

# 初识 ModelScope-Agent

给我一个足够长的杠杆和一个支点,我可以撬动整个地球。

——阿基米德

## 3.1 ModelScope-Agent 简介



10min

随着大语言模型的兴起,生产力的深刻变革正在被见证,这些模型不断挑战创新与创造的传统边界。在这一历史性的转折点上,研发与之相匹配的生产力工具显得尤为迫切。这些工具不仅是激发大模型潜能、推动时代进步的关键,更是满足各行各业特定需求、推动内容创作、产品设计及决策分析等多领域革命性飞跃的核心力量。在众多工具中,ModelScope-Agent 被看作撬动生产力爆发的关键之一。

### 3.1.1 什么是 ModelScope-Agent

ModelScope-Agent<sup>[29]</sup>,作为一个灵活且通用的平台,为开发者和企业提供了智能体定制化开发的全新解决思路。它不仅是一个模型部署工具,更是一个集模型管理、多工具集成和定制于一体的生态系统。用户可以在 ModelScope-Agent 的基础上构建各种类型的人工智能应用,无论是基础的问答系统,还是复杂的业务流程自动化。此外,ModelScope-Agent 还拥有强大的社区和生态支持,作为 ModelScope 平台的一部分,用户可以获取最新的模型研究成果、教程和最佳实践。这使非专业背景的开发者也能轻松地参与到人工智能应用的创新中来,降低了技术壁垒,让更多用户能够快速分享人工智能技术带来的便利。

### 3.1.2 为什么选择 ModelScope-Agent

之所以选择 ModelScope-Agent 作为构建智能体的框架,是因为它在众多人工智能智能体框架和平台中具有显著的优势和特色。相较于其他框架,如 AutoGPT<sup>[30]</sup>、FastGPT 和 DB-GPT<sup>[31]</sup>,ModelScope-Agent 提供了一个更为通用和高度可定制的开源解决方案。

AutoGPT 利用 GPT-4 技术实现了人工智能的自主性操作,能够在多个领域自主规划和执行任务,然而,这种过分的自主性导致使用者能够参与的部分太少,从而诱发了结果的

不可控。与此同时,使用者参与度低也导致了无法针对特定场景开发更实用的应用。

FastGPT 优化了知识管理流程,简化了人工智能知识库的构建,并具备提升企业效率的高级特性,但 FastGPT 的智能体更多的是为知识库服务,因此定制化能力稍显不足,并且应用场景多围绕知识库展开,在其他领域稍显欠缺。

DB-GPT 变革了数据库交互方式,通过自然语言查询增强了数据库操作的安全性和灵活性,并支持多样化插件集成,然而,它同样存在与 FastGPT 类似的问题,即过分侧重于数据库的查询,而弱化了智能体的能力,例如,在 DB-GPT 中,智能体是通过代理工作流表达语言(Agentic Workflow Expression Language, AWEL)来创建的,而这种语言是基于一些给定的模块来进行编排的,当使用场景超出给定模块能够解决的范围时,构建智能体便会变得困难。

相比之下,ModelScope-Agent 则是一个相对灵活的平台,它不仅支持广泛的类型,还可以自定义工具并集成多种外部 API。这使用户能够根据需求轻松地构建和部署定制化智能体。此外,ModelScope-Agent 背后有强大的 ModelScope 社区支持,为用户提供最新的研究成果、教程和最佳实践,这些特点也使 ModelScope-Agent 成为本书中创建智能体的首选。当然,这并不意味着其他框架比 ModelScope-Agent 差,而是在本书对应的场景中,ModelScope-Agent 可以最大限度地为场景构建提供便利。

### 3.1.3 Single-Agent: 单智能体

单智能体系统涉及一个独立的智能体在特定环境中学习和做出决策。这个智能体的目标通常是最大化某种累积奖励信号,通过与环境的交互学习最优策略,例如,使用深度 Q 网络<sup>[32]</sup>或策略梯度方法如近端策略优化<sup>[33]</sup>(Proximal Policy Optimization, PPO)的机器人导航、游戏人工智能等都属于单智能体系统的应用。以下是单智能体的一些特点。

(1) 单一决策结构: 单智能体系统的核心在于拥有一个中央决策单元,该单元负责接收环境反馈、处理信息并做出决策。这种结构简化了决策过程设计,避免了多智能体系统中可能出现的协调难题和复杂的交互逻辑,使系统设计更加直观和易于理解。

(2) 直接环境交互: 智能体直接与外部环境互动,通过感知环境状态变化并采取行动以影响环境,进而达到其目标。这种直接交互模式使智能体的学习过程更加聚焦于自身行为对结果的影响。

(3) 学习目标明确: 单智能体系统通常围绕一个明确的目标进行优化,如最大化累积奖励、最小化错误率或成本等。这种明确的目标设定使智能体的学习具有量化性和可衡量性,便于采用强化学习、监督学习或无监督学习等技术进行训练。

