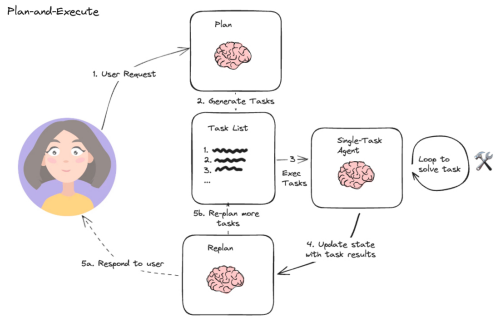
Plan-and-Execute 计划与执行 ¶

This notebook shows how to create a "plan-and-execute" style agent. This is heavily inspired by the Plan-and-Solve paper as well as the Baby-AGI project.
本笔记本展示了如何创建“计划并执行”风格的代理。这很大程度上受到了 Plan-and-Solve 论文以及 Baby-AGI 项目的启发。

The core idea is to first come up with a multi-step plan, and then go through that plan one item at a time. After accomplishing a particular task, you can then revisit the plan and modify as appropriate.
核心思想是首先制定一个多步骤计划，然后一次一项地执行该计划。完成特定任务后，您可以重新审视计划并进行适当修改。

The general computational graph looks like the following:
一般的计算图如下所示：



This compares to a typical ReAct style agent where you think one step at a time. The advantages of this "plan-and-execute" style agent are:
这与典型的 ReAct 风格代理相比，您一次思考一步。这种“计划和执行”风格代理的优点是：

1. Explicit long term planning (which even really strong LLMs can struggle with)
   明确的长期规划（即使是非常强大的LLMs也可能会遇到困难）
2. Ability to use smaller/weaker models for the execution step, only using larger/better models for the planning step
   能够在执行步骤中使用较小/较弱的模型，仅在规划步骤中使用较大/更好的模型

The following walkthrough demonstrates how to do so in LangGraph. The resulting agent will leave a trace like the following example: (link).
以下演练演示了如何在 LangGraph 中执行此操作。生成的代理将留下如下例所示的痕迹：（链接）。

Setup 设置

First, we need to install the packages required.
首先，我们需要安装所需的软件包。

In [1]: 在[1]中:
```
%%capture --no-stderr
%pip install --quiet -U langchain-community langchain-openai tavily-pyt
```

Next, we need to set API keys for OpenAI (the LLM we will use) and Tavily (the search tool we will use)
接下来，我们需要为 OpenAI（我们将使用的 LLM）和 Tavily（我们将使用的搜索工具）设置 API 密钥

In [1]: 在[1]中:
```
import os
import getpass


def _set_env(var: str):
    if not os.environ.get(var):
        os.environ[var] = getpass.getpass(f"{var}: ")


_set_env("OPENAI_API_KEY")
_set_env("TAVILY_API_KEY")
```

Optionally, we can set API key for LangSmith tracing, which will give us best-in-class observability.
或者，我们可以为 LangSmith 跟踪设置 API 密钥，这将为我们提供一流的可观察性。

In [2]: 在[2]中:
```
os.environ["LANGCHAIN_TRACING_V2"] = "true"
_set_env("LANGCHAIN_API_KEY")
os.environ["LANGCHAIN_PROJECT"] = "Plan-and-execute"
```

Define Tools 定义工具

We will first define the tools we want to use. For this simple example, we will use a built-in search tool via Tavily. However, it is really easy to create your own

tools - see documentation here on how to do that.
我们将首先定义我们想要使用的工具。对于这个简单的示例，我们将通过 Tavily 使用内置搜索工具。然而，创建自己的工具确实很容易 - 请参阅此处的文档了解如何做到这一点。

In [3]: 在[3]中:
```
from langchain_community.tools.tavily_search import TavilySearchResults

tools = [TavilySearchResults(max_results=3)]
```

Define our Execution Agent
定义我们的执行代理

Now we will create the execution agent we want to use to execute tasks. Note that for this example, we will be using the same execution agent for each task, but this doesn't HAVE to be the case.
现在我们将创建要用来执行任务的执行代理。请注意，对于此示例，我们将为每个任务使用相同的执行代理，但情况并非必须如此。

