МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по научно-исследовательской работе

Тема: Обучение агентов для выработки неконфликтных действий, направленных на достижение конкретных целей, с помощью кооперативного взаимодействия

Студент гр. 8303 _____ Удод М.Н.

Руководитель _____ Филатов А.Ю.

Санкт-Петербург
2023

ЗАДАНИЕ

НА НАУЧНО-ИССЛЕДОВАТЕЛЬСКУЮ РАБОТУ

Студент Удод М. Н.

Группа 8303

Тема практики: Обучение агентов для выработки неконфликтных действий, направленных на достижение конкретных целей, с помощью кооперативного взаимодействия

Задание на практику:

Необходимо разработать библиотеку, с помощью которой упрощается разработка агентов, направленных на взаимодействие с человеком.

Сроки прохождения практики:

Дата сдачи отчета:

Дата защиты отчета:

Студент _____ Удод М.Н.

Руководитель _____ Филатов А.Ю.

АННОТАЦИЯ

В данной работе рассматриваются проблема низкой эффективности взаимодействия человека и агента. Это связано с тем, что их поведение может отличаться, от предполагаемого агентами, а так же с дороговизной использования людей во время обучения. В качестве решения предлагается алгоритм, заключающийся в обучении человекоподобного агента с помощью алгоритмов имитационного обучения и дальнейшего дообучения остальных агентов с помощью многоагентного обучения в среде с человекоподобным агентом. После чего разрабатывается библиотека, реализующая описанный алгоритм.

SUMMARY

This paper addresses the problem of low efficiency of human-agent interaction. This is due to the high cost of using people during training, as well as the fact that their behavior may differ from that expected by the agents. As a solution, an algorithm is proposed that consists of training a humanoid agent using imitation learning algorithms and further training of the remaining agents using multi-agent learning in an environment with a humanoid agent. After which a library is developed that implements the described algorithm.

Содержание

1. ПОСТАНОВКА ЗАДАЧИ

В настоящее время искусственный интеллект (ИИ) все сильнее проникает в жизнь людей. Так, по данным SimilarWeb[1], число ежедневных уникальных пользователей ChatGPT превысило 25 миллионов спустя 2 месяца после запуска и продолжает расти. На ноябрь 2023-го данное число составляет 56 миллионов.

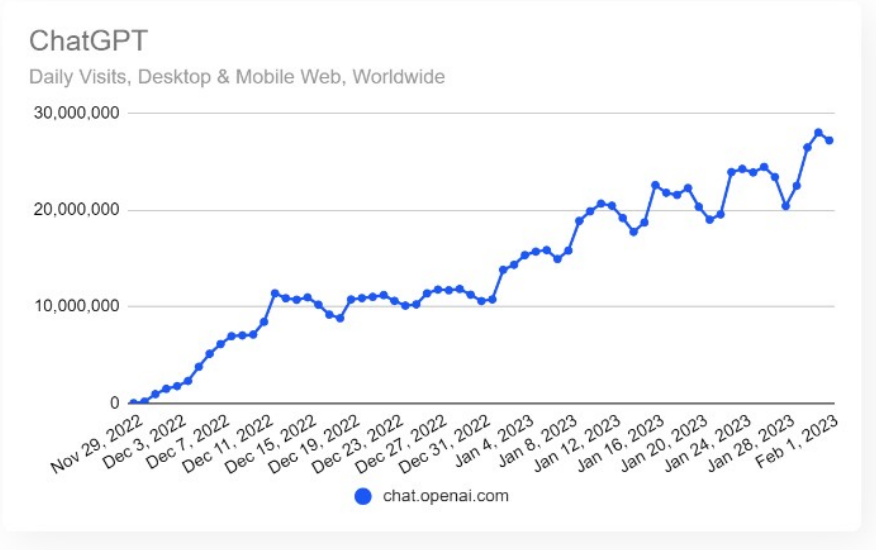

Рисунок 1. График роста числа пользователей ChatGPT

Другим примером такого продукта являются автомобили, имеющие возможность автопилота. Искусственный интеллект может применяться как на этапе восприятия, для решения задачи распознавания образов на изображении, так и на этапе планирования маршрута [2]. Согласно отчету за 2021 год [3], продажи электромобилей компании Tesla, частично обладающие такими возможностями [4], в 2021 году составили около 1 миллиона штук, что на 87% больше по сравнению с 2020 годом.

Из приведенных данных можно сделать вывод, что увеличивается число ситуаций в которых человек взаимодействует с ИИ, а значит необходимы исследования по правильной организации такого взаимодействия. Существующие решения в данной области обычно рассматривают ситуацию, когда человек может считаться экспертом. Но с развитием ИИ возрастет число задач, в которых он достигает достаточной производительности, чтобы взаимодействовать с человеком на равных или превосходить его. Примерами являются: интеллектуальное распределение трафика [5], управление автомобилем [6][7] и роботами [8].

При обучении подобным задачам каждый агент управляется своим собственным ИИ из-за чего у них могут появляться предположения о поведении друг-друга. Так как поведение человека может отличаться, то во время использования таких систем может наблюдаться снижение производительности.

Цель работы состоит в разработке инструмента для обучения агентов выработки неконфликтных действий в условиях взаимодействия с человеком

Задачи, которые необходимо выполнить для достижения поставленной цели:

1. Исследование методов машинного обучения, используемых во многоагентных средах
2. Исследование методов имитационного обучения
3. Разработка алгоритма обучения агента в условиях взаимодействия с человеком
4. Разработка способа описания пространств действий и наблюдения с помощью которого упрощается описание среды и человеческой политики
5. Тестирование разработанного алгоритма путем проведения сравнения результатов до и после дообучения

Проблема, рассматриваемая в данной работе - низкая эффективность взаимодействия человека и агента

Объектом исследования является обучение интеллектуальных агентов.

Предмет исследования - методы обучения агентов, направленные на выработку неконфликтных действий

Практическая ценность работы заключается в том, что разработанный инструмент позволит увеличить производительность решений на основе искусственного интеллекта в условиях взаимодействия с человеком.

