МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ
Кафедра информационных систем управления

ПЕТРИКЕВИЧ Тимофей Юрьевич

АЛГОРИТМЫ ПРОХОЖДЕНИЯ ПЕРЕКРЕСТКОВ В МОДЕЛЯХ СИМУЛЯЦИИ ДОРОЖНОГО ДВИЖЕНИЯ

Дипломная работа

Научный руководитель:
доктор технических наук,
профессор В.В. Краснопрошин

Допущена к защите
«____»_____20___г.
Заведующий кафедрой ИСУ
доктор технических наук, доцент А.М. Недзьведь

Минск, 2025

ОГЛАВЛЕНИЕ

РЕФЕРАТ

Дипломная работа: 53 страницы, 30 рисунков, 17 источников

Ключевые слова: ПЕРЕКРЁСТОК, АВТОНОМНЫЙ АГЕНТ, МОДЕЛИРОВАНИЕ, АЛГОРИТМЫ ПРОХОЖДЕНИЯ, МАШИННОЕ ОБУЧЕНИЕ С ПОДКРЕПЛЕНИЕМ, ОБОБЩЕННАЯ СРЕДА, СЦЕНАРИИ, ТЕСТИРОВАНИЕ, ИМИТАЦИЯ ДОРОЖНОГО ДВИЖЕНИЯ.

Объект исследования – алгоритмы прохождения перекрёстков в моделях имитации дорожного движения с применением машинного обучения с подкреплением.

Предмет исследования – методы разработки и тестирования обобщённой среды для обучения и оценки алгоритмов пересечения перекрёстков автономными агентами.

Цель исследования – анализ и сравнение существующих подходов к реализации алгоритмов пересечения перекрёстков автономными агентами, построение двухмерной обобщённой среды для их обучения, разработка набора сценариев с вариативными условиями и оценка эффективности алгоритмов в этой среде.

Методы исследования – методы машинного обучения, алгоритмы обучения с подкреплением, симуляционные модели дорожного движения, методы анализа и сравнения производительности, методы генерализации и тестирования.

Полученные результаты и их новизна – разработана двухмерная обобщённая тестовая среда с параметрической генерацией сценариев перекрёстков, реализованы и протестированы различные алгоритмы прохождения перекрёстков, проведена комплексная оценка их производительности в условиях неопределённости и изменяющихся параметров среды. Новизна заключается в оптимизации процесса тестирования алгоритмов путем перехода от трехмерной к двухмерной платформе для универсального тестирования алгоритмов принятия решений на перекрёстках с возможностью гибкой генерации сценариев и анализа способности моделей к обобщению.

Достоверность материалов и результатов дипломной работы – все представленные расчётные и аналитические материалы объективно отражают состояние исследуемой области, а все теоретические и методологические положения, заимствованные из литературных и иных источников, сопровождаются корректными ссылками.

Область возможного практического применения – разработанная среда может использоваться в академических и прикладных исследованиях в области автономного вождения, для тестирования и обучения алгоритмов в системах симулированного или реального автономного транспорта.

РЭФЕРАТ

Дыпломная праца: 53 старонкі, 30 малюнкаў, 17 крыніц

Ключавыя словы: ПЕРАКРЭСЦЕ, АЎТАНОМНЫ АГЕНТ, МАДЭЛІРАВАННЕ, АЛГАРЫТМЫ ПРАХОДЖАННЯ, ПАДМАЦАВАНАЕ МАШЫННАЕ НАВУЧАННЕ, АБАГУЛЕНАЕ АСЯРОДДЗЕ, СЦЭНАРЫІ, ТЭСЦІРАВАННЕ, ІМІТАЦЫЯ ДАРОЖНАГА РУХУ.

Аб'ект даследавання – алгарытмы праходжання перакрыжаванняў у мадэлях імітацыі дарожнага руху з прымяненнем падмацаванага машыннага навучання.

Прадмет даследавання – метады распрацоўкі і тэставання абагуленага асяроддзя для навучання і ацэнкі алгарытмаў праходжання перакрыжаванняў аўтаномнымі агентамі.

Мэта даследавання – аналіз і параўнанне існуючых падыходаў да рэалізацыі алгарытмаў праходжання перакрыжаванняў аўтаномнымі агентамі, пабудова двухмернага ўніфікаванага асяроддзя для іх навучання, распрацоўка набору сцэнараў з варыятыўнымі ўмовамі і ацэнка эфектыўнасці алгарытмаў у гэтым асяроддзі.

Метады даследавання – метады машыннага навучання, алгарытмы падмацаванага навучання, імітацыйныя мадэлі дарожнага руху, метады аналізу і параўнання прадукцыйнасці, метады абагульнення і тэставання.

Атрыманыя вынікі і іх навізна – распрацавана двухмернае ўніфікаванае тэставае асяроддзе з параметрычнай генерацыяй сцэнараў перакрыжаванняў, рэалізаваны і пратэставаны розныя алгарытмы праходжання перакрыжаванняў, праведзена комплексная ацэнка іх прадукцыйнасці ва ўмовах нявызначанасці і змяняльных параметраў асяроддзя. Навізна заключаецца ў аптымізацыі працэсу тэставання алгарытмаў шляхам пераходу ад трохмернай да двухмернай платформы для ўніверсальнага тэставання алгарытмаў прыняцця рашэнняў на перакрыжаваннях з магчымасцю гнуткай генерацыі сцэнараў і аналізу здольнасці мадэляў да ўніфікацыі.

Дакладнасць матэрыялаў і вынікаў дыпломнай працы – усе прадстаўленыя аналітычныя і разліковыя матэрыялы правільна і аб'ектыўна адлюстроўваюць стан даследаванай галіны, а ўсе тэарэтычныя і метадалагічныя палажэнні, запазычаныя з літаратурных і іншых крыніц, суправаджаюцца адпаведнымі спасылкамі.

Вобласць магчымага практычнага прымянення – распрацаванае асяроддзе можа выкарыстоўвацца ў навуковых і прыкладных даследаваннях у галіне аўтаномнага транспарту, для тэставання і навучання алгарытмаў у сістэмах імітацыйнага або рэальнага аўтаномнага кіравання.

SUMMARY

Diploma work: 53 pages, 30 figures, 17 references

Keywords: INTERSECTION, AUTONOMOUS AGENT, SIMULATION, CROSSING ALGORITHMS, REINFORCEMENT LEARNING, GENERALIZED ENVIRONMENT, SCENARIOS, TESTING, TRAFFIC SIMULATION

The object of the research – algorithms for intersection crossing in traffic simulation models using reinforcement learning.

The subject of the research – methods for developing and evaluating a generalized environment for training and testing autonomous agents' intersection-crossing algorithms.

The aim of the research – to analyze and compare existing approaches to implementing intersection-crossing algorithms for autonomous agents, to develop a two-dimensional generalized environment for training them, to design a set of scenarios with varying conditions, and to evaluate the efficiency of these algorithms within the environment.

Research methods – machine learning techniques, reinforcement learning algorithms, traffic simulation models, performance analysis and comparison methods, generalization and testing methods.

The results of the work and their novelty – a two-dimensional generalized test environment with parametric intersection scenario generation was developed; various intersection-crossing algorithms were implemented and tested; and a comprehensive evaluation of their performance under uncertainty and changing environmental parameters was conducted. The novelty lies in optimizing the algorithm testing process by transitioning from a three-dimensional to a two-dimensional platform, enabling universal testing of decision-making algorithms at intersections with flexible scenario generation and the ability to analyze the generalization capabilities of models.

Authenticity of the materials and results of the diploma work – all analytical and computational materials presented correctly and objectively reflect the state of the studied problem, and all theoretical and methodological concepts borrowed from literature and other sources are accompanied by appropriate references.

Recommendations on the usage – the developed environment can be used in scientific and applied research in the field of autonomous transport for testing and training algorithms in simulated or real-world autonomous driving systems.

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ

ПО – программное обеспечение

A2C – Advantage Actor-Critic (актор-критик с преимуществом)

A3C – Asynchronous Advantage Actor-Critic (асинхронный актор-критик с преимуществом)

ADAM – Adaptive Moment Estimation (адаптивная оценка моментов)

API – Application Programming Interface (интерфейс прикладного программирования)

CMDP – Contextual Markov Decision Process (контекстный марковский процесс принятия решений, КМППР)

CNN – Convolutional Neural Network (сверточная нейронная сеть)

C51 – Categorical DQN with 51 atoms (категориальный DQN с 51 категорией)

Curriculum Learning – поэтапное обучение

DDPG – Deep Deterministic Policy Gradient (глубокий детерминированный градиент стратегии)

DNN – Deep Neural Network (глубокая нейронная сеть)

DQN – Deep Q-Network (глубокая Q-сеть)

DVS – Dynamic Vision Sensor (динамический датчик видения)

Ensemble Methods – ансамбли стратегий

GAE – Generalized Advantage Estimation (обобщенная оценка преимущества)

GMM – Gaussian Mixture Model (смесь гауссовых моделей)

GNSS – Global Navigation Satellite System (глобальная навигационная спутниковая система)

GPS – Global Positioning System (система глобального позиционирования)

GRU – Gated Recurrent Units (управляемые рекуррентные блоки)

HMM – Hidden Markov Models (скрытые модели Маркова)

IID – Independent and Identically Distributed (независимо и одинаково распределенные)

IMU – Inertial Measurement Unit (гиростабилизатор)

KL – Kullback-Leibler divergence (дивергенция Кульбака–Лейблера)

LiDAR – Light Detection and Ranging (обнаружение и определение дальности с помощью света)

LSTM – Long Short Term Memory (нейронная сеть с долгой краткосрочной памятью)

MAML – Model-Agnostic Meta-Learning (мета-обучение, не зависящее от модели)

MDP – Markov Decision Process (марковский процесс принятия решений, МППР)

Meta-RL – Meta Reinforcement Learning (мета-обучение с подкреплением)

MVP – Minimum Viable Product (минимально жизнеспособный продукт)

Off-policy – обучение с использованием данных, полученных другой стратегией

OOD – Out-of-Distribution (вне распределения)

PPO – Proximal Policy Optimization (оптимизация ближайшей стратегии)

PoC – Proof of Concept (доказательство концепции)

Q-learning – метод обучения действий по оценочной функции

RL – Reinforcement Learning (обучение с подкреплением)

RNN – Recurrent Neural Network (рекуррентная нейронная сеть)

RSS – Residual Sum of Squares (остаточная сумма квадратов)

SAC – Soft Actor-Critic (мягкий актор-критик)

TD – Temporal Difference (временная разность)

TD3 – Twin Delayed DDPG (отложенный двойной DDPG)

TRPO – Trust Region Policy Optimization (оптимизация стратегии с доверительным ограничением)

V2X – Vehicle to X (транспорт с X)

ВВЕДЕНИЕ

С развитием технологий автономного транспорта возрастает потребность в создании безопасных и эффективных алгоритмов управления транспортными средствами в условиях реального дорожного движения. Одной из важнейших задач для автономных транспортных средств является пересечение перекрестков, которые являются одними из самых сложных и потенциально опасных участков на дорогах.

Перекрестки характеризуются высокой степенью неопределенности и требуют точного и быстрого принятия решений, чтобы предотвратить аварийные ситуации и обеспечить безопасность всех участников движения.

Основная проблема, связанная с пересечением перекрестков, заключается в необходимости учета множества факторов, таких как скорость и траектория движения других транспортных средств, приоритет проезда, а также наличие пешеходов и велосипедистов.

Для решения этой задачи исследователи разрабатывают различные подходы, в том числе с использованием методов машинного обучения и искусственного интеллекта, которые позволяют моделировать поведение автомобилей и адаптировать их действия в реальном времени.

Одним из перспективных подходов является использование симуляционных моделей дорожного движения для тестирования и оптимизации алгоритмов прохождения перекрестков. Такие симуляции позволяют создавать виртуальные условия, максимально приближенные к реальным, что существенно упрощает процесс разработки и тестирования алгоритмов перед их внедрением.

В современных исследованиях часто используются виртуальные среды обучения, которые предоставляют возможности для глубокого анализа различных стратегий прохождения перекрестков на основе алгоритмов глубокого обучения и методов обучения с подкреплением.

Однако существующие виртуальные среды работают с трехмерной моделью окружения, что приводит к накладным затратам вычислительной мощности графического процессора, который также используется и для глубокого обучения.

Целью данной работы является создание двухмерной модели симуляции дорожного движения для обучения автономных агентов прохождению перекрестков, исследование существующих алгоритмов прохождения перекрестков при использовании машинного обучения с подкреплением, сравнение этих алгоритмов, их анализ, формирование генератора сценариев для созданной модели симуляции и его применение для подсчета производительности рассмотренных алгоритмов.

Такие генераторы сценариев характеризуются разнообразием условий, их рандомизацией и генерацией при каждой новой симуляции, при этом постоянством в постановке условий для разрабатываемой модели.

Развитие моделей и алгоритмов, способных показывать высокую производительность на генерализованных наборах сценариев, является важным и неотъемлемым шагом в разработке автономных агентов для управления дорожным транспортом.

Кроме того, динамическая среда с большим количеством сценариев и варьируемых условий является необходимой для развития актуальных моделей машинного обучения с подкреплением в общем.

Однако на данный момент количество сред машинного обучения с подкреплением, приемлемых для обучения генерализованных автономных агентов, ограничено всего несколькими примерами, лишь малая часть из которых связана с областью автономного транспорта.

Данный факт делает создание инфраструктуры в виде обобщенной среды для обучения автономного агента особенно актуальной задачей как в области симуляции дорожного движения, так и в машинном обучении с подкреплением в целом.

ГЛАВА 1 ПОСТАНОВКА ЗАДАЧИ

1.1 Общее описание автономного управления дорожным транспортом

Задача разработки алгоритмов для автономного транспорта может варьироваться в зависимости от уровня взаимодействия автомобилей друг с другом и окружающей средой. Существует множество подходов к организации коммуникации: от полностью независимых моделей, которые ориентируются исключительно на собственные датчики, до, так называемых, V2X систем, объединяющих транспортные средства с другими (V2V), инфраструктурой (V2I), спутниками (V2N) и пешеходами (V2P) с возможностью обмена информацией, такой как планируемые маневры, скорость и другие параметры (рисунок 1.1).

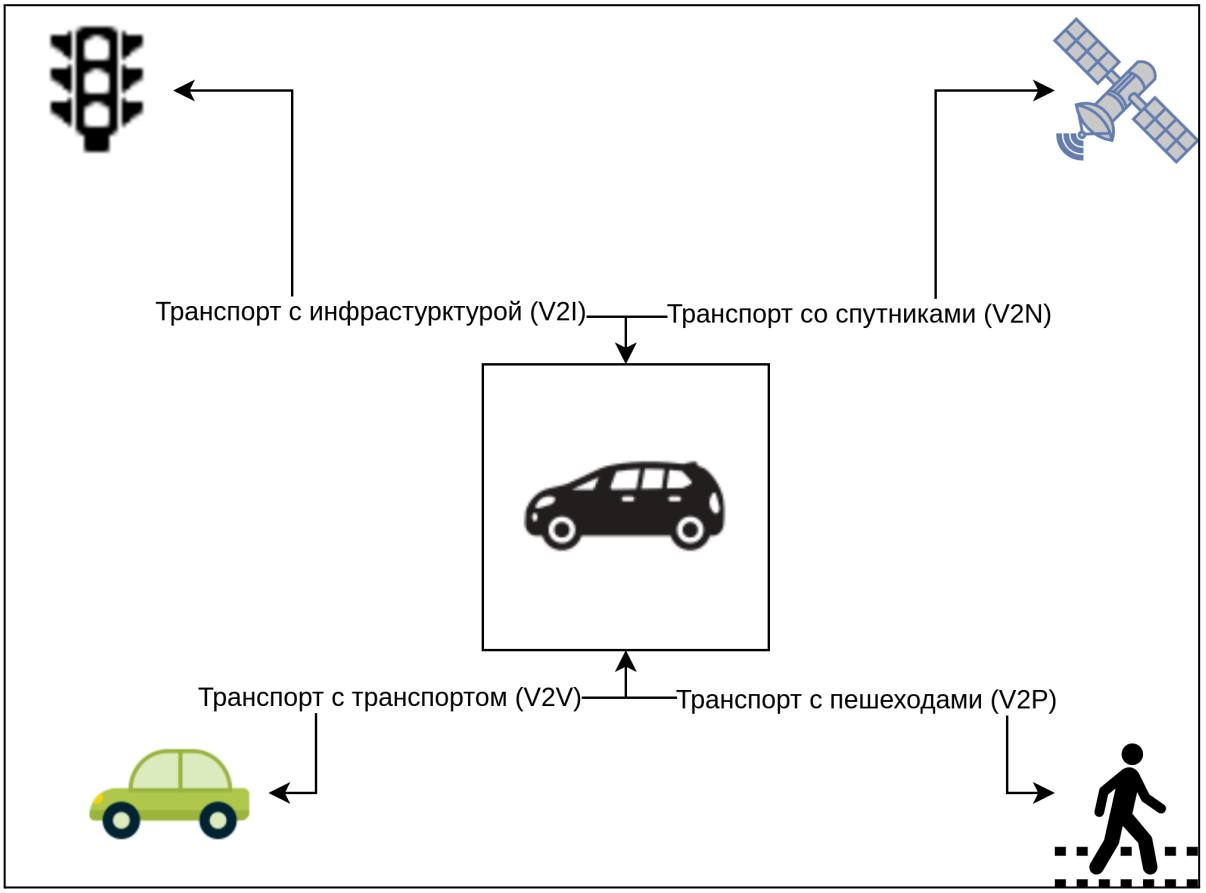

Рисунок 1.1 – Схема V2X коммуникации в области автономного транспорта

В рамках нашей задачи мы фокусируемся на сценарии, где автономные транспортные средства не имеют прямой связи с другими участниками движения. Это означает, что каждый автомобиль принимает решения, опираясь исключительно на информацию, доступную через его собственные датчики и сенсоры, без возможности напрямую получать данные о намерениях и маневрах других машин. Управляющая модель в таком случае взаимодействует с

"эго-транспортом" (лидирующим автомобилем), который оборудован различными смарт-датчиками, позволяющими собирать информацию об окружающей среде и других объектах на дороге. Обрабатывая эту информацию, автономный транспорт получает набор высокоуровневых признаков, на основе которых принимает решения.

Для разработки такого вида автономного транспорта могут использоваться различные наборы и комбинации смарт-датчиков. Например, среда симуляции CARLA [4], которая является открытым стандартом в области обучения моделей управления автономным транспортом, предоставляет следующие датчика для выбора:

– детектор столкновений;
– глубинная камера;
– датчик GNSS;
– датчик IMU;
– детектор вторжения в полосу движения;
– датчик LIDAR;
– детектор препятствий;
– радарный датчик;
– RGB-камера;
– RSS-датчик;
– семантический датчик LIDAR;
– камера семантической сегментации;
– камера для сегментации объектов;
– DVS-камера;
– камера оптического потока;
– датчик V2X (абстракция над V2X сетями).

Для решения разных задач автономного управления транспортом требуются соответственно и разные датчики, поэтому итоговая система оснащается заданным необходимым набором (рисунок 1.2).

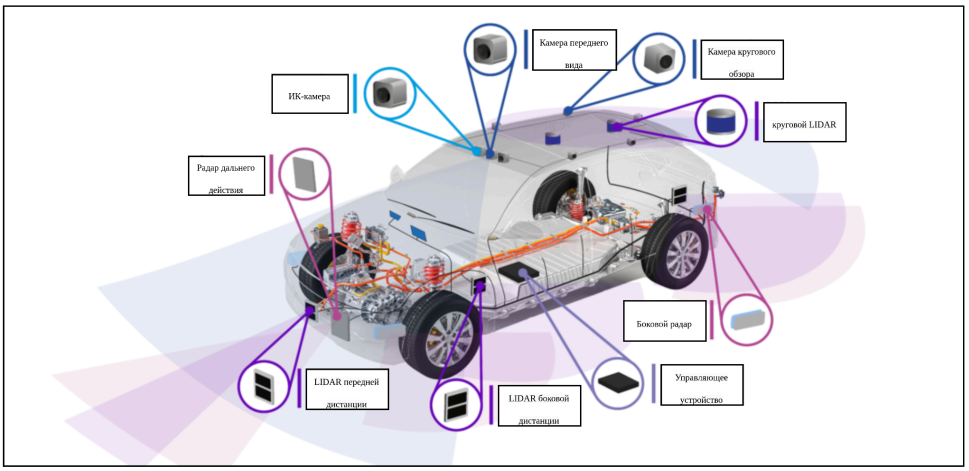

Рисунок 1.2 – Модель автономного транспорта со смарт-датчиками

Таким образом, задача автономного управления дорожным транспортом сводится к разработке алгоритмов, способных эффективно интерпретировать данные, поступающие от собственных сенсоров транспортного средства. Это требует формирования управляющей модели, опирающейся на высокоуровневые признаки окружающей среды, извлеченные из показаний таких датчиков, как камеры, лидары, радары и другие.

1.2 Подходы к автономному управлению

Для решения задачи автономного управления необходимо связать набор данных от датчиков с управляющим решением. Существует два основных подхода: непрерывный и модульный (рисунок 1.3).

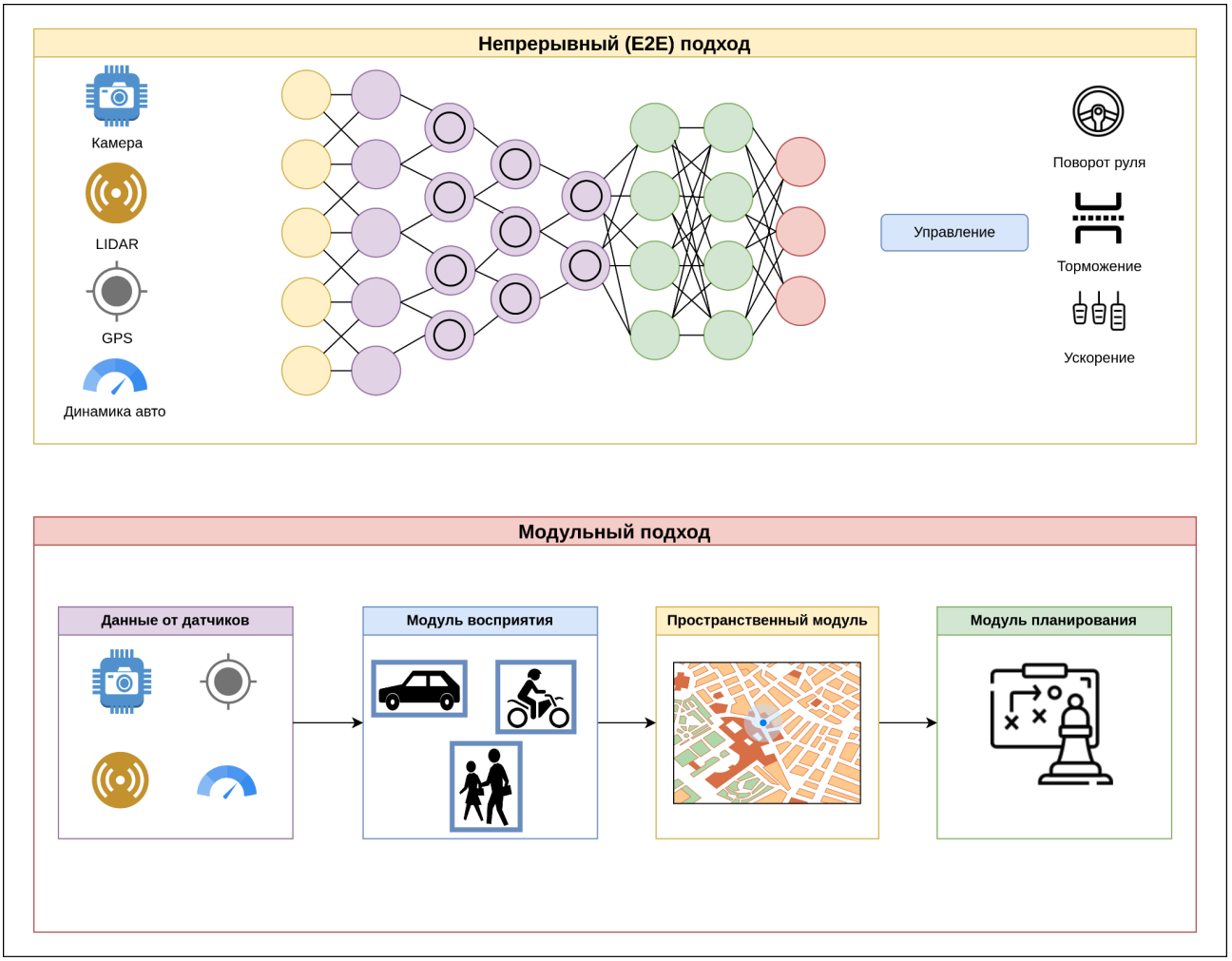

Рисунок 1.3 – Сравнение непрерывного и модульного подходов

Модульный подход предполагает разделение пути от данных к решению на этапы, или так называемые модули. Например модули:

– “восприятия”, который отвечает за распознавание окружающих объектов на основе данных от камер, датчиков LiDAR и методов сегментации;

– локализации, определяющий локальное и глобальное местоположение транспортного средства;

– планирования, задающий долгосрочные и краткосрочные цели передвижения.

Разделение на модули может отличаться и в конечном счете определяет точность и производительность модели, принимающей решение.

Непрерывный подход же предполагает принятие решение на основе непосредственно данных от датчиков. Такой подход становится все более и более актуальным за счет развития технологий машинного обучения.

На практике в интеллектуальных системах управления автономным транспортом непрерывный и модульный подходы комбинируются, в результате чего нейросетевые алгоритмы непрерывного подхода принимают решение на основании не только данных от датчиков, но и результатов деятельности отдельных модулей системы – такой подход называется гибридным.

В рамках данной работы будут рассмотрены, реализованы и проанализированы нейросетевые алгоритмы непрерывного подхода с использованием преобразованных данных (скорость и расположение транспортных средств, расстояние до ближайших автомобилей) вместо необработанных данных от датчиков, что позволит сконцентрироваться на развитии наиболее ключевого компонента системы, абстрагируясь от задачи преобразования данных.

Таким образом, существует модульный, непрерывный и гибридный подход к решению задачи автономного управления. В рамках работы будет решена задача принятия управляющего решения на основе преобразованных данных, что соответствует гибридному подходу, применяемому в прикладной области на практике.

1.3 Автономное управление с использованием машинного обучения

Принятие управляющего решения в рамках выбранного подхода предполагает использование методов машинного обучения. Наиболее актуальным методом в области автономного управления дорожным транспортом является машинное обучение с подкреплением (RL – Reinforcement Learning).

В отличие от классического машинного обучения с учителем, данный подход не предполагает наличие обучающей выборки, вместо этого обучаемый агент взаимодействует с, так называемой, средой, которая моделирует процесс в рамках прикладной области. Взаимодействуя со средой, агент изменяет ее состояние и получает обратную связь – вознаграждение, на основании которого и происходит обучение оптимальной модели.

В задаче автономного управления дорожным транспортом агентом является автономный транспорт, а средой – дорожная обстановка, представленная в виде множества наблюдений, которые могут быть

представлены данными от датчиков, преобразованными данными от различных модулей транспорта или данными от систем V2X коммуникации.

На практике создание автономного транспорта предполагает реализацию всех этапов формирования данных для машинного обучения – фильтрацию, первичную обработку и преобразование данных от различных датчиков в более высокоуровневое представление.

В свою очередь данная работа предполагает разработку модели симуляции дорожного движения, что позволяет опустить этапы обработки данных и работать с уже преобразованными. Такой подход, как было сказано выше, позволяет сконцентрироваться на ключевой проблеме – принятии управляющего решения.

Используя модель симуляции, можно обучать автономных агентов в реалистичных условиях, с быстрыми итерациями и контролируемой средой, что позволяет сформировать фундамент для разработки автономного агента, способного работать в реальных дорожных условиях.

Одной из сложностей применения машинного обучения с подкреплением для решения задачи автономного управления при этом является необходимость к генерализации обученной модели. Так классические алгоритмы способны решать поставленную задачу в рамках заданной среды, при этом изменения в контексте этой среды негативно влияют на производительность модели.

Данная проблема особенно актуальна для автономного управления дорожным транспортом, так как реальные дорожные условия и обстановка могут меняться совершенно непредсказуемо.

По этой причине, одной из задач данной работы является разработка генератора обучающих сценариев, способного моделировать сложности прикладной задачи.

Таким образом, применение обучения с подкреплением позволяет формировать поведение агента в условиях динамичной и изменчивой дорожной среды, моделируя процесс принятия решений на основе взаимодействия с окружающей обстановкой. Использование модели симуляции позволяет отказаться от трудоемкой предварительной обработки данных и сосредоточиться на самой задаче обучения. При этом ключевым вызовом остается обеспечение способности модели к обобщению, что делает актуальной задачу генерации разнообразных и реалистичных обучающих ситуаций. Такой подход закладывает основу для создания устойчивых и адаптивных систем автономного управления, способных функционировать в условиях реального мира.

1.4 Постановка задачи и схема реализации

Сформулируем постановку задачи. Создание интеллектуальной системы автономного управления дорожного транспорта включает в себя множество задач, включая упомянутое преобразование данных, разработку модулей планирования, задачи машинного зрения и задачу принятия управляющего решения, которую будет решена в рамках данной работы. При этом ограничим

задачу до наиболее актуальной и сложной ее составляющей – автономного управления при прохождении перекрестков, так как элементы парковки и другие подзадачи управления на практике описываются другими моделями и решаются соответственно иначе.

Тогда поставленная задача – разработать модель автономного транспорта, способную решать задачу прохождения перекрестков в рамках модели симуляции.

Для реализации поставленной задачи необходимо:

– разработать модель симуляции дорожного движения;

– создать генератор сценариев для машинного обучения с подкреплением в рамках этой модели;

– определить методику оценки производительности агента;

– реализовать выбранные алгоритмы машинного обучения с подкреплением и сравнить по этой методике.

Примером существующей модели симуляции дорожного движения является упомянутая выше среда симуляции CARLA (рисунок 1.4).

Рисунок 1.4 – Среда симуляции CARLA

При этом главная сложность при работе с CARLA – необходимость в высокой вычислительной мощности. Для моделирования близких к реальности трехмерных объектов используются на максимум возможности графических процессоров. В свою очередь машинное обучение использует графический процессор для параллельного подсчета большого количества арифметических операций при выполнении и оптимизации нейросетей. Вследствие этого нагрузка на графический процессор возрастает еще больше, что делает CARLA

доступной только для пользователей с дорогим и продвинутым компьютерным оборудованием.

Для решения данной проблемы предлагается разработать двухмерную модель симуляции, которая упростит визуальные детали и позволит работать с более высокоуровневыми представлениями окружения, фокусируясь исключительно на задаче принятия управляющего решения.

Формирование набора сценариев является центральным этапом в построении системы оценки и обучения агентов в задачах автономного управления. От качества и полноты сценарного покрытия зависит как эффективность обучения, так и способность агента к генерализации, особенно в условиях разнообразия дорожной среды и поведения других участников движения.

В рамках данной работы процесс определения набора сценариев был выполнен с учётом существующих подходов к построению RL-бенчмарков, в первую очередь — CARLA Leaderboard (рисунок 1.5), представляющего собой промышленно-ориентированную платформу для оценки автономных агентов в фотореалистичном симуляторе CARLA. Изучение структуры CARLA Leaderboard позволило выделить ряд ключевых характеристик, на которых строится реалистичная и репрезентативная система сценариев:

– наличие сценариев с различной плотностью трафика;

– вариативность погодных условий и времени суток;

– задания с различными маршрутами и приоритетами;

– тестирование поведенческой устойчивости при взаимодействии с другими участниками движения (пешеходами, нарушителями, агрессивными водителями);

– оценки производительности, учитывающие безопасность, эффективность и соблюдение ПДД.

Рисунок 1.5 – Схема набора сценариев в CARLA Leaderboard

Поскольку в рамках работы среда моделирования ориентирована на упрощённые, но настраиваемые симуляции дорожного движения, требовалась адаптация подхода Leaderboard к ограниченному, но контролируемому окружению. Для этого был разработан набор сценариев, объединяющий:

– базовые шаблоны поведения и дорожной обстановки;

– персонализированные расширения, созданные вручную, с измененными параметрами конфигурации;

– единый интерфейс запуска и оценки агентов по всем задачам.

Всего бенчмарк CARLA Leaderboard содержит 21 сценарий. Среди них сценарии в категориях:

– “Потеря контроля”;

– “Движение в трафике”;

– “Магистральное движение”;

– “Объезд препятствий”;

– “Торможение и смена полос”;

– “Парковка”.

Некоторые сценарии из вышеперечисленных категорий не подходят для адаптации или переноса в разрабатываемую среду, так как требуют неоправданно много проработанных деталей в модели симуляции, например имитацию погодных условий. В то же время многие из этих сценариев с трудом вписываются в концепцию тестирования базовых алгоритмов машинного обучения с подкреплением.

В число таких сценариев входят:

– “01 - Потеря контроля без предварительных действий” (рисунок 1.6);

– “06 - Пересечение со встречными велосипедистами”;

– “11 - Уступить дорогу автомобилю скорой помощи”;

– “18 - Пешеход выходит из-за припаркованного автомобиля”.

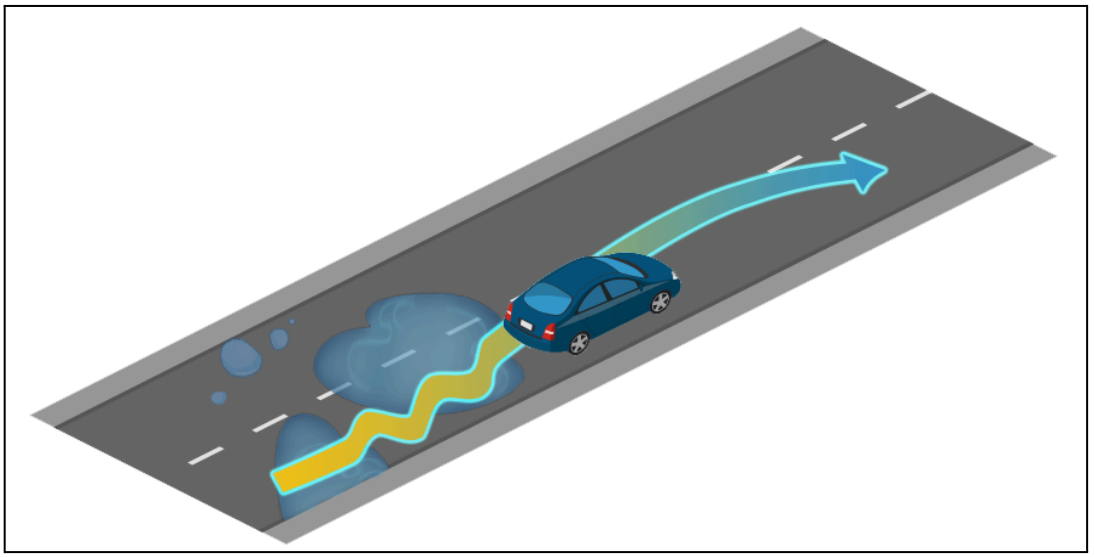

Рисунок 1.6 – Сценарий “01 - Потеря контроля без предварительных действий”

Однако многие из перечисленных сценариев могут быть заменены на аналогичные, не упрощая при этом требования к генерализации автономного агента.

Выделены эквивалентные или близкие ситуации, которые могут быть смоделированы, включая:

– “02 - Левый поворот на перекрестке со встречным движением” (рисунок 1.7);

– “03 - Правый поворот на перекрестке с попутным движением”;

– “04 - Проезд нерегулируемого перекрестка с движением автомобилей в различных направлениях”;

– “07 - Перестроение из полосы разгона в крайнюю полосу движения”;

– “08 - Движение в крайней полосе при наличии полосы разгона”;

– “09 - Пропуск перестраивающегося автомобиля”;

– “10 - Съезд с магистральной дороги”;

– “11 - Перестроение из полосы с дорожным препятствием”

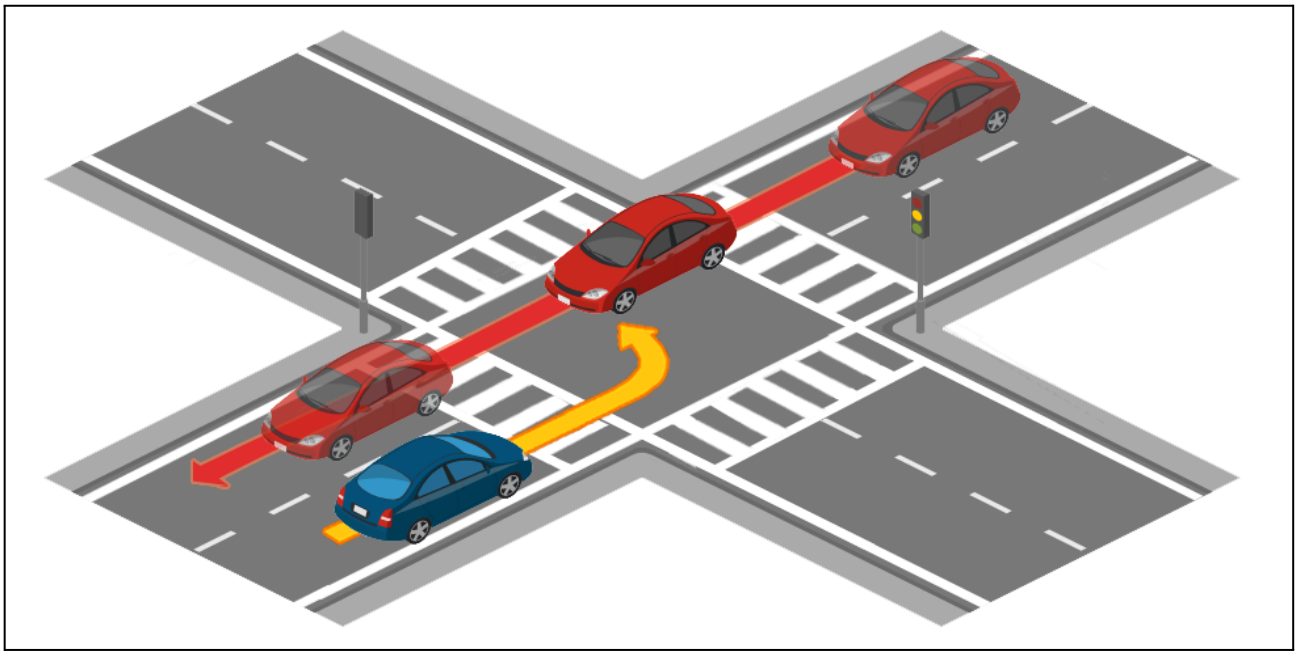

Рисунок 1.7 – Сценарий “02 - Левый поворот на перекрестке со встречным движением”

Для расширения возможностей модели симуляции были созданы:

– симуляции с нерегулярным поведением трафика;

– ситуации с плотным движением на съездах/въездах;

– "выбросы" неожиданного поведения соседних агентов (резкое торможение, нарушение дистанции).

Затем все сценарии необходимо привести к единому формату. Интегрировать их в общий RL-бенчмарк, построить систему оценки как для обучения генерализованных агентов, так и для объективной оценки модели при ее развертке. Наконец, улучшить подсчет комплексных метрик как безопасность вождения, скорость, соблюдение правил, скорость реакции и эффективность.

Такие метрики могут послужить основой для подробного анализа тестируемых моделей и подходов.

Таким образом, в рамках данной главы была обоснована необходимость разработки упрощенной, но функционально насыщенной модели симуляции для задачи автономного управления транспортом при прохождении перекрестков. В связи с высокой вычислительной сложностью среды CARLA, предложено использовать двумерную модель с акцентом на принятие управляющих решений, что позволяет сосредоточиться на ключевых аспектах поведения автономных агентов без излишней нагрузки на вычислительные ресурсы. На основе анализа платформы CARLA Leaderboard была выработана адаптированная методика построения набора сценариев, учитывающая особенности разрабатываемой среды. Отобраны и модифицированы сценарии, наиболее релевантные поставленной задаче, исключены те, реализация которых требует чрезмерно сложного моделирования или не способствует обучению обобщающего поведения. При этом обеспечено разнообразие дорожных ситуаций, включая повороты, съезды, перестроения и взаимодействие с другими участниками движения. Всё это создает основу для построения RL-бенчмарка, ориентированного на проверку и сравнение различных алгоритмов машинного обучения с подкреплением в условиях, приближенных к реальной дорожной среде.

ГЛАВА 2 МОДЕЛИ И АЛГОРИТМЫ

2.1 Общие сведения о машинном обучении с подкреплением

Модель обучения с подкреплением можно описать как агента, который взаимодействует с окружением на протяжении какого-то дискретного числа шагов t, каждый шаг выполняя действие *A* и в ответ наблюдая пару состояние окружения *S* и вознаграждение от принятого действия *R* (рисунок 2.1).

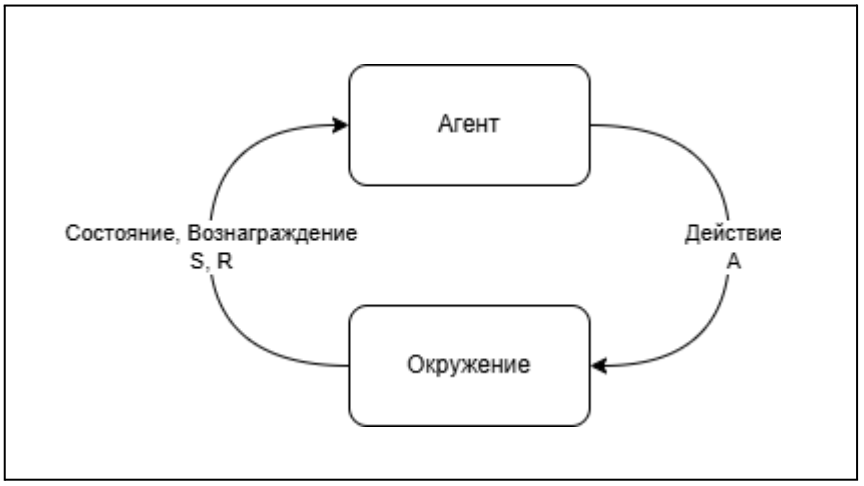

Рисунок 2.1 – Модель обучения с подкреплением

В нашем случае таким агентом является автономное транспортное средство, а окружением – модель симуляции.

Такая модель окружения абстрагирует некоторые сложности решения задачи прохождения перекрестка. В частности, она предоставляет состояние окружения в том виде, в каком его бы обрабатывала система автономного управления после решения задачи сегментации дорожной разметки и определения скорости движения соседних транспортных средств.

При этом в рамках прикладных задач, вместо состояния агент может получать от окружения его составную часть – так называемое наблюдение *O*. Это позволяет использовать одну модель окружения для решения различных задач, предоставляя агенту лишь нужный набор наблюдений. Кроме того, та или иная часть наблюдений может игнорироваться при проектировании для повышения точности, производительности или при наличии прикладных требований, например разработчик заранее знает, что будущая система автономного транспорта не сможет быть укомплектована LiDAR датчиком.

В глубоком обучении с подкреплением состояния и наблюдения обычно описываются как вектор, матрица или многомерный массив вещественных значений. Например, визуальное наблюдение с RGB-камеры можно описать как трехмерную матрицу длиной и шириной, соответствующей разрешению съемки, и глубиной, равной числу цветовых каналов.

Упомянутые действия являются элементами множества, называемого пространством действий. Такие пространства могут быть дискретными, когда число валидных действий ограничено, или непрерывными, когда действие представляет собой вектор вещественных значений.

Разница между дискретным и непрерывным пространством действий играет ключевую роль при выборе алгоритма решения, так как не каждый алгоритм разработан с целью универсально решать задачу, как для дискретных, так и для непрерывных пространств.

Принятием решения в моделях машинного обучения с подкреплением управляют стратегии. Стратегия – это правило, используемое агентом для принятия решения о том, какие действия предпринять на текущем шаге. Она может быть детерминированной и обозначаться как μ:

$$a = \mu(s) \tag{2.1}$$

или вероятностной и иметь обозначение π:

$$a \sim \pi(\cdot \mid s) \tag{2.2}$$

Наиболее сложной и важной частью обучения с подкреплением является правильное описание вознаграждения, зависящего от текущего состояния окружения, принятого действия и следующего состояния окружения:

$$r_t = R(s_t, a, s_{t+1}) \tag{2.3}$$

Хотя зачастую функция вознаграждения может зависеть и только от пары текущее состояния, принятое действие или вовсе только от текущего состояния.

Тогда нам необходимо решить задачу выбора оптимальной стратегии для максимизации функции вознаграждения. Для решения этой задачи необходимо ввести понятие траектория – последовательность состояний и действий в рамках окружения:

$$\tau = (s_0, a_0, s_1, a_1, \ldots) \tag{2.4}$$

Вероятность траектории длиной T шагов можно описать следующей формулой:

$$P(\tau|\pi) = \rho_0(s_0) \prod_{t=0}^{T-1} P(s_{t+1}|s_t, a_t)\pi(a_t, s_t) \tag{2.5}$$

При этом существуют различные методы описания эффективности модели на заданной траектории, называемые оценками. Например, конечная не дисконтированная оценка:

$$R(\tau) = \sum_{t=0}^{T} r_t \tag{2.6}$$

или бесконечная дисконтированная:

$$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t \tag{2.7}$$

где $\gamma \in [0, 1]$ – фактор дисконтирования. Тогда ожидаемую оценку при любом методе можно записать как:

$$J(\pi) = \int_{\tau} P(\tau|\pi) R(\tau) \tag{2.8}$$

Таким образом, задачу оптимизации можно формально описать как:

$$\pi^* = argmax_{\pi} J(\pi) \tag{2.9}$$

На данном этапе стоит отметить, что методы обучения с подкреплением делятся на методы с использованием модели окружения и методы без использования модели окружения. Последние, например AlphaZero [5], позволяют агенту прогнозировать результаты действий и заранее оценивать их, что повышает эффективность в работе с данными.

В нашем же случае использование модели окружения является невозможным, так как в условиях прикладной задачи невозможно точно смоделировать поведение других автомобилей на дороге.

Таким образом, модель обучения с подкреплением предоставляет универсальный формализм для построения адаптивных интеллектуальных агентов, способных принимать оптимальные решения в условиях неопределенности. В рамках рассматриваемой прикладной задачи автономного вождения такая модель позволяет абстрагировать сложность реального мира и представить взаимодействие транспортного средства с окружающей средой в виде формализованного процесса принятия решений. Введенные понятия состояний, наблюдений, действий, стратегии и функции вознаграждения служат фундаментом для построения и анализа алгоритмов, обеспечивающих обучение агента на основе опыта.

2.2 Семейства алгоритмов для оптимизации стратегии агента

В рамках алгоритмов без использования моделей окружения выделяются два ключевых подхода: оптимизация стратегии и Q-learning (оптимизация оценочных функций).

Алгоритмы первого семейства явно задают стратегию как параметрическую функцию. Они оптимизируют ее параметры напрямую, с использованием градиентного спуска по ожидаемой оценке или косвенно, максимизируя ее локальные аппроксимации.

Оптимизация обычно основана на результатах выполнения актуальной версии стратегии.

Примером таких алгоритмов являются TRPO [12] и PPO [8].

В задачах обучения с подкреплением без использования модели среды (model-free RL) агента обучают выбирать действия так, чтобы максимизировать суммарное дисконтированное вознаграждение.

Классические методы градиентного подъема политики (policy gradient) могут быть нестабильны при больших шагах обновления.

Trust Region Policy Optimization (TRPO) предложил решение этой проблемы за счёт жёсткого ограничения изменения политики в шаге (доверительной области), наложенного через ограничение на дивергенцию Кульбака–Лейблера (KL). Однако TRPO оказывается вычислительно сложным: требуется вычислять гессианы и использовать метод сопряжённых градиентов.

PPO (Proximal Policy Optimization) был предложен как более простая альтернатива TRPO. В отличие от TRPO, где используются сложные методы оптимизации второго порядка, PPO – это семейство методов первого порядка, которые достигают похожего эффекта «неотдаления» новой политики от старой, но через более простой механизм обрезки (clipping) в целевой функции.

Благодаря этому PPO существенно легче реализовать, при этом он демонстрирует сопоставимую эффективность и стабильность обучения.

Ниже подробно рассмотрим механизмы PPO: целевую функцию, использование функции преимущества, политику-клипер (clipping) и шаги оптимизации. Также упомянем о доверительной области и приведем пример применения в задачах управления движением автомобиля.

TRPO формулирует задачу оптимизации политики как максимизацию суррогатной функции полезности при ограничении на отклонение новой политики от старой. Формально, при использовании текущей политики π_{θ_k} TRPO решает задачу:

$$\begin{gathered}\max_{\theta} \ \mathbb{E}_{s,a\sim\pi_{\theta_k}}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}\hat{A}^{\pi_{\theta_k}}(s,a)\right], \\ \mathbb{E}_{s\sim\pi_{\theta_k}}\left[D_{KL}\left(\pi_{\theta_k}(\cdot|s) \parallel \pi_\theta(\cdot|s)\right)\right] \leq \delta\end{gathered} \tag{2.11}$$

где $A^{\pi}(s,a)$ – функция преимущества для политики π, KL – дивергенция Кульбака–Лейблера, а δ – заданный малый параметр. Это ограничение гарантирует «доверительный регион», внутри которого новая политика не слишком отклоняется от старой, что повышает стабильность. Однако практическая реализация TRPO требует вычисления гессианов по параметрам политики, что делает алгоритм сложным и ресурсоемким.

Из-за этих сложностей и предложили PPO – менее ресурсоемкий подход, сохраняющий ключевую идею осторожного обновления политики, но без жёсткого второго порядка метода.

PPO стремится максимизировать суммарное вознаграждение, подобно TRPO, но использует другой подход к ограничению изменения политики. Пусть π_{θ} – текущая параметрическая политика, а $\pi_{\theta_{\text{old}}}$ – её «копия» перед обновлением. В PPO вводится отношение вероятностей действий:

$$r_t(\theta) = \frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \tag{2.12}$$

которое показывает, во сколько раз обновлённая политика увеличивает (или уменьшает) вероятность выбранного действия a_t в состоянии s_t. Если $r_t(\theta)$ сильно отличается от 1, это означает, что политика изменилась слишком резко. Цель – ограничить такие резкие изменения.

Ключевой компонент PPO – использование функции преимущества. Она показывает, насколько данное действие лучше или хуже среднего ожидания. Обычно используется разность между накопленной дисконтированной наградой (return) и ценностью состояния:

$$A^{\pi}(s_t, a_t) = R_t - V^{\pi}(s_t) \tag{2.13}$$

где $R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$ – накопленная награда от t до конца эпизода, а $V^{\pi}(s_t)$ – оценка ценности состояния s_t (baseline). Такая оценка преимущества помогает убрать дисперсию градиента и понять, какие действия стоит усиливать, а какие – ослаблять.

Исходная суррогатная функция (без ограничений) напоминала бы TRPO-формулировку: $\mathbb{E}[r_t(\theta)A_t]$. Но прямая максимизация может привести к большим скачкам политики. Поэтому в PPO вводят урезанную суррогатную (clipped surrogate) функцию потерь. Наиболее популярный вариант (PPO-Clip) формулируется так:

$$L^{CLIP}(\theta) = \mathbb{E}_t\left[\min\left(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right] \tag{2.14}$$

где $\hat{A}_t$ – оценка преимущества (часто полученная по текущей политике), а $\varepsilon > 0$ – небольшой гиперпараметр.

Таким образом, если $r_t(\theta)$ выходит за эти рамки, при вычислении минимума используется усеченное значение, и дальнейшее увеличение отклонения не приносит выгоды при максимизации цели.

Например, если $\hat{A}_t > 0$, стараемся увеличивать вероятность удачных действий ($r_t > 1$), но как только $r_t(\theta)$ превысит $1 + \varepsilon$, вклад этого слагаемого перестанет расти (будет равен $(1 + \varepsilon)\hat{A}_t$). Аналогично, для $\hat{A}_t < 0$ механизм клиппинга предотвращает слишком сильное уменьшение вероятности. Благодаря этому обновления становятся более «бережными» и стабильными.

Идея в том, что цель PPO разрешает обновлять политику только в разумных пределах: «если обновление выходит за установленные рамки, функция потерь не позволит сделать слишком большой шаг, сохраняя стабильность обучения».

Политика-клипер – это именно описанный выше механизм усечения отношения $r_t(\theta)$. Он выполняет роль встроенного «ограничителя»: во время оптимизации целевой функции модель уже не получает никакой дополнительной выгоды от того, чтобы $r_t(\theta)$ выросло или упало вне диапазона $[1 - \varepsilon, 1 + \varepsilon]$. Это приближает поведение алгоритма к идее доверительного региона (области, где разрешены изменения политики) без явного введения жестких ограничений. Проще говоря, клиппинг как предохранитель «ограничивает изменения в пределах разумного»: модель усиливает хорошую политику и ослабляет плохую, но делает это мягко, без резких скачков.

Благодаря этому новый градиент указывает на движение политики, которое не выходит за установленные «доверительные границы».

В TRPO доверительная область явно задаётся ограничением на KL-дивергенцию между старой и новой политиками. В PPO концепция доверительного региона сохраняется неявно: вместо жёсткого ограничения мы просто не даем целевой функции поощрять сильные отклонения. Таким образом, PPO «держит новую политику близко к старой» тем же образом, но через механизм функции потерь, а не через решение ограниченной задачи оптимизации.

Это позволяет использовать обычный SGD/Adam-оптимизатор на параметрах и не вычислять сложные метрики. Стоит отметить, что существует и другой вариант PPO – PPO-Penalty, где в функцию потерь добавляют штрафное слагаемое $\beta, \mathrm{KL}(\pi_{\theta_{\text{old}}} | \pi_\theta)$, адаптивно подбирая β, – он приближает TRPO через штраф за изменение KL. Но наиболее распространённая реализация – именно с обрезкой, о которой говорилось выше.

Алгоритм PPO выполняется итеративно по эпохам. В каждой итерации (эпохе) действия агента генерируются с помощью текущей политики, затем политика обновляется. Условно шаги PPO можно описать так:

1. Сбор данных. Запустить текущую стохастическую политику π_{θ_k} в среде несколько раз, собрать батч траекторий (эпизодов или фиксированного числа

шагов) τ_i, содержащих состояния s_t, действия a_t и полученные вознаграждения r_t на каждом шаге.

2. Вычисление вознаграждений и преимуществ. Для каждой траектории по шагам $t = 0, \ldots, T$ вычислить накопленные дисконтированные вознаграждения (returns):

$$R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'} \tag{2.15}$$

а затем, при наличии оценочной функции ценности $V_\phi(s_t)$ (актора-критика), вычислить оценки преимущества:

$$\hat{A}_t = R_t - V(s_t) \tag{2.16}$$

Если используется расширенный подход (GAE), то $\hat{A}_t$ может рассчитываться с учетом кумулятивной ошибки прогноза.

3. Обновление политики. Оптимизировать параметры θ политики, максимизируя усеченную целевую функцию PPO-Clip. Обычно это делается методом стохастического градиентного подъёма (в реальности – спуск по отрицательной функции потерь) по мини-батчам данных. Часто на одном батче делают несколько проходов (эпох) градиентного спуска, пока не достигнут критерий остановки (например, количество итераций или достижение порога изменения KL). Итоговый шаг может выглядеть как:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|D_k|} \sum_{(s_t, a_t) \in D_k} \min\left(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \tag{2.17}$$

где $\mathcal{D}_k$ – выборка из опыта на итерации k. Таким образом, параметры θ обновляются так, чтобы увеличить вероятность действий с положительным преимуществом и уменьшить с отрицательным, но без выхода за допустимые границы.

4. Обновление value-функции (при наличии). Если используется критик (actor-critic), обновить параметры ϕ функции ценности $V_\phi(s)$, минимизируя среднеквадратичную ошибку прогноза вознаграждений (например, $\frac{1}{2}(V_\phi(s_t) - R_t)^2$ по всем собранным данным). Это служит для улучшения качества оценки преимущества в следующих итерациях.

5. Переход к следующей итерации. Скопировать новую политику в $\theta_{\text{old}} := \theta$ и повторить процесс с шагом 1. С накоплением опыта и повторными обновлениями политика постепенно улучшается. Этот цикл гарантирует последовательные, сравнительно маленькие шаги в параметрическом пространстве политики, что делает обучение стабильным.

Таким образом, Proximal Policy Optimization (PPO) – это современный метод безмодельного обучения с подкреплением, который сохраняет идею доверительной области из TRPO, но заменяет сложные ограничения механизмом усечения в целевой функции.

Благодаря этим свойствам PPO успешно применяется в задачах автономного управления (выбор траектории, регулирование скорости и др.), обеспечивая баланс между эффективностью обучения и стабильностью поведения агента.

Алгоритмы семейства Q-learning же обучают аппроксиматор $Q_\theta(s,\ a)$ для оптимальной функции ценности $Q^*(s,\ a)$. Обычно они используют целевую функцию, основанную на уравнении Беллмана. Эта оптимизация почти всегда проходит с использованием данных, собранных в любой момент обучения, независимо от того, как агент собирался исследовать среду в момент получения этих данных. Соответствующая политика получается через связь между Q^* и π^*:

$$a(s) \ = \ argmax_a Q_\theta(s,a) \tag{2.18}$$

Примерами Q-learning алгоритмов являются DQN [7] и C51 [1].

DQN – это алгоритм обучения с подкреплением, в котором функция ценности $Q(s,a)$ аппроксимируется нейронной сетью. Агент взаимодействует со средой (например, highway-env): в момент времени t он наблюдает состояние s_t, выбирает действие a_t (обычно по ε-жадной стратегии), получает вознаграждение r_t и попадает в следующее состояние s_{t+1}. Переход $\tau_t = (s_t, a_t, r_t, s_{t+1})$ сохраняется в буфере опыта (replay memory) D.

Этот буфер фиксированного размера (например, $|D| = N$) хранит последние N переходов. Для обучения регулярно берутся случайные мини-батчи (s, a, r, s') из D.

Такая схема разбивает корреляцию между последовательными наблюдениями и обеспечивает повторное использование данных. Благодаря этому каждый переход может участвовать во множестве обновлений, что повышает эффективность использования данных, а случайная выборка уменьшает дисперсию оценок градиента.

Кроме того, усреднение по буферу опыта «сглаживает» распределение данных и препятствует нежелательным осцилляциям или расходимости параметров сети.

Обучение DQN основано на итеративном обновлении в соответствии с уравнением Беллмана. Для каждого перехода (s, a, r, s') вычисляется целевое значение y (Bellman target). В случае, когда s' – терминальное состояние, принимают $y = r$. В противном случае целевое значение задается как сумма награды и дисконтированной оценки максимальной будущей ценности от s':

$$y = r + \gamma \max_{a'} Q(s', a'; \theta^-) \tag{2.19}$$

где $\gamma \in [0, 1]$ – коэффициент дисконтирования, а $Q(s', a'; \theta^-)$ – оценка целевой сети с зафиксированными параметрами θ^-. Функция потерь при обучении сети с параметрами θ вычисляется как средний квадратичный разрыв между предсказанным и целевым значением:

$$L(\theta) = \mathbb{E}_{(s,a,r,s')\sim D}\left[(y - Q(s, a; \theta))^2\right] \tag{2.20}$$

Задача оптимизации – минимизировать $L(\theta)$ по θ. На практике используется стохастический градиентный спуск по мини-батчам из D. Один шаг градиентного спуска по θ дает правило обновления параметров вида

$$\theta \leftarrow \theta - \alpha \nabla_\theta \left(y - Q(s, a; \theta)\right)^2 \tag{2.21}$$

где α – шаг обучения. Это эквивалентно модификации Q-значений по правилу $Q(s, a) \to Q(s, a) + \alpha(y - Q(s, a))$, что напоминает стандартное Q-обновление. Такой подход формально соответствует обучению с подкреплением по методу Монте-Карло с семантикой $\left(r + \gamma \max_{a'} Q(s', a'; \theta^-)\right)$ в качестве таргета.

Хранение переходов (s, a, r, s') и случайная выборка из них разрывает временную корреляцию последовательных наблюдений, что существенно снижает дисперсию градиентных оценок. Кроме того, опыт из разных эпизодов усредняется, что «сглаживает» распределение данных и препятствует осцилляциям или расходимости параметров сети.

Таким образом реплей-буфер позволяет многократно использовать один и тот же опыт, повышая эффективность обучения, и предотвращает возникновение нежелательных петлей обратной связи в обновлениях Q-функции.

Заметим, что из-за переработки опыта обучение становится off-policy, что оправдывает применение Q-обновлений, а не on-policy.

Чтобы ещё более стабилизировать обучение, DQN использует вторую копию сети с параметрами θ^-, называемую целевой сетью. Параметры этой сети периодически (например, каждые C итераций) копируются из обучаемой сети ($\theta^- \leftarrow \theta$).

При вычислении целевых значений y используется именно $Q(s', a'; \theta^-)$. Это разъединяет целевую величину и параметры сети: обновление θ не сразу влияет на цель, что предотвращает положительную обратную связь и расходимость. Как показано в оригинальной работе, фиксирование таргетов на нескольких шагах обучения добавляет задержку между изменением $Q(s, a)$ и

соответствующим изменением цели $r + \gamma \max_{a'} Q(s', a')$, что делает обучение намного более устойчивым.

Обновление параметров сети проводится стохастическим градиентным спуском по мини-батчам, отобранным из буфера опыта. На каждом шаге вычисляются градиенты $\nabla_\theta L(\theta)$ по выбранным переходам и параметры θ обновляются по правилу $\theta \leftarrow \theta - \alpha \nabla_\theta L$. После некоторого числа шагов (гиперпараметр C) выполняется обновление целевой сети: $\theta^- \leftarrow \theta$.

Как правило, помимо MSE-ошибки можно использовать усеченную Huber-функцию для повышения устойчивости, однако суть метода в том же – минимизация квадратичной ошибки между предсказанными $Q(s, a; \theta)$ и целевыми y.

Расширение C51 (Categorical DQN) основано на дистрибутивном подходе к Q-обучению. Вместо предсказания одного числа $Q(s, a)$ нейросеть выдаёт дискретное распределение вероятностей по заранее заданной сетке значений (например, по 51 «атому»). То есть модель параметризует случайную величину $Z(s, a)$, аппроксимирующую распределение вознаграждений, а значение $Q(s, a)$ вычисляется как ожидание:

$$Q(s, a) \approx \sum_{i=1}^{N} z_i \, p_i(s, a) \tag{2.22}$$

где z_i – опорные значения (атома), а $p_i(s, a)$ – предсказываемые вероятности. При обучении применяется распределенный оператор Беллмана, а обучающая функция потерь минимизирует расхождение (например, по Кульбака–Лейблеру) между проекцией таргет-распределения $r + \gamma Z(s', a^*)$ и текущим предсказанным распределением.

Такой подход в ряде задач показал более стабильное поведение и лучшее качество по сравнению с классическим DQN, поскольку учитывает не только среднее, но и дисперсию будущих вознаграждений.

Компромиссом между алгоритмами этих семейств является стабильность и сходимость. Так как первые оптимизируют непосредственно используемые стратегии, то обычно имеют более высокую стабильность. В то время как методы Q-learning могут не сходиться или иметь низкую стабильность. При этом обычно такие методы показывают более высокую эффективность за счет переиспользования данных, полученных в процессе обучения, что отличает их от большинства методом оптимизации стратегии.

Кроме того существуют гибридные алгоритмы, совмещающие в себе оба подхода, например алгоритм Advantage Actor Critic (A2C), или алгоритм актора-критика с преимуществом, и его асинхронная версия Asynchronous Advantage Actor-Critic (A3C) [3]. Актором здесь называется компонента, которая оптимизирует стратегию, а критиком — компонента, которая подсчитывает ценности состояний.

Алгоритм A2C сочетает подходы политик и оценки ценности. При обучении агент многократно взаимодействует со средой, генерируя траектории. В каждой временной метке t он получает состояние s_t, выбирает действие $a_t \sim \pi_\theta(\cdot|s_t)$ согласно текущей стохастической политике и наблюдает вознаграждение r_{t+1} и следующее состояние s_{t+1}. Собираются записи $(s_t, a_t, r_{t+1}, s_{t+1})$, а также логарифм вероятности выбранного действия $\log \pi_\theta(a_t|s_t)$ и оценка состояния критиком $V_\phi(s_t)$.

Параметры сети актора θ и критика ϕ могут разделять начальные слои (общая сверточная или полносвязная «ядро»), чтобы совместно использовать представление состояния и уменьшить число параметров.

Для каждой сохраненной серии шагов вычисляются кумулятивные (дисконтированные) вознаграждения и оценки преимущества. Пусть n – число шагов или до конца эпизода. Можно определить n-шаговые возвращенные значения (targets) для критика:

$$R_t = \sum_{k=0}^{n-1} \gamma^k r_{t+k+1} + \gamma^n V_\phi(s_{t+n}) \tag{2.23}$$

где $\gamma \in (0, 1]$ – коэффициент дисконтирования. Часто на практике берут n шагов, равных длине пакета, и используют $V_\phi(s_{t+n}) = 0$ для терминального состояния.

Оценка TD-ошибки (одношаговой разницы) на временном шаге t задается формулой

$$\delta_t = r_{t+1} + \gamma V_\phi(s_{t+1}) - V_\phi(s_t) \tag{2.24}$$

и в базовом A2C такая δ_t используется как приближённое значение преимущества:

$$A(s_t, a_t) \approx \delta_t \tag{2.25}$$

Использование преимущества вместо возврата снижает дисперсию градиента и стабилизирует обучение.

Для более точной оценки преимущества применяют обобщенную оценку преимущества (GAE). В схеме GAE вводится дополнительный коэффициент $\lambda \in [0, 1]$ и вычисляется взвешенное усреднение многошаговых TD-ошибок:

$$\hat{A}_t^{\mathrm{GAE}(\gamma,\lambda)} = \sum_{l=0}^{n-1} (\gamma\lambda)^l \, \delta_{t+l} \tag{2.26}$$

что даёт компромисс между смещением и дисперсией оценки. При $\lambda = 1$ это сходится к полному дисконтированному возврату, при $\lambda = 0$ – к одношаговому TD. В дальнейшем в формулах используется либо одношаговое $A(s_t, a_t) = \delta_t$, либо $\hat{A}_t^{\text{GAE}}$.

На основании собранных данных вычисляются функции потерь для актера и критика. Политика параметризована $\pi_\theta(a|s)$, функция ценности – $V_\phi(s)$. Обозначим оценку преимущества по выбору a_t в состоянии s_t через $\hat{A}_t$. Тогда политика обновляется по градиенту правдоподобия с учетом преимущества: цель – увеличить вероятность действий с положительным преимуществом и уменьшить – с отрицательным. Функция потерь актёра записывается как

$$L_{\text{actor}}(\theta) = -\sum_t \log \pi_\theta(a_t \mid s_t)\, \hat{A}_t + \beta\, H\big(\pi_\theta(\cdot \mid s_t)\big) \tag{2.27}$$

где в скобках – сумма по всем шагам в батче, $\hat{A}_t$ трактуется как константа при расчёте градиентов, H – энтропия политики (обычно с коэффициентом β для усиления исследования).

Критик оптимизируется по MSE между прогнозируемой ценностью и целью (дисконтированным суммарным вознаграждением R_t или $r_{t+1} + \gamma V_\phi(s_{t+1})$):

$$L_{\text{critic}}(\phi) = \sum_t \big(R_t - V_\phi(s_t)\big)^2 \approx \sum_t \big(r_{t+1} + \gamma V_\phi(s_{t+1}) - V_\phi(s_t)\big)^2 \tag{2.28}$$

В частности, можно взять $y_t = r_{t+1} + \gamma V_\phi(s_{t+1})$ в качестве цели, тогда:

$$L_{\text{critic}} = \sum_t (y_t - V_\phi(s_t))^2 \tag{2.29}$$

Общая функция потерь часто задается как сумма (или взвешенная сумма) этих двух компонент: $L_{\text{total}} = L_{\text{actor}} + c \cdot L_{\text{critic}}$, где $c \approx 0.5$ – коэффициент значимости value-loss.

Параметры сети обновляются с помощью градиентного спуска/подъёма. Вычисляется градиент функции потерь по параметрам сети. Для актёра выполняется градиентный подъем (увеличение ожидаемой награды), что эквивалентно минимизации L_{actor} (обычно через ADAM). Для критика – градиентный спуск по L_{critic}.

Таким образом, существует ряд алгоритмов машинного обучения с подкреплением для обучения автономного агента с целью управления дорожным транспортом. Каждый из этих алгоритмов может решать отдельные задачи в данной области, но, лишь разработав набор тестовых сценариев и применив упомянутые алгоритмы на нем, мы сможем судить об их пригодности

к генерализации, о наиболее оптимальных подходах к ней, а также о перспективах применения машинного обучения с подкреплением и генерализованных агентов в моделях дорожного движения.

2.3 Генерализация машинного обучении с подкреплением

В отличие от традиционных работ в RL, предполагающих совпадение тренировочных и тестовых сред, задача генерализации требует от алгоритмов способности переносить поведение на новые распределения задач — как в пределах одного типа среды (IID-генерализация), так и за его пределами (OOD-генерализация). Автономный автомобиль, обученный в симуляторе, должен демонстрировать устойчивое поведение при выходе на реальную дорогу с иными условиями — от погодных и дорожных до поведенческих паттернов других участников движения.

Каждое задание или среда в этой структуре определяется контекстом — будь то начальное состояние, параметры симуляции или конфигурация среды. Генерализация проверяется путем разделения множества контекстов на тренировочные и тестовые подмножества. В этом формализме задача сводится к "zero-shot transfer" — переносу стратегии на ранее не виденные контексты.

Для тестирования способности к генерализации в RL классифицируют существующие среды по типу варьирования параметров, структуре контекста и уровню управляемости. Важно различать среды с процедурной генерацией (например, OpenAI Procgen), где пространство контекстов неструктурировано, и среды с управляемыми параметрами (Distracting Control Suite, Meta-World), где можно тестировать более тонкие аспекты генерализации — от интерполяции до экстраполяции.

В типичных RL-бенчмарках часто создаётся неоднозначная ситуация: при обучении и оценке используются одни и те же константные настройки среды, что приводит к переобучению. Поэтому стоит задача оптимизации агента для наилучшей производительности не на одном окружении, а на их наборе.

Для формализации такой задачи существует понятие контекстный марковский процесс принятия решений (КМППР или англ. CMDP от Contextual Markov Decision Process) [14]. Эта модель представляет собой производную от классического марковского процесса принятия решений (МППР), являющегося основой машинного обучения с подкреплением.

Отличием является то, что состояние процесса описывается не только состоянием среды, но и текущим ее контекстом, где контекст зависит от каждой подзадачи в рамках одной задачи генерализации и может быть представлен как (рисунок 2.2):

– начальное состояние генератора псевдослучайных чисел (ГПСЧ) в задач с процедурной генерацией окружения, например в OpenAI Procgen [16];

– уникальный идентификатор конкретной подзадачи, например в Meta-World [15];

– вектор параметров контролируемой симуляции, например в Distracting Control Suite [17];

– комбинацией вышеперечисленного, что соответствует реализации предложенной в рамках данной работы.

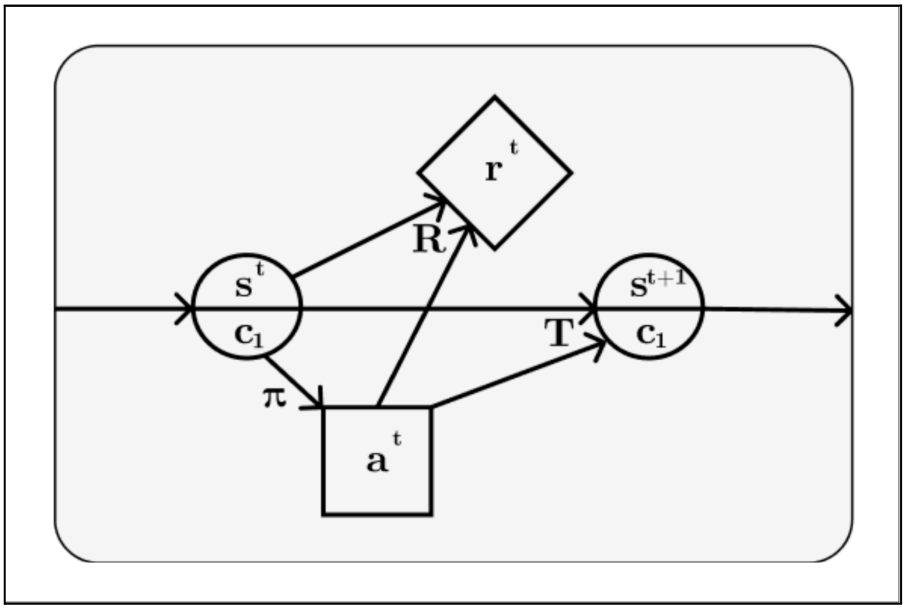

Рисунок 2.2 – Контекстный марковский процесс принятия решений

Тогда в рамках обобщенного машинного обучения с подкреплением множество таких контекстов делится на тренировочное и тестовое, что в свою очередь соответствует следующей категоризации задач (рисунок 2.3):

– одиночные окружения, для которых множество тестовых контекстов соответствует множеству тренировочных контекстов;

– IID-генерализации, для которых множество тестовых контекстов распределено так же, как множество тренировочных контекстов;

– OOD-генерализации, для которых множество тестовых контекстов не принадлежит распределению множества тестовых.

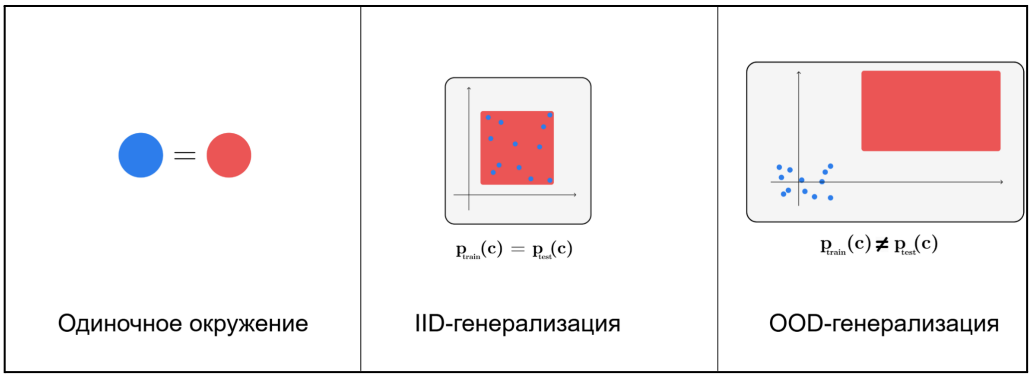

Рисунок 2.3 – Виды задач машинного обучения с подкреплением

В силу того, что обучение моделей будет проходить на разрабатываемом наборе сценариев, то задачу данной работы можно категоризировать как IID-генерализацию.

Определившись с понятием и постановкой задачи генерализации, рассмотрим подходы к ее решению. Существует три основных категории методов:

– приближение множества тренировочных контекстов к множеству тестовых;

– явное решение проблем, связанных с разницей двух множеств;

– методы специфичные для машинного обучения с подкреплением и задач оптимизации [13].

Наиболее распространенными подходами к генерализации в RL являются:

– Domain Randomization. Техника случайного варьирования физических и визуальных параметров симуляции. Метод активно применяется при переносе поведения из симуляции в реальный мир (sim-to-real). В задачах автономного вождения это может включать варьирование характеристик сенсоров, динамики автомобилей, уровней шума и т.п.;

– Regularization и Dropout. При обучении нейросетей используются регуляризаторы, предотвращающие переобучение на конкретные траектории. Также может применяться Dropout для повышения устойчивости к случайным возмущениям во входных данных;

– Обучение на латентных представлениях. В задачах с визуальным вводом используется обучение агентов в пространстве признаков, полученных из предварительно обученных автоэнкодеров или сверточных слоев. Это повышает устойчивость к изменению внешнего вида среды;

– Meta-RL (мета-обучение). Подход позволяет агенту быстро адаптироваться к новым задачам на основе ранее полученного опыта. На практике такие алгоритмы, как MAML (Model-Agnostic Meta-Learning), могут быть использованы для настройки поведения на новые дорожные ситуации без длительного переобучения;

– Curriculum Learning (поэтапное обучение). Агент обучается на задачах возрастающей сложности. Сначала он осваивает упрощённые сценарии (например, движение по прямой с одним препятствием), и только затем сталкивается со сложными многоагентными ситуациями;

– Ensemble Methods (ансамбли стратегий). Использование нескольких моделей или стратегий, каждая из которых обучалась в разных условиях, а в процессе выполнения выбирается наилучшая. Это также позволяет повысить устойчивость агента к неопределенности среды;

– Off-policy методы с переиспользованием данных. Такие алгоритмы, как DDPG [9], TD3 [10] или SAC [11], позволяют агенту обобщать поведение на основе старых данных, не зависящих от текущей стратегии, что существенно увеличивает объем доступной информации;

Таким образом, задача генерализации в обучении с подкреплением приобретает критически важное значение в приложениях, где агенты сталкиваются с постоянно меняющимися условиями, как, например, в автономном вождении. В отличие от стандартного подхода, ориентированного

на оптимизацию поведения в фиксированной среде, обобщающее RL требует устойчивости к вариациям контекста — от начальных условий до параметров симуляции и целей агента. Формализация задачи через контекстный MDP позволяет не только структурировать пространство подзадач, но и систематизировать методы оценки и сравнения стратегий. В данной работе задача классифицируется как IID-генерализация, при которой тренировочные и тестовые контексты подчиняются одному распределению. Это создает условия для систематического анализа способности алгоритмов переносить поведение, не прибегая к переобучению.

ГЛАВА 3 ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

3.1 Модель симуляции дорожного движения

Разработка модели симуляции дорожного движения должна учитывать тот факт, что симулятор будет в первую очередь использован в качестве среды для машинного обучения с подкреплением. Поэтому одной из ключевых компонент, использованных в работе, является библиотека Gymnasium, разработанная Open AI для формального описания задач обучения с подкреплением.

Взяв за основу программный пакет highway-env, представляющий собой формально описанную среду для машинного обучения на основе Open AI Gymnasium с возможностью двухмерной визуализации, разработаем конфигурацию этой среды для решения задачи с учетом требований к генерализации, которая и послужит моделью симуляции.

Среда обучения highway-env предоставляет следующие виды наблюдений:

– кинематические, таблица размером *V* на *F*, описывающая *V* автомобилей по *F* признакам, например название (с выделением эго-транспорта), точка координат *(x; y)* и вектор скорости *(vx; vy)*;

– изображение сцены в полутонах, размером *W* на *H,* где преобразование RGB в градации серого представляет собой взвешенную сумму, при этом размеры и веса регулируются параметрами настройки окружения;

– сетка занятости, размером *W* на *H* на *F*, представляющая собой дискретизацию пространства вокруг эго-автомобиля на равномерные прямоугольные ячейки, где каждая ячейка описывает характеристики выбранные в рамках конфигурации среды, а размер задается параметром разрешения сетки;

– время до столкновения, представляющее собой абстракцию над реальным датчиком столкновений в виде прогнозируемого времени до столкновения наблюдаемых транспортных средств на той же дороге, что и эго-транспорт;

– LiDAR, который делит пространство вокруг автомобиля на угловые сектора и возвращает массив с одной строкой на угловой сектор и двумя столбцами: расстоянием до ближайшего объекта (транспортного средства или препятствия) и компонентом относительной скорости объектов вдоль этого направления.

При этом, по умолчанию, среда для обучения прохождению перекрестков настроена на возврат только кинематических наблюдений, соответственно и решать задачу мы будем опираясь именно на них, как на минимально необходимый набор данных.

В среде обучения заданы следующие пространства действий:

– непрерывное: в этом пространстве заданы действия, позволяющие управлять движением, с помощью параметра разгона a и поворота δ;

– дискретное: равномерное квантование непрерывного пространства, с настраиваемым квантом;

– дискретные мета-действия: слой регулировки скорости и рулевого управления поверх непрерывного пространства.

Для решения задачи будем использовать пространство дискретных мета-действий, в котором определены следующие действия:

– перестроиться влево;

– продолжать езду;

– перестроиться вправо;

– замедлиться;

– ускориться.

Таким образом, мы получим конфигурацию среды, предполагающую дискретное пространство мета-действия для управления эго-транспортом, активную генерацию потока проезжающих автомобилей и кинематическое пространство наблюдений, состоящее из позиций и векторов скорости всех участников движения (рисунок 3.1).

```
{
    "observation": {
        "type": "Kinematics",
        "vehicles_count": 15,
        "features": ["presence", "x", "y", "vx", "vy", "cos_h",
"sin_h"],
        "features_range": {
            "x": [-100, 100],
            "y": [-100, 100],
            "vx": [-20, 20],
            "vy": [-20, 20],
        },
        "absolute": True,
        "flatten": False
    },
    "action": {
        "type": "DiscreteMetaAction",
        "longitudinal": False,
        "lateral": True
    },
    "spawn_probability": 0.6,
}
```

Рисунок 3.1 – Конфигурация среды

Функция вознаграждения спроектирована с целью сделать вождение агента безопасным и является взвешенной суммой награждения за высокую скорость и штрафа за столкновения:

$$R(s,\ a)\ =\ a\ \frac{v - v_{min}}{v_{max} - v_{min}}\ -\ b\ collision \tag{3.1}$$

Создадим соответствующую среду в рамках нашего программного модуля и интегрируем ее с python классом Monitor, который позволяет отслеживать награду и длительность каждого эпизода обучения модели.

Эти данные будут необходимы для непосредственного обучения модели в рамках выбранного алгоритма, для визуализации результатов и, наконец, для отслеживания и анализа прогресса по ходу обучения (рисунок 3.2).

```
import gymnasium as gym
import highway_env
import os
from stable_baselines3.common.monitor import Monitor

log_dir = "./logs/"
model_dir = "./model"
model_name = "a2c_intersection"
model_version = "v2"

# 1. КОНФИГУРАЦИЯ СРЕДЫ

# Создание директориев для логирования
os.makedirs(log_dir, exist_ok=True)

# Создаем среду и оборачиваем в класс Monitor
env = gym.make('intersection-v0')
env = Monitor(env, filename=os.path.join(log_dir,
"monitor_train_log.csv"))
```

Рисунок 3.2 – Создание среды обучения в обертке Monitor

Чтобы визуализировать этот прогресс воспользуемся программным средством Tensorflow Tensorboard, который предоставляет подробные данные об обучении модели в виде графиков, временных рядов и таблиц на специальной веб-странице, созданной средствами приложения командного терминала (рисунок 3.3).

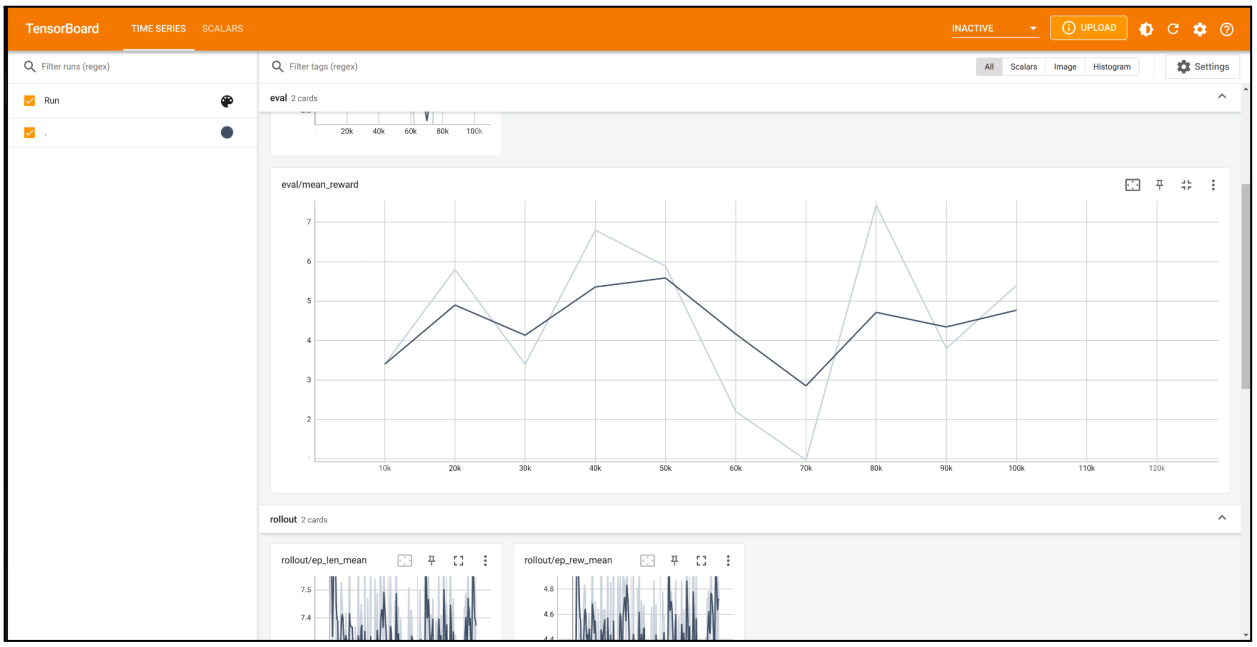

Рисунок 3.3 – График средней функции вознаграждения в TensorFlow TensorBoard

Обучение и тестирование модели может выполняться автономно на основании значений вознаграждения за эпизоды, однако в случае интеллектуальной системы автономного транспорта значения могут быть обманчивыми и требуют очень точного задания функции вознаграждения, что всегда затруднительно. Для более наглядного тестирования модели в среде предусмотрена визуализация сцены.

Рендеринг сцены в среде обучения highway-env реализован с помощью средств игрового движка Pygame. Данный движок работает с версией Python не ниже 3.5 и требует отдельной установки на персональном компьютере.

Сам движок выполнен на основе средств Simple Direct Media Layer (SDL) – кросс-платформенной библиотеки, написанной на языке C и предоставляющей абстракции для работы с мультимедиа устройствами ввода и вывода, а также графическими картами через интерфейсы OpenGL и Direct3D [2].

Как следствие установка Pygame предполагает и установку соответствующих библиотек-привязок для Python со стороны SDL.

В свою очередь среда highway-env предоставляет возможность выбора визуализации из графики в нескольких режимах: графика может быть полностью отключена, либо использоваться в режиме human (интерактивная визуализация), либо в режиме rgb array (формирование изображений для последующей обработки или сохранения). В случае обучения агентов визуализация обычно отключается для снижения нагрузки на графическую подсистему и ускорения симуляции. Однако при тестировании и анализе поведения модели предпочтительнее использовать режим human, так как он обеспечивает наглядное отображение текущей ситуации на дороге и действий агента.

Важной особенностью среды highway-env является то, что она построена на базе gymnasium, но отличается от стандартного поведения. В частности, в

стандартной реализации gymnasium видео обычно записывается покадрово при каждом вызове метода step(), что подходит для пошаговой симуляции. Однако в highway-env агент, как правило, действует на низкой частоте (например, 1 Гц), тогда как симуляция между этими действиями продолжается на более высокой частоте (например, 15 кадров на одно действие), чтобы обеспечить реалистичную физику и плавность движения.

Поэтому для корректной записи всех промежуточных кадров, соответствующих одному действию, необходимо обеспечить передачу записи кадров от симуляции к видеозаписывающему механизму среды. Это достигается специальной настройкой среды, которая позволяет записывать не только шаги агента, но и все промежуточные состояния между ними.

Так как визуализация среды использует значительный объем вычислительных ресурсов графической карты, будем использовать автономный подход на этапе обучения.

При этом, чтобы избежать последствий проблемы переобучения модели, будем сохранять модель каждую десятую часть всего обучения. Так мы сможем протестировать с визуализацией всего 10 моделей на следующем этапе, сравнить соответствующие им значения вознаграждения и выбрать лучшую. Для этого соответствующий образом настроим процесс обучения (рисунок 3.4).

```
# Сохранение модели с регулярным интервалом
checkpoint_callback = CheckpointCallback(save_freq=5000,
save_path=model_dir, name_prefix=f"{model_name}_{model_version}")

# Тестирование модели
eval_env = gym.make('intersection-v0')
eval_env = Monitor(eval_env, filename=os.path.join(log_dir,
f"monitor_eval_log_{model_version}.csv"))
eval_callback = EvalCallback(eval_env, best_model_save_path=model_dir,
log_path=log_dir, eval_freq=10000)

# Отслеживание оставшегося числа эпизодов при обучении
progress_bar_callback = ProgressBarCallback()
```

Рисунок 3.4 – Настройка процедур, вызываемых в процессе обучения

Затем на этапе тестирования, при создании соответствующей среды машинного обучения с подкреплением, выберем режим рендеринга соответствующий визуализации понятной для человека и отследим результаты каждой обученной модели (рисунок 3.5).

```
env = gym.make('intersection-v0', render_mode="human")
env = Monitor(env, filename=os.path.join(log_dir,
f"monitor_test_log_{model_version}.csv"))

# Load the trained model for evaluation
trained_model = PPO.load(os.path.join(model_dir,
f"{model_name}_{model_version}"))

# Evaluate the trained model over 10 episodes
mean_reward, std_reward = evaluate_policy(trained_model, env,
n_eval_episodes=20, render=True)
```

Рисунок 3.5 – Развертывание модели в тестовой среде

При выполнении соответствующего программного модуля, мы сможем увидеть графическое представление сцены и ее участников, обновляемое в реальном времени (рисунок 3.6).

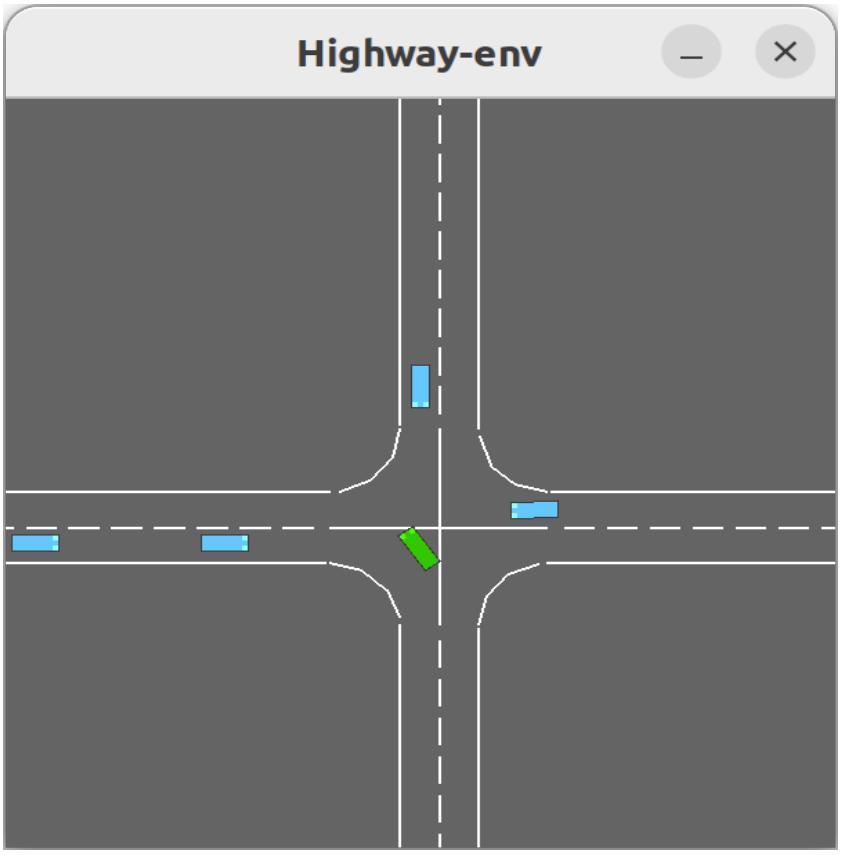

Рисунок 3.6 – Графическое представление среды симуляции

Таким образом, была представлена и реализована модель симуляции дорожного движения, адаптированная под задачи обучения с подкреплением. В основе модели лежит среда highway-env, конфигурируемая через API библиотеки Gymnasium, с акцентом на использование кинематических наблюдений как минимально достаточного и интерпретируемого входа для агентов. Управление эго-автомобилем осуществляется через дискретное пространство мета-действий, обеспечивающее баланс между простотой управления и достаточной гибкостью поведения.

Функция вознаграждения разработана с учетом необходимости безопасного и эффективного движения: она поощряет достижение высоких скоростей в пределах допустимых значений и одновременно штрафует за столкновения, тем самым формируя желательное поведение агента.

Также была реализована интеграция среды с механизмами логирования (через Monitor) и визуализации результатов обучения с использованием TensorBoard, что позволяет контролировать динамику обучения модели. Благодаря регулярному сохранению контрольных точек модели возможно проведение независимого тестирования с визуализацией поведения агента и выбор наиболее успешной политики.

Особое внимание уделено особенностям симуляции в highway-env, включая асинхронность визуализации и необходимость настройки промежуточной записи кадров, что отличает ее от стандартных реализаций на базе Gymnasium. Визуализация, реализованная через движок Pygame, используется в основном на этапе тестирования для анализа поведения модели.

Таким образом, созданная симуляционная модель не только соответствует требованиям RL-среды, но и обеспечивает инструменты для гибкой настройки, отслеживания, сохранения и анализа обучения, что делает ее пригодной для использования в задачах автономного вождения.

3.2 Программный пакет сценариев

Адаптация сценариев для создания бенчмарка предполагает объединение пространства действий и пространства наблюдений каждой среды. При этом функции вознаграждения при обучении могут задаваться индивидуально для оптимального обучения модели. В то же время при исполнении набора сценариев на конкретной модели необходимо разработать объективную и стабильную систему оценки модели по таким параметрам как эффективность, безопасность и скорость вождения.

Сама же адаптация предполагает под собой создание программного класса на языке программирования Python, наследуемого от базового в highway-env. В рамках этого класса задается дорожная топология сценария, плотность трафика, поведение окружающих агентов, уровень самостоятельности управления автомобилем для агента (путем ограничения множества дискретных мета-действий до его подмножества, например только действия перестроения или только ускорение/замедление автомобиля).

При этом каждый сценарий может вводить свой уникальный функционал, включая процедурную генерацию на основе генератора псевдослучайных чисел, например уплотненный трафик, с определенным временным интервалом появляющийся на сцене.

Такой подход позволяет создавать разнообразные задачи для автономного агента в парадигме IID-генерализации, не меняя суть задачи, но варьируя ее конфигурацию и входные данные.

Так были созданы сценарии, формирующие основу разрабатываемой среды.

Сценарий “Поворот налево со встречным и пересекающимся движением” комбинирует в себе сложности сразу нескольких сценариев набора CARLA Leaderboard (рисунок 3.7).

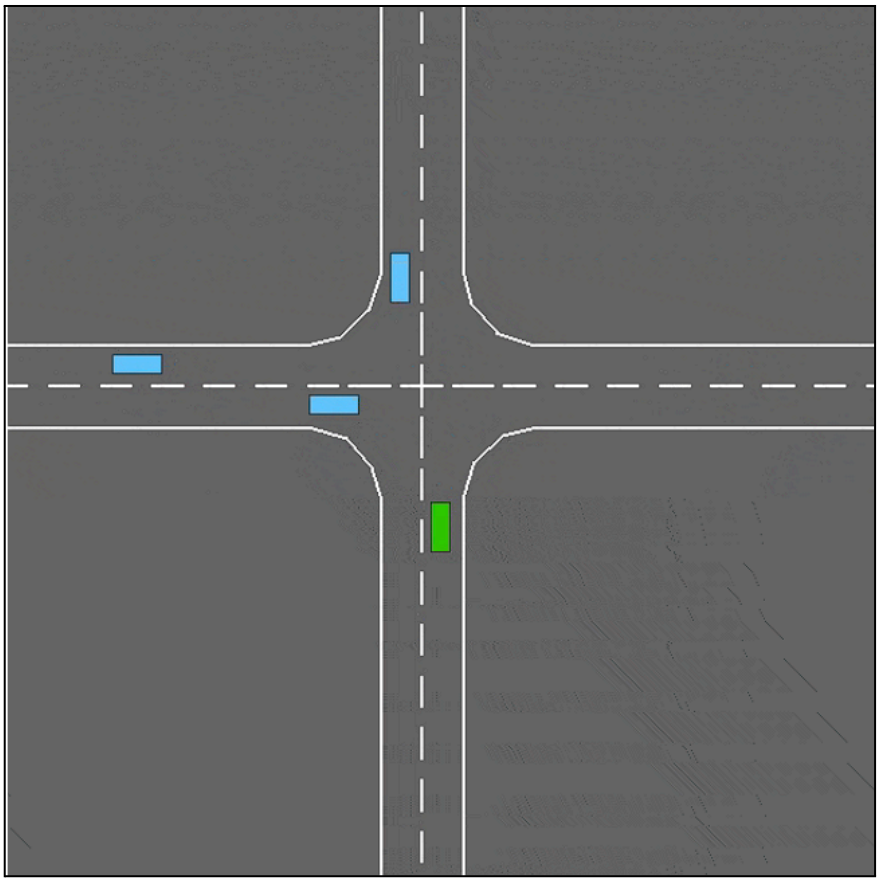

Рисунок 3.7 Сценарий “Поворот налево со встречным и пересекающимся движением”

В то же время сценарий “Круговое движение” усложняет задачу генерализации, так как усложняется форма наблюдений, доступных модели (рисунок 3.8).

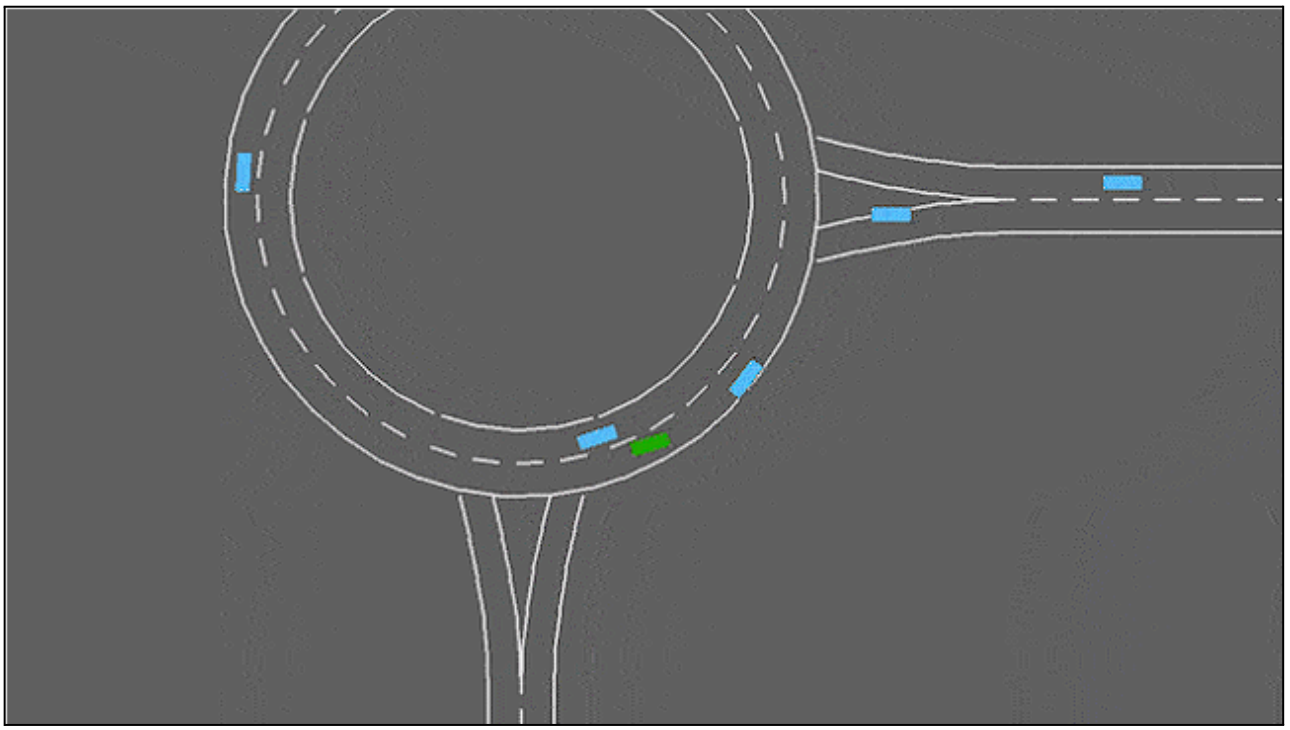

Рисунок 3.8 Сценарий “Круговое движение”

В свою очередь сценарий “Магистраль” стимулирует модель двигаться со скоростью трафика – то есть наиболее безопасной (рисунок 3.9). Адаптируясь к быстрой скорости движения соседних автомобилей, модель становится более гибкой и эффективной.

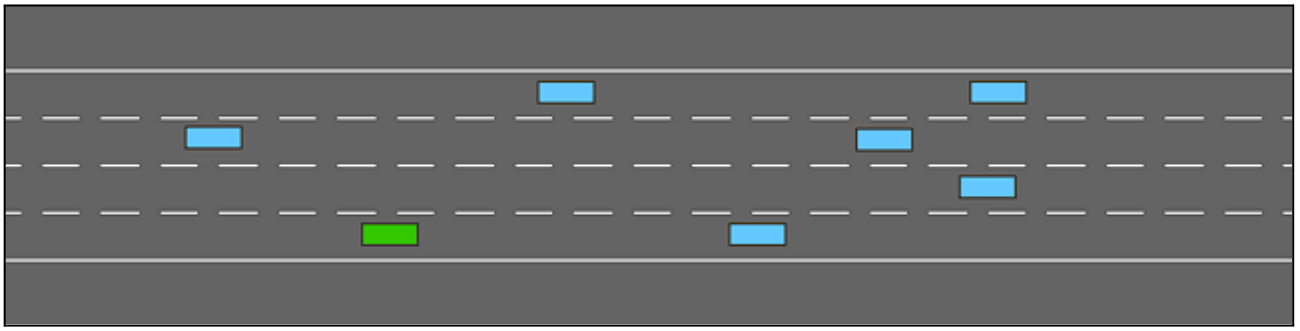

Рисунок 3.9 Сценарий “Магистраль”

Сценарий “Слияние полос” создает перед моделью сложность в виде внезапно появившегося препятствия (рисунок 3.10). Делая поведение агента на полосе разгона случайным и менее предсказуемым при обучении еще больше усложняет сценарий, однако при этом адаптирует модель к сложностям, которые могут возникнуть в реальной обстановке, что позволяет лучше оценить производительность модели и подхода в конечном итоге.

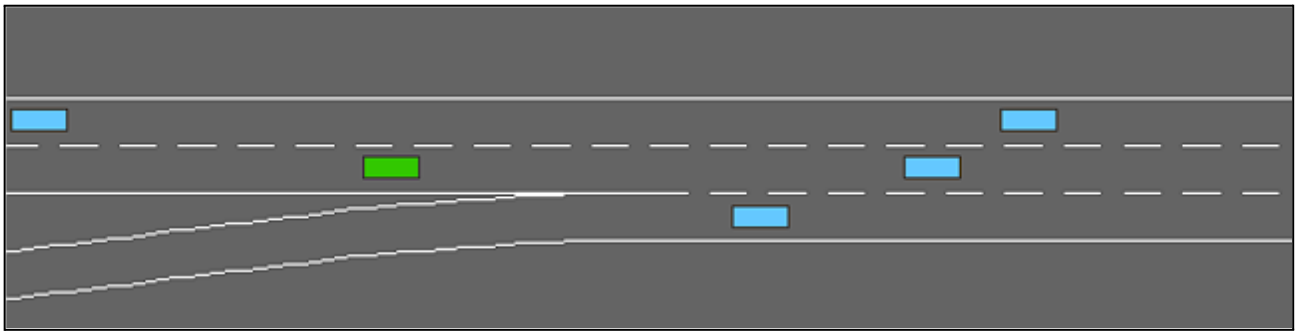

Рисунок 3.10 Сценарий “Слияние полос”

Сценарий “Проезд перекрестка прямо с пересекающимся движением” описывает классическую ситуацию дорожного движения. При этом несмотря на то, что визуально перекресток выглядит как необозначенный, движение на нем смоделировано таким образом, чтобы воссоздать сценарий проезда регулируемого перекрестка (рисунок 3.11). Это реализовано путем описания периодичности трафика с запада на восток и востока на запад. Сама периодичность же смоделирована как случайная величина с целью не дать автономному агенту переучиться на одинаковом случае.

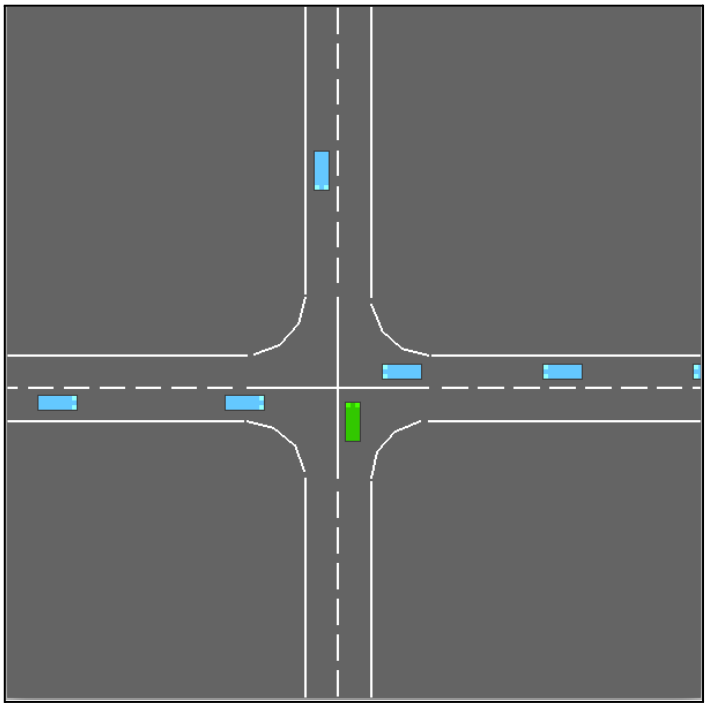

Рисунок 3.11 Сценарий “Проезд перекрестка прямо с пересекающимся движением”

Сценарий “U-образный поворот” усложняет кинематику движения путем введения поворота сложной формы. Такое усложнение позволяет проверить, насколько хорошо модель адаптирована к поворотам дороги и нелинейному движению (рисунок 3.12).

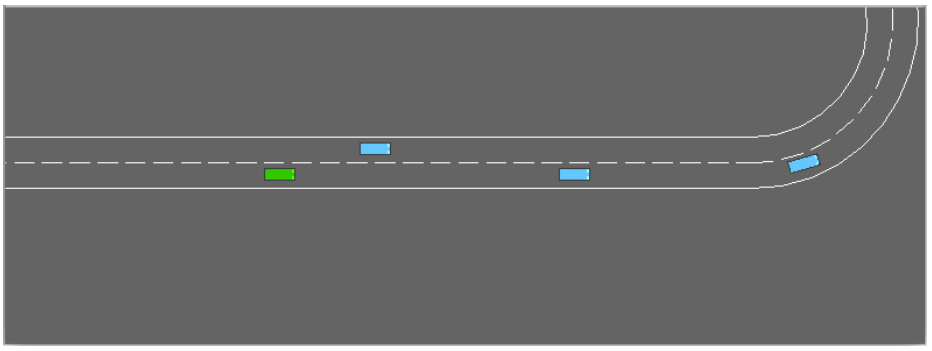

Рисунок 3.12 Сценарий “U-образный поворот”

Сценарий “Выезд со второстепенной дороги на главную” моделирует ситуацию, когда автономному агенту нужно выбрать подходящий момент, чтобы совершить поворот и встроиться в движущийся поток автомобилей (рисунок 3.13). В отличие от других сценариев не предполагается возможность наличия попутных автомобилей, тем самым усложняя задачу, если ранее автономный агент переучился следовать за другим попутным агентом.

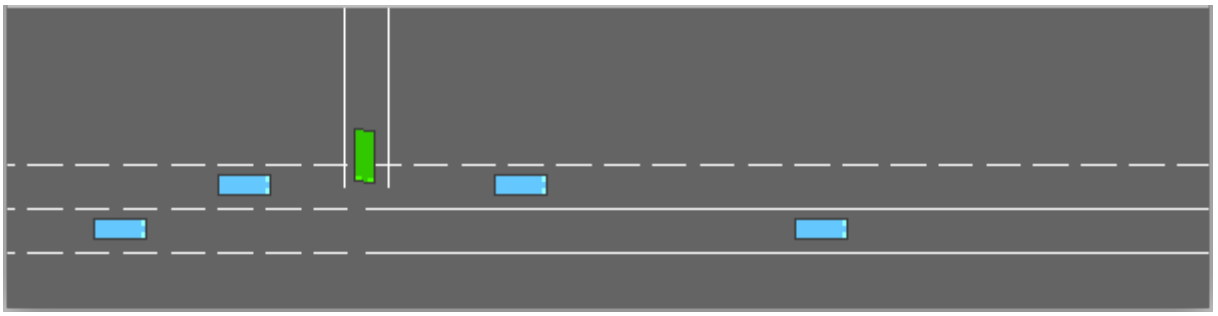

Рисунок 3.13 Сценарий “Выезд со второстепенной дороги на главную”

Таким образом, на основе реальных задач дорожного движения был сформирован набор сценариев для разработанной среды. Сценарии в наборе объединены общим пространством действий, формой наблюдений и состояний. В свою очередь их разнообразие обеспечивает сложность задачи IID-генерализации. В будущем он может улучшаться, расширяться и модифицироваться. Однако на данном этапе важно провести тестирования существующих моделей и подходов на текущем наборе, чтобы убедиться в решаемости задачи генерализации для такой среды.

3.3 Методика оценки автономного агента

По результатам выполнения сценариев программный пакет формирует итоговый отчет. Данный отчет содержит подробную информацию о результатах на каждом сценарии, включая математическое ожидание суммарного вознаграждения, его стандартное отклонение, минимальное и максимальное значение (рисунок 3.14). Кроме того, рассчитываются аналогичные показатели для такой величины как длина эпизода.

```
=== Leaderboard Evaluation Summary ===

Scenario: u-turn-v0
  Mean reward: 7.36
  Std reward:  0.91
  Min reward:  5.65
  Max reward:  8.39
  Mean episode length: 8.80
  Std episode length:  0.98
  Min episode length:  7.00
  Max episode length:  10.00

Scenario: exit-v0
  Mean reward: 0.01
  Std reward:  0.00
  Min reward:  0.01
  Max reward:  0.01
  Mean episode length: 18.00
  Std episode length:  0.00
  Min episode length:  18.00
  Max episode length:  18.00

Scenario: two-way-v0
  Mean reward: 7.56
  Std reward:  1.38
  Min reward:  5.00
  Max reward:  8.60
  Mean episode length: 12.00
  Std episode length:  1.10
  Min episode length:  11.00
  Max episode length:  14.00
```

Рисунок 3.14 – Отчет выполнения по сценариям

В свою очередь итоговая часть отчета агрегирует показатели по всем сценариям, формируя итоговый результат выполнения набора для данного автономного агента (рисунок 3.15).

```
=== Aggregated Data ===
  Overall mean reward: 9.07
  Overall std reward:  8.72
  Overall mean episode length: 15.17
  Overall std episode length:  11.20
```

Рисунок 3.15 – Итоговый отчет о выполнении всего набора сценариев

На основании таких отчетов программный пакет помогает сравнить различные подходы в машинном обучении с подкреплением для области автономного управления дорожным транспортом. Сформировав отчеты выполнения разработанного набора сценариев для существующих подходов в генерализации можно провести подробный анализ и пригодность этих методов для данной области, что в дальнейшем может послужить основой для развития и активного использования таких подходов.

Таким образом, анализ итоговых отчетов по выполнению набора сценариев предоставляет важные количественные показатели, позволяющие объективно сравнивать различные методы обучения с подкреплением. Благодаря агрегированным метрикам становится возможным не только оценить текущий уровень адаптивности и обобщающей способности агента, но и определить направления для дальнейшей оптимизации и улучшения используемых алгоритмов.

3.4 Пример использования

Протестируем программный пакет на примере алгоритма Proximal Policy Optimization (PPO). На первом этапе используем набор сценариев для обучения и генерализации автономного агента (рисунок 3.16).

```
total_timesteps = 500_000
scenarios = [
    "u-turn-v0",
    "exit-v0", "two-way-v0", "intersection-v0",
    "roundabout-v0",
    "highway-v0", "merge-v0"
]

env = DummyVecEnv([lambda: MultiTaskEnv(scenarios)])
env = VecMonitor(env)

model = PPO("MlpPolicy", env, verbose=1)
model.learn(total_timesteps=total_timesteps)

model.save(os.path.join(model_dir, f"{model_name}_{model_version}"))
env.close()
```

Рисунок 3.16 – Пример обучения автономного агента

Затем выполним набор сценариев с помощью обученной модели с генерацией итогового отчета (рисунок 3.17).

```
log_dir = "../highwayIM/logs/"
model_dir = "../highwayIM/model/"
model_name = "ppo_multitask"
model_version = "v2"

trained_model = PPO.load(os.path.join(model_dir,
f"{model_name}_{model_version}"))

scenarios = [
    "u-turn-v0",
    "exit-v0", "two-way-v0", "intersection-v0", "roundabout-v0",
    "highway-v0", "merge-v0"
]

evaluation_results = evaluate_leaderboard(
    scenarios,
    trained_model,
    num_episodes=5,
    log_path=os.path.join(log_dir,
"leaderboard_evaluation_report.json")
)
```

Рисунок 3.17 – Пример загрузки и использования автономного агента для прохождения набора сценариев

В результате выполнения набора, математическое ожидание функции вознаграждения для автономного агента равно 8.96 условным единицам (рисунок 3.18).

```
=== Aggregated Data ===
  Overall mean reward: 8.96
  Overall std reward:  8.80
  Overall mean episode length: 15.23
  Overall std episode length:  11.19
```

Рисунок 3.18 – Отчет о выполнении набора сценариев агентом, обученным по алгоритму PPO

Для сравнения, у агента, обученного по тому же алгоритму и на том же количестве тренировочных попыток, но для прохождения только сценария “Проезд перекрестка прямо с пересекающимся движением”, этот показатель равен 5.52 условным единицам (рисунок 3.19).

```
=== Aggregated Data ===
  Overall mean reward: 5.52
  Overall std reward:  4.95
  Overall mean episode length: 7.69
  Overall std episode length:  5.11
```

Рисунок 3.19 – Отчет о выполнении набора сценариев агентом, обученным только на одном сценарии

При этом, в полном отчете о выполнении набора можно заметить, что одну из наиболее низких оценок агент получил в сценарии “Слияние полос” (рисунок 3.20).

```
Scenario: exit-v0
  Mean reward: 0.14
  Std reward:  0.08
  Min reward:  0.05
  Max reward:  0.25
  Mean episode length: 3.40
  Std episode length:  1.20
  Min episode length:  2.00
  Max episode length:  5.00
```

Рисунок 3.20 – Отчет о выполнении сценария “Слияние полос”

На основании полного отчета и сравнения с агентом, обученном на одном сценарии можно сделать вывод, что тестируемый алгоритм проявляет способность к генерализации, а, увеличив количество попыток на этапе обучения, можно добиться еще более высокой производительности.

Таким образом, представленный программный пакет успешно демонстрирует свою пригодность для обучения и оценки моделей с обобщающей способностью в среде автономного вождения. Повышение количества тренировочных итераций и возможное введение дополнительных сценариев в тренировочный набор являются перспективными направлениями для дальнейшего улучшения результатов.

ЗАКЛЮЧЕНИЕ

В рамках данной работы была поставлена и решена задача разработки среды тестирования генерализованных автономных агентов управления дорожным транспортом.

Основное внимание было уделено исследованию алгоритмов обучения с подкреплением и их способности к генерализации в сложных дорожных условиях. Проведен всесторонний анализ существующих подходов, выделены ключевые проблемы и разработан программный пакет, позволяющий объективно оценивать эффективность стратегий управления.

В процессе выполнения курсового проекта были получены следующие результаты:

– Сформулирована детальная постановка задачи пересечения перекрестков автономными транспортными средствами, включая моделирование и формализацию критериев оценки;

– Проведен анализ современных подходов к решению задач обучения с подкреплением в контексте управления автономным транспортом;

– Реализована и протестирована специализированная среда на основе пакета highway-env, которая позволяет моделировать сложные дорожные ситуации с динамическим взаимодействием участников;

– Визуализированы результаты работы модели с помощью Pygame, что позволило оценить эффективность ее поведения на перекрестках в симулируемой среде;

– Проведен обзор инструментов отладки и анализа, включая TensorBoard, с их помощью проведен детальный анализ процесса обучения и характеристик модели;

– Сформирована архитектура целевой системы, демонстрирующая возможности использования RL-алгоритмов для управления автономным транспортом;

– Сформирован фундамент для создания полноценного бенчмарка оценки генерализованных агентов;

– В результате тестирования получена модель, которая демонстрирует высокую устойчивость и способность к безопасному пересечению перекрестков в симулируемых дорожных условиях.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Marc G. Bellemare, Will Dabney, and Rémi Munos. A Distributional Perspective on Reinforcement Learning – In Proceedings of the 34th International Conference on Machine Learning – Volume 70, 2017.

2. Ian Millington and John Funge. Artificial Intelligence for Games – Morgan Kaufmann Publishers Inc., 2009.

3. Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Tim Harley, Timothy P. Lillicrap, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In Proceedings of the 33rd International Conference on Machine Learning – Volume 48, 2016.

4. Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, Vladlen Koltun. CARLA: An Open Urban Driving Simulator – Conference on Robot Learning, 2017.

5. Silver David, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, L. Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan and Demis Hassabis. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm – 2017.

6. Brockman, Greg Cheung, Vicki Pettersson, Ludwig Schneider, Jonas Schulman, John Tang, Jie Zaremba, Wojciech. OpenAI Gym – 2016.

7. Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, Martin Riedmiller. Playing Atari with Deep Reinforcement Learning – 2017.

8. John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov. Proximal Policy Optimization Algorithms – 2017.

9. Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, Daan Wierstra. Continuous control with deep reinforcement learning – 2015.

10. Scott Fujimoto, Herke van Hoof, David Meger. Addressing Function Approximation Error in Actor-Critic Methods – 2018.

11. Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, Sergey Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor – 2018.

12. John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, Pieter Abbeel. Trust Region Policy Optimization – 2015.

13. Robert Kirk, Amy Zhang, Edward Grefenstette, Tim Rocktäschel. A Survey of Zero-shot Generalisation in Deep Reinforcement Learning – 2023.

14. Assaf Hallak, Dotan Di Castro, Shie Mannor. Contextual Markov Decision Processes – 2015.

15. Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Avnish Narayan, Hayden Shively, Adithya Bellathur, Karol Hausman, Chelsea Finn, Sergey Levine. Meta-World: A Benchmark and Evaluation for Multi-Task and Meta Reinforcement Learning – 2015.

16. Karl Cobbe, Christopher Hesse, Jacob Hilton, John Schulman. Leveraging Procedural Generation to Benchmark Reinforcement Learning – 2016.

17. Austin Stone, Oscar Ramirez, Kurt Konolige, Rico Jonschkowski. The Distracting Control Suite: A Challenging Benchmark for Reinforcement Learning from Pixels – 2021.