:2101.06804 (2021).

[3] Lu et al. "Fantastically Ordered Prompts and Where to Find Them: Overcoming Few-Shot Prompt Order Sensitivity." ACL 2022

[4] Ye et al. "In-Context Instruction Learning." arXiv preprint arXiv:2302.14691 (2023).

[5] Su et al. "Selective annotation makes language models better few-shot learners." arXiv preprint arXiv:2209.01975 (2022).

[6] Rubin et al. "Learning to retrieve prompts for in-context learning." NAACL-HLT 2022

[7] Wei et al. "Chain of thought prompting elicits reasoning in large language models." NeurIPS 2022

[8] Wang et al. "Self-Consistency Improves Chain of Thought Reasoning in Language Models." ICLR 2023.

[9] Diao et al. "Active Prompting with Chain-of-Thought for Large Language Models." arXiv preprint arXiv:2302.12246 (2023).

[10] Zelikman et al. "STaR: Bootstrapping Reasoning With Reasoning." arXiv preprint arXiv:2203.14465 (2022).

[11] Ye & Durrett. "The unreliability of explanations in few-shot in-context learning." arXiv preprint arXiv:2205.03401 (2022).

[12] Trivedi et al. "Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions." arXiv preprint arXiv:2212.10509 (2022).

[13] Press et al. "Measuring and narrowing the compositionality gap in language models." arXiv preprint arXiv:2210.03350 (2022).

[14] Yao et al. "ReAct: Synergizing reasoning and acting in language models." ICLR 2023.

[15] Fu et al. "Complexity-based prompting for multi-step reasoning." arXiv preprint arXiv:2210.00720 (2022).

[16] Wang et al. "Rationale-augmented ensembles in language models." arXiv preprint arXiv:2207.00747 (2022).

[17] Zhang et al. "Automatic chain of thought prompting in large language models." arXiv preprint arXiv:2210.03493 (2022).

[18] Shum et al. "Automatic Prompt Augmentation and Selection with Chain-of-Thought from Labeled Data." arXiv preprint arXiv:2302.12822 (2023).

[19] Zhou et al. "Large Language Models Are Human-Level Prompt Engineers." ICLR 2023.

[20] Lazaridou et al. "Internet augmented language models through few-shot prompting for open-domain question answering." arXiv preprint arXiv:2203.05115 (2022).

[21] Chen et al. "Program of Thoughts Prompting: Disentangling Computation from Reasoning for Numerical Reasoning Tasks." arXiv preprint arXiv:2211.12588 (2022).

[22] Gao et al. "PAL: Program-aided language models." arXiv preprint arXiv:2211.10435 (2022).

[23] Parisi et al. "TALM: Tool Augmented Language Models" arXiv preprint arXiv:2205.12255 (2022).

[24] Schick et al. "Toolformer: Language Models Can Teach Themselves to Use Tools." arXiv preprint arXiv:2302.04761 (2023).

[25] Mialon et al. "Augmented Language Models: a Survey" arXiv preprint arXiv:2302.07842 (2023).

[26] Yao et al. "Tree of Thoughts: Deliberate Problem Solving with Large Language Models." arXiv preprint arXiv:2305.10601 (2023).