In [5]: 在[5]中:
```
from langchain import hub
from langchain_openai import ChatOpenAI
from langgraph.prebuilt import create_react_agent

# Get the prompt to use - you can modify this!
prompt = hub.pull("wfh/react-agent-executor")
prompt.pretty_print()

# Choose the LLM that will drive the agent
llm = ChatOpenAI(model="gpt-4-turbo-preview")
agent_executor = create_react_agent(llm, tools, messages_modifier=promp
```

```
================================ System Message ================================

You are a helpful assistant.

============================= Messages Placeholder =============================

{{messages}}
```

In [7]: 在[7]中:
```
agent_executor.invoke({"messages": [("user", "who is the winnner of the
```

Out[7]: 输出[7]:
```
{'messages': [HumanMessage(content='who is the winnner of the us open',
id='7c491c9f-cdbe-4761-b93b-3e4eeb526c97'),
  AIMessage(content='', additional_kwargs={'tool_calls': [{'id': 'call_
MMmwmxwxRH2hrmMbuBeMGsXW', 'function': {'arguments': '{"query":"US Open
2023 winner"}', 'name': 'tavily_search_results_json'}, 'type': 'functio
n'}]}, response_metadata={'token_usage': {'completion_tokens': 23, 'pro
mpt_tokens': 97, 'total_tokens': 120}, 'model_name': 'gpt-4-turbo-previ
ew', 'system_fingerprint': None, 'finish_reason': 'tool_calls', 'logpro
bs': None}, id='run-855f7cff-62a2-4dd8-b71b-707b507b00a4-0', tool_calls
=[{'name': 'tavily_search_results_json', 'args': {'query': 'US Open 202
3 winner'}, 'id': 'call_MMmwmxwxRH2hrmMbuBeMGsXW'}]),
  ToolMessage(content='[{"url": "https://www.bbc.com/sport/tennis/66766
337", "content": ": Stephen Nolan goes in to find out\\nRelated Topics
\\nTop Stories\\nTen Hag on Rashford plus transfer news, WSL deadline d
ay\\nSpinner Leach doubtful for second Test in India\\nMcIlroy \'change
s tune\' on LIV players\' punishment\\nElsewhere on the BBC\\nDiscover
the tropical paradise of Thailand\\nFrom the secrets of the South to th
e mysterious North...\\n Djokovic offered to help up Medvedev when the
Russian fell to the court in the third set\\nDjokovic\'s relentless ret
urning continued to draw mistakes out of Medvedev, who was serving poor
ly and making loose errors, at the start of the second set.\\n It was c
lear to see Medvedev had needed to level by taking that second set to s
tand any real chance of victory and the feeling of the inevitable was h
eightened by the Russian needing treatment on a shoulder injury before
the third set.\\n Djokovic shows again why he can never be written off
\\nWhen Djokovic lost to 20-year-old Carlos Alcaraz in the Wimbledon fi
nal it felt like a changing-of-the-guard moment in the men\'s game.\\n
The inside story of Putin\\u2019s invasion of Ukraine\\nTold by the Pre
sidents and Prime Ministers tasked with making the critical decisions
\\nSurvival of the wittiest!\\n"}, {"url": "https://www.usopen.org/en_U
S/news/articles/2023-09-10/novak_djokovic_wins_24th_grand_slam_singles_
title_at_2023_us_open.html", "content": "WHAT HAPPENED: Novak Djokovic
handled the weight of history to defeat Daniil Medvedev on Sunday in th
e 2023 US Open men\'s singles final. With a 6-3, 7-6(5), 6-3 victory, t
he 36-year-old won his 24th Grand Slam singles title, tying Margaret Co
urt\'s record and bolstering his case to be considered the greatest ten
nis player of all time."}, {"url": "https://apnews.com/article/us-open-
final-live-updates-djokovic-medvedev-8a4a26f8d77ef9ab2fb3efe1096dce7e",
"content": "Novak Djokovic wins the US Open for his 24th Grand Slam tit
le by beating Daniil Medvedev\\nNovak Djokovic, of Serbia, holds up the
championship trophy after defeating Daniil Medvedev, of Russia, in the
men\\u2019s singles final of the U.S. Open tennis championships, Sunda
y, Sept. 10, 2023, in New York. (AP Photo/Manu Fernandez)\\nDaniil Medv
edev, of Russia, sits on the court after a rally against Novak Djokovi
c, of Serbia, during the men\\u2019s singles final of the U.S. Open ten
nis championships, Sunday, Sept. 10, 2023, in New York. (AP Photo/Manu
Fernandez)\\nDaniil Medvedev, of Russia, sits on the court after a rall
y against Novak Djokovic, of Serbia, during the men\\u2019s singles fin
al of the U.S. Open tennis championships, Sunday, Sept. 10, 2023, in Ne
w York. (AP Photo/Manu Fernandez)\\nDaniil Medvedev, of Russia, sits on
the court after a rally against Novak Djokovic, of Serbia, during the m
en\\u2019s singles final of the U.S. Open tennis championships, Sunday,
Sept. 10, 2023, in New York. Novak Djokovic, of Serbia, reveals a t-shi
rt honoring the number 24 and Kobe Bryant after defeating Daniil Medved
ev, of Russia, in the men\\u2019s singles final of the U.S. Open tennis
championships, Sunday, Sept. 10, 2023, in New York."}]', name='tavily_s
earch_results_json', id='ca0ff812-6c7f-43c1-9d0e-427cfe8da332', tool_ca
ll_id='call_MMmwmxwxRH2hrmMbuBeMGsXW'),
  AIMessage(content="The winner of the 2023 US Open men's singles was N
ovak Djokovic. He defeated Daniil Medvedev with a score of 6-3, 7-6(5),
6-3 in the final, winning his 24th Grand Slam singles title. This victo
ry tied Margaret Court's record and bolstered Djokovic's claim to be co
nsidered one of the greatest tennis players of all time.", response_met
adata={'token_usage': {'completion_tokens': 89, 'prompt_tokens': 972,
'total_tokens': 1061}, 'model_name': 'gpt-4-turbo-preview', 'system_fin
gerprint': None, 'finish_reason': 'stop', 'logprobs': None}, id='run-ef
37a655-1ea6-470e-a310-8f125ca48015-0')]}
```