2. РЕЗУЛЬТАТЫ РАБОТЫ В ОСЕННЕМ СЕМЕСТРЕ

План работ на осенний семестр:

1. Изучение методов имитационного обучения
2. Добавление в HAGL поддержки сред PettingZoo
3. Добавление поддержки интерфейса PettingZoo в MARLlib
4. Реализация возможности обучения агентов с использованием MARLlib, где один из агентов обучен с помощью имитационного обучения

Исходный код разработанного инструмента доступен на GitHub [9]. Пункты 3 и 4 описаны в рамках описания метода решения.

2.1. Добавление в HAGL поддержки сред PettingZoo

Human Abstraction Gymnasiu Language (HAGL) — это разработанный раннее способ описания пространств Gymnasium на более высоком уровне абстракции, благодаря чему можно избежать явного написания кода для преобразования полученных средой наблюдений в форимат, удобный для написания кода. В ходе предыдущей работы была добавлена поддержка сред Gymnasium и теперь была расширена и для PettingZoo — версии Gymnasium для многоагентных сред.

Как и в случае с Gymnasium, для этого был разработана среда, которая принимает на вход другую среду, использующую HAGL, и при этом предоставляет интерфейс PettingZoo. Код такой среды-оболочки реализован таким образом, чтобы преобразовать запросы к интерфейсу PettingZoo в запросы к оригинальной среде.

Рисунок 2. Среда-оболочка для реализации PettingZoo

2.2. Изучение методов имитационного обучения

В некоторых ситуациях требуется обучить политику на основе известной траектории. Траекторией называется набор последовательных переходов. Переход - это действие и полученные в ответ на него наблюдение и вознаграждение. Для получения траектории может использоваться как человек, так и другая обученная политика. Для обучения политики на основе указанных данных могут использоваться алгоритмы имитационного обучения.

Наивный подход к решению задачи имитационного обучения заключается в использовании обучения с учителем [10]. Так, на основе траекторий можно получить набор пар наблюдений и совершенных в ответ действий. Благодаря этому задача обучения может быть сформулирована как задача регрессии и решаться соответствующими алгоритмами.

При использовании обучения с учителем не учитывается, что действия обучающейся политики влияют на наблюдения, которые он получит в будущем. Таким образом возникает проблема нестационарности, так как нарушаются предположения. на которых основаны классические методы обучения с учителем. Одно из решений данной проблемы предлагается в алгоритме DAger [11]. В нем после каждого шага обучения эксперту необходимо предоставить новые траектории, но уже в среде с новой, обученной на основе предыдущих данных, политикой. Затем обучение повторяется с использованием информации, полученной на всех предыдущих шагах.

В предыдущих подходах не учитывалась информация о вознаграждении. Один из способов использовать ее заключается в обучении сети, вычисляющей вознаграждение на основе данных наблюдений и действий, данный вид алгоритмом получил название обратного обучения с подкреплением. После обучения функции вознаграждения агент может быть получен с помощью алгоритмов обучения с подкреплением на основе обученной функци. Недостатком данного подхода является необходимость большого числа вычислений.

Попытка преодоления ограничений обратного обучения с подкреплением представлена в алгоритме Generative Adversarial Imitation Learning (GAIL) [12]. Для этого вместо явного обучения функции вознаграждения предлагается использовать генеративно-состязательные сети, одна из которых будет генерировать действия, а вторая пытаться отличить сгенерированные действия

от действия эксперта. При этом во время обучения у среды запрашивается информация о новых переходах, что, в отличии от DAger, не требует прямого взаимодействия с экспертом.

Во всех описанных выше алгоритмах обучение основано на данных, полученных от эксперта, но в некоторых ситуациях их получение затруднено. Примером такой задачи может быть управлением не гуманоидным роботом. Также не для всех задач может быть четко сформулировано вознаграждение. В таких случаях может быть использован алгоритмы на основе сравнения предпочтений [13]. В них вместо записей траекторий предлагается демонстрировать эксперту две сгенерированных траекторий и просить его выбрать лучшую из них.

3. ОПИСАНИЕ ПРЕДПОЛАГАЕМОГО РЕШЕНИЯ

3.1. Описание алгоритма

Как было сказано выше, при взаимодействии человека и агента возникают проблемы, связанные с тем, что поведение человека может отличаться от поведения агента, который использовался во время обучения, из-за чего производительности может оказаться ниже. Данная проблема может быть решена участием человека уже во время обучения, но это:

- медленно - скорость принятия решения нейронными сетями выше, чем у человека
- дорого - при глубокое обучение с подкреплением требует для достижения результатов необходимо совершить в среде много миллионов шагов [14], что, с учетом времени принятия решения людьми, может оборачиваться в необходимость оплаты многих дней работы

Для решения этих проблем был предложен следующий алгоритм:

1. Получение начального приближения агентов с помощью глубокого многоагентного обучения с подкреплением
2. Запись человеческих траекторий
3. Обучение человекоподобного агента на основе полученных записей
4. Дообучение агентов в среде, где один из агентов заменен на человекоподобного

Таким образом во время обучения участвует не человек, но агент, имитирующий его поведение. В дальнейшем каждый шаг рассматривается более подробно.

3.2. Обучение с использованием многоагентных алгоритмов

Для того, чтобы записи траекторий имели значение с точки зрения отображения поведения человека необходимо, чтобы действия остальных агентов приводили к достижению поставленных целей. Таким образом, цель первого шага обучения заключается в получении начального приближения агентов, показывающего приемлемое поведение при взаимодействии с любым человеком.

Для этого проводится многоагентное обучение с использованием библиотеки MARLlib [15]. Ниже представлен алгоритм данного шага.

```
настройки = прочитать_настройки(аргументы.путь_к_файлу_настроек)
```

```
загрузить_пользовательский_код(настройки.путь_к_файлу_с_кодом)
настройки_алгоритма =
настроить_оптимизацию_гиперпараметров(настройки.алгоритм)
настройки_модели =
настроить_оптимизацию_гиперпараметров(настройки.модель)
путь_для_сохранения, путь_для_загрузки =
получить_настройки_сохранения(настройки.сохранение.шаг_1)
среда = создать_среду(настройки.среда)
алгоритм =
создать_алгоритм(настройки.название_алгоритма,настройки_алгоритма)
модель = создать_модель(среда, алгоритм, настройки_модели)
провести_обучение(среда, модель, настройки.шаг_1.время_обучения,
путь_для_сохранения, путь_для_загрузки)
```

Во время выбора алгоритма обучения используется алгоритм, указанный пользователем в настройках. Также для него устанавливаются указанные параметры, такие как:

- Число шагов для которых собирается информация во время одного цикла применения оптимизации
- Флаг перемешивания элементов в пакете перед применением оптимизации
- Размер шага оптимизации

И множество других, включая параметры, специфичные для выбранного алгоритма. При этом для всех настроек MARLlib предоставляет значения по умолчанию, поэтому обучение может быть запущено без их настройки. Это упрощает использование инструмента для пользователя, так как не от него не требуется глубокого понимание алгоритмов обучения.

