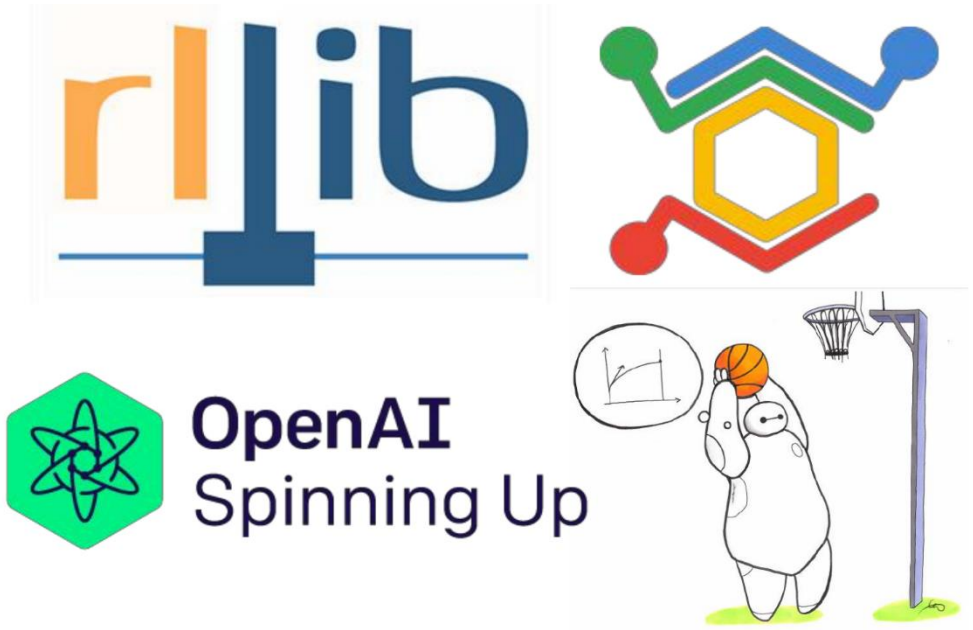
翁家翌

2020.10.11

天授：https://github.com/thu-ml/tianshou

Motivation

- 现有使用较为广泛的深度强化学习平台包括OpenAI的Baselines、SpinningUp，Berkeley的开源分布式强化学习框架RLlib、rlpyt，Google的Dopamine，以及其他独立开发平台Stable-Baselines、keras-rl等。



Motivation

- 这些强化学习算法平台大部分支持至少4种model-free强化学习算法，支持对训练环境进行自定义配置，但是缺陷也不少：
  - 算法模块化不足；
  - 实现算法种类有限；
  - 代码实现复杂度过高；
  - 部分平台性能不佳；
  - 缺少完整单元测试；
  - 环境定制支持不足。

算法特性

- 模块化设计，基于PyTorch，总共代码约2500行，方便搭积木式的构建+二次开发
- 目前支持如下的Model-free算法：
  - DQN及其变种（Double DQN、Dueling DQN、Prioritized Experience Replay (PER)）
  - Policy Gradient、A2C、PPO with GAE
  - DDPG、TD3、SAC、Discrete SAC
  - 上述所有算法均支持RNN
  - 上述所有基于Q-Learning的算法均支持N-step Bootstrapping和PER
- 以及一个Model-based算法：Posterior Sampling Reinforcement Learning (PSRL)
- 以及最基本的Imitation Learning (Behavior Cloning && Dagger)
- 以及最基本的Multi-Agent Reinforcement Learning

- 还有其他的算法特性（比如numba加速等等），这里地方太小，写不下

社区支持

- 代码开源于GitHub，https://github.com/thu-ml/tianshou，目前已有2.1K star和300+ forks（欢迎三连）
- 支持pip和conda的一键安装：
  - pip install tianshou
  - conda –c conda-forge install tianshou
- 有较为完善的英文文档和中文文档，每个算法平均配有两个example
- 拥有较为完善的单元测试：
  - 基本功能测试
  - 训练流程测试
  - 代码风格测试（PEP8）
  - 静态类型测试（Type Check）
  - 文档测试
- 4个core contributor，每个Pull Request都必须经过至少两个人的code review与approve
- GitHub Issue平均第一次回复时间少于一小时

