

# How to Join LightZero

## 一、项目介绍

### 项目资料

#### 背景介绍

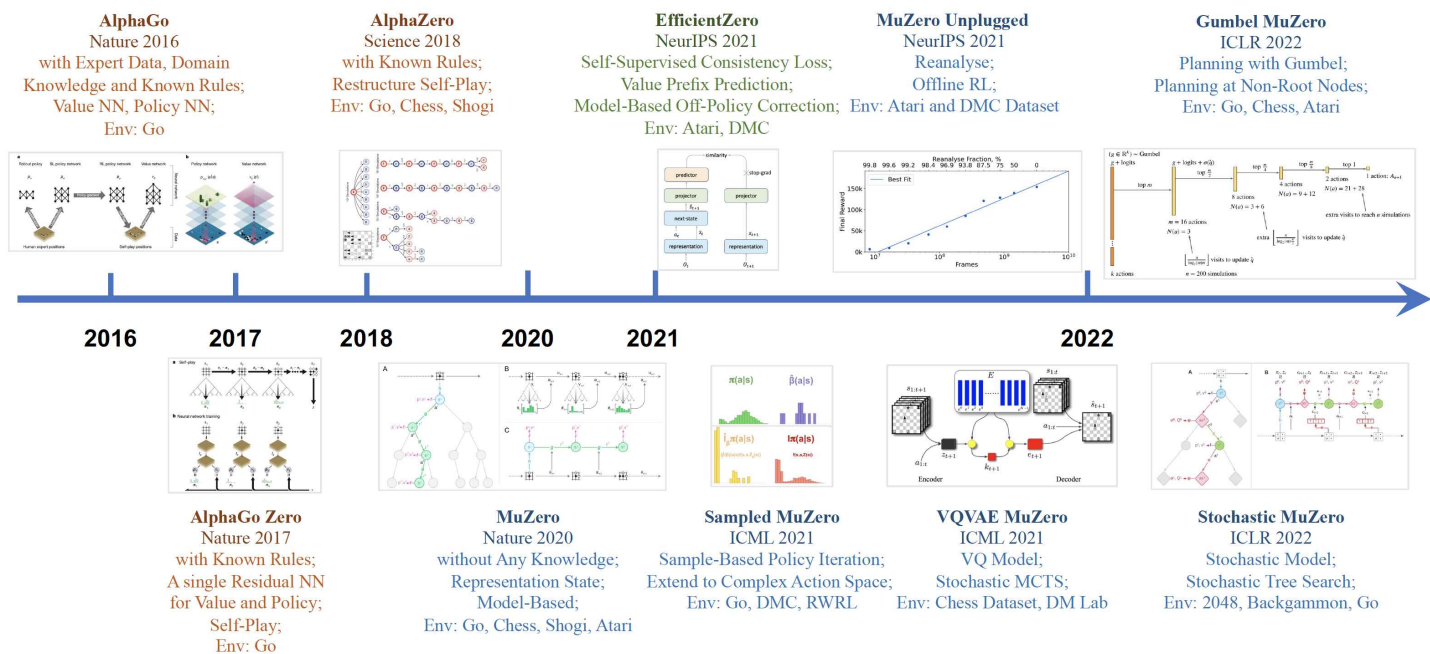
- 强化学习基础概念介绍: [传送门](#)
- Model-Based RL 知识库: [传送门](#)

#### 技术文档

- 论文笔记: [传送门](#)
- 实验结果: [Benchmark](#)
- 代码仓库地址:
  - GitHub:  
<https://github.com/opendilab/LightZero>

### 背景

以 AlphaGo, AlphaZero, MuZero 为代表的结合蒙特卡洛树搜索 (Monte Carlo Tree Search, MCTS) 和深度强化学习 (Deep Reinforcement Learning, DRL) 的方法, 在诸如围棋, Atari 等各种游戏上取得了超人的水平, 也在诸如蛋白质结构预测, 矩阵乘法算法寻找等科学领域取得了可喜的进展。但是目前国内外, 基本没有融合各种相关算法的开源实现, 为此, 本项目 LightZero 以轻量, 高效, 易懂为目标, 融合 MCTS+RL 算法相关的各种环境和算法, 给出基准实验结果和有趣的应用, 期待为入门者提供良好的学习实践平台, 推动 MCTS+RL 在更多领域产生应用与创新。



