工具LLM：促进大型语言模型掌握16000多个真实世界API

秦宇佳[1*], 梁世豪[1*], 叶一宁[1], 朱坤伦[1], 严岚[1], 卢雅熙[1], 林彦凯[3†], 丛鑫[1], 唐湘如[4], 钱彬[4], 赵思涵[1], 洪蕾伦[1], 田润楚[1], 谢若冰[5], 周杰[5], 马克・格尔斯坦[4], 李大海[2,6], 刘志远[1†], 孙茂松[1† 1]清华大学 [2]模型最佳公司 [3]中国人民大学

[4]耶鲁大学 [5]微信AI，腾讯公司 [6]知乎公司
yujiaqin16@gmail.com

摘要

尽管开源大型语言模型（LLMs）如LLaMA取得了进展，它们在工具使用能力方面仍然受到显著限制，即使用外部工具（API）来满足人类指令。 原因在于当前指令调整主要集中在基本语言任务上，但忽略了工具使用领域。
这与最先进的封闭源LLMs（如ChatGPT）的出色工具使用能力形成对比。为了弥合这一差距，我们引入了ToolLLM，一个涵盖数据构建、模型训练和评估的通用工具使用框架。 我们首先提出ToolBench，一个用于工具使用的指令调整数据集，该数据集是使用ChatGPT自动构建的。具体而言，构建可以分为三个阶段：（i）API收集：我们从RapidAPI Hub收集了16,464个涵盖49个类别的真实RESTful API；（ii）指令生成：我们提示ChatGPT生成涉及这些API的多样指令，涵盖单工具和多工具场景；（iii）解决路径注释：我们使用ChatGPT为每个指令搜索有效的解决路径（API调用链）。 为了增强LLM的推理能力，我们开发了一种基于深度优先搜索的决策树算法。 它使LLM能够评估多个推理轨迹并扩展搜索空间。 此外，为了评估LLM的工具使用能力，我们开发了一个自动评估器：ToolEval。基于ToolBench，我们微调LLaMA以获得一个LLM ToolLLaMA，并配备了一个神经API检索器，为每个指令推荐适当的API。 实验证明，ToolLLaMA表现出执行复杂指令和泛化到未见API的显著能力，并展示出与ChatGPT相当的性能。我们的ToolLLaMA还在一个超出分布的工具使用数据集APIBench中展示了强大的零-shot泛化能力。代码、训练模型和演示可在https://github.com/OpenBMB/ToolBench 公开获取。

1 引言

工具学习（秦等人，2023b）旨在释放大型语言模型（LLMs）的力量，以有效地与各种工具（API）互动，完成复杂任务。 通过将LLMs与API集成，我们可以极大地扩展它们的效用，并使它们成为用户和广泛应用生态系统之间高效的中介。 尽管开源LLMs，例如LLaMA（Touvron等人，2023a），通过指导调整（Taori等人，2023；Chiang等人，2023）已经实现了多功能能力，但它们仍然缺乏在执行更高级任务方面的复杂性，例如适当地与工具（API）互动以完成复杂的人类指令。 这种不足是因为当前的指导调整主要集中在基本语言任务上，相对忽视了工具使用领域。 另一方面，当前的最先进（SOTA）LLMs（例如ChatGPT（OpenAI，

* 表示相等的贡献。
† 通讯作者。

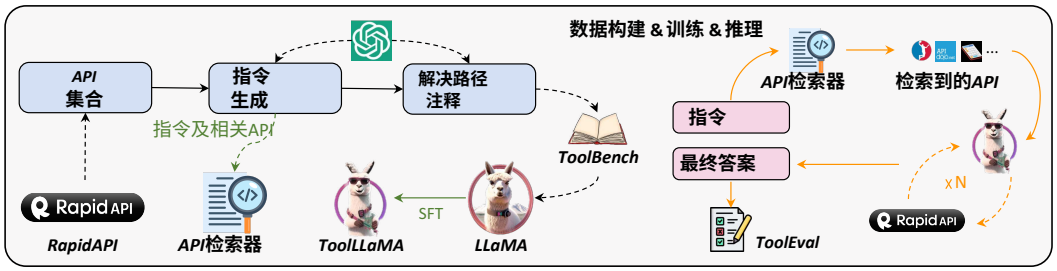

图1： 构建ToolBench的三个阶段以及我们如何训练我们的API检索器和ToolLLaMA。
在执行指令期间，API检索器向ToolLLaMA推荐相关的API，后者进行多轮API调用以得出最终答案。
整个推理过程由ToolEval评估。

2022年)和GPT-4 (OpenAI, 2023))已展示出在利用工具方面具有令人印象深刻的能力(Bubeck等, 2023)，但它们作为闭源系统，内部机制不透明。这限制了AI技术的民主化以及社区驱动的创新和发展范围。
在这方面，我们认为迫切需要 赋予开源*LLM*熟练掌握各种*API*的能力。

尽管先前的研究已经探讨了为工具使用构建指令调整数据的方法(Li等, 2023a; Patil等, 2023; Tang等, 2023; Xu等, 2023b)，但它们未能充分激发LLM内的工具使用能力，并具有固有的局限性：(1)**有限的API**：它们要么未涉及真实世界的API（例如RESTAPI）(Patil等, 2023; Tang等, 2023)，要么仅考虑了范围有限且缺乏多样性的API(Patil等, 2023; Xu等, 2023b; Li等, 2023a);
（2）受限场景：现有作品仅限于涉及单个工具的指令。 相比之下，现实世界的场景可能要求多个工具交错使用，进行多轮工具执行以解决复杂任务。 此外，它们经常假设用户事先手动指定给定指令的理想API集，这在拥有大量真实世界API的情况下是不可行的；（3）较差的规划和推理：现有的工作要么采用CoT（Wei等人，2023年）要么采用ReACT（Yao等人，2022年）进行模型推理，这不能充分激发LLM中存储的能力，因此无法处理复杂的指令。 此外，一些工作甚至不执行API以获得真实响应（Patil等人，2023年；唐等人，2023年），这对后续模型规划是重要信息。

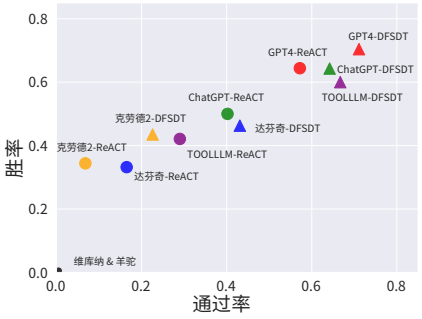

图2：不同方法在工具使用评估中的通过率（↑）和胜率（↑）
对于胜率，我们将每种方法与ChatGPT-ReACT进行比较。 DFSDT是我们在ReACT上改进的推理策略。 TOOLLLM超越了Text-Davinci-003、Claude-2，并且几乎与ChatGPT表现相当。

为了促进开源LLM内的工具使用能力，我们引入 **ToolLLM**，一个包括数据构建、模型训练和评估的通用工具使用框架。 如图1所示，我们收集了一个高质量的指令调整数据集 **ToolBench**。 它是自动构建的，使用了已升级为具有函数调用功能的ChatGPT（*gpt-3.5-turbo-16k*）。

ToolBench与先前的工作的比较列在表1中。 具体来说，ToolBench的构建包括三个阶段：

- **API收集**：我们从RapidAPI（链接）收集了16,464个表征状态转移（REST）API，这是一个由开发人员提供的大量真实世界API的平台。 这些API涵盖了49个不同的类别，如社交媒体、电子商务和天气。 对于每个API，我们从RapidAPI抓取详细的API文档，包括功能描述、所需参数、用于API调用的代码片段等。通过理解这些文档来学习执行API，LLMs可以推广到训练期间未见过的新API；

- **指令生成**：我们首先从整个集合中对API进行抽样，然后提示ChatGPT生成这些API的多样化指令。为了涵盖实际场景，我们策划指令

资源	工具台 (本文)	API台 (Patil等人，2023)	API银行 (Li等人，2023a)	ToolAlpaca (Tang等人，2023)	工具台 (Xu等人，2023b)
真实世界的API?	✓	✗	✓	✗	✓
真实API调用和响应?	✓	✗	✓	✗	✓
多工具场景?	✓	✗	✗	✗	✗
API检索?	✓	✓	✗	✗	✓
多步推理?	✓	✗	✓	✓	✓
工具数量	**3451**	3	53	400	8
API数量	**16464**	1645	53	400	232
实例数量	**126486**	17002	274	3938	2746
真实API调用次数	**469585**	0	568	0	3926
平均推理轨迹	4.0	1.0	2.1	1.0	**5.9**

表1：我们的ToolBench与用于工具学习的著名指导调整数据集的比较。

这涉及到既有单工具又有多工具场景。 这确保了我们的模型不仅学会如何与单个工具交互，还学会如何将它们结合起来完成复杂任务；

- 解决方案路径注释：每个解决方案路径可能包含多轮模型推理和实时API调用，以得出最终响应。 然而，即使是最复杂的LLM，即GPT-4，对于复杂的人类指令的通过率也很低，使得注释效率低下。 为此，我们开发了一种新颖的基于深度优先搜索的决策树（DFSDT）来增强LLM的规划和推理能力。与传统的ReACT相比，DFSDT使LLM能够评估多种推理路径，并做出谨慎的决策，要么撤回步骤，要么沿着一个有希望的路径继续前进。 在实验中，DFSDT显著提高了注释效率，并成功完成了那些无法使用ReACT完成的复杂指令。

为了评估LLM的工具使用能力，我们开发了一个自动评估器，ToolEval，由ChatGPT支持。它包括两个关键指标：（1）通过率，衡量LLM在有限预算内成功执行指令的能力，以及（2）双赢率，比较两个解决方案路径的质量和有用性。 我们证明ToolEval与人类评估具有很高的相关性，并为机器工具使用提供了稳健、可扩展和可靠的评估。通过在ToolBench上对LLaMA进行微调，我们获得 ToolLLaMA。 根据我们的ToolEval评估，我们得出以下发现：

