

基于 AutoGen 具备 function-call、rag 能力的 multi-agent 金融 AI 助理

AutoBnc

1. 简介

在当今金融行业中，随着科技的迅猛发展和数据量的激增，传统的金融分析和决策方式已经难以满足市场需求。金融市场日益复杂，投资者和金融机构需要更加智能、高效和精准的工具来进行数据分析、风险评估和投资决策。人工智能（AI）和多代理系统（Multi-Agent Systems）为解决这些挑战提供了新的思路和解决方案。

AutoGen 是一个强大的框架，结合了功能性和基于 RAG（Retrieval-Augmented Generation）的信息检索能力，能够实现复杂的多代理协作。利用 AutoGen，可以开发一个具备多代理系统特性的金融 AI 助理，能够在金融数据分析、市场预测、交易决策和风险管理等多个方面提供支持和自动化服务。

本项目旨在开发一个基于 AutoGen 的多代理金融 AI 助理，利用其强大的数据处理和分析能力，为用户提供全面的金融服务。通过集成多种先进的 AI 技术，包括自然语言处理（NLP）、机器学习和深度学习等，本项目将打造一个智能、高效、可靠的金融 AI 助理系统，以帮助用户在瞬息万变的金融市场中获得竞争优势。

本项目的目标包括：

- 构建一个具备信息检索（RAG）能力的多代理系统，能够高效地处理和分析海量金融数据。
- 实现自动化的金融数据收集、处理和分析，提供实时的市场预测和投资建议。

通过本项目的开发和实施，期望能够显著提升金融数据分析和决策的效率和准确性，推动金融科技的发展和创新，为用户提供更优质的金融服务体验。

2. 系统设计

本系统设计了多个智能代理，每个代理有各自的职责和功能，共同协作完成金融 AI 助理的整体任务。系统主要分为以下几个模块：


```
@dataclass
class PastRun:
    feedback: str
    intents_info: str
```

Endreason: 记录对话结束的标志:

```
class EndReason(Enum):
    TERMINATE = "TERMINATE"
    GOAL_NOT_SUPPORTED = "GOAL_NOT_SUPPORTED"
```

RunResult: 记录运行后的谈话记录、总消耗、结束原因、系统消息记录等。

```
@dataclass
class RunResult:
    summary: str
    chat_history_json: str
    intents: list[Intent]
    end_reason: EndReason
    total_cost_without_cache: float
    total_cost_with_cache: float
    info_messages: list[str]
```

Intent: 构建订单的数据结构，方便 `biananceSystem` 进行相关操作:

```
class IntentType(str, Enum):
    SEND = "send"
    BUY = "buy"
    SELL = "sell"

class IntentBase(BaseModel):
    type: IntentType
    summary: str
```

3.2 Agents

具体通过 `agents` 目录下的多个 `agents` 进行了解，具体的形式主要如下:

```
def build(user_proxy: UserProxyAgent, agents_information:
str, interactive: bool,
         get_llm_config) -> AssistantAgent:
```

需要传的参数主要有: 用户代理 `agent`, `agent` 信息, 是否可以交互的 `bool` 变量。
详情的 `agent` 定义文档可以参照 `autogen` 官网教程:

[Getting Started | AutoGen \(microsoft.github.io\)](https://microsoft.github.io/autogen/Getting%20Started/)

3.3 tool 绑定

如果 agent 要实现 functioncall 的功能，需要进行 tool 的绑定，其中 tool 绑定的相关工具文件为 tool.build.py:

```
from typing import Any, Callable, TYPE_CHECKING,
Coroutine, Union
from autogen import AssistantAgent, UserProxyAgent
import autogen
if TYPE_CHECKING:
    from autobnc.AutoBnc import AutoBnc

class FunctionBase:
    name: str
    description: str

    def build(self, autobnc: 'AutoBnc') -> Union[Callable[...,
Any] , Callable[..., Coroutine[Any, Any, Any]]]:
        raise NotImplementedError

    def register(self, autobnc: 'AutoBnc', caller:
AssistantAgent, user_proxy: UserProxyAgent) -> None:
        func = self.build(autobnc)
        autogen.agentchat.register_function(
            func,
            caller=caller,
            executor=user_proxy,
            description=self.description,
        )
```

3.4 订单执行（BinanceSystem）

将讨论中生成的所有 intent，即订单意愿统一交由 bianceSystem.py 进行处理，保证了订单执行过程中的安全性以及可控性。

交易函数如下：

```
from autobnc.intent import Intent, IntentType
from typing import List
from binance.client import Client
def run_intents(client: Client, intents: List[Intent]):
    results_with_info = []
    print("begin run intents:")
    for intent in intents:
        print(intent.summary, '\n')
```

```

if intent.type == IntentType.BUY:
    order = client.order_market_buy(
        symbol=intent.symbol,
        quantity=intent.amount
    )
    results_with_info.append((order, intent.summary))
if intent.type == IntentType.SELL:
    order = client.order_market_sell(
        symbol=intent.symbol,
        quantity=intent.amount
    )
    results_with_info.append((order, intent.summary))
if intent.type == IntentType.SEND:
    order = client.withdraw(
        asset=intent.symbol,
        address=intent.receiver,
        amount=intent.amount,
        network=intent.network # This parameter is optional
    )
    results_with_info.append((order, intent.summary))
return results_with_info