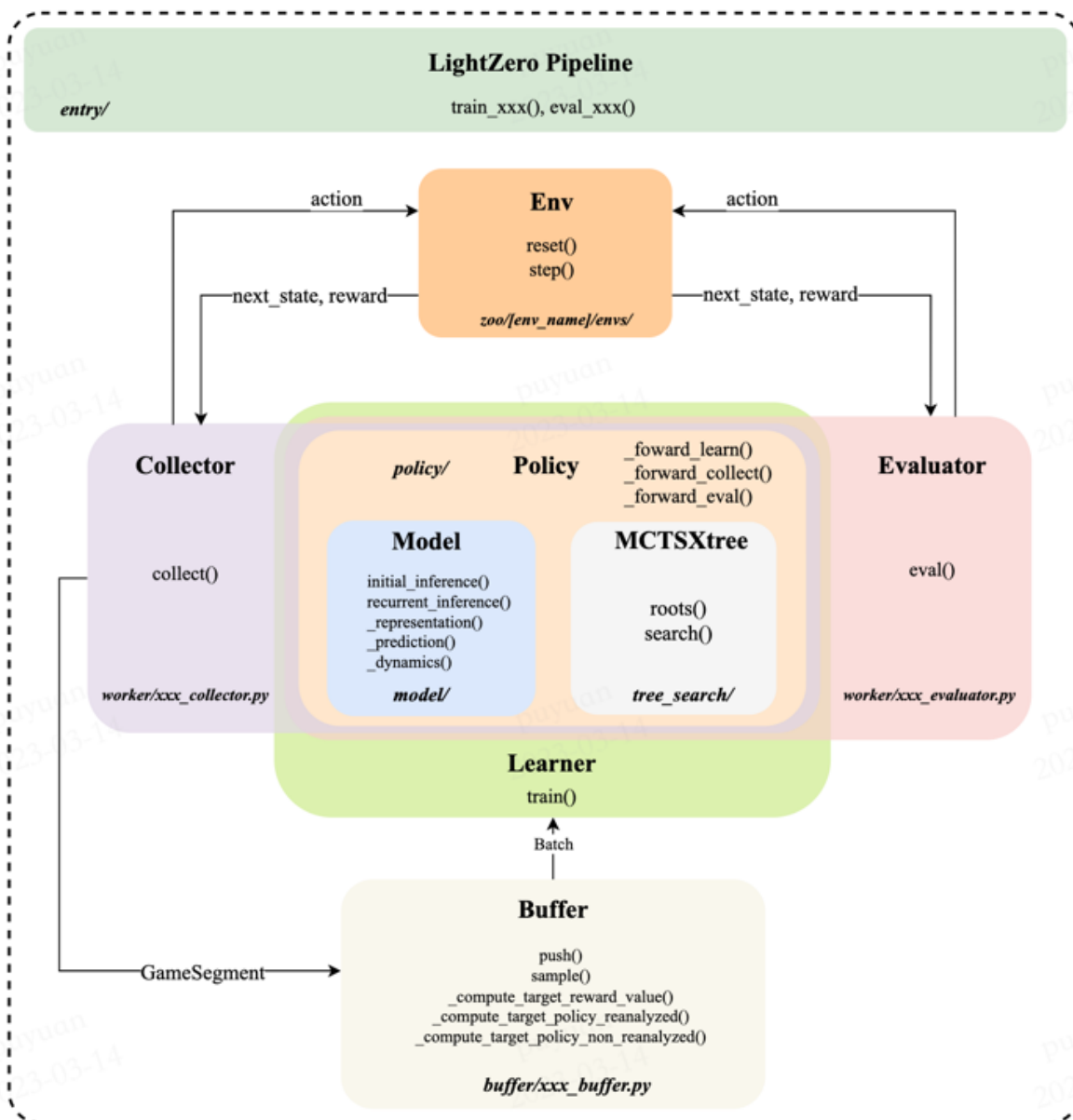
(图1: MCTS+RL 算法研究图。)

