# Аннотация

**ФИО автора**: Геворкян Артём Арзикович.

**Название работы**: Модульная нейронная сеть с плавающим влиянием на вывод.

Направление: 09.03.04 Программная инженерия.

Номинация: программный документ.

Год выполнения работы: 2025 г.

Объём работы: 88 страниц.

Количество приложений (страниц): 6 (страниц — 48).

Количество иллюстраций: 9.

Количество таблиц: 10.

Количество использованных источников: 28.

Характеристика работы: данная работа имеет практическое решение в виде программного обеспечения по декомпозиции больших нейронных  
сетей на меньшие по размеру модули.

Актуальность работы: повышение эффективности обучения и поддержания сложных нейронных сетей.

Цель разработки (научной работы): проектирование и разработка модульной архитектуры для восприятия сложной динамической  
системы.

Основные результаты: упрощён процесс создания нейронных сетей под сложные среды.

Практическая ценность: новый подход к решению проблем взаимодействия со сложной системой при большом количестве компонентов среды.

Ключевые слова: модульность, сложные динамические системы, обучение с подкреплением, модульная нейронная сеть, Pytorch, Python.

# Содержание

[АННОТАЦИЯ -](#_Аннотация)

[СПИСОК ИСПОЛЬЗУЕМЫХ СОКРАЩЕНИЙ -](#_Список_используемых_сокращений)

[ВВЕДЕНИЕ -](#_Введение)

[ЛИТЕРАТУРНЫЙ ОБЗОР -](#_Литературный_обзор)

[1 ИССЛЕДОВАТЕЛЬСКИЙ РАЗДЕЛ -](#_1_Исследовательский_раздел)

[1.1 Анализ предметной области -](#_1.1_Анализ_предметной_области)

[1.2 Анализ среды обучения нейронной сети -](#_1.2_Анализ_среды_обучения_нейронной_сети)

[1.3 Постановка цели и задачи исследования -](#_1.3_Постановка_цели_и_задачи_исследования)

[1.4 Техническое задание -](#_1.4_Техническое_задание)

[1.4.1 Этапы разработки -](#_1.4.1_Этапы_разработки)

[1.4.2 Основные требования -](#_1.4.2_Основные_требования)

[1.4.3 Описание архитектуры -](#_1.4.3_Описание_архитектуры)

[1.4.4 Функциональные требования -](#_1.4.4_Функциональные_требования)

[1.4.5 Нефункциональные требования -](#_1.4.5_Нефункциональные_требования)

[2 АНАЛИТИЧЕСКИЙ РАЗДЕЛ -](#_2_Аналитический_раздел)

[2.1 Описание свойств сложной динамической системы -](#_2.1_Описание_свойств_сложной_динамической_системы)

[2.2 Анализ потоков данных -](#_2.2_Анализ_потоков_данных)

[2.3 Анализ требований к модулям -](#_2.3_Анализ_требований_к_модулям)

[3 ТЕХНОЛОГИЧЕСКИЙ РАЗДЕЛ -](#_3_Технологический_раздел)

[3.1 Реализация модулей -](#_3.1_Реализация_модулей)

[3.1.1 Виды модулей -](#_3.1.1_Виды_модулей)

[3.1.2 Реализация в коде -](#_3.1.2_Реализация_в_коде)

[3.2 Реализация нейронной сети оценки важности модулей -](#_3.2_Реализация_нейронной_сети_оценки_важности_модулей)

[3.3 Реализация связи модуля оценки важности и модулей -](#_3.3_Реализация_связи_модуля_оценки_важности_и_модулей)

[3.4 Тестирование -](#_3.4_Тестирование)

[3.5 Расчёт надёжности -](#_3.5_Расчёт_надёжности)

[3.6 Вывод по разделу -](#_3.6_Вывод_по_разделу)

[4 ЭКОНОМИЧЕСКИЙ РАЗДЕЛ -](#_4_Экономический_раздел)

[4.1 Организация и планирование работ -](#_4.1_Организация_и_планирование_работ)

[4.2 Расчёт затрат на проведение работ -](#_4.2_Расчёт_затрат_на_проведение_работ)

[4.2.1 Статья 1. Сырьё и материалы -](#_4.2.1_Статья_1._Сырьё_и_материалы)

[4.2.2 Статья 2. Основная заработная плата -](#_4.2.2_Статья_2._Основная_заработная_плата)

[4.2.3 Статья 3. Дополнительная заработная плата -](#_4.2.3_Статья_3._Дополнительная_заработная_плата)

[4.2.4 Статья 4. Страховые взносы -](#_4.2.4_Статья_4._Страховые_взносы)

[4.2.5 Статья 5. Амортизация -](#_4.2.5_Статья_5._Амортизация)

[4.2.6 Статья 6. Прочие расходы -](#_4.2.6_Статья_6._Прочие_расходы)

[4.2.7 Итог -](#_4.2.7_Итог)

[4.3 Вывод по разделу -](#_4.3_Вывод_по_разделу)

[ЗАКЛЮЧЕНИЕ -](#_Заключение)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ -](#_Список_использованных_источников)

[ПРИЛОЖЕНИЯ -](#_Приложения)

[Приложение А - Реализация модуля оценки важности -](#_Приложение_А_-_Реализация_модуля_оценки_важности)

[Приложение Б - Контроллер модулей -](#_Приложение_Б_-_Контроллер_модулей)

[Приложение В - Классы обучения нейронных сетей -](#_Приложение_В_-_Классы_обучения_нейронных_сетей)

[Приложение Г - Структуры данных, используемые системой -](#_Приложение_Г_-_Структуры_данных,_используемые_системой)

[Приложение Д - Достижения -](#_Приложение_Д_-_Достижения)

[Приложение Е - Презентационный материал -](#_Приложение_Е_-_Презентационный_материал)

# Список используемых сокращений

— **МНС** — модульная нейронная сеть;

— **МОВ** — модуль оценки важности;

— **CUDA** (Compute Unified Device Architecture) — это архитектура, которая позволяет использовать графический процессор (GPU) для повышения производительности параллельных вычислений.

# Введение

В настоящей работе решается задача оптимизации нейронных сетей путём распределения вычислений на отдельных менее затратных целостных моделях.

Решение, выдвинутое в данной работе, предусматривает возможность отключения отдельных модулей с целью уменьшения нагрузки на вычислительные мощности, а также уменьшение влияние отдельных модулей на конечный результат, что позволяет использовать нужные нейронные сети в зависимости от контекста сложной системы.

Во введении обоснована актуальность выбранной темы, определены объект и предмет исследования.

Современные большие нейронные сети используют огромное количество вычислительной мощи лишь для одного запроса, а обучаются ещё дольше. По информации на сайте Reddit, предположительное время обучения чат-бота ChatGPT-4 составляет от 3 до 4 месяцев, и с каждой модель от OpenAI это время увеличивается. Появление мощных моделей связано с улучшением архитектуры и увеличением количества весов, поэтому оптимизация уже существующих решений и нахождение путей для оптимизации ускоряет обучение нейронных сетей.

В настоящее время существуют большие нейронные сети общего назначения, вроде *ChatGPT*, *Llama*, *Mixtral*, *Claude*. Однако универсальность нейронным сетям не всегда нужна, порой обучение таких нейронных сетей может вызвать лишь создание ненужных весов, которые не имеют важного влияния на конечный результат в нейронной сети в целом, поэтому в настоящей работе предлагается разбивать одну большую нейронную сеть на некоторое количество модулей. Для связывания модулей между собой предлагается создать нейронную сеть оценки важности модулей на основе контекста окружения агента.

разработка модульной архитектуры для восприятия сложной динамической системы

Объектом исследования выпускной квалификационной работы являются модульность нейронной сети.

Предметами исследования – неглубокие нейронные сети, методы анализа и обработки значений на выходе нейронных сетей, обучение с подкреплением, градиентный спуск, сложные динамические системы.

Настоящая ВКР решает следующие современные проблемы:

— невозможность обучения нейронных сетей частично, без подключения всей нейронной сети (не включая дообучение: оно требует загрузки всей нейронной сети);

— трудности обучения больших моделей;

— трудности адаптации нейронных сетей под сложные системы.

Особенно это важно во времена написания настоящей ВКР: тенденция увеличения весов моделей для улучшения показателей нейронной сети всё ещё актуальна в настоящий момент, поэтому появление способа декомпозиции больших моделей создаст новый подход к создания нейронных сетей.

В исследовательском разделе анализируется предметная область, существующие решения, устанавливаются цели, задачи и техническое задание.

В аналитическом разделе описываются свойства сложной динамической системы, анализируется потоки данных и требования к отдельным модулям.

В технологическом разделе описывается реализация модулей, МОВ, связи модулей и МОВ и проводится тестирование решения.

В экономическом разделе описывается организация и планирование работ, расчёт затрат на проведение работа.

В Приложении Е представлен графический материал (презентация) выпускной квалификационной работы.

В процессе написания выпускной квалификационной работы руководствовался следующими нормативными актами:

1. «О защите населения и территории от чрезвычайных ситуаций природного и техногенного характера» от 21.12.1994 № 68-ФЗ.

2. «Об основах охраны здоровья граждан в Российской Федерации» от 21.11.2011 № 323-ФЗ.

3. «О гражданской обороне» от 12.02.1998 № 28-ФЗ.

4. Приказ Минздравсоцразвития РФ от 04.05.2012 № 477н «Об утверждении перечня состояний, при которых оказывается первая помощь, и перечня мероприятий по оказанию первой помощи».

5. Трудовой кодекс Российской Федерации от 30.12.2001 № 197-ФЗ (ред. 07.04.2025).

6. СанПин — 2.2.2/542-96 «Гигиенические требования к видеодисплейным терминалам, персональным электронно-вычислительным машинам и организации работы».

# Литературный обзор

В процессе изучения предметной области способствовали следующие литературные издания:

1. **Машинная теория разума** [3]. Здесь рассматривается обучение машин теории разума через нейросеть, использующую мета-обучение для предсказания психических состояний агентов. Эксперименты подтверждают эффективность подхода для многопользовательских ИИ систем.

2. **MIND-Stack для навигации** [4]. Предложен модульный стек MIND-Stack, сочетающий сеть локализации и контроллер Стэнли для навигации автомобилей. Обеспечивает интерпретируемость и дифференцируемость, снижая ошибки управления. Планируется расширение функционала для реальных автономных систем.

3. **Универсальный алгоритм модульного сложения** [5]. Гипотеза: нейросети (перцептроны, трансформеры) используют общий алгоритм — приближенную теорему об остатках. Эксперименты показывают, что для решения достаточно O(log n) признаков. Открывает путь к изучению группового умножения.

4. **Модульная нейросеть с переключением** [6]. Децентрализованный фреймворк с динамической активацией нейронов для специализации на данных. Поддерживает федеративное обучение, что полезно для распределённых и периферийных вычислений.

# 1 Исследовательский раздел

В данном разделе анализируется предметная область, существующие решения, устанавливаются цели, задачи и техническое задание.

## 1.1 Анализ предметной области

Неглубокие нейронные сети обладают малым количеством слоёв, обеспечивая высокую производительность. Они склонны к быстрому обучению, и высокую интерпретируемость по сравнению с глубокими архитектурами. Их результаты просто оценить, что повышает точность обучения. Так как настоящая работа предполагает создание простых модулей, неглубокие нейронные сети представляются лучшим выбором.

Обучение с подкреплением, как парадигма, где агент учится через взаимодействие со средой, активно использует нейронные сети для представления стратегий. Ключевой проблемой является нестабильность градиентов и переоценка Q-значений, что решается модификацией алгоритмов, например, ограничением на обновления политик. Динамические системы, являющиеся типичной областью применения обучения с подкреплением, добавляют дополнительные сложности из-за нелинейности и хаотичности поведения. Здесь неглубокие нейронные системы могут служить инструментом предсказания состояний системы или синтеза управляющих воздействий, но их эффективность зависит от способности адаптироваться к изменяющимся условиям. В настоящем решение данная проблема решается абстрагированием, вычленением компонентов системы и обучения модулей. Хотя данное решение неполное, так как полностью вычленить компонент из системы не представляется возможным, модули всё равно имеют возможность обучаться в данном контексте.

Градиентный спуск и его адаптивные варианты (Adam, RMSprop) остаются основой оптимизации параметров неглубоких нейронных сетей, но в условиях высокой дисперсии данных, характерных для обучения с подкреплением, требуют точной настройки. В динамических системах, где требуется быстрое реагирование, компромисс между скоростью и точностью становится особенно важным, что актуализирует исследование настоящей ВКР.

Таким образом, исследование лежит на пересечении машинного обучения, теории оптимизации и анализа динамических систем, предлагая подходы к повышению надежности решений на основе неглубоких сетей в условиях неопределенности при использовании обучения с подкреплением.

Сайт [7] предоставляет список самых популярных языков программирования на основе количества запросов в поисковых системах [8]. Самым популярным языком программирования на май 2025 года является Python с отрывом в ~15% [7]. Далее идут языки в следующем порядке: C++, C, Java, C# и другие. Подробный график популярности языков со временем представлен на рисунке 1.1.

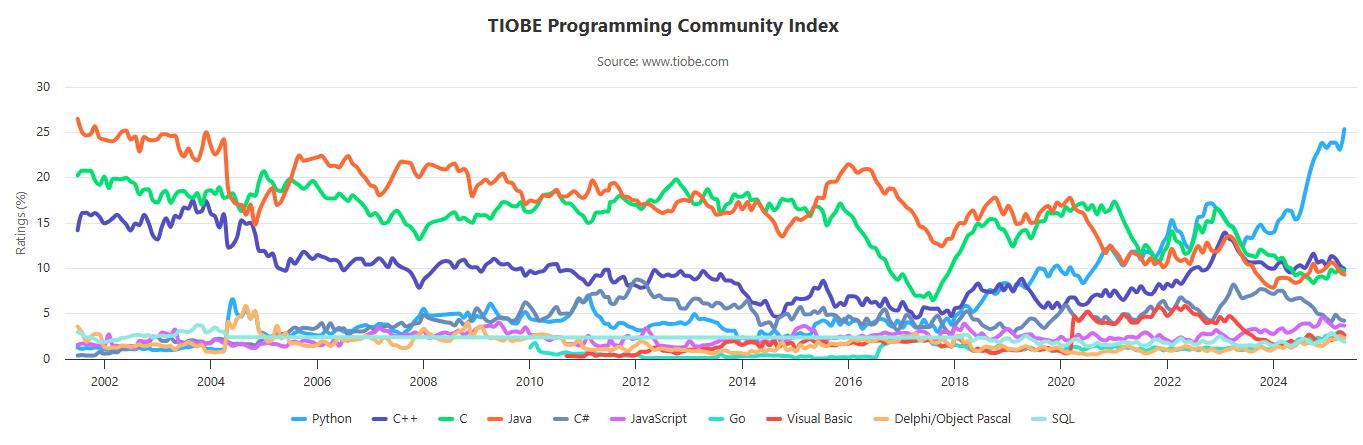


Рисунок 1.1 - Популярность языков программирования во времени

Лучшими языками для программирования являются Python, Java, C++, C# [9] [10] [11] [12]. Разберём их по-отдельности.

