

А. Д. ОБУХОВ, А. А. ВОЛКОВ, Д. В. ТЕСЕЛКИН,
Ю. В. НИКИТНИКОВ, М. А. ШИЛЬЦЫН

СИСТЕМНЫЙ АНАЛИЗ, ОБРАБОТКА ДАННЫХ И УПРАВЛЕНИЕ В ПРИЛОЖЕНИЯХ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА И ВИРТУАЛЬНОЙ РЕАЛЬНОСТИ



Тамбов
Издательский центр ФГБОУ ВО «ТГТУ»
2026

Министерство науки и высшего образования Российской Федерации

**Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Тамбовский государственный технический университет»**

**А. Д. ОБУХОВ, А. А. ВОЛКОВ, Д. В. ТЕСЕЛКИН,
Ю. В. НИКИТНИКОВ, М. А. ШИЛЬЦЫН**

**СИСТЕМНЫЙ АНАЛИЗ,
ОБРАБОТКА ДАННЫХ И УПРАВЛЕНИЕ
В ПРИЛОЖЕНИЯХ ИСКУССТВЕННОГО
ИНТЕЛЛЕКТА И ВИРТУАЛЬНОЙ РЕАЛЬНОСТИ**

Утверждено Ученым советом
ФГБОУ ВО «Тамбовский государственный технический университет»
в качестве учебного пособия для студентов бакалавриата направлений
09.03.01 «Информатика и вычислительная техника»
и 09.03.03 «Прикладная информатика»

Учебное электронное издание



Тамбов
Издательский центр ФГБОУ ВО «ТГТУ»
2026

УДК 004.89
ББК 16.6я73
С34

Рецензенты:

Доктор технических наук, профессор, профессор
Высшей школы технологий искусственного интеллекта СПбПУ
А. А. Большаков

Доктор технических наук, профессор, профессор
кафедры КИСМ ФГБОУ ВО «ТГТУ»
С. В. Карпушкин

- С34 **Системный** анализ, обработка данных и управление в приложениях искусственного интеллекта и виртуальной реальности [Электронный ресурс] : учебное пособие / А. Д. Обухов, А. А. Волков, Д. В. Теселкин, Ю. В. Никитников, М. А. Шильцын. – Тамбов : Издательский центр ФГБОУ ВО «ТГТУ», 2026. – 1 электрон. опт. диск (CD-ROM). – Системные требования : ПК не ниже класса Pentium IV ; RAM 512 Mb ; необходимое место на HDD 2,4 Mb ; Windows 7/8/10/11 ; дисковод CD-ROM ; мышь. – Загл. с экрана.
ISBN 978-5-8265-3030-6

Последовательно сформировано целостное представление о жизненном цикле AI/VR-систем: от формализации требований и проектирования архитектуры до сбора данных, построения моделей машинного обучения и решения задач управления, включая обучение с подкреплением. Рассмотрены методологии системного анализа, нотации и схемы (структурные, потоков данных, UML), современные движки и стандарты XR (Unity/Unreal, OpenXR), а также инструменты ИИ (Python, Keras, PyTorch, scikit-learn). Отдельные главы посвящены практикам предобработки и валидации данных, задачам классификации и регрессии, обработке телеметрии VR-сессий (позы, взаимодействия, логирование), применению нейросетей в мониторинге и автоматическом управлении, методам RL. Каждая глава содержит иллюстрации, полностью запускаемые примеры кода на Python и C#, контрольные вопросы.

Предназначено для студентов бакалавриата направлений 09.03.01 «Информатика и вычислительная техника» и 09.03.03 «Прикладная информатика». Может использоваться как основной учебный текст для дисциплин «Системный анализ», «Искусственный интеллект», «Технологии виртуальной/дополненной реальности», а также как практическое руководство при выполнении курсовых и проектных работ.

УДК 004.89
ББК 16.6я73

*Все права на размножение и распространение в любой форме остаются за разработчиком.
Нелегальное копирование и использование данного продукта запрещено.*

ISBN 978-5-8265-3030-6

© Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Тамбовский государственный технический
университет» (ФГБОУ ВО «ТГТУ»), 2026

ВВЕДЕНИЕ

В современном мире технологии искусственного интеллекта (ИИ) и виртуальной реальности (VR) все более тесно переплетаются, создавая инновационные приложения в самых разных областях – от обучения и промышленных симуляторов до игр и сервисов. Инженеры и программисты, разрабатывающие такие системы, должны обладать знаниями как в области системного анализа программных комплексов, так и в методах подготовки данных и применения методов ИИ для решения задач управления. Данное учебное пособие призвано помочь студентам разобраться в фундаментальных принципах системного анализа, особенностях обработки данных в приложениях ИИ и VR, а также освоить основы управления сложными системами на базе нейронных сетей, включая методы обучения с подкреплением. Материал ориентирован на бакалавров и магистров, обучающихся по направлениям информационные технологии, и учитывает применение современных технологий (Python, библиотеки *Keras/PyTorch*, игровой движок *Unity* с языком *C#*, *Unreal Engine Blueprints* и др.).

Структура учебного пособия включает следующие разделы:

1. Системный анализ приложений искусственного интеллекта и виртуальной реальности – рассмотрение методологий разработки таких систем, формализация требований и архитектуры, обзор основных технологий (игровые движки для VR, библиотеки машинного обучения для ИИ).

2. Обработка данных в приложениях искусственного интеллекта – методы предварительной подготовки данных для обучения моделей, а также решение типичных задач машинного обучения: классификации и регрессии.

3. Обработка данных в приложениях виртуальной реальности – сбор и обработка данных, генерируемых в VR-тренажерах и системах (двигательная активность пользователя, взаимодействие со сценой и объектами, применение стандартов вроде OpenXR для кросс-платформенного доступа к устройствам).

4. Решение задач управления в приложениях искусственного интеллекта и виртуальной реальности – применение нейронных сетей для задач управления и мониторинга, основы обучения с подкреплением, примеры задач управления в виртуальных тренажерах и подходы к их решению.

Учебное пособие предназначено для студентов бакалавриата по направлениям 09.03.01 «Информатика и вычислительная техника» и 09.03.03 «Прикладная информатика».

1. СИСТЕМНЫЙ АНАЛИЗ ПРИЛОЖЕНИЙ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА И ВИРТУАЛЬНОЙ РЕАЛЬНОСТИ

Системный анализ представляет собой совокупность методических подходов к **проектированию сложных систем**. В контексте приложений, использующих технологии искусственного интеллекта или виртуальной/дополненной реальности, системный анализ позволяет формализовать требования и спланировать архитектуру будущей системы еще до начала разработки. Рассмотрим основные шаги и методы системного анализа применительно к AI/VR-приложениям.

1.1. МЕТОДОЛОГИИ РАЗРАБОТКИ И ЖИЗНЕННЫЙ ЦИКЛ AI/VR-СИСТЕМ

При создании приложений ИИ и VR используются общие методологии разработки программного обеспечения (жизненный цикл проекта), адаптированные под специфику домена. К классическим подходам относятся водопадная модель, итеративная и инкрементальная разработка, гибкие методологии (Agile) и др. В случае VR- и AI-проектов часто оправдан итеративный подход с быстрым прототипированием, поскольку важно учитывать опыт пользователей и проводить испытания на реальных данных (рис. 1.1).

Как правило, процесс начинается с этапа **концептуального проектирования**, на котором формулируется идея и цели приложения. Важно определить, **какую проблему решает приложение и для какой аудитории оно предназначено** – ответы на эти вопросы задают направление всей разработки [1].

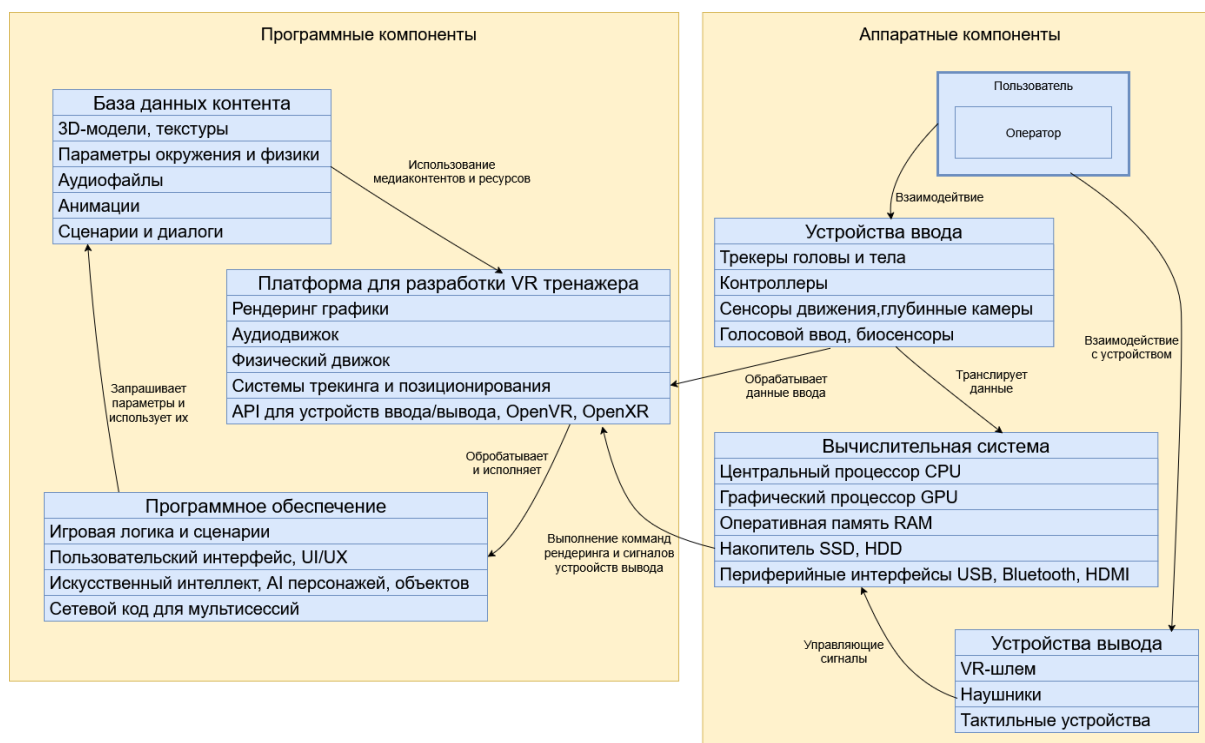


Рис. 1.1. Общая структурная схема системы виртуальной реальности, включающая аппаратные компоненты (пользователь, устройства ввода/вывода, вычислительная платформа) и программные компоненты (VR-движок/платформа, программное обеспечение виртуальной среды, база данных контента). В приложениях ИИ подобные схемы также отражают потоки данных от источников через модели ИИ к выходным действиям

Например, при системном анализе VR-тренажера для обучения персонала нужно четко определить учебные цели симуляции, а при анализе задачи для ИИ – например, системы рекомендации или распознавания образов – необходимо понять бизнес-цель модели и доступные данные. На данном этапе составляется **техническое задание**: описываются функциональные требования (что система должна делать) и ограничения (по времени отклика, ресурсам, совместимости с оборудованием и т.д.) [1].

Далее, **планирование архитектуры** включает выбор платформы и технологий. Для VR-систем важен выбор аппаратной платформы и движка (например, *Unity* или *Unreal Engine*) исходя из требуемого качества графики, поддержки нужных VR-устройств и квалификации команды разработчиков [2]. Для AI-приложений планируется инфраструктура для сбора и

хранения данных, инструменты для обучения моделей (фреймворки *TensorFlow*, *PyTorch*, *Keras* и др.), а также среды для деплоя моделей (серверные платформы, облачные сервисы и т.п.) [2].

