# Reverse Chain: A Generic-Rule for LLMs to Master Multi-API Planning

张颖而

# 目录

# 项目背景：Tool-Augmented LLMs

- **动机：**LLMs 本身能力有限，不具备某些专业的功能（如计算器，天气预报等），因此能实现的任务也很有限。如果LLMs 具备使用工具的能力，将极大的拓展 LLMs 处理任务的范围。
- **用法：**LLMs 作为 controllers:

1. 理解用户意图；2. 选择并使用正确的工具来完成任务。

- **Example:**

QUERY: What's the weather in New York ?

API PLANNING: getWearther (city='New York')

# 动机：任务的分类

有以下三种任务类型：相比于single-tool, multi-tool 任务更难，其中**compositional multi-tool**是最困难的，因为推理路径较长，且存在多层嵌套。

| Task Type | Example | API planning |
|---|---|---|
| Single-tool | What's the weather in New York ? | *getWearther*(city='New York') |
| Independent multi-tool | What's the weather in New York? When's my next meeting? | *getWearther*(city='New York') *showCalendar*(event='next meeting') |
| Compositional multi-tool | I'm Lucas, Could you find a flight and book it to my destination ? | *BookFlight*(flight_ID=*FindFlight*(destination =*GetUserDestination*(userName='Lucas')) |

Finetuned or In-context learning?

