ChatGPT

ExchangeV2 项目在 Google Colab Pro+ 上的使用说明

1. 环境准备与配置

1.1 启用 GPU 运行时：在 Colab Pro+ 中，新建 Notebook 后，点击菜单的「Runtime」>「Change runtime type」，将硬件加速器设置为 GPU，以便利用 15GB 显存加速计算。

1.2 挂载 Google Drive（可选）：建议挂载谷歌云盘以持久化实验数据。例如，在 Colab 单元格中执行：

```
from google.colab import drive
drive.mount('/content/drive')
```

这样会将您的云盘挂载到 `/content/drive` 路径，方便保存模型和日志 [1]。

1.3 克隆代码仓库：获取 `xiaoa5/ExchangeV2` 项目代码。在 Colab 中运行以下命令克隆 GitHub 仓库并进入项目目录 [2]：

```
!git clone https://github.com/xiaoa5/ExchangeV2.git
%cd ExchangeV2
```

1.4 安装依赖：项目依赖包括 PyTorch、Gymnasium 等。在克隆仓库后，安装所需的 Python 包 [3]：

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

安装完成后，确保依赖版本满足要求，例如 PyTorch ≥2.0、Gymnasium ≥0.29 等 [4] [5]。

1.5 环境验证：可选地，验证当前运行环境。例如，检查是否正确使用 GPU：

```
import torch
if torch.cuda.is_available():
    print(f"✓ Using GPU: {torch.cuda.get_device_name(0)}")  # 打印GPU名称和显存
    print(f"  Memory: {torch.cuda.get_device_properties(0).total_memory/1e9:.
1f} GB")
else:
    print("⚠ GPU not available, using CPU")
```

上述代码将显示所选 GPU 的名称和总显存 [6]。Colab Pro+ 提供约15GB显存，可满足大规模仿真需求。

2. 运行项目

2.1 启动仿真实验

准备好环境后，可以启动多智能体交易仿真实验。您可以直接使用项目提供的环境类 `Environment` 来设置实验参数并运行。以下是一个基本示例：

```
from configs.time_config import TimeConfig, TimeConfigPresets
from environment import Environment
from agents.random_agent import RandomAgent

# 配置时间参数：每20步形成一根K线
time_config = TimeConfigPresets.HIGHFREQ  # 高频交易预设: 1分钟K线，每根K线20步 [7]
time_config = TimeConfig(steps_per_bar=20, bar_interval="5min",
bars_per_day=200, trading_hours=4.0)
# 上面自定义TimeConfig将每20步合成1根K线，每日产生200根K线，共4小时交易。总Steps =
20*200 = 4000。

# 创建仿真环境，设定仿真天数
env = Environment(time_config=time_config, num_days=1)  # 1个交易日，大约4000步
print(f"总Steps设置: {env.time_config.steps_per_day * env.num_days}")  # 验证总
步数
```

上述代码通过 `TimeConfig` 指定了**步长和K线周期**，使得每20个 step 聚合成1根K线 [8]。我们将 `bars_per_day` 设为200，这样1天内生成约200根K线，对应总共约4000个 step。您也可以直接使用内置的 `HIGHFREQ` 预设，其 `steps_per_bar=20` [7]，然后通过调整 `num_days` 来控制总步数。

接下来，添加智能体并运行仿真：

```
# 添加智能体，例如50个随机交易智能体
for i in range(50):
    agent = RandomAgent(agent_id=i, initial_cash=100000.0,
trade_probability=0.3)
    env.add_agent(agent)  # 将智能体加入环境 [9]
print(f"添加智能体数量: {len(env.agents)}")  # 验证智能体总数

# 运行仿真实验
stats = env.run(verbose=True)  # verbose=True 将每隔100步打印进度 [10]
```

上述示例创建了50个随机智能体（每个初始资金10万，30% 概率参与每步交易）并运行仿真 [9] [11]。`env.run()` 将自动执行仿真循环直到完成预设的步骤数，并返回统计结果 [12]。在输出中，可以看到总步数、当前K线索引、交易天数等进度信息 [13]。例如，每经过100步会打印一次进度，包含当前已执行百分比、Step编号、Bar编号等 [14]。

提示：您也可以运行项目自带的示例或测试脚本来验证运行是否正常。例如执行：

```
!python project/tests/test_day5.py       # 完整环境集成测试
!python project/examples/complete_example.py   # 运行完整示例脚本
```