## 概览

## 特点

- **轻量**：LightZero 中集成了多种 MCTS 族算法，能够在同一框架下轻量化地解决多种属性的决策问题。
- **高效**：LightZero 针对 MCTS 族算法中耗时最长的环节，采用混合异构计算编程提高计算效率。
- **易懂**：LightZero 为所有集成的算法提供了详细文档和算法框架图，帮助用户理解算法内核，在同一范式下比较算法之间的异同。同时，LightZero 也为算法的代码实现提供了函数调用图和网络结构图，便于用户定位关键代码。

## 框架结构



(图2: LightZero 框架流程图。)

上图展示了 LightZero 的框架流程图。接下来，将简要介绍其中的三个核心模块：

- **Policy:** `Policy` 描述了网络更新方式以及与环境交互的方式，它包括三个过程：训练过程 (learn)、采样过程 (collect) 和评估过程 (evaluate)。
- **Model:** `Model` 负责定义网络结构，包含 `__init__` 函数用于初始化网络结构，以及 `forward` 函数用于计算网络的前向传播。
- **MCTS:** `MCTS` 则定义了蒙特卡洛搜索树的结构以及与 `Policy` 的交互方式。`MCTS` 的实现有 python 和 cpp 两个版本，分别在 `ptree` 和 `ctree` 中实现。

LightZero 仓库的设计理念注重用户友好性，旨在让开发者能够轻松地将新环境集成到项目中，并便捷地使用各类算法。具体示例如下所示：

```
1 # 下面的 main_config, create_config 分别对应算法的主要参数设置和算法的各模块创建的设置，具体可以参见 cartpole_muzero_config
```

```

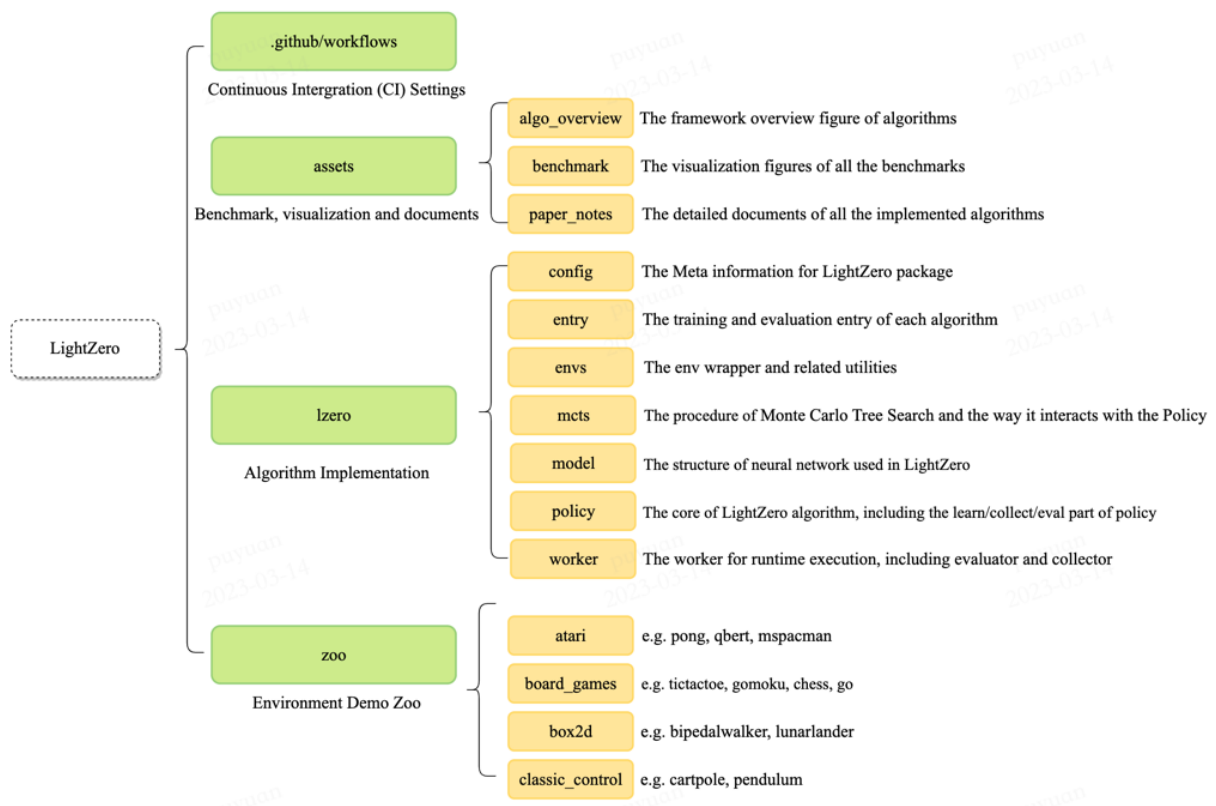
3 # 对于 LightZero zoo 里面已经集成的环境，通过下面的入口函数，便可以调用 LightZero 集成的算法
4 from lzero.entry import train_muzero
5 train_muzero([main_config, create_config], seed=0, max_env_step=int(1e5))
6
7 # 对于用户自定义的以标准 Gym 接口封装好的环境，通过下面的入口函数，便可以调用 LightZero 集成的算法
8 from lzero.entry import train_muzero_with_gym_env
9 train_muzero_with_gym_env([main_config, create_config], seed=0,
    max_env_step=int(1e5))