```

3.5 AutoBnc 管理

AutoBnc 是一个集成了多代理系统的金融 AI 助理框架，旨在为用户提供自动化和智能化的金融数据分析、市场预测、交易执行和风险管理服务。该框架利用了 Binance API 来访问市场数据，并通过多代理系统的协作实现高效和精确的任务执行。以下是 AutoBnc.py 的详细介绍：

3.5.1 核心组件

1. Client

Binance API 的客户端，用于访问实时市场数据和执行交易操作。

2. Intent

意图对象，定义了 binancesystem 需要完成的具体任务。

3. Manager

管理代理，负责协调和管理其他代理的工作，确保系统高效运行。

4. Agents*

- 各种功能的代理（user_proxy、manager、clarifier_agent、trader_agent、scientist_agent、transfer_agent），分别负责不同的任务，如用户交互、数据分析、交易执行等。

5. Config

- 系统配置文件，定义了各代理的参数和系统运行的设置。

6. RunResult

- 运行结果对象，记录了每次任务执行的结果，包括摘要、聊天历史、意图列表、结束原因、成本等信息。

7. PastRun

- 记录过去运行的任务和用户反馈，用于优化和改进后续任务的执行。

8. EndReason

- 定义任务结束的原因，包括正常结束、用户终止等。

9. Color

- 颜色常量，用于在终端输出中高亮显示不同类型的消息。

10. Constants

- 系统常量，定义了 **Binance API** 密钥等重要参数。

11. BinanceSystem

- **Binance** 系统模块，包含了运行意图的具体实现。

3.5.2. 任务执行与管理

- 通过 **Manager** 代理协调多个功能代理的工作，执行用户请求的任务。支持非交互式和交互式两种运行模式。

- 通过 **Binance API** 获取实时市场数据，并由 **Scientist Agent** 进行分析和处理。

- 由 **Trader Agent** 创建和执行虚拟货币的买卖订单。

- User Proxy** 代理模拟用户与系统的交互，确保系统理解并完成用户的目标。

- Clarifier Agent** 检查用户输入是否符合当前上下文，并在必要时提示用户重新输入。

- 记录和监控每次任务执行的成本，包括使用缓存和不使用缓存的情况。

以下是一个简单的使用示例，展示了如何初始化和运行 **AutoBnc**:

```
from autobnc.agents import
manager,transfer_agent,user_proxy,clarifier_agent,scientist_agent
from autobnc.agent_tool import get_agents_information
import autogen
from textwrap import dedent
from autobnc.util.constants import BINANCE_API_KEY
import os
if __name__ == '__main__':

    AgentInfoLIST =
[transfer_agent.transfer_info(),clarifier_agent.clarifier_info(),scientist_agent.scientis
t_info()]
    information = get_agents_information(AgentInfoLIST)

    config_list = autogen.config_list_from_json(
```

```

env_or_file="OAI_CONFIG_LIST.json",
filter_dict={
    "model": ["gpt-4"],
},
)

gpt3_config = {
    "cache_seed": 42, # change the cache_seed for different trials
    "temperature": 0,
    "config_list": config_list,
    "timeout": 120,
}
print(type(gpt3_config))
print("user begin:")
prompt = input()
print("Running AutoTx with the following prompt: " + prompt)
user = user_proxy.build(prompt,information,gpt3_config)
print("transfer begin:")
transfer = transfer_agent.build(user,gpt3_config)
print("clarifier_agent begin:")
clarifier = clarifier_agent.build(user,information,False,gpt3_config)
scientist = scientist_agent.build(user, information, False, gpt3_config)
Agents = [user,transfer,clarifier,scientist]

manager = manager.build(Agents,10,False,gpt3_config)

chat = user.initiate_chat(
    manager,
    message=dedent(
        f"""
            I am currently connected with the Binance_API_Key:
{BINANCE_API_KEY},

            My goal is: {prompt}
        """
    )
)
print("char finished")

if "ERROR:" in chat.summary:
    error_message = chat.summary.replace("ERROR: ", "").replace("\n", "")
    print(error_message, "red")
else:
    print(chat.summary, "green")

```

```
is_goal_supported = chat.chat_history[-1]["content"] != "Goal not supported:
TERMINATE"
```

```
os.system('pause')
```

AutoBnc 通过 **Manager** 代理协调各功能代理之间的协作，每个代理专注于其特定任务，并在必要时与其他代理协同工作。例如，**User Proxy** 代理接收用户请求并传递给 **Manager** 代理，**Manager** 代理根据请求选择合适的功能代理执行任务，如 **Trader Agent** 进行交易执行，**Scientist Agent** 进行数据分析等。

AutoBnc 通过集成多代理系统和 **Binance API**，为用户提供了一个功能强大且高效的金融 AI 助理，能够在复杂多变的金融市场中提供可靠的支持和服务。