以确保环境配置正确且核心功能正常[15]。这些脚本会构建环境、运行一段仿真并输出结果供您参考。

2.2 配置参数文件

ExchangeV2 提供了多个配置模块，允许灵活调节实验参数：

- **时间配置（`configs/time_config.py`）**：决定 Step 与 K线、交易日的对应关系。您可以使用内置的 `TimeConfigPresets` 快速设定常用周期（如 HIGHFREQ/标准/日线等）[7]。例如上述示例使用的 HIGHFREQ 预设对应每1分钟一个Bar，每个Bar 20个Step，每日240根K线[7]。也可以自定义 `TimeConfig` 实例来调整 `steps_per_bar`（每根K线包含步数）和 `bars_per_day`（每日K线数）等参数。

- **订单簿配置（`configs/orderbook_config.py`）**：控制订单簿深度、价格范围、TTL等。例如 `OrderBookConfig(max_depth_per_side=1000, default_ttl=50, price_range_pct=0.10)` 定义每侧最多挂1000单、默认订单寿命50步、价格撮合范围±10%[16]。

- **撮合配置（`configs/match_config.py`）**：设置撮合引擎参数，如手续费率、是否允许自成交等。例：`MatchConfig(fee_rate=0.001, enable_self_trade=False)` 表示手续费万分之一，禁止自成交[17]。

这些配置可在创建 `Environment` 时传入：例如

```
from configs import orderbook_config, match_config
env = Environment(time_config=time_config,
                  orderbook_config=orderbook_config.OrderBookConfig(...),
                  match_config=match_config.MatchConfig(...),
                  num_days=1)
```

如果不传，则会使用默认配置。通过调整配置文件或构造参数，可以**微调实验环境**，例如改变手续费、订单有效期等，以符合您的实验需求。

3. 关键参数调节

进行多智能体交易实验时，以下关键参数可以控制实验规模和过程：

- **智能体数量**：智能体数量由您添加的代理个数决定。可通过多次调用 `env.add_agent()` 增加智能体。例如，循环添加50个 `RandomAgent` 已在上例演示[9]。最初建议从较少的智能体开始（如几十个），待系统稳定再逐步增加到上百甚至上千。根据项目文档，仿真框架支持同时运行100+个智能体[18]且性能良好；内部压力测试中甚至模拟了1000个智能体并发交易[19]。在 Colab Pro+ 资源下，1000个智能体×5000步的大规模仿真在几分钟内即可完成[19]。

- **步数（`steps`）**：每次实验运行的总步数决定了仿真持续的时间长短。在 `Environment` 中总步数由时间配置和仿真天数推算得出，即 `total_steps = steps_per_day * num_days`[20]。您可以

通过调整 `num_days` 或自定义 `steps_per_bar` / `bars_per_day` 来改变总步数。例如，使用前述自定义配置将 `steps_per_bar=20` 且 `bars_per_day=200`，配合 `num_days=1`，则总步数约为20×200=4000步，相当于200根K线。若希望增加步数，可提高 `bars_per_day` 或增加 `num_days`；反之减少步数可降低这些参数。

- **K线间隔**：每多少个 step 聚合为一根K线由时间配置的 `steps_per_bar` 决定[8]。例如，设定 `steps_per_bar=20` 则**每20步形成一根K线**。内置的 HIGHFREQ 预设已经将该值设为20[7]，方便进行高频小周期的实验。如果您需要不同的K线间隔，比如每10步一根K线或每50步一根K线，可以自定义 TimeConfig 修改这个值。K线间隔影响到每轮实验产生的总K线数量，例如总步数4000且20步/Bar会生成≈200根K线。

- **初始资金和交易概率**：对于随机或启发式智能体，可以配置每个智能体的初始现金，以及每步参与交易的概率等参数。在上例中，每个RandomAgent初始现金为100000，trade_probability=0.3 表示每个step 有30% 概率尝试下一笔交易[21]。这些参数可根据实验需要调整，例如降低交易概率以模拟低频交易者，或增加初始资金规模。

- **其他参数**：如每笔订单的默认TTL（生命周期步数）、订单提交率等也可影响实验行为。例如 TTL决定订单挂在订单簿上多久会过期撤单；提交率在自定义模拟中控制每步产生订单的概率。在大规模仿真测试中，提交率被设置为20% 以减少订单总量、缓解压力[22][23]。您可以根据需要调整 `RandomAgent` 或自定义智能体的下单逻辑，使每步提交订单的比例符合实验要求。