环境支持

- 需要遵守gym的接口（step/reset/seed/…）
- 可以支持任意自定义的环境，例如state/action是一个嵌套很多层的字典，甚至支持传一个class当作state（比如networkx），文档里面给了很多关于内部数据结构Batch的例子和用法

```
{
    'done': done,
    'reward': reward,
    'state': {
        'camera': camera,
        'sensory': sensory
    }
    'action': {
        'direct': direct,
        'point_3d': point_3d,
        'force': force,
    }
}
```

```
>>> import networkx as nx
>>> b = ReplayBuffer(size=3)
>>> b.add(obs=nx.Graph(), act=0, rew=0, done=0)
>>> print(b)
ReplayBuffer(
    act: array([0, 0, 0]),
    done: array([0, 0, 0]),
    info: Batch(),
    obs: array([<networkx.classes.graph.Graph object at 0x7f5c607826a0>, None,
                None], dtype=object),
    policy: Batch(),
    rew: array([0, 0, 0]),
)
```

天授：https://github.com/thu-ml/tianshou

环境支持

- 需要遵守gym的接口（step/reset/seed/…）
- 可以支持任意自定义的环境，例如state/action是一个嵌套很多层的字典，甚至支持传一个class当作state（比如networkx），文档里面给了很多关于内部数据结构Batch的例子和用法

```
{
    'done': done,
    'reward': reward,
    'state': {
        'camera': camera,
        'sensory': sensory
    }
    'action': {
        'direct': direct,
        'point_3d': point_3d,
        'force': force,
    }
}
```

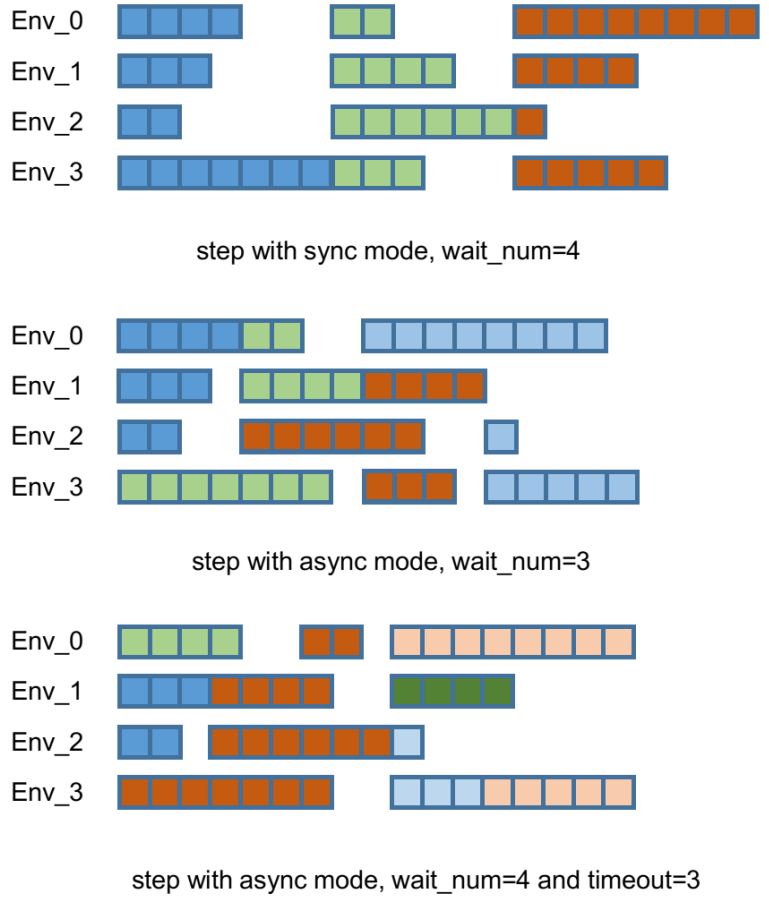
```
>>> import networkx as nx
>>> b = ReplayBuffer(size=3)
>>> b.add(obs=nx.Graph(), act=0, rew=0, done=0)
>>> print(b)
ReplayBuffer(
    act: array([0, 0, 0]),
    done: array([0, 0, 0]),
    info: Batch(),
    obs: array([<networkx.classes.graph.Graph object at 0x7f5c607826a0>, None,
                None], dtype=object),
    policy: Batch(),
    rew: array([0, 0, 0]),
)
```