```

MuZero 相关的核心代码链接：

- [policy](#)
- [model](#)
- [tree\\_search](#)
- [ctree\\_muzero](#)
- [entry](#)
- [buffer](#)

关于 LightZero 的文件结构，请参考图3 [lightzero\\_file\\_structure](#)。



(图3: LightZero 文件结构图。)

## 集成算法

LightZero 集成了具有连续与离散动作、向量与图像输入、是否完全信息博弈、是否包含环境随机性等不同属性的任务。在统一的框架下，LightZero 支持多种 MCTS 系列算法，可轻松解决各类属性的决策问题，具备优秀的扩展性，支持快速上手自定义环境，更易适应复杂决策问题中的各类需求。

LightZero 主要基于深度学习框架 PyTorch [15] 和强化学习平台 DI-engine [16] 实现。

目前 LightZero 中集成的算法包括：

- AlphaZero
- MuZero
- EfficientZero
- Sampled MuZero

目前，LightZero 支持的环境及算法如下表所示，其中"✓"表示相应项目已完成并经过充分测试。"🔒"表示相应项目正在等待列表中（进行中）。"---"表示该算法不支持此环境。

	A	B	C	D	E
1	Env./Alg.	AlphaZero	MuZero	EfficientZero	Sampled EfficientZero
2	Atari	---	✓	✓	✓
3	tictactoe	✓	✓	🔒	🔒
4	gomoku	✓	✓	🔒	🔒
5	lunarlander	---	✓	✓	✓
6	bipedalwalker	---	✓	✓	✓
7	cartpole	---	✓	✓	✓
8	pendulum	---	✓	✓	✓

（表1: LightZero 支持的环境及算法。）

## 二、项目目标

LightZero 是一个结合了蒙特卡洛树搜索 (MCTS) 和强化学习 (RL) 的开源算法工具包。它支持一系列基于 MCTS 的 RL 算法，希望具有以下优点：轻量，高效，易懂。详情请参考[特点](#)、[框架结构](#)和[集成算法](#)。

LightZero 的目标是标准化 MCTS 算法族，以加速相关研究和应用。[Benchmark](#) 中介绍了目前所有已实现算法的性能比较。

为此，项目共分为下面4部分目标：

Updated on 2023.05.25



集成棋类与非棋类游戏



复现 MCTS 算法族



各个算法不同环境上的基准



在棋类或动作

## 集成棋类游戏

- Tictactoe 3\*3
- Gomoku (6\*6, 15\*15)
- Chess
- Go

## 集成非棋类游戏

- Atari
- Classic Control
- Box2d
- MuJoCo
- MiniGrid
- DeepMindControl (DMC)

## AlphaZero

- MuZero
- EfficientZero
- Sampled MuZero
- Gumbel MuZero
- Stochastic MuZero
- MuZero Unplugged
- ROSMO