После концепции и архитектуры следует этап **проектирования и разработки**. Он тесно связан с системным анализом: на этом шаге продумываются конкретные модули системы, их взаимодействие, алгоритмы работы. В случае VR-приложения происходит разработка 3D-сцены, логики взаимодействий и интерфейса; при этом требуются совместные усилия программистов, дизайнеров и 3D-художников. В случае AI-систем – подготовка данных, выбор модели и реализация обучения. Здесь системный анализ помогает разбить сложную систему на компоненты (например, модуль сбора данных, модуль обработки/машинного обучения, модуль пользовательского интерфейса) и определить четкие интерфейсы между ними.

Наконец, обязательными этапами являются **тестирование и развертывание**. Для VR-приложений проводится тестирование на разных устройствах (разные VR-гарнитуры) и проверяется удобство использования, производительность (число кадров в секунду, отсутствие задержек). Для AI-моделей, помимо функционального тестирования, важно проверить качество модели на отложенных данных, убедиться в отсутствии критичных ошибок обобщения. После успешного тестирования система разворачивается в целевом окружении – будь то публикация VR-приложения в магазине приложений или деплой модели на сервер для интеграции с основным приложением.

После релиза система переходит на этап **поддержки и развития**, который включает мониторинг производительности и стабильности работы. Для AI-компонентов критически важен постоянный мониторинг качества предсказаний и переобучение моделей в случае «дрейфа данных». Параллельно формируется план развития продукта для разработки новых функций и улучшения пользовательского опыта на основе аналитики (табл. 1.1).

1.1. Основные этапы создания VR/AR-систем

Этап	Основные действия	Ключевые результаты
1. Концептуальное проектирование	– Формулировка идеи и целей; – определение целевой аудитории; – анализ решаемой проблемы; – составление технического задания	Техническое задание, функциональные требования, ограничения
2. Планирование архитектуры	– Выбор платформы и технологий; – для VR: выбор движка и аппаратной платформы; – для AI: планирование инфраструктуры данных; – подбор инструментов разработки	Архитектурная схема, список технологий, план инфраструктуры
3. Проектирование и разработка	– Проектирование модулей системы; – разработка 3D-сцен и логики (VR); – подготовка данных и обучение моделей (AI); – создание пользовательского интерфейса	Рабочие модули системы, обученные модели, прототип приложения
4. Интеграция и тестирование	– Интеграция компонентов системы; – тестирование на разных устройствах (VR); – проверка качества моделей (AI); – тестирование производительности	Отчет о тестировании, метрики качества, список исправлений
5. Развертывание и запуск	– Публикация в магазинах приложений (VR); – деплой моделей на сервер (AI); – настройка мониторинга; – подготовка документации	Работающая система, документация, план поддержки
6. Поддержка и развитие	– Сбор пользовательского фидбека; – мониторинг работы системы; – обновление и улучшение моделей (AI); – оптимизация производительности	План обновлений, roadmap развития, метрики удовлетворенности

Кратко жизненный цикл AI/VR-системы можно свести к следующим этапам:

- **Анализ и определение требований** – цель приложения, задачи пользователей, критерии успеха.
- **Выбор платформ и технологий** – решения по аппаратной части (тип VR-гарнитуры, контроллеры, датчики) и программной (игровой движок, ML-фреймворки, языки программирования).
- **Проектирование системы** – разбивка на подсистемы, проработка архитектурных схем, алгоритмов функционирования.
- **Разработка и прототипирование** – создание компонентов, написание кода, наращивание функциональности.
- **Тестирование (unit, интеграционное, пользовательское)** – выявление и исправление ошибок, улучшение UX, оптимизация производительности.
- **Внедрение и сопровождение** – развертывание для конечных пользователей, мониторинг работы, сбор обратной связи, дальнейшие обновления.

Отметим, что для VR-разработки существуют рекомендации по организации процесса: начиная от планирования пространства разработки (360° дизайн, учет комфорта пользователя, во избежание укачивания) до оптимизации 3D-контента для разных устройств. Особой сложностью является разработка автономных VR-приложений для систем, работающих без подключения к ПК, что требует тщательной оптимизации производительности, а также создания адаптивного контента для различной вычислительной мощности устройств.

Для AI-проектов существуют свои стандарты (например, **CRISP-DM** – Cross-Industry Standard Process for Data Mining), предполагающие пошаговый процесс: анализ предметной области, анализ данных, подготовка данных, моделирование, оценка, внедрение.

1.2. ФОРМАЛИЗАЦИЯ ТРЕБОВАНИЙ И ПРОЕКТИРОВАНИЕ: СХЕМЫ И АЛГОРИТМЫ

Важной частью системного анализа является **формализация** – представление требований и поведения системы в виде строгих моделей, понятных как людям, так и, потенциально, компьютеру. Здесь на помощь приходят различные схемы и нотации:

- **Блок-схемы алгоритмов.** Для описания логики функционирования отдельных компонентов полезно использовать блок-схемы, где шаги алгоритма представлены графически (операции, решения, циклы). Например, алгоритм обработки входного сигнала от VR-контроллера или последовательность действий агента ИИ можно изобразить как блок-схему, что облегчает понимание и поиск ошибок в логике до написания кода (рис. 1.2).

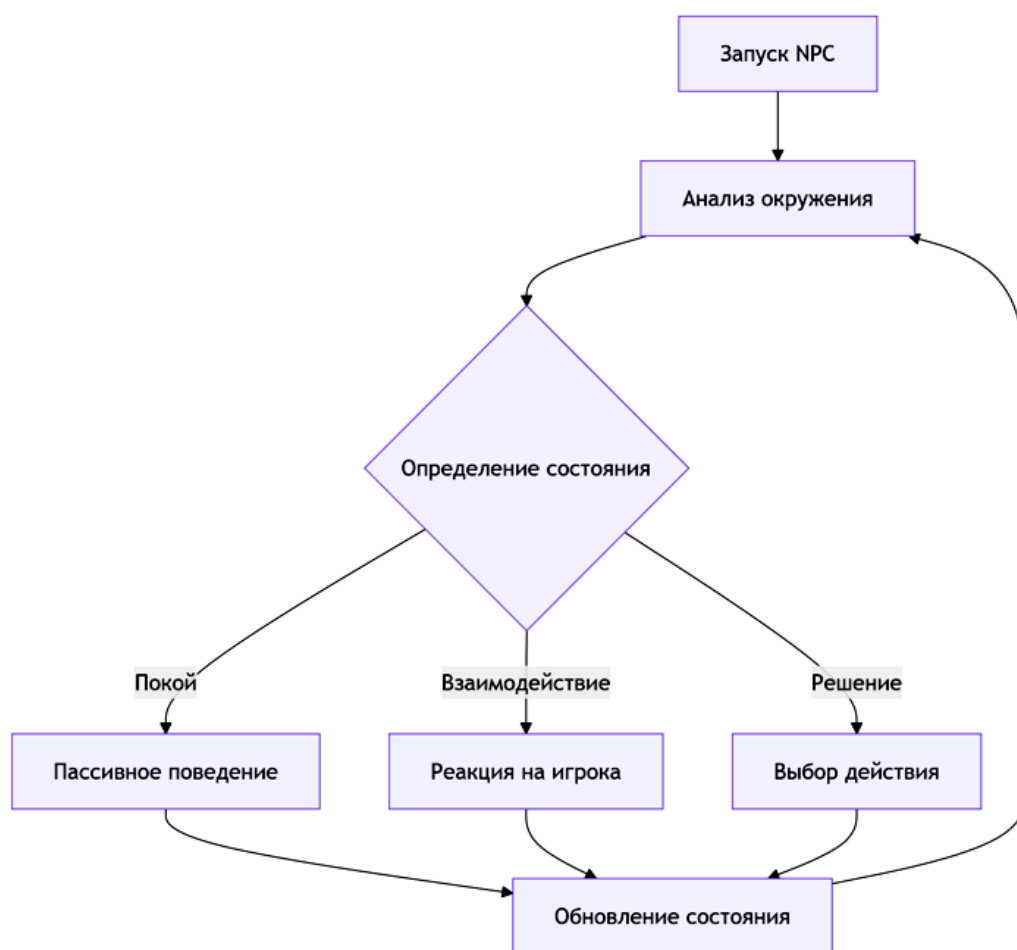


Рис. 1.2. Блок-схема алгоритма работы AI-модели для агента ИИ в VR

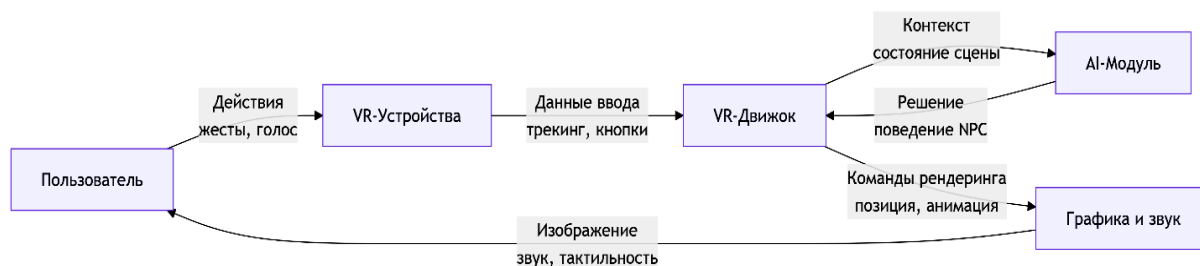


Рис. 1.3. Диаграмма потоков данных между компонентами системы VR с AI

- **Диаграммы потоков данных.** Такие диаграммы (DFD, Data Flow Diagrams) показывают, как данные перемещаются между частями системы. В приложении III диаграмма потоков данных может отразить путь данных от источника (сенсора, базы, пользователя) через этапы предобработки, затем через модель машинного обучения к выходному действию или решению. В VR-приложении можно показать, как данные от датчиков позиции и ориентации пользователя поступают в движок, как обновляется сцена и как генерируются выходные визуальные и аудио-сигналы (рис. 1.3).

- **UML-диаграммы.** Язык UML (Unified Modeling Language) предоставляет целый набор видов диаграмм для проектирования систем. В контексте AI/VR-приложений могут использоваться диаграммы случаев использования (описывают, какие действия могут выполнять пользователи и система), диаграммы классов (структура программных классов и их связи – актуально, например, для описания классов игрового приложения или классов, реализующих нейросеть), диаграммы последовательностей (динамика взаимодействия объектов во времени), состояния (например, состояния VR-тренажера: меню, запуск симуляции, пауза, завершение – и переходы между ними) (рис. 1.4).

- **Структурные схемы системы.** В разделе 1.1 (см. рис. 1.1) мы уже приводили пример общей структурной схемы VR-системы. Такие схемы, по сути, являются результатом системного анализа: они демонстрируют компоненты (модули) системы и связи между ними.