天授：https://github.com/thu-ml/tianshou

环境支持（续）

- 上述所有算法**均**支持多个环境的并行模拟与数据采集
- 对于运行较慢或者运行效率波动较大的自定义环境，天授提供了异步方式的环境模拟（所有算法也都支持！），只需要额外添加1~2个参数（wait_num或timeout）

- 如果环境的效率实在太低的话，还能把数据存下来（使用pickle.save(buffer)和pickle.load(buffer)）下次接着用

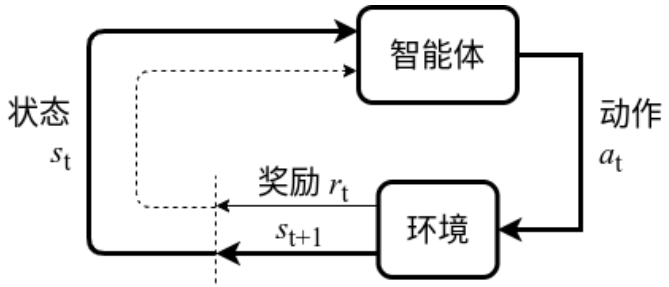


以及速度还挺快的

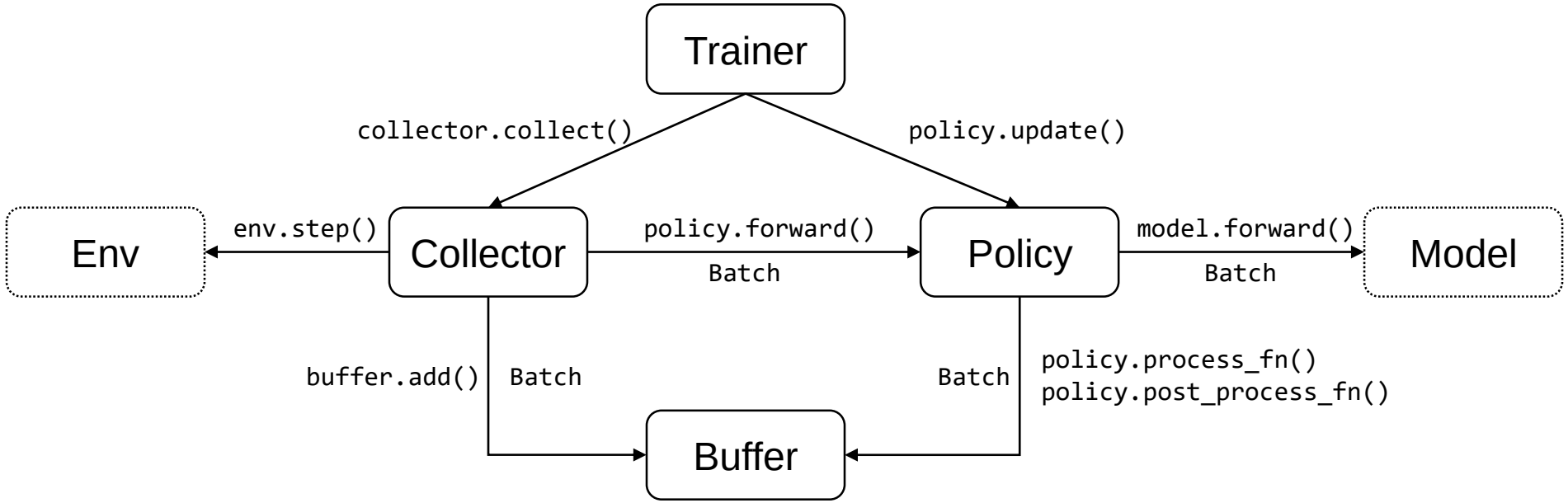
- 使用多个env并行收集数据加速
- 使用Numba加速
- 平台开销小，正常跑Atari环境的速度能达到3000 FPS（实际上是12000，因为通常都是连续4帧拼在一起）
- 其他各种细节优化（地方太小写不下）