(4) 算法相对简单: 相较于多智能体系统,单智能体的学习算法设计和实现通常更简单、直接。

(5) 易于调试和解释: 由于只有一个决策中心,所以单智能体系统的问题定位和调试工作相比多智能体系统要直接和简便。当智能体行为不符合预期时,可以通过跟踪决策过程和分析学习日志来快速识别问题所在,有利于算法的优化和改进。

(6) 广泛的应用性：单智能体系统因其设计的简洁性和高效性，被广泛地应用于游戏人工智能、推荐系统、自动化控制、自然语言处理、图像识别等多个领域。无论是提高个人用户的体验，还是优化工业生产线的效率，单智能体系统都能提供针对性的解决方案。

单智能体系统凭借其直观的决策架构、直接的环境互动、清晰的目标导向、算法的高度灵活性、便捷的调试过程及广泛的应用范围，已成为大模型引领下的新一代人工智能领域中不可或缺的组成部分。

### 3.1.4 Multi-Agent：多智能体

多智能体系统(Multi-Agent Systems, MAS)由两个或多个相互作用的智能体组成，它们可能追求相同或不同的目标，在合作、竞争或多种动机交织的环境中，智能体可以共同或独立地进行学习和决策。此外，这些系统通过监督、竞争和对抗等活动，能够实现更广泛、更开放的成果。这类系统模拟了现实世界中多方主体互动的场景，例如交通管控、团队协作型机器人、多人在线游戏等。多智能体系统的主要特点如下。

(1) 分布式决策与协同作业：多智能体系统的核心特征之一是存在多个独立的决策中心，每个智能体都有自己的感知、决策和行动能力。这些智能体通过相互交流与合作，实现对复杂任务的分布式处理和协同完成。这种分布式特性使系统能够处理超出单个智能体处理能力的大型问题，提高了问题解决的效率和范围。

(2) 交互与通信机制：多智能体系统内部的智能体之间存在直接或间接的交互，需要有效的通信协议和机制来传递信息。这包括但不限于直接消息传递、共享环境状态、隐含的交互规则等。合理的通信机制对于协调智能体行为、避免冲突、促进合作至关重要。在ModelScope-Agent中，多智能体的交互和通信机制可以选择使用Ray这样一个开源框架来实现。

(3) 复杂动态行为与涌现现象：多智能体系统经常展现出个体行为无法预测的集体动态和涌现现象，如自组织、集群行为、分层结构的形成等。这些涌现特性使系统能够适应复杂多变的环境，解决那些没有预设解决方案的问题，体现了高度的灵活性和创新能力。

(4) 多样化目标与策略：在多智能体系统中，不同的智能体可能追求不同的目标，甚至目标之间可能存在冲突。这要求系统设计必须考虑到目标的协调与冲突解决机制，如采用博弈论、合作与竞争模型等，以实现整体系统的稳定性和效率。

(5) 算法与学习方式的调整：多智能体系统的算法设计比单智能体更复杂，涉及多目标优化、信用分配、策略同步与异步更新及在部分可观测环境下的决策等问题。这可能需要考虑诸如多智能体深度强化学习、多智能体路径规划、联盟学习等高级算法的参与。

(6) 适应性和稳健性：多智能体系统在面对环境变化、智能体失效等情况时，表现出较强的适应性和稳健性。通过智能体之间的重新配置和任务分配，系统能够继续运作或快速恢复，这对于构建弹性系统、灾难响应、军事战略等应用尤为重要。

(7) 广泛的应用范围：多智能体系统被广泛地应用于多个领域，包括但不限于智能交通系统、无人机编队飞行、机器人足球、电子市场模拟、社交网络分析、网络安全性等。这些

应用展示了多智能体系统在处理大规模、动态、复杂问题上的潜力和优势。

多智能体系统以其分布式决策、协同作业、交互通信机制、复杂动态行为与涌现现象、多样化目标与策略、算法与学习方式的调整、适应性和稳健性及广泛的应用范围等特点，成为模拟和解决现实世界中多方主体互动问题的有效工具。这些系统通过智能体之间的相互作用，不仅能处理单个智能体难以应对的复杂任务，还能在多变的环境中展现出高度的灵活性和创新能力，为交通管理、机器人协作、在线游戏等多个领域提供了革命性的解决方案，展现了人工智能技术在处理大规模、动态问题上的巨大潜力。



11min

## 3.2 主要模块及参数配置

### 3.2.1 ModelScope-Agent 的组成模块

2.3 节深入剖析了智能体的关键模块。ModelScope-Agent 作为一个智能体框架，全面覆盖了智能体运行所需的所有基础功能模块。不仅如此，除了这些核心模块，ModelScope-Agent 还精心设计了一系列额外的功能模块。ModelScope-Agent 的核心系统架构如图 3-1 所示。