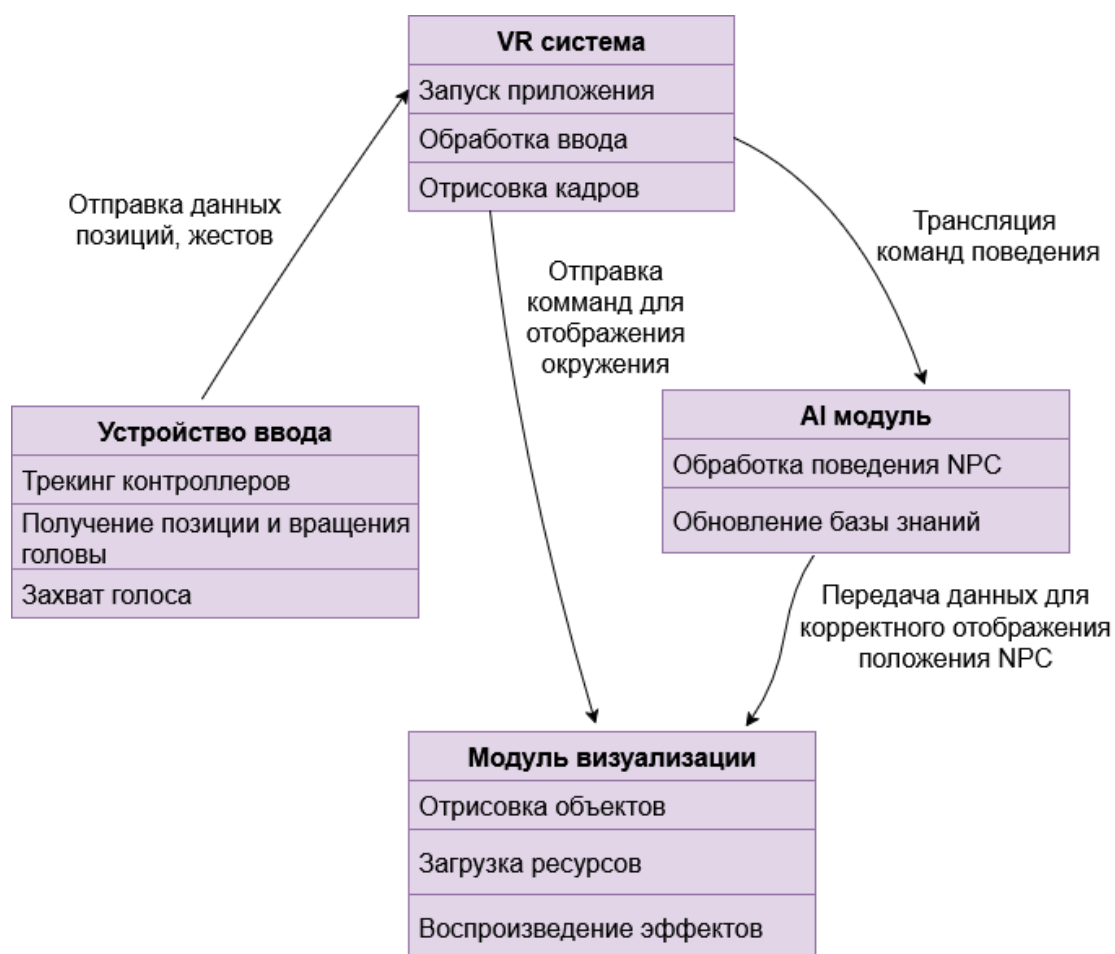


Рис. 1.4. UML-диаграмма классов системы взаимодействия VR с AI

Например, структурная схема приложения дополненной реальности может включать подсистемы: *Сбор данных с камеры*, *Обработка изображения (компьютерное зрение)*, *Отображение доп. объектов*, *База данных контента*, *Интерфейс пользователя* – соединенные потоками данных (видео, координаты, команды и т.д.). Для AI-системы, решающей бизнес-задачу, структурная схема может показать модуль получения данных, модуль обучения модели, модуль выдачи предсказаний и модуль мониторинга качества.

- **Материалы технического задания (ТЗ).** Помимо графических схем, формализация включает подготовку текстовых спецификаций: описание функциональных требований, нефункциональных требований (производительность, безопасность, совместимость), сценариев использования (use cases), а также, при необходимости, математических моделей.

Например, если VR-тренажер моделирует физический процесс, в ТЗ прописываются формулы или ссылки на модели, которые будут реализованы (например, законы физики, применимые в симуляции). Если используется ИИ-модель, то в спецификации можно указать метрики качества, которые она должна достичь (например, точность классификации не ниже 90% на тестовых данных).

Формализация помогает **минимизировать неоднозначности**. Четко прорисованные диаграммы и описанные алгоритмы позволяют всем членам команды (и заказчику) одинаково понимать, как должна работать система. Более того, это облегчает последующую реализацию: фактически разработка превращается в процесс кодирования уже продуманной схемы.

Пример: допустим, мы проектируем систему **виртуальной реальности для обучения пожарных**. Требования – имитация различных ситуаций пожара, проверка навыков пользователя по правильности действий. Сначала, системный аналитик описывает основные случаи использования: *«Пользователь надевает гарнитуру VR и попадает в виртуальное здание. Возникает очаг возгорания. Пользователь должен найти огнетушитель и потушить пожар»*. Далее строится структурная схема: модуль *Сценарий и логика обучения* (генерирует события – появление огня, контроль времени), модуль *Взаимодействие пользователя* (отслеживает движения, действия с виртуальными объектами), модуль *Физическая симуляция* (распространение огня, задымленность), модуль *Обратная связь* (оценка действий, подсказки) и т.д., все это на движке Unity с использованием SDK VR. Затем алгоритмы: например, **алгоритм оценки действий** – шаги:

- 1) определить, взял ли пользователь правильное средство тушения;
- 2) проверить, направлен ли огнетушитель на базу пламени;
- 3) при нажатии кнопки – уменьшать объем огня и рассчитывать оставшийся ресурс огнетушителя;

4) если пожар потушен – фиксировать успех, иначе по истечении времени – зафиксировать неудачу и сгенерировать вывод о допущенных ошибках.

Такой алгоритм можно прописать текстово или блок-схемой. В итоге, имея все формальные описания, команда приступает к реализации, сведя к минимуму риск, что что-то упустят или реализуют не так.

1.3. ОСНОВНЫЕ ТЕХНОЛОГИИ: ИГРОВЫЕ ДВИЖКИ, ФРЕЙМВОРКИ И БИБЛИОТЕКИ ИИ

Разработка приложений VR и ИИ невозможна без использования мощных современных инструментов. Рассмотрим основные из них:

- **Игровые движки для VR/AR.** Лидирующими платформами разработки интерактивных 3D-приложений (включая виртуальную реальность) сегодня являются **Unity** и **Unreal Engine**. *Unity* популярен как среди новичков, так и среди профессионалов благодаря сравнительно невысокому порогу входа, огромному количеству готовых компонентов (asset store) и поддержке множества платформ [6]. Он хорошо подходит для прототипирования и инди-проектов VR, имеет богатые возможности по работе с VR-гарнитурами (через встроенные инструменты XR или плагины типа SteamVR). *Unreal Engine* славится реалистичной графикой и часто используется в высокобюджетных проектах, где качество визуализации критично – например, в AAA-играх VR или сложных промышленных симуляциях. Однако Unreal несколько сложнее в освоении, особенно для начинающих, и требует больше мощностей для комфортной работы [8].

Unreal Engine особенно выделяется в создании проектов с фотореалистичной графикой благодаря таким технологиям, как Virtual Shadow Maps и Temporal Super Resolution. Система Blueprint Visual Scripting позволяет быстро создавать прототипы без глубоких знаний программирования, что ускоряет разработку. Epic Games активно развивает экосистему

MetaHuman для создания реалистичных цифровых людей, что критически важно для социальных VR-приложений. Особого внимания заслуживает Epic Games Fab – облачная платформа, предоставляющая инструменты для совместной работы, хранения контента и публикации проектов. Fab интегрируется с Unreal Engine, предлагая мощные возможности для управления цифровыми активами, многопользовательской разработки и аналитики, что особенно ценно для распределенных команд, работающих над сложными VR-проектами.

Оба движка постоянно развиваются: к примеру, *Unreal Engine 5* предлагает революционные технологии Nanite и Lumen для детализации и освещения. Также *Unity* интегрирует все новые возможности рендеринга, физики и поддержки XR-устройств.

Помимо этих двух, существуют и другие движки и фреймворки:

- *Godot Engine* – открытый движок, также поддерживающий VR через модули; привлекает легковесностью и открытым исходным кодом, но менее распространен.
- *CryEngine*, *Source*, *Amazon Lumberyard* – мощные движки, однако в сфере VR применяются реже (отчасти из-за меньшей документации или специализированности под определенные задачи).

Для мобильной AR широко используются специализированные SDK:

- **ARCore** (Android) и **ARKit** (iOS) для дополненной реальности, которые можно интегрировать и в Unity/Unreal, и в нативные мобильные приложения.
- **WebXR** – стандарт для браузерной VR/AR: позволяет создавать простые VR/AR-приложения, которые работают прямо в веб-браузере, с помощью WebGL и JavaScript-фреймворков (например, *three.js*, *A-Frame*).

При работе с VR-оборудованием важен еще один стандарт – **OpenXR**. Это универсальный API, разработанный консорциумом Khronos, который позволяет приложению взаимодействовать с различными

VR/AR- устройствами через единый интерфейс. Ранее разработчикам приходилось писать код отдельно под каждую платформу (Oculus SDK, SteamVR, Windows MR и др.), что приводило к фрагментации [8]. OpenXR решает эту проблему, выступая прослойкой: достаточно один раз реализовать поддержку OpenXR, и приложение будет совместимо с любым шлемом или контроллером, для которого есть соответствующий runtime. К 2025 году OpenXR получил широкую поддержку: его реализовали Oculus/Meta, SteamVR, Microsoft, HTC, Pico, Valve и многие другие производители, что существенно упрощает разработку кросс-платформенных VR-приложений. В рамках Unity и Unreal Engine поддержка OpenXR также доступна «из коробки» (начиная с Unity 2020+ и Unreal 4.27+). Например, разработчик может использовать OpenXR-плагин в Unity и получать доступ к позициям/ориентациям контроллеров, отслеживанию рук и др., не беспокоясь о конкретном устройстве. Мы подробнее коснемся сбора данных в VR-приложении в разделе 3, включая пример кода использования OpenXR/Unity API [5].

- **Языки программирования.** В VR-разработке основными языками являются C# (в Unity) и C++ (в Unreal Engine). Unity использует C# для скриптов поведения объектов, написания логики; благодаря сборщику мусора и высокой абстракции C# позволяет быстро писать код, не думая о низкоуровневых деталях памяти (что удобно для быстрого прототипирования). Unreal Engine, напротив, предоставляет как C++ API, так и визуальный скриптовый язык *Blueprints*. Blueprints – это система **визуального программирования**, где разработчик соединяет ноды (блоки функций, событий и переменных) в графическом редакторе, фактически создавая логику игры/симуляции без написания кода (рис. 1.5). Это снижает порог входа для новичков и дизайнеров, позволяя создавать интерактивность, не владея синтаксисом языка. Тем не менее для сложных задач и оптимизации в Unreal обычно прибегают к чистому C++, который обеспечивает максимальную производительность.