- Cartpole-v0（离散动作空间任务）平均约10s
- Pendulum-v0（连续动作空间任务）平均约30s
- Atari Pong使用DQN单机训练半小时就能结束（+20 reward）
- 详见readme

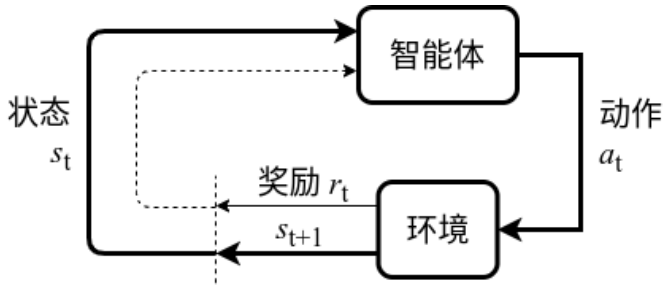
基本概念与模块化设计



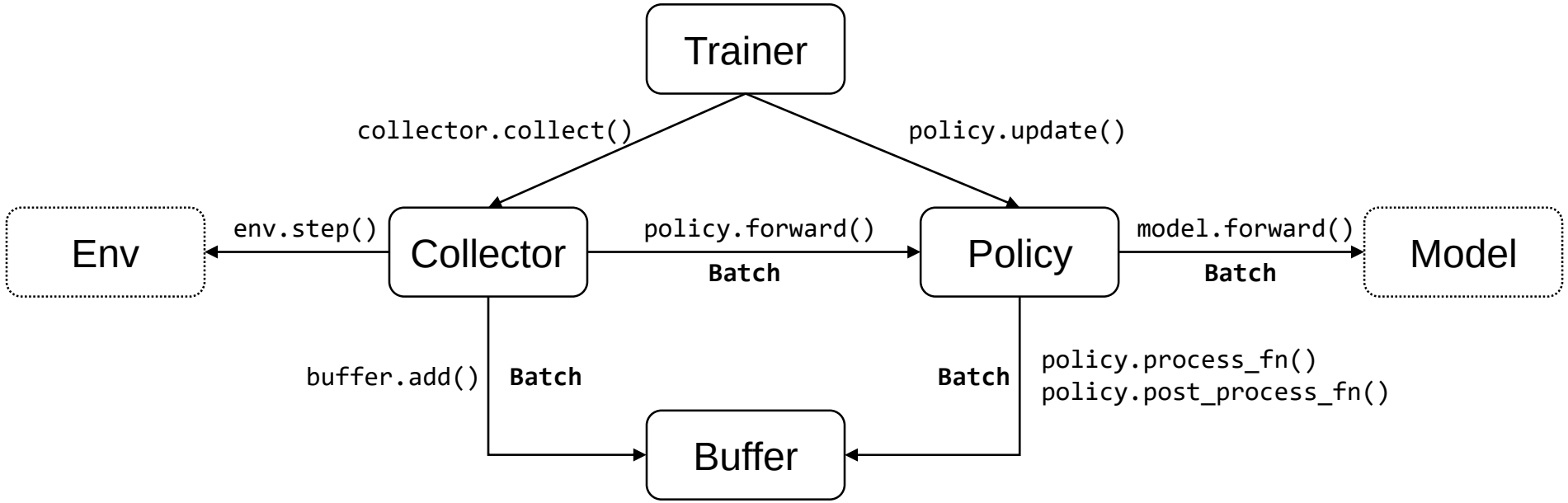
State for policy		policy.training	policy.updating
Training State	Collecting State	True	False
	Updating State	True	True
Testing State		False	False



基本概念与模块化设计



State for policy		policy.training	policy.updating
Training State	Collecting State	True	False
	Updating State	True	True
Testing State		False	False



内部数据结构：Batch

- 天授提供了 Batch 作为内部模块传递数据所使用的数据结构，它既像字典又像数组，可以以这两种方式组织数据和访问数据

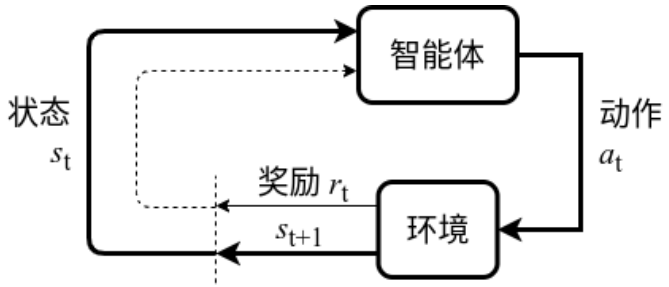
```
>>> from tianshou.data import Batch
>>> data = Batch(a=4, b=[5, 5], c='2312312', d=('a', -2, -3))
>>> # the list will automatically be converted to numpy array
>>> data.b
array([5, 5])
>>> data.b = np.array([3, 4, 5])
>>> print(data)
Batch(
    a: 4,
    b: array([3, 4, 5]),
    c: '2312312',
    d: array(['a', '-2', '-3'], dtype=object),
)
```

内部数据结构：Batch

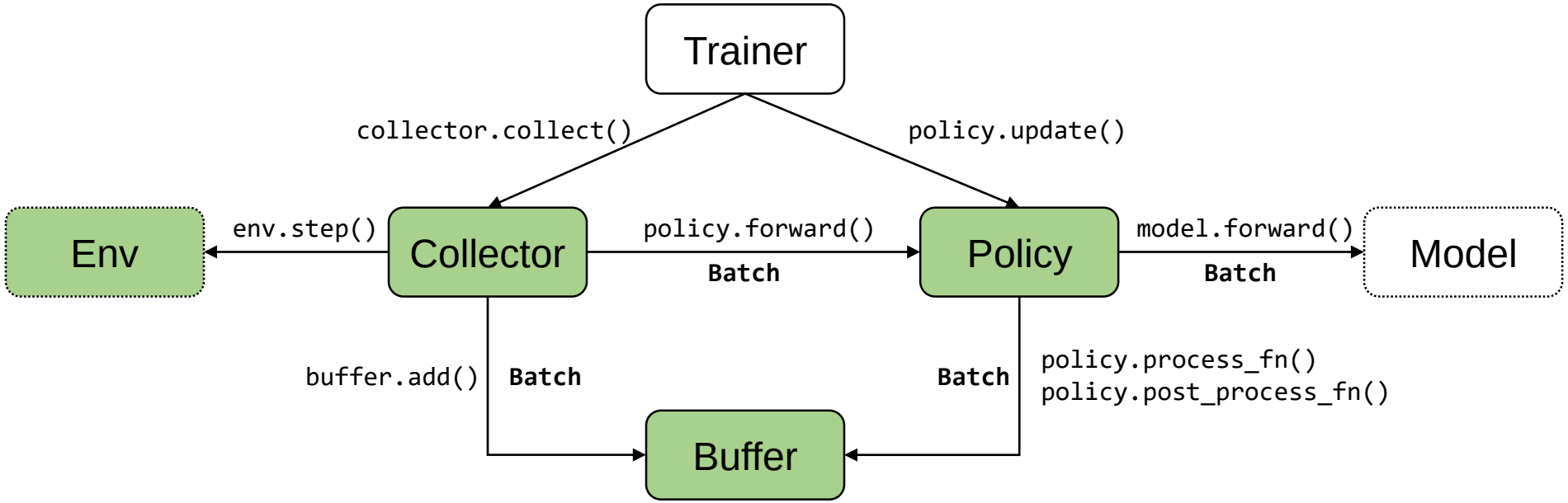
- 天授提供了 Batch 作为内部模块传递数据所使用的数据结构，它既像字典又像数组，可以以这两种方式组织数据和访问数据
- 甚至能批量处理数据！

```
>>> data = Batch(obs={'index': np.zeros((2, 3))}, act=torch.zeros((2, 2)))
>>> data[:, 1] += 6
>>> print(data[-1])
Batch(
    obs: Batch(
             index: array([0., 6., 0.]),
         ),
    act: tensor([0., 6.]),
)
```