Во время создания архитектуры сети также применяются пользовательские настройки. В MARLlib реализовано две архитектуры:

- Многослойный перцептрон
- Рекуррентная нейронная сеть - GRU [16] или LSTM [17]

Также для каждой их их можно добавить кодировщик, чьи слои будут находиться перед слоями основной сети. Доступны следующие кодировщики:

- Полносвязный
- Сверточная нейронная сеть

Также пользователь может настроить число слоев и количество нейронов в них. Размеры первого и последнего слоя сети определяются на основе пространств действий и наблюдений выбранной среды.

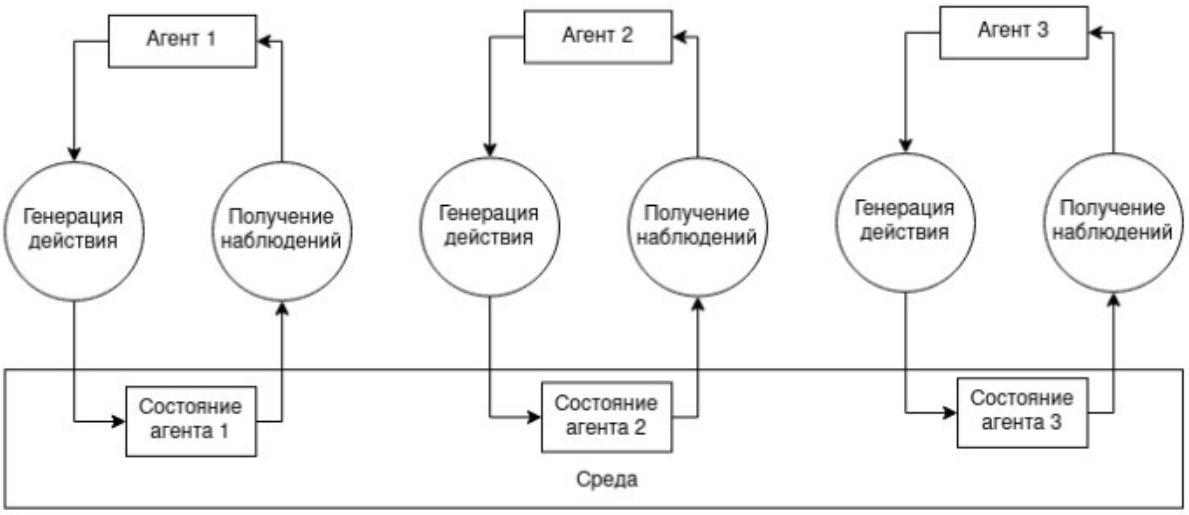

Рисунок 3. Обучение нескольких агентов

3.3. Сбор человеческих траекторий

Следующий шаг - сбор человеческий траекторий для дальнейшего обучения на их основе человекоподобного агента. Траектория - это набор действий и наблюдений, которые были получены от среды в ответ на эти действия. Для этого управление над одним из агентов, чье имя указывается в настройках, передается человеку. Так как человеку проще взаимодействовать с графическим интерфейсом, то для отображения среды используется метод render, имеющийся как в интерфейсе Gymnasium, так и в PettingZoo.

Траектории собираются с помощью библиотеки Minari [18], которая реализует среду DataCollectorV0. На вход данная среда принимает другую среду, реализующую интерфейс Gymnasium, записывает все действия, которые передаются в качестве аргумента при вызове метода step. Затем DataCollectorV0 вызывает метод step от оригинальной среды и, записав полученные наблюдения, возвращает их.