Define the State 定义状态

Let's now start by defining the state the track for this agent.
现在让我们开始定义该代理的跟踪状态。

First, we will need to track the current plan. Let's represent that as a list of strings.
首先，我们需要跟踪当前的计划。让我们将其表示为字符串列表。

Next, we should track previously executed steps. Let's represent that as a list of tuples (these tuples will contain the step and then the result)
接下来，我们应该跟踪之前执行的步骤。让我们将其表示为元组列表（这些元组将包含步骤，然后包含结果）

Finally, we need to have some state to represent the final response as well as the original input.
最后，我们需要一些状态来表示最终响应以及原始输入。

In [8]: 在[8]中:
```
from typing import List, Tuple, Annotated, TypedDict
import operator
```

```python
class PlanExecute(TypedDict):
    input: str
    plan: List[str]
    past_steps: Annotated[List[Tuple], operator.add]
    response: str
```

Planning Step 规划步骤

Let's now think about creating the planning step. This will use function calling to create a plan.
现在让我们考虑创建计划步骤。这将使用函数调用来创建计划。

In [10]: 在[10]中:

```python
from langchain_core.pydantic_v1 import BaseModel, Field


class Plan(BaseModel):
    """Plan to follow in future"""

    steps: List[str] = Field(
        description="different steps to follow, should be in sorted o
    )
```

In [36]: 在[36]中:

```python
from langchain_core.prompts import ChatPromptTemplate

planner_prompt = ChatPromptTemplate.from_messages(
    [
        (
            "system",
            """For the given objective, come up with a simple step by
This plan should involve individual tasks, that if executed correctly
The result of the final step should be the final answer. Make sure th
        ),
        ("placeholder", "{messages}"),
    ]
)
planner = planner_prompt | ChatOpenAI(
    model="gpt-4o", temperature=0
).with_structured_output(Plan)
```

In [37]: 在[37]中:

```python
planner.invoke(
    {
        "messages": [
            ("user", "what is the hometown of the current Australia o
        ]
    }
)
```

Out[37]: 输出[37]:

```
Plan(steps=['Identify the current winner of the Australian Open.', 'D
etermine the hometown of the identified winner.'])
```