综上，通过**配置智能体数量、时间步数/K线周期、智能体属性**等，您可以灵活控制每次实验的规模和节奏，满足多次重复实验、渐进扩展智能体数量（从少量到千级）的需求。

4. 保存实验结果

为了方便分析和在会话结束后保留数据，您应及时保存实验输出的模型、日志和可视化结果：

- **模型检查点**：如果进行了强化学习训练或智能体策略优化，需要定期保存模型参数。在 Colab 中，因会话可能中断，建议将模型checkpoint保存到Google Drive。项目提供了示例检查点管理器，可将智能体策略网络参数、优化器状态等保存到指定目录[24][25]。例如：

```
from agents import rl_agent  # 假设存在RL智能体及优化器
from utils import CheckpointManager

ckpt_mgr = CheckpointManager(save_dir='/content/drive/MyDrive/ExchangeV2/
checkpoints')
ckpt_mgr.save(agent, optimizer, episode=100, metrics={"reward": 123.4})
# ... 在新会话中
last_ep, metrics = ckpt_mgr.load('/content/drive/MyDrive/ExchangeV2/
checkpoints/checkpoint_ep100.pt', agent, optimizer)
```

上述流程将在云盘的 `checkpoints` 文件夹保存模型，方便后续加载继续实验[24][26]。您也可以直接使用 `torch.save()` 将模型 `state_dict` 序列化保存，再用 `torch.load()` 恢复。在训练长时任务时，建议每隔一段时间保存一次模型参数，以防中途意外中断训练进程。

- **日志保存**：对于训练过程中的日志和控制台输出，建议使用 TensorBoard 或将日志重定向到文件保存。例如，您可以启用 TensorBoard 以实时监控训练指标。在代码中配置 TensorBoard 日志目录（如

`runs/`），并确保将其映射到云盘路径，这样即使会话结束日志也不会丢失。项目计划中包含集成 TensorBoard 的选项（如 `use_tensorboard: true`） [27]，您可以据此启动 TensorBoard：

```
%load_ext tensorboard
%tensorboard --logdir /content/ExchangeV2/runs   # 假设TensorBoard日志保存在runs目录
```

这样可以在浏览器中实时查看训练曲线。若不用 TensorBoard，也可以通过Python内置日志将重要信息写入文件，再保存到Drive。例如：

```
log_file = open('/content/drive/MyDrive/ExchangeV2/logs/exp1.txt', 'w')
print("Step,Reward,...", file=log_file)
# 在训练循环中定期写入日志
log_file.flush()
```

确保在实验结束或会话断开前关闭文件并将其保存。

- **数据输出**：ExchangeV2 提供便捷的方法获取仿真生成的数据。例如：
- 使用 `env.get_klines()` 获取所有生成的K线数据列表 [28]。
- 使用 `env.get_trades_df()` 获取所有成交记录的 Pandas DataFrame [29]。
- 使用 `env.export_data(output_dir)` 一键导出全部数据到指定文件夹下的CSV文件 [30]。例如：

  ```
  env.export_data('/content/drive/MyDrive/ExchangeV2/output')
  ```

  将会在输出目录下生成诸如 `trades.csv`、`cancels.csv`、`steps_summary.csv` 等文件 [30]，包含每笔成交、撤单和每步摘要等数据。您可以下载这些CSV到本地进一步分析。

- **可视化结果**：为了直观分析结果，可以利用项目自带的可视化工具或Matplotlib绘图。例如，在获得K线数据后，可绘制价格走势和成交量柱状图。项目附带的综合测试中，有绘制K线图的示例 [31] [32]：通过遍历 `bars` 数据构造蜡烛图（红绿K线）并存为图片。简要步骤如下：

```
import matplotlib.pyplot as plt
bars = env.get_klines()
opens = [bar.open for bar in bars]
closes = [bar.close for bar in bars]
highs = [bar.high for bar in bars]; lows = [bar.low for bar in bars]
volumes = [bar.volume for bar in bars]
indices = range(len(bars))

# 绘制K线实体
colors = ['g' if closes[i] >= opens[i] else 'r' for i in range(len(bars))]
for i in indices:
    plt.bar(i, abs(closes[i]-opens[i]), bottom=min(opens[i], closes[i]),
```

```
color=colors[i])
    plt.plot([i, i], [lows[i], highs[i]], color='k')  # 影线
plt.title('K线图')
plt.xlabel('K线索引'); plt.ylabel('价格')
plt.show()
```