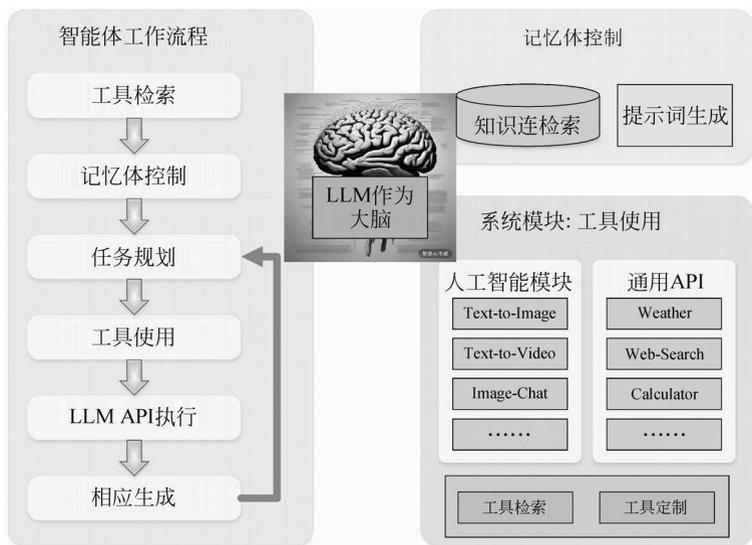


图 3-1 ModelScope-Agent 的核心系统架构<sup>[29]</sup>

智能体工作流定义了智能体如何执行其功能的一系列步骤。首先，智能体会根据需求检索合适的工具，然后它会对内存进行管理以存储和访问必要的信息。接下来，智能体会制订任务计划来指导其行动。在实际操作阶段，智能体会利用检索到的工具并调用相应的 API 来完成任务。API 执行完成后，智能体会产生一个响应或输出，从而结束这个工作流程。记忆体控制部分负责处理与知识检索和提示生成相关的工作。知识检索确保智能体可

以获取执行任务所需的信息。同时,提示词生成模块会生成系统提示和其他相关信息,帮助智能体理解用户的需求或上下文环境。系统模块则包含了工具库和一系列工具检索及自定义服务。工具库中包含了许多实用的工具和 API,如文本转图像、天气查询、计算器、地图导航、音乐播放器、购物应用等。此外,用户还可以根据自己的需求对这些工具进行检索和个性化定制。总体来讲,ModelScope-Agent 是一个全面而复杂的系统,它将训练框架、智能体工作流程、记忆体控制和系统模块有机地结合起来,实现了高效的人机交互和任务执行能力。

### 3.2.2 配置参数

在着手进行参数配置之前,再次审视智能体的 5 个核心构成模块:大语言模型、存储记忆、行为、感知和规划。接下来,将依照这 5 个模块逐一探讨配置的相关问题。首先,从一张图像生成的示例入手,这个示例是在官方提供的图像生成示例的基础上稍做调整而成的,旨在更清晰地阐述智能体参数配置的过程,示例代码如下:

```
//modelscope-agent/demo/chapter_3/Example_Code_3-1.py
import sys
import os
sys.path.append('/mnt/d/modelscope/modelscope-agent')
from modelscope_agent.agents.role_play import RolePlay

os.environ['DASHSCOPE_API_KEY']='YOUR_DASHSCOPE_API_KEY'

dialog_history = []
def test_image_gen_role():
    role_template = '你扮演一个画家,用尽可能丰富的描述信息调用工具绘制图像。'
    dialog_history.append(role_template)

    llm_config = {'model': 'qwen-max', 'model_server': 'dashscope'}
    dialog_history.append(llm_config)

    #input tool args
    function_list = ['image_gen']
    dialog_history.append(function_list)

    bot = RolePlay(
        function_list=function_list, llm=llm_config, instruction=role_template)

    response = bot.run('画一张猫的图像')
    dialog_history.append(response)

    text = ''
    for chunk in response:
        text += chunk
    print(text)
    assert isinstance(text, str)
```

```
print(dialog_history)

if __name__ == '__main__':
    test_image_gen_role();
```

上述示例代码主要由以下几部分构成。

(1) 导入必要的模块和设置路径：代码首先导入了 `sys` 和 `os` 模块，用于操作系统的路径设置，确保能够正确地引用 `ModelScope-Agent` 的相关功能模块。

(2) 配置大语言模型的 API 服务：在此部分，通过设置环境变量 `DASHSCOPE_API_KEY` 来配置大语言模型的 API 服务。这一步骤是关键，它使程序能够安全地与大语言模型进行交互。