**Python** имеет большое количество библиотек для работы с нейронными сетями (TensorFlow, PyTorch, Keras, Scikit-learn), инструменты предобработки (NumPy, Pandas), а также обширное сообщество [13]. Также присутствие высокого уровня абстракции, динамической типизации и интерактивности позволяет просто написать код. Поддержка CUDA библиотеками PyTorch и Tensorflow позволяет обучать нейронные сети на GPU. Из минусов стоит отметить более медленная работа по сравнению с C++ и C#, однако основной код библиотек PyTorch и Tensorflow оригинально написан на компилируемых языках программирования, что даёт прирост к производительности. GIL (глобальная блокировка интерпретатора) — способ синхронизации потоков, который используется в Python. Это ограничивает параллельность многопоточности, что не позволит в данной работе проверить скорость вычислений многопоточности.

Вторым языков является **C++**. Самым главным преимуществом данного языка является производительность. C++ имеет прямой доступ к железу и оптимизация под CPU/GPU и не имеет падения производительности как в managed-языках, вроде C#, Java. Популярные библиотеки TensorFlow и PyTorch имеют C++ API, а сам язык подходит для интеграции и встраивания в высоконагруженные системы. Есть управление памятью и многопоточностью, что одновременно и преимущество и недостаток. Точная настройка памяти важна для пользовательских архитектур, однако это сильно усложняет разработку. Также недостатком C++ является меньшее количество библиотек по сравнению с Python для разработки нейронных сетей.

Третьим языком в сравнении является **C#**. Данный язык имеет фреймворк для машинной обучения ML.NET, а также биндинги (связь с C++ библиотекой) TensorFlow.NET и TorchSharp. Имеет большую производительность чем Python, но менее быстрый по сравнению с C++. Большим плюсом является хорошая интеграция с Windows. Количество библиотек намного меньше чем во всех представленных выше языках.

Итог разбора языков приведён в таблице 1.1.

Таблица 1.1 — Сравнение языков программирования для машинного обучения

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Критерий** | **Python** | **Java** | **C++** | **C#** |
| Простота | 5 | 3 | 2 | 4 |
| Производит. | 2 | 4 | 5 | 4 |
| Библиотеки | 5 | 4 | 3 | 3 |
| Глубокое обучение | 5 | 3 | 4 | 2 |
| Enterprise | 3 | 5 | 4 | 5 |
| Исследования | 5 | 2 | 1 | 2 |
| MLOps | 5 | 4 | 3 | 3 |

## 1.2 Анализ среды обучения нейронной сети

Предполагается, что модульная нейронная сеть с модулем оценки важности используется для задач, которые решить неглубокие, простые нейронные сети не в силах. К таким задачам относятся предсказание изменений среды на основе действий агента, когда система слишком сложная и/или имеет разные компоненты, которые, взаимодействуя, создают тяжело предсказуемые ситуации.

Для тонкой настройки нейронных сетей, их связи и обучения будет использоваться библиотека Torch [14]. *Torch* — библиотека для построения и обучения нейронных сетей на языке Python. Она позволяет совершать операции вроде умножения, сложения, возведения в степень, объединения, разъединения без потери градиента, что позволит обучить нейронные сети МОВ и модулей.

## 1.3 Постановка цели и задачи исследования

Целью данной выпускной квалификационной работы является разработка модульной архитектуры для восприятия сложной динамической системы

Для данной цели следует решить следующие задачи:

— написать техническое задание на модульную нейронную сеть;

— реализовать модульную нейронную сеть;

— обучить нейронные сети на примерной задаче;

— изменить или подкорректировать структуру нейронных сетей по необходимости.

## 1.4 Техническое задание

В этом разделе будут описаны назначение и цели разработки клиента как отдельной системы. Требования к интерфейсу, вид принимаемых и отправляемых данных.

### 1.4.1 Этапы разработки

1. Проектирование архитектуры — выбор и настройка архитектуры системы в целом, отдельных модулей, МОВ, слоя объединения выходов;

2. Реализация базовых модулей — написание различных модулей под конкретную среду (используемую в тестировании);

3. Разработка слоя динамического взвешивания — написание МОВ;

4. Реализация слоя объединения результатов — написание алгоритма по объединению выходов модулей на основе выхода МОВ;

5. Тестирование и отладка — тестирование системы в реальных условиях;

6. Оптимизация производительности — дообучение модулей или МОВ, доработка модулей простых алгоритмов;

7. Документирование — написание документации по использованию модульной нейронной сети.

### 1.4.2 Основные требования

В модульной архитектуре представляются требования как к отдельным модулям, так и к системе в целом:

1. **Модульность** — архитектура должна поддерживать подключение произвольного числа модулей (алгоритмов или нейронных сетей) c переобучением МОВ.

2. **Динамическое взвешивание** — для каждого входного контекста система должна определять степень влияния каждого модуля на итоговый вывод.

3. **Объединение результатов** — система должна комбинировать результаты работы модулей с учетом их весов.

4. **Масштабируемость** — архитектура должна быть легко расширяемой для добавления новых модулей.

5. **Интерпретируемость** — должна быть возможность анализа весов модулей для понимания их вклада в итоговый вывод;

### 1.4.3 Описание архитектуры

Архитектура модульной нейронной сеть с плавающим влиянием на вывод должна состоять из нескольких ключевых компонентов, которые взаимодействуют друг с другом для получения итогового результата. Рассмотрим каждый компонент подробно.

1. Входной слой: прием и предварительная обработка входных данных.

— Входные данные могут быть любого типа: изображения, тексты, числовые данные и так далее;

— Данные передаются всем подключенным модулям для дальнейшей обработки;

— При необходимости выполняется предварительная обработка данных (нормализация, токенизация, аугментация и другое).

2. Модули (подсети или алгоритмы): обработка входных данных и генерация промежуточных результатов.

— Каждый модуль представляет собой независимую/зависимую нейронную сеть или алгоритм, который специализируется на решении определенной задачи;

— Модули могут быть разнородными: например, один модуль — свёрточная нейронная сеть (CNN), другой — рекуррентная нейронная сеть (RNN), третий — алгоритм машинного обучения (например, SVM или случайный лес). Также модули могут быть обычными алгоритмами;

— Каждый модуль записывает результат в список выходов всех модулей.

3. Слой динамического взвешивания: определение важности каждого модуля для текущего контекста системы.

— На основе входных данных вычисляется вес каждого модуля, который отражает его важность для итогового результата;

— Для вычисления весов используется дополнительная нейронная сеть или алгоритм (например, полносвязная сеть);

— Веса нормализуются (например, с помощью функции softmax), чтобы их сумма равнялась 1;

— Веса могут изменяться в зависимости от контекста входных данных (например, для одних данных важнее один модуль, для других — другой).

4. Слой объединения результатов: комбинирование результатов работы модулей с учетом их весов.

— Результаты работы модулей объединяются с использованием весов, полученных на слое динамического взвешивания;

— Методы объединения могут быть разными: взвешенная сумма, конкатенация, усреднение, другие;

— После объединения может быть добавлен дополнительный слой (например, полносвязный) для финальной обработки.

5. Выходной слой: возврат итогового результата.

— Выходной слой возвращает итоговый результат в зависимости от задачи (например, класс, вероятность, регрессионное значение);

— Для классификации может использоваться функция softmax, для регрессии — линейная активация;

— Возвращает итоговый результат на основе объединенных данных.

В общем архитектура модульной нейронной сеть с плавающим влиянием на вывод предоставляет возможность объединять большой количество модулей, взвешивать их важность в контексте и объединять их выводы по определённому алгоритму.

### 1.4.4 Функциональные требования

1. **Обучение**: возможность обучения как отдельных модулей, так и всей системы в целом.

— Каждый модуль может обучаться независимо;

— Слой динамического взвешивания и слой объединения результатов обучаются совместно с использованием обратного распространения ошибки;

— Поддержка передачи градиентов от выходного слоя к модулям для их тонкой настройки.

2. **Интерфейс**: API для добавления новых модулей и анализа их вклада.

— Методы для добавления и удаления модулей;

— Методы для получения весов модулей и анализа их вклада в итоговый результат;

— Возможность сохранения и загрузки модели.

3. **Динамическое взвешивание**: веса модулей должны вычисляться на основе контекста входных данных.

— Использование дополнительной нейронной сети для вычисления весов;

— Нормализация весов с помощью функции softmax.

4. **Комбинирование весов**: результаты модулей должны комбинироваться с учетом их весов.

— Поддержка различных методов объединения (взвешенная сумма, конкатенация, усреднение);

— Возможность выбора метода объединения в зависимости от задачи.

Решение настоящей работы должно позволять обучать отдельные модули и сам МОВ.

### 1.4.5 Нефункциональные требования

1. **Производительность**:

— Минимизация времени обработки данных;

— Поддержка работы на GPU для ускорения вычислений.

2. **Масштабируемость**:

— Возможность работы с большим количеством модулей;

— Возможность сохранять МОВ с определённым количество модулей.

3. **Надежность**:

— Устойчивость к ошибкам в работе отдельных модулей.

4. **Удобство**:

— Простота в использовании конечному программисту;

— Меньшее количество действий для выполнения сложных операций;

— Возможность получить справку по функциям.

# 2 Аналитический раздел

В данном разделе описываются свойства сложной динамической системы, анализируется потоки данных и требования к отдельным модулям.

## 2.1 Описание свойств сложной динамической системы

Для того чтобы дать определение системе, нужно знать определение слова «система».

**Система** — совокупность элементов, находящихся в отношениях и связях друг с другом, которая образует определённую целостность, единство [2].

Теперь, понимая смысл слова «система», можно дать определение сложной динамической системе:

**Сложная динамическая система** — это система, в которой происходят частые и непредсказуемые изменения, которые сложно описать с помощью простых алгоритмов или математических моделей.

Сложная динамическая система отличается от обычной следующими особенностями:

— Нелинейность — небольшие изменения в начальных условиях могут приводить к большим изменениям в поведении системы;

— Многокомпонентность — система состоит из множества **взаимодействующих** элементов, которые по-разному влияют на агента;

— Непредсказуемость — поведение системы трудно предсказать в долгосрочной перспективе.

В качестве примера таких систем можно привести экономические системы: Экономика формируется взаимодействием множества агентов — потребителей, производителей, инвесторов и государственных учреждений, что делает её сложной и непредсказуемой [15].

Такие системы описываются сложными, большими нейронными сетями, но они достаточно требовательные [16], что повышает цену на каждый цикл работы нейронной сети.

Так как настоящая работа предполагает модульность, необходимо обсудить эмерджентность системы.

Эмердженти́зм (англ. emergent - неожиданно появляющийся) — совокупность теорий в философии сознания, допускающих появление у системы новых свойств, не присущих её элементам в отдельности, а также принимающих автономность и несводимость свойств системы к совокупности суммы свойств её элементов [17].

Эмерджéнтность — наличие у системы свойств целостности, то есть таких свойств, которые не присущи составляющим элементам [18].

Поведение сложной системы можно считать эмерджентным, если его нельзя заранее предсказать только на основании свойств ее частей [19]. Благодаря эмерджентности, большие языковые модели могут генерировать новые ответы, не присущие данным её обучения.

Эмерджентность в настоящей работе появляется в больших модулях, выполняющие какую-либо комплексную работу. Например, при применении модульной системы для самоуправляемых (беспилотных) автомобилей, таким комплексным модулем будет отслеживание опасностей. Алгоритмы и маленькие модели нейронных сетей не будут иметь свойство эмерджентности, что не позволит всесторонне обучить предсказание опасностей.

Также эмерджентность появляется в модуле оценки важности. Так как модуль оценки важности по своей сути управляет влиянием модулей, МОВ выполняет роль объединителя модулей, или, если упростить, контролера модулей. При соединении выводов модулей и появляется эмерджентность: хотя модули имеют разные цели, их объединение позволит системе генерировать новые действия на основе контекста среды. Например, если роботу с колёсами настроить модули для езды по ровной поверхности и по лестнице, то, езда в горку будет являться смежной областью этих двух модулей, что в некоторых случаях позволит забираться роботу в горку на основе выводов двух связанных модулей. Целью МОВ и является определение контекста для работы модулей. Таким образом, наличие модуля оценки важности, который динамически управляет влиянием модулей создаёт условия для возникновения эмерджентных свойств в рамках всей системы.

## 2.2 Анализ потоков данных

Поток данных в модульной системе представлен на Рис. 2.1.

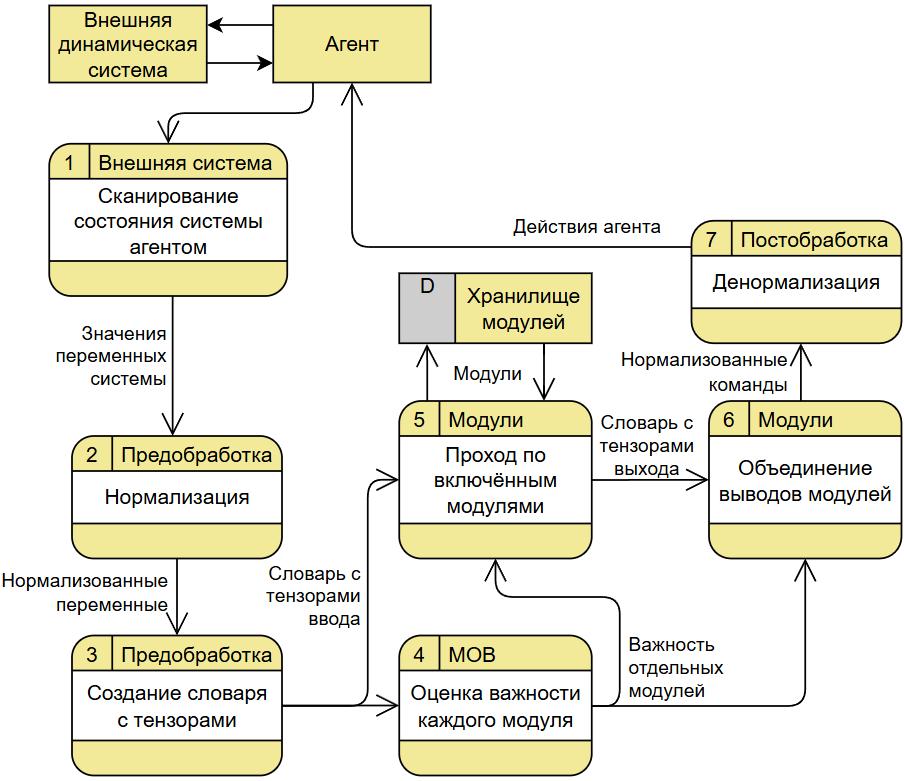


Рисунок 2.1 - Поток данных модульной системы

Предполагается, что система является динамической и сложной, посему работа в ней осуществляется через агента, который получает полное или ограниченное количество информации о среде вокруг него.