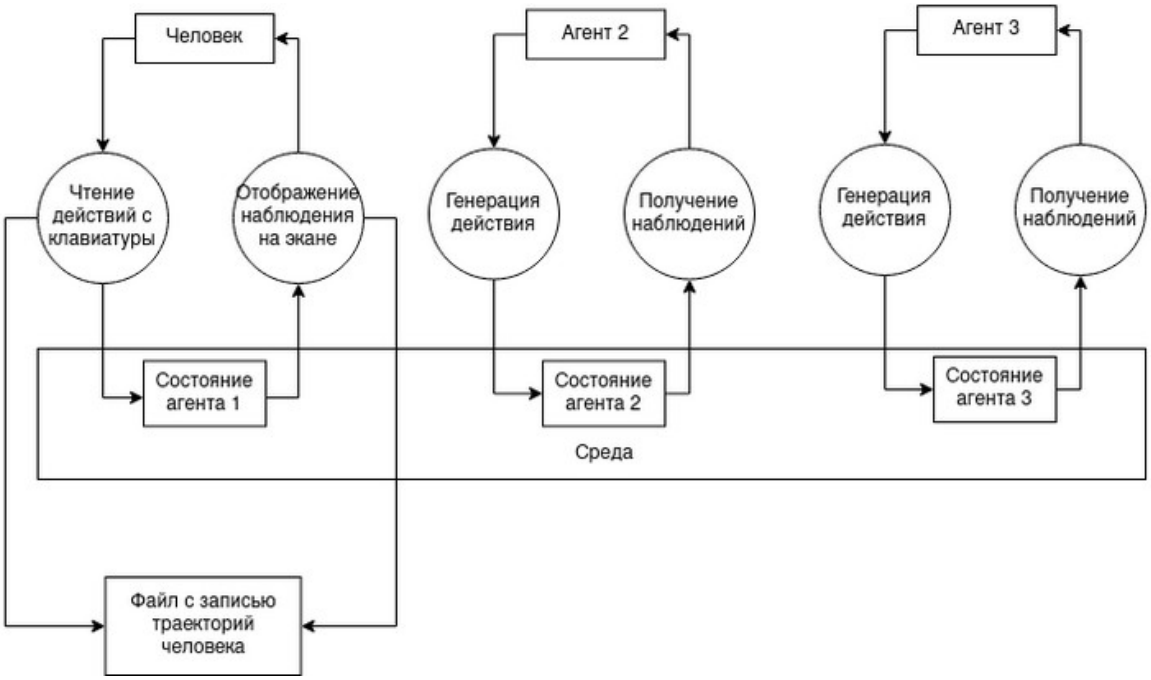

Рисунок 4. Замена одного из агентов на человека и запись его траекторий

Для того, чтобы использовать DataCollectorV0 необходимо преобразовать среду PettingZoo в Gymnasium. Так как на данном шаге нет необходимости обучать интеллектуальных агентов, то достаточно реализовать среду, которой на вход передаются среда и политики для всех агентов, кроме одного. Затем, при совершения действия, будут получены действия ото всех агентов и переданы оригинальной среде. Ниже представлен алгоритм для выполнения шага в такой среде:

```
функция шаг(действие):
действие_всех_агентов = {}
для каждой (агент, политика) в эта_среда.предустановленные_политики:
действие_всех_агентов[агент] = политика(эта_среда.наблюдения[агент])
действие_всех_агентов[эта_среда.управляемый_агент] = действие
эта_среда.наблюдение, вознаграждение, эпизод_завершен =
эта_среда.оригинальная_среда.шаг(действие_всех_агентов)
вернуть эта_среда.наблюдение[эта_среда.управляемый_агент],
вознаграждение[эта_среда.управляемый_агент],
эпизод_завершен[эта_среда.управляемый_агент]
```

Затем проводится сбор данных, для этого в среде выполняется число эпизодов, указанное в настройках. Этот процесс описывается следующим алгоритмом:

```
функция сбор_траекторий(среда, политика, число_эпизодов):
текущий_эпизод = 0
пока текущий_эпизод !=число_эпизодов:
эпизод_завершен = 0
наблюдение = среда.сбросить()
пока не эпизод_завершен:
действие = политика(наблюдение)
```

```
наблюдение, вознаграждение, эпизод_завершен = среда.шаг(действие)
среда.отобразить()
текущий_эпизод += 1
```

3.3.1. Сбор траекторий с помощью CrowdPlay

Недостаток описанного выше способа заключается в том, что для него требуется передавать человеку как модель, полученную на шаге 1, так и программу для сбора траекторий. Также у пользователя может быть устройство, чьей мощности окажется недостаточно для запуска модели. В таком случае может использоваться CrowdPlay [19]. Как и описанный выше способ, CrowdPlay позволяет записывать траекторий. Главное отличие заключается в том, что CrowdPlay представляет собой web-приложение с клиент-серверной архитектурой. Таким образом человеку для записи траекторий необходимо только перейти по адресу и выбрать среду. Во время нахождении в среде на клиент будут отправляться наблюдение, которые отображаются человеку. Все действия, собираемые клиентом, будут отправляться серверу, который передаст их функции “шага” среды.

CrowdPlay имеет модульную архитектуру, которая позволяет легко добавлять новые среды, но используемый интерфейс несовместим с PettingZoo. Для преодоления этого была написана среда, реализующая интерфейс CrowdPlay и принимающая на вход PettingZoo-среду. Имеются следующие отличия:

- Для получения пространств действий и наблюдений агента используется поле action_space и observation_space, представляющие собой словарь из имени агента в соответствующее ему пространство. Для их создания в конструкторе оболочки для каждого агента вызывается функция action_space и observation_space.
- Наличие поля list_of_agents, который возвращает список всех возможных агентов. Так как смысл этого поля аналогичен полю possible_agents из PettingZoo, то было использовано именно оно.
- Наличие метода get_noop_action, возвращающего действие, совершение которого можно интерпретировать как “ничего не делать”. Он был реализован, как возвращающий ноль или его аналог в зависимости от пространства, используемого средой. Так, это может быть массивом заданной формой из нулей для пространст Box, MultiBinary, MultiDiscrete, ноль для Discrete, словарь с необходимыми ключами и

нулем, соответствующим пространству для этого ключа, в случае Dict и массив с соответствующими нулями для Tuple.

Другой особенностью CrowdPlay является то, что для хранения траекторий в нем используется база данных на основе MySQL. При этом предоставляется инструмент для их экспорта в SQLLite, но не имеется поддержки Minary. Для этого был разработан еще один инструмент, который принимает на вход путь до папки с траекториями в формате SQLlite и создающий на основе их набор траекторий в формате Minari. Как и в описанном выше случае, для их создания используется среда DataCollectorV0, но в данном случае нет среды, которую можно было бы передать ему на вход. Для этого была создана среда, которая на вход получает массив действия и вознаграждений для каждого шага, запоминает их, и каждый раз при вызове метода step возвращает элемент, соответствующий текущему шагу.

CrowdPlay также предоставляет интерфейс для подстановки интеллектуальных агентов в среду, при этом часть из них может управляться людьми. Для этого необходимо загрузить модель, обученную на шаге 1. Это было реализовано в классе MARLibPretrained, который принимает на вход путь до файла с сохраненной моделью и имя агента.

3.4. Обучение человекоподобного агента

Для обучения человекоподобного агента использовалась библиотека Imitation []. Разработанный инструмент позволяет выбирать алгоритм для имитационного обучения, но из-за особенностей каждого алгоритма был реализован собственный интерфейс.

У алгоритмов есть следующие особенности:

- Behaviour Cloning имеет собственный механизм сохранения и загрузки политики.
- У алгоритмов на основе генеративного подхода (GAIL, AIRL) в качестве аргументов принимается алгоритм Stable Baselines 3 и объект для обучения сети вознаграждения.
- В реализации SQIL не предусмотрено механизмов загрузки политики, поэтому они были написаны с использованием внутреннего поля, содержащего алгоритм Stable Baselines 3

Как и в случае предыдущего шага, Imitation принимает на вход Gymnasium-среду, поэтому на данном шаге также используется оболочка, подставляющая агентов, обученных на шаге 1.