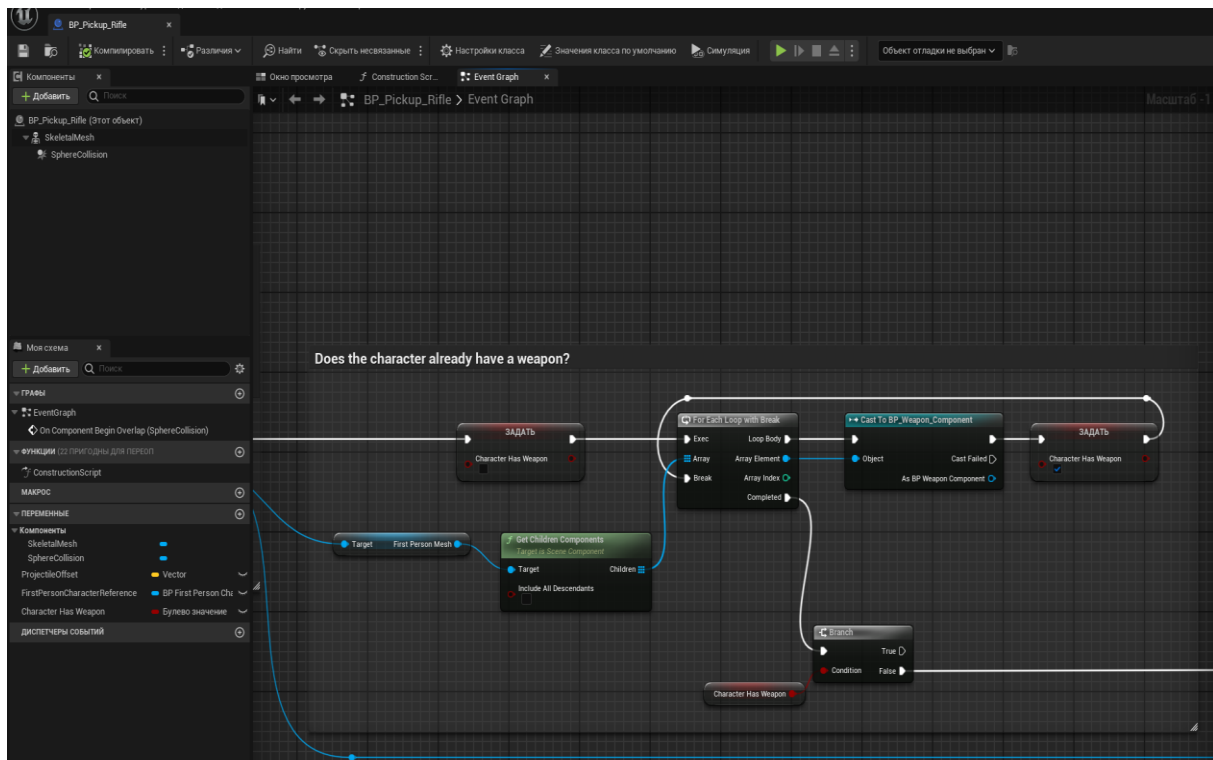


Рис 1.5. Использование визуального программирования (Blueprints) в Unreal Engine

В приложениях ИИ доминируют другие языки – прежде всего **Python**, благодаря богатой экосистеме библиотек для научных вычислений и машинного обучения. На Python написаны популярные ML-библиотеки (*TensorFlow*, *PyTorch*, *scikit-learn*, *pandas* и др.). Поэтому разработчики AI-систем чаще всего прототипируют и обучают модели именно на Python. Однако, если требуется встроить ИИ-модель, например в VR-симулятор на Unity, то возможны разные подходы: либо обучить модель в Python и затем экспортировать ее (например, в формат ONNX, TensorFlow Lite и т.п.) для использования в C#/C++ среде, либо вызывать Python-код отдельно (через сетевое взаимодействие, если архитектура это позволяет). Например, Unity предлагает пакет **ML-Agents**, где среда симуляции реализована на C#, а обучение агентов (с использованием обучения с подкреплением) ведется Python-сценарием, взаимодействующим с игрой. Таким образом, при интеграции AI и VR знание как C#/C++, так и Python становится весьма ценным.

- **Библиотеки и фреймворки искусственного интеллекта.** Если проект включает компоненты ИИ (например, распознавание образов, принятие решений, оптимизация), разработчики используют готовые библиотеки, реализующие алгоритмы машинного обучения и нейросети. К основным относят:

- **TensorFlow** – открытый фреймворк для машинного обучения, особенно глубокого обучения. Позволяет строить и обучать нейронные сети практически любой архитектуры (от простых полносвязных до сверточных, рекуррентных, трансформеров и т.д.). TensorFlow часто используется в связке с высокоуровневым API Keras (ранее самостоятельным, а теперь интегрированным), который облегчает создание и обучение нейросетей.

- **PyTorch** – фреймворк, альтернативный TensorFlow, популярный среди исследователей и разработчиков благодаря удобству и более “pythonic” стилю. PyTorch позволяет определять модели динамически (что облегчает отладку), имеет широкую поддержку сообществом. Многие современные научные разработки (NLP, CV) публикуются с реализацией на PyTorch.

- **Scikit-learn** – библиотека Python для классических методов машинного обучения (решающие деревья, случайный лес, SVM, регрессии, кластеризация и др.). Хорошо подходит для относительно простых задач и табличных данных; предоставляет единый интерфейс для подготовки данных и обучения моделей.

- **Pandas, NumPy, SciPy** – библиотеки для работы с данными: Pandas для табличных данных (DataFrame), NumPy для эффективных операций с массивами, SciPy – набор научных алгоритмов (оптимизация, статистика и пр.). В AI-проектах они используются на этапе анализа и предварительной обработки данных.

- **OpenCV** – библиотека компьютерного зрения, позволяющая обрабатывать изображения и видео (детекция объектов, фильтрация, трансформации и пр.). Может применяться как отдельно, так и совместно с нейросетями (например, для предобработки видео в VR или анализа изображений).

- **Фреймворки для обучения с подкреплением:** *OpenAI Gym/Gymnasium* – стандартный API для сред обучения агентов; *Stable Baselines3* – набор готовых реализаций RL-алгоритмов; *Rllib* и др. Они облегчают разработку систем, где агент должен сам учиться действовать, пробуя взаимодействовать с окружающей средой (подробнее в разделе 4).

- **Специализированные библиотеки:** например, для обработки естественного языка (NLTK, spaCy, transformers от HuggingFace) – актуальны, если в AI-компоненте нужно понимать команды пользователя или генерировать текст; или для аудио (Librosa, Soundfile) – если проект связан с голосовым управлением или анализом звука в VR.

- **Аппаратное обеспечение.** Отдельно стоит упомянуть об аппаратной составляющей. VR-приложения требуют наличия VR-гарнитуры (очки/шлем виртуальной реальности) и, зачастую, контроллеров или других устройств ввода. Популярные гарнитуры на 2025 год: *Quest 2/3*, *HTC Vive Pro*, *Valve Index*, *HP Reverb G2*, *Sony PlayStation VR2*, а также множество автономных и мобильных решений. Каждая имеет свои характеристики (разрешение, угол обзора, наличие трекинга рук и позиционного трекинга и т.п.), что влияет на разработку (например, под автономные гарнитуры нужно сильнее оптимизировать графику). При системном анализе проекта VR необходимо заложить требования к целевому оборудованию: будет ли приложение работать на автономных (standalone) устройствах или только с подключением к ПК, нужен ли дополнительный **трекинг движений** (например, системы типа OptiTrack для полного тела) или достаточно встроенных средств гарнитуры. В учебных VR-тренажерах иногда исполь-

зуются специальные контроллеры-симуляторы оборудования (например, муляж штурвала самолета с датчиками) – их тоже надо учитывать в системе.

Для AI-систем важной частью инфраструктуры являются **вычислительные ресурсы**: серверы или облачные инстансы с GPU/TPU для ускорения обучения нейросетей, устройства для развертывания моделей (может потребоваться отдельный сервер с мощным CPU/GPU, если модель должна отвечать в реальном времени многим пользователям). Также могут использоваться аппаратные ускорители, такие как NVIDIA Jetson (для внедрения ИИ на роботах или edge-устройствах), или нейронные сопроцессоры в смартфонах для мобильных AI-приложений.

- **Кросс-платформенные решения и другие языки.** При этом для разных платформ и языков программирования существуют специализированные решения.

- **OpenNN (C++)** – высокопроизводительная библиотека для глубокого обучения, ориентированная на скорость вычислений и эффективное использование ресурсов. Особенно востребована в задачах, требующих максимальной производительности.

- **ML.NET (C#)** – комплексный фреймворк от Microsoft для .NET-разработчиков, предоставляющий инструменты для создания и развертывания ML-моделей непосредственно в экосистеме Microsoft.

- **TensorFlow.js (JavaScript)** – полнофункциональная версия TensorFlow для браузеров и Node.js, позволяющая запускать и тренировать ML-модели непосредственно в веб-среде, что открывает возможности для браузерных AI-приложений.

- **Deeplearning4j (Java/Kotlin)** – распределенная библиотека глубокого обучения для JVM-экосистемы, поддерживающая распределенные вычисления и интеграцию с большими данными через Apache Hadoop и Spark.

- **Core ML (Swift/Objective-C)** – оптимизированный фреймворк Apple для seamless-интеграции ML-моделей в приложения iOS, macOS, watchOS и tvOS с автоматической оптимизацией под оборудование Apple.

- **MediaPipe (C++/Python)** – кросс-платформенный фреймворк Google для построения сложных конвейеров обработки мультимедийных данных с поддержкой ML-моделей, особенно эффективен для задач компьютерного зрения и аудиообработки.

- **ONNX Runtime** – кросс-платформенная среда выполнения для моделей в формате ONNX, поддерживающая различные языки программирования и аппаратные ускорители, что обеспечивает переносимость моделей между разными фреймворками.

Выбор конкретной библиотеки зависит от платформы целевого приложения, требований к производительности и экспертизы команды разработчиков. Для встраивания AI в мобильные приложения часто выбирают Core ML или ML.NET, тогда как для высокопроизводительных серверных решений предпочтительнее C++-библиотеки вроде OpenNN. При этом Python остается наиболее популярным языком для исследований и быстрого прототипирования благодаря богатой экосистеме и простоте интеграции с VR/AR-системами.

Подводя итог, системный анализ AI/VR-приложения включает выбор и обоснование технологий, на которых оно будет базироваться. Так, **современные виртуальные тренажеры** опираются на целый ряд передовых технологий: от VR/AR для погружения пользователя в цифровое пространство до нейронных сетей и машинного обучения для решения задач анализа, обработки информации и управления процессами. Задача инженера – из этого технологического арсенала выбрать оптимальный набор под требования проекта, учесть совместимость компонентов и выстроить целостную архитектуру системы.

КОНТРОЛЬНЫЕ ВОПРОСЫ К РАЗДЕЛУ 1

1. В чем состоит цель этапа системного анализа при разработке программного приложения?
2. Перечислите основные шаги жизненного цикла разработки VR/AR-приложения. Какие особенности имеются на этапах тестирования таких систем?
3. Какие методологии проектирования ПО Вы знаете? Чем характеризуется итеративный подход и почему он полезен в VR/AI проектах?
4. Что такое формализация требований? Какие диаграммы и схемы используются для формализации архитектуры и алгоритмов системы?
5. Какие компоненты можно выделить в структурной схеме типичного VR-приложения? Опишите на примере.
6. Для чего используются блок-схемы и диаграммы состояний при проектировании систем?
7. Назовите два самых популярных игровых движка для разработки VR-приложений. В чем их ключевые отличия?
8. Какие существуют способы интеграции модели машинного обучения в приложение виртуальной реальности?
9. Что такое OpenXR и какую проблему разработчиков VR/AR он решает?
10. Перечислите три библиотеки Python, которые наиболее часто применяются для реализации искусственного интеллекта в приложениях. Какого рода задачи они помогают решать?
11. Какие этапы жизненного цикла AI/VR-систем требуют наибольшего внимания к производительности и почему?
12. Как выбор игрового движка (Unity vs Unreal Engine) влияет на архитектуру VR-приложения?
13. В чем преимущества использования OpenXR по сравнению с разработкой под конкретные устройства VR?