- TOOLLLM展示了处理单工具和复杂多工具指令的引人注目的能力。 如图2所示，TOOLLLM优于Text-Davinci-003和Claude-2，达到了与“教师模型”ChatGPT相当的性能，仅略逊于GPT4。此外，TOOLLLM展现出对以前未见API的强大泛化能力，仅需要API文档即可有效适应新API。 这种灵活性使用户能够无缝地整合新的API，从而增强模型的实用性。

- 我们展示了我们的DFSDT作为增强LLM推理能力的一般决策策略。DFSDT通过考虑多个推理轨迹扩大了搜索空间，并且比ReACT表现出显著更好的性能。

- 我们训练了一个神经**API检索器**，实际上减轻了从大型API池中手动选择的需求。 如图1所示，给定一条指令，API检索器推荐一组相关的API，这些API被发送到ToolLLaMA进行多轮决策，以得出最终答案。 尽管在大量API中进行筛选，检索器表现出出色的检索精度，返回与真实情况密切相关的API。

- ToolLLaMA在一个分布外（OOD）数据集APIBench（Patil等，2023年）上展现出强大的泛化性能。 尽管没有在APIBench上训练任何API或指令，ToolLLaMA的表现与Gorilla相当，后者是专门为APIBench设计的流水线。

2 数据集构建

我们介绍了ToolBench的三阶段构建过程：API收集（§ 2.1）、指令生成（§ 2.2）和解决路径注释（§ 2.3）。所有程序都基于ChatGPT（*gpt-3.5-turbo-16k*），需要最少的人工监督，并且可以轻松扩展到新的API。

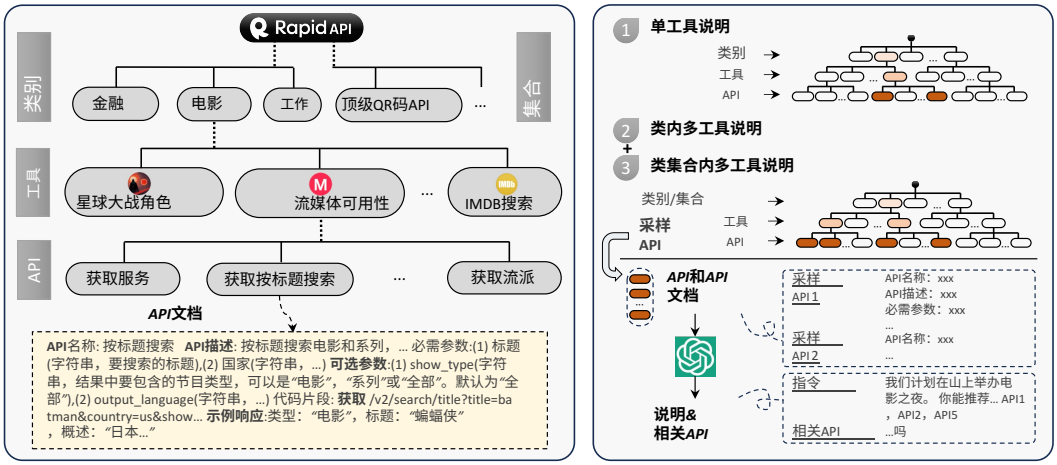

图3：RapidAPI的层次结构（左）和指令生成过程（右）。

2.1 API集合

我们首先介绍RapidAPI及其层次结构，然后介绍我们如何爬取和过滤API。RapidAPI Hub RapidAPI是一个领先的API市场，将开发人员与数千个真实世界API连接起来，简化了将各种服务集成到应用程序中的过程。 开发人员只需注册一个RapidAPI密钥，即可测试和连接各种API。RapidAPI中的所有API都可以分类为 49个粗粒度类别（链接），如体育、金融和天气。 这些类别将API与最相关的主题关联起来。 此外，该中心还提供 $500+$细粒度分类，称为**集合**(链接)，例如，中文API和数据库API。同一集合中的API共享共同特征，通常具有类似的功能或目标。

RapidAPI的层次结构如图3所示，每个工具可能由多个API组成。对于每个工具，我们抓取以下信息：工具的名称和描述，主机的URL以及属于该工具的所有可用API；对于每个API，我们记录其名称、描述、HTTP方法、必需参数、可选参数、请求体、用于API调用的可执行代码片段以及示例API调用响应。 这些丰富而详细的元数据为LLMs提供了宝贵的资源，以便了解和有效使用API，甚至在零-shot方式下。

API过滤最初，我们从RapidAPI收集了 $10,853$个工具（$53,190$个**API**）。然而，这些**API**的质量和可靠性可能会有很大差异。 特别是，一些**API**可能没有得到很好的维护，比如返回404错误或其他内部错误。 为此，我们进行了严格的过滤过程（详细信息见附录**A**.1），以确保ToolBench的最终工具集可靠且功能齐全。最终，我们仅保留了$3,451$个高质量工具（$16,464$个**API**）。

2.2 指令生成

与先前的工作不同，我们专注于指令生成的两个关键方面：（1）多样性：训练LLMs处理各种API使用场景，从而提高其泛化能力和鲁棒性；以及（2）多工具使用：模拟现实世界中经常需要多个工具相互配合的情况，提高LLMs的实际适用性和灵活性。为此，我们不是从头开始构思指令，然后搜索相关API，而是对API进行不同组合的采样，并制作涉及它们的各种指令。

为API生成指令将总API集定义为 $\mathbb{S}_{\text{API}}$，在每个时间点，我们从 $\mathbb{S}$**API**中抽样几个**API**:Ssub N ={**API**1,· · ·,**API** N}。 我们提示ChatGPT理解这些API的功能，然后生成(1)涉及ssub N 中**API**的可能指令(**In**$^{\text{s}}$**t**$*$)，以及(2)每个指令(Inst$*$)的相关**API**(s**rel**$*\subset$ssub N)，即 $\{[\mathbb{S}_1^{\text{rel}}, \text{Inst}_1], \cdots, [\mathbb{S}^{\text{relN}'}, \text{Inst}_{N'}]\}$，其中N′表示生成的实例数量。 这些(指令，相关API)对将用于

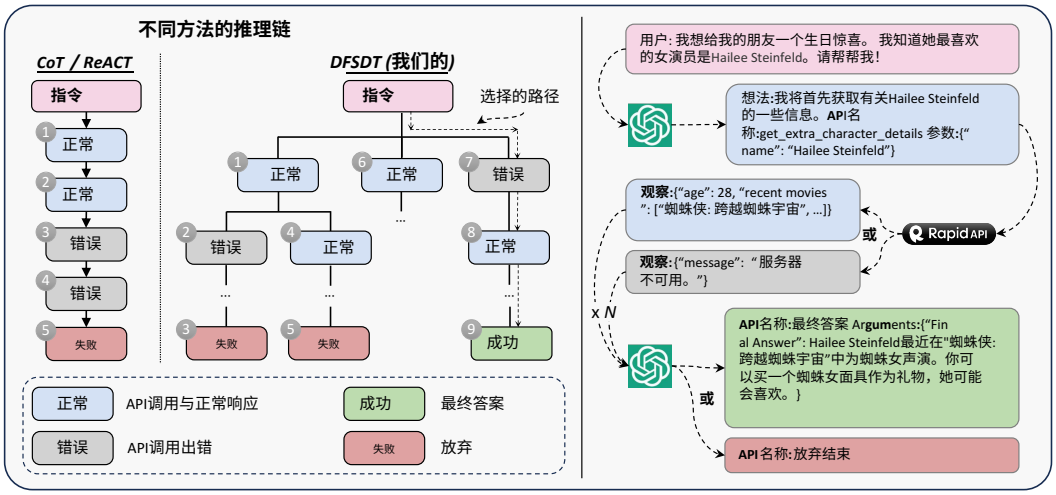

图4：我们的DFSDT与传统的CoT或ReACT在模型推理过程中的比较（左）。 我们展示了使用ChatGPT进行解决路径注释过程的一部分（右）。

在§ 3.1中训练API检索器。我们使用不同的采样策略（稍后介绍）来覆盖所有API及其大部分组合，从而确保我们指令的多样性。

ChatGPT的提示由以下内容组成：（1）对预期指令生成任务的一般描述，（2）每个API在$\mathbb{S}_{\text{N}}^{\text{sub}}$中的全面文档，帮助ChatGPT理解它们的功能和相互作用，以及（3）三个上下文种子示例$\{\text{seed}_1, \text{seed}_2, \text{seed}_3\}$。 每个种子示例都是由人类专家编写的理想指令生成。 这些种子示例被利用来通过上下文学习更好地调节ChatGPT的行为。 总共，我们为单工具/多工具设置编写了12 / 36个不同的种子示例（$\mathbb{S}_{\text{种子}}$），并在每次随机抽取三个示例。 有关指令生成的详细提示请参见附录A.7。总体而言，生成过程可以表述如下：

$$\underset{\{\text{API}_1,\cdots,\text{API}_\text{N}\}\in\mathbb{S}_{\text{API}},\{\text{seed}_1,\cdots,\text{seed}_3\}\in\mathbb{S}_{\text{seed}}}{\text{ChatGPT}}(\{[\mathbb{S}_1^{\text{rel}}, \text{Inst}_1], \cdots, [\mathbb{S}_{\text{N}'}^{\text{rel}}, \text{Inst}_{\text{N}'}]\}|\text{API}_1, \cdots, \text{API}_\text{N}, \text{seed}_1, \cdots, \text{seed}_3).$$

不同场景的抽样策略如图3所示，对于单工具指令（I1），我们遍历每个工具并生成其API的指令。然而，对于多工具设置，由于RapidAPI中不同工具之间的互联关系稀疏，从整个工具集中随机抽样工具组合通常会导致一系列无法以自然方式覆盖的无关工具。为了解决稀疏性问题，我们利用了RapidAPI的层次信息。 由于属于同一RapidAPI类别或集合的工具通常在功能和目标上相关，我们随机选择2-5个来自同一类别/集合的工具，并最多抽样每个工具的3个API来生成指令。 我们将生成的指令标记为类内多工具指令（I2）和集合内多工具指令（I3），分别。 通过严格的人类评估，我们发现以这种方式生成的指令已经具有高度多样性，涵盖了各种实际场景。 我们还提供了使用Atlas（链接）对指令进行可视化以支持我们的说法。

在生成初始一组指令后，我们通过评估它们是否存在于$\mathbb{S}_{\text{N}}^{\text{sub}}$中，进一步过滤那些具有虚构相关API的指令。 最后，我们收集了近200k个合格的（指令，相关API）对，包括分别为I1、I2和I3的87413、84815和25251个实例。

2.3 SOLUTION PATH ANNOTATION

如图4所示，给定一个指令Inst_*，我们提示ChatGPT搜索一个有效的动作序列：{a1, · · · , aN}。 ChatGPT将这样一个多步决策过程视为一个多轮对话。在每一轮t中，模型根据先前的交互生成一个动作at，即ChatGPT(at|{a1, r1, · · · , at−1, rt−1}, Inst∗)，其中r∗表示真实的API响应。对于每一个

在Cha_tGPT中，应该指定其“思考”的API使用方式和具体参数，即，ath$a$s $_t$he following forma_t:“Though$_t$:・・・,API N$a$me:・・・,Par$a$me$_t$ers:・・・”.