Абстрагированная схема, содержащая поток данных лишь для модулей представлено на рисунке 2.2.

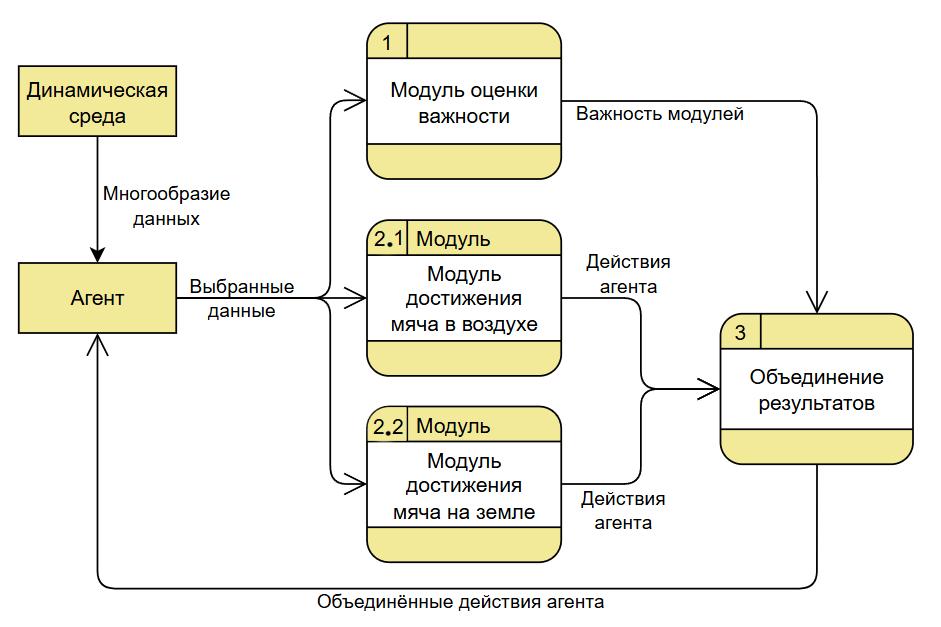


Рисунок 2.2 - Поток данных в модулях

## 2.3 Анализ требований к модулям

К модулям представляются *функциональные* и *нефункциональные* требования.

**Функциональные** требования:

— Обучение модуля как отдельной нейронной сети;

— Запуск (подключение, включение возможности получения выхода на основе входа) модуля как отдельной нейронной сети;

— Возможность временной приостановки работы модуля с его выгрузкой из памяти;

— *Совместимость* — модуль не должен конфликтовать с другими модулями;

— *Отказоустойчивость* — во время выполнения модуль не должен выдавать неестественные значения в ответ на естественные значения входа.

**Нефункциональные** требования:

— *Производительность* — низкое потребление мощностей системы;

— *Масштабируемость* — возможность улучшения модели без изменения количества/типов входов и выходов;

— *Уникальность* — у каждого модуля должна быть своя уникальная задача, которую решает модуль.

Если **каждое** из функциональных и нефункциональных требований выполнено для **каждого** модуля, тогда и только тогда обучение МОВ становится возможным.

# 3 Технологический раздел

В данном разделе описывается реализация модулей, МОВ, связи модулей и МОВ и проводится тестирование решения.

В качестве инструментов реализации работы используется язык [20]. Он позволяет быстро реализовать необходимый функционал, имеет понятную документацию и множество библиотек, упрощающие программирование.

В качестве основной библиотеки по обучению нейронных сетей используется torch [14]. Она позволяет гибко управлять тензорами без необходимости управления памяти с помощью аргумента \**requires\_grad* [21], представленный в каждой функции и тензоре torch.

Всю реализацию можно разделить на 4 этапа:

1. Реализация интерфейсов для создания отдельных модулей и их обучения.

2. Реализация МОВ.

3. Связь МОВ с системой для автоматического включение/отключение модулей во время работы.

4. Полное тестирование системы с обученными модулями.

Сохранение модульной нейронной сети происходит путём создания папки с названиями модулей, в которую сохраняются модули и МОВ. Пример форматирования представлен в таблице.

Ниже представлена реализация каждого этапа.

## 3.1 Реализация модулей

Для понимания реализации модулей, их базовых классов надо понимать какие существуют виды модулей.

### 3.1.1 Виды модулей

По принципу работы существует 2 типа модулей:

— Простые алгоритмы — выполняемый код;

— Нейронные сети (и глубокое обучение) — целые нейронные сети на Torch.

Сравнение простых алгоритмов и нейронных сетей представлен в Табл. 3.1.

Таблица 3.1 — Виды модулей по принципу работы

|  |  |  |
| --- | --- | --- |
| **Название** | **Простые алгоритмы** | **Нейронные сети** |
| Скорость подключения/отключения | O(1) | O(n) |
| Занимаемая память | O(1) | O(n) |
| Адаптивность | Отсутствует | Присутствует |
| Сложность настройки под задачу | Низкая | Высокая |
| Требуется обучение | Нет | Да |

По влиянию на конечный результат модули разделяются на два типа:

— Основные — модули напрямую влияют на вывод из модульной нейронной сети;

— Вспомогательные — результатами вывода модуля пользуются другие модули.

Возьмём в качестве примера автопилот автомобиля. Автомобиль едет по дороге, а параллельно ей идут пешеходы. Впереди пешеходный переход. Основная сеть вводит автомобиль, когда вспомогательная может просчитывать вероятность перехода. Данный подход позволяет не создавать сложнообучаемые, большие нейронные сети, а декомпозировать обучение. Более подробное описание представлено в Табл. 3.2.

Таблица 3.2 — Виды модулей по влиянию на конечный результат

|  |  |  |
| --- | --- | --- |
| **Вид модуля** | **Основной** | **Вспомогательный** |
| Напрямую влияет на вывод | Всегда | Никогда |
| Косвенно влияет на вывод | Если используется в качестве вспомогательного | Всегда |
| Входит в расчёт модулем МОВ | Всегда | Никогда |
| Включается ... | Модулем МОВ или другими модулями в качестве вспомогательного | По цепочке зависимостей основными модулями |

Модули могут как включаться и отключаться, так и загружаться и выгружаться. Изначально модуль выключен и выгружен. Если в какой-то момент модуль оценки важности считает, что модуль может понадобится, даётся запрос на загрузку модуля. Загрузка предполагает собой долгую операцию (для больших моделей нейронных сетей) по загрузке весов в оперативную память (видеопамять). Когда модуль загружен, его можно включать и отключать. Включение и отключение влияет лишь на математические операции, проводимые внутри системы: если важность модуля слишком низкая, его значениями можно пренебречь и повысить производительность системы в целом. Если модуль продолжительное время отключён, есть смысл выгрузить его из памяти для сбережения оперативной памяти. Упрощённое представление вышеописанных состояний представлено в таблице 3.3.

Таблица 3.3 — Состояния модулей

|  |  |  |
| --- | --- | --- |
|  | **Включённый** | **Выключенный** |
| Загруженный | Работает | Пока что значения данного модуля не требуются, но скоро могут понадобится, что делает выгрузку модуля нецелесообразным |
| Выгруженный | Не происходит | Для текущего состояния среды модуль не требуется |

В перспективе настоящая работа предполагает создание модульной нейронной сети с плавающим влиянием для динамический сред с количество модулей, превышающих доступную оперативную память (видеопамять) оборудования в несколько раз.

### 3.1.2 Реализация в коде

В реализации модули разделяются только по принципу работы, так как простые алгоритмы необучаемые и подключаются/отключаются за время O(1) (в виду отсутствия какой-либо загрузки/выгрузки данных из оперативной памяти или диска). Разделения на основные и вспомогательные осуществляется созданием переменной, сообщающая о зависимостях некоторого модули. При включении определённого модуля система проверяет, включены ли его зависимости.

Управление всеми модулями осуществляется контроллером, представленным на Листинг 1. В нём реализованы функции включение и отключения модулей, загрузки и выгрузки, а также функция для изменения влияния модуля.

Модули сохраняются с помощью функции *torch.save()*.

## 3.2 Реализация нейронной сети оценки важности модулей

Сам модуль оценки важности представляет собой обычную нейронную с некоторым количество линейных слоёв. В зависимости от сложности ситуации использования модулей допускается усложнение архитектуры нейронной сети МОВ.

На вход МОВ поступает состояние системы, а на выходе ожидается тензор со значениями плавающей точки от 0 до 1. Размер входа зависит от переменных, получаемой от среды, а размер выхода зависит от количества модулей, используемые в обучении/использовании.

При изменении размера данных требуется переобучать или обучать новую МОВ. Решение настоящей ВКР требует указания названия модулей для начала обучения/использования с целью различать модули и удобства использования. Пример представлен в листинге 1 ниже.

Листинг 1. Пример структуры сохранения

|  |
| --- |
| . └── reachBallAir\_reachBallFloor/  ├── mai.pt  ├── reachBallAir.pt  └── reachBallFloor.pt |

Для обучения модуля оценки важности используется алгоритм TD3 (Twin Delayed DDPG) [22].

## 3.3 Реализация связи модуля оценки важности и модулей

Прямой связи между МОВ и модулями нет. Выходом МОВ является тензор, содержащей числа с плавающей точки, преимущественно в пределах от 0 до 1. Данный тензор конвертируется в массив *numpy*, который потом используется в функции *numpy.average* для вычисления средневзвешенного значения по одной переменной действия агента. Реализация данного алгоритма приведена в Приложении Б. Формула нахождения среднего значения с весами представлена на формуле 3.1.

|  |  |
| --- | --- |
|  | (3.1) |

где v — среднее значение,

x — отдельное значение,

з — вес значения.

Если взять в качестве примера два числа 0.25 и 0.75, и уменьшать/увеличивать вес каждого соответственно, получится линейный график (см. Рис. 3.1).

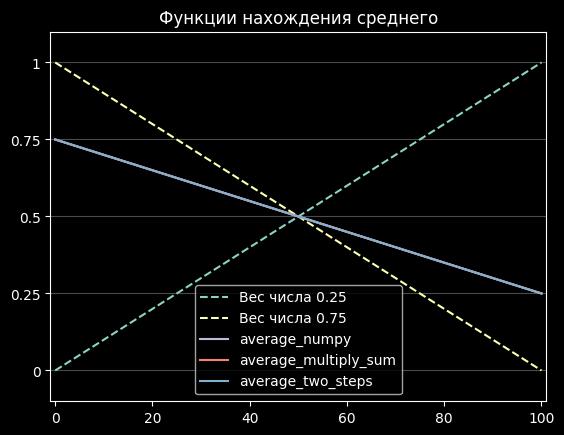


Рисунок 3.1 - Функции нахождения среднего

Функция применена в классе ModulesOutputMapping (см. Листинг 2).

- - ERROR: No function called "TODO" - -

Решение предусматривает использование разработчиками нейронных сетей, что требует адаптации кода для своей среды и агента. Для этого требуется изменить константы входов и выходов, определённые в файле со структурами данных, представленные в Приложении Г. Для корректной работы следует определить:

1. Входные данные — переопределение константы *STATE\_KEYS*. В ней описаны все переменные, которые должны быть в *ModulesOutputMapping* в начале каждой итерации.

2. Выходные данные — переопределение константы *CONTROLS\_KEYS*. В ней описаны все переменные, которые должны быть в *ModulesOutputMapping* в конце каждой итерации.

Когда входные и выходные данные определены, нужно заменить все существующие модули на необходимые для работы в новой среде. Все модули определяются в *src\mai\ai\networks*, и должны наследоваться от класса:

— **ModuleBase**, если модуль является простым алгоритмом. Требуется переопределение классового метода *inference(ModulesOutputMapping, requires\_grad: bool = False)*;

— **NNModuleBase**, если модуль является обучаемой нейронной сетью. Требуется переопределение классового метода *\_create()*.

Каждый класс содержит в себе абстрактные методы, необходимые для переопределения, включая обязательные классовые переменные *output\_types* и *input\_types*, содержащие информацию о входных и выходных данных модуля.

Пример класса представлен на листинге Листинг 2.

Листинг 2. Пример модуля

|  |
| --- |
| from mai.capnp.data\_classes import ModulesOutputMapping from mai.ai.networks.base import ModuleBase   class Module(ModuleBase):  input\_types = (  'state.car.position.x',  'state.car.position.y',  'state.car.position.z',  # 'state.car.velocity.x',  # 'state.car.velocity.y',  # 'state.car.velocity.z',  # 'state.car.rotation.pitch',  # 'state.car.rotation.roll',  # 'state.car.rotation.yaw',  # 'state.ball.position.x',  # 'state.ball.position.y',  # 'state.ball.position.z',  # 'state.ball.velocity.x',  # 'state.ball.velocity.y',  # 'state.ball.velocity.z',  # 'state.boostAmount',  )  output\_types = (  'controls.throttle',  # 'controls.steer',  'controls.pitch',  'controls.yaw',  'controls.roll',  'controls.boost',  'controls.jump',  # 'controls.handbrake',  # 'controls.dodgeVertical',  # 'controls.dodgeStrafe',  )   def load(self) -> None:  super().load()   def inference(  self,  tensor\_dict: ModulesOutputMapping,  requires\_grad: bool = False  ) -> None:  raise NotImplementedError()   def requires(self) -> set[str]:  return set() |

Разберём каждый метод по отдельности:

1. *load*() — Метод вызывается, когда важность модуля превышает пороговое значение и скоро он понадобится для работы. Операции загрузки выполняются в отдельном потоке, что позволяет функционал загрузки не влиять на главный поток. Метод может быть блокирующим.