## 对比结果

### 棋类

- tictactoe
- gomoku 6\*6 15\*15
- Go

### Atari:

- Pong
- Qbert
- Mspacman
- Breakout

### Classic Control

### box2d

- lunarlander
- bipedalwalker

### MuJoCo

- Hopper
- Halfcheetah
- Walker2d
- Humanoid

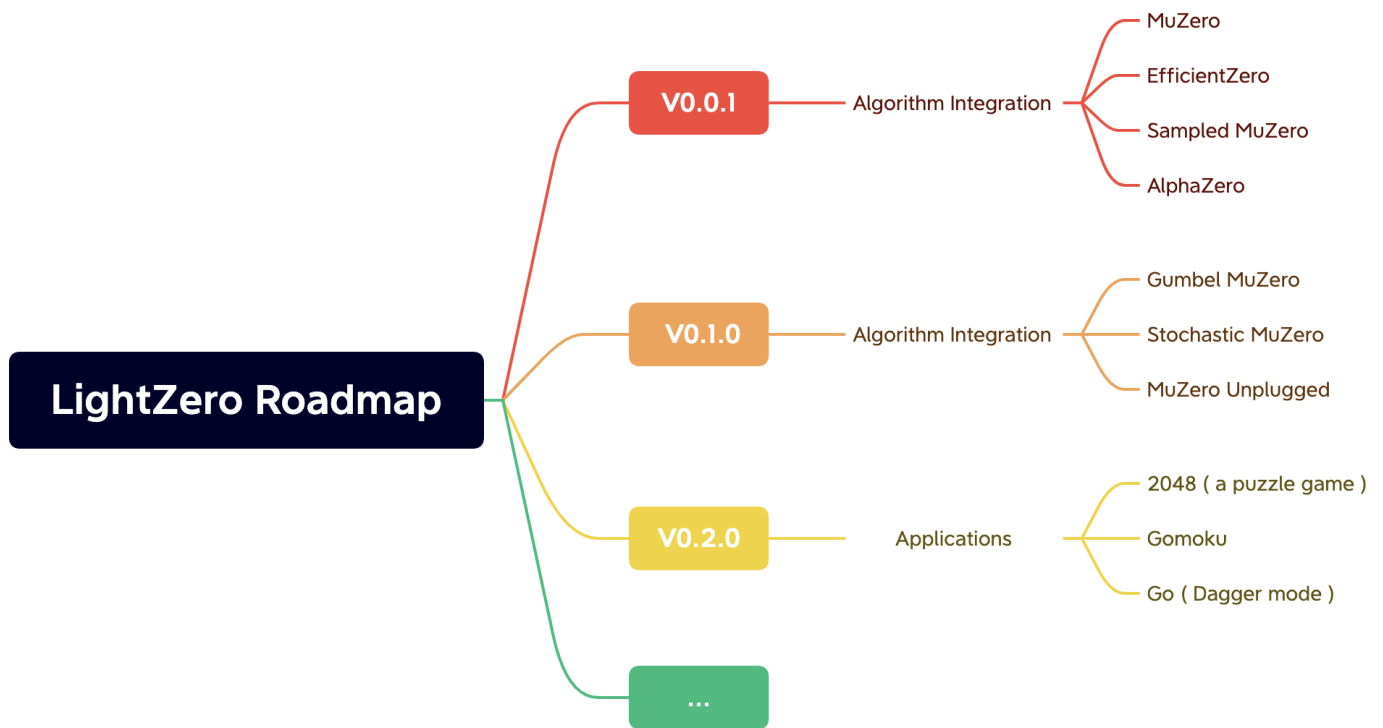
### DMC (state -> pixel)

- cartpole balance
- hopper hop
- humanoid stand
- manipulator bring\_ball

## 类游戏上的有趣应用

- 围棋
- 围棋（飞刀）

## 三、Roadmap



## 四、warm up 任务拆解

- 通用流程
  - 先以跑通代码为契机熟悉仓库整体流程
  - 再加新环境增强对仓库细节的理解，写算法或环境文档，增强对具体任务的理解
  - 最后，向仓库中添加新算法或新特性，达到融汇贯通。
- 针对具体任务的具体处理。