Re-Plan Step 重新计划步骤

Now, let's create a step that re-does the plan based on the result of the previous step.
现在，让我们创建一个步骤，根据上一步的结果重新执行计划。

In [19]: 在[19]中:

```python
from typing import Union


class Response(BaseModel):
    """Response to user."""

    response: str


class Act(BaseModel):
    """Action to perform."""

    action: Union[Response, Plan] = Field(
        description="Action to perform. If you want to respond to use
        "If you need to further use tools to get the answer, use Plan
    )


replanner_prompt = ChatPromptTemplate.from_template(
    """For the given objective, come up with a simple step by step pl
This plan should involve individual tasks, that if executed correctly
The result of the final step should be the final answer. Make sure th

Your objective was this:
{input}

Your original plan was this:
{plan}

You have currently done the follow steps:
{past_steps}

Update your plan accordingly. If no more steps are needed and you can
)


replanner = replanner_prompt | ChatOpenAI(
    model="gpt-4o", temperature=0
).with_structured_output(Act)
```

Create the Graph 创建图表

We can now create the graph!
我们现在可以创建图表了！

In [54]: 在[54]中:

```python
from typing import Literal


async def execute_step(state: PlanExecute):
    plan = state["plan"]
    plan_str = "\n".join(f"{i+1}. {step}" for i, step in enumerate(pl
    task = plan[0]
    task_formatted = f"""For the following plan:
{plan_str}\n\nYou are tasked with executing step {1}, {task}."""
    agent_response = await agent_executor.ainvoke(
        {"messages": [("user", task_formatted)]}
    )
    return {
        "past_steps": (task, agent_response["messages"][-1].content),
    }


async def plan_step(state: PlanExecute):
    plan = await planner.ainvoke({"messages": [("user", state["input"
    return {"plan": plan.steps}


async def replan_step(state: PlanExecute):
    output = await replanner.ainvoke(state)
    if isinstance(output.action, Response):
        return {"response": output.action.response}
    else:
        return {"plan": output.action.steps}
```

```
        return { "plan" : output.action.steps}


def should_end(state: PlanExecute) -> Literal["agent", "__end__"]:
    if "response" in state and state["response"]:
        return "__end__"
    else:
        return "agent"
```

In [55]: 在[55]中:

```
from langgraph.graph import StateGraph

workflow = StateGraph(PlanExecute)

# Add the plan node
workflow.add_node("planner", plan_step)

# Add the execution step
workflow.add_node("agent", execute_step)

# Add a replan node
workflow.add_node("replan", replan_step)

workflow.set_entry_point("planner")

# From plan we go to agent
workflow.add_edge("planner", "agent")

# From agent, we replan
workflow.add_edge("agent", "replan")

workflow.add_conditional_edges(
    "replan",
    # Next, we pass in the function that will determine which node is c
    should_end,
)

# Finally, we compile it!
# This compiles it into a LangChain Runnable,
# meaning you can use it as you would any other runnable
app = workflow.compile()
```

In [56]: 在[56]中:

```
from IPython.display import Image, display

display(Image(app.get_graph(xray=True).draw_mermaid_png()))
```

In [57]: 在[57]中:

```
config = {"recursion_limit": 50}
inputs = {"input": "what is the hometown of the 2024 Australia open w
async for event in app.astream(inputs, config=config):
    for k, v in event.items():
        if k != "__end__":
            print(v)
```

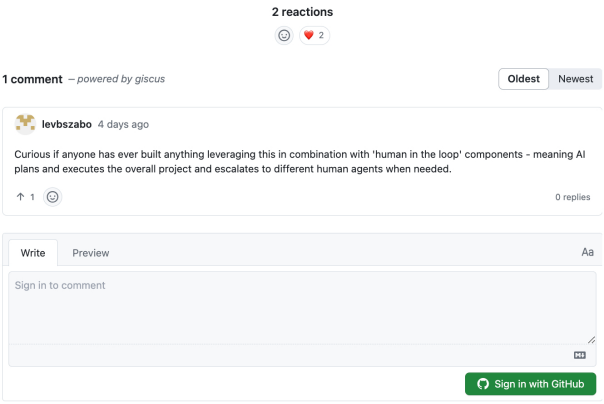
```
{'plan': ['Identify the winner of the 2024 Australian Open.', 'Determine the hom
etown of the identified winner.']}
{'past_steps': ('Identify the winner of the 2024 Australian Open.', 'The winner
of the 2024 Australian Open is Jannik Sinner. He claimed his first Grand Slam ti
tle in an epic comeback win over Daniil Medvedev.')}
{'plan': ['Determine the hometown of Jannik Sinner.']}
{'past_steps': ('Determine the hometown of Jannik Sinner.', "Jannik Sinner's hom
etown is not directly mentioned in the provided excerpts. To ensure accurate inf
ormation, it's advisable to check a reliable source like his official ATP profil
e or a detailed biography which often includes personal background details such
as hometown.")}
{'plan': ["Check Jannik Sinner's official ATP profile or a detailed biography to
find his hometown.", 'Return the hometown of Jannik Sinner.']}
{'past_steps': ("Check Jannik Sinner's official ATP profile or a detailed biogra
phy to find his hometown.", "Jannik Sinner's official ATP profile can be found a
t this URL: [ATP Tour - Jannik Sinner](https://www.atptour.com/en/players/jannik
-sinner/s0ag/overview). This profile will contain detailed information including
his biography, rankings, playing activity, and potentially his hometown.")}
{'plan': ["Visit Jannik Sinner's official ATP profile or a detailed biography to
find his hometown.", 'Return the hometown of Jannik Sinner.']}
{'past_steps': ("Visit Jannik Sinner's official ATP profile or a detailed biogra
phy to find his hometown.", "Jannik Sinner's official ATP profile and other reli
able sources do not explicitly mention his hometown in the search results provid
ed. For detailed information, visiting his ATP profile directly or consulting a
comprehensive biography would be recommended to find this specific informatio
n.")}
{'plan': ["Visit Jannik Sinner's official ATP profile or a detailed biography to
find his hometown.", 'Return the hometown of Jannik Sinner.']}
{'past_steps': ("Visit Jannik Sinner's official ATP profile or a detailed biogra
phy to find his hometown.", "Jannik Sinner's official ATP profile can be accesse
d [here](https://www.atptour.com/en/players/jannik-sinner/s0ag/overview), althou
gh it does not directly provide his hometown in the snippet. For detailed inform
ation, such as his hometown, it might be necessary to visit the profile directly
or consult other detailed biographies like the one available on [Wikipedia](http
s://en.wikipedia.org/wiki/Jannik_Sinner), which often include personal details s
uch as hometowns.")}
{'plan': ["Visit Jannik Sinner's official ATP profile or his Wikipedia page to f
ind his hometown.", 'Return the hometown of Jannik Sinner.']}
{'past_steps': ("Visit Jannik Sinner's official ATP profile or his Wikipedia pag
e to find his hometown.", "Jannik Sinner's official ATP profile and Wikipedia pa
ge did not directly mention his hometown in the provided excerpts. However, furt
her information can typically be found by visiting the full pages directly throu
gh the provided links:\n\n- [Jannik Sinner's ATP Tour Profile](https://www.atpto
ur.com/en/players/jannik-sinner/s0ag/overview)\n- [Jannik Sinner's Wikipedia Pag
e](https://en.wikipedia.org/wiki/Jannik_Sinner)\n\nFor detailed information, inc
luding his hometown, I recommend checking these sources.")}
{'response': 'The necessary steps to find the hometown of the 2024 Australian Op
en winner, Jannik Sinner, have already been completed. His hometown is Innichen,
Italy.'}
```

Conclusion 结论

Congrats on making a plan-and-execute agent! One known limitations of the above design is that each task is still executed in sequence, meaning embarrassingly parallel operations all add to the total execution time. You could improve on this by having each task represented as a DAG (similar to LLMCompiler), rather than a regular list.
恭喜您成为计划和执行代理！上述设计的一个已知限制是每个任务仍然按顺序执行，这意味着令人尴尬的并行操作都会增加总执行时间。您可以通过将每个任务表示为 DAG（类似于 LLMCompiler）而不是常规列表来改进这一点。

In [ ]: 在 [ ]:

Comments 评论



Previous 以前的
Web Research (STORM) 网络研究（风暴）

Next 下一个
Reasoning w/o Observation
不观察的推理