为了利用ChatGPT的**function call**功能，我们将每个API视为一个特殊函数，并将其API文档输入到ChatGPT的函数字段中。 通过这种方式，模型了解如何调用API。对于每个指令Inst$_*$，我们将所有抽样的APIs $\mathbb{S}_{\mathrm{N}}^{\mathrm{sub}}$作为可用函数输入到ChatGPT中。 为了让ChatGPT完成一个动作序列，我们定义了两个额外的函数，即“以最终答案结束”和“放弃结束”。前者的参数对应于原始指令的详细最终答案；而后者设计用于在多次API调用尝试后无法完成原始指令的情况。

基于深度优先搜索的决策树在我们的试点研究中，我们发现CoT（Wei等人，2023年）或ReACT（Yao等人，2022年）存在固有的局限性：（1）**错误传播**：错误的行动可能会进一步传播错误，导致模型陷入故障循环，比如持续以错误的方式调用API或产生API幻觉；（2）**有限探索**：CoT或ReACT只探索一种可能的方向，导致对整个行动空间的探索有限。 因此，即使是GPT-4经常无法找到有效的解决路径，使注释变得困难。

为此，我们建议构建一个决策树来扩展搜索空间，增加找到有效路径的可能性。 如图4所示，我们的DFSDT允许模型评估不同的推理路径，并选择要么（1）沿着有希望的路径继续，要么（2）通过调用“放弃”函数放弃现有节点并扩展一个新节点。 在节点扩展过程中，为了使子节点多样化并扩展搜索空间，我们提示ChatGPT使用先前生成的节点信息，并明确鼓励模型生成一个不同的节点。 在搜索过程中，我们更喜欢深度优先搜索（DFS）而不是广度优先搜索（BFS），因为只要找到一个有效路径，注释就可以完成。 使用BFS会导致过多的OpenAI API调用。 更多细节请参见附录A.8。我们对所有生成的指令执行DFSDT，并仅保留通过解决方案路径的那些。 最终，我们生成了12$6$，48$6$（指令，解决方案路径）对，用于训练第3.2节中的ToolLLaMA。

3 实验

在本节中，我们调查ToolLLM框架的性能。 我们首先介绍评估指标，并评估API检索器和DFSDT的有效性在第3.1节。然后我们在第3.2节中呈现主要实验，随后在第3.3节进行一项泛化实验。

3.1 P初步实验

ToolEval 考虑到RapidAPI上API的时间变化性以及对于一条指令的无限潜在解决路径，为每个测试指令注释一个固定的真实解决路径是不可行的。 此外，在比较不同模型时，确保它们在评估过程中使用相同版本的API是至关重要的。 考虑到人工评估可能耗时，我们遵循AlpacaEval（Li等人，2023b）的做法，基于ChatGPT开发了一个高效的评估器ToolEval，其中包含两个评估指标（详见附录A.5）：（1）通过率：它计算在有限预算内成功完成指令的比例。 该指标衡量了LLM执行指令的可执行性，可以视为理想工具使用的基本要求；以及（2）胜率：我们向ChatGPT评估器提供一条指令和两条解决路径，并获取其偏好（即哪一条更好）。 我们为度量标准预先定义了一组标准，这些标准被组织为我们的ChatGPT评估器的提示。 我们基于ChatGPT进行多次评估以提高可靠性。 然后我们从评估器中计算平均结果。

通过严格测试（详细信息见附录A.5），我们发现ToolEval在通过率方面与人类标注者达成高达87.1%的高度一致性，在胜率方面达到80.3%。 这表明ToolEval在很大程度上可以反映和代表人类评估。

API检索器的有效性API检索器旨在检索与指令相关的API。 我们采用Sentence-BERT（Reimers＆Gurevych，2019）来训练基于BERT-BASE（Devlin等人，2019）的密集检索器。API检索器将指令和API文档编码为两个嵌入，并通过嵌入相似性计算它们的相关性。 对于训练，我们将§ 2.2中生成的每个指令的相关API视为正例，并随机抽取其他一些

方法	I1 NDCG		I2 NDCG		I3 NDCG		平均 NDCG	
	@1	@5	@1	@5	@1	@5	@1	@5
BM25	18.4	19.7	12.0	11.0	25.2	20.4	18.5	17.0
Ada	57.5	58.8	36.8	30.7	54.6	46.8	49.6	45.4
我们的	**84.2**	**89.7**	**68.2**	**77.9**	**81.7**	**87.1**	**78.0**	**84.9**

表2：我们的API检索器与三种指令（I1、I2、I3）的两个基线进行对比。我们报告NDCG@1和NDCG@5。

方法	I1	I2	_	I3 平均
ReACT	37.8	40.6	27.6	35.3
ReACT@N	49.4	49.4	34.6	44.5
DFSDT	**58.0**	**70.6**	**62.8**	**63.8**

表3：基于ChatGPT的三种指令（I1、I2、I3）不同推理策略的通过率。

API作为对比学习的负例。 对于基线，我们选择BM25（Robertson等，2009）和OpenAI的text-embedding-ada-002（链接）。 我们使用NDCG（Jarvelin & Kek ̈al ̈ainen，2002）来评估检索性能。 我们在单工具指令（I1）、类内多工具指令（I2）和类间
多工具指令（I3）上训练和评估我们的模型。

如表2所示，我们的API检索器在所有设置下始终优于基线，表明其在具有大量API的实际场景中的可行性。此外，I1的NDCG分数通常高于I2和I3，这意味着单工具指令的检索比多工具设置更简单。

在解决路径注释之前，我们验证了**DFSDT**相对于**ReACT**的优越性。基于ChatGPT，我们使用通过率指标比较**DFSDT**和**ReACT**。 由于**DFSDT**比**ReACT**消耗更多的OpenAI API调用，为了进行更公平的比较，我们还建立了一个“**ReACT**@N”基准，该基准执行多次**ReACT**直到总成本达到**DFSDT**相同水平。一旦**ReACT**@N找到有效解决方案，我们认为它通过了
。

从表3可以看出，在所有情况下，DFSDT明显优于两个基准。 由于我们只保留那些通过的注释作为训练数据，给定相同的预算，使用DFSDT可以注释更多的指令。 这使得DFSDT成为一种更有效的方式，可以节省总注释成本。 我们还发现，DFSDT的性能改善在更难的指令（即I2和I3）上比那些更简单的指令（I1）更为明显。 这意味着通过扩大搜索空间，DFSDT可以更好地解决那些即使反复执行也无法回答的困难、复杂的指令。 在我们的数据集中涉及这种“难题”可以充分激发工具使用能力，应对那些复杂情境。

3.2 M 主实验

ToolLLaMA 我们使用指令-解决方案对LLaMA-2 7B模型（Touvron等人，2023b）进行微调。 原始的LLaMA-2模型的序列长度为 4096，在我们的设置下不够，因为API响应可能非常长。 为此，我们使用位置插值（Chen等人，2023）将上下文长度扩展到 8192（附录A.3中的训练细节）。设置理想情况下，通过扩大训练数据中指令和独特工具的数量和多

样性，预期ToolLLaMA能够推广到训练期间未见过的新指令和API。 这是有意义的，因为用户可以定义定制的API，并期望ToolLLaMA根据文档进行调整。 为此，我们努力评估ToolLLaMA在三个层面上的泛化能力：(1)实例：训练数据中相同工具的未见指令，(2)工具：

未见工具属于训练数据中同一（已见）类别的工具，以及(3)**Cat.**：
未见工具属于训练数据中不同（未见）类别的工具。

我们在三种情景下进行实验：单工具指令（I1）、同一类别多工具指令（I2）和不同类别多工具指令（I3）。 对于I1，我们对上述三个级别（I1-Inst.、I1-Tool和I1-Cat.）进行评估；对于I2，由于训练指令已涉及同一类别的不同工具，我们仅对泛化评估进行第1级和第3级（I2-Inst.和I2-Cat.）；同样，对于I3，我们仅进行第1级泛化评估（I3-Inst.），因为它已涵盖涉及不同类别工具组合的指令（RapidAPI集合中的工具可能来自不同的RapidAPI类别）。

对于每个测试指令，我们将真实世界的API（oracle） $\mathbb{S}_N^{sub}$馈送给每个模型。 这模拟了用户指定他们偏好的API集的情景。

基线我们选择了两个为通用对话进行微调的LLaMA变体，即Vicuna（Chiang等，2023年）和Alpaca（Taori等，2023年）。 我们还选择了“教师模型”