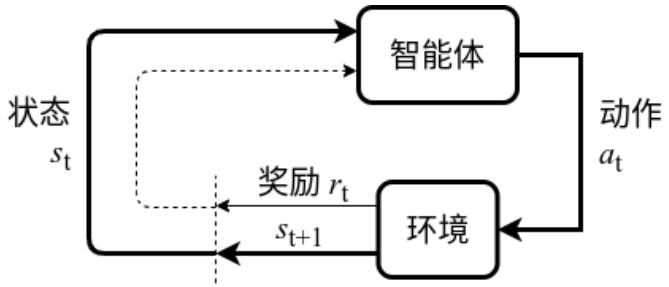
基本概念与模块化设计



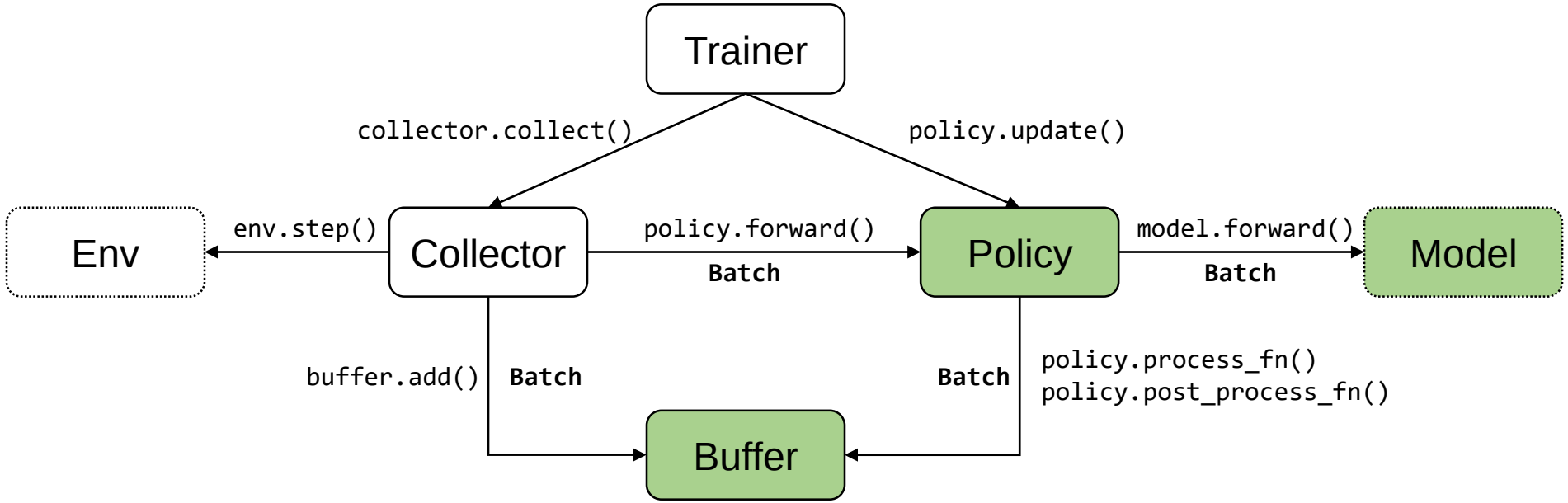
State for policy		policy.training	policy.updating
Training State	Collecting State	True	False
	Updating State	True	True
Testing State		False	False