Рисунок 5. Обучение человекоподобного агента

Как и в случае обучения с использованием многоагентных алгоритмов, на данном шаге также поддерживаются различные настройки используемых алгоритмов. Часть из алгоритмов обучения используют генеративный подход, поэтому для них могут настраиваться оба алгоритма. Так, для алгоритма копирования поведения доступны следующие настройки:

- Размер пакета экспертных данных
- Размер минипакета экспертных данных - градиентных спуск будет применяться до того момента, пока число обработанных данных не достигнет числа, указанного в предыдущей настройки, но за один раз будет использовано только число элементов, указанных в данной настройке. Это полезно в ситуациях, когда необходимо уменьшить количество потребляемой видеопамяти.
- Множитель, который применяется к значению энтропии
- Множитель, который применяется к значению евклидова расстояния

Для других алгоритмов в качестве настройки может быть указан “внутренний” алгоритм. Поддерживаются следующие алгоритмы: A2C [21], DDPG [22], DQN [23], PPO [24], QR-DQN [25], TD3 [26], TQC [27] и TRPO [28]. Для него также могут быть указаны настройки, предоставляемые Stable Baselines 3. Для GAIL и AIRL также может быть настроена еще одна сеть, используемая для отличия действий созданных генеративным алгоритмом и человеком.

3.5. Дообучение в условиях взаимодействия с человекоподобным агентом

Цель четвертого шага - обучение агентов в условиях взаимодействия с человеком. Для этого управление одним из агентов передается человекоподобному агенту, обученному на прошлом шаге.

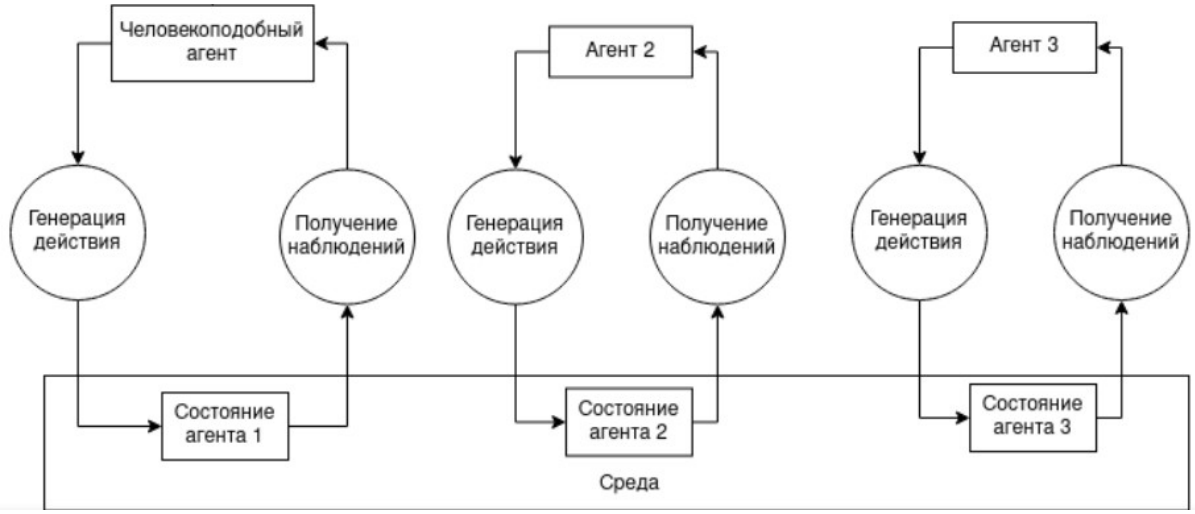

Рисунок 6. Замена одного из агентов на человекоподобного

Основная сложность реализации данного шага заключается в том, что необходимо обучать агентов с помощью MARLlib, используя при этом imitation для одного из них. Метод с создание особой среды с подставленными агентами неприменим в данном случае из-за необходимости обучения. Для решения может использоваться одна из настроек Ray RLlib, на которой основан MARLlib, позволяющая указать в качестве политики свой собственный объект. Сложность заключается в том, что MARLlib изолирует интерфейс Ray RLlib и предлагает свой, более простой. Для обхода этого в инструменте реализован вызов внутренних функций MARLlib напрямую.

На рисунке ниже представлена структура кода MARLlib. Разработчиками данной библиотеки предлагается использовать функцию Algo.fit, внутри которой, в зависимости от выбранного алгоритма, вызывается одна из трех функций:

- run_cc - для алгоритмов на основе централизованного критика
- run_vd - для алгоритмов на основе распределения вознаграждения
- run_il - для независимого обучения каждого агента с помощью алгоритмов глубокого обучения с подкреплением

Задача этих функций - подготовить соответствие агентов и политик, которыми они управляются. В реализованном инструменте данное соответствие строится самостоятельно. При этом для одного из агентов в качестве политики

устанавливается обученная на предыдущем шаге. В дальнейшем указанные функции вызывают функцию, соответствующую текущему алгоритму. Разработанный инструмент действует аналогичным образом, но передает в них собственный объект настроек.

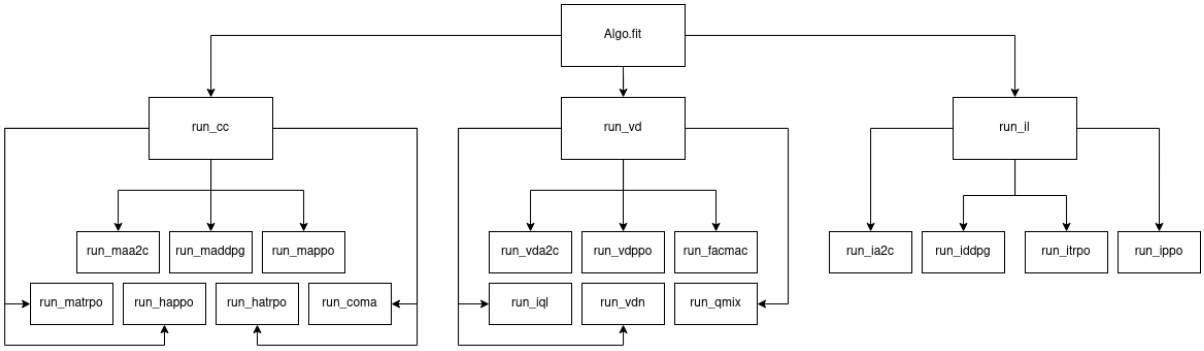

Рисунок 7. Функции, используемые MARLlib

Для описания соответствия политики и агента в Ray RLlib используется функция, которая принимает строку с именем агента и возвращает строку с именем политики. Соответствие имени политики и объекта, описывающего ее, также передается в настройках. В Ray RLlib политика - это экземпляр класса, унаследованного от Policy со следующими реализованными методами [29]:

- compute_actions - вычисляет действие для соответствующего наблюдения. Так как различным алгоритмам во время обучения может понадобиться дополнительная информация, то на вход принимается словарь, в котором кроме самих наблюдений может передаваться, например, в случае централизованного критика, действия других агентов
- postprocess_trajectory - реализует специфичную для алгоритма постобработку
- loss - вычисляет функцию потерь

Так как в данном случае требуется реализовать политику, которая передаст наблюдения политике из imitation, то функция postprocess_trajectory ничего не делала. Функцию loss не обязательно реализовывать, если не указать имя политики в настройке policies_to_train. Функция compute_actions передает полученные наблюдения в политику imitation и возвращает полученные от нее действия.

3.6. Просмотр результатов

Каждый из рассмотренных выше шагов оставляет после себя различные артефакты. При этом общая производительность решения, с точки зрения вознаграждения, может оказаться низкой, если не будет получен приемлемый

результат после каждого шага. Так, при недостаточно хорошем обучении агентов на шаге 1, траектории будут собраны в ситуации, когда агенты не смогут отреагировать на действия человека надлежащим образом. Ошибка во время обучения человекоподобного агента приведет к ситуации, что он не будет в достаточной мере передавать поведение человека и поэтому дообучение не покажет улучшения результатов. Из-за этого пользователю может понадобиться изучение результатов после каждого шага, а не только финального результата.

Для этого были реализованы следующие инструменты:

- После шага 1 - визуализация среды в которой действия всех агентов совершаются обученными политиками
- После шага 3 - визуализация среды в которой действия агентов контролируются политиками с шага 1, действия агента для которого собирались траектории на шаге 2 управляются человекоподобной политикой
- После шага 4 - визуализация среды, в которой используется человекоподобный агент, а действия остальных контролируются дообученными политиками

Также во время обучения собираются различные данные. Среди них:

- Средняя длина эпизода
- Максимальное, минимальное и среднее вознаграждение для каждого агента
- Значение функции потерь

Для визуализации данной информации может использоваться Tesorboard [30].

3.7. Настройка

В разрабатываемом инструменте используется множество библиотек, каждая из которых предоставляет свои настройки. Из-за этого необходим способ настройки, который бы позволил установить необходимые параметры для всего процесса обучения. Для этого в качестве аргумента командной строки можно передать путь до файла с настройками. Данный файл использует формат yml и состоит из следующих разделов:

- env - используется для настройки используемой среды, ее имя, карту, параметры самой среды и несколько параметров, предоставляемых MARLlib:
    - force_coop - используется для того, чтобы в качестве вознаграждения использовалась сумма вознаграждение всех агентов
    - mask_flag - некоторые среды предоставляют информацию о том, какие действия может совершить агент в текущий момент времени. Если значение данного параметра было установлено в положительное значение, то эта информация будет использоваться алгоритмами обучения
    - global_state_flag - алгоритмы многоагентного обучения, основанные на централизованном критике, используют во время обучения информацию обо всех агентах. Обычно для этого на вход критику подается наблюдения и действия сокомандников, но некоторые среды могут предоставлять более подробную информацию. При использовании данного флага алгоритмы обучения будут использовать эту информацию.
    - opp_action_in_cc - при использовании методов на основе централизованного критика на вход критику будут подаваться действия всех агентов, не только сокомандников
    - agent_level_batch_update - данный флаг показывает, что обновление политики должно происходить индивидуально для каждого агента, так как в данной работе необходимо заменить одного из агентов на человека, то их политики обязательно должны быть разделимы, поэтому для разработанного инструмента данное значение всегда истинно
- multiagent - описывает параметры первого шага обучения, время обучения в шагах среды, а также настройки алгоритма обучения и архитектуры сети, описанные выше
- rollout - позволяет указать параметры сбора траекторий, если для этого используется разработанный инструмент. Указывается число эпизодов и имя агента, над которым человек получит управление

- imitation - используется для настройки третьего шага, позволяет указать его длительность и настройки алгоритма, описанные выше
- retraining - описываются настройки четвертого шага обучения, указывается имена агентов, для которых необходимо продолжить обучения и его длительность
- save - позволяет указать пути, по которым будут сохраняться артефакты для каждого шага. Так, для шагов 1, 3 и 4 это политики, для шага 2 - записанные траектории.

При этом в инструменте реализована возможность продолжения обучения. Для этого в качестве сохранения можно передать объект с двумя полями:

- dir - по указанному пути будут сохраняться новые результаты
- checkpoint - путь до сохраненных политик

Данные настройки также учитываются при просмотре результатов - для визуализации будут использованы политики, загруженные из указанного пути. При этом, если запустить просмотр результатов без указания конкретного пути до сохранения, будет использован результат последнего обучения.