### 4.1 通用流程

#### 熟悉 LightZero 框架

- 跑通已有 MuZero atari 实验：运行 [atari\\_muzero\\_config.py](#)，跑 PongNoFrameskip-v4 muzero 的实验，熟悉整体流程，并记录详细的 tb，理清 tb 中各项指标的具体含义。
- 参考 (图2: LightZero 框架流程图) 和 (图3: LightZero 文件结构图) 熟悉框架总流程和文件结构。
- 阅读整理的 MCTS+RL 系列算法的[论文笔记](#)，与任务相关的算法保证详细阅读。
- 任务遵循准则加样例的准则，拆分成小的子任务
- 通过阅读之前的基准测试和探究实验，明晰算法各个关键参数的作用，补充更多探究实验。
- 通过补充单元测试等小的任务，熟悉关键模块，单元测试参考[单元测试指南](#)。

- 例如增加对 efficientzero 算法3项改进各自的单元测试。

## 开发某个算法或特性

### 开发前

- 💡
  - （如果有开源实现）调研已有开源实现的代码库，理解它的核心部分实现逻辑，跑通代码，记录训练曲线。
- 与项目开发者讨论确定实验计划
  - 选择什么测试环境，用什么镜像，可以在仓库目前的哪些代码上开始开发，效率最高。
  - 停止程序的 max\_env\_steps, stop\_value 应该设置为多少合适。eval reward 达到什么程度算作复现的算法收敛了。

### 开发中

- 📌
  - 写了一个功能后，参考[单元测试指南](#)，进行单元测试，测试成功后，再添加下一个功能，有利于排除错误。
- 遇到报错如何解决：
  - 尽量用英文谷歌搜索更有可能找到你想要的答案，如果半小时内查询不到或解决不了，在群里及时反馈。
- 调试参考资料
  - 一、处理程序运行时偶发出现 NaN 的参考调试流程
  - 二、在程序运行无错误但性能未达到预期时的分析思路

### 开发后

- 🏆
  - 论文中有哪些探究实验也值得尝试。
- 总结沉淀
  - 撰写实现算法的文档，与其他类似算法的分析比较
  - 对复现过程中出现问题及其解决方案的总结，以及有哪些值得研究的问题

## 4.2 针对具体任务的具体处理





例如，对于 MuZero Unplugged，有下面的特定任务：

1. 仿照 [atari\\_env](#) 写 d4rl 对应的 lightzero\_env 和对应的 tests
2. 根据建议完善相应的算法文稿
3. 给出将 MuZero Unplugged 迁移到 LightZero 的计划，并按计划开始实现
  - 参考 <https://github.com/DHDev0/Muzero-unplugged>

## 五、项目圆桌讨论会



### 会议指南：

- **开会前：** 提倡“先阅读后讨论”，请需要与他人讨论/存在风险阻碍的开发者准备好会议文档，明确会议主题，让所有人了解参会预期，所有参会人提前文档，并针对有疑问的部分评论，提出希望进一步了解或讨论的内容。
- **开会中：**
  - 全员完成阅读后，大家针对开发文档内的评论和建议逐条探讨，当场解决问题或明确后续安排。
- **开会后：**
  - 会议记录人在本文档添加待办事项，明确下一步行动和预期时间
  - 相关负责人完成任务后，点击标记任务列表，实时更新任务完成情况

## 附录

### 一、处理程序运行时偶发出现 NaN 的参考调试流程

在 PyTorch 程序运行过程中，有时可能会遇到 NaN（Not a Number，非数值）现象。面对这种情况，可以参考以下调试流程进行问题排查：

#### 1. 排查可能导致 NaN 的典型情况

- a. **检查输入数据：** 确保输入数据经过正确的归一化处理，且不包含 NaN 或 Inf 等异常值。
- b. **检查数值不稳定性：** 如果模型涉及指数或对数等函数，请务必检查数值不稳定性并采用数值稳定性技巧，如对数求和技巧或带温度的 softmax。例如针对  $\log(0)$ 、除以 0 和  $\text{arctanh}(0)$  等情况，需要添加一个较小的值（如  $1e-6$ ）以满足数值稳定性要求。
- c. **应用梯度裁剪：** 梯度裁剪可以防止梯度爆炸现象，从而避免 NaN 值的产生。