模型	方法	I1-Inst.		I1-Tool		I1-Cat.		I2-Inst.		I2-Cat.		I3-Inst.		平均	
		Pass	Win	Pass	Win	Pass	Win	Pass	Win	Pass	Win	Pass	Win	Pass	Win
ChatGPT	ReACT	41.5	-	44.0	-	44.5	-	42.5	-	46.5	-	22.0	-	40.2	-
	DFSDT	54.5	60.5	65.0	62.0	60.5	57.3	75.0	72.0	71.5	**64.8**	62.0	69.0	64.8	64.3
Claude-2	ReACT	5.5	31.0	3.5	27.8	5.5	33.8	6.0	35.0	6.0	31.5	14.0	47.5	6.8	34.4
	DFSDT	20.5	38.0	31.0	44.3	18.5	43.3	17.0	36.8	20.5	33.5	28.0	65.0	22.6	43.5
Text-Davinci-003	ReACT	12.0	28.5	20.0	35.3	20.0	31.0	8.5	29.8	14.5	29.8	24.0	45.0	16.5	33.2
	DFSDT	43.5	40.3	44.0	43.8	46.0	46.8	37.0	40.5	42.0	43.3	46.0	63.0	43.1	46.3
GPT4	ReACT	53.5	60.0	50.0	58.8	53.5	63.5	67.0	65.8	72.0	60.3	47.0	78.0	57.2	64.4
	DFSDT	60.0	**67.5**	**71.5**	**67.8**	**67.0**	**66.5**	79.5	**73.3**	**77.5**	63.3	**71.0**	**84.0**	**71.1**	**70.4**
Vicuna	ReACT & DFSDT	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Alpaca	ReACT & DFSDT	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
ToolLLaMA	ReACT	25.0	45.0	29.0	42.0	33.0	47.5	30.5	50.8	31.5	41.8	25.0	55.0	29.0	47.0
	DFSDT	57.0	55.0	61.0	55.3	62.0	54.5	77.0	68.5	77.0	58.0	66.0	69.0	66.7	60.0
	DFSDT-Retriever	**64.0**	62.3	64.0	59.0	60.5	55.0	**81.5**	68.5	68.5	60.8	65.0	73.0	67.3	63.1

表4：ToolBench的主要实验。胜率是通过将每个模型与ChatGPT-ReACT进行比较来计算的。胜率高于 50%意味着该模型表现优于ChatGPT-ReACT。除了ToolLLaMA-DFSDT-Retriever外，所有方法均使用Oracle API检索器（即地面真实API）。

ChatGPT，Text-Davinci-003，GPT-4和Claude-2作为基线，并对它们应用DFSDT和ReACT。 在计算胜率时，每个模型都与ChatGPT-ReACT进行比较。主要结果结果显示

在表4中，我们得出：

1. 尽管我们进行了广泛的提示工程，但Vicuna和Alpaca都未能通过任何指令（通过率和胜率= 0），这意味着它们的指令遵循能力不涵盖工具使用领域。 这凸显了当前指令调整尝试的不足，主要集中在语言技能上;

2. 对于所有LLMs，使用DFSDT在通过率和胜率方面明显优于ReACT。
值得注意的是，ChatGPT +DFSDT在通过率上超过了GPT-4+ReACT，并在胜率上表现相当。 这凸显了DFSDT在决策方面优于ReACT的优越性; 3. 在使用DFSDT时，Tool
LLaMA比Text-Dainci-003和Claude-2表现得更好，并几乎与ChatGPT（教师模型）持平。
总的来说，尽管可以泛化到看不见的指令和工具，ToolLLaMA +DFSDT在所有场景中展现出竞争性的泛化性能，实现了仅次于GPT4+DFSDT的通过率。总的来说，这些结果表明ToolBench可以充分激发LLMs内的工具使用能力，并使它们能够熟

练掌握甚至看不见的各种指令的API。

将API检索器与TOOLLLM集成在现实世界的场景中，要求用户从大量API中手动推荐可能并不实际。为了模拟这种实际情况并测试我们的API检索器的效率，我们将API检索器推荐的前 5个API（而不是地面真实API $\mathbb{S}_{\mathrm{N}}^{\mathrm{sub}}$）提供给TOOLLLM。如表4所示，使用检索到的API甚至提高了性能（通过率和胜率），与地面真实API集相比。这是因为地面真实API集中的许多API可以被其他功能更好的类似API替代，而我们的API检索器可以成功识别。 换句话说，我们的检索器扩展了相关API的搜索空间，并为当前指令找到了更合适的API。 这为我们的API检索器检索相关API的出色能力提供了强有力的证据，特别是考虑到我们的API检索器从中选择的庞大API池（16，000+）。

3.3 OUT-OF-DISTRIBUTION (OOD) GENERALIZATION TO APIBENCH (PATIL ET AL., 2023)

设置 我们进一步扩展了ToolLLaMA，以便在OOD数据集APIBench上验证其泛化能力。 为了评估ToolLLaMA在这些新领域的泛化能力，我们为ToolLLaMA配备了两个检索器：我们训练的API检索器和oracle检索器。 我们评估了APIBench的三个领域，即TorchHub、TensorHub和HuggingFace。我们将ToolLLaMA与Gorilla进行比较，Gorilla是使用APIBench的训练数据微调的LLaMA-7B模型。按照原始论文，我们采用了Gorilla的两个官方设置：零炮击设置（ZS）和检索感知设置（RS）。 后者（RS）意味着检索到的API被发送到模型作为提示的一部分；而前者（ZS）在训练模型时不将API包含在提示中。 我们采用官方评估指标，并报告AST准确率以及幻觉率。

方法	拥抱脸		火炬中心		张力中心	
	幻觉率 (↓)	AST准确率 (↑)	幻觉率 (↓)	AST准确率 (↑)	幻觉率 (↓)	AST准确率 (↑)
TOOLLLM + 我们的检索器	10.60	**16.77**	15.70	**51.16**	6.48	40.59
大猩猩-ZS + BM25	46.90	10.51	17.20	44.62	20.58	34.31
大猩猩-RS + BM25	**6.42**	15.71	**5.91**	50.00	**2.77**	**41.90**
TOOLLLM + 甲骨文	8.66	88.80	14.12	85.88	7.44	88.62
大猩猩-ZS + 甲骨文	52.88	44.36	39.25	59.14	12.99	83.21
大猩猩-RS + 甲骨文	**6.97**	**89.27**	**6.99**	**93.01**	**2.04**	**94.16**

表5：在APIBench上进行的OOD泛化实验。对于大猩猩条目，ZS / RS表示大猩猩在APIBench上以零-shot / 检索感知设置进行训练。我们报告幻觉率和AST准确率。

结果结果显示在表5中。 总体而言，TOOLLLM在三个数据集上都取得了显著的OOD泛化性能，尽管它是在完全不同的API领域和指令领域进行训练的。 具体来说，TOOLLLM+我们的API检索器在HuggingFace和TorchHub上的AST准确性方面在两种训练设置（ZS / RS）下均优于Gorilla+BM25。在相同的oracle检索器下，与Gorilla-ZS相比，TOOLLLM始终表现更优。值得注意的是，由于我们更复杂的设置，如多工具使用和多步推理，Gorilla模型无法推广到我们的ToolBench数据集。

4 相关工作

工具学习最近的研究揭示了LLMs在掌握工具和在复杂环境中做出决策方面的不断增长的能力（Vemprala等，2023年；Nakano等，2021年；Qin等，2023a年；Shen等，2023年；Wu等，2023年；Schick等，2023年；Hao等，2023年；Qian等，2023年；Song等，2023年；Zhuang等，2023年；Gao等，2023年）。 获得对外部工具的访问使LLMs具有实时的事实知识（Yang等，2023年），多模态功能（Gupta＆Kembhavi，2023年）以及垂直领域的专业技能（Jin等，2023年）。
然而，开源LLMs在工具使用方面仍远远落后于SOTA LLMs，SOTA LLMs如何获得工具使用能力仍不清楚。 在本文中，我们旨在弥合这一差距并探究其中的潜在机制。

指令调整指令调整增强LLMs对人类指令的理解并生成适当的响应（Wei等，2021年；Bach等，2022年；Mishra等，2022年）。 由于手动注释指令调整数据耗时，自我指导（Wang等，2022年）提出从SOTA LLMs生成高质量数据，这促进了最近关于多轮对话数据整理的趋势（Taori等，2023年；Chiang等，2023年；Xu等，2023a；Penedo等，2023年；Ding等，2023年）。 然而，与对话相比，工具学习在API的广泛多样性和多工具指令的复杂性方面更具挑战性。 因此，即使是GPT-4经常无法找到有效的解决方案路径。 然而，现有的工具学习数据集（Li等，2023a；Patil等，2023；Tang等，2023；Xu等，2023b）及其构建方法无法有效地满足实际人类需求，如第1节所述。 相反，我们的ToolBench专为实际场景设计，并改进了先前用于工具学习数据构建的流程。

促使**LLMs**进行决策促使**LLMs**将高级任务分解为子任务，并生成基于计划的计划（Ahn等，2022；Huang等，2022a;b；Ye等，2023）。 ReACT（Yao等，2022）通过允许**LLMs**为行动提供适当的理由并将环境反馈纳入推理中，将推理与行动相结合。 然而，这些研究并未纳入决策撤销机制，这变得棘手，因为初始错误可能导致一系列后续错误。 最近，Reflexion（Shinn等，2023）通过要求**LLMs**反思先前的失败来缓解这个问题。 我们的DFSDT通过允许**LLMs**评估不同的推理路径并选择最有前途的路径，将Reflexion扩展为一种更通用的方法。值得注意的是，DFSDT与一项并行工作——思维树（ToT）推理（Yao等，2023）——分享了类似的思想。 然而，我们的DFSDT针对决策空间为无限的一般决策问题，与ToT相比，后者相对简单的任务可以通过蛮力搜索解决，比如24点游戏和填字游戏。DFSDT和ToT之间的不同目标决定了在实现细节上的显著差异。

5 C结论

在这项工作中，我们介绍了如何引出LLMs内的工具使用能力。我们首先提出了一个指令调整数据集，ToolBench，涵盖了16k+个真实世界API和各种实际使用案例场景，包括单工具和多工具任务。 ToolBench的构建纯粹使用ChatGPT，并且需要最少的人工监督。 此外，我们提出DFSDT来加强LLMs的规划和推理能力，使它们能够战略性地穿越推理路径。 为了有效评估工具学习，我们设计了一个自动评估器ToolEval。通过在ToolBench上对LLaMA进行微调，得到的模型ToolLLaMA与ChatGPT的性能相匹配，并展现出对未见API的显著泛化能力。此外，我们开发了一个神经API检索器，为每个指令推荐相关的API。 检索器可以与ToolLLaMA集成，形成一个更自动化的工具使用流程。 在实验中，我们展示了我们的管道对于超出分布领域的泛化能力。 总的来说，这项工作为将指令调整和LLMs的工具使用交叉研究铺平了道路。

参考文献

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog等人。不要像我说的那样做，而是像我能做的那样：将语言基于机器人的可行性。Ar*X*iv预印本，abs/2204.01691，2022年。

Stephen Bach, Victor Sanh, Zheng Xin Yong, Albert Webson, Colin Raffel, Nihal V Nayak, Abheesht Sharma, Taewoon Kim, M Saiful Bari, Thibault Fevry等人。Promptsource：自然语言提示的集成开发环境和存储库。 在计算语言学协会第*60*届年会论文集: 系统演示, pp. 93–1*0*4, 2*0*22.

Sebastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar,
tt Lundberg, 等人。人工 *arXiv*预印本 *arXiv:2303.12712*, 2023.

Shouyuan Chen, Sherman Wong, Liangjian Chen, 和 Yuandong Tian。通过位置插值扩展大型语言模型的上下文窗口。*arXiv*预印本 *arXiv:2306.15595*, 2023.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, 和 Eric P. Xing。Vicuna: 一个开源聊天机器人，以90%*的chatgpt质量打动gpt-4，2023年3月。 网址 https://lmsys.org/blog/2023-03-30-vicuna/.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, 和 Kristina Toutanova. BERT: 深度双向转换器的预训练用于语言理解。 在*2019*年北美计算语言学协会会议论文集：人类语言技术，第*1*卷（长篇和短篇）, pp. 4171–4186, 明尼阿波利斯，明尼苏达州，2019年。

计算语言学协会。doi: 10.18653/v1/N19-1423. 网址 https://aclanthology.org/N19-1423.

宁丁，陈玉琳，徐博凯，秦宇佳，郑智，胡胜定，刘志远，孙茂松，和周博文。通过扩展高质量指导对话来增强聊天语言模型。*arXiv*预印本 *arXiv:2305.14233*, 2023年.

高迪飞，季磊，周洛伟，林庆宏，陈娇雅，范子涵，和郑铭。Assistgpt：一个通用的多模态助手，可以规划，执行，检查和学习。*arXiv*预印本*arXiv:2306.08640*, 2023年.

Tanmay Gupta和Aniruddha Kembhavi。视觉编程：无需训练的组合视觉推理。 在*IEEE/CVF*计算机视觉与模式识别会议论文集, pp. 14953–14962, 2023年.

郝世博，刘天阳，王震，和胡志婷。Toolkengpt：通过工具嵌入扩展冻结语言模型。*arXiv*预印本 *arXiv:2305.11554*, 2023年.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, 和 Igor Mordatch。语言模型作为零-shot规划器：提取行动知识以供实体代理使用。 在Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gáng Niu, 和 Sivan Sabato (编辑)的国际机器学习大会, *ICML 2022, 2022*年*7*月*17-23*日, 美国马里兰州巴尔的摩, 机器学习研究论文集第162卷, 页码9118–9147。PMLR, 2022a。

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, 等。内心独白：通过语言模型进行规划的具身化推理。*ArXiv*预印本, abs/2207.05608, 2022b.

Kalervo Järvelin和Jaana Kẻk al¨ ainen。 基于累积增益的信息检索技术评估。 *ACM* 信息系统交易（*TOIS*）, 20(4):422–446, 2002.

乔金，易凡杨，庆宇陈和志勇卢。Genegpt：利用领域工具增强大型语言模型，以改善对生物医学信息的访问。 *ArXiv*, 2023.

李明浩，宋飞凡，余博文，余海洋，李周军，黄飞和李永斌。Api-bank：用于工具增强的llms的基准。*arXiv*预印本*arXiv:2304.08244*, 2023a.

李学臣，张天一，杨迪，罗翰陶瑞，伊尚古尔拉贾尼，卡洛斯格斯特林，珀西梁和桥本辰德。Alpacaeval：一种自动评估指令遵循模型的工具。`https://github.com/tatsu-lab/alpaca_eval`，2023b。

Swaroop Mishra，Daniel Khashabi，Chitta Baral和Hannaneh Hajishirzi。通过自然语言众包指令实现跨任务泛化。 在计算语言学协会第*60*届年会论文集（第*1*卷：长文）中，页码347*0*-3487，2*0*22年。

Reiichiro Nakano，Jacob Hilton，Suchir Balaji，Jeff Wu，Long Ouyang，Christina Kim，Christopher Hesse，Shantanu Jain，Vineet Kosaraju，William Saunders等。Webgpt：带有人类反馈的浏览器辅助问答。*ArXiv*预印本，abs/2112.09332，2021年。

OpenAI。 OpenAI：介绍ChatGPT，2022年。 网址 `https://openai.com/blog/chatgpt`.

OpenAI。 Gpt-4技术报告，2023年。

Shishir G Patil，Tianjun Zhang，Xin Wang和Joseph E Gonzalez。Gorilla：与大量API连接的大型语言模型。*arXiv*预印本*arXiv:2305.15334*，2023年。

Guilherme Penedo，Quentin Malartic，Daniel Hesslow，Ruxandra Cojocaru，Alessandro Cappelli，Hamza Alobeidli，Baptiste Pannier，Ebtesam Almazrouei和Julien Launay。Falcon llm的精细网络数据集：通过网络数据和仅有网络数据胜过精心策划的语料库。 *arXiv*预印本*arXiv:2306.01116*，2023年。

Cheng Qian，Chi Han，Yi R Fung，Yujia Qin，Zhiyuan Liu和Heng Ji。Creator：通过工具创建解开大型语言模型的抽象和具体推理。*arXiv*预印本*arXiv:2305.14318*，2023年。

秦宇佳，蔡子涵，金典，严岚，梁世豪，朱坤伦，林彦凯，韩旭，丁宁，王华东等。Webcpm：用于中文长篇问题回答的交互式网络搜索。
*arXiv*预印本*arXiv:2305.06849*，2023年。

秦宇佳，胡胜鼎，林彦凯，陈维泽，丁宁，崔甘曲，曾振宜，黄宇飞，肖超军，韩驰等。基于基础模型的工具学习。 *arXiv*预印本*arXiv:2304.08354*，2023年。

Nils Reimers和Iryna Gurevych。Sentence-bert：使用孪生bert网络的句子嵌入。
*arXiv*预印本 *arXiv:1908.10084*, 2019.

Stephen Robertson, Hugo Zaragoza等。概率相关性框架：BM25及其发展。
信息检索的基础与趋势, 3(4):333–389, 2009.

Timo Schick, Jane Dwivedi-Yu, Roberto Dess`ı, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda和Thomas Scialom。Toolformer：语言模型可以自学使用工具。*ArXiv*预印本, abs/2302.04761, 2023.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, 和 Yueting Zhuang。Hugginggpt：使用chatgpt及其在huggingface中的朋友解决ai任务，2023年。

Noah Shinn, Federico Cassano, Beck Labash, Ashwin Gopinath, Karthik Narasimhan, 和 Shunyu Yao. Reflexion: 语言代理与口头强化学习, 2023.

Yifan Song, Weimin Xiong, Dawei Zhu, Cheng Li, Ke Wang, 叶田, 和 Sujian Li. Restgpt: 通过restful apis连接大型语言模型与真实世界应用*arXiv* 预印本*arXiv:2306.06624*, 2023.

Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, 和 Le Sun. Toolalpaca: 针对语言模型的通用工具学习，涵盖3000个模拟案例*arXiv* 预印本 *arXiv:2306.05301*, 2023.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, 和 Tatsunori B. Hashimoto. Stanford alpaca: 一个遵循指令的羊驼模型。`https://github.com/tatsu-lab/stanford_alpaca`, 2023.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothee´ Lacroix, Baptiste Roziere, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand douard Grave, 和 Guillaume Lample. Llama: 开放且高效的基础语言模型。*arXiv*预印本 *arXiv:2302.13971*, 2023a.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, 等。Llama 2: 开放基础和精细调整的聊天模型。*arXiv*预印本 *arXiv:2307.09288*, 2023b.

Sai Vemprala, Rogerio Bonatti, Arthur Bucker和Ashish Kapoor。Chatgpt用于机器人技术：设计原则和模型能力。 技术报告MSR-TR-2023-8，微软，2023年2月。

王一中，Yeganeh Kordi，Swaroop Mishra，Alisa Liu，Noah A Smith，Daniel Khashabi和Hannaneh Hajishirzi。自我指导：将语言模型与自动生成的指令对齐。*arXiv*预印本 *arXiv:2212.10560*, 2022年。

Jason Wei，Maarten Bosma，Vincent Y Zhao，Kelvin Guu，Adams Wei Yu，Brian Lester，Nan Du，Andrew M Dai和Quoc V Le。微调语言模型是零-shot学习者。*arXiv*预印本*arXiv:2109.01652*，2021年。

Jason Wei，Xuezhi Wang，Dale Schuurmans，Maarten Bosma，Brian Ichter，Fei Xia，Ed Chi，Quoc Le和Denny Zhou。思维链提示引发大型语言模型的推理，2023年。

Chenfei Wu，Shengming Yin，Weizhen Qi，Xiaodong Wang，Zecheng Tang和Nan Duan。Visual ChatGPT：使用视觉基础模型进行对话、绘图和编辑。a*rXiv*预印本，abs/2303.04671，2023年。

Can Xu，Qingfeng Sun，Kai Zheng，Xiubo Geng，Pu Zhao，Jiazhan Feng，Chongyang Tao和Daxin Jiang。WizardLM：赋予大型语言模型遵循复杂指令的能力，2023年。

Qiantong Xu, Fenglu Hong, Bo Li, Changran Hu, Zhengyu Chen, 和 Jian Zhang. 关于开源大型语言模型的工具操作能力。*arXiv*预印本 *arXiv:2305.16504*,2023b.