2. *inference*(ModulesOutputMapping, bool = False) — Основной алгоритм модуля. в словаре *tensor\_dict* обязательно присутствуют значения из классовой переменной *input\_types*, а также алгоритм обязательно должен добавлять в него значения, перечисленные в *output\_types*.

3. *requires*() -> set[str] — если модуль требует выходные данные какого-то другого модуля, их названия данный метод должен вернуть.

## 3.4 Тестирование

В качестве тестирования будет использоваться среда компьютерной игры [23]. Rocket League — 3D игра сочетания аркадного футбола автомобилями и простого управления, основанных на законах физики.

Для этой игры создан плагин [24], позволяющий выполнять код C++ внутри игры. С его помощью можно, как и получать информация о среде внутри игры, так и эмулировать ввод.

Данная среда позволяет в полном объёме обучить нейронную сеть по модулям и сравнить её с существующими решениями. В ней существует возможность создавать пользовательские тренировки с разными сценариями, а также уже существуют некоторые версии ботов на основе как алгоритмов, так и нейронных сетей, обученных с помощью обучения с подкреплением.

Поток данных в приложении представлен на Рис. 3.2.

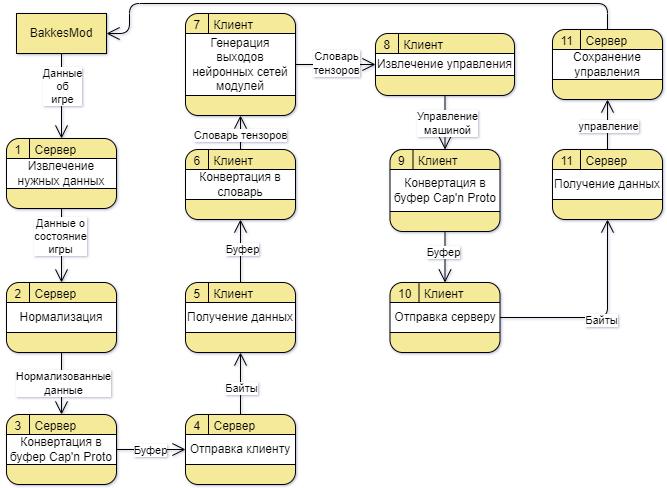


Рисунок 3.2 - Поток данных в тестировании

- - ERROR: No function called "TODO" - -

Кадр из обучения модуля оценки важности представлен на Рис. 3.3.



Рисунок 3.3 - Пример обучения МОВ

В примере окно компьютерной игры на весь экран, поверх него слева направо:

1. Окно графиков наград — на нём представлены награды за разные достижения агентов в среде. В данном случае:

— *Ball Shoot Speed* — скорость полёта меча после касания (быстрее — лучше);

— *Car Velocity* — скорость автомобиля (быстрее — лучше);

— *Closing To Ball* — скорость сближения автомобиля с мячом (быстрее — лучше);

— *Constant Distance To Ball* — дистанция от автомобиля до мяча (ближе — лучше);

— *Constant Reward Distance to Ball* — постоянная награда при приближении к мячу (низкое влияние в примере);

— *Constant Velocity To Ball* — Постоянная награда за скорость автомобиля по отношению к мячу (чем больше вектор скорости автомобиля направлен к мячу — тем лучше).

2. Основное окно системы. В данной примере в окне отображаются параметры обучения. Здесь можно увидеть:

— условия перезапуска цикла обучения (смена эпохи);

— шанс случайных шагов (для обучения модулей);

— тип состояние системы (игра, тренировка, специальная тренировка).

3. Гистограмма влияния модулей на вывод. В данном примере используется только вывод модуля *reachCenterFloor*. В обучение используются три: *reachBallAir*, *reachBallFloor*. *reachCenterFloor*.

## 3.5 Расчёт надёжности

Расчёт надёжности произведён по модели Миллса. В систему было искусственно внесено 10 ошибок. После модульного тестирования было обнаружено 6 внесённых ошибок и 2 собственных:

— ошибка в обнаружении модулей нейронной сети;

— ошибки при получении сообщений от сервера;

— ошибки в расчётах клиента при конвертации данных.

Тогда первоначальное число ошибок в программе N можно оценить по формуле модели Миллса (Формула 3.2).

|  |  |
| --- | --- |
|  | (3.2) |

где n — найденные собственные ошибки,

S — всего внесённых ошибок,

v — найденные внесённые ошибки.

|  |  |
| --- | --- |
|  | (3.3) |

где S – количество искусственно внесённых ошибок. Таким образом, делается предположение о количестве необнаруженных ошибок, (Формула 3.3).

|  |  |
| --- | --- |
|  | (3.3) |

Тогда по формуле соотношения (Формула 3.4):

|  |  |
| --- | --- |
|  | (3.4) |

Можно сказать с вероятностью 0,615, что в программе только две первоначальные ошибки.

Внесённые и обнаруженные первоначальные ошибки исправлены по окончанию тестирования.

## 3.6 Вывод по разделу

В данном разделе проведена разработка системы в целом, адаптирована под тестовую среду, получены результаты по возможности реализации модульной нейронной нейронной сети на явном примере.

# 4 Экономический раздел

В данном разделе описывается организация и планирование работ, расчёт затрат на проведение работа.

Схема взаимодействия состава задействованных в работе лиц представлена на рисунке 4.1.



Рисунок 4.1 - Состав задействованных в работе

## 4.1 Организация и планирование работ

Участники работы представлены в таблице 4.1:

Таблица 4.1 — Участники работы

|  |  |  |  |
| --- | --- | --- | --- |
| **Роль** | **Должность** | **ФИО** | **Деятельность** |
| Разработчик | Студент | Геворкян Артём Арзикович | Реализация продукта, написание кода |
| Консультант | Старший преподаватель кафедры экономики | Потрясаева Елена Александровна | Консультации по экономической части работы |
| Руководитель | кандидат технических наук доцент кафедры вычислительной техники | Штрекер Евгений Николаевич | Организация и контроль исполнения, давать указания по реализации продукта разработчику |

Всего есть 5 главных этапов, разделённые на различные этапы более низкого уровня. Их названия, исполнители, трудоёмкость и продолжительность работ представлены в таблице 4.2.

Таблица 4.2 — Этапы и исполнители с продолжительностью

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Название этапа** | **Исполнитель** | **Трудоёмкость, чел/дни** | **Продолжительность работ, дни** |
| 1 | Разработка и утверждение технического задания | Руководитель | 5 | 5 |
| Консультант | 1 |
| Разработчик | 5 |
| 2 | Технические предложения | Руководитель | 7 | 7 |
| Разработчик | 7 |
| 3 | Эскизный проект: |  |  | 16 |
| 3.1 | Анализ исходных данных и требований | Разработчик | 9 |
| 3.2 | Постановка задачи и цели | Консультант | 1 |
| 3.3 | Разработка описания основного алгоритма | Руководитель | 2 |
| Разработчик | 7 |
| 4 | Технический проект: |  |  | 15 |
| 4.1 | Определение типов входных и выходных данных | Руководитель | 2 |
| Разработчик | 5 |
| 4.2 | Разработка структуры программы и логической структуры базы данных | Руководитель | 2 |
| Разработчик | 10 |
| 5 | Рабочий проект: |  |  | 47 |
| 5.1 | Программирование и отладка программы | Разработчик | 24 |
| 5.2 | Испытание программы | Разработчик | 4 |
| 5.3 | Корректировка программы по результатам испытаний | Разработчик | 5 |
| 5.4 | Подготовка технической документации на программный продукт | Консультант | 1 |
| Разработчик | 7 |
| 5.5 | Сдача готового продукта и внедрение | Руководитель | 2 |
| Разработчик | 7 |
| Итого | | | | 90 |

Календарный график исполнения работы, который соответствует таблице 4.3, представлен на рисунке 4.2. Из рисунка 4.2 так же видно, что общий срок разработки составит 90 дней.

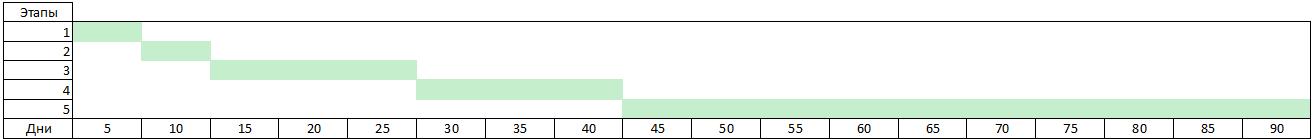


Рисунок 4.2 - Календарный график выполнения работ

## 4.2 Расчёт затрат на проведение работ

Себестоимость проектирования и разработки ИТ-решения складывается из затрат по следующим статьям:

1. Сырье и материалы.

2. Основная заработная плата.

3. Дополнительная заработная плата.

4. Страховые взносы.

5. Амортизация.

6. Прочие расходы.

Рассмотрим каждую статью по-отдельности.

### 4.2.1 Статья 1. Сырьё и материалы

Расчёт затрат на сырьё и материалы представлен в Табл. 4.3:

Таблица 4.3 — Сырьё и материалы

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№ пп** | **Наименование материала** | **Единицы измерения** | **Количество** | **Цена за единицу (руб)** | **Стоимость (руб)** |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | Ручка | шт | 5 | 30 | 150 |
| 2 | Карандаш | шт | 5 | 20 | 100 |
| 3 | Ластик | шт | 5 | 40 | 200 |
| 4 | Блокнот 24 листа | шт | 5 | 415 | 2075 |
| 5 | Бумага A4 | пачка | 2 | 500 | 1000 |
| Итого материалов | | | | | 3525 |
| Транспортно-заготовительные расходы | | | | | 177 |
| Итого | | | | | 3702 |

Таким образом общая сумма затрат по статье «Сырье и материалы» составит 3 702 рублей (три тысячи семьсот два рубля).

### 4.2.2 Статья 2. Основная заработная плата

К статье «Основная заработная плата» относится оплата труда непосредственно занятых выполнением конкретной работы в проекте, включая внештатный состав. Расчёт заработной платы проводится на основе трудоёмкости исполняемых ими работ.

Проект будет создаваться при шестидневной рабочей неделе. Согласно Производственному календарю 2025 года [25] к годовой фонд рабочего времени составляет 299 дней.

Заработная плата рассчитывается по формуле 4.1.

|  |  |
| --- | --- |
|  | (4.1) |

где ОК — месячный оклад работника,

НРВ — годовой фонд рабочего времени.

В Табл. 4.4 приведён результат расчётов заработной платы исполнителей. Количество рабочих дней приведён в Табл. 4.4.

Таблица 4.4 — Расчёт основной заработанной платы

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№ пп** | **Наименование этапа** | **Должность (исполнитель)** | **Месячный оклад (руб)** | **Трудоёмкость (чел/дни)** | **Оплата за день (руб)** | **Оплата за этап (руб)** |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | Техническое задание | Руководитель | 141 000 | 5 | 5659 | 28295 |
| Консультант | 112 000 | 1 | 4495 | 4495 |
| Разработчик | 0 | 5 | 0 | 0 |
| 2 | Технические предложения | Руководитель | 141 000 | 7 | 5659 | 39613 |
| Разработчик | 0 | 7 | 0 | 0 |
| 3 | Эскизный проект | Руководитель | 141 000 | 2 | 5659 | 11318 |
| Консультант | 112 000 | 1 | 4495 | 4495 |
| Разработчик | 0 | 16 | 0 | 0 |
| 4 | Технический проект | Руководитель | 141 000 | 4 | 5659 | 22636 |
| Разработчик | 0 | 15 | 0 | 0 |
| 5 | Рабочий проект | Руководитель | 141 000 | 2 | 5659 | 11318 |
| Консультант | 112 000 | 1 | 4495 | 4495 |
| Разработчик | 0 | 47 | 0 | 0 |
| Итого | | | | | | 122615 |

### 4.2.3 Статья 3. Дополнительная заработная плата

Дополнительной заработной считают выплаты, выдаваемые за неотработанные по уважительным причинам время. Рассчитываются они по формуле 4.2:

|  |  |
| --- | --- |
|  | (4.2) |

где ДЗП — дополнительная заработная плата,

ОЗП — основная заработная плата.

Учитывая основную заработную плату, описанную в статье 2, получаем следующий результат:

|  |  |
| --- | --- |
|  | (4.3) |

Дополнительная заработная плата составляет 30653,75 рублей.

### 4.2.4 Статья 4. Страховые взносы

Для расчёта взносов используется фонд оплаты труда (ФОТ), состоящий из основной и дополнительной заработной платы.

|  |  |
| --- | --- |
|  | (4.4) |

где ФОТ — фонд оплаты труда,

ОЗП — основная заработная плата,

ДЗП — дополнительная заработная плата.

В расчёт страховых взносов входит настоящей ВКР:

1. Единый тариф страховых взносов 30% [26].

2. Травматизм, ставка которого в РТУ МИРЭА составляет *0,2%* [27].

Учитывая вышеперечисленное, ставка выплат составляет *30,2* процентов от ФОТ. Посчитаем страховые взносы (см. Формула 4.5):

|  |  |
| --- | --- |
|  | (4.5) |

С учётом ставки и фонда оплаты труда, страховые взносы составляют 46 287,16 рублей (сорок шесть тысяч двести восемьдесят семь рублей шестнадцать копеек).

### 4.2.5 Статья 5. Амортизация

Амортизация — отчисления части стоимости основных фондов (например, стоимости оборудования) для возмещения их износа. Амортизация включается в издержки производства. Размер амортизационных отчислений определяется на основе установленных норм и балансовой стоимости основных фондов, на которые начисляется амортизация [28].

|  |  |
| --- | --- |
|  | (4.6) |

где А — месячная сумма амортизационных отчислений,

С — первоначальная стоимость объекта,

Т — срок полезного использования в месяцах.

Таблица 4.5 — Амортизационные отчисления

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Наименование оборудования** | **Первоначальная стоимость, руб** | **Срок полезного использования, мес.** | **Месячная сумма амортизационных отчислений** | **Период эксплуатации в месяцах** | **Сумма, руб.** |
| Ноутбук | 144 999 | 36 | 4 027,75 | 3 | 12083,25 |

В итоге сумма амортизационных отчислений составляет 120 083,25 рублей (сто двадцать тысяч восемьдесят три рубля двадцать пять копеек).

### 4.2.6 Статья 6. Прочие расходы

К прочим расходам прежде всего отнесены накладные расходы. Это расходы на содержание и ремонт зданий, сооружений, оборудования, инвентаря. Это затраты, сопутствующие основному производству, но не связанные с ним напрямую, не входящие в стоимость труда и материалов.

Сумма данных расходов определяется процентом от суммы основной заработной платы (ОЗП) и на разных предприятиях, в зависимости от их структуры, технологического процесса и системы управления находится в широком диапазоне — от 100 до 130 процентов. В настоящей работе используется 100%, так как вся работа проходит в ноутбуке, где легко найти альтернативы продуктам. В формуле 4.7 представлен расчёт:

|  |  |
| --- | --- |
|  | (4.7) |

где НР — накладные расходы,

ОЗП — основная заработная плата.

В итоге сумма прочих расходов составляет 134 876,5 рублей (сто тридцать четыре тысячи восемьсот семьдесят шесть рублей пятьдесят копеек).

### 4.2.7 Итог

Вышеперечисленные статьи затрат и результаты расчётов по ним обобщаются в таблице 4.6

Таблица 4.6 — Структура затрат в таблице

|  |  |  |  |
| --- | --- | --- | --- |
| **№ пп** | **Номенклатура статей расходов** | **Затраты (руб.)** | **Доля затрат, %** |
| 1 | 2 | 3 | 4 |
| 1 | Сырьё и материалы | 3 702 | 1,1 |
| 2 | Основная заработная плата | 122 615 | 35,0 |
| 3 | Дополнительная заработная плата | 30 653,75 | 8,8 |
| 4 | Страховые взносы | 46 287,16 | 13,2 |
| 5 | Амортизация | 12 083,25 | 3,5 |
| 6 | Прочие расходы | 134 875,5 | 38,4 |
|  | Итого | 350 216,66 | 100,0 |

Полученные результаты работы будут использоваться внутри университета (организации), поэтому расчет договорной цены не целесообразен.

Для визуализации долевого состава статей затрат в общей себестоимости представим круговую диаграмму на рисунке 4.3.

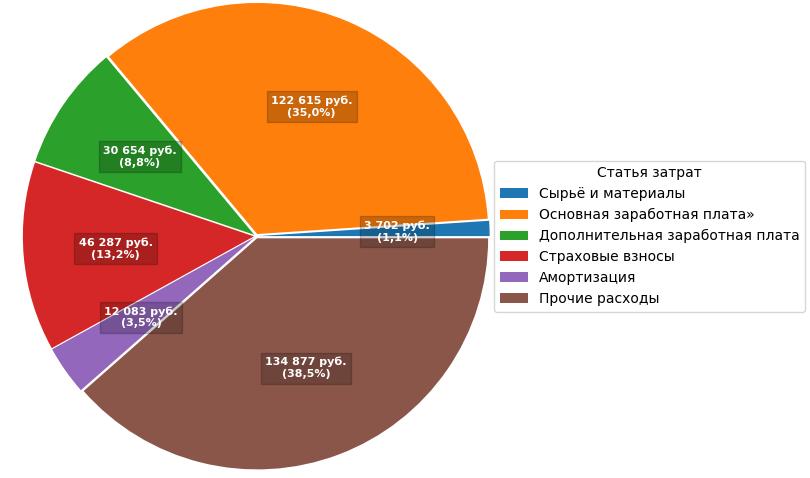


Рисунок 4.3 - Структура затрат на диаграмме

## 4.3 Вывод по разделу

В рамках данного экономического раздела было выполнено планирование работ по теме, был проведен расчет стоимости затрат.

# Заключение

Разработка модульной нейронной сети с плавающим влиянием на вывод представляет собой современный и актуальный подход к решению сложных задач в области искусственного интеллекта и машинного обучения. Такая архитектура позволяет эффективно комбинировать результаты работы нескольких моделей или алгоритмов, учитывая их важность в зависимости от контекста задачи. Это особенно важно в условиях, когда среда является сложной, но позволяет изучать отдельные ситуации обособленно.

В настоящей работе предложено актуальное решение насущных проблем больших моделей нейронных сетей, позволяющая декомпозировать сложную системы на компоненты и адаптировать поведение агента на основе динамически меняющегося контекста. В перспективе данный подход позволит увеличивать сложность нейронных моделей не в размерах, а в количестве и точности (целенаправленность) нейронных сетей, то есть модулей.

The development of a modular neural network with floating influence on inference is a modern and relevant approach to solving complex problems in the field of artificial intelligence and machine learning. This architecture allows you to effectively combine the results of several models or algorithms, taking into account their importance depending on the context of the task. This is especially important in conditions where the environment is complex, but allows you to study individual situations in isolation.

In this paper, we propose an urgent solution to the pressing problems of large neural network models, which makes it possible to decompose a complex system into components and adapt agent behavior based on a dynamically changing context. In the future, this approach will increase the complexity of neural models not in size, but in the number and accuracy (focus) of neural networks, that is, modules.

# Список использованных источников

1. ГОСТ 34.602-89 «Автоматизированные системы. Требования к содержанию и оформлению документов» [Электронный ресурс]. – Режим доступа: <https://docs.cntd.ru/document/9018363>, свободный.

2. Берталанфи Л. фон. Общая теория систем – критический обзор // Исследования по общей теории систем: Сборник переводов / Общ. ред. и вступ. ст. В. Н. Садовского и Э. Г. Юдина. – М.: Прогресс, 1969. – 520 с.

3. Frank Perbet, Chiyuan Zhang, Matthew Botvinick. Machine Theory of Mind — arXiv, 2018.

4. Felix Jahncke, Johannes Betz. MIND-Stack: Modular, Interpretable, End-to-End Differentiability for Autonomous Navigation — arXiv, 2025.

5. Gavin McCracken, Vincent Letourneau, Doina Precup, Jonathan Love. Uncovering a Universal Abstract Algorithm for Modular Addition in Neural Networks — arXiv, 2025.

6. Surajit Majumder, Paritosh Ranjan, Prodip Roy, Bhuban Padhan. Switch-Based Multi-Part Neural Network — arXiv, 2025.

7. «TIOBE Index - TIOBE», [Электронный ресурс]. — URL: https://www.tiobe.com/tiobe-index/ (дата обращения: 27.04.2025).

8. «TIOBE Index - TIOBE», [Электронный ресурс]. — URL: https://www.tiobe.com/tiobe-index/programminglanguages\_definition/ (дата обращения: 16.04.2025).

9. «5 лучших языков программирования для создания нейросетей.», [Электронный ресурс]. — URL: https://tproger.ru/articles/5-luchwih-yazykov-programmirovaniya-dlya-sozdaniya-nejrosetej-253661 (дата обращения: 30.03.2025).

10. «Лучшие языки программирования для ИИ и нейросетей: какой выбрать?», [Электронный ресурс]. — URL: https://rb.ru/story/kakie-yazyki-programmirovaniya-uchit-dlya-raboty-s-ai/ (дата обращения: 24.04.2025).

11. «Самые популярные языки программирования в сфере нейросетей — Илья Букаев на vc.ru», [Электронный ресурс]. — URL: https://vc.ru/id1651755/686392-samye-populyarnye-yazyki-programmirovaniya-v-sfere-neirosetei (дата обращения: 06.05.2025).

12. «Лучшие языки программирования для ИИ в 2025 году | Обзор и советы», [Электронный ресурс]. — URL: https://knewit.kz/2025/05/20/top-5-yazykov-programmirovaniya-dlya-iskusstvennogo-intellekta/ (дата обращения: 27.03.2025).

13. «59% of developers use AI tools &amp; 25.2M JavaScript users», [Электронный ресурс]. — URL: https://www.slashdata.co/post/59-of-developers-use-ai-tools-25-2m-javascript-users (дата обращения: 03.05.2025).

14. «PyTorch», [Электронный ресурс]. — URL: https://pytorch.org (дата обращения: 26.03.2025).

15. «Как новости влияют на рынок», [Электронный ресурс]. — URL: https://bcs-express.ru/novosti-i-analitika/kak-novosti-vliiaiut-na-rynok (дата обращения: 10.04.2025).

16. «Потребление ChatGPT», [Электронный ресурс]. — URL: https://epoch.ai/gradient-updates/how-much-energy-does-chatgpt-use (дата обращения: 19.04.2025).

17. «Эмерджентизм. Большая российская энциклопедия», [Электронный ресурс]. — URL: https://bigenc.ru/c/emerdzhentizm-2a3346 (дата обращения: 28.03.2025).

18. «ЭМЕРДЖЕНТНОСТЬ | это... Что такое ЭМЕРДЖЕНТНОСТЬ?», [Электронный ресурс]. — URL: https://dic.academic.ru/dic.nsf/dic\_fwords/712/ЭМЕРДЖЕНТНОСТЬ (дата обращения: 26.04.2025).

19. «Загадки эмерджентности: как в сложных системах возникает строгий порядок | Идеономика – Умные о главном», [Электронный ресурс]. — URL: https://ideanomics.ru/articles/33110 (дата обращения: 12.03.2025).

20. «About Python&trade; | Python.org», [Электронный ресурс]. — URL: https://www.python.org/about/ (дата обращения: 22.02.2025).

21. «Autograd mechanics &mdash; PyTorch 2.7 documentation», [Электронный ресурс]. — URL: https://docs.pytorch.org/docs/2.7/notes/autograd.html#setting-requires-grad (дата обращения: 03.05.2025).

22. «GitHub - DLR-RM/stable-baselines3: PyTorch version of Stable Baselines, reliable implementations of reinforcement learning algorithms.», [Электронный ресурс]. — URL: https://github.com/DLR-RM/stable-baselines3 (дата обращения: 10.05.2025).

23. «Описание игры Rocket League», [Электронный ресурс]. — URL: https://store.epicgames.com/ru/p/rocket-league (дата обращения: 31.03.2025).

24. «BakkesMod», [Электронный ресурс]. — URL: https://bakkesmod.com (дата обращения: 25.04.2025).

25. «Производственный календарь на 2025 год (6-дневная рабочая неделя) - nalog-nalog.ru», [Электронный ресурс]. — URL: https://nalog-nalog.ru/proizvodstvennyj\_kalendar/2025-6/ (дата обращения: 25.03.2025).

26. «1.1. Единый тариф (единые размеры) страховых взносов на ОПС, ОСС, ОМС для плательщиков, производящих выплаты физическим лицам, на 2025 г. \ КонсультантПлюс», [Электронный ресурс]. — URL: https://www.consultant.ru/document/cons\_doc\_LAW\_93256/97a8d65271700b79d87a7edf33832b5f3ce0fd18/ (дата обращения: 26.04.2025).

27. «Приказ Минтруда России от 30.12.2016 N 851н  
(ред. от 16.05.2025)  
"Об утверждении Классификации видов экономической деятельности по классам профессионального риска"  
(Зарегистрировано в Минюсте России 18.01.2017 N 45279) \ КонсультантПлюс», [Электронный ресурс]. — URL: https://www.consultant.ru/document/cons\_doc\_LAW\_211247/ (дата обращения: 24.05.2025).

28. «НК РФ Статья 256. Амортизируемое имущество \ КонсультантПлюс», [Электронный ресурс]. — URL: https://www.consultant.ru/document/cons\_doc\_LAW\_28165/df53ee1751d3e93dbf8c0d34076675da18a2fd06/ (дата обращения: 12.05.2025).

# Приложения

Приложение А - Реализация модуля оценки важности

Приложение Б - Контроллер модулей

Приложение В - Классы обучения нейронных сетей

Приложение Г - Структуры данных, используемые системой

Приложение Д - Достижения

Приложение Е - Презентационный материал

## Приложение А - Реализация модуля оценки важности

Листинг 1. Реализация модуля оценки важности

|  |
| --- |
| from typing import Iterable, Sequence from pathlib import Path  from torch import nn, load  from mai.functions import \_init\_weights   class MAINet(nn.Module):  def \_\_init\_\_(self, modules\_amount: int):  assert isinstance(modules\_amount, int)  assert modules\_amount > 1  super().\_\_init\_\_()  from mai.capnp.data\_classes import STATE\_KEYS  self.fc1 = nn.Linear(len(STATE\_KEYS)-1, 256)  self.fc2 = nn.Linear(256, modules\_amount)  self.activation1 = nn.ReLU()  self.activation2 = nn.ReLU()   def forward(self, x):  x = self.fc1(x)  x = self.activation1(x)  x = self.fc2(x)  x = self.activation2(x)  return x   def init\_weights(self) -> None:  nn.init.kaiming\_uniform\_(self.fc1.weight.data)  self.fc1.bias.data.fill\_(0.0)  nn.init.kaiming\_uniform\_(self.fc2.weight.data)  self.fc2.bias.data.fill\_(0.0)   @staticmethod  def mai\_name(modules: Iterable[str]) -> str:  return f'mai\_{'\_'.join(i for i in modules)}'   @classmethod  def load[T: MAINet](  cls: type[T],  modules\_folder: Path,  modules: Sequence[str]  ) -> T:  mai\_name = f'{cls.mai\_name(modules)}.state\_dict'  mai\_path = modules\_folder / mai\_name  net = cls(len(modules))  if mai\_path.exists():  state\_dict = load(mai\_path)  net.load\_state\_dict(state\_dict)  else:  \_init\_weights(net)  return net |

## Приложение Б - Контроллер модулей

Листинг 2. Контроллер модулей

|  |
| --- |
| from io import StringIO from typing import Generator, Mapping, TYPE\_CHECKING from pathlib import Path  import torch  from mai.capnp.data\_classes import ModulesOutputMapping from .networks import build\_networks, ModuleBase, NNModuleBase  if TYPE\_CHECKING:  from mai.capnp.names import MAIGameState   class ModulesController:  \_\_slots\_\_ = (  #   '\_all\_modules', '\_ordered\_modules',   # Torch mandatory variables  '\_training', '\_device',   # public:  'state', 'models\_folder',  )  \_all\_modules: Mapping[str, ModuleBase]  \_ordered\_modules: list[ModuleBase]  \_device: torch.device  models\_folder: Path | None   def \_\_init\_\_(  self,  \_device: torch.device | None = None,  models\_folder: Path | None = None  ) -> None:  super().\_\_init\_\_()   assert models\_folder is None or isinstance(models\_folder, Path)  self.models\_folder = models\_folder   if \_device is None:  \_device = torch.device(  'cuda' if torch.cuda.is\_available() else 'cpu'  )  self.\_device = \_device   self.\_all\_modules = {k: m(self.models\_folder) for k, m in build\_networks().items()}  self.\_ordered\_modules = []  self.\_training = False   def \_\_call\_\_[T](  self, value: T  ) -> T:  if isinstance(value, ModulesOutputMapping):  for module in self.\_ordered\_modules:  assert any((  module.enabled and module.power >= 0.1,  not module.enabled and module.power <= 0.1,  )), module  module.inference(value)  return value  if isinstance(value, list):  for v in value:  self(v)  return value  else:  raise RuntimeError(f"Can't inference {type(value)}({value})")   def \_\_repr\_\_(self) -> str:  io = StringIO()  io.write('<MC')  if self.models\_folder is not None:  io.write(f' path={self.models\_folder}')  loaded = sum((1 if m.loaded else 0 for m in self.get\_all\_modules()))  enabled = sum((1 if m.loaded else 0 for m in self.get\_all\_modules()))  io.write(f" with {enabled}/{loaded}/{len(self.\_all\_modules)}")  io.write('>')  return io.getvalue()   def save(self) -> None:  for module in self.get\_all\_modules():  module.save()   @property  def training(self) -> bool:  return self.\_training   @training.setter  def training(self, value: bool) -> None:  assert isinstance(value, bool)  if self.\_training == value: return  for model in self.get\_all\_modules():  model.training = value  self.\_training = value   @property  def device(self) -> torch.device:  return self.\_device   @device.setter  def device(self, value: torch.device) -> None:  assert isinstance(value, torch.device)  if self.\_device == value: return  self.\_device = value  for module in self.get\_all\_modules():  module.set\_device(value)   def enabled\_parameters(self) -> Generator[torch.nn.Parameter, None, None]:  for module in self.get\_all\_modules():  if not module.loaded:  continue  for parameter in module.enabled\_parameters():  assert parameter.dtype == torch.get\_default\_dtype()  yield parameter   def iter\_models(self) -> Generator[torch.nn.Module, None, None]:  for module in self.\_ordered\_modules:  if not isinstance(module, NNModuleBase):  continue  assert module.\_model is not None  yield module.\_model   def get\_all\_modules(self) -> Generator[ModuleBase , None, None]:  for module in self.\_all\_modules.values():  yield module   def get\_module(self, \_module: str) -> ModuleBase:  assert isinstance(\_module, str)  module = self.\_all\_modules.get(\_module, None)  if module is None:  raise KeyError(  f"Module {\_module} is not found. "  "possible modules: "  f"{', '.join((i for i in self.\_all\_modules.keys()))}"  )  return module   def module\_enable(self, \_module: str) -> int:  """ Enables module, which includes it in `NNController.\_ordered\_modules`  :param name: Name of the module  :return: Index of inserted module in `NNController.\_ordered\_modules`  """  assert isinstance(\_module, str)  module = self.get\_module(\_module)  assert isinstance(module, ModuleBase), module  assert module not in {i for i in self.\_ordered\_modules}, \  f"{self.\_ordered\_modules}{module.enabled}"  if not module.loaded:  self.module\_load(module.name)  requirements = module.requires()  requirements\_indexes = []  index = 0  while requirements:  required\_module\_name = requirements.pop()  assert required\_module\_name in self.\_all\_modules.keys()  found\_module = None  for module in self.\_ordered\_modules[index:]:  index += 1  if module.name == required\_module\_name:  found\_module = module  break  if not found\_module:  index = self.module\_enable(required\_module\_name) + 1  requirements\_indexes = [i if i < index else (  i+1  ) for i in requirements\_indexes]  requirements\_indexes.append(index)  if requirements\_indexes:  index = max(requirements\_indexes) + 1  self.\_ordered\_modules.insert(index, module)  module.enabled = True  return index   def module\_disable(self, \_module: str) -> None:  assert isinstance(\_module, str)  module = self.get\_module(\_module)  assert isinstance(module, ModuleBase), module  assert module in {i for i in self.\_ordered\_modules}  index = self.\_ordered\_modules.index(module)   self.\_ordered\_modules.pop(index)  module\_output = set(module.output\_types)  dependencies: list[str] = []  for next\_module in self.\_ordered\_modules[index:]:  intersection = set(next\_module.input\_types).intersection(module\_output)  if intersection:  dependencies.append(next\_module.name)  for dependency\_name in dependencies:  self.module\_disable(dependency\_name)  module.enabled = False   def module\_load(self, \_module: str) -> None:  assert isinstance(\_module, str)  module = self.get\_module(\_module)  assert module not in {i for i in self.\_ordered\_modules}  if not module.loaded:  module.load()   def module\_unload(  self,  module\_name: str,  save: bool = False  ) -> None:  assert isinstance(module\_name, str)  module = self.get\_module(module\_name)  if module.enabled:  self.module\_disable(module.name)  assert module not in {i for i in self.\_ordered\_modules}  module.unload(save=save)   def module\_power(  self,  module\_name: str,  power: float  ) -> None:  assert isinstance(module\_name, str)  assert isinstance(power, float)  assert 0 <= power <= 1, power  module = self.get\_module(module\_name)  if power < 0.1:  if module.enabled:  self.module\_disable(module\_name)  module.power = power  return  module.power = power   def unload\_all\_modules(  self,  save: bool = True,  \_exceptions: set[ModuleBase | str] | None = None  ) -> None:  if \_exceptions is None:  exceptions = set()  else:  exceptions = set()  for exc in \_exceptions:  if isinstance(exc, str):  exceptions.add(self.get\_module(exc))  else:  assert isinstance(exc, ModuleBase)  exceptions.add(exc)  for module in self.\_ordered\_modules[::-1].copy():  if module.name in exceptions:  continue  if not module.loaded:  continue  self.module\_unload(module.name, save=save)   def copy[T: ModulesController](  self: T,  /,  copy\_models\_folder: bool = False,  copy\_device: bool = False  ) -> T:  models\_folder = None  if copy\_models\_folder:  models\_folder = self.models\_folder  device = None  if copy\_device:  device = self.device  controller = type(self)(device, models\_folder)  del device, models\_folder   assert isinstance(self.\_ordered\_modules, list)  for module in self.\_ordered\_modules:  new\_module = type(module)(module.models\_path)  new\_module.training = module.training  new\_module.enabled = module.enabled  new\_module.power = module.power  if isinstance(module, NNModuleBase):  assert isinstance(new\_module, NNModuleBase)  new\_module.\_model = module.\_create()  if module.\_model is not None:  new\_module.\_model.load\_state\_dict(  module.\_model.state\_dict(), strict=True  )  controller.\_ordered\_modules.append(new\_module)  return controller |

## Приложение В - Классы обучения нейронных сетей

Листинг 3. Классы обучения нейронных сетей

|  |
| --- |
| from abc import ABC, abstractmethod from pathlib import Path from typing import (  Generator, Mapping, TYPE\_CHECKING ) import random # from itertools import chain, pairwise  import torch  from mai.ai.mcs\_controller import MCsController from mai.ai.mai\_model import MAINet from mai.capnp.data\_classes import (  ModulesOutputMapping,  # Vector,  STATE\_KEYS,  CONTROLS\_KEYS,  RunParameters ) from .memory import ReplayMemory, TensorReplayMemory from .rewards import NNRewardBase, build\_rewards from .controller import ModulesController from .utils import polyak\_update  if TYPE\_CHECKING:  from mai.capnp.data\_classes import RunParameters  from mai.ai.networks.base import ModuleBase, NNModuleBase   \_\_all\_\_ = (  'BaseTrainer',  'ModuleTrainer',  'MAITrainer', )   class Critic(torch.nn.Module):  def \_\_init\_\_(self, hidden\_size: int= 32, modules\_amount: int = 0):  assert isinstance(hidden\_size, int)  assert isinstance(modules\_amount, int)  assert 0 <= modules\_amount  assert hidden\_size > 8  super(Critic, self).\_\_init\_\_()  self.fc1 = torch.nn.Linear(  len(STATE\_KEYS)+modules\_amount-1,  hidden\_size  )  self.fc2 = torch.nn.Linear(hidden\_size, hidden\_size)  self.fc3 = torch.nn.Linear(hidden\_size, hidden\_size)  self.fc4 = torch.nn.Linear(hidden\_size, 1)   def forward(self, state, powers):  x = torch.cat((state, powers), dim=1)  x = torch.relu(self.fc1(x))  x = torch.relu(self.fc2(x))  x = torch.relu(self.fc3(x))  return self.fc4(x)   class ModulesGetter:  def \_\_init\_\_(self) -> None:  pass   def \_\_get\_\_(  self,  instance: 'ModuleTrainer',  \_: type['ModuleTrainer']  ) -> MCsController:  mcs = []  if instance.\_mc is not None:  mcs.append(instance.\_mc)  if instance.\_target\_mc is not None:  mcs.append(instance.\_target\_mc)  return MCsController(\*mcs)   def \_\_set\_\_(self, instance: 'ModuleTrainer', value):  raise RuntimeError("Modules can't be set")   def \_\_delete\_\_(self, instance):  raise RuntimeError("Modules can't be deleted")   class BaseTrainer(ABC):  \_slots\_\_ = (  '\_gen', '\_loaded',  'params'  )  \_instance: 'BaseTrainer | None' = None   def \_\_init\_\_(  self,  params: 'RunParameters'  ) -> None:  assert isinstance(params.modules, list)  assert params.modules  assert all((isinstance(i, str) for i in params.modules))  super().\_\_init\_\_()  self.\_loaded = False  self.\_gen = None  self.params = params   def \_\_enter\_\_[T: BaseTrainer](self: T) -> T:  BaseTrainer.\_instance = self  if not self.\_loaded:  self.prepare()  return self   def \_\_exit\_\_(self, \*args) -> None:  self.\_loaded = False  BaseTrainer.\_instance = None   def prepare(self) -> None:  assert not self.\_loaded  assert self.\_gen is None, self.\_gen  self.\_gen = self.inference\_gen()  next(self.\_gen)  self.\_loaded = True   def inference(self, state\_map: ModulesOutputMapping) -> None:  assert self.\_loaded  assert not state\_map.has\_controls()  assert self.\_gen is not None  self.\_gen.send(state\_map)  # assert state\_map.has\_controls(), state\_map   @abstractmethod  def inference\_gen(self) -> Generator[None, ModulesOutputMapping, None]:  raise NotImplementedError()   @abstractmethod  def epoch\_end(self) -> None:  pass  class ModuleTrainer(BaseTrainer):  \_\_slots\_\_ = (  '\_loaded', '\_gen',  '\_all\_rewards', '\_loss',  '\_memory',   # Hyperparameters  'batch\_size',  'gamma',  'tau',   # Variables  '\_epoch\_num',   # Model fields  '\_mc', '\_target\_mc',  '\_mc\_optimizer',  'mc\_lr',   # Critic fields  '\_critic', '\_target\_critic',  '\_critic\_optimizer',  'critic\_lr',  )  \_memory: ReplayMemory[ModulesOutputMapping]  \_all\_rewards: Mapping[str, 'NNRewardBase']  batch\_size: int  modules = ModulesGetter()   \_critic: Critic  \_target\_critic: Critic  \_critic\_optimizer: torch.optim.Optimizer   \_mc: ModulesController  \_target\_mc: ModulesController  \_mc\_optimizer: torch.optim.Optimizer   def \_\_init\_\_(  self,  params: 'RunParameters'  ) -> None:  super().\_\_init\_\_(params)  self.\_all\_rewards = {k: m() for k, m in build\_rewards().items()}   def \_\_exit\_\_(self, \*args) -> None:  super().\_\_exit\_\_(\*args)  self.\_target\_mc.save()   def prepare(self) -> None:  super().prepare()  self.batch\_size = 10  self.mc\_lr = 1e-4  self.critic\_lr = 1e-4  self.gamma = 0.97  self.tau = 0.001   self.\_memory = ReplayMemory()  self.\_loss = torch.nn.MSELoss()  self.\_epoch\_num = 0  next(self.\_gen)   self.\_target\_mc = ModulesController(  models\_folder=Path()  )  self.\_mc = self.\_target\_mc.copy(  copy\_device=True,  copy\_models\_folder=False,  )  self.\_mc.models\_folder = None   self.modules.training = True  def assign\_power(m: 'ModuleBase'):  m.power = 1.  for module in self.params.modules:  self.modules.enable(module)  self.modules.module\_apply(module, assign\_power)   self.\_mc\_optimizer = torch.optim.Adam(  list(self.modules.\_mc.enabled\_parameters()),  lr=self.mc\_lr,  amsgrad=True  )   critic\_hidden\_size = 256  self.\_critic = Critic(critic\_hidden\_size)  self.\_target\_critic = Critic(critic\_hidden\_size)  self.\_critic\_optimizer = torch.optim.Adam(  self.\_critic.parameters(recurse=True),  lr=self.critic\_lr,  amsgrad=True  )   self.\_loaded = True   #  # Training  #   def \_select\_action(self, state: ModulesOutputMapping) -> None:  assert self.\_loaded  assert isinstance(state, ModulesOutputMapping)  assert state.has\_state()  assert not state.has\_controls()  if random.random() > self.params.random\_threshold:  self.\_target\_mc(state)  else:  state.update(ModulesOutputMapping.create\_random\_controls(  random\_jump=self.params.random\_jump  ))  assert state.has\_any\_controls()   def inference\_gen(self) -> Generator[None, ModulesOutputMapping, None]:  state = yield  assert self.\_loaded   print('Starting training')  print('Epoch | Reward sum | Reward avg | Critic loss | MC loss')   while True:  assert state.has\_all\_state(), state.keys()  assert not state.has\_any\_controls(), state.keys()  self.\_select\_action(state)  self.\_memory.add(state)   observations = yield  state = observations   def epoch\_end(self) -> None:  assert self.\_loaded  self.\_epoch\_num += 1   # batch = self.\_memory.sample(len(self.\_memory))  batch = self.\_memory  batch.rewards\_differentiate()  batch.rewards\_affect\_previous(percent=0.3, start\_multiplier=0.3)  batch.rewards\_normalize()  states, actions, next\_states, rewards = batch.to\_canonical()  rewards = rewards.unsqueeze(1)   # import numpy as np  # with open('temp.csv', mode='w', encoding='utf-8') as f:  # keys = (\*STATE\_KEYS, \*CONTROLS\_KEYS)  # f.write(','.join(keys))  # f.write(',reward')  # for m in batch:  # f.write('\n')  # for key in keys:  # v = m.\_avg\_from\_dict(key, requires\_grad=False)  # assert v is not None  # f.write(str(float(v)))  # f.write(',')  # f.write(str(m.reward))   # Update critic network  all\_actions\_m: list[ModulesOutputMapping] = self.\_target\_mc(batch)  next\_actions = torch.stack([m.extract\_controls().toTensor() for m in all\_actions\_m[1:]])  assert states .shape == next\_states .shape  assert actions.shape == next\_actions.shape  assert states.shape[0] == actions.shape[0]  assert next\_states.shape[0] == actions.shape[0]  assert states.shape[1] == len(STATE\_KEYS)  assert actions.shape[1] == len(CONTROLS\_KEYS)  target\_q\_values: torch.Tensor = self.\_target\_critic(next\_states, next\_actions)  real\_q\_values: torch.Tensor = self.\_critic(states, actions)  assert target\_q\_values.shape == real\_q\_values.shape, f"{target\_q\_values.shape} {real\_q\_values.shape}"  assert rewards.shape == target\_q\_values.shape, f"{rewards.shape} {target\_q\_values.shape}"  expected\_q\_values: torch.Tensor = + (self.gamma \* target\_q\_values)   assert real\_q\_values.shape == expected\_q\_values.shape  critic\_loss = self.\_loss(real\_q\_values, expected\_q\_values.detach())  self.\_critic\_optimizer.zero\_grad()  critic\_loss.backward()  self.\_critic\_optimizer.step()   # Update MC  mc\_loss = -self.\_critic(  states,  torch.stack([m.extract\_controls().toTensor() for m in all\_actions\_m[:-1]])  ).mean()  self.\_mc\_optimizer.zero\_grad()  mc\_loss.backward()  self.\_mc\_optimizer.step()   # Update target networks  for t\_p, p in zip(self.\_target\_mc.enabled\_parameters(), self.\_mc.enabled\_parameters()):  t\_p.data.copy\_(self.tau \* p.data + (1.0 - self.tau) \*t\_p.data)  for t\_p, p in zip(self.\_target\_critic.parameters(), self.\_critic.parameters()):  t\_p.data.copy\_(self.tau \* p.data + (1.0 - self.tau) \*t\_p.data)   reward\_sum = sum((i.reward for i in self.\_memory))  print(  f"{self.\_epoch\_num: >5}"[:5],  f"{reward\_sum}"[:9],  f"{reward\_sum/len(self.\_memory)}"[:9],  f"{critic\_loss}"[:11],  f"{mc\_loss}"[:7],  sep=' | '  )  self.\_memory.clear()   def close(self) -> None:  pass   class MAITrainer(BaseTrainer):  \_\_slots\_\_ = (  '\_memory', '\_mc', '\_mse\_loss',  '\_mai\_net', '\_epoch\_num',   'models\_folder',  'mai\_lr', 'critic\_lr',  'tau', 'gamma',  )   def \_\_init\_\_(  self,  params: RunParameters  ) -> None:  self.models\_folder = Path()  self.\_mc = ModulesController(  models\_folder=self.models\_folder  )  super().\_\_init\_\_(params)   def prepare(self) -> None:  assert self.\_gen is None  super().prepare()  assert self.\_gen is not None  self.critic\_lr = 1e-4  self.mai\_lr = 1e-4  # Rate at which critic weights transferred  # to target critic  self.tau = 0.001  # Constant reward decrease  self.gamma = 0.97  critic\_hidden\_size = 256   self.\_mse\_loss = torch.nn.MSELoss()  self.\_memory = TensorReplayMemory()   self.\_mc.training = False  self.\_mc.unload\_all\_modules(save=False)  for module\_name in self.params.modules:  module = self.\_mc.get\_module(module\_name)  if not module.loaded:  module.load()  module.training = False  module.power = 1.   self.\_mai\_net = MAINet.load(  self.models\_folder,  self.params.modules  )  self.\_mai\_optimizer = torch.optim.Adam(  self.\_mai\_net.parameters(recurse=True),  lr=self.mai\_lr,  amsgrad=True  )   critic\_args = (  critic\_hidden\_size,  len(self.params.modules)  )  self.\_critic = Critic(\*critic\_args)  self.\_target\_critic = Critic(\*critic\_args)  self.\_critic\_optimizer = torch.optim.Adam(  self.\_critic.parameters(recurse=True),  lr=self.critic\_lr,  amsgrad=True  )  self.\_epoch\_num = 0   self.\_loaded = True   def inference\_gen(self) -> Generator[None, ModulesOutputMapping, None]:  state = yield  assert self.\_loaded  assert self.\_gen  assert hasattr(self, '\_mai\_net')   print('Epoch | Reward sum | Reward avg | Critic loss | MAI loss')  for module\_name in self.params.modules:  self.\_mc.module\_load(module\_name)  self.\_mc.module\_enable(module\_name)  self.\_mc.module\_power(module\_name, 1.)  self.\_mai\_net.train()   while True:  assert state.has\_all\_state(), state.keys()  assert not state.has\_any\_controls(), state.keys()  powers: torch.Tensor = self.\_mai\_net(  state  .extract\_state(requires\_grad=True)  .toTensor(requires\_grad=True)  )  powers = powers.clip(0.1, 1.)  assert all(0 <= i.item() <= 1 for i in powers), powers  state.powers = powers  assert powers.shape == (len(self.params.modules),)  self.\_memory.add\_v(  state=state,  powers=powers  )  assert all(0 <= i.item() <= 1 for i in powers), powers  for module\_name, power in zip(self.params.modules, powers):  power = power.item()  self.\_mc.module\_power(  module\_name,  power  )   assert not state.has\_any\_controls()  self.\_mc(state)  # from pprint import pp  # pp(state)   observations = yield  state = observations   def epoch\_end(self) -> None:  """Using TD3 from stable-baselines3  https://github.com/DLR-RM/stable-baselines3/blob/master/stable\_baselines3/td3/td3.py  """  assert self.\_loaded  self.\_epoch\_num += 1   # batch = self.\_memory.sample(len(self.\_memory))  batch = self.\_memory  batch.rewards\_differentiate()  batch.rewards\_affect\_previous(percent=0.3)  batch.rewards\_normalize()  observations, actions, next\_observations, rewards, powerss = batch.to\_canonical()  rewards = rewards.unsqueeze(1)  actor\_losses, critic\_losses = [], []   with torch.no\_grad():  # Select power according to policy  next\_powerss = (self.\_mai\_net(next\_observations)).clamp(-1, 1)   # Compute the next Q-values: min over all critics targets  next\_q\_values = self.\_target\_critic(next\_observations, next\_powerss)  next\_q\_values, \_ = torch.min(next\_q\_values, dim=1, keepdim=True)  target\_q\_values = rewards \* self.gamma \* next\_q\_values   # Get current Q-values estimates for each critic network  current\_q\_values = self.\_critic(observations, powerss)   # Compute critic loss  critic\_loss = self.\_mse\_loss(current\_q\_values, target\_q\_values)  assert isinstance(critic\_loss, torch.Tensor)  critic\_losses.append(critic\_loss.item())   # Optimize the critics  self.\_critic\_optimizer.zero\_grad()  try:  critic\_loss.backward()  except RuntimeError:  self.\_memory.clear()  print('Critic loss inplace')  return  self.\_critic\_optimizer.step()   critic\_forward = self.\_critic.forward  # assert isinstance(critic\_forward, torch.Module)  # assert not isinstance(critic\_forward, torch.Tensor)  mai\_loss = -critic\_forward(observations, self.\_mai\_net(observations)).mean()  actor\_losses.append(mai\_loss.item())   # Optimize the actor  self.\_mai\_optimizer.zero\_grad()  mai\_loss.backward()  self.\_mai\_optimizer.step()   polyak\_update(self.\_critic.parameters(), self.\_target\_critic.parameters(), self.tau)  polyak\_update(self.\_mai\_net.parameters(), self.\_mai\_net.parameters(), self.tau)  # Copy running stats, see GH issue #996  # polyak\_update(self.critic\_batch\_norm\_stats, self.critic\_batch\_norm\_stats\_target, 1.0)  # polyak\_update(self.actor\_batch\_norm\_stats, self.actor\_batch\_norm\_stats\_target, 1.0)   print(  f"{self.\_epoch\_num: >5}"[:5],  f"{self.\_memory.sum\_reward()}"[:9],  f"{self.\_memory.avg\_reward()}"[:9],  f"{critic\_loss}"[:11],  f"{mai\_loss}"[:7],  sep=' | '  )  self.\_memory.clear() |

