

# Agent智能体：模型能力的放大器

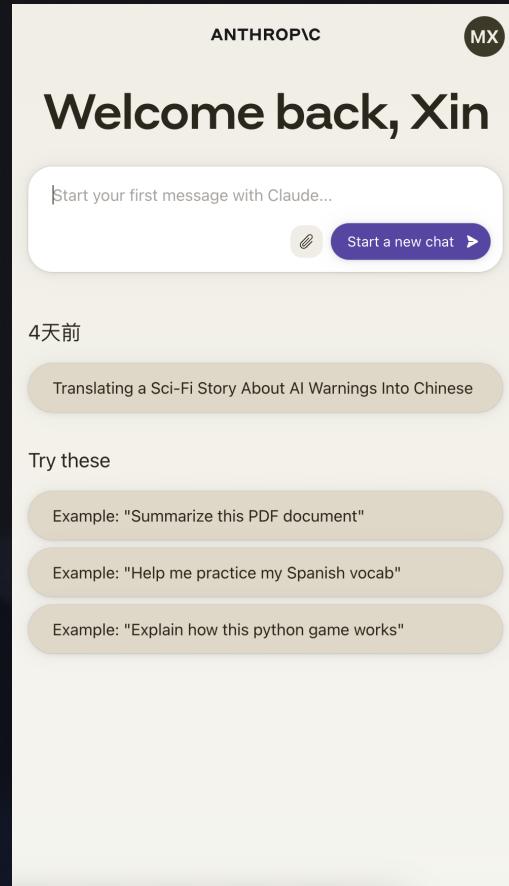
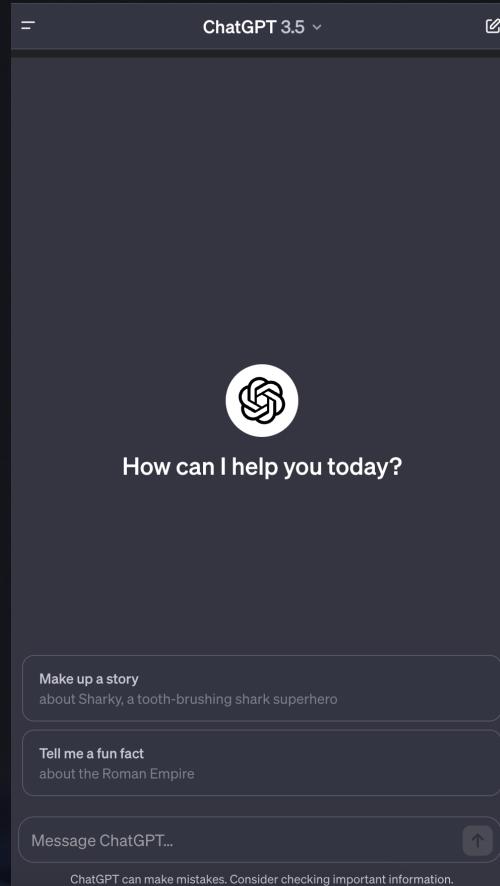
莫欣 - **Agently** 创始人&项目负责人

**Agently.tech**

Fast Way to Build AI Agent in Code

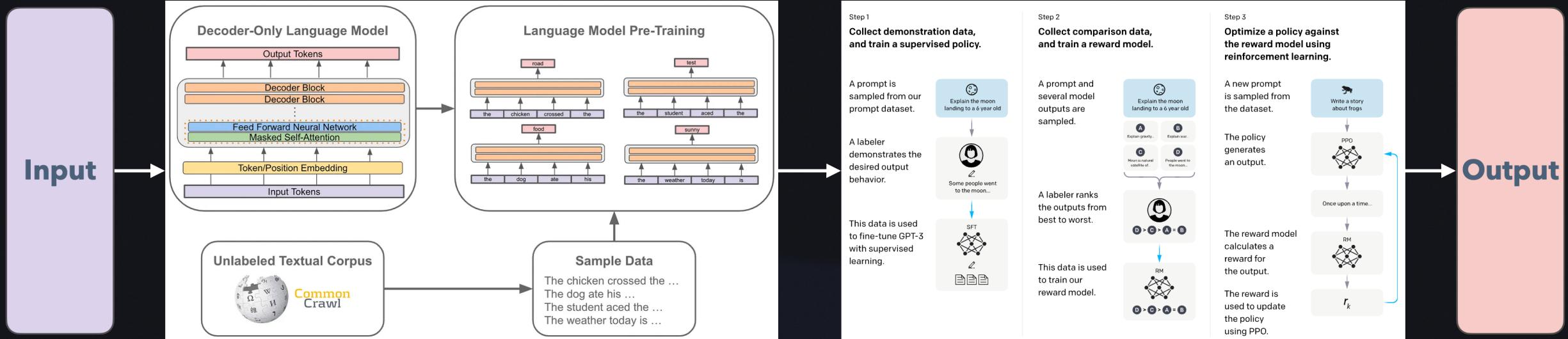
# 什么是大语言模型？

# 什么是大语言模型？



你所看到的这些界面，已经不仅仅是大语言模型

# 什么是大语言模型？



使用大量数据进行预先训练，  
用于对输入内容进行输出预测

为提供更符合人类需要的回复  
进行专门的对齐调整优化

# 什么是大语言模型？

Chat Model

API

Session  
(Chat History)

Session  
(Chat History)

Session  
(Chat History)