- d. **插入 NaN 检查点：**在训练循环的关键位置添加 NaN 检查，有助于隔离问题的根源。可以使用 `torch.isnan()` 函数进行检查。
- e. **审查模型架构：**检查模型架构是否正确实现，排除可能存在的设计缺陷或实现错误。
- f. **检查优化器配置：**确认优化器配置正确，学习率设置得当，避免过大或过小的学习率。
- g. **调整批量大小：**如果模型在较小的批量大小下能够正常运行，可以尝试减小批量大小，以便隔离问题所在的样本。
- h. **利用 PyTorch 的自动微分性能分析器：**自动微分性能分析器能够追踪计算过程并确定生成 NaN 值的具体操作。

## 2. 探究现场

在深度学习训练过程中，可能会遇到一些运行错误，如在运行若干 epoch 之后突然出现 NaN 等问题。这种情况可能是由于代码中缺乏对特殊值或离群点的处理导致的。为了解决这类问题，我们需要在代码中加入一些额外的处理步骤以便于复现报错现场。

推荐的解决方案是，在报错时（结合算法逻辑和(或)try except语句）将数据和模型存储下来，以便复现报错现场。本示例将以 LightZero 系列中的 Sampled EfficientZero Policy 为例，展示如何处理 NaN 报错问题。



 template\_for\_debugging\_nan.py

## 3. 分析错误原因并针对性地进行解决

在对运行错误进行调试时，我们需要分析错误原因并针对性地进行解决。以前述示例为例，通过加载存储好的报错现场，我们发现在执行以下语句段后出现了 NaN：

```
1 target_sampled_actions_before_tanh = torch.arctanh(target_sampled_actions_not_cl
```

通过查阅 `torch.arctanh` 函数的文档，我们发现当输入值超过  $(-1, 1)$  范围时，函数返回值将为 NaN。因此，这里出现 NaN 的原因是由于数值不稳定性导致的。

**解决方法：**为避免数值不稳定，我们可以在执行 `torch.arctanh` 函数之前先将输入值限制在  $[-1 + 1e-6, 1 - 1e-6]$  范围内。修改后的代码如下：

```
1 # 注意：为确保数值稳定性
2 target_sampled_actions_clamped = torch.clamp(
3     target_sampled_actions[:, k, :],
4     torch.tensor(-1 + 1e-6),
5     torch.tensor(1 - 1e-6)
6 )
7
8 target_sampled_actions_before_tanh = torch.arctanh(target_sampled_actions_clampe
```

通过这种针对性的解决方法，我们可以防止数值不稳定性导致的 NaN 问题，从而提高代码的稳定性和可靠性。在实际编程过程中，我们需要针对各种可能出现的问题，分析其原因并进行相应的解决。

## 二、在程序运行无错误但性能未达到预期时的分析思路

当程序运行未出现错误，但性能没有达到期望值时，可以参考以下分析思路：

### 1. 分析代码修改

使用 vimdiff、PyCharm Compare with Clipboard 等工具，检查代码修改是否符合预期。例如，全局替换时可能错误地将变量 A 替换成了 B，虽然程序能正常运行，但算法逻辑错误可能导致模型无法收敛。

- 实例：在调试 Pong Sampled EfficientZero 算法时，收集到的奖励一直在 -20 左右，最后发现是全局替换时错误地将 value 替换成了 value\_prefix。

### 2. 分析输出结果

完整记录 collect、eval、learn 的 TensorBoard 输出，并分析 TensorBoard 曲线是否存在异常，

- 例如某一项损失值未发生变化，可能是该损失项未进行梯度更新。
- 如果某项损失值明显偏离正常值，需要检查损失项输入是否正确，是否需要经过 softmax 操作。
- 实例：