3.7.1. Настройка окружения

Одна из проблем, решаемая разрабатываемым инструментом - сложность установки всех необходимых библиотек в одном окружении. Так, по умолчанию нельзя установить imitation и MARLlib из-за того, что они используют одни и те же зависимости, но с разными версиями. Для разрешения конфликтов была создан сценарий на языке Bash, создающий окружение conda [31] с версиями библиотек, обеспечивающих правильную работу. Использованием других подходов, таких как создание файла requirments.txt [32] невозможно, так как не существует такого набора версий, при котором нет конфликта. В разработанном сценарии используется особенность pip [33], заключающаяся в том, что при установке проверяются конфликты только среди устанавливаемых библиотек, но не с уже существующими. Это позволяет принудительно установить необходимые версии после установки основных библиотек:

- Используются самые новые версии PettingZoo и supersuit - как было описано выше, разрабатываемый инструмент предоставляет свой

механизм добавления сред и поэтому данное изменение не влияет на работу MARLlib

- Используется версия numpy 1.21
- Используется torch версии 1.9.0 - из-за того, что imitation основан на библиотеке Stable Baselines 3, которая рассчитана на более новою версию, не удалось добиться работы SAC в качестве генеративного алгоритма

3.7.2. Оптимизация гиперпараметров

У алгоритмов машинного обучения существует множество параметров, которые влияют на обучение. При этом их вручную может занять много времени, для того, чтобы упростить данный процесс существуют инструменты, автоматизирующие его. Существует множество стратегий выбора параметров, в данной работе рассматриваются следующие:

- Полный поиск в указанном пространстве - для каждого параметра пользователь задает все возможные значения, в процессе поиска будут использованы все возможные комбинации
- Случайный поиск - для каждого параметра указывается вероятностное распределение с помощью которого оно должно генерироваться, в процессе поиска будет сгенерировано указанное число наборов параметров и для каждого проведено обучение

На шаге 1 используется библиотека MARLlib, основанная на Ray. Ray уже включает в себя инструмент оптимизации гиперпараметров Ray Tune [34] и MARLlib позволяет использовать его, но поддерживается только полный поиск.

На шаге 3 используется библиотека Imitation, у которой нет встроенной поддержки средств оптимизации гиперпараметров. Так как поддержка Ray Tune связана со сложностями из-за необходимости сериализации объектов, то было использовано другое средство оптимизации гиперпараметров - Optuna [35]. При этом была реализована также поддержка случайного поиска. Как и Ray, Optuna поддерживает распараллеливание и раннее прекращение обучения, если его результаты хуже предыдущих попыток.

Для настройки оптимизации гиперпараметров в файле настроек вместо конкретного значения указать одно из следующих значений:

- grid_search - полный поиск, в качестве параметра передаются все возможные значения параметра

- uniform - равномерное распределение, указываются минимальное и максимальное значение
- loguniform - логарифмически равномерное распределение, указываются минимальное и максимальное значение
- discrete_uniform - дискретное равномерное распределение - указываются минимальное и максимальное значение, а также шаг дискретизации
- int - равномерное распределение целых чисел, указываются минимальное и максимальное значение, шаг дискретизации и флаг, при указании которого используется логарифмически равномерное распределение
- float - вещественное распределение, аргументы аналогичны целочисленному распределению, но возвращает вещественные числа

3.7.3. Пользовательские среды

Модели на основе машинного обучения имеют большое количество применения в самых разных ситуациях, поэтому разработанный инструмент не может ограничиваться каким-то списком поддерживаемых сред. Так как в данной работе среды используют интерфейс PettingZoo, то пользователь может реализовать свою среду и добавить его в разработанный инструмент. Для этого в файле настройки может быть указан путь до файла с программой на языке Python, чей код будет исполнен при запуске инструмента. С ее помощью пользователь может зарегистрировать среду и, если ее имя будет указано в файле настроек, то при обучении агентов будет использована именно она.

При этом в MARLlib не реализована поддержка PettingZoo, а используется интерфейс из Ray с одним дополнением - среда также должна реализовать метод get_env_info, который возвращает дополнительную информацию о среде, включающую следующие поля:

- space_obs - пространство наблюдений
- space_act - пространство действий
- num_agents - число агентов в среде, если в течении времени оно может меняться, то указывается максимальное значение
- episode_limit - максимальное число шагов, которое может быть совершено в среде

- policy_mapping_info - содержит информацию о всех сценариях, реализованных в среде, и об агентах в них
    - Имя и описание сценария
    - team_prefix - указывает список комманд, при этом предполагается, что имена агентов начинается с названия команды - это необходимо для того, чтобы понять к какой команде принадлежит агент
    - all_agents_one_policy - флаг, указывающий, что при обучении в данной среде для всех агентов может быть использована одна политика. Так как в данной работе необходимо заменить одного из агентов на человека, то данная возможность не используется
    - one_agents_one_policy - флаг, указывающий, что при обучении в среде для каждого агента может использоваться отдельная политика

Так как PettingZoo не предоставляет такой информации, то она указывается пользователем при регистрации среды. Процесс регистрации заключается в добавлении в созданный для этих целей словарь, сопоставляющий имени среды функцию, с помощью которой она может быть создана. Разработанный инструмент предоставляет свои собственные функции регистрации, так как указанная функция должна не только создать PettingZoo-среду, но и обернуть ее в специальный объект, который будет предоставлять интерфейс, требуемый MARLlib, но в их реализации использовать методы оригинальной среды.

3.7.4. Политика человека

В разных средах реализация ее взаимодействия с человеком может отличаться. Так, для сред, подразумевающих графическое представление, для этого может понадобиться зафиксировать взаимодействие человека с устройством, такое как нажатие клавиши, движения мыши и затем сопоставить ей какое-то действие в среде. При этом в средах без графического представления может использоваться ввод и вывод в терминал. Для того, что бы пользователь мог сам реализовать необходимый способ взаимодействия, в файле с кодом предоставляется возможность указать необходимую функцию в переменной policy. В дальнейшем такой объект будет называться человеческой

политикой. Может быть указан любой объект, реализующий оператор вызова, который принимает на вход наблюдение и возвращает действие.

Также в реализованном инструменте имеется поддержка PyGame. Пользователь может обернуть описанный выше объект в вызов PyGamePolicy и в таком случае человеческая политика будет принимать на вход и текущую нажатую клавишу.

Подобный подход также позволяет упростить взаимодействие человека со средой путем дополнительных вычислений. Так, в футболе пространство действий состоит из направления движения и направления удара. Для указания направления удара могут быть использованы клавиши WASD или стрелочки, но направление удара может указываться разными способами, такими как указание направление мышью. В данной работе человеку не требуется указывать направление удара вовсе, так как оно вычисляется на основе текущего положения агента и положении ворот противника.