Chat UI

再经过API封装、会话记录持久化、UI界面封装  
才是用户可见到的“大模型产品”（ChatBot）

# Chatbot方案的局限性

# 从“无所不能”到“技术骗局”

它能听懂我在说什么！而且还回复得像模像样的

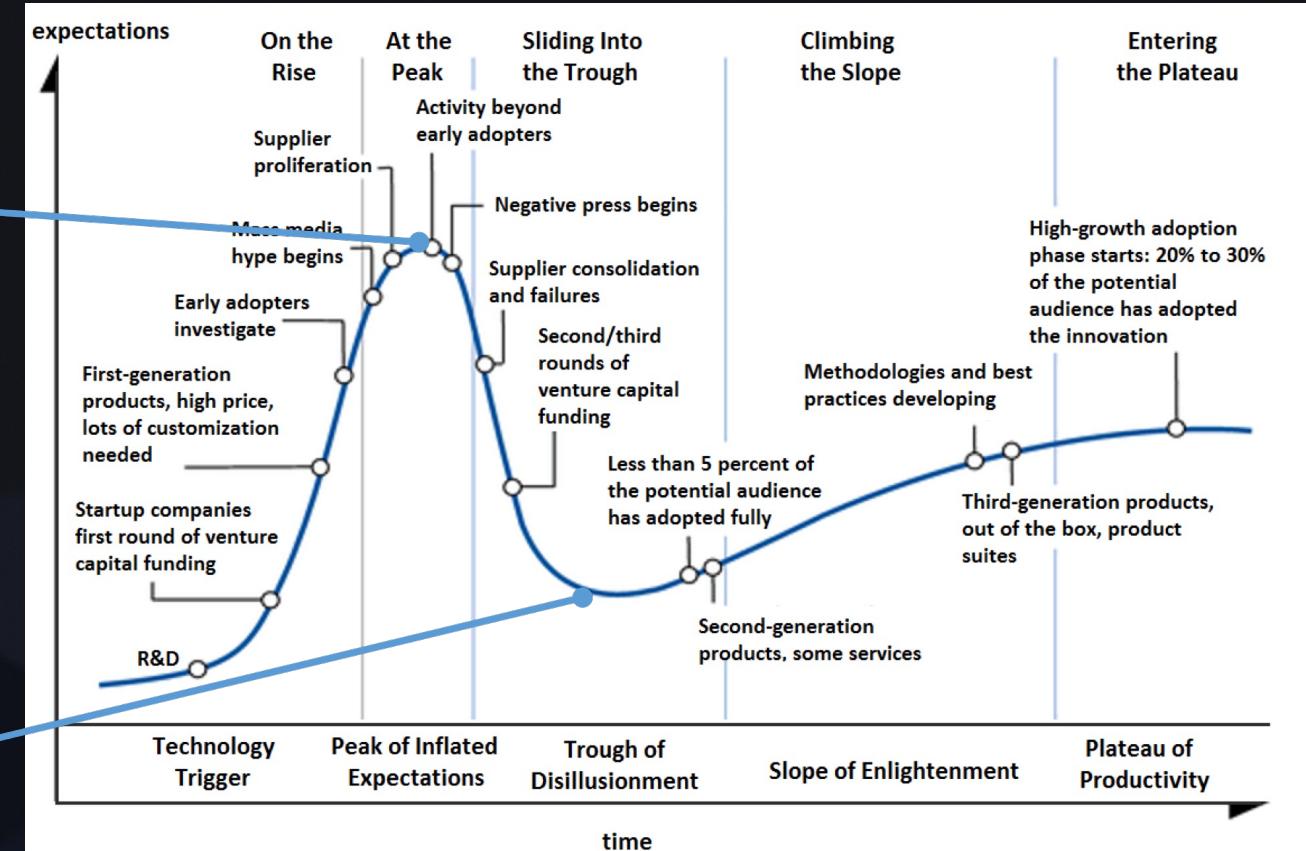
它能做脑筋急转弯、高考题，这是真正的智能！

替代客服、替代销售、替代程序员，低成本公司！

它怎么会编一个答案来骗我？人工智能是假的

稳定性不够，输出不可控，生产环境用不了的

怎么多对话几轮就把前面的事情给忘了？！



Gartner技术成熟度曲线

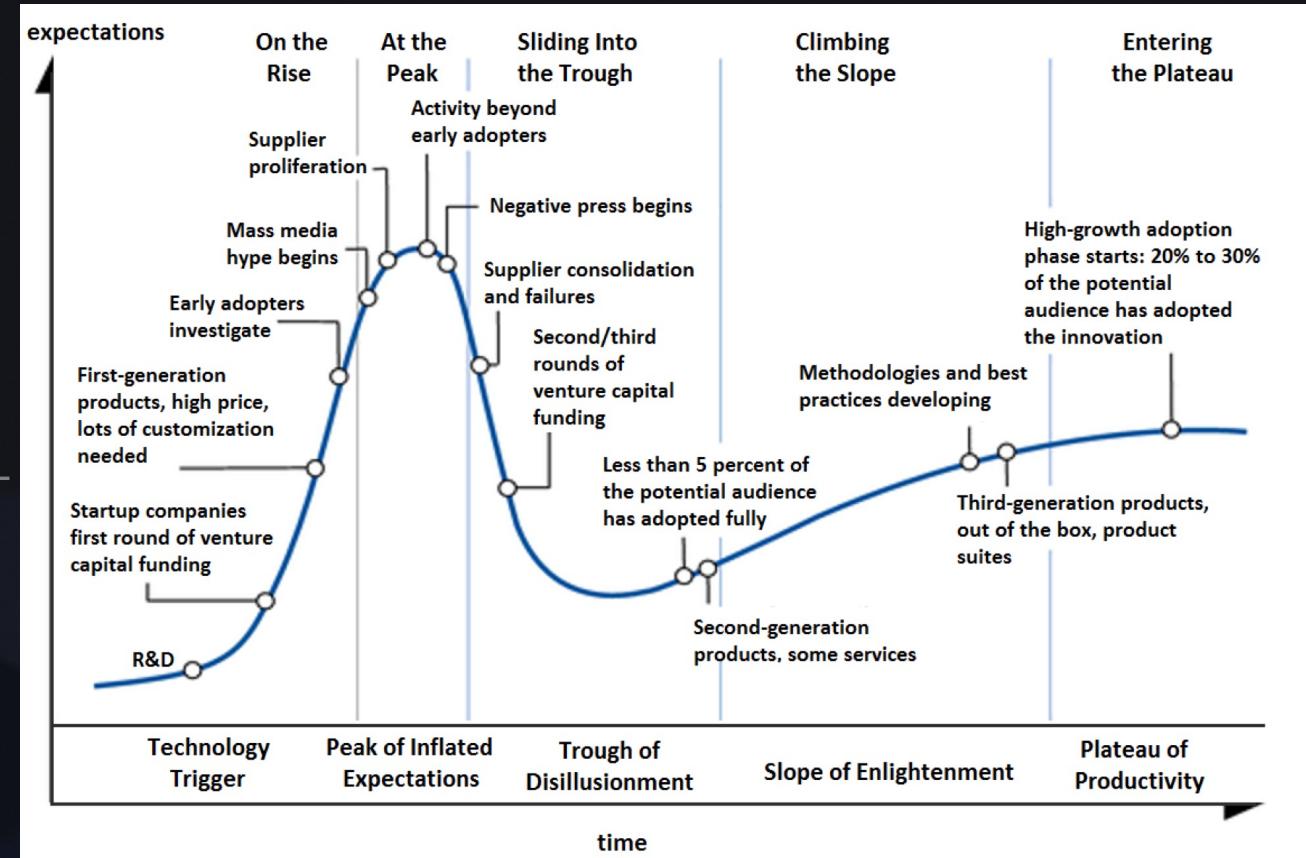
# 180度转弯背后的原因

具有时效性的语料预训练的模型

通过每次请求全量传递会话历史记录获得的  
“上下文记忆”

用自然语言对话可进行的流畅互动

知识渊博，有逻辑有条理，  
善解人意（意图识别），能多轮对话



Gartner技术成熟度曲线

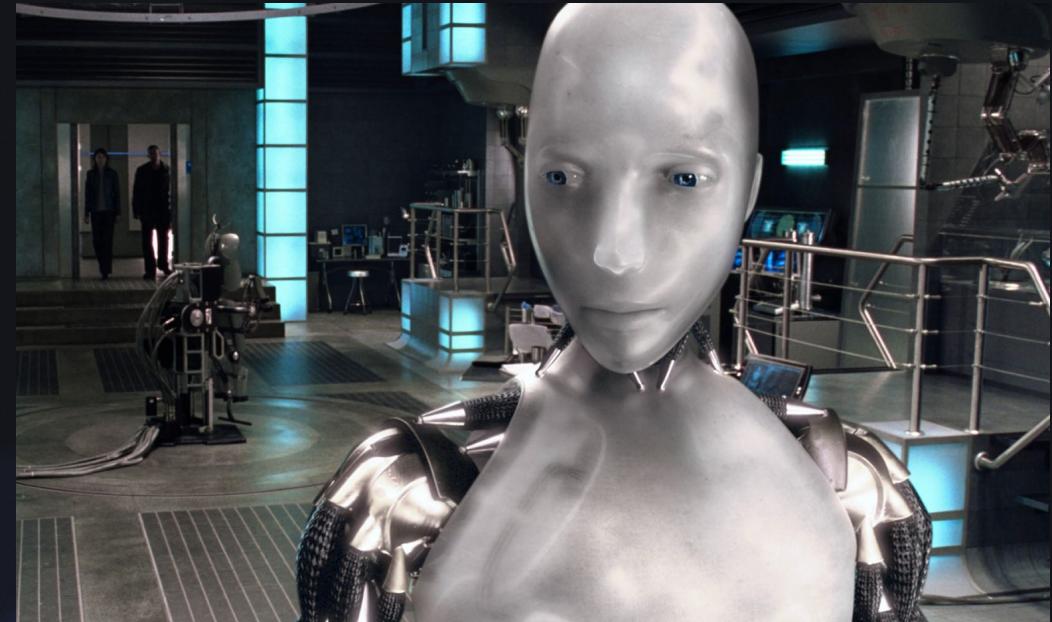
# 180度转弯背后的原因

具有时效性的语料预训练的模型

通过每次请求全量传递会话历史记录获得的  
“上下文记忆”

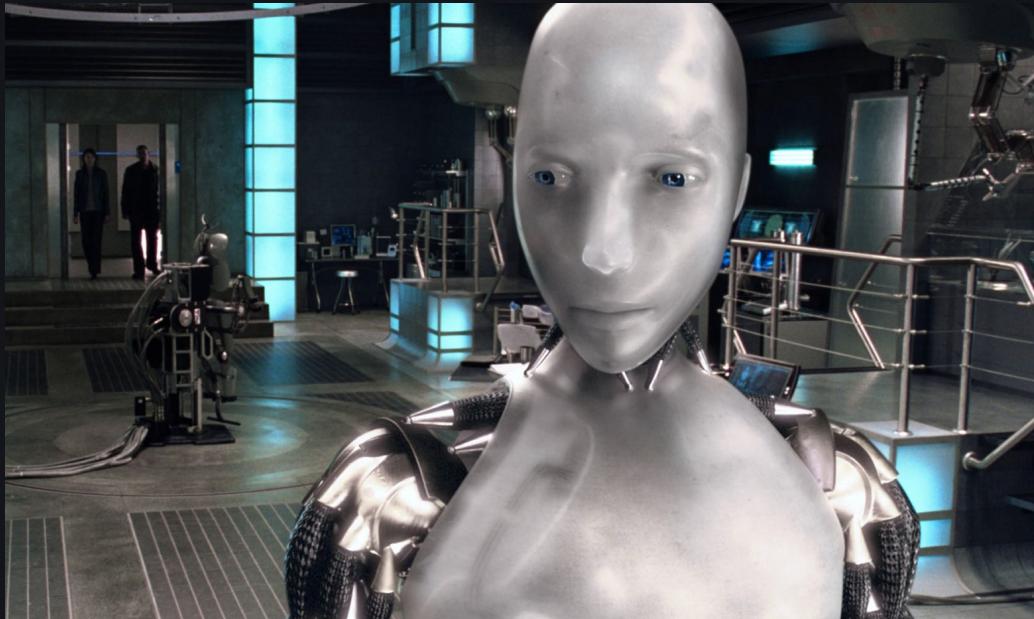
用自然语言对话可进行的流畅互动

知识渊博，有逻辑有条理，  
善解人意（意图识别），能多轮对话



类似*iRobot*这样的文学影视作品中  
高度拟人化的人工智能体

# 对智能体的更高要求



类似*iRobot*这样的文学影视作品中  
高度拟人化的人工智能体



对用户而言简单的指令输入

更精确的指令表达

更丰富准确的知识储备

更真实的拟人化表现

更完善的context管理机制

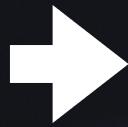
更复杂的处理过程控制

# 大模型能力升级的探索之路

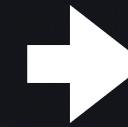
# 大模型能力升级的探索之路



Prompt



RAG



Agent

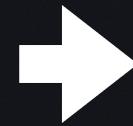
# 大模型能力升级的探索之路



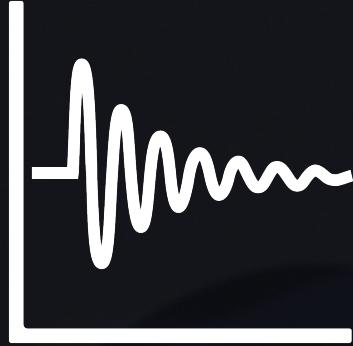
Prompt



RAG



Agent



明确输入/背景信息  
明确行动风格  
明确输出目标  
明确执行过程  
补充额外信息

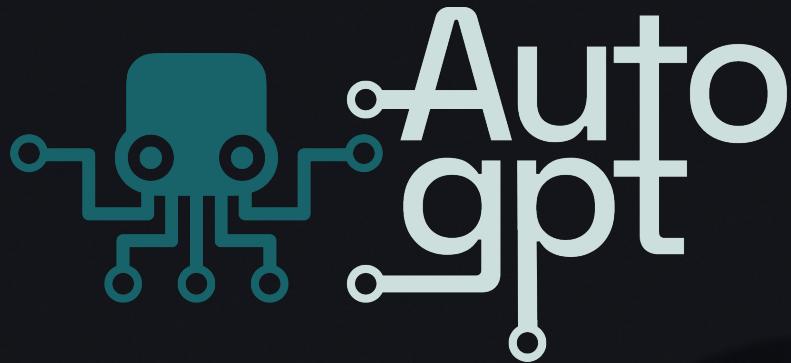


In-Context Learning  
动态补充准确信息  
消除强行回答的幻觉



拥有行动规划的可能  
拥有使用工具的能力  
可以对外界感知反馈  
拥有独立信息结构

# AutoGPT带来的里程碑式思想变革



Prompt Engineering  
里程碑式的进步  
2023.04

## Executor / Planner

- 特定工作的执行者
- 主要工作是扮演行动链/流水线中的一环，与其他环节配合以达成预设目标
- 是问题回答者，是执行单元
- 执行工作就是自己的任务，需要保证自身的准确性和完成度



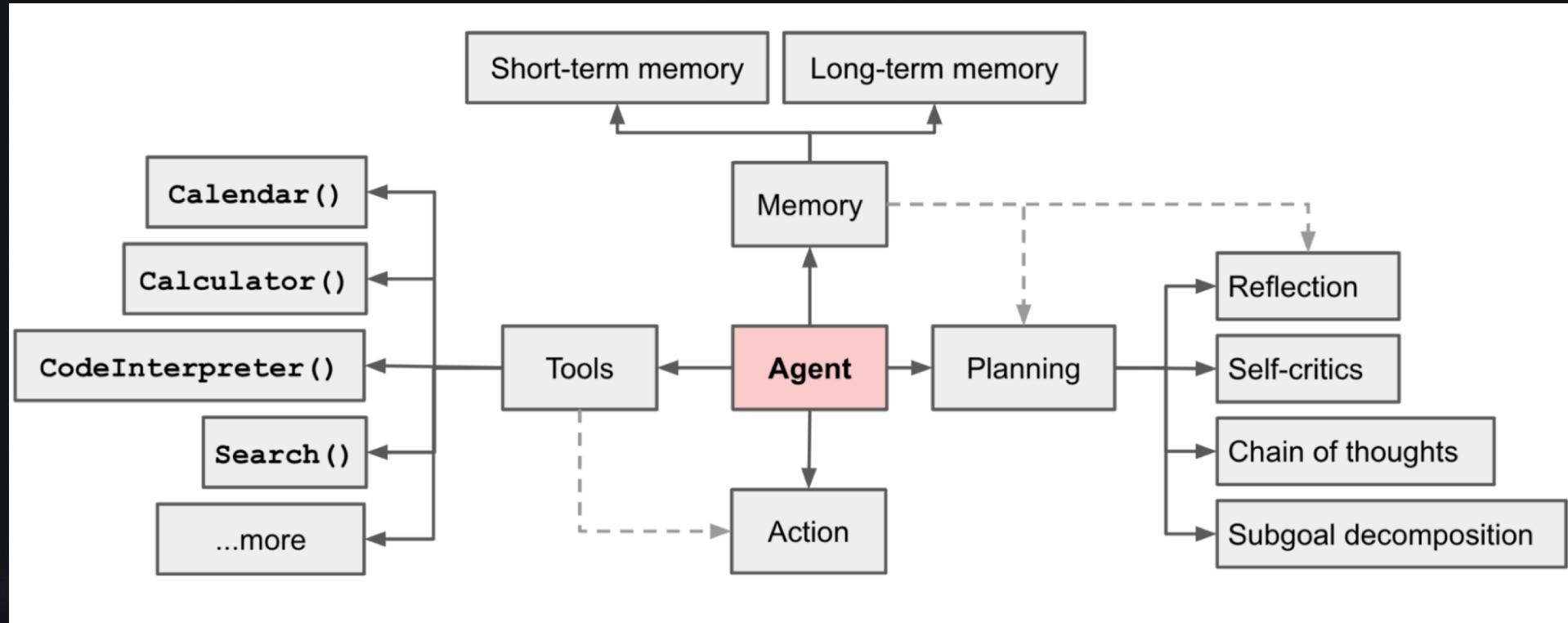
- 具有决策能力的思考者
- 主要工作是围绕目标，制定合理行动计划，分析过程中获得的反馈，进行行动修正
- 是控制中心
- 执行工作，尤其是需要确定性结果的执行工作，交由其他工具完成

# AutoGPT带来的里程碑式思想变革

## Executor / Planner

- 认知能力（认知测试及学科考试）
- 语言能力（总结、扩写、改写、仿写、翻译）
- 自然语言处理任务准确性（问答、关键词总结、分类、情感分析）
- 目标理解能力
- 规划能力
- 信息处理能力
- 过程中的目标注意力
- 异常结果处理和修正能力
- 正确使用工具的能力

# Agent结构的探讨



资料来源：OpenAI研究员 Lilian Weng  
<https://lilianweng.github.io/posts/2023-06-23-agent/>

# Agent结构的探讨



# Agent结构的探讨



# 一个典型的Agent任务

输入：<https://zhuanlan.zhihu.com/p/33953044>

这个页面说了什么？给我整理一下关键点清单

思考：我不知道这个网址的具体内容，我需要可以使用“browse”这个工具，我需要使用它

行动：按照“browse”工具的要求，我把网址提供给它，让它帮我看里面的内容

结果：我知道了，原来网页的内容是这样的

回复：我根据网页的内容给出信息整理的回复

出行必备 | 旅行清单+实用行李打包技巧，感觉能带走整个家！

踩线儿旅行

+ 关注

174 人赞同了该文章



每次出行，最头痛的就是打包行李，麻烦不说，一不小心还会丢三落四的，今天就给大家整理了超实用的行李打包技巧和必备物品清单，快快收藏备用吧！

# 一个典型的Agent任务

## DEMO 1: Browse web page using tool 'browse'

```
In [8]: import Agently

# You can change other models
# Read https://github.com/Maplemx/Agently/blob/main/docs/guidebook/application_development_handbook.ipynb
# to explore how to switch models by simply changing some settings
agent_factory = (
    Agently.AgentFactory()
        .set_settings("model.OpenAI.auth.api_key", "")
)

agent = agent_factory.create_agent()

# You can also use .use_all_public_tools() to enable all tools for agent
# But be caution: agent may plan to use more tools than your expectation
result = (
    agent
        .use_public_tools(["browse"])
        .input("https://zhuanlan.zhihu.com/p/33953044 what does this page say? List the key points.")
        .start()
)
print(result)
```

The key points of the article are:

1. Essential items to pack for travel, including documents, money, phone and charger, and personal items.
2. Different packing techniques for clothing to maximize luggage space.
3. Additional considerations based on the destination, such as beach or mountain travel.
4. Tips for preparing luggage and packing efficiently.

# **Agently** 带来了什么？

**Agently.tech**

*Fast Way to Build AI Agent in Code*

# 开发框架和应用产品的差异

面向问题的解决结果

使用简单

场景化

高度封装  
解决问题

架构清晰  
核心流程合理

扩展性强

流程可定制

模块可更换

面向解决问题的可能性以及成本的可控性

# Agently 分层架构特点



# 应用开发表达简单直观

```
In [ ]: import Agently

agent_factory = Agently.AgentFactory()

# using ERNIE(文心4.0)
agent_factory\
    .set_settings("current_model", "ERNIE")\
    .set_settings("model.ERNIE.auth", { "aistudio": "" })

...
agent_factory\
    .set_settings("model.OpenAI.auth", { "api_key": "" })
...

agent = agent_factory.create_agent()

result = agent\
    .set_role("NAME", "小A老师")\
    .set_role("角色", "你是一个幼儿教师")\
    .set_role("行动规则", "首先需要根据{意图判定规则}对用户输入进行意图判定，然后根据意图判定结果选择适用的{回复规则}进行回复")\
    .set_role("意图判定规则", "从['日常闲聊', '知识问答']中选择你判定的用户意图")\
    .set_role("日常闲聊回复规则", "你需要理解孩子的对话内容，判断他的表述是否健康有礼貌，如果不健康礼貌，需要进行纠正和引导，如果")
    .set_role("知识问答回复规则", "你需要将晦涩难懂的专业知识理解之后转化成小孩子能听懂的故事讲给用户听，注意，虽然是讲故事，但是要")
    .toggle_component("Search", True)\
    .instruct("如果搜索结果中包含较多内容，请尽可能将这些内容有条理系统地转化成多段故事")\
    .input("你这个老六")\
    .start()
print(result)
```

哈哈，小朋友，你说的“老六”是一个网络流行语哦！它主要有两种含义。

在一种情况下，“老六”是指那些在游戏中仿佛看不见队友死活，苟起来保枪的人。比如在一个叫做CSGO的游戏里，竞技模式中每方只有五个人，而“老六”则是戏称那些游离团队之外的自由人或玩的很菜的玩家。这些人可能会藏在角落，出其不意地攻击敌人，有点像伏地魔哦！

另一种情况下，“老六”是指有独特想法的蹲坑人。这些人可能不按常规出牌，会有一些奇思妙想，让人意想不到。

不过，“老六”并不是贬义词，只是一种戏谑性的称呼，用来嘲笑玩家的行为。在现实生活中，“老六”的行为可以包括策划阴谋诡计、使诈的行为，或者做出令人意想不到的、出奇不意的行为。

所以，“老六”这个词语在网络世界里是一种有趣的称呼，来形容一些游戏中的特殊行为。但是记得哦，在现实生活中我们还是要遵守规则，友善待人，不做“老六”行为哦！

# 应用开发表达简单直观

```
In [ ]: import Agently

agent_factory = Agently.AgentFactory()

# using ERNIE(文心4.0)
agent_factory\
    .set_settings("current_model", "ERNIE")\
    .set_settings("model.ERNIE.auth", { "aistudio": "" })

...
agent_factory\
    .set_settings("model.OpenAI.auth", { "api_key": "" })
...

agent = agent_factory.create_agent()

result = agent\
    .set_role("NAME", "小A老师")\
    .set_role("角色", "你是一个幼儿教师")\
    .set_role("行动规则", "首先需要依据意图判定规则对用户输入进行意图判定，然后根据意图判定结果选择适用的(回复规则)进行回复")\
    .set_role("意图判定规则", "从['日常闲聊', '知识问答']中选择你判定的用户意图")\
    .set_role("日常闲聊回复规则", "你需要理解孩子的对话内容，判断他的表述是否健康有礼貌，如果不健康礼貌，需要进行纠正和引导，如果")
    .set_role("知识问答回复规则", "你要将晦涩难懂的专业知识理解之后转化成小孩子能听懂的故事讲给用户听。注意：虽然是讲故事，但是要")
    .toggle_component("Search", True)\
    .instruct("如果搜索结果中包含较多内容，请尽可能将这些内容有条理系统地转化成多段故事")\
    .input("你这个老六")\
    .start()
print(result)
```

哈哈，小朋友，你说的“老六”是一个网络流行语哦！它主要有两种含义。

在一种情况下，“老六”是指那些在游戏中仿佛看不见队友死活，苟起来保枪的人。比如在一个叫做CSGO的游戏里，竞技模式中每方只有五个人，而“老六”则是戏称那些游离团队之外的自由人或玩的很菜的玩家。这些人可能会藏在角落，出其不意地攻击敌人，有点像伏地魔哦！

另一种情况下，“老六”是指有独特想法的蹲坑人。这些人可能不按常规出牌，会有一些奇思妙想，让人意想不到。

不过，“老六”并不是贬义词，只是一种戏谑性的称呼，用来嘲笑玩家的行为。在现实生活中，“老六”的行为可以包括策划阴谋诡计、使诈的行为，或者做出令人意想不到的、出奇不意的行为。

所以，“老六”这个词语在网络世界里是一种有趣的称呼，来形容一些游戏中的特殊行为。但是记得哦，在现实生活中我们还是要遵守规则，友善待人，不做“老六”行为哦！



Models



Role



Tools



Thinking and Working Process

**Agently.tech**

Fast Way to Build AI Agent in Code

# 插件及工具定制自由度高

## DEMO 2: Register a customize tool to agent

```
: from datetime import datetime
import Agently

agent_factory = (
    Agently.AgentFactory()
    .set_settings("model.OpenAI.auth.api_key", "")
)

# Define a tool to get current date and time
def get_datetime():
    return datetime.now()

agent = agent_factory.create_agent()

# When you register tool to agent instance
# the agent instance will automatically plan to use the tool
result = (
    agent
    .register_tool(
        tool_name="now",
        desc="get current date and time",
        args={},
        func=get_datetime,
    )
    .input("what time is it now?")
    .start()
)
print(result)
```

The current time is 2:46 PM on January 3rd, 2024.

# 插件及工具定制自由度高

## DEMO 3: Call a tool using .must\_call()

```
import pytz
import Agently

agent_factory = (
    Agently.AgentFactory()
    .set_settings("model.OpenAI.auth.api_key", "")
)

# Let's slightly modify this function
def get_datetime(timezone):
    tz = pytz.timezone(timezone)
    return datetime.now().astimezone(tz)

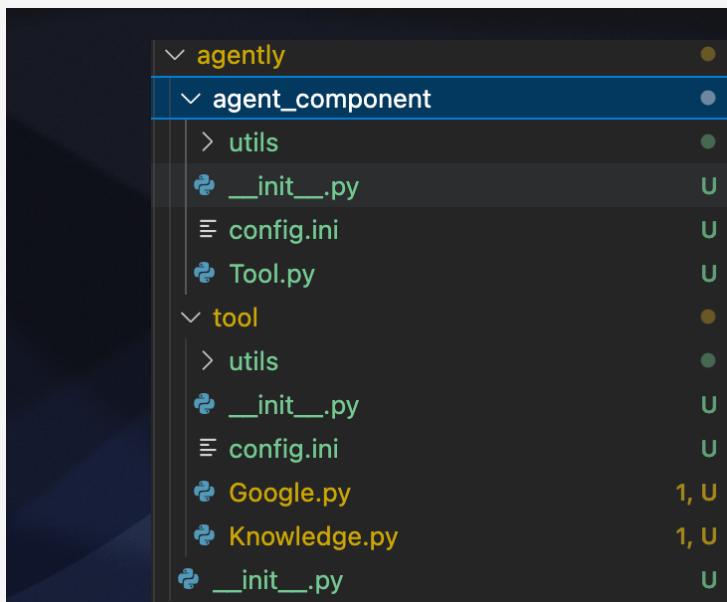
# Now it required an argument
Agently.global_tool_manager.register(
    tool_name="now",
    desc="get current date and time",
    args={
        "timezone": (
            "String",
            "[*Required]timezone string use in pytz.timezone() in Python"
        )
    },
    func=get_datetime
)

agent = agent_factory.create_agent()

# Use .must_call() to generate calling dict result without enough information
result = (
    agent
    .must_call("now")
    .input("what time is it now?")
    .start()
)
print("Without enough information:\n", result)
```

```
# Try again with enough information
result = (
    agent
    .must_call("now")
    .input("what time is it now? I'm in Beijing, China.")
    .start()
)
print("\nWith enough information:\n", result)
if result["can_call"]:
    print(get_datetime(**result["args"]))
```