上述代码绘制简单K线图并显示。如需保存图片，可使用 `plt.savefig('kline.png')` 将其保存到云盘。项目的 `comprehensive_test.py` 脚本演示了生成并保存K线图（`kline_chart.png`）和价格趋势图（`price_trend.png`）的过程 [32] [33]。您可以参考其中实现，或直接运行：

```
!python comprehensive_test.py --large-only
```

这将执行项目自带的大规模仿真测试并在 `output_charts` 目录输出示例K线图文件，供您检视仿真结果。

通过以上方式，您可以**保存模型、日志以及各类实验数据**到持久存储，并利用可视化工具深入分析每轮实验的行情走势和交易行为。

5. 多轮实验管理

在 Colab 中反复进行多轮实验时，需要妥善管理环境状态和中间结果：

- **重复实验**：如果希望连续执行多次独立实验（例如不同参数设置或随机种子下重复试验），可以在一次实验结束后**重置环境**或新建环境实例。`Environment` 提供了 `reset()` 方法用于将环境恢复初始状态，以便再次运行仿真 [34]。示例：

```
for run in range(5):
    env.reset()  # 重置环境 [34]
    stats = env.run(verbose=False)
    env.export_data(f"/content/drive/MyDrive/ExchangeV2/output/run{run}")
    print(f"第{run+1}次实验完成，成交笔数: {stats['environment']
['total_trades_executed']}")
```

上述代码片段演示了进行5次独立实验，每次重置后运行并将结果导出到不同目录。请注意，每次重置会保留之前添加的智能体列表，但清空其持仓等状态；如果需要更改智能体数量或类型，可在重置前移除或重新添加智能体，或者直接新建一个新的 `Environment` 实例。

- **检查点续跑**：对于需要跨越多个Colab会话的长期实验（例如训练一个强化学习智能体进行多轮迭代），建议使用检查点机制以便“续跑”。在前述模型保存部分，我们已介绍如何保存模型参数和优化器状态 [24]。在新开启的会话中，先确保代码环境配置与之前一致，然后载入先前保存的检查点：

```
last_episode, metrics = ckpt_mgr.load('checkpoint_ep100.pt', agent,
optimizer)
print(f"已加载第{last_episode}轮的模型，继续训练...")
```

通过记录上次训练的轮数或步数，您可以从断点继续。本项目提供的 `CheckpointManager.load()` 会返回保存时的轮次编号和指标，便于恢复训练计数 [26]。同时也应确保随机数种子等与之前一致以获得可复现结果。

- **状态快照**：对于仿真环境本身（非学习场景），如果需要在中途暂停并恢复，可以利用环境的时间管理器拍摄"快照"。例如：

```
snapshot = env.time_manager.snapshot()
# 运行一部分实验...
env.time_manager.restore(snapshot)
```

以上机制可将当前市场状态保存并还原 [35]。不过在Colab短时会话中通常无需对纯仿真做状态保存，直接重复实验更简单；快照机制更适合调试或需要回溯模拟状态的场景。

通过上述方法，您可以**在多轮实验之间保持连续性**：无论是重复独立实验以求平均效果，还是断点续训一个持续改进的智能体策略，都能确保在Colab环境限制的条件下最大程度地利用先前计算成果，不必每次从头开始。

6. Colab 环境建议

针对在 Colab Pro+ 上运行本项目的一些注意事项和优化建议：

- **会话时长与超时**：Colab Pro+ 单次会话最长可运行约20小时，但仍可能因为空闲或资源原因提前中断。为保证实验顺利完成，**尽量在20小时内结束单次实验**（例如控制训练轮次或步数），或者做好中途保存以便下次会话续接。长时间运行时，可以尝试保持Notebook界面开启并定期有输出，以防被判定为空闲而断开。

- **数据持久化策略**：Colab的临时存储（如 `/content` 目录）会在会话结束后释放，**重要数据一定要保存到持久位置**。推荐使用前述挂载的Google Drive路径来保存模型、日志和结果 [36]。例如，将 `CheckpointManager` 默认保存路径指向 `/content/drive/MyDrive/...`。另外，可考虑在实验完成后将关键结果文件手动备份到云端（Drive或GitHub仓库）。