## Приложение Г - Структуры данных, используемые системой

Листинг 4. Структуры данных системы

|  |
| --- |
| from typing import (  Literal, Type, NamedTuple, Annotated,  Mapping, Iterable,  TYPE\_CHECKING ) import enum from dataclasses import dataclass, field from math import tanh, exp  import numpy as np import torch  from mai.capnp.names import (  MAIControls,  MAIGameState,  MAIVector,  MAIRotator ) from mai.settings import Settings  if TYPE\_CHECKING:  from numpy import typing as npt  import torch  from mai.ai.networks.base import ModuleBase  CAR\_OR\_BALL = Literal['car'] | Literal['ball']  V\_OR\_AV = Literal['v'] | Literal['av']  MAGNITUDE\_OFFSET\_TYPING = dict[CAR\_OR\_BALL, dict[V\_OR\_AV, float]]  NumpyVector = Annotated[npt.NDArray[np.float32], Literal[3]]   ARENA\_SIZE: 'Vector' DENORMALIZATION\_VECTOR: 'Vector' STATE\_KEYS = (  'state.car.position.x',  'state.car.position.y',  'state.car.position.z',  'state.car.velocity.x',  'state.car.velocity.y',  'state.car.velocity.z',  'state.car.rotation.pitch',  'state.car.rotation.roll',  'state.car.rotation.yaw',  'state.car.angularVelocity.x',  'state.car.angularVelocity.y',  'state.car.angularVelocity.z',  'state.ball.position.x',  'state.ball.position.y',  'state.ball.position.z',  'state.ball.velocity.x',  'state.ball.velocity.y',  'state.ball.velocity.z',  'state.ball.rotation.pitch',  'state.ball.rotation.roll',  'state.ball.rotation.yaw',  'state.ball.angularVelocity.x',  'state.ball.angularVelocity.y',  'state.ball.angularVelocity.z',  'state.dead',  'state.boostAmount',  'MAIGameState', ) CONTROLS\_KEYS = (  'controls.throttle',  'controls.steer',  'controls.pitch',  'controls.yaw',  'controls.roll',  'controls.boost',  'controls.jump',  'controls.handbrake',  'controls.dodgeVertical',  'controls.dodgeStrafe', ) REWARD\_KEY = '\_reward' POWERS\_LEY = '\_powers'   class DodgeVerticalType(enum.IntEnum):  BACKWARD = -1  NONE = 0  FORWARD = 1  class DodgeStrafeType(enum.IntEnum):  LEFT = -1  NONE = 0  RIGHT = 1  class RunType(str, enum.Enum):  CUSTOM\_TRAINING = 'Custom training'  FREEPLAY = 'Freeplay'  v11 = '1v1'  v22 = '2v2'  v33 = '3v3'  v44 = '4v4'  class RestartReason(str, enum.Enum):  BALL\_TOUCH = 'Ball touch'  BALL\_EXPLODE = 'Goal explode'  TIMEOUT = 'Timeout'  class RunParameters(NamedTuple):  type: RunType  modules: list[str]  rewards: Mapping[str, float]  restart\_reasons: set[RestartReason]  restart\_timeout: int  random\_threshold: float  random\_jump: bool  class Vector:  \_\_slots\_\_ = ('\_arr',)   def \_\_init\_\_(self, arr: 'NumpyVector') -> None:  self.\_arr = arr   def \_\_str\_\_(self) -> str:  return str(self.\_arr)   def \_\_repr\_\_(self) -> str:  return f"<V {self.\_arr[0]:.2f} {self.\_arr[1]:.2f} {self.\_arr[2]:.2f}>"   def \_\_sub\_\_(self: 'Vector', other: 'Vector') -> 'Vector':  return type(self)(self.\_arr - other.\_arr)   def \_\_mul\_\_(self: 'Vector', other: 'Vector') -> 'Vector':  return type(self)(self.\_arr \* other.\_arr)   def \_\_getitem\_\_(self, index: int) -> float:  assert isinstance(index, int)  assert 0 <= index <= 2  return self.\_arr[index]   @property  def x(self) -> float:  return self.\_arr[0]   @property  def y(self) -> float:  return self.\_arr[1]   @property  def z(self) -> float:  return self.\_arr[2]   def magnitude(self) -> float:  return (((  self.\_arr\*DENORMALIZATION\_VECTOR.\_arr  )\*\*2).sum())\*\*0.5   @classmethod  def from\_mai[T:Vector](cls: Type[T], v: MAIVector) -> T:  return cls(np.array((v.x, v.y, v.z)))  @dataclass(frozen=True, slots=True, kw\_only=True) class NormalControls:  throttle: float = field(default=0)  steer: float = field(default=0)  pitch: float = field(default=0)  yaw: float = field(default=0)  roll: float = field(default=0)  boost: bool = field(default=False)  jump: bool = field(default=False)  handbrake: bool = field(default=False)  dodgeVertical: DodgeVerticalType = field(default=DodgeVerticalType.NONE)  dodgeStrafe: DodgeStrafeType = field(default=DodgeStrafeType.NONE)   def toMAIControls(self) -> MAIControls:  assert -1 <= self.throttle <= 1  assert -1 <= self.steer <= 1  assert -1 <= self.pitch <= 1  assert -1 <= self.yaw <= 1  assert -1 <= self.roll <= 1  assert isinstance(self.boost, bool)  assert isinstance(self.jump, bool)  assert isinstance(self.handbrake, bool)  assert isinstance(self.dodgeVertical, DodgeVerticalType)  assert isinstance(self.dodgeStrafe, DodgeStrafeType)   controls = MAIControls.new\_message()  if abs(self.throttle) > Settings.control\_apply\_threshold:  controls.throttle = self.throttle  if abs(self.steer) > Settings.control\_apply\_threshold:  controls.steer = self.steer  if abs(self.pitch) > Settings.control\_apply\_threshold:  controls.pitch = self.pitch  if abs(self.yaw) > Settings.control\_apply\_threshold:  controls.yaw = self.yaw  if abs(self.roll) > Settings.control\_apply\_threshold:  controls.roll = self.roll  if self.boost: controls.boost = True  if self.jump: controls.jump = True  if self.handbrake: controls.handbrake = True  if self.dodgeVertical.value != 0: controls.dodgeForward = float(self.dodgeVertical.value)  if self.dodgeStrafe.value != 0: controls.dodgeStrafe = float(self.dodgeStrafe.value)  return controls   @dataclass(frozen=True, slots=True, kw\_only=True) class FloatControls:  throttle: float = field(default=0)  steer: float = field(default=0)  pitch: float = field(default=0)  yaw: float = field(default=0)  roll: float = field(default=0)  boost: float = field(default=0)  jump: float = field(default=0)  handbrake: float = field(default=0)  dodgeVertical: float = field(default=0)  dodgeStrafe: float = field(default=0)   @staticmethod  def absMax(value\_n: float, value\_p: float) -> float:  """  absMax(0.5, 0.7) -> 0.7  absMax(0.7, 0.5) -> -0.7  absMax(0.7, 0.5) -> -0.7  absMax(0.5, 0.5) -> 0.5  """  assert isinstance(value\_n, float)  assert isinstance(value\_p, float)  assert 0 <= value\_n <= 1  assert 0 <= value\_p <= 1  # return -value\_n if abs(value\_n) > value\_p else value\_p  return value\_p-value\_n   def toNormalControls(self) -> NormalControls:  assert -1 <= self.throttle <= 1  assert -1 <= self.steer <= 1  assert -1 <= self.pitch <= 1  assert -1 <= self.yaw <= 1  assert -1 <= self.roll <= 1  assert 0 <= self.boost <= 1  assert 0 <= self.jump <= 1  assert 0 <= self.handbrake <= 1  assert -1 <= self.dodgeVertical <= 1  assert -1 <= self.dodgeStrafe <= 1   return NormalControls(  throttle=self.throttle,  steer=self.steer,  pitch=self.pitch,  yaw=self.yaw,  roll=self.roll,  boost=bool(round(self.boost)),  jump=bool(round(self.jump)),  handbrake=bool(round(self.handbrake)),  dodgeVertical=DodgeVerticalType(round(self.dodgeVertical)),  dodgeStrafe=DodgeStrafeType(round(self.dodgeStrafe))  )   @classmethod  def from\_dict\_float[T:FloatControls](cls: Type[T], d: Mapping[str, float]) -> T:  return cls(  throttle = d.get('throttle', 0),  steer = d.get('steer', 0),  pitch = d.get('pitch', 0),  yaw = d.get('yaw', 0),  roll = d.get('roll', 0),  boost = d.get('boost', 0),  jump = d.get('jump', 0),  handbrake = d.get('handbrake', 0),  dodgeVertical = d.get('dodgeVertical', 0),  dodgeStrafe = d.get('dodgeStrafe', 0)  )  @dataclass(frozen=False, slots=True, kw\_only=True) class AdditionalContext:  """  Additional context values, that changes based on messages from server  """  # Multiplier based on player team  # If on car spawn y is negative, than we place here -1, else 1  team\_multiplier: Literal[-1, 1] = field(default=1)  latest\_message: MAIGameState.MessageType = field(default='none')  exchanges\_since\_latest\_message: int = field(default=0)  magnitude\_offsets: 'MAGNITUDE\_OFFSET\_TYPING' = field(  default\_factory=lambda: {'car': {'v': 0, 'av': 0}, 'ball': {'v': 0, 'av': 0}}  )   class ModulesOutputMapping(dict):  @classmethod  def create\_random\_controls[T: ModulesOutputMapping](  cls: Type[T],  /,  requires\_grad: bool = True,  random\_jump: bool = False  ) -> T:  assert isinstance(requires\_grad, bool)  assert isinstance(random\_jump, bool)  output = cls({  key: [torch.rand((), requires\_grad=requires\_grad)] for key in CONTROLS\_KEYS  })  if not random\_jump:  output['controls.jump'] = [torch.tensor(0., dtype=torch.get\_default\_dtype())]  return output   @classmethod  def fromMAIGameState[T: ModulesOutputMapping](  cls: Type[T],  s: MAIGameState,  requires\_grad: bool = True  ) -> T:  d = dict()  vtt = cls.\_vector\_to\_tensors  rtt = cls.\_rotator\_to\_tensors  car\_position = vtt(s.car.position)  car\_velocity = vtt(s.car.velocity)  car\_rotation = rtt(s.car.rotation)  car\_angularVelocity = vtt(s.car.angularVelocity)  ball\_position = vtt(s.ball.position)  ball\_velocity = vtt(s.ball.velocity)  ball\_rotation = rtt(s.ball.rotation)  ball\_angularVelocity = vtt(s.ball.angularVelocity)  dead = torch.tensor(  float(s.dead),  requires\_grad=requires\_grad  )  boostAmount = torch.tensor(  float(s.boostAmount),  requires\_grad=requires\_grad  )  for name, value in [  ('state.car.position.x', [car\_position[0]]),  ('state.car.position.y', [car\_position[1]]),  ('state.car.position.z', [car\_position[2]]),  ('state.car.velocity.x', [car\_velocity[0]]),  ('state.car.velocity.y', [car\_velocity[1]]),  ('state.car.velocity.z', [car\_velocity[2]]),  ('state.car.rotation.pitch', [car\_rotation[0]]),  ('state.car.rotation.roll', [car\_rotation[1]]),  ('state.car.rotation.yaw', [car\_rotation[2]]),  ('state.car.angularVelocity.x', [car\_angularVelocity[0]]),  ('state.car.angularVelocity.y', [car\_angularVelocity[1]]),  ('state.car.angularVelocity.z', [car\_angularVelocity[2]]),  ('state.ball.position.x', [ball\_position[0]]),  ('state.ball.position.y', [ball\_position[1]]),  ('state.ball.position.z', [ball\_position[2]]),  ('state.ball.velocity.x', [ball\_velocity[0]]),  ('state.ball.velocity.y', [ball\_velocity[1]]),  ('state.ball.velocity.z', [ball\_velocity[2]]),  ('state.ball.rotation.pitch', [ball\_rotation[0]]),  ('state.ball.rotation.roll', [ball\_rotation[1]]),  ('state.ball.rotation.yaw', [ball\_rotation[2]]),  ('state.ball.angularVelocity.x', [ball\_angularVelocity[0]]),  ('state.ball.angularVelocity.y', [ball\_angularVelocity[1]]),  ('state.ball.angularVelocity.z', [ball\_angularVelocity[2]]),  ('state.dead', [dead]),  ('state.boostAmount', [boostAmount]),  ('MAIGameState', s)  ]:  d[name] = value  assert set(d.keys()) == set(STATE\_KEYS),\  f"{set(d.keys())}, {set(STATE\_KEYS)}"  return cls(d)   @classmethod  def fromTensor[T: ModulesOutputMapping](cls: Type[T], tensor: 'torch.Tensor') -> T:  assert tensor.shape == (26,), tensor  return cls([  ('state.car.position.x', [tensor[0]]),  ('state.car.position.y', [tensor[1]]),  ('state.car.position.z', [tensor[2]]),  ('state.car.velocity.x', [tensor[3]]),  ('state.car.velocity.y', [tensor[4]]),  ('state.car.velocity.z', [tensor[5]]),  ('state.car.rotation.pitch', [tensor[6]]),  ('state.car.rotation.roll', [tensor[7]]),  ('state.car.rotation.yaw', [tensor[8]]),  ('state.car.angularVelocity.x', [tensor[9]]),  ('state.car.angularVelocity.y', [tensor[10]]),  ('state.car.angularVelocity.z', [tensor[11]]),  ('state.ball.position.x', [tensor[12]]),  ('state.ball.position.y', [tensor[13]]),  ('state.ball.position.z', [tensor[14]]),  ('state.ball.velocity.x', [tensor[15]]),  ('state.ball.velocity.y', [tensor[16]]),  ('state.ball.velocity.z', [tensor[17]]),  ('state.ball.rotation.pitch', [tensor[18]]),  ('state.ball.rotation.roll', [tensor[19]]),  ('state.ball.rotation.yaw', [tensor[20]]),  ('state.ball.angularVelocity.x', [tensor[21]]),  ('state.ball.angularVelocity.y', [tensor[22]]),  ('state.ball.angularVelocity.z', [tensor[23]]),  ('state.dead', [tensor[24]]),  ('state.boostAmount', [tensor[25]]),  ])   @staticmethod  def \_vector\_to\_tensors(  vector: MAIVector,  requires\_grad: bool = True  ) -> list['torch.Tensor']:  return [torch.tensor(  i, requires\_grad=requires\_grad  ) for i in vector.to\_dict().values()]   @staticmethod  def \_rotator\_to\_tensors(  rotator: MAIRotator,  requires\_grad: bool = True  ) -> list['torch.Tensor']:  return [torch.tensor(  i, requires\_grad=requires\_grad  ) for i in rotator.to\_dict().values()]   def \_avg\_from\_dict(  self, name: str, requires\_grad: bool = True  ) -> 'torch.Tensor | None':  values: list[torch.Tensor] = self.get(name, [])  if not values:  return None  assert isinstance(values, list), values  assert all((isinstance(i, torch.Tensor) for i in values)), values  try:  tensors = torch.tensor(  values,  dtype=torch.float,  requires\_grad=requires\_grad  )  except RuntimeError:  print(values)  raise  result = tensors.mean() # type: ignore  return result   def has\_state(self) -> bool:  return STATE\_KEYS[0] in self   def has\_all\_state(self) -> bool:  return all((key in self for key in STATE\_KEYS))   def has\_controls(self) -> bool:  return CONTROLS\_KEYS[0] in self   def has\_all\_controls(self) -> bool:  return all((key in self for key in CONTROLS\_KEYS))   def has\_any\_controls(self) -> bool:  return any((key in self for key in CONTROLS\_KEYS))   def has\_reward(self) -> bool:  return REWARD\_KEY in self   def has\_powers(self) -> bool:  return POWERS\_LEY in self   @property  def reward(self) -> float:  assert self.has\_reward()  return self[REWARD\_KEY]   @reward.setter  def reward(self, reward: float) -> None:  assert isinstance(reward, float)  self[REWARD\_KEY] = reward   @property  def powers(self) -> torch.Tensor | None:  assert self.has\_powers()  return self[POWERS\_LEY]   @powers.setter  def powers(self, powers: torch.Tensor) -> None:  assert isinstance(powers, torch.Tensor)  assert len(powers.shape) == 1  # assert all(0 <= i.item() <= 1 for i in powers), powers  self[POWERS\_LEY] = powers   @property  def origin(self) -> 'MAIGameState | None':  return self.get('MAIGameState', None)   def is\_complete(self) -> bool:  return self.has\_state() and self.has\_controls()   def extract\_keys[T: ModulesOutputMapping](  self: T,  keys: Iterable[str],  requires\_grad: bool = True  ) -> T:  output: Mapping[str, list[torch.Tensor]] = dict()  for i in keys:  value = self.\_avg\_from\_dict(i, requires\_grad=requires\_grad)  if value is not None:  output[i] = [value]  else:  output[i] = [torch.tensor(0.0, requires\_grad=requires\_grad)]  return type(self)(output)   def extract\_controls[T: ModulesOutputMapping](  self: T,  requires\_grad: bool = True  ) -> T:  return self.extract\_keys(  keys=CONTROLS\_KEYS,  requires\_grad=requires\_grad  )   def extract\_state[T: ModulesOutputMapping](  self: T,  requires\_grad: bool = True  ) -> T:  return self.extract\_keys(  keys=STATE\_KEYS[:-1],  requires\_grad=requires\_grad  )   def getNormalizedNegative(self, key: str):  assert isinstance(key, str)  v = self.get(key, (0.0,))   return tanh(float(  sum(v)/len(v)  ))   def getNormalizedPositive(self, key: str):  assert isinstance(key, str)  v = self.get(key, (0.0,))  mean\_value = sum(v) / len(v)  # Clamp the value to avoid overflow  if mean\_value < -709: # This is a common threshold for float overflow  return 0.0  elif mean\_value > 709:  return 1.0  return 1 / (1 + exp(-mean\_value))   def get\_average(self, key: str, powers: dict[str, list[float]]) -> float:  weights = powers[key]  if not weights:  return 0  if key not in self:  return 0  assert isinstance(weights, list)  assert all(isinstance(i, float) for i in weights)  v: list[torch.Tensor] = self[key]  if len(v) == 1:  return v[0].item()  assert len(v) == len(weights), f"{key}: {self}, {powers}"  return float(np.average(v, weights=weights))   def toFloatControlsWithPowers(self, modules: list['ModuleBase']) -> FloatControls:  powers: dict[str, list[float]] = {  name: [] for name in CONTROLS\_KEYS  }  for module in modules:  for output\_value in module.output\_types:  if output\_value not in powers:  continue  if module.power < 0.1:  continue  powers[output\_value].append(module.power)   return FloatControls(  throttle = self.get\_average('controls.throttle', powers),  steer = self.get\_average('controls.steer', powers),  pitch = self.get\_average('controls.pitch', powers),  yaw = self.get\_average('controls.yaw', powers),  roll = self.get\_average('controls.roll', powers),  boost = self.get\_average('controls.boost', powers),  jump = self.get\_average('controls.jump', powers),  handbrake = self.get\_average('controls.handbrake', powers),  dodgeVertical = self.get\_average('controls.dodgeVertical', powers),  dodgeStrafe = self.get\_average('controls.dodgeStrafe', powers)  )   def toFloatControls(self) -> FloatControls:  return FloatControls(  throttle = self.getNormalizedNegative('controls.throttle'),  steer = self.getNormalizedNegative('controls.steer'),  pitch = self.getNormalizedNegative('controls.pitch'),  yaw = self.getNormalizedNegative('controls.yaw'),  roll = self.getNormalizedNegative('controls.roll'),  boost = self.getNormalizedPositive('controls.boost'),  jump = self.getNormalizedPositive('controls.jump'),  handbrake = self.getNormalizedPositive('controls.handbrake'),  dodgeVertical = self.getNormalizedNegative('controls.dodgeVertical'),  dodgeStrafe = self.getNormalizedNegative('controls.dodgeStrafe')  )   def toTensor(self, requires\_grad: bool = True) -> 'torch.Tensor':  values = [  sum(i)/len(i) if len(i) > 1 else i[0].item() for i in self.values()  ]  return torch.tensor(values, requires\_grad=requires\_grad)   def toDiscreteTensor(self, requires\_grad: bool = True) -> 'torch.Tensor':  values = []  for i in self.values():  if len(i) > 1:  v = sum(i)/len(i)  else:  v = float(i[0].item())  values.append(round(v))  return torch.tensor(values, requires\_grad=requires\_grad)   def randomize[T: ModulesOutputMapping](  self: T, requires\_grad: bool = True  ) -> T:  for key in self:  self[key] = [torch.rand((), requires\_grad=requires\_grad)]  return self   # # Constant variables based on data classes # ARENA\_SIZE = Vector(np.array((4096, 5120+880, 2044))) DENORMALIZATION\_VECTOR = Vector(np.array((  1,  ARENA\_SIZE.y / ARENA\_SIZE.x,  ARENA\_SIZE.z / ARENA\_SIZE.z ))) |

## Приложение Д - Достижения

## Приложение Е - Презентационный материал

s