Linyao Yang, Hongyang Chen, Zhao Li, Xiao Ding, 和 Xindong Wu. Chatgpt 不够: 通过知识图谱增强大型语言模型以进行事实感知语言建模。 *arXiv*预印本 *arXiv:2306.11489*, 2023.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, 和 Yuan Cao。React: 在语言模型中协同推理和行动。*ArXiv*预印本, abs/2210.03629, 2022.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L Griffiths, Yuan Cao, 和 Karthik Narasimhan. 思维之树: 通过大型语言模型进行有意识的问题解决。 *arXiv*预印本 *arXiv:2305.10601*, 2023.

叶一宁，丛鑫，秦宇佳，林燕凯，刘知远，孙茂松。大型语言模型作为自主决策者。*arXiv* 预印本 *arXiv:2308.12519*, 2023.

庄宇辰，于悦，王宽，孙浩天，张超。Toolqa：带有外部工具的llm问题回答数据集。*arXiv*预印本 *arXiv:2306.13304*, 2023.

附录

实现细节

A.1 过滤快速API详细信息

我们进行了严格的过滤过程，以确保ToolBench的最终工具集可靠且功能正常。过滤过程如下：(1)初始测试：我们首先测试每个API的基本功能，以确定它们是否可操作。我们丢弃任何不符合这一基本标准的API；(2)示例响应评估：我们进行API调用以获取示例响应。然后我们通过响应时间和质量来评估它们的有效性。 那些一直表现出较长响应时间的API被省略。 此外，我们会过滤掉响应质量低的API，比如HTML源代码或其他错误信息。

A.2 API响应压缩

在检查每个API返回的响应时，我们发现一些响应可能包含冗余信息，过长无法输入LLMs。这可能会因为LLMs的有限上下文长度而导致问题。因此，我们进行响应压缩，减少API响应的长度同时保留关键信息。

由于每个API都有固定的响应格式，我们使用ChatGPT分析一个响应示例，并删除响应中不重要的键，以减少其长度。 ChatGPT的提示包含每个API的以下信息：（1）工具文档，包括工具名称、工具描述、API名称、API描述、参数和一个示例API响应。 这为ChatGPT提供了API功能的线索；(2) 3上下文学习示例，每个示例包含专家编写的原始API响应和压缩响应模式。 通过这种方式，我们获得了所有API的响应压缩策略。在推断过程中，当API响应长度超过 1024个标记时，我们通过删除不重要的信息来压缩响应。 如果压缩后的响应仍然超过 1024个标记，我们只保留前 1024个标记。 通过人工评估，我们发现这种压缩保留了API响应中包含的重要信息，并成功去除了噪音。

A.3 Details for Training ToolLLaMA

我们以多轮对话模式训练模型。 对于训练数据格式，我们保持输入和输出与ChatGPT相同。由于不清楚ChatGPT如何组织函数调用字段，我们只是将此信息连接到输入中，作为ToolLLaMA的提示的一部分。对于训练超参数，我们使用学习率为 5×10^{-5}，热身比率为 4×10^{-2}，总批量大小为 64，最大序列长度为 8192，并使用位置插值比率为 2。 我们对模型进行两轮训练，并选择在开发集上表现最佳的模型检查点，然后在测试集上进行评估。

A.4 D DFSDT的详细信息

在实践中，平衡效果与成本（OpenAI API调用次数）至关重要。
经典的DFS算法在每一步生成多个子节点，然后对所有子节点进行排序，并选择得分最高的节点进行扩展。 在贪婪地扩展到终端节点后，DFS会回溯以探索附近节点，扩展搜索空间。 在整个算法过程中，最消耗资源的部分是子节点的排序过程。 如果我们使用LLM一次评估两个节点，那么需要大约 $O(n \log n)$的OpenAI API调用复杂度，其中 n是子节点的数量。

事实上，我们经验性地发现，在大多数情况下，排名最高的节点通常是最先生成的节点。因此，我们跳过子节点的排序过程，选择树搜索的先序遍历（DFS的一种变体）。 这种设计具有以下优点：

- 如果模型不撤销某个操作（例如，对于简单指令的情况），那么DFSDT会退化为ReACT，使其与ReACT一样高效。

- 算法完成后，通过这种方法探索的节点几乎与经典DFS搜索找到的节点相同。 因此，它还可以处理只有DFS才能解决的复杂指令。

总体而言，这种设计在显著降低成本的同时实现了与DFS相似的性能。

还应注意，ReACT可以被视为DFSDT的退化版本。因此，尽管ToolLLaMA是在由DFSDT创建的数据上训练的，但在推断过程中模型可以通过ReACT或DFSDT使用。

A.5 工具评估细节

我们采用两种指标来评估自动工具使用能力：通过率和胜率。

通过率详细信息为了评估解决方案路径是否完成了原始指令中概述的任务并成功通过了它，我们首先需要考虑指令的可解性。 原则上，指令可以被分类为(1)可解的：例如，提供的工具中至少有一个在解决原始指令时有潜在帮助；或者(2)不可解的：例如，所有API与指令无关或指令提供无效信息，如无效的电子邮件地址。

要确定解决方案路径是否被视为通过，我们需要考虑指令是可解还是不可解。 在我们的评估中，每个解决方案路径可以被赋予三种类型的标签，即通过，失败和不确定。 具体来说，我们定义不同的规则如下：

如果指令是可解的：

1. 如果模型给出“放弃结束”类型，
   (a) 在尝试所有API并在收到API提供的信息后没有得到帮助时，解决方案被视为通过。
   (b) 如果模型只调用了少数API或从API中获得了有效信息，解决方案被视为失败。

2. 如果模型给出“最终答案结束”类型，
   (a) 如果API没有提供有效信息，并且模型已尝试所有API以检索有用信息，但最终答案仍未解决原始指令或传达拒绝（例如“对不起，因为工具不可用，我无法为您提供此信息”），解决方案被视为通过。
   (b) 如果工具提供有效信息，但最终答案并未完全解决指令或是一种拒绝，解决方案被视为失败。
   (c) 如果最终答案完全解决了原始指令，解决方案被视为通过。
   (d) 如果无法根据最终答案的内容确定指令是否已解决，则将解决路径视为不确定。

如果指令无法解决：

1. 如果模型给出“最终答案结束”类型，
   (a) 如果最终答案解决了最初被认为无法解决的指令，则将解决路径视为通过。
   (b) 如果最终答案是拒绝，则将解决路径视为通过。
   (c) 如果模型自己产生幻觉并提供错误的积极响应（例如“我已完成任务，最终答案是*”），则将解决路径视为失败。

2. 如果模型给出“放弃结束”类型，
   (a) 在这种情况下，将解决路径视为 通过。

对于每条解决路径，我们指示ChatGPT评估器生成多个（≥ 4）预测，并进行多数投票以得出最终通过率。

胜率详情由于通过率只衡量指令是否完成，而不是完成得有多好，我们采用另一个指标：胜率。 它是通过比较给定指令的两条解决路径来衡量的。 我们假设通过的候选者比失败的候选者更好，并且只比较那些ChatGPT评估器标记为“通过”或“失败”的解决路径。 请注意，与另一条解决路径相比，一条解决路径将被标记为以下之一：胜利，失败或平局。 我们为评估器的行为建立规则，以决定哪条解决路径更好，具体标准如下：

1. 信息丰富度：最终答案是否包含回答原始指令所需的所有必要信息。 更丰富的答案更好，而对于足以回答问题的相似丰富度则为平局。

2. 事实性：是否准确描述了最终做了什么，以及最终失败的原因。
   最终答案中更准确的描述更好。

3. 推理：如果查询仍未解决，是否提供了详细准确的失败原因。 更详细的原因更好。

4. 里程碑：计算执行过程中达到的里程碑数。

5. 探索：在执行过程中是否尝试了更多潜在有用的API。 使用更多的API更好。

6. 成本：如果使用的API数量相同，则具有较少重复（冗余）的API调用更好。

对于每个解决方案路径，我们还生成多个（≥ 4）预测，然后进行多数投票以得出最终胜率。 在表4中，为了便于阅读，我们将 `tie`的比率分成两部分，并分别添加到 `win`和 `lose`中。 在表6中，我们报告原始数字作为参考。

比较人类评估和ToolEval 为了验证ChatGPT评估器在通过率和胜率方面的可靠性，我们从四种不同方法（ChatGPT+ReACT、ChatGPT+DFSDT、ToolLLaMA+DFSDT和GPT4+DFSDT）中抽样，为每种方法的 300个测试指令获取解决方案对。 然后我们让人类对ChatGPT+DFSDT、ToolLLaMA+DFSDT和GPT4+DFSDT的通过率以及ChatGPT+ReACT和ChatGPT+DFSDT之间的胜率进行注释。我们的ChatGPT评估器在通过率方面与人类注释者达成高达87.1%的一致性，在胜率方面达到80.3%的一致性。 这个结果表明我们的评估器生成的评估结果与人类非常相似，可以被视为一个可信的评估器，模拟人类在通过率和胜率上的评估。

值得注意的是，工具学习的评估比传统任务如对话要复杂得多。 原因在于每个指令可能存在无限多个“正确”的解决路径。 在我们的初步调查中，令人惊讶的是，即使是人类专家在决定哪条解决路径更好时也经常意见不一，导致一致性相对较低。 例如，一个人可能更喜欢使用只有几个API的解决方案路径来快速得出最终答案；而另一个人可能更喜欢使用广泛尝试所有API来交叉验证特定信息的解决方案路径。 在这方面，我们认为在工具使用领域的公平评估仍有很长的路要走，我们相信这项工作为此铺平了道路。我们期待更多未来的工作来探索这个有趣的研究问题。

A.6 DAPIB ENCH上的实验细节

将ToolLLaMA推广到APIBench时，没有对ToolLLaMA进行任何训练更新，而是将提示中的每个API视为一个函数调用。 我们定义一个函数，代表选择一个API，提供调用它的代码，并用自然语言描述生成的输出。 我们不考虑APIBench的零-shot设置，即提示中不包含任何API描述，因为在训练过程中从未遇到过来自三个测试领域的API。