14. Какие инструменты и методы используются для формализации требований к AI-компонентам системы?

15. Как согласуются этапы разработки AI-моделей и VR-приложений в рамках одного проекта?

16. Какие риски возникают при интеграции AI-моделей в реальном времени в VR-приложения?

17. Какую роль играет системный анализ в обеспечении кросс-платформенности VR-приложения?

18. Какие методы используются для оценки производительности и качества AI-компонентов на этапе тестирования?

19. Как учитываются ограничения аппаратного обеспечения (например, автономных VR-шлемов) при проектировании архитектуры?

20. Какие стандарты и методологии (кроме CRISP-DM) применяются в AI-проектах и как они сочетаются с этапами разработки VR?

2. ОБРАБОТКА ДАННЫХ В ПРИЛОЖЕНИЯХ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

Любое приложение, основанное на методах искусственного интеллекта (машинного обучения), в своей работе опирается на **данные**. Качество этих данных и умение правильно их обработать во многом определяют успех модели. Согласно распространенной оценке, **подготовка данных может занимать до 80% времени**, затрачиваемого на проект машинного обучения. В данном разделе мы рассмотрим, какие этапы включает предобработка данных и как решаются основные типы задач ИИ – **классификация и регрессия** – на подготовленных данных [3].

2.1. ПРЕДВАРИТЕЛЬНАЯ ОБРАБОТКА ДАННЫХ ДЛЯ МОДЕЛЕЙ МАШИННОГО ОБУЧЕНИЯ

Предварительная обработка данных (data preprocessing) – это этап, на котором сырые, необработанные данные приводятся в удобный для анализа вид. Реальные данные часто содержат ошибки, пропуски, шум, имеют разные масштабы измерений; напрямую «кормить» их алгоритму машинного обучения нельзя без потери качества. Цель предобработки – очистить и трансформировать данные, выделить информативные признаки, тем самым повысив способность модели обучаться и давать точные предсказания.

Основные шаги предобработки данных можно обобщить следующим образом [3]:

- **Проверка и очистка данных:** на этом этапе анализируют набор данных на наличие аномалий. Выявляются и устраняются **проблемы**

качества данных – дубликаты записей, противоречивые или заведомо ошибочные значения, выбросы (аномальные значения, сильно отличающиеся от остальных), а также **пропуски** (отсутствующие значения признаков). Для работы с пропущенными значениями применяют разные стратегии: удалить неполные записи, заполнить средним, медианой или специальным маркером и др. – выбор зависит от доли пропусков и природы данных. Для обработки выбросов используют методы межквартильного размаха (IQR), z-score, или изолирующий лес (Isolation Forest), а в бизнес-контексте – их согласование с экспертами предметной области. Категориальные данные проверяются на нестандартные кодировки и опечатки, например, «Москва», «мск» и «Мск» приводятся к единому формату. Временные ряды анализируются на наличие пропусков в хронологической последовательности. Результатом этапа становится чистый и согласованный датасет, готовый для дальнейшего преобразования.

- **Трансформация и приведение данных:** включает операции по преобразованию формата и структуры данных. Например, **кодирование категориальных признаков** – преобразование текстовых меток в числа (с помощью *one-hot encoding*, порядкового кодирования и т.п.), **нормализация числовых признаков** – приведение различных числовых признаков к сопоставимому масштабу (например, путем вычитания среднего и деления на стандартное отклонение или приведения диапазона к $[0,1]$). Нормировка важна, чтобы ни один признак не доминировал из-за размерности (например, зарплата в рублях и возраст в годах – масштаб разный, но надо сделать их сопоставимыми). Также сюда относится **выбор нужных столбцов**, **фильтрация данных** по критериям, объединение разрозненных источников данных, сортировка, группировка. Для текстовых данных применяются методы векторизации – TF-IDF, Word2Vec или BERT-эмбединги, преобразующие слова в числовые векторы. Временные

признаки часто разлагают на составляющие: час, день недели, месяц, что помогает моделям выявлять сезонные закономерности. Для работы с изображениями выполняется изменение размера, приведение к одному цветовому пространству и нормализация пикселей. Особое внимание уделяют созданию новых производных признаков, например вычислению возраста из даты рождения или площади из длины и ширины. Эти преобразования значительно повышают информативность данных и помогают алгоритмам машинного обучения выявлять более сложные зависимости.

- **Обогащение (дополнение) данных:** создание новых признаков (feature engineering) из исходных, если это может помочь модели. Пример – в задаче прогнозирования продаж на основе даты можно из даты создать признаки «месяц», «день недели», «праздничный/непраздничный день». Новые признаки иногда существенно улучшают качество. Также к обогащению относится **агрегирование** данных (например, расчет средних значений по группам) или **генерация синтетических данных** для балансировки выборки (например, методы увеличения выборки для классов с малым числом примеров, такие как SMOTE в задачах классификации). Для текстовых данных создаются признаки длины текста, тональности, наличия ключевых слов или эмбединги целых предложений. В геоданных можно из координат извлечь расстояние до центра города, плотность объектов в районе или принадлежность к административным зонам. При работе с временными рядами эффективно создание лаговых признаков (значений в предыдущие моменты времени) и скользящих статистик (среднее, максимум за последний период). Для борьбы с дисбалансом классов также применяются методы ансамблирования и взвешивания объектов при обучении. Качественное обогащение данных требует глубокого понимания предметной области и часто оказывается решающим фактором для достижения высокой точности модели.

- **Оптимизация и отбор признаков:** на заключительных шагах предварительной обработки может проводиться **снижение размерности** – например методами главных компонент (РСА) или отбором по важности признаков (feature selection). Это делается, чтобы убрать избыточные или неинформативные признаки, сократить размерность задачи и тем самым упростить обучение модели и уменьшить риск переобучения. Кроме того, убирают коррелирующие признаки или признаки с низкой вариативностью, поскольку они не несут новой информации. Для отбора признаков применяют как статистические методы (фильтры), так и модели (врапперы) и встроенные методы (эмбединг). Фильтры оценивают значимость признаков независимо от модели, например с помощью корреляции с целевой переменной или дисперсионного анализа. Врапперы последовательно добавляют или удаляют признаки, оценивая качество модели на кросс-валидации. Встроенные методы, такие как L1-регуляризация (Lasso), автоматически обнуляют веса неважных признаков в процессе обучения. Результатом становится компактный и информативный набор признаков, который ускоряет обучение без потери точности и повышает интерпретируемость модели.

После таких подготовительных шагов данные готовы для подачи в алгоритм обучения. Заметим, что предобработка – итеративный процесс: часто выявляются новые проблемы по мере того, как модель обучается (например, модель чувствительна к паре выбросов – тогда возвращаемся и удаляем/корректируем их, и так по циклу). Кроме того, часть этапов (масштабирование, кодирование) необходимо выполнять не только на обучающем наборе, но и затем на новых данных (тестовых, поступающих от пользователей) – поэтому практикуется сохранение параметров обработки (например, значений средних, используемых для нормализации), чтобы одинаково трансформировать и будущие данные.

Приведем небольшой пример предобработки на Python с использованием популярных библиотек:

```

import pandas as pd
import numpy as np

from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.feature_selection import SelectKBest, f_classif

# Загрузка данных из CSV файла
df = pd.read_csv('dataset.csv')

# 1. Первичный анализ данных
print(«Размер датасета:», df.shape)
print(«\nТипы данных:»)
print(df.dtypes)
print(«\nПервые 5 строк:»)
print(df.head())

# 2. Очистка: удаляем дубликаты и строки с пустыми критически
важными значениями
print(f«\nДо очистки: {df.shape[0]} строк»)
df.drop_duplicates(inplace=True)
df.dropna(subset=['ImportantFeature1', 'ImportantFeature2'], inplace=True)
print(f«После очистки: {df.shape[0]} строк»)

# 3. Обработка выбросов для числовых признаков
def remove_outliers(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    return df[(df[column] >= lower_bound) & (df[column] <= upper_bound)]

numeric_cols = ['Age', 'Salary', 'Score']

```

```

for col in numeric_cols:
    df = remove_outliers(df, col)

# 4. Обработка пропусков
for col in numeric_cols:
    df[col].fillna(df[col].median(), inplace=True) #
Медиана более устойчива к выбросам

# Для категориальных признаков
categorical_cols = ['Department', 'City']
for col in categorical_cols:
    df[col].fillna('Unknown', inplace=True)

# 5. Создание новых признаков
df['AgeGroup'] = pd.cut(df['Age'],
                        bins=[0, 25, 35, 45, 55, 100],
                        labels=['18-25', '26-35', '36-45', '46-55', '55+'])
df['SalaryToAgeRatio'] = df['Salary'] / df['Age']

# 6. Кодирование категориальных признаков
# One-Hot Encoding для признаков с небольшим количеством категорий
df = pd.get_dummies(df, columns=['Department'], prefix='Dept')

# Label Encoding для бинарных признаков
df['Gender'] = df['Gender'].map({'Male': 0, 'Female': 1})

# Для многоклассовых категориальных признаков
label_encoder = LabelEncoder()
df['City_encoded'] = label_encoder.fit_transform(df['City'])

# 7. Нормализация и стандартизация
# Min-Max Scaling
df['Salary_norm'] = (df['Salary'] - df['Salary'].min()) / (df['Salary'].max() -
df['Salary'].min())

```

```

# Standard Scaling (Z-score normalization)
scaler = StandardScaler()
df[['Age_scaled', 'Score_scaled']] = scaler.fit_transform(df[['Age', 'Score']])

# 8. Работа с датами (если есть)
if 'RegistrationDate' in df.columns:
    df['RegistrationDate'] = pd.to_datetime(df['RegistrationDate'])
    df['RegistrationYear'] = df['RegistrationDate'].dt.year
    df['RegistrationMonth'] = df['RegistrationDate'].dt.month
    df['DaysSinceRegistration'] = (pd.Timestamp.now() - df['RegistrationDate']).dt.days

# 9. Отбор признаков
# Удаляем исходные колонки, которые были преобразованы
columns_to_drop = ['City', 'RegistrationDate'] # добавляем ненужные
колонки
df.drop(columns=columns_to_drop, inplace=True, errors='ignore')

# 10. Автоматический отбор признаков для целевой переменной (если она
есть)
if 'Target' in df.columns:
    X = df.drop('Target', axis=1)
    y = df['Target']
    # Выбираем 10 лучших признаков
    selector = SelectKBest(score_func=f_classif, k=10)
    X_new = selector.fit_transform(X, y)
    # Получаем маску выбранных признаков
    selected_features = X.columns[selector.get_support()]
    print(f"\nОтобранные признаки: {list(selected_features)}")

# 11. Финальная проверка данных

```

```
print(f»\nФинальный размер датасета: {df.shape}»)
print(«\nИнформация о данных:»)
print(df.info())
print(«\nСтатистика числовых признаков:»)
print(df.describe())
# Сохранение обработанных данных
df.to_csv('cleaned_dataset.csv', index=False)
print(«\nОбработанные данные сохранены в 'cleaned_dataset.csv'»)
```

В этом примере мы:

1. **Проводим первичный анализ данных.** Скрипт начинает работу с вывода базовой информации о датасете – размера, типов данных и первых строк таблицы, что позволяет быстро оценить объем и структуру данных.

2. **Выполняем комплексную очистку данных.** После удаления дубликатов и записей с критическими пропусками добавляется фильтрация выбросов с помощью межквартильного размаха (IQR), что исключает аномальные значения из числовых признаков.