- 在调试 Pong EfficientZero 算法时，收集到的奖励一直在 -20 左右，最后发现少了一行目标模型更新的代码。

在实际应用中，我们需要时刻关注算法的表现，并根据损失值的变化及时调整模型，以确保模型能够达到预期的效果。

### 3. 分析输入数据

分析输入数据是否符合预期，例如检查数据是否错位，或是否进行了错误的数值类型转换。

- 实例：
  - 在调试 Pong EfficientZero 算法时，收集到的奖励一直在 -20 左右，最后发现错误地将输入数据转换为了 uint8 类型。由于原始输入为 [0,1] 之间的浮点数，转换为 uint8 后全部变为 0。
  - 在调试 R2D2 算法时，发现不收敛的原因是 done 数据与预期相差一位，导致 TD 目标计算错误。

### 4. 最小单元分析

若在代码 A 的基础上增加模块 m 得到代码 B，且代码 A 收敛但代码 B 不收敛，则可以将模块 m 拆分为最小单元，逐个加入代码 A，分析在哪一步算法开始不收敛，从而重点检查该最小单元的实现是否有误。

- 实例：
  - 在重构代码后，MuZero+TicTacToe 算法一直不收敛，但重构前是收敛的。首先进行重构后 MuZero+Pong 实验发现收敛，说明 MuZero 算法部分可能无误，问题出在 TicTacToe 环境部分。仔细对比重构前后的 TicTacToe 环境，发现 to\_play 设置不一致，进一步分析发现正是这一设置导致算法在某一 if 语句中进入了错误的代码块。

### 5. 分析算法各个节点的变量

在算法各个节点插入断点，监控重要变量是否符合预期。

- 实例：
  - 在调试 Gumbel MuZero 的 collect 过程中，发现 collect 样本的平均奖励很低。通过在 collect 的各个节点插入断点（如 MCTS simulation 中、MCTS 返回 action probs 到 policy、policy 选择 action 等），监控 action probs 和 action 值，发现 MCTS 返回的 action probs 是正确的。但 policy 错误地输出了 visit count 最大的 action，而非 action probs 最高的 action，导致无法 collect 到高奖励样本。

### 6. 针对 MCTS+RL 算法的一般分析思路

以下内容由 GPT4 产生的回答修改而来。

当使用蒙特卡洛树搜索（MCTS）和强化学习（RL）的组合算法时，如果程序运行无错误，但性能未达到预期，可以尝试以下分析思路：

- **检查 MCTS 实现：**审查 MCTS 算法的实现，确保选择、扩展、模拟和回溯阶段均正确实现。检查 UCT 公式和搜索参数（如探索常数）是否设置合理。
- **检查 RL 实现：**检查强化学习算法（如 Q-Learning、DQN 或 PPO）的实现，确保损失函数、优化器、激活函数等设置正确。同时，检查经验回放缓冲区和学习的频率是否恰当。
- **调整 MCTS 和 RL 参数：**尝试调整 MCTS 和 RL 的超参数，例如搜索次数、探索常数、学习率、折扣因子等。这些参数对算法性能有很大影响，可能需要多次尝试以找到最佳组合。
- **评估神经网络架构：**检查使用的神经网络架构是否适当，如层数、神经元数量和激活函数等。过于复杂或过于简单的网络结构都可能影响性能。
- **验证数据和环境：**检查强化学习环境是否正确设置，如状态表示、奖励函数、动作空间等。确认训练数据质量和格式符合预期。
- **监控训练过程：**观察训练过程中的损失值、奖励值和其他指标的变化，以便了解算法是否收敛。同时，可以使用单步调试和日志记录来检查 MCTS 和 RL 算法的内部状态。
- **使用可视化工具：**利用可视化工具（如 TensorBoard）来展示训练过程中的各种指标。这有助于发现问题并优化算法。
- **对比基准和参考实现：**查找与当前问题相关的基准结果或参考实现，以便了解预期性能。将自己的实现与基准或参考实现进行对比，以找出可能的性能差异来源。
- **寻求外部帮助：**在遇到难以解决的问题时，可以参考相关论文、博客文章，或在论坛、社区中提问以寻求帮助。与同行交流和分享经验，有助于发现问题并找到解决方案。

通过以上分析思路，我们可以逐步找到影响 MCTS+RL 算法性能的因素，并进行相应的优化。这个过程可能需要时间和耐心，但在不断尝试和改进的过程中，我们将不断提高自己的技能和经验。