模型	方法	I1-Inst.		I1-Tool		I1-Cat.		I2-Inst.		I2-Cat.		I3-Inst.		平均	
		胜利	平局	胜利	平局	胜利	平局	胜利	平局	胜利	平局	胜利	平局		
ChatGPT	DFSDT	52.5	16.0	55.0	14.0	47.5	19.5	**67.0**	10.0	**58.5**	12.5	61.0	16.0	56.9	14.7
Claude-2	ReACT	27.0	8.0	24.0	7.5	29.5	8.5	32.0	6.0	28.5	6.0	43.0	9.5	30.7	7.5
	DFSDT	34.0	8.0	41.0	6.5	39.5	7.5	32.5	9.5	33.5	0.0	65.0	0.0	40.8	5.3
Text-Davinci-003	ReACT	23.5	10.0	28.5	13.5	27.0	8.0	26.5	6.5	25.5	8.5	41.0	8.0	28.7	9.1
	DFSDT	35.0	10.5	37.5	12.5	40.0	13.5	36.5	8.0	40.0	6.5	60.0	6.0	41.5	9.5
GPT4	ReACT	52.5	15.0	53.5	10.5	56.0	15.0	59.5	12.5	52.5	15.5	76.0	4.0	58.3	12.1
	DFSDT	**60.5**	14.0	**62.5**	10.5	**58.0**	17.0	**67.0**	12.5	57.0	12.5	**80.0**	8.0	**64.2**	12.4
Vicuna	(ReACT & DFSDT)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Alpaca	(ReACT & DFSDT)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
ToolLLaMA	ReACT	40.0	10.0	36.5	11.0	42.0	11.0	45.5	10.5	37.5	8.5	51.0	8.0	42.1	9.8
	DFSDT	48.5	13.0	50.5	9.5	49.5	10.0	62.5	12.0	52.0	12.0	68.0	2.0	55.2	9.8
	检索器	58.0	8.5	54.5	9.0	51.0	8.0	64.5	8.0	56.0	9.5	71.0	4.0	59.2	7.8

表6：在合并平局标签之前的胜率结果。 胜率是通过将每个模型与ChatGPT-ReACT进行比较来计算的。胜率高于 50%意味着该模型的性能优于ChatGPT-ReACT。除了ToolLLaMA-DFSDT-Retriever之外，所有方法都使用了oracle API检索器（即，地面真实API）。

A.7 P用于指令生成的提示

下面我们列出了详细的指导生成提示，包括四个部分：任务描述，上下文学习示例，抽样的API列表和其他要求。

单工具说明任务描述：
您将获得一个工具，它的描述，所有可用的API函数，这些API函数的描述以及每个API函数所需的参数。 您的任务是创建10个不同、创新和详细的用户查询，使用工具的多个API函数。 例如，如果工具‘气候新闻’有三个API调用 - ‘获取所有气候变化新闻’，‘查看今天的气候’和‘历史气候’，您的查询应该表达类似于：首先，确定今天的天气，然后验证九月份俄亥俄州下雨的频率，最后，找到关于气候变化的新闻，帮助我了解气候是否会很快改变。

这个查询示范了如何利用‘气候新闻’的所有API调用。 只使用一个API调用的查询将不被接受。 此外，您必须包含每个API调用所需的输入参数。 为了实现这一点，为所需参数生成随机信息，如IP地址、位置、坐标等。例如，不要仅仅说‘一个地址’，提供确切的道路和区域名称。 不要仅仅提到‘一个产品’，而是具体指明可穿戴设备、牛奶、一条蓝色毯子、一个平底锅等。不要提到‘我的公司’，而是创造一个公司名称。 前十个查询中的前七个应该非常具体。 每个单独的查询应以不同方式结合所有API调用的使用，并包括必要的参数。 请注意，您不应该问‘使用哪个API’，而是简单陈述这些API可以解决的需求。您还应避免询问API调用所需的输入参数，而是直接在查询中提供参数。

最后三个查询应该是复杂且冗长的，描述一个复杂的场景，在这个场景中所有的API调用都可以被利用来提供帮助。 你应该首先考虑可能的相关API组合，然后提出你的查询。 相关API是可以用于给定查询的API；这些相关API必须严格来自提供的API名称。 对于每个查询，应该有多个相关API；对于不同的查询，相关API的重叠应尽可能少。 以这种格式提交你的回答：[查询1: ....., ‘相关API’:[api1, api2, api3...],查询2: ....., ‘相关API’:[api4, api5, api6...],查询3: ....., ‘相关API’:[api1, api7, api9...], ...]

多工具说明任务描述:
你将会被提供几个工具、工具描述，每个工具的所有可用API函数，这些API函数的描述，以及每个API函数所需的参数。 您的任务包括创建10个多样化、创新和详细的用户查询，利用多个工具的API函数。 例如，给定三个工具‘nba news’、‘cat-facts’和‘hotels’：‘nba news’具有API函数‘获取单个NBA新闻源’和‘获取所有NBA新闻’，‘cat-facts’具有API函数‘获取有关猫的所有事实’和‘获取一个关于猫的随机事实’，‘hotels’具有API函数‘properties/get-details (已弃用)’、‘properties/list (已弃用)’和‘locations/v3/search’。 您的查询应该表达类似于：‘我想给我的新生猫取名科比，并主持一个

派对来庆祝它的诞生。 给我一些猫的事实和NBA新闻，以获取灵感为猫取名。 此查询示例说明如何利用所有给定工具的API调用，在休斯顿市中心找到我家附近合适的酒店举办派对。 只使用一个工具的API调用的查询将不被接受。 此外，您必须包含每个API调用所需的输入参数。 为了实现这一点，为所需参数生成随机信息，如IP地址、位置、坐标等。例如，不要仅仅说‘一个地址’，提供确切的道路和区域名称。 不要仅仅提到‘一个产品’，而是具体指定可穿戴设备、牛奶、一条蓝色毯子、一口锅等。不要提到‘我的公司’，而是想出一个公司名称。 前十个查询中的前七个应该非常具体。 每个单独的查询应以各种方式结合不同工具的API调用，并包括必要的参数。 请注意，您不应该问‘要使用哪个API’，而是简单陈述这些API可以解决的需求。您还应避免询问API调用所需的输入参数，而是直接在查询中提供参数。

最后三个查询应该是复杂且冗长的，描述一个复杂的情景，在这个情景中，所有提供的API调用都可以被利用来在一个查询中提供帮助。 你应该首先考虑可能的相关API组合，然后提出你的查询。 相关API是可以用于给定查询的API；这些相关API必须严格来自提供的API名称。 对于每个查询，应该有多个相关API；对于不同的查询，相关API的重叠应尽可能少。 以这种格式提交你的回答：[查询1: ....., ‘相关API’:[[工具名称，API名称]，[工具名称，API名称]，[工具名称，API名称]...]，查询2: ....., ‘相关API’:[[工具名称，API名称]，[工具名称，API名称]，[工具名称，API名称]...]，查询3: ....., ‘相关API’:[[工具名称，API名称]，[工具名称，API名称]，[工具名称，API名称]...]，...]

上下文种子示例。 接下来，我们展示一个单工具指令种子示例和一个多工具指令种子示例。

例如，使用ASCII艺术工具，给定的API名称为‘figlet’、‘list figlet styles’、‘cowsay’、‘list cowsay styles’、‘matheq’。
一些示例查询和相关API可能是：
“查询”：“需要创建一个数学方程的ASCII艺术表示。 方程是‘y = mx + c’，其中m和c是常数。 帮我生成这个方程的ASCII艺术。
还请生成文本‘牛顿第二运动定律’的ASCII艺术表示。",“相关API”:[‘figlet’，‘列出figlet样式’，‘matheq’]
“查询”：“正在研究关于牛的论文，需要包含各种牛的ASCII艺术表示。 您能先检索可用的牛ASCII艺术风格吗？ 那么，你能为像泽西牛、荷斯坦牛和根西牛这样的牛生成ASCII艺术吗？ 最后，我希望这头牛在ASCII艺术中说‘哞！’",“相关API”:[’figlet’，‘列出figlet样式’，‘cowsay’，‘列出cowsay样式’]“查询”：“我正在撰写一篇关于ASCII艺术的博文，需要包含一些示例。 你能为以下字符串生成ASCII艺术吗：‘ASCII’、‘艺术’和‘画廊’？ 你可以首先检索可用的figlet样式，然后使用这些样式为字符串生成ASCII艺术。",“相关API”:[’figlet’，‘列出figlet样式’]

“查询”：“问候！我正在制作一个有关我们毛茸茸朋友的古怪幻灯片，需要你的帮助来添加一些ASCII艺术的元素。 你能帮我获取可用于动物的ASCII艺术样式目录吗？ 此外，我特别想展示熊猫、牛、大象和企鹅等生物的ASCII艺术。 如果它们能够像ASCII艺术一样说一些可爱的话，比如‘你好！’或‘拥抱！’，那就太完美了！”，“相关API”:[’figlet’，‘列出figlet样式’，‘cowsay’，‘列出cowsay样式’]

例如，使用工具[’创业者心态收集’，‘随机单词’，‘thedigitalnews-feederapi’，‘化学元素’]，给定的API名称为（工具‘创业者心态收集’）‘以JSON格式获取随机引用’，（工具‘随机单词’）‘获取多个随机单词’，（工具‘随机单词’）‘获取一个随机单词’，（工具‘thedigitalnewsfeederapi’）‘获取特定的板球文章’，（工具‘thedigitalnewsfeederapi’）‘获取板球文章’，（工具‘thedigitalnewsfeederapi’）‘获取特定的新闻文章’，（工具‘thedigitalnewsfeederapi’）‘获取新闻文章’，（工具‘thedigitalnewsfeederapi’）‘获取所有新闻文章’，（工具‘化学元素’）‘获取所有化学元素
一些示例查询和相关API可能是：
“查询”：“为我最好朋友的惊喜生日派对，我需要一些关于派对游戏和装饰的灵感。 请推荐一些可以作为派对主题的随机词汇。 此外，我对收集关于最新派对趋势的新闻文章很感兴趣，以确保现代化的庆祝活动。

另外，我希望了解我所在地区的当地酒店的详细信息，以供选择住宿。 非常感谢您的帮助。”，“相关API”: [[’ 随机词汇‘，‘获取多个随机词汇’]，[’数字新闻提供API’，‘获取新闻文章’]，[’ 数字新闻提供API’，‘获取所有新闻文章’]]

“查询”：“在组织一场为我尊敬的公司举办的团队建设活动中，我急切地寻求您宝贵的意见来激励活动。 我可以请求一些体现团队合作和激励精神的随机引语集合吗？ 此外，我热衷于探索展示成功团队建设活动的新闻文章，因为它们是灵感的源泉。”，“相关API”: [[’企业家心态收集’，‘JSON格式的随机引语’]，[’数字新闻提供API’，‘获取新闻文章’]] “查询”：“我需要关于运动对健康益处的具体板球文章，以供我在运动研究论文中使用。 我也想知道与锻炼有关的化学元素，比如增加的铁（Fe）及其对骨髓的影响。", “相关API”: [[’thedigitalnewsfeederapi’，‘获取特定板球文章’], [’ 化学元素’，‘获取所有化学元素’]]

-

“查询”：“我正在开始一项新的商业冒险，我需要发表一篇宣布新时代的演讲。 给我一些引言和词语，让我开始。 我想收集关于成功企业家的新闻文章以获得灵感。", “相关API”: [[’企业家心态收集’，‘JSON格式的随机引言’], [’随机词语’，‘获取多个随机词语’], [’thedigital- newsfeederapi’，‘获取特定新闻文章’]]

这些只是示例，展示如何编写查询。 不要使用上述示例中列出的API，而是使用下面输入中列出的API。

抽样的*API*列表(一个例子)


```
{
    "tool_description": "EntreAPI Faker用于动态创建模拟、演示、测试
       和样本数据", "name": "EntreAPI Faker", "api_list": [


        {
            "name": "经度",
            "url": "https://entreapi-faker.p.rapidapi.com/address/
               longitude",
            "description": "生成随机经度。",
            "method": "GET",
            "required_parameters": [],
            "optional_parameters": [
                {
                    "name": "max",
                    "type": "NUMBER",
                    "description": "纬度的最大值。",
                    "default": ""
                },
                {
                    "name": "min",
                    "type": "NUMBER",
                    "description": "纬度的最小值。",
                    "default": ""
                },
                {
                    "name": "precision",
                    "type": "NUMBER",
                    "description": "纬度的精度。",
                    "default": ""
                }
            ],
            "tool_name": "EntreAPI Faker",
            "category_name": "数据"
```

```
},
{
    "name": "布尔值",
    "url": "https://entreapi-faker.p.rapidapi.com/datatype
       /boolean",
    "description": "随机生成一个布尔值。",
    "method": "GET",
    "required_parameters": [],
    "optional_parameters": [],
    "tool_name": "EntreAPI Faker",
    "category_name": "数据"
},
{
    "name": "过去",
    "url": "https://entreapi-faker.p.rapidapi.com/date/
       past",
    "description": "随机生成过去的日期值。", "method": "GET", "
       require
    d_parameters": [
    ], "optional_parameters": [

        {
            "name": "refDate",
            "type": "字符串",
            "description": "起始参考日期",
            "default": ""
        },
        {
            "name": "年份",
            "type": "数字",
            "描述": "日期范围的年数。", "默认": ""


        }
    ],
    "tool_name": "EntreAPI Faker",
    "category_name": "数据"
},
{
    "name": "图片网址",
    "url": "https://entreapi-faker.p.rapidapi.com/image/
       imageUrl",
    "描述": "随机生成一个图片URL。", "方法": "GET", "必需参数
    ": [], "可选参数":
    [

        {
            "name": "width",
            "type": "NUMBER",
            "description": "图像的宽度。 默认为
                        640。",
            "default": ""
        },
        {
            "name": "height",
            "type": "NUMBER",
            "description": "图像的高度。 默认
               为480。",
            "default": ""
```

```
            },
            {
                "name": "useRandomize",
                "type": "BOOLEAN",
                "description": "在返回的URL中添加一个随机数参数。"
                   , "default": ""

            },
            {
                "name": "category",
                "type": "STRING",
                "description": "图像的类别。
                   可以是以下之一：抽象、动物、头像、商业、猫、
                   城市、时尚、食品、自然、夜生活、人物、运动
                   、技术、交通", "default": ""


            }
        ],
        "tool_name": "EntreAPI Faker",
        "category_name": "数据"
    },
    {
        "name": "Sentence",
        "url": "https://entreapi-faker.p.rapidapi.com/lorem/
           sentence",
        "description": "随机生成一句Lorem Ipsum。", "method": "GE
           T", "req
        uired_parameters
        ": [], "optional_parameters
        ": [
            {
                "name": "wordCount",
                "type": "NUMBER",
                "description": "句子中的单词数。", "defau
                   lt": ""

            }
        ],
        "tool_name": "EntreAPI Faker",
        "category_name": "数据"
    },
    {
        "name": "性别",
        "url": "https://entreapi-faker.p.rapidapi.com/name/
           gender",
        "description": "随机选择一个性别。",
        "method": "GET",
        "required_parameters": [],
        "optional_parameters": [
            {
                "name": "useBinary",
                "type": "BOOLEAN",
                "description": "仅使用二进制性别。",
                "default": ""
            }
        ],
        "tool_name": "EntreAPI Faker",
        "category_name": "数据"
```

```
},
{
    "name": "前缀",
    "url": "https://entreapi-faker.p.rapidapi.com/name/
       prefix",
    "描述": "随机生成一个前缀（例如，先生，夫人等）。", "方法": "GE
        T", "必需参数"
    : [], "可选参数": [


        {
            "name": "gender",
            "type": "STRING",
            "description": "可选性别。",
            "default": ""
        }
    ],
    "tool_name": "EntreAPI Faker",
    "category_name": "数据"
},
{
    "name": "数组元素",
    "url": "https://entreapi-faker.p.rapidapi.com/random/
       arrayElement",
    "描述": "随机选择一个数组元素。",
    "方法": "GET",
    "必需参数": [],
    "可选参数": [
        {
            "名称": "数组",
            "类型": "数组",
            "描述": "要选择的元素列表。 默认为 [\"a\", \"b\", \"c
                                              \"].", "默认": ""

        }
    ],
    "tool_name": "EntreAPI Faker",
    "category_name": "数据"
},
{
    "名称": "数字值",
    "网址": "https://entreapi-faker.p.rapidapi.com/random/
       number",
    "描述": "随机生成一个数字值。",
    "方法": "GET",
    "必需参数": [],
    "可选参数": [
        {
            "name": "min",
            "type": "NUMBER",
            "描述": "最小值。",
            "默认": ""
        },
        {
            "name": "max",
            "type": "NUMBER",
            "描述": "最大值。",
            "默认": ""
        },
```

```
                {
                    "name": "precision",
                    "type": "NUMBER",
                    "描述": "数字的精度。",
                    "默认": ""
                }
            ],
            "tool_name": "EntreAPI Faker",
            "category_name": "数据"
        },
        {
            "名称": "网址",
            "网址": "https://entreapi-faker.p.rapidapi.com/internet
                /url",
            "描述": "随机生成一个网址。",
            "方法": "GET",
            "必需参数": [],
            "可选参数": [],
            "工具名称": "EntreAPI Faker",
            "类别名称": "数据"
        }
    ]
}
```

其他要求：
请根据给定的要求和输入生成十个查询。 这十个查询应该展示多样的句子结构：一些查询应该是祈使句，其他是陈述句，还有一些是疑问句。 同样，它们应该包含各种语气，有些礼貌，有些直接。 确保它们在长度上有所变化，并包含各种主题：我自己，我的朋友，家人和公司。 目标是包含一些引人入胜的查询，只要它们与API调用相关。 请记住，对于每个查询，仅调用一个API是不够的；每个查询应调用两到五个API。但是，请尽量避免在查询中明确指定要使用哪个API。 每个查询应至少包含三十个词。

A.8 PROMPTS FOR SOLUTION PATH ANNOTATION

在搜索解决方案路径时，我们使用以下提示。 在扩展子节点时，我们使用多样性用户提示，显示先前子节点的信息。

```
-------------------------------------------------------------------
system_prompt:
您是Tool-GPT，能够利用众多工具和功能完成给定任务。

1.首先，我会为您提供任务描述，然后您的任务将开始。

2.在每个步骤中，您需要分析当前状态，并通过执行函数调用确定下一步操作
    。

3.在调用之后，您将收到结果，转移到新状态。 随后，您将分析当前状态，做出关于
    下一步的决定，并重复此过程。


4.经过几次思考和函数调用的迭代后，您最终将完成任务并提供最终答案。

记住：
1.状态更改是不可逆的，您无法返回到先前状态。
```

2.保持思维简洁，限制在最多五句话内。

3.您可以进行多次尝试。 如果您计划连续尝试不同条件，请每次尝试一个条件。 4.
如果您认为已经收集足够的信息，请调用函数"Finish: give_answer"
来提供您对任务的答案。

5.如果您觉得无法处理从这一步开始的任务，请调用函数"Finish: give_up_and_r
estart"。
让我们开始吧!
任务描述：{task_description}
---
diversity_user_prompt:
这不是您第一次尝试这个任务，所有以前的尝试都失败了。

在您为这个状态生成想法之前，我将首先向您展示这个状态的以前的操作，然后您必须生
成与所有这些操作都不同的操作。 这里有一些以前的操作候选人：{previous_
candidate}

请记住，您现在处于一次尝试的中间状态，您将首先分析当前状态和以前的操作候选
人，然后执行与所有以前操作不同的操作。

Finish_function_description:
```
{
    "name": "Finish",
    "描述": "如果您认为已经得到了可以回答任务的结果，请调用此函数提供最终答案
        。 或者，如果您意识到在当前状态下无法继续执行任务，请调用此函数重
        新启动。 记住：您必须始终在尝试结束时调用此函数，用户将看到的唯一部
        分是最终答案，因此它应包含足够的信息。", "参数": { "类型": "
        对象", "属性": { "返回类型": { "类型": "字符串",
                    "枚举": ["给出答案","放弃并重新开始"],
                },
                "最终答案": {
                    "类型": "字符串",
                    "描述": "您想要给用户的最终答案。 如果 \"返回类型\"==\"给
                        出答案\"，则应该有此字段",

                }
            },
            "必需": ["返回类型"],
    }
}
```