4. ПЛАН РАБОТ НА ВЕСЕННИЙ СЕМЕСТР

1. Валидация результатов работы инструмента
2. Измерение производительности инструмента — под производительностью понимается изменение вознаграждения за эпизод после дообучения
3. Публикация инструмента в PyPi

ЗАКЛЮЧЕНИЕ

В ходе данной работы были исследованы алгоритмы глубокого имиационного обучения, после чего был предложен алгоритм, который позволяет обучить агента в условиях взаимодействия с человеком. Затем был разработан инструмент, реализующий описанный алгоритм. Разработка велась с помощью языка Python и с использованием библиотек MARLlib и imitation. Также в HAGL была добавлена поддержка сред, использующих интерфейс PettingZoo. Как и в случаи Gymnasium, это было реализовано с помощью среды-обертки. Исходный код разработанного инструмента доступен на GitHub [9].

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Статистика числа пользователей ChatGPT // SimilarWeb URL: https://www.similarweb.com/blog/insights/ai-news/chatgpt-25-million (Дата обращения: 15.12.2023)
2. You, Changxi, et al. "Advanced planning for autonomous vehicles using reinforcement learning and deep inverse reinforcement learning." *Robotics and Autonomous Systems* 114 (2019): 1-18.
3. Отчет за 4 квартал 2021 года и 2021 год // Tesla CDN URL: https://tesla-cdn.thron.com/static/WIIG2L_TSLA_Q4_2021_Update_O7MYNE.pdf?xseo=&response-content-disposition=inline%3Bfilename%3D (Дата обращения: 15.12.2023)
4. Автопилот в автомобилях Tesla // Tesla URL: https://www.tesla.com/autopilot (Дата обращения: 15.12.2023)
5. Kouchaki, Mohammadreza, and Vuk Marojevic. "Actor-critic network for O-RAN resource allocation: xApp design, deployment, and analysis." *2022 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2022.
6. Li, Quanyi, et al. "Metadrive: Composing diverse driving scenarios for generalizable reinforcement learning." *IEEE transactions on pattern analysis and machine intelligence* 45.3 (2022): 3461-3475.
7. Peng, Zhenghao, et al. "Learning to simulate self-driven particles system with coordinated policy optimization." *Advances in Neural Information Processing Systems* 34 (2021): 10784-10797.
8. Levine, Sergey, et al. "End-to-end training of deep visuomotor policies." *The Journal of Machine Learning Research* 17.1 (2016): 1334-1373.
9. Исходный код разработанного инструмента // GitHub URL: https://github.com/TheShenk/hai-nonconflict-action-learning (Дата обращения: 15.12.2023)
10. Ross, Stéphane, and Drew Bagnell. "Efficient reductions for imitation learning." *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2010.
11. Ross, Stéphane, Geoffrey Gordon, and Drew Bagnell. "A reduction of imitation learning and structured prediction to no-regret online learning." *Proceedings of the fourteenth international conference on artificial*

intelligence and statistics. JMLR Workshop and Conference Proceedings, 2011.

12. Ho, Jonathan, and Stefano Ermon. "Generative adversarial imitation learning." *Advances in neural information processing systems* 29 (2016).
13. Christiano, Paul F., et al. "Deep reinforcement learning from human preferences." *Advances in neural information processing systems* 30 (2017).
14. Yu, Yang. "Towards Sample Efficient Reinforcement Learning." *IJCAI*. 2018.
15. Hu, Siyi, et al. "MARLlib: A Scalable and Efficient Multi-agent Reinforcement Learning Library." *Journal of Machine Learning Research* 24.315 (2023): 1-23.
16. Cho, Kyunghyun, et al. "Learning phrase representations using RNN encoder-decoder for statistical machine translation." *arXiv preprint arXiv:1406.1078* (2014).
17. Hochreiter, Sepp & Schmidhuber, Jürgen. (1997). Long Short-term Memory. Neural computation. 9. 1735-80. 10.1162/neco.1997.9.8.1735.
18. Minari Documentation // Minari Documentation URL: https://minari.farama.org (Дата обращения: 15.12.2023)
19. Gerstgrasser, Matthias, Rakshit Trivedi, and David C. Parkes. "CrowdPlay: Crowdsourcing Human Demonstrations for Offline Learning." *International Conference on Learning Representations*. 2021.
20. Gleave, Adam, et al. "imitation: Clean imitation learning implementations." *arXiv preprint arXiv:2211.11972* (2022).
21. Mnih, Volodymyr, et al. "Asynchronous methods for deep reinforcement learning." *International conference on machine learning*. PMLR, 2016.
22. Lillicrap, Timothy P., et al. "Continuous control with deep reinforcement learning." *arXiv preprint arXiv:1509.02971* (2015).
23. Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." *arXiv preprint arXiv:1312.5602* (2013).
24. Schulman, John, et al. "Proximal policy optimization algorithms." *arXiv preprint arXiv:1707.06347* (2017).
25. Dabney, Will, et al. "Distributional reinforcement learning with quantile regression." *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. No. 1. 2018.

26. Fujimoto, Scott, Herke Hoof, and David Meger. "Addressing function approximation error in actor-critic methods." *International conference on machine learning*. PMLR, 2018.
27. Kuznetsov, Arsenii, et al. "Controlling overestimation bias with truncated mixture of continuous distributional quantile critics." *International Conference on Machine Learning*. PMLR, 2020.
28. Schulman, John, et al. "Trust region policy optimization." *International conference on machine learning*. PMLR, 2015.
29. Base Policy Class // Ray Rllib URL: https://docs.ray.io/en/releases-1.11.2/rllib/package_ref/policy/policy.html (Дата обращения: 15.12.2023)
30. Tensorboard // Tensorflow URL: https://www.tensorflow.org/tensorboard (Дата обращения: 15.12.2023)
31. Conda // Conda Documentation URL: https://docs.conda.io/en/latest/ (Дата обращения: 15.12.2023)
32. Requirments File Format // Pip Documentation URL: https://pip.pypa.io/en/stable/reference/requirements-file-format/ (Дата обращения: 15.12.2023)
33. Pip // Pip Documentation URL: https://pip.pypa.io/en/stable/ (Дата обращения: 15.12.2023)
34. Liaw, Richard, et al. "Tune: A research platform for distributed model selection and training." *arXiv preprint arXiv:1807.05118* (2018).
35. Akiba, Takuya, et al. "Optuna: A next-generation hyperparameter optimization framework." *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 2019.