中文 | English

Qwen-Agent是一个开发框架。开发者可基于本框架开发Agent应用，充分利用基于通义千问模型（Qwen）的指令遵循、工具使用、规划、记忆能力。本项目也提供了浏览器助手、代码解释器、自定义助手等示例应用。

更新

- Dec 3, 2024: GUI 升级为基于 Gradio 5。注意：如果需要使用GUI，Python版本需要3.10及以上。
- 🔥🔥🔥 Sep 18, 2024: 新增Qwen2.5-Math Demo以展示Qwen2.5-Math基于工具的推理能力。注意：代码执行工具未进行沙箱保护，仅适用于本地测试，不可用于生产。

开始上手

安装

- 从 PyPI 安装稳定版本：

```
pip install -U "qwen-agent[rag,code_interpreter,python_executor,gui]"
# 或者，使用 `pip install -U qwen-agent` 来安装最小依赖。
# 可使用双括号指定如下的可选依赖：
#   [gui] 用于提供基于 Gradio 的 GUI 支持；
#   [rag] 用于支持 RAG；
#   [code_interpreter] 用于提供代码解释器相关支持；
#   [python_executor] 用于支持 Qwen2.5-Math 基于工具的推理。
```

- 或者，你可以从源码安装最新的开发版本：

```
git clone https://github.com/QwenLM/Qwen-Agent.git
cd Qwen-Agent
pip install -e ./"[rag,code_interpreter,python_executor]"
# 或者，使用 `pip install -e ./` 安装最小依赖。
```

如果需要内置 GUI 支持，请选择性地安装可选依赖：

```
pip install -U "qwen-agent[gui,rag,code_interpreter]"
# 或者通过源码安装 `pip install -e ./"[gui,rag,code_interpreter]"`
```

准备：模型服务

Qwen-Agent支持接入阿里云DashScope服务提供的Qwen模型服务，也支持通过OpenAI API方式接入开源的Qwen模型服务。

- 如果希望接入DashScope提供的模型服务，只需配置相应的环境变量 `DASHSCOPE_API_KEY` 为您的DashScope API Key。
- 或者，如果您希望部署并使用您自己的模型服务，请按照Qwen2的README中提供的指导进行操作，以部署一个兼容OpenAI接口协议的API服务。 具体来说，请参阅vLLM一节了解高并发的GPU部署方式，或者查看Ollama一节了解本地CPU（+GPU）部署。

快速开发

框架提供了大模型（LLM，继承自 `class BaseChatModel` ，并提供了Function Calling功能）和工具（Tool，继承自 `class BaseTool` ）等原子组件，也提供了智能体（Agent）等高级抽象组件（继承自 `class Agent` ）。

以下示例演示了如何增加自定义工具，并快速开发一个带有设定、知识库和工具使用能力的智能体：

```python
import pprint
import urllib.parse
import json5
from qwen_agent.agents import Assistant
from qwen_agent.tools.base import BaseTool, register_tool


# 步骤 1（可选）：添加一个名为 `my_image_gen` 的自定义工具。
@register_tool('my_image_gen')
class MyImageGen(BaseTool):
    # `description` 用于告诉智能体该工具的功能。
    description = 'AI 绘画（图像生成）服务，输入文本描述，返回基于文本信息绘制的
    # `parameters` 告诉智能体该工具有哪些输入参数。
    parameters = [{
        'name': 'prompt',
        'type': 'string',
        'description': '期望的图像内容的详细描述',
        'required': True
    }]

    def call(self, params: str, **kwargs) -> str:
        # `params` 是由 LLM 智能体生成的参数。
        prompt = json5.loads(params)['prompt']
        prompt = urllib.parse.quote(prompt)
        return json5.dumps(
            {'image_url': f'https://image.pollinations.ai/prompt/{prompt}'}
            ensure_ascii=False)


# 步骤 2：配置您所使用的 LLM。
llm_cfg = {
    # 使用 DashScope 提供的模型服务：
    'model': 'qwen-max',
    'model_server': 'dashscope',
    # 'api_key': 'YOUR_DASHSCOPE_API_KEY',
    # 如果这里没有设置 'api_key'，它将读取 `DASHSCOPE_API_KEY` 环境变量。

    # 使用与 OpenAI API 兼容的模型服务，例如 vLLM 或 Ollama：
    # 'model': 'Qwen2-7B-Chat',
    # 'model_server': 'http://localhost:8000/v1',  # base_url，也称为 api_ba
    # 'api_key': 'EMPTY',

    # （可选） LLM 的超参数：
    'generate_cfg': {
        'top_p': 0.8
    }
}

# 步骤 3：创建一个智能体。这里我们以 `Assistant` 智能体为例，它能够使用工具并读
system_instruction = '''你是一个乐于助人的AI助手。
在收到用户的请求后，你应该：
- 首先绘制一幅图像，得到图像的url，
```

```
- 然后运行代码`request.get`以下载该图像的url，
- 最后从给定的文档中选择一个图像操作进行图像处理。
用 `plt.show()` 展示图像。
你总是用中文回复用户。'''
tools = ['my_image_gen', 'code_interpreter']  # `code_interpreter` 是框架自
files = ['./examples/resource/doc.pdf']  # 给智能体一个 PDF 文件阅读。
bot = Assistant(llm=llm_cfg,
                system_message=system_instruction,
                function_list=tools,
                files=files)

# 步骤 4：作为聊天机器人运行智能体。
messages = []  # 这里储存聊天历史。
while True:
    # 例如，输入请求 "绘制一只狗并将其旋转 90 度"。
    query = input('用户请求: ')
    # 将用户请求添加到聊天历史。
    messages.append({'role': 'user', 'content': query})
    response = []
    for response in bot.run(messages=messages):
        # 流式输出。
        print('机器人回应:')
        pprint.pprint(response, indent=2)
    # 将机器人的回应添加到聊天历史。
    messages.extend(response)
```