(3) 初始化对话历史记录存储空间：为了实现多轮对话的连贯性，定义一个名为 `dialog_history` 的列表，用于存储对话的历史信息。

(4) 定义图像生成函数 `test_image_gen_role`：此函数封装了图像生成的逻辑，通过定义角色模板和配置大模型，使智能体能够按照指定的角色进行图像生成。

(5) 程序入口：在主程序入口部分，调用了 `test_image_gen_role` 函数，启动了整个图像生成过程。

特别值得一提的是，第(2)部分中的大语言模型配置，出于安全性考虑，可以在专门的配置文件中完成 API 的配置工作。该配置文件位于 `ModelScope-Agent` 的 `llm` 文件夹下，其中包含了多个大模型 API 的配置选项，如 `zhipu` 和 `openai` 等。

关于记忆存储模块，虽然在官方的单智能体示例中并未直接提供，但为了实现多轮对话的历史记录功能，示例代码中手动初始化了一个空的列表 `dialog_history`。

最后，行为模块的引入是代码的关键所在。在示例中，使用了一个典型的行为模块——图像生成工具 `image_gen`，这个工具实际上是一个函数，大模型通过调用它来生成所需的图像，从而实现了智能体的特定行为。

在上述示例中，由于仅涉及一个简单的执行动作，并未包含与外界的交互，因此并未涉及感知模块的调用。为了深化对感知模块的理解，以下呈现了一个更复杂的场景，示例代码：

```
//modelscope-agent/demo/chapter_3/Example_Code_3-2.py
import sys
import os
sys.path.append('/mnt/d/modelscope/modelscope-agent')
from modelscope_agent.agents.role_play import RolePlay

os.environ['DASHSCOPE_API_KEY']='YOUR_DASHSCOPE_API_KEY'

dialog_history = []
def test_image_gen_role():
    role_template = '你扮演一个画家，用尽可能丰富的描述信息调用工具绘制图像。'
    dialog_history.append(role_template)
```

```

llm_config = {'model': 'qwen-max', 'model_server': 'dashscope'}
dialog_history.append(llm_config)

#input tool args
function_list = ['image_gen', 'amap_weather', 'web_browser']
dialog_history.append(function_list)

bot = RolePlay(
    function_list=function_list, llm=llm_config, instruction=role_template)

response = bot.run('请查询当前地区的天气状况,并根据当前的天气状况绘制一张包括猫
和当地地标建筑的图像')
dialog_history.append(response)

text = ''
for chunk in response:
    text += chunk
print(text)
assert isinstance(text, str)
print(dialog_history)

if __name__ == '__main__':
    test_image_gen_role();

```

在上述代码示例中,绘制图像前需先确定所在地区的天气状况和地标建筑。地标建筑的信息对于大模型来讲可能已知,也可能因未曾学习而未知。至于天气状况,作为一种实时信息,大模型自然无法直接获取,因此,需要借助专门的感知工具进行查询。本示例运用了两种感知工具:一种是高德提供的基于地理位置的天气查询工具 `amap_weather`,另一种是网页浏览器工具 `web_browser`。通过这两个工具的协同作用,大模型得以感知到当地的地标建筑及当前的天气情况,进而基于这些整合信息创作出一张图像。下面给出了基于上述示例代码生成图片的过程:

**【示例 3-1】** 调用天气查询工具和图像生成工具根据输入位置和描述信息生成一幅图像

Action: `amap_weather`

Action Input: {"location": "武汉市"}

Observation: <result>武汉市的天气是晴温度是 27℃。</result>

Answer: 武汉市目前天气晴朗,气温为 27℃。接下来,我将根据这一天气状况,创作一张包含猫与武汉地标性建筑黄鹤楼的图像。设想在一个晴朗的日子里,一只猫坐在黄鹤楼前的台阶上,享受着温暖的阳光,远处的蓝天白云映衬着古老的塔楼,一切都显得格外宁静美好。

Action: `image_gen`

Action Input: {"text": "一只猫坐在晴朗天空下的武汉黄鹤楼前台阶上,享受阳光,黄鹤楼雄伟古老,背景是蓝天白云,气温宜人,绿树环绕。", "resolution": "1024 \* 1024"}

Observation: < result >! [IMAGEGEN] (<https://dashscope-result-sh.oss-cn-shanghai.aliyuncs.com/1d/f7/20240515/1b61f1c0/48f239fc-ce2f-4bc6-9e72-ac7617ee6712-1.png?Expires=1715848273&OSSAccessKeyId=LTAI5tQZd8AEcZX6KZV4G8qL&Signature=TWGU88IJh3RpCxvZ10eSEsP83m4%3D>)</result >