In-context learning 方案更直观更简单，而且更容易被泛化到新增API上。

# 动机：现存方法的缺陷

***BookFlight(flight_ID = FindFlight (destination = GetUserDestination(username = 'Lucas'))***

- CoT Planning/ ReAct

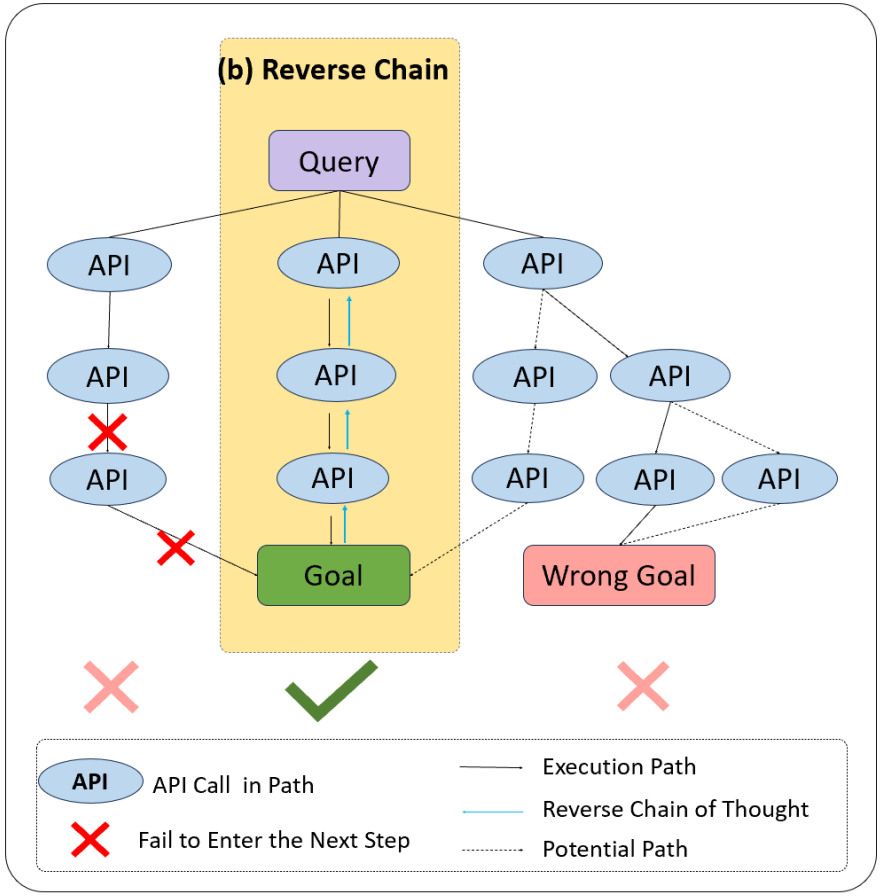
缺点1：**step之间未必可执行 (存在参数编造)**

*BookFlight (flight_ID = FindFlight (='NewYork'))*

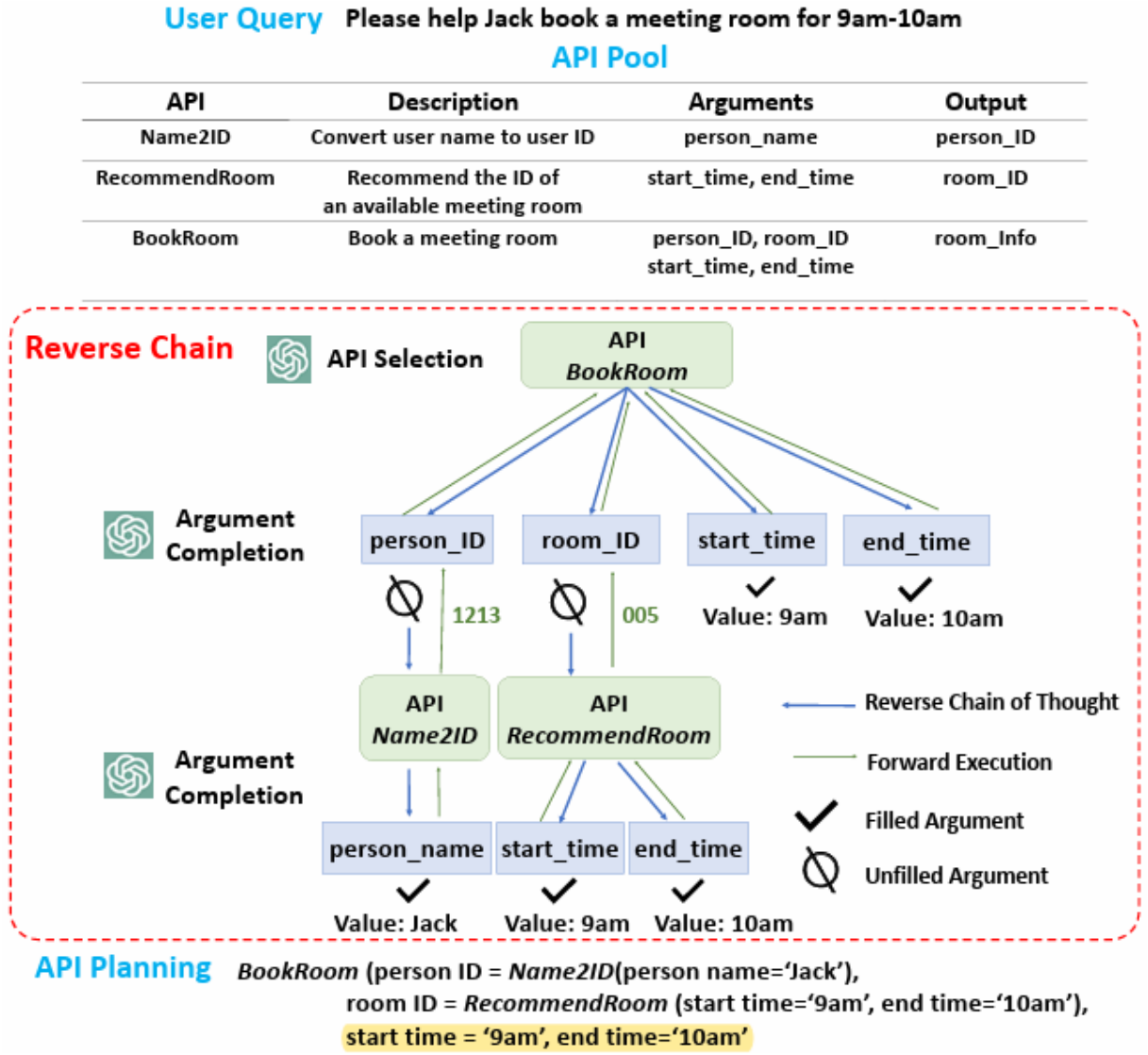
缺点2：偏离最终目标*destination*

*GetUserDestination(userName='Lucas')->destination,*

*flight_ID=FindFlight(destination)->FinalAnswer*



# 方案: Reverse Chain




- **Backward Reasoning**
从最终意图出发推理，保证了最终目标不发生偏移，且路径是唯一的。

- **Generic rule + Decomposed task**
把复杂的API planning 任务拆分成两个处理自然语言的简单任务：API Selection + Argument Completion；

且填写arguments 的交互能够大大减少模型参数填写错误的概率。

# 方案：LLM Modules in Reverse Chain

(a) API Selection

We have N APIs:
=====
{"name": BookRoom, "description": Book a meeting room}
.....
{"name": Weather, "description": Query weather}
=====
If someone is saying: "Please help Jack book a meeting room for 9:00-10:00"
Which final API should we use for this instruction? Only return API code. Only return one word!

- **API selection**

LLM 根据PI的功能描述选择API

- **Argument Completion**

根据 argument的描述和数据类型，选择：
1. 直接从query中抽取value
2. 返回能够生成该argument value的API name
3. None, 触发反问机制

(b) Argument Completion

You are an argument extractor. For each argument, you need to determine whether you can extract the value from user input directly or you need to use an API to get the value. The output should be in Json format, key is the argument, and value is the value of argument or the API name, return None if you cannot get value or API name.

The Arguments to be extracted are:
person_ID: {"description": person's employee ID, "type": Integer}