- **GPU资源分配**：Colab的GPU需合理使用。确保**模型的张量运算在GPU上进行**（PyTorch会自动使用可用GPU [6]），同时避免将不必要的操作放在GPU上以浪费显存。例如，仿真环境的大部分撮合计算在CPU完成即可，只有神经网络前向/后向应使用GPU。利用 `torch.cuda.is_available()` 检查GPU，`torch.cuda.memory_allocated()` 监控显存使用。如遇显存不足错误，可适当降低批量大小或模型规模。

- **内存管理**：尽管Colab Pro+提供约50GB内存，大规模长时间仿真仍需注意内存释放。项目实现了 **内存管理辅助**，例如提供了 `CoLabMemoryManager.clear_memory()` 来清理Python垃圾回收并释放GPU缓存 [37]。您可以在每轮实验后或适当时机调用该方法，释放不用的内存，以防内存占用率持续升高超过阈值 [38]。另外，`Environment` 的 `max_steps_in_memory` 参数默认存储10000步的数据在内存 [39]。如果您计划模拟非常长的时间（>10000步）且内存紧张，可调小该阈值或启用 `external_storage_path` 将超出部分的数据直接写入硬盘 [40]。例如：

```
env = Environment(time_config=..., num_days=10, max_steps_in_memory=5000,
external_storage_path="./data_export")
```

这样每当内存中步数数据超过5000时，旧的数据将自动导出到`./data_export`目录并清理，避免内存无限增长。

- **GPU算力充分利用**：在进行强化学习训练时，充分利用GPU并行加速。例如使用较大的mini-batch、并行环境等策略。但在仿真环境部分，由于单步撮合是序贯过程，提升CPU性能（通过优化算法或使用更高CPU配额）有时比GPU更实际。Colab Pro+通常提供更高的CPU和内存资源，您可以利用这些来支撑上千智能体的并发仿真。

- **性能调优**：根据测试，ExchangeV2 在标准配置下仿真速度约可达每秒2000步 [18]。压测显示1000个智能体×5000步的仿真可在数分钟内完成 [19]，因此Colab Pro+的算力完全能够胜任。不过要避免一些降低性能的行为：**减少不必要的打印**就是关键之一。请将`verbose`设置为False或提高`progress_interval`（如每500或1000步打印一次）来降低IO开销 [13]。同时，确保不要在主循环中执行阻塞操作（如过多的绘图或文件读写）。将可视化和数据导出放到仿真结束后进行会更高效。

- **监控与容错**：在长时间运行时，建议定期监控Colab的资源利用情况（内存、GPU利用率）。如果发现异常增长，可提前采取措施（如保存状态然后重启运行时清理）。另外，可以对关键代码添加try-except以捕获错误，避免一场长跑在最后因未捕获的异常中断。

遵循以上建议，您可以最大程度地发挥 Colab Pro+ 的资源优势，安全、高效地完成ExchangeV2多智能体交易实验。在保证每轮实验顺利进行的同时，将宝贵的数据和成果持久保存，不受会话限制影响。

通过本指南，您将能够在 Google Colab Pro+ 上成功部署和运行`ExchangeV2`项目，进行多智能体的交易市场仿真实验。从环境配置、项目运行到参数调节、结果保存和多轮实验管理，我们提供了逐步说明。请根据自身实验需求调节相应参数，并注意Colab环境的限制和优化技巧。祝您的多智能体交易实验一切顺利！ [18] [19]

---

[1] [2] [3] [4] [5] [6] [24] [25] [26] [27] [36] [37] [38] MARL_IMPLEMENTATION_PLAN.md
https://github.com/xiaoa5/ExchangeV2/blob/e3e958c39a2dd874233ee7a2e3b250b9183a0247/MARL_IMPLEMENTATION_PLAN.md

[7] [8] time_config.py
https://github.com/xiaoa5/ExchangeV2/blob/e3e958c39a2dd874233ee7a2e3b250b9183a0247/project/configs/time_config.py

[9] [11] [15] [16] [17] [18] [21] [29] [35] [40] README.md
https://github.com/xiaoa5/ExchangeV2/blob/e3e958c39a2dd874233ee7a2e3b250b9183a0247/project/README.md

[10] [12] [13] [14] [20] [28] [30] [34] [39] environment.py
https://github.com/xiaoa5/ExchangeV2/blob/e3e958c39a2dd874233ee7a2e3b250b9183a0247/project/environment.py

[19] [22] [23] [31] [32] [33] comprehensive_test.py
https://github.com/xiaoa5/ExchangeV2/blob/e3e958c39a2dd874233ee7a2e3b250b9183a0247/comprehensive_test.py