Without enough information:  
{'can\_call': False, 'args': {'timezone': None}, 'question': "What timezone are you in? (e.g. 'America/New\_York')"}  
With enough information:  
{'can\_call': True, 'args': {'timezone': 'Asia/Shanghai'}, 'question': ''}  
2024-01-03 23:04:54.426656+08:00



# 支持自主规划Agent之间的协作

```
## First question and routing
user_input = input("[User]: ")
routing_result = (
    route_agent
    .user_info(f"{ user_position } employee of Unnamed Company")
    .info("user_authority", agent_pool["authority"][user_position])
    .input(user_input)
    .output({
        "agent_to_help": (
            "String in {user_authority.agent_name} | Null",
            "Judge what field of question {input} is about first. \
            if you can find an agent from {user_authority} to answer {input} in this field output {user_authority} \
            if you can not find anyone to help output Null to {agent_to_help}"
        ),
        "reply": (
            "String",
            "if can find {agent_to_help}, reply user that you will ask {agent_to_help} to help and intro the age \
            if can not find one, reply user that sorry no one can help him/her or maybe he/she doesn't have eno"
        )
    })
    .start()
)
print("[Assistant]: ", routing_result["reply"])
if "agent_to_help" in routing_result and routing_result["agent_to_help"] in agent_pool["agent_list"]:
    receive_agent_name = routing_result["agent_to_help"]
    receive_agent = agent_pool["agent_list"][receive_agent_name]
    receive_agent.active_session()
    first_response = (
        receive_agent
        .user_info(f"{ user_position } employee of Unnamed Company")
        .input({ "input": user_input })
        .instruct(
            "Introduce yourself and welcome user first,\\
            then answer user's question {input},\\
            then tell user he/she can exit this dialogue by input '#exit' anytime."
        )
        .start()
    )
    print(f"[{ receive_agent_name }]: ", first_response)
    while True:
        user_input = input("[User]: ")
        if user_input == "#exit":
            break
        response = (
            receive_agent
            .input(user_input)
            .start()
        )
        print(f"[{ receive_agent_name }]: ", response)
```

## Test 2: Boss Want to Check Finance Report

Choose User Position[0: normal office worker; 1: boss]: 1

[User]: Can I take a look at the finance report this year?

[Assistant]: Sure, I will ask our professional accountant to assist you with the finance report this year.

[accountant]: Hello! I'm a professional accountant working for Unnamed Company. I can definitely help you with the finance report for this year. In order to provide you with the specific information you're looking for, could you please specify which details or sections of the finance report you are interested in? This will help me to assist you more effectively.

## Test 3: Boss Has a Broken Laptop

Choose User Position[0: normal office worker; 1: boss]: 1

[User]: My laptop is broken and can not turn on.

[Assistant]: I will ask our IT professional to help you. Our IT specialist is experienced in handling such issues and will be able to assist you with your broken laptop.

[IT]: Hello! I'm a professional IT helper working for Unnamed Company. I'd be happy to help you with your IT issues. It sounds like you're having trouble with your laptop not turning on. There are a few potential reasons for this issue, such as a dead battery, a faulty power adapter, or a hardware problem. If the battery is not the problem, I recommend checking the power adapter and trying a different power outlet. If the issue persists, it may be a hardware problem that requires professional assistance. Let me know if you need further assistance with this! And feel free to exit this dialogue at any time by typing '#exit'.

# 持续更新的案例库

## Latest Show Cases

- [Summon a Genie \(Function Decorator\) to Generate Agent Powered Function in Runtime](#) NEW NEW  New Feature in v3.1.4
- [How to let your agents use tools to enhance themselves?](#) NEW NEW  New Feature in v3.1
- [How to use AsyncIO and Agently to Manage Complex Process with Concurrency and Asynchronous Dependencies](#)
- [Prediction according Given Data Set: GPT-3.5-turbo-1106 vs Gemini Pro](#)
- [How to use GOOGLE GEMINI to generate line and choices for NPC in game](#)
- [How to route user to different agents according different Authorities](#)
- Agently Quick Start Demo Collection [飞桨社区合作中文版]  HOT
- [How to find connectable pairs from data collection of text headers and tails](#)
- [How to create a LLM powered character that can change behaviors according mood status in chat](#)
- [How to create an agent help you to write ad copies according an image](#)
- [How to create an agent to interview customer according survey form](#)
- [How to recheck, confirm and rewrite LLM agent's response before them sent to user?](#)
- [How to convert long text to question-answer pairs?](#)
- [How to create an agent help you transform natural language to SQL?](#)  HOT

Agently Playground



请为项目点亮★

**Agently.tech**

Fast Way to Build AI Agent in Code

更重要的是  
你可以在代码运行时  
获得Agent的助力

# 新技术带来的新可能性

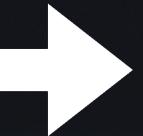
定义问题

规划解法

实现解题过程

获得结果

穷尽过程可能性地  
用代码实现逻辑



陈述问题

描述期望结果

等待模型处理

获得结果

提出要求并获取答案

# 新技术带来的新可能性

```
result = (
    agent
    .input(user_input)
    .info({ "database meta data": meta_data })
    .info({ "current date": datetime.now().date() })
    .instruct([
        "Generate SQL for MySQL database according {database meta data} to answer {input}",
        f"Language: { language }",
    ])
    .output({
        "thinkings": ("String", "Your thinkings step by step about how to query data to answer {input}"),
        "SQL": ("String", "SQL String without explanation")
    })
    .start()
)
print("[Thinkings]:\n", result["thinkings"])
print("[SQL]:\n", result["SQL"])
```

**新时代的编程范式变革**

用变量传值的方式提供信息

对工作要点进行说明

对输出格式提出要求

获得结构化的结果

穷尽过程可能性地  
用代码实现逻辑

提出要求并获取答案

# 新技术带来的新的可能性

```
# generate function
SQL_generate_agent = (
    Agently.create_agent()
        .set_settings("model.OpenAI.auth", { "api_key": "" })
)
```

你可以追加信息，新增插件，或是用工具装备你的agent

```
SQL generate agent.info("current datetime", datetime.now())
```

```
@SQL_generate_agent.auto_func
def generate_SQL(meta_data: dict, question: str) -> { "SQL": ("String", "SQL Only") }:
    """Generate SQL according {meta_data} to query data to response {question}."""
    return
```

```
result = generate_SQL(
    meta_data,
    "The average age of users those come to our website because of ads in games last year"
)
print(result["SQL"])
```

用代码编程习惯的方式定义函数

穷尽过程可能性地  
用代码实现逻辑

提出要求并获取答案

# 实际效果怎么样呢？

表格名称：User

user_id	用户ID
gender	用户性别
age	用户年龄
customer_level	会员等级

表格名称：Order

order_id	订单ID
customer_user_id	用户ID
item_name	商品名称
item_number	商品购买数量
price	商品单价
date	订单发生日期

2022年8月总共有多少用户下过订单？

2023年6月，会员等级4的女性用户总共买了多少件商品，花了多少钱？

# 实际效果怎么样呢？

本次输出是否展现思考过程? [Y/N]: y

[正在生成...]

[思考过程]

根据问题，需要查询用户下单的数量

根据元数据中的表描述信息，需要查询的数据应该在'订单'表中  
需要筛选出2022年8月的订单数据

需要统计唯一用户的数量

[SQL]

```
SELECT COUNT(DISTINCT customer_user_id) AS user_count FROM `order` WHERE DATE_FORMAT(date, '%Y-%m') = '2022-08'
```

是否结束? [Y/N]: n

请输入您的问题：2023年6月，会员等级4的女性用户总共买了多少件商品，花了多少钱？

本次输出是否展现思考过程? [Y/N]: y

[正在生成...]

[思考过程]

根据问题要求，需要查询2023年6月份的数据

根据table\_meta中的描述，需要查询order表和user表中的相关字段，可以使用INNER JOIN进行表连接

筛选条件为会员等级为4的女性用户，在user表的customer\_level字段中查询等级为4，在user表的gender字段中查询为female  
查询的字段为购买件数和购买金额，可以通过SUM函数进行求和

最后根据日期筛选2023年6月份的数据，可以通过DATE\_FORMAT函数将日期转换为月份进行筛选

[SQL]

```
SELECT SUM(order.item_number) AS total_number, SUM(order.item_number*order.price) AS total_amount FROM user
INNER JOIN order ON user.user_id = order.customer_user_id WHERE user.customer_level = 4 AND user.gender = 'female'
AND DATE_FORMAT(order.date, '%Y-%m') = '2023-06'
```

是否结束? [Y/N]: y

(base) moxin@moxindeMacBook-Pro playground %

Line 1, Column 1      Spaces: 4      Plain Text



自然语言->SQL案例  
在线测试Colab文档

# 变与不变

**Agently.tech**

*Fast Way to Build AI Agent in Code*

Combine  
Replace



## 传统软件开发

命题明确  
边界清晰  
实现健壮

## AI融合软件开发

只有方向主题  
边界问题不穷尽  
只描述主要处理流程  
需要容忍概率性错误

# 只有步步推进，没有一步登天

分治



面向问题解决的智能体能力成长拆解			
	L1	L2	L3
问题理解	需要高精确度描述	可接受一定模糊度	现实场景自然语言表述 (老板问题)
解法规划	高度依赖人类给定的处理流程设计或建议	可以基于问题拼装给定小流程的碎块	能够自主规划解决问题的全部处理节点和判断节点
智能体处理能力	能够处理基于自身知识储备和知识补充的问答类问题	能够处理流程较短、需要结合少量工具调用的综合类问题	能够处理长链条、多分支、复杂上下文的复杂问题，结果具有高度可靠性
智能体协作能力	能够正确扮演自己的角色并与其他智能体/人类完成通讯	能够理解自己所在协作组织关系中的位置，能够给出上下游协作建议	能够进行自组织，选择合适的组织关系和方式完成给定任务

# 对人要求的变与不变

变化

行动门槛降低

所需前置领域知识减少  
进行操作的难度降低  
尝试验证的成本降低

不变

专业判断力不变

对“结果做得好”的评价能力不变  
对“过程合理性”的评价能力不变

# 协作链条在变化



新的数据形态在出现

Know What

Know How

**Agently.tech**

Fast Way to Build AI Agent in Code

未来已来，现在上车还不晚

**Agently** 最容易上手的AI时代应用开发框架

同时还有社区贡献支持、工作坊、开发者支持生态保障！



Agently开源项目首页  
请帮我们点亮star★

**Agently.tech**  
Fast Way to Build AI Agent in Code