Answer: 根据您的要求,这是描绘了在晴朗天气下,一只猫坐在武汉黄鹤楼前台阶上享受阳光的画面。黄鹤楼雄伟地矗立着,背景是明媚的蓝天白云,整幅画面和谐而宁静。请欣赏这幅画作:![晴朗日子下的黄鹤楼与猫] (<https://dashscope-result-sh.oss-cn-shanghai.aliyuncs.com/1d/f7/20240515/1b61f1c0/48f239fc-ce2f-4bc6-9e72-ac7617ee6712-1.png?Expires=1715848273&OSSAccessKeyId=LTAI5tQZd8AEcZX6KZV4G8qL&Signature=TWGU88IJh3RpCxvZ10eSEsP83m4%3D>)

---

**注意:** 示例 3-1 中的 Action: amap\_weather 和 Action: image\_gen 表明智能体采取了行动,具体来讲调用了两个工具,一个是 amap\_weather,这是一个在线天气查询工具;另一个 image\_gen 是绘图工具; Action Input 则需要用户输入,为了便于用户理解,系统也给出了一些提示信息,例如 location,即要求输入位置,text 则要求输入文本信息; Observation 是智能体观察到的结果,是一个中间过程,为最终的输出答案服务,而用户需要重点关注的是 Answer 后输出的结果。

---

在上述输出示例中,展现了一个清晰的逻辑流程:

(1) 首先执行的是感知工具 amap\_weather,紧接着根据大模型从提示词中解析出的地理位置为武汉,并得到了武汉市的天气是晴,温度是 27℃ 的查询结果。

(2) 随后,将这一天气信息与原始提示词根据当前的天气状况绘制一张包括猫和当地地标建筑的图像相结合,构建了一个新的文生图提示词“武汉市目前天气晴朗,气温为 27℃”。接下来,根据这一天气状况,创作一张包含猫与武汉地标性建筑黄鹤楼的图像。设想在一个晴朗的日子里,一只猫坐在黄鹤楼前的台阶上,享受着温暖的阳光,远处的蓝天白云映衬着古老的塔楼,一切都显得格外宁静美好。

(3) 接着,通过调用图像生成工具 image\_gen,并结合上述文生图提示词,形成了图像生成的输入参数。text: 一只猫坐在晴朗天空下的武汉黄鹤楼前台阶上,享受阳光,黄鹤楼雄伟古老,背景是蓝天白云,气温宜人,绿树环绕。resolution: 1024 \* 1024。

(4) 最后,利用图像生成工具和生成的提示词,完成了图像的创作。由于大模型的编译环境未配备图像显示模块,因此提供了一个链接。通过这个链接,可以查看生成的图像,该图像如图 3-2 所示。

上述逻辑流程可视为智能体核心模块中的规划过程,其中涉及的天气查询工具和网页浏览器工具则是感知模块的典型代表。实际上,感知模块与行为模块之间存在诸多相似之



图 3-2 调用了天气查询和地标查询后生成的图像

处,如两者均涉及与大模型外部环境的交互及工具的调用,然而,感知模块的主要职能是将外部信息输入大模型,而行为模块则负责将大模型生成的内容输出至外部环境。

### 3.2.3 可直接调用的工具

ModelScope-Agent 官方给出了部分可以直接调用和配置的工具,但相关的解释较少,本书中,通过整理参考资料,给出了较为完善的工具介绍。

(1) image\_gen: 图像生成工具,需要提前在环境变量中配置 DASHSCOPE\_API\_KEY。通过调用通义万相文本生成图像大模型来根据用户输入的文字内容生成符合语义描述的多样化风格图像,克服了大语言模型无法生成图像的缺陷。支持的图像风格包括但不限于水彩、油画、中国画、素描、扁平插画、二次元、三维卡通等;支持中英文双语输入;支持输入参考图片进行参考内容或者参考风格迁移。风格迁移前后的效果如图 3-3 所示。



图 3-3 风格迁移前后的效果

注意：图 3-3 来源于 ModelScope-Agent 官方，链接如下：<https://help.aliyun.com/zh/dashscope/developer-reference/api-details-9?spm=a2c4g.11186623.0.0.592b2a19wVdY48>。

(2) code\_interpreter: 代码解释器。代码解释器使用了 Jupyter\_client 库，而 Jupyter\_client 库则是 Jupyter 生态系统中的一个核心库，扮演着桥梁的角色，负责在 Jupyter 的不同组件之间建立和管理通信。ModelScope-Agent 在处理代码相关问题时，大语言模型会调用 Jupyter\_client 来完成，进而获得更好的代码体验。

(3) web\_browser: 网页浏览器。网页浏览器在执行信息检索时，一般涉及 3 个关键步骤：通过搜索组件（如 GoogleSearchAPIWrapper）将查询转换为网址，利用加载工具（如 AsyncHtmlLoader 或 AsyncChromiumLoader）将网址转换为 HTML 页面，再通过转换技术（如 HTML2Text 或 BeautifulSoup）将 HTML 内容处理为格式化文本。网页浏览器的工作原理如图 3-4 所示。

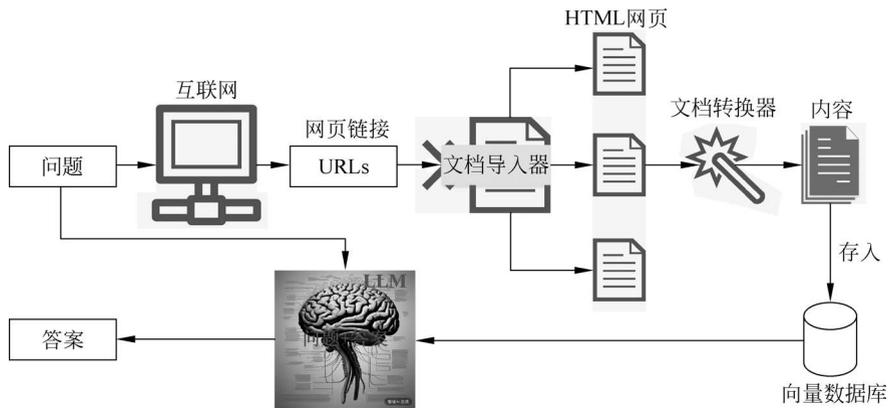


图 3-4 网页浏览器的工作原理

(4) amap\_weather: 实时天气情况查询。这里给出的实时天气情况查询调用了高德的服务，因此首先需要申请对应的 API，然后在环境变量中配置 AMAP\_TOKEN 即可。

(5) wordart\_texture\_generation: 艺术字纹理生成，需要提前在环境变量中配置 DASHSCOPE\_API\_KEY。生成可以对输入的文字内容或文字图片进行创意设计，根据提示词内容对文字添加材质和纹理，实现立体材质、场景融合、光影特效等效果，生成效果精美、风格多样的艺术字，结合背景可以直接作为文字海报使用。图 3-5 和图 3-6 展示了两个艺术字纹理生成的官方示例。

(6) qwen\_vl: 视觉理解工具，需要提前在环境变量中配置 DASHSCOPE\_API\_KEY。视觉理解工具是基于通义千问的开源视觉理解大模型 Qwen-VL 来实现的，可以实现通用 OCR、视觉推理、中文文本理解等基础能力及“看图做题”等高阶应用。

(7) style\_repaint: 人像风格重绘工具，需要提前在环境变量中配置 DASHSCOPE\_API\_KEY。人像风格重绘工具可以对输入的人物图像进行多种风格化的重绘生成，使新生成的图像在兼顾原始人物相貌的同时，带来不同风格的绘画效果，当前支持的风格有复古漫



图 3-5 以“瀑布流水”为预设风格生成的艺术字



图 3-6 以“国风建筑”为预设风格生成的艺术字

画、三维童话、二次元、小清新、未来科技等。该工具实现的功能与 image\_gen 有一定的重叠,但 image\_gen 处理风格迁移除外,还具备从无到有的图像生成能力。ModelScope-Agent 官方给出的人像风格重绘示例如图 3-7 所示。



图 3-7 人物风格重绘工具效果示例

---

**注意：**图 3-7 来源于 ModelScope-Agent 官方，链接如下：<https://help.aliyun.com/zh/dashscope/developer-reference/tongyi-wanxiang-style-repaint?spm=a2c4g.11186623.0.0.4446602dyvTuuy>。

---

(8) image\_enhancement：图像增强与超分辨率工具，需要提前在环境变量中配置 DASHSCOPE\_API\_KEY。图像增强与超分辨率是两种常用的图像处理技术，其中图像增强用于改善图片的亮度、对比度等，让图片看起来更鲜明，但不改变像素数量，而超分辨率则是利用算法将低清图片变成高清图，实际增加了像素，提升了细节，两者都是为了让图片质量更好，ModelScope-Agent 官方给出的图像增强与超分辨率示例如图 3-8 所示。



图 3-8 图像增强与超分辨率示例

---

**注意：**图 3-8 来源于 ModelScope-Agent 官方，链接如下：<https://github.com/dreamoving/Phantom>。

---

(9) speech-generation：语音生成工具，需要提前在环境变量中配置 DASHSCOPE\_API\_KEY。语音生成，又称语音合成，是一种能够将文本数据转换成逼真的人类语音输出的技术。这一过程涉及多个技术层面，核心是通过算法和模型模拟人类语音的特征，包括音节的发音、音调的变化、语速的快慢及各种语言特有的韵律，使机器能够像真人一样说话。

(10) video-generation：视频生成工具，需要提前在环境变量中配置 DASHSCOPE\_API\_KEY。视频生成工具以视频生成模型为基础，该模型基于多阶段文本到视频生成扩散模型，输入描述文本，返回符合文本描述的视频。目前仅支持英文输入。



6min

### 3.3 手把手教你搭建 ModelScope-Agent 应用环境

在探索智能体的过程中，掌握一套性能强劲的算法框架至关重要，而搭建高效的应用环境是每位开发者的必备技能。本节旨在提供一份详尽的 ModelScope-Agent 应用环境搭建指南，细致讲解从确认系统要求到安装运行环境，再到具体的安装步骤的每一环节。

### 3.3.1 准备工作

在开始搭建 ModelScope-Agent 应用环境之前,确保已经完成了以下准备工作。

(1) 系统要求: 推荐使用最新版本的 Linux 操作系统,因其广泛支持开发环境和稳定性较高。对于 Windows 用户,可能会遇到特定的兼容性问题,从而导致安装或运行不畅,因此可以考虑安装 Windows Subsystem for Linux (WSL),以获得更稳定的 Linux 环境体验。至于 macOS,未进行过具体测试,可以参考相关社区或官方文档验证具体应用的兼容性。

(2) 安装及运行环境搭建: 建议将 ModelScope-Agent 部署到虚拟环境中,以便与其他环境隔离。为此,建议安装 Docker、Conda 或 pip 中的一种虚拟环境,而具体虚拟环境的创建可以通过开源社区查询获得。

(3) Python 环境配置: 安装 Python 并设置好 Python 环境,推荐 3.11.9 版本。

(4) 安装依赖: 确保安装必要的依赖包,如通过 `pip install -r requirements.txt` 命令来安装所有必需的依赖,这些是运行 ModelScope-Agent 及其他模型的基础。

### 3.3.2 几种推荐的安装方式

下面将分别介绍如何在 Ubuntu 20.04 系统中通过两种不同方式安装 ModelScope-Agent: 一种是利用 Docker 容器进行安装,另一种是直接在系统环境中通过创建 Conda 或 pip 环境来进行安装。

#### 1. 直接在 Docker 容器中进行安装

首先假设 Docker Hub 中有 ModelScope-Agent 的 Docker 镜像,安装方式如下。

(1) 首先确保 Ubuntu 系统上已安装 Docker。如果未安装,则可以通过命令行方式安装,命令如下:

```
sudo apt update
sudo apt install docker.io
```

(2) 启动 Docker,命令如下:

```
sudo systemctl start docker
sudo systemctl enable docker
```

(3) 获取 ModelScope-Agent Docker 镜像。

查找 ModelScope-Agent 的官方 Docker 镜像: 访问 Docker Hub 或 ModelScope 的 GitHub 页面查看是否有官方发布的 Docker 镜像。这里假设有一个名为 `modelscope-agent` 的镜像(实际名称以官方发布为准),则可以拉取 Docker 镜像,代码如下:

```
docker pull modelscope-agent
```

(4) 下载 ModelScope-Agent 的工程,有两种方法,一种是直接从 GitHub 上手动下载;另一种是通过 git 克隆,代码如下:

```
git clone https://github.com/modelscope/modelscope-agent.git
cd modelscope-agent
```

(5) 利用拉取的镜像创建容器,命令如下:

```
docker run --name llama-pro -itd --shm-size=1g --gpus all -v /mnt/d/llama-pro:/: /
llama-pro ID_modelscope-agent_images /bin/bash
```

(6) 查看 Docker 容器的 ID\_container,命令如下:

```
docker ps -a 或 docker ps
```

(7) 开启 Docker 容器,命令如下:

```
docker start ID_container
```

(8) 进入 Docker 容器,命令如下:

```
docker exec -it ID_container /bin/bash
```

如果无法在 Docker Hub 上找到对应的镜像,则需要基于一个基础镜像来创建对应的容器,与上述步骤最大的不同在于需要自己安装依赖。

(1) 首先确保 Ubuntu 系统上已安装 Docker。如果未安装,则可以通过命令行方式安装,命令如下:

```
sudo apt update
sudo apt install docker.io
```

(2) 启动 Docker,命令如下:

```
sudo systemctl start docker
sudo systemctl enable docker
```

(3) 获取 ModelScope-Agent 工作环境镜像,假设该镜像的名称为 NVIDIA/cuda:11.7.1-cuDNN8-devel-Ubuntu20.04,则可以拉取该 Docker 镜像,命令如下:

```
docker pull NVIDIA/cuda:11.7.1-cuDNN8-devel-Ubuntu20.04
```

(4) 下载 ModelScope-Agent 的工程,有两种方法,一种是直接从 GitHub 上手动下载;另一种是通过 git 克隆,命令如下:

```
git clone https://github.com/modelscope/modelscope-agent.git
cd modelscope-agent
```

(5) 利用拉取的镜像创建容器,命令如下:

```
docker run --name llama-pro -itd --shm-size=1g --gpus all -v /mnt/d/llama-pro:/llama-pro ID_modelscope-agent_images /bin/bash
```

(6) 查看 Docker 容器的 ID\_container, 命令如下:

```
docker ps -a 或 docker ps
```

(7) 开启 Docker 容器, 命令如下:

```
docker start ID_container
```

(8) 进入 Docker 容器, 命令如下:

```
docker exec -it ID_container /bin/bash
```

(9) 在 Docker 容器内安装 ModelScope-Agent 运行所必需的依赖, 命令如下:

```
pip install -r requirements.txt
```

显然在有 ModelScope-Agent Docker 镜像和没有 ModelScope-Agent Docker 镜像两种情况下, 最大的区别在于步骤(3)和步骤(9), 在有 ModelScope-Agent Docker 镜像的情况下, 并不需要步骤(9)。

## 2. 直接在 Ubuntu 系统环境安装 ModelScope-Agent

(1) 下载 Miniconda 安装包, 命令如下:

```
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
```

或者

```
curl -O https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
```

(2) 验证下载文件的完整性(可选), 命令如下:

```
sha256sum Miniconda3-latest-Linux-x86_64.sh
```

(3) 安装 Miniconda, 命令如下:

```
chmod +x Miniconda3-latest-Linux-x86_64.sh  
./Miniconda3-latest-Linux-x86_64.sh
```

安装过程中会显示许可协议, 按需阅读后, 输入 yes 继续安装。安装程序会询问是否愿意将 Miniconda 的 bin 目录添加到你的 PATH 环境变量中, 通常推荐输入 yes。最后, 安装完成后会询问是否希望安装 Anaconda 的额外软件包, 可以根据个人需求进行选择。

(4) 完成安装和验证, 命令如下:

```
source ~/.bashrc
```

或者

```
source ~/.zshrc
```

验证安装是否成功可以输入的命令如下：

```
conda --version
```

(5) 创建 Conda 环境,命令如下：

```
conda create -n myenv python=3.9
```

(6) 激活这个环境,命令如下：

```
conda activate myenv
```

### 3. 创建虚拟环境安装 ModelScope-Agent

除了上述两种方法外,还有一种更简单的方式,即直接创建虚拟环境。这种方法只要保证当前系统中安装好了 Python 及最新版的 pip,便可以通过以下方式创建虚拟环境。

(1) 创建一个新目录,用于存放虚拟环境,命令如下：

```
mkdir my_project  
cd my_project
```

(2) 创建虚拟环境,命令如下：

```
python3 -m venv env
```

(3) 激活虚拟环境,命令如下：

```
source env/bin/activate
```

(4) 退出虚拟环境,命令如下：

```
deactivate
```

---

**注意：**安装完虚拟环境后并不能直接进行后续工作,依然要安装对应的依赖包等。

---

### 3.3.3 配置大模型 API

ModelScope-Agent 与大模型交互时,需要配置 API 密钥以访问 ModelScope 平台(或其他大模型应用平台,例如智谱 AI、讯飞星火和通义千问等)上的模型服务。本书以 ModelScope 平台为例,具体步骤如下。

(1) 注册 ModelScope 账号:访问 ModelScope 官网,注册账号并登录。

(2) 获取 API 密钥:在账号设置中找到 API 密钥部分,生成一个新的密钥对。

(3) 配置环境变量：将 API 密钥设置为环境变量，以便 ModelScope-Agent 能够访问。在终端中，可以这样操作(将 YOUR\_API\_KEY\_HERE 替换为实际密钥)，代码如下：

```
export MODELSCOPE_API_KEY=YOUR_API_KEY_HERE
```

或者在代码中直接指定：

```
from modelscope.utils.config import Config
config = Config()
config.api_key = 'YOUR_API_KEY_HERE'
```

## 3.4 本章小结

本章主要介绍了 ModelScope-Agent 智能体框架的概念、特点及应用环境的搭建方法。ModelScope-Agent 是一个灵活且通用的平台，不仅提供了模型部署工具，还集成了模型管理、多工具集成和综合框架定制等功能，使用户能够轻松地构建各种人工智能应用。与同类框架相比，ModelScope-Agent 具有显著的优势和特色，能够满足不同场景的需求。

本章详细阐述了单智能体和多智能体系统的特点，以及 ModelScope-Agent 的主要模块和参数配置。通过具体示例，展示了如何配置大模型、存储记忆、行为等模块，以及如何调用官方提供的工具。在应用环境搭建方面，提供了详细的准备工作、安装步骤和配置大模型 API 的方法。介绍了在 Linux 系统下通过 Docker 容器和 Conda 或 pip 环境安装 ModelScope-Agent 的两种方式，为开发者搭建高效的应用环境提供了指导。