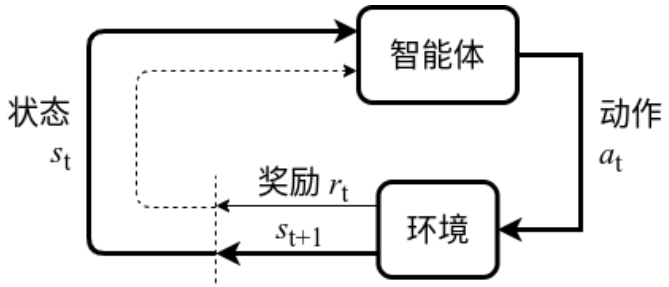
基本概念与模块化设计



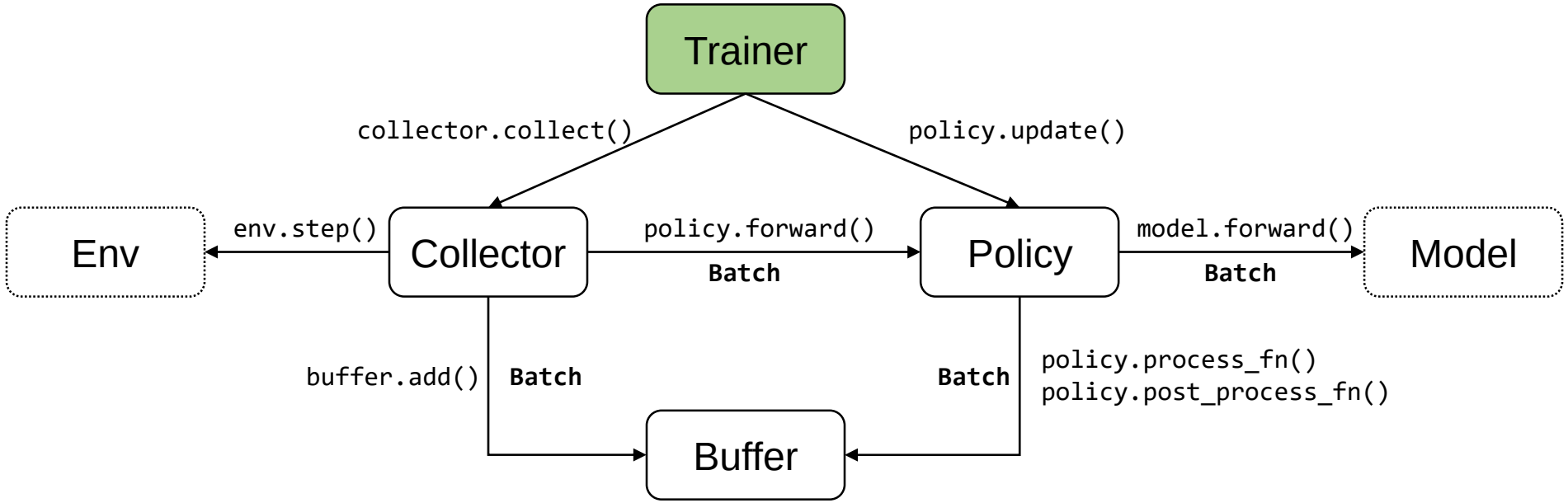
State for policy		policy.training	policy.updating
Training State	Collecting State	True	False
	Updating State	True	True
Testing State		False	False



基本概念与模块化设计



State for policy		policy.training	policy.updating
Training State	Collecting State	True	False
	Updating State	True	True
Testing State		False	False



基本概念与模块化设计（续）

- policy是核心模块，包含了强化学习算法的核心实现
  - __init__：策略初始化，比如初始化自定义模型和target network等；
  - forward：给定观测值o_t计算动作值a_t；
  - process_fn：在获取训练数据之前和Buffer进行交互，比如使用GAE或者N-step估计优势函数；
  - learn：使用一个数据组Batch更新策略；
  - post_process_fn：在更新策略后和Buffer进行交互，比如PER需要更新优先权重；
  - update：相当于连续调用process_fn->learn->post_process_fn。
- 例子

后续工作

- 提供更多的Benchmark Result（正在进行）
- 提供更多的model-free算法：Rainbow、Ape-X、IMPALA、TRPO、……
- 提供更多的Imitation Learning算法：Inverse RL、GAIL、……
- 提供更多的Exploration与Estimation：Curiosity、HER、V-trace、……
- 完善Multi-agent的功能、提供更多的Model-based算法（MCTS、AlphaZero）