room_ID: {"description": person's employee ID, "type": Integer}
start_time: {"description": start time of meeting, "type": Time}
end_time: {"description": end time of meeting, "type": Time}

The API you can use includes:
{"name": RecommendRoom, "description": Recommend the ID of an available meeting room}
.....
Now, Let's start.
=>
If someone is saying: "Please help Jack book a meeting room for 9am-10am"
Arguments :

# 实验效果

| Method | level 1 | level 2 | level 3 | Overall |
|---|---|---|---|---|
| Zero-Shot | 72.06 | 67.68 | 42.37 | 68.97 |
| Few-Shot | 86.46 | 77.48 | 71.18 | 81.87 |
| Zero-Shot-CoT | 82.45 | 81.38 | 57.62 | 81.29 |
| Few-Shot-CoT | 89.72 | 85.71 | 66.10 | 87.16 |
| ReAct | 72.68 | 69.11 | 45.76 | 70.06 |
| **Reverse Chain** | **93.99** | **90.33** | **86.44** | **92.06** |

- **Experiment setting:**
  - **Dataset:**

    通过GPT-4和ChatGPT自动构造+人工过滤；
    包含涉及20个不同领域的825个不同的API，共1550条样本；
    样本包含2,3,4,5 层的嵌套。平均每个样本包含2.93个 function calls。
  - **Metrics:**

    主要评测API的选择， API 参数值，API 之间的调用关系是否正确；
    针对不同baseline methods 的设计不同的GPT-4.0 自动评测方案，抽样得到89%的人工一致性。

- **Results:**

  以ChatGPT作为基座LLM，Reverse chain 相比于其他方法都有显著的提升，并且在嵌套层数多的时候提升更加明显。

# 参考文献

[1] Hugging-gpt: Solving ai tasks with chatgpt and its friends in huggingface.

[2] Taskmatrix.ai: Completing tasks by connecting foundation models with millions of apis.

[3] Chain-of-thought prompting elicits reasoning in large language models.

[4] React: Synergizing reasoning and acting in language models.

[5] Rest-gpt: Connecting large language models with real-world applications via restful apis.

[6] Tptu: Task planning and tool usage of large language model-based ai agents.