除了使用框架自带的智能体实现（如 `class Assistant` ），您也可以通过继承 `class Agent` 来自行开发您的智能体实现。

框架还提供了便捷的GUI接口，支持为Agent快速部署Gradio Demo。 例如上面的例子中，可以使用以下代码快速启动Gradio Demo：

```
from qwen_agent.gui import WebUI
WebUI(bot).run()  # bot is the agent defined in the above code, we do not r
```

现在您可以在Web UI中和Agent对话了。更多使用示例，请参阅examples目录。

FAQ

支持函数调用（也称为工具调用）吗？

支持，LLM类提供了函数调用的支持。此外，一些Agent类如FnCallAgent和ReActChat也是基于函数调用功能构建的。

如何让AI基于超长文档进行问答?

我们已发布了一个快速的RAG解决方案，以及一个虽运行成本较高但准确度较高的智能体，用于在超长文档中进行问答。它们在两个具有挑战性的基准测试中表现出色，超越了原生的长上下文模型，同时更加高效，并在涉及100万字词上下文的"大海捞针"式单针查询压力测试中表现完美。欲了解技术细节，请参阅博客。

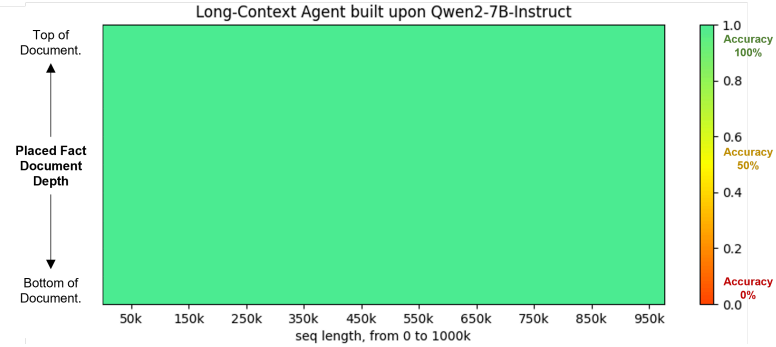
Result 1. Comparison among the Native Model, RAG, and Agent

Needle-Bench	(0k, 8k]	(8k, 32k]	(32k, 128k]	(128k, 256k]
32k-Model	87.50	81.19	46.28	0.41
4k-RAG	85.75	78.43	73.26	70.00
4k-Agent	85.41	85.43	85.52	81.82

LV-Eval	(0k, 16k]	(16k, 32k]	(32k, 64k]	(64k, 128k]	(128k, 256k]
32k-Model	49.06	45.72	29.17	12.01	1.32
4k-RAG	49.28	48.90	49.14	49.24	45.80
4k-Agent	51.61	51.52	52.15	51.37	46.60

(Blue = Significantly Better than 32k-Model; Red = Significantly Worse than 32k-Model)

Result 2. Finding the Needle in a 1M-Context Hay Stack



应用： BrowserQwen

BrowserQwen 是一款基于 Qwen-Agent 构建的浏览器助手。如需了解详情，请参阅其文档。

免责声明

代码解释器未进行沙盒隔离，会在部署环境中执行代码。请避免向Qwen发出危险指令，切勿将该代码解释器直接用于生产目的。