3. **Осуществляем расширенную обработку пропущенных значений.** Для числовых признаков пропуски заполняются медианными значениями как более устойчивыми к выбросам, а для категориальных данных используется заполнение значением 'Unknown'.

4. **Производим создание производных признаков.** Скрипт генерирует новые признаки, такие как возрастные группы и отношение зарплаты к возрасту, что может повысить информативность данных для моделей машинного обучения.

5. **Применяем многовариантное кодирование категориальных данных.** Используется one-hot-кодирование для признаков с малым числом категорий и label encoding – для бинарных и многоклассовых пере-

менных, обеспечивая корректное преобразование всех типов категориальных данных.

6. **Проводим двойное масштабирование числовых признаков.** Используется как Min-Max-нормализация для приведения к диапазону $[0, 1]$, так и стандартизация по Z-score, что позволяет выбрать оптимальный метод для конкретной модели.

7. **Выполняем расширенную работу с временными рядами.** Для дат регистрации извлекаются год, месяц и вычисляется количество дней с момента регистрации, раскрывая временные закономерности в данных.

8. **Осуществляем интеллектуальный отбор признаков.** С помощью метода SelectKBest автоматически выбираются наиболее значимые признаки на основе статистических тестов, что сокращает размерность данных.

9. **Проводим комплексную валидацию результатов.** Финальный этап включает подробную статистику обработанных данных и сохранение результата в новый CSV-файл, обеспечивая воспроизводимость preprocessing pipeline.

2.2. ЗАДАЧИ КЛАССИФИКАЦИИ И РЕГРЕССИИ: ПОСТАНОВКА И РЕШЕНИЕ

После того как данные подготовлены, их можно использовать для обучения модели. В машинном обучении (с учителем) почти все задачи делятся на **классификацию** или **регрессию**. Формально, разница в типе целевого признака (выхода модели):

- **Задача регрессии** – это предсказание непрерывного численного значения по заданным признакам. Примеры: прогноз цены квартиры по ее характеристикам, предсказание температуры на завтра, оценка скорости реакции водителя по данным сенсоров. На выходе модели регрессии получается число (вещественное), например стоимость 3,5 млн руб., темпера-

тура 18,2 °С, время 0,3 с. По определению, «*Задача регрессии – прогноз на основе выборки объектов с различными признаками, на выходе должно получиться вещественное число (например, цена квартиры, ожидаемый доход магазина)*» [4].

- **Задача классификации** – это предсказание категориальной величины, т.е. отнесение объекта к одному из заданного набора классов. Часто результатом классификации является дискретная метка или логическое решение *да/нет*. Примеры: определение, болен ли пациент определенной болезнью (классы: болен/не болен), распознавание рукописной цифры (классы 0 – 9), классификация изображения (кошка/собака/другое). Иными словами, «*задача классификации – получение категориального ответа на основе набора признаков, с конечным количеством вариантов ответа (например, есть ли на снимке опухоль: да или нет)*» [4].

К этим двум базовым типам сводится большинство практических задач ИИ. Иногда выделяют и другие (кластеризация, понижение размерности, генерация и т.д.), но в рамках нашего пособия сосредоточимся на задачах с учителем – классификации и регрессии, поскольку они часто возникают при создании умных приложений.

Решение задачи классификации предполагает обучение модели на размеченных данных, чтобы она могла правильно присваивать метки новым примерам. Рассмотрим кратко процесс на примере: допустим, мы хотим обучить ИИ распознавать, изображен ли на фотографии **огнетушитель** (это может пригодиться в системе поддержки обучения пожарных – камера следит, правильно ли стажер выбрал предмет). У нас собрана база из N фотографий, каждая из которых отмечена меткой: *огнетушитель* или *не огнетушитель*. После выполнения предобработки (например, приведения изображений к единому размеру, нормализации пикселей), мы выбираем алгоритм – пусть это будет нейронная сеть (конволюционная). Архитектура сети может быть, к примеру, такой: несколько сверточных

слоев для выделения признаков изображения, потом полносвязный слой и выход из одного нейрона с активацией *sigmoid*, который будет выдавать значение от 0 до 1 (интерпретируемое как вероятность класса «огнетушитель»). Мы разбиваем данные на обучающую и тестовую выборки, затем тренируем модель на обучающей, минимизируя функцию потерь (например, *бинарную кросс-энтропию* – она подходит для задач «да/нет»). После обучения проверяем качество на тестовой выборке: смотрим метрику *Accuracy* (долю правильных ответов) или *Precision/Recall*, строим матрицу ошибок (сколько ложных пропусков, ложных срабатываний). Если качество приемлемое (например, *ассигасу* = 0,95), внедряем модель в наше приложение: теперь, получив новое изображение, сеть выдаст вероятность – скажем, 0,87 – и если она $>0,5$, классификатор решает «объект на изображении – огнетушитель».

Современные инструменты существенно упрощают реализацию такого пайплайна. Например, используя высокоуровневое API Keras (на базе TensorFlow), можно задать и обучить модель в несколько строк. Ниже приведен условный пример создания простой модели классификации на Keras:

```
import numpy as np
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# 1. ПОДГОТАВЛИВАЕМ ДАННЫЕ
# Допустим, у нас есть исходные данные: X (признаки) и y (метки 0/1)
print(«Форма исходных данных:»)
print(f»X: {X.shape}, y: {y.shape}»)
# Разделяем данные на обучающую и тестовую выборки
```

```

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42, stratify=y)

# Масштабируем признаки для лучшей сходимости модели
scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

print(f'\nРазмеры после разделения:»)
print(f"Обучающая выборка: {X_train_scaled.shape}")
print(f"Тестовая выборка: {X_test_scaled.shape}")

# 2. СОЗДАЕМ АРХИТЕКТУРУ НЕЙРОННОЙ СЕТИ
features_dim = X_train_scaled.shape[1]

model = keras.Sequential([

    # Входной полносвязный слой с 16 нейронами и ReLU-активацией
    layers.Dense(16, activation='relu', input_shape=(features_dim,)),
    # Добавляем Dropout для регуляризации (предотвращение переобучения)
    layers.Dropout(0.3),
    # Второй скрытый слой с 8 нейронами
    layers.Dense(8, activation='relu'),
    # Выходной слой с сигмоидной активацией для бинарной классификации
    layers.Dense(1, activation='sigmoid')
])

# 3. КОНФИГУРИРУЕМ ПРОЦЕСС ОБУЧЕНИЯ
model.compile(

    optimizer='adam',          # Адаптивный оптимизатор
    loss='binary_crossentropy', # Функция потерь для бинарной
классификации
    metrics=['accuracy', 'precision', 'recall'] # Метрики качества

```

```

# Выводим информацию о архитектуре модели
print(«\nАрхитектура модели:»)
model.summary()

# 4. ЗАПУСКАЕМ ОБУЧЕНИЕ МОДЕЛИ
print(«\nНачинаем обучение модели...»)
history = model.fit(
    X_train_scaled, y_train,
    epochs=50,          # Количество эпох обучения
    batch_size=32,      # Размер мини-батча
    validation_split=0.2, # Доля данных для валидации
    verbose=1,          # Вывод прогресса обучения
    callbacks=[
        # Ранняя остановка при отсутствии улучшений
        keras.callbacks.EarlyStopping(patience=5, restore_best_weights=True),
        # Сохранение лучшей модели
        keras.callbacks.ModelCheckpoint('best_model.h5', save_best_only=True)
    ]
)

# 5. ОЦЕНИВАЕМ КАЧЕСТВО МОДЕЛИ
print(«\nОценка качества на тестовых данных...»)
test_loss, test_accuracy, test_precision, test_recall = model.evaluate(X_test_scaled, y_test, verbose=0)
print(f«\nРезультаты на тестовой выборке:»)
print(f"Точность (Accuracy): {test_accuracy:.3f}")
print(f"Precision: {test_precision:.3f}")
print(f"Recall: {test_recall:.3f}")
print(f"F1-score: {2 * (test_precision * test_recall) / (test_precision + test_recall):.3f}")

```

```

# 6. ДЕЛАЕМ ПРОГНОЗЫ НА НОВЫХ ДАННЫХ
print(«\nПрогнозирование на новых данных...»)
predictions = model.predict(X_test_scaled)
predicted_classes = (predictions > 0.5).astype(“int32”)
print(f»Примеры прогнозов:»)
print(f»Первые 10 предсказанных вероятностей:
{predictions[:10].flatten()})»)
print(f»Первые 10 предсказанных классов: {predict-
ed_classes[:10].flatten()})»)
print(f»Первые 10 истинных меток: {y_test[:10]})»)
# 7. СОХРАНЯЕМ ОБУЧЕННУЮ МОДЕЛЬ
model.save('final_model.h5')
print(«\nМодель сохранена в файл 'final_model.h5'»)

```

В этом коде мы:

1. **Проводим подготовку данных.** Выполняем разделение исходного датасета на обучающую и тестовую выборки, сохраняя баланс классов. Затем осуществляем масштабирование числовых признаков для обеспечения стабильной работы алгоритма градиентного спуска во время обучения модели.

2. **Выполняем построения архитектуры сети.** Создаем последовательную модель нейронной сети с несколькими полносвязными слоями и регуляризацией Dropout. Используем активационные функции ReLU в скрытых слоях и сигмоиду в выходном слое для получения вероятностных предсказаний.

3. **Осуществляем настройку параметров обучения.** Конфигурируем процесс обучения с выбором оптимизатора Adam и функции потерь `binary_crossentropy`. Дополнительно настраиваем метрики качества `accuracy`, `precision` и `recall` для комплексной оценки работы модели.

4. **Запускаем процесс обучения модели.** Проводим итеративное обучение модели на подготовленных данных с использованием callbacks для ранней остановки. Одновременно осуществляем мониторинг процесса обучения через валидацию на отложенной выборке.

5. **Проводим валидацию результатов.** Выполняем тестирование обученной модели на независимой тестовой выборке для объективной оценки качества. Рассчитываем комплекс метрик включая F1-score для анализа различных аспектов работы классификатора.

6. **Выполняем прогнозирование на новых данных.** Применяем обученную модель для предсказания на новых данных и получаем вероятностные оценки. Проводим сравнение предсказанных значений с фактическими метками для демонстрации эффективности модели.

7. **Осуществляем сохранение обученной модели.** Сохраняем итоговую модель в файл для последующего использования в production-среде. Обеспечиваем возможность повторного использования модели без необходимости повторного обучения.

Решение задачи регрессии во многом аналогично, только выходная переменная – число, и модель оптимизируются под другую метрику (например, MSE – среднеквадратичную ошибку). Пример: хотим предсказывать **время выполнения упражнения** в VR-тренажере на основе ряда параметров (уровень подготовки пользователя, сложность сценария, количество совершенных ошибок и т.д.). Собираем данные: для множества сессий имеем параметры и фактическое время. Обучаем модель (пусть это будет, для разнообразия, не нейросеть, а алгоритм градиентного бустинга решающих деревьев, например XGBoost или LightGBM – они хорошо подходят для табличных данных). На этапе обучения модель минимизирует MSE между предсказанным и реальным временем. После обучения можем использовать ее в приложении: подаем параметры текущей сессии – получаем прогноз времени. Если, скажем, прогноз слишком большой (что озна-

чает, пользователь, вероятно, запаздывает), система может адаптивно упростить сценарий или выдать подсказку.

Код для регрессии с использованием библиотеки `scikit-learn` может выглядеть так (например, линейная регрессия):

```
# ИМПОРТИРУЕМ НЕОБХОДИМЫЕ БИБЛИОТЕКИ

import numpy as np
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score

import matplotlib.pyplot as plt

# 1. ЗАГРУЖАЕМ И ПОДГОТАВЛИВАЕМ ДАННЫЕ
print(«Загрузка и подготовка данных...»)

# Предположим, что данные загружены в DataFrame df
# X – матрица признаков, y – целевая переменная
X = df.drop('target_column', axis=1)
y = df['target_column']

print(f"Размерность данных: X {X.shape}, y {y.shape}")

# 2. РАЗДЕЛЯЕМ ДАННЫЕ НА ОБУЧАЮЩУЮ И ТЕСТОВУЮ ВЫБОР-
КИ

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42

print(f"Обучающая выборка: {X_train.shape[0]} samples")
print(f"Тестовая выборка: {X_test.shape[0]} samples")
```

3. МАСШТАБИРУЕМ ПРИЗНАКИ ДЛЯ ЛУЧШЕЙ СХОДИМОСТИ

```
scaler = StandardScaler()
```

```
X_train_scaled = scaler.fit_transform(X_train)
```

```
X_test_scaled = scaler.transform(X_test)
```

```
print(«Масштабирование признаков завершено»)
```

4. СОЗДАЕМ И ОБУЧАЕМ МОДЕЛЬ ЛИНЕЙНОЙ РЕГРЕССИИ

```
print(«Создание и обучение модели...»)
```

```
reg = LinearRegression()
```

```
reg.fit(X_train_scaled, y_train)
```

5. АНАЛИЗИРУЕМ ПАРАМЕТРЫ МОДЕЛИ

```
print(«\nПараметры модели:»)
```

```
print(f»Коэффициенты: {reg.coef_}»)
```

```
print(f»Свободный член: {reg.intercept_:.2f}»)
```

6. ДЕЛАЕМ ПРОГНОЗЫ НА ТЕСТОВОЙ ВЫБОРКЕ

```
y_pred = reg.predict(X_test_scaled)
```

7. ОЦЕНИВАЕМ КАЧЕСТВО МОДЕЛИ

```
mse = mean_squared_error(y_test, y_pred)
```

```
mae = mean_absolute_error(y_test, y_pred)
```

```
r2 = r2_score(y_test, y_pred)
```

```
print(«\nМЕТРИКИ КАЧЕСТВА МОДЕЛИ:»)
```

```
print(f»Mean Squared Error (MSE): {mse:.2f}»)
```

```
print(f»Mean Absolute Error (MAE): {mae:.2f}»)
```

```
print(f»R2 Score: {r2:.3f}»)
```

8. ВИЗУАЛИЗИРУЕМ РЕЗУЛЬТАТЫ

```
plt.figure(figsize=(10, 6))
```

```
plt.scatter(y_test, y_pred, alpha=0.7)
```

```
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r-', lw=2)
```

```
plt.xlabel('Фактические значения')
```

```

plt.ylabel('Предсказанные значения')
plt.title('Фактические vs Предсказанные значения')
plt.grid(True)
plt.show()

# 9. АНАЛИЗИРУЕМ ОСТАТКИ
residuals = y_test - y_pred
plt.figure(figsize=(10, 6))
plt.scatter(y_pred, residuals, alpha=0.7)
plt.axhline(y=0, color='r', linestyle='--')
plt.xlabel('Предсказанные значения')
plt.ylabel('Остатки')
plt.title('Анализ остатков')
plt.grid(True)
plt.show()

# 10. СОХРАНЯЕМ МОДЕЛЬ ДЛЯ ДАЛЬНЕЙШЕГО ИСПОЛЬЗОВАНИЯ
import joblib

joblib.dump(reg, 'linear_regression_model.pkl')
joblib.dump(scaler, 'scaler.pkl')

print(«\nМодель и скейлер сохранены в файлы»)

```

1. Проводим загрузку и первичный анализ данных. Выполняем импорт необходимых библиотек и загрузку исходного датасета. Разделяем данные на матрицу признаков и целевую переменную, анализируем размерность и структуру данных для понимания объема информации.

2. Осуществляем разбиение данных на обучающую и тестовую выборки. Применяем функцию `train_test_split` для разделения данных в соотношении 80/20. Сохраняем случайное состояние для воспроизводимости результатов и выводим информацию о размерах полученных выборок.

3. Выполняем предобработку и масштабирование признаков. Используем `StandardScaler` для приведения всех признаков к единому масштабу. Выполняем `fit_transform` на обучающей выборке и `transform` на тестовой для предотвращения `data leakage`.

4. Создаем и обучаем модель линейной регрессии. Инициализируем модель `LinearRegression` и проводим обучение на масштабированных данных. Модель автоматически подбирает оптимальные коэффициенты для минимизации ошибки предсказания.

5. Проводим анализ параметров обученной модели. Изучаем полученные коэффициенты регрессии и свободный член. Анализируем влияние каждого признака на целевую переменную через величину и знак коэффициентов.

6. Выполняем прогнозирование на тестовых данных. Используем обученную модель для предсказания значений на тестовой выборке. Получаем массив предсказанных значений для последующей оценки качества модели.

8. Осуществляем комплексную оценку качества модели. Рассчитываем основные метрики регрессии: `MSE`, `MAE` и R^2 . Анализируем каждую метрику для понимания различных аспектов качества предсказаний.

9. Проводим визуализацию результатов прогнозирования. Строим диаграмму рассеяния для сравнения фактических и предсказанных значений. Добавляем линию идеальных предсказаний для наглядной оценки точности модели.

10. Выполняем анализ остатков модели. Исследуем распределение остатков для проверки предположений линейной регрессии. Визуализируем остатки для выявления паттернов и потенциальных проблем модели.

11. Сохраняем модель и препроцессор для продакшена. Сохраняем обученную модель и скейлер в файлы для последующего использования. Обеспечиваем возможность развертывания модели в `production`-среде.

Линейная регрессия старается подобрать коэффициенты при признаках, чтобы наилучшим образом приблизить целевую переменную. Для более сложных зависимостей можно использовать полиномиальные признаки или перейти к нелинейным моделям (например, все тому же бустингу деревьев). Принципиально, однако, подход един: обучить по примерам, минимизируя ошибку прогноза.

Выбор решения для задачи. Разработчик, столкнувшийся с практической задачей (например, нужно распознавать объект или предсказывать некоторое число), должен сперва ясно определить: классификация это или регрессия? От этого зависит и выбор алгоритмов, и метод оценки качества. Для классификации важны метрики точности, полноты и т.п., для регрессии – ошибки вроде MSE, MAE. Некоторые алгоритмы могут решать и то, и другое (например, нейронные сети универсальны), но их настройка (активации, функции потерь) будет разной под разные задачи.

В контексте нашего пособия, приложения ИИ могут включать множество таких частных задач. Например, **в системе виртуальной реальности** можно встроить классификатор жестов пользователя (класс «рука поднята» или «рука опущена» по данным контроллера) – это задача классификации. Или же модель, оценив качество прохождения тренажера, может попытаться прогнозировать уровень навыка (числом) – задача регрессии. Разработчику важно понимать, как решить каждую из них, поэтому навыки предварительной обработки данных и применения моделей классификации/регрессии являются базовыми.

КОНТРОЛЬНЫЕ ВОПРОСЫ К РАЗДЕЛУ 2

1. Зачем требуется предварительная обработка данных перед обучением модели машинного обучения? Какие проблемы в сырых данных она помогает решить?
2. Приведите примеры трех методов обработки пропущенных значений в наборе данных. В каких ситуациях каждый из них предпочтителен?
3. Что такое нормализация признаков и для чего она необходима? Чем отличается нормализация от стандартизации?
4. Объясните на примере, что такое кодирование категориальных признаков. Что означает термин «one-hot encoding»?
5. Какие цели преследует снижение размерности данных? Назовите хотя бы один метод снижения размерности.
6. Каковы отличительные особенности задачи классификации и задачи регрессии? Чем различаются выход модели и метрики качества в этих задачах?
7. Приведите пример прикладной задачи из области виртуальной реальности или игровой индустрии, которая является задачей классификации. А аналогичный пример для задачи регрессии.
8. Почему при обучении модели классификации важно разбивать данные на обучающую и тестовую выборки? Что может произойти, если этого не сделать?
9. Какую функцию потерь обычно используют для обучения модели классификации на два класса? А какую – для задачи регрессии?
10. В коде на Keras (см. Пример в тексте) для выходного слоя классификатора использована сигмоидная функция активации, а для обучения – бинарная кросс-энтропия. Почему выбраны именно они? Что было бы, если заменить сигмоиду на *softmax* или на отсутствие активации?

11. Какие основные этапы предобработки данных для изображений в компьютерном зрении и чем они отличаются от обработки табличных данных?

12. Что такое «утечка данных» (data leakage) в контексте подготовки данных и к каким последствиям она может привести при оценке модели?

13. В чем разница между методами обработки выбросов: удаление, преобразование и моделирование? Когда целесообразно применять каждый из них?

14. Какие методы feature engineering наиболее эффективны для временных рядов и почему?

15. Как выбор методов масштабирования (MinMaxScaler, StandardScaler, RobustScaler) влияет на разные алгоритмы машинного обучения?

16. Какие стратегии балансировки классов в задачах классификации вы знаете? Опишите их преимущества и недостатки.

17. Как кросс-валидация помогает оценить устойчивость модели к разным подвыборкам данных?

18. В чем заключаются различия между параметрами и гиперпараметрами модели? Приведите примеры методов настройки гиперпараметров.

19. Какие метрики оценки качества регрессии, кроме MSE и MAE, вы знаете и в каких ситуациях их следует применять?

20. Как можно интерпретировать веса признаков в линейной модели регрессии и какие ограничения есть у такой интерпретации?

3. ОБРАБОТКА ДАННЫХ

В ПРИЛОЖЕНИЯХ ВИРТУАЛЬНОЙ РЕАЛЬНОСТИ

Приложения виртуальной реальности и тренажеры генерируют и потребляют большой объем данных, связанный с действиями пользователя и состоянием виртуальной среды. В отличие от данных для классических ML-задач (таблицы, тексты и пр.), здесь речь идет об оперативных потоках данных — координаты и повороты контроллеров, нажатия кнопок, события столкновений объектов, показатели состояния пользователя и т.д.

Высокая частота и низкая задержка. Данные трекинга (позиция головы, контроллеров) должны обновляться с частотой 90 Гц и выше, а задержка обработки должна быть минимальной (менее 20 мс) для предотвращения киберболезни.

Многомодальность. Данные поступают из разнородных источников — системы трекинга, контроллеры, биометрические датчики, аудиопоток, что требует их синхронизации и комплексного анализа.

Пространственный характер. Большинство данных так или иначе связано с 3D-пространством (координаты, повороты, векторы), что требует применения специализированных математических аппаратов (линейная алгебра, кватернионы).

Контекстуальная зависимость. Одни и те же данные (например, резкое движение головой) могут иметь разный смысл в разных ситуациях.

Грамотная организация сбора и обработки этих данных имеет ключевое значение для успеха VR-приложения, особенно если оно включает элементы искусственного интеллекта или сложной симуляции.

3.1. КЛАССИФИКАЦИЯ И ИСТОЧНИКИ ДАННЫХ В VR

Для эффективного системного анализа и построения конвейеров обработки информации в VR-приложениях необходима четкая классификация поступающих данных. Многомодальные потоки данных в VR можно категоризировать по их источнику, назначению и частоте обновления. Рассмотрим основные классы данных, формируемые в процессе работы VR-системы в реальном времени (рис. 3.1).

- **Данные позиций и ориентаций (трекинг).** Это фундаментальный и наиболее критичный к задержкам тип данных. Гарнитура VR отслеживает положение и ориентацию головы пользователя в пространстве, обеспечивая 6 степеней свободы (6DoF): линейные координаты (x, y, z) и углы поворота (рыскание – yaw, тангаж – pitch, крен – roll). Данные поступают с высокой частотой, обычно от 72 до 120 раз в секунду. Аналогично отслеживаются позиция и ориентация контроллеров.

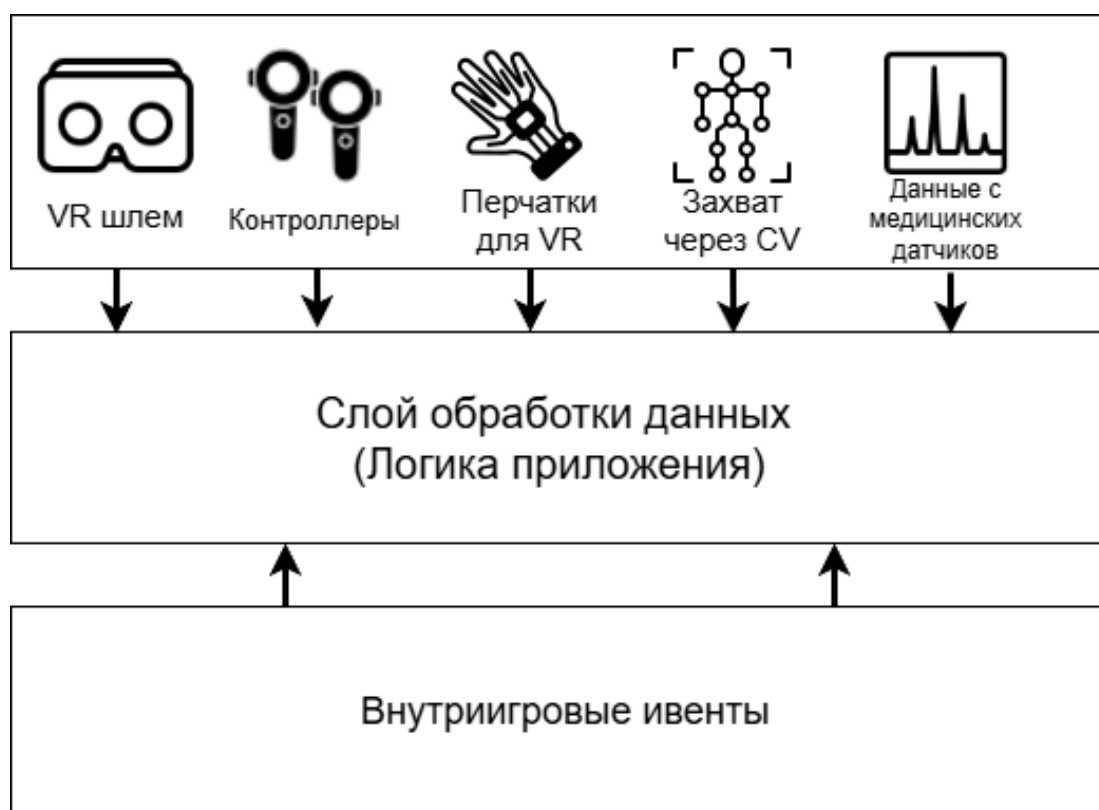


Рис. 3.1. Схема обработки данных

Назначение: обновление видовой камеры (точки обзора пользователя) и виртуальных представлений контроллеров или рук в сцене.

Требования: от точности, частоты и, главное, минимальной задержки (латентности) трекинга напрямую зависят реализм погружения и комфорт пользователя. Задержки более 20 мс хорошо заметны и могут портить эффект погружения. Таким образом, сбор и обработка данных трекинга – наиболее непрерывный и критичный процесс в VR-системе.

- **Данные ввода от контроллеров.** Помимо пространственной информации, контроллеры (например, Oculus Touch, HTC Vive Controller) генерируют данные о взаимодействии пользователя с их элементами управления.

Типы данных:

1. *Дискретные события:* нажатие/отпускание кнопок (бинарные данные).

2. *Непрерывные данные:* значение аналоговых триггеров, положение джойстиков (ось X/Y).

3. *Тактильная отдача (Haptics):* управляющий сигнал для вибромотора контроллера.

Назначение: обработка действий пользователя (выстрел, хватка объекта, вызов меню, перемещение). Эти данные должны обрабатываться мгновенно для обеспечения непосредственной обратной связи.

- **События взаимодействия в виртуальной среде.** Данные, генерируемые логикой самой виртуальной среды в ответ на действия пользователя или внутренние процессы.

Примеры событий:

1. *Столкновения (Collision):* факт физического контакта двух виртуальных объектов.

2. *Активация триггерных зон (Trigger Volumes):* вход/выход объекта (часто самого пользователя) в заданную область.

3. *События сценария*: завершение анимации, достижение контрольной точки, изменение состояния игровой логики.

Назначение: регистрация действий пользователя для анализа (логирование), активация скриптов и изменение состояния приложения. Например, в VR-тренажере по технике безопасности пересечение виртуальной «опасной зоны» генерирует событие, которое уменьшает «очки успеха» обучаемого и записывается в отчет.

- **Биометрические и поведенческие данные пользователя.** Этот класс данных используется в продвинутых и специализированных VR-системах для глубокого анализа состояния и реакций пользователя.

Источники и типы данных:

1. *Трекинг всего тела (Body Tracking)*: данные с дополнительных трекеров на ногах, поясице для реконструкции полной позы.

2. *Трекинг взгляда (Eye-Tracking)*: направление взгляда, диаметр зрачка (пупиллометрия), частота моргания.

3. *Перчатки ввода данных (Data Gloves)*: сгибы пальцев, жесты.

4. *Трекинг рук с помощью (Computer Vision)*: доступен на некоторых типах шлемов.

5. *Биометрические датчики*: частота сердечных сокращений (ЧСС, ЭКГ), кожно-гальваническая реакция (КГР, уровень потоотделения), электроэнцефалография (ЭЭГ, активность мозга).

Назначение: оценка когнитивной нагрузки, эмоционального состояния (стресс, вовлеченность), фокуса внимания. Широко применяется в тренажерах для пилотов, операторов сложных систем, в клинической психологии (лечение фобий) и системах физической реабилитации для объективной оценки двигательной активности пациента.

- **Системные и диагностические данные.** Данные, характеризующие не действия пользователя, а техническое состояние и производительность самой VR-системы.

Параметры:

- 1) *частота кадров (FPS)*: ключевой показатель плавности работы;
- 2) *задержка (Motion-to-Photon Latency)*: общее время между движением головы и обновлением изображения;
- 3) *загрузка вычислительных ресурсов*: использование CPU, GPU, памяти;
- 4) *температура устройства*: актуально для автономных (standalone) гарнитур.

Назначение: мониторинг стабильности работы, отладка (дебаггинг) и динамическая оптимизация. Например, при падении FPS система может автоматически снизить детализацию графики для поддержания комфортного режима работы.

Сводная классификация данных в VR-приложениях

Категория	Типы данных	Частота обновления	Критичность задержки	Основное назначение
Трекинг	Позиция (x, y, z), ориентация (yaw, pitch, roll)	72...120 Гц	Очень высокая	Рендеринг, навигация
Ввод с контроллеров	События кнопок, аналоговые значения	90 Гц	Высокая	Взаимодействие
События среды	Столкновения, активация триггеров	Переменная (по событию)	Средняя	Игровая логика, анализ
Биометрические данные	Поза, взгляд, пульс, ЭЭГ	50...250 Гц	Низкая/средняя*	Анализ состояния пользователя
Системные данные	FPS, загрузка CPU/GPU	1...10 Гц	Низкая	Мониторинг и оптимизация

*Зависит от приложения: для адаптивного ИИ в реальном времени задержка может быть важна.

Важно отметить, что в VR-приложении **все эти данные обычно обрабатываются в режиме реального времени**. Это значит, что архитектура ПО должна обеспечивать прием данных (например, от шлема и контроллеров) с минимальными задержками и сразу использовать их для обновления виртуальной сцены и логики.

3.2. СБОР ДАННЫХ: API И ИНСТРУМЕНТЫ (ПРИМЕР OPENXR)

Эффективный сбор данных от разнородного VR-оборудования невозможен без использования стандартизированных программных интерфейсов (API). Исторически разработчикам приходилось создавать отдельные реализации для устройств разных производителей (Oculus, HTC, Windows Mixed Reality), что значительно усложняло поддержку и распространение приложений. Решением этой проблемы стал открытый, кроссплатформенный стандарт OpenXR, который предоставляет единый API для взаимодействия с любым совместимым VR-оборудованием и программным обеспечением (рантаймом). OpenXR выступает в роли промежуточного слоя между приложением и аппаратным обеспечением. Его ключевая задача – абстрагировать разработчика от специфики конкретного устройства (рис. 3.2).

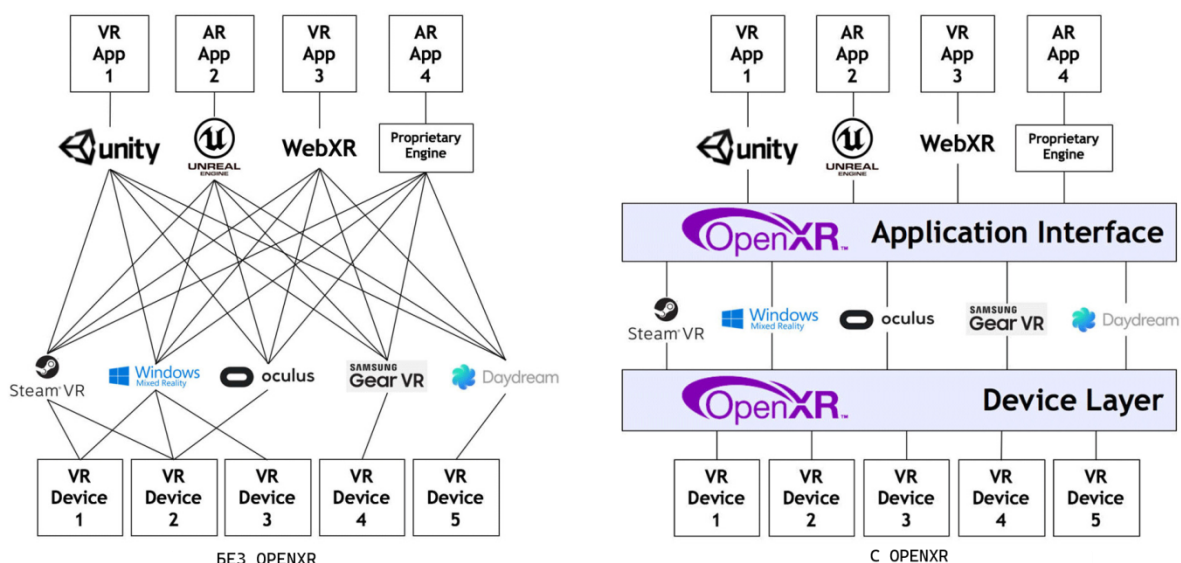


Рис. 3.2. Архитектура при работе без/с OpenXR

Как же практически выглядит получение, скажем, координат контроллера в приложении? Рассмотрим пример с Unity (язык C#) с использованием XR API. Unity предоставