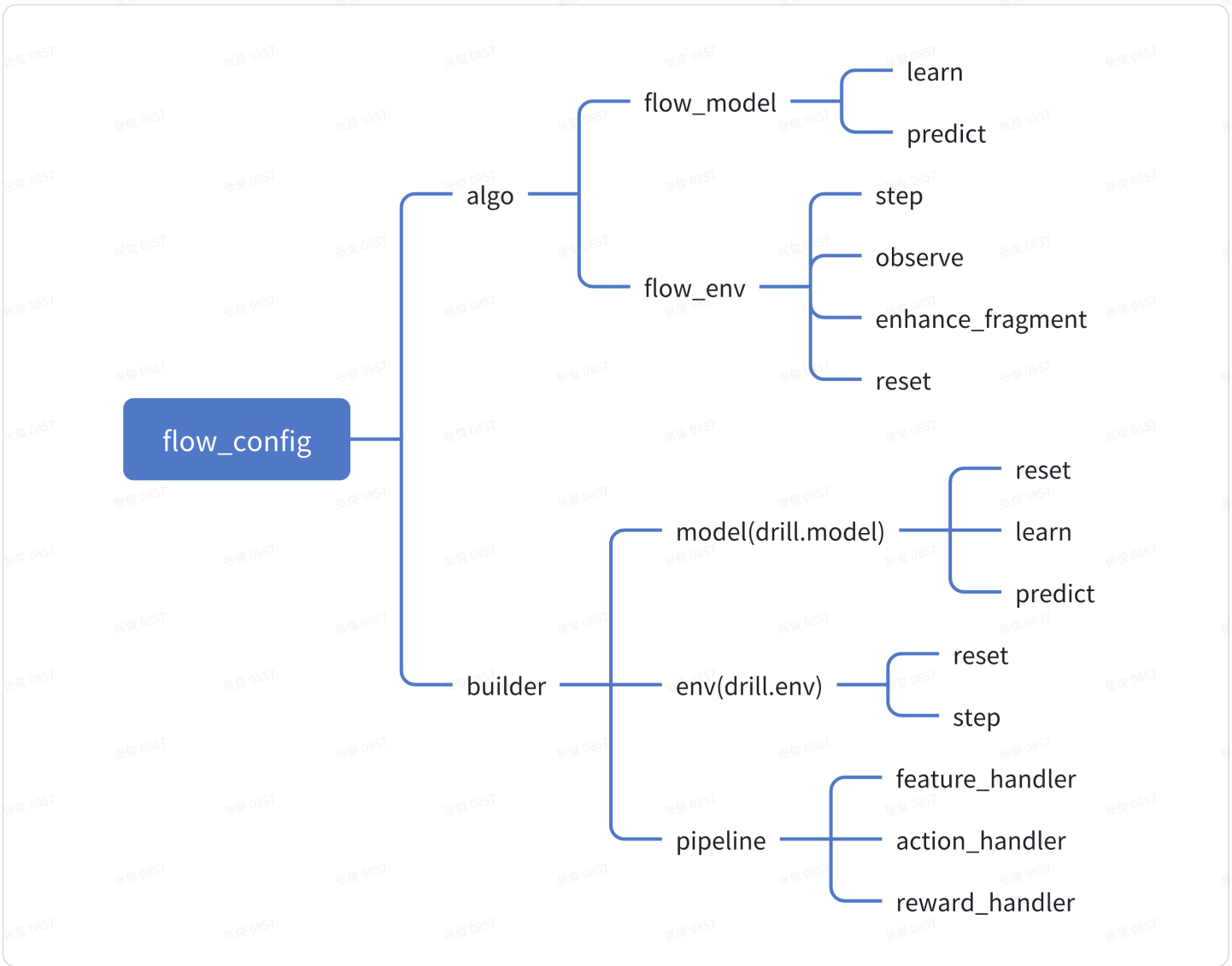
训练云接口梳理



总述：

训练云训练需要智能体与仿真交互，产生训练数据并进行强化学习训练，FlowEnv和FlowModel为两个基础类。

为方便用户开发，drill提供了drill env和drill model两个类，实际实现时，可将一些基础流程性的逻辑固化到FlowEnv和FlowModel中（如对pipeline的处理），在drill env和drill model里实现算法定制化内容。其中drill env和drill model是在FlowEnv和FlowModel中通过builder的函数实例化的。

自定义算法实现流程

FlowEnv、drill env开发-->FlowModel、drill model开发-->如果需要使用drill的pipeline，则需要进行piepline函数开发（feature_handler、action_handler、reward_handler），用户也可以根据需要自定义pipeline-->builder成员函数定制化改造

1、实现环境对接接口Env类（drll env）

基类：from drill.env import Env

```python
class Env(ABC):

    @abstractmethod
    def __init__(self, env_id: int, extra_info: dict):
        raise NotImplementedError

    @abstractmethod
    def reset(self) -> Any:
        """ 重置环境并返回初始的observation
        Returns:
            observation(object): 初始的observation
        """
        raise NotImplementedError

    @abstractmethod
    def step(self, command_dict: Dict) -> Tuple:
        """ 环境执行command指令
        我们的Env.step接口和gym.Env.step有些不同，gym.Env.step要求返回(observation, reward, done, info),
        相比gym.Env.step，我们并没有返回reward，原因是工业场景下很多env并没有返回reward的能力，比如说，reward
        可能需要agent自己根据observation进行提取。当然，如果env本身可以返回reward，可以通过info返回。
        Args:
            command_dict(Dict): 每个agent执行的指令
        Returns:
            observation(object): 环境当前的observation
            done(bool): episode是否结束
            info(dict): 其他有用的信息
        """
        raise NotImplementedError
```

实现FlowEnv类，__init__函数中self._env = builder.build_env(env_id, env_extra_info)获取的env，就是上述drill env实例。

```python
from __future__ import annotations

from collections import defaultdict
from typing import TYPE_CHECKING, Any, Dict, List, Union

import numpy as np

if TYPE_CHECKING:
    from drill.builder import Builder


class FlowEnvBase:
    """ [flow.api.Environment](https://docs.inspir.work/flow/flow/tutorial.html#flow.api.Environment) 的一个实现,
    负责和 model 交互产生样本

    Attributes
    ----------
    environment_description: flow.api.EnvironmentDescriptor
        云端训练，使用 flow.EnvironmentDescriptor
        本地调试，使用 drill.local.local.EnvironmentDescription
    """

    def __init__(self, environment_description: 'flow.EnvironmentDescriptor'):
        builder: Builder = environment_description.environment_creator_user_args['builder']
        env_id = environment_description.node_id * environment_description.num_envs_on_this_node \
                 + environment_description.environment_id_on_this_node
        env_extra_info = {'node_id': environment_description.node_id,
                          'num_envs_on_this_node': environment_description.num_envs_on_this_node,
                          'environment_id_on_this_node': environment_description.environment_id_on_this_node}
        env_extra_info.update(environment_description.environment_creator_user_args['extra_info'])
        self._builder = builder
        self._env = builder.build_env(env_id, env_extra_info)
        self._pipeline = builder.build_pipeline()

        self.flow_env_config = {}
        self._episode_done = None  # record whether the episode is over
        self._obs_data = None

```

```python
        # 上个 action, logits, value
        # <key = agent name, value = {logits, action, value}>
        self._last_hidden_state_dict = defaultdict(dict)
        # 当前剩余 agents
        self._agent_names = []
        # mask history {agent_name: [{decoder_name: mask}]}
        # self._dynamic_mask_history = defaultdict(list)
        self._last_agent_to_reward = {}

    @property
    def agent_names(self) -> List[str]:
        """ 返回环境中当前剩余的 agents 的名字

        Returns
        -------
        List[str]
            agent_names
        """
        return self._agent_names

    @property
    def env(self):
        return self._env

    @property
    def builder(self):
        return self._builder

    def reset(self) -> None:
        """ 重置状态，开始一个新的 episode
        """
        raise NotImplementedError

    def _get_hidden_state(self, agent_name) -> Dict:
        """ 获取 agent 的 hidden_state

        Parameters
        ----------
        agent_name : str
            agent 的名字

        Returns
        -------
        Any
            hidden_state
        """
        raise NotImplementedError
```

```
86
87      def observe(self) -> Dict[str, Dict[str, Union[Dict, str]]]:
88          """ 获取 observation
89
90          Returns
91          -------
92          Dict[str, Dict[str, Union[Dict, str]]]
93              需要让 model inference 的 agent 的 observation。包括网络需要的输入以及对
    应的 model,
94              另外还包括 reward 和 done
95
96          Examples
97          --------
98          return
99          ```python
100         observe_return = {
101             "red_agent":
102                 {"obs":
103                     {"spatial": np.ndarray,
104                         "entity": np.ndarray,
105                         "reward": array,
106                         "hidden_state": Any,
107                         ...
108                     },
109                 "model": "battle_model"}}
110             "blue_agent": ...
111         }
112         ```
113         """
114         # 1. 调用 pipeline 将环境返回的状态信息(observation)处理成网络可识别
115         # 的 state, reward (环境可能不会返回reward)
116         # example state_dict format {agent_name: {fs.name: processed_obs}}
117         # example reward_dict format {agent_name: reward}
118         raise NotImplementedError
119
120     def step(self,
121              agent_name: str,
122              predict_output: Dict[str, Any]
123              ) -> Dict[str, Dict[str, np.ndarray]]:
124         """ 根据 model inference 的结果和环境进行交互。
125         model 根据 observation 做 inference 得到的结果通过参数 `predict_output` 返
    回。
126         注意: 多智能体时, `observe` 可能同时返回所有 agent 的 observation 给 model
    做
127         inference, 但是 `step` 的参数 `predict_output` 只包含参数 `agent_name` 对应
    的
128         inference 结果。且不保证不同 agent step 调用的顺序。
```

```
129
130         Parameters
131         ----------
132         agent_name : str
133             agent 的名字
134         predict_output : Dict[str, Any]
135             `agent_name` 对应的 model inference 的结果
136             包含该智能体的 action, logits, value, hidden_state。
137
138         Returns
139         -------
140         Dict[str, Dict[str, np.ndarray]]
141             decoder_mask。复杂一点的场景，action 是有多头的（multi-head），而
    action 会作为样本收集起来参与训练，
142             但并不一定每一个 head 都是“有效的”。举个例子，假设动作空间为
143
144             * meta: 移动/攻击
145             * postion: 目标位置
146             * target: 目标单位
147
148             当 meta 为移动的时候，postion 这个 head 是有效的，而 target 无效，所以
    target 这个 head 就不应该参与
149             loss 计算。decoder mask 的作用就是讲“无效的” head mask 掉，不参与 loss
    计算。
150
151         """
152         raise NotImplementedError
153
154     def enhance_fragment(self, agent_name: str, fragments: List[Dict]) -> None:
155         """ 对不断 `observe` 和 `step` 收集的数据进行处理，这里计算了 GAE
156
157         什么时候调用这个方法?
158         一次 `observe` 和 `step` 收集的数据记为一个 fragment，当收集到的数据达到
159         `fragment_size` （一个配置参数）时调用此方法
160
161         注意: fragments 只能原地修改，这个方法不接受返回值，这是由 flow 决定的
162
163         Parameters
164         ----------
165         agent_name : str
166             agent 的名字
167         fragments : List[Dict]
168             长度为 `fragment_size`，每一个元素都是 3 元组，分别对应 `observe` 的
169             返回值（准确的说是 `observe_return[agent_name]["obs"]`，`step` 的参数
170             `predict_output` 和 `step` 的返回值。
171         """
172         raise NotImplementedError
```

```
173
174     def render(self, **kwargs):
175         raise NotImplementedError
```

2、实现神经网络和强化学习算法接口Model类（drill model）

基类：from drill.model.model import Model

```
1  from abc import ABC, abstractmethod
2
3
4  class Model(ABC):
5
6      @property
7      @abstractmethod
8      def network(self):
9          raise NotImplementedError
10
11     @abstractmethod
12     def predict(self, state_dict: dict) -> dict:
13         """
14
15         Parameters
16         ----------
17         state_dict : dict
18             前向所需要的输入，具体内容由Pipeline.pre_process决定。对于RLPipeline
   来说，
19             会包含obs2state返回的对应agent的所有key value。 Example
20
21                 {
22                     "units": xxx,
23                     "image": xxx,
24                     "common": xxx
25                 }
26
27         Returns
28         -------
29         predict_output_dict : dict
30             前向预测输出，通常包含logits，action，
31             value，hidden state(如果网络中有rnn的话) 。Example
32
33                 {
34                     "logits_move": xxx,
```

```
                    "logits_attack": xxx,
                    "action_move": xxx,
                    "action_attack": xxx,
                    "value": xxx,
                    "hidden_state": xxx
                }

        """
        raise NotImplementedError

    def learn(self, state_dict: dict, behavior_info_dict: dict) -> dict:
        """some quick notes

        Parameters
        ----------
        state_dict : dict
            参见Agent.predict，不同的是这里的state_dict
            可能是从replay buffer中采样得到的
        behavior_info_dict : dict
            对于state_dict，agent作出的决策信息。
            包含Agent.predict对应的输出，还可能包含Pipeline.post_process
            以及Pipeline.batch_process的输出，具体的内容依赖于与计算引擎
            的对接实现，比如对于flow.FlowModel以及flow.FlowEnv，会包含
            decoder_mask和advantage。Example

                {
                    "logits_move": xxx,
                    "logits_attack": xxx,
                    "action_move": xxx,
                    "action_attack": xxx,
                    "value": xxx,
                    "hidden_state": xxx,
                    "advantage": xxx,
                    "decoder_mask": xxx,
                }

        Returns
        -------
        summary : dict
            想要记录在tensorboard中的统计信息，注意dict的value只能是一个scalar。
Example

                {
                    "loss": xxx,
                    "value_loss": xxx,
                    "policy_loss": xxx,
                    "entropy": xxx
```

```
81                 }
82         """
83         raise NotImplementedError
```

3、实现pipeline基础函数：

action_handler、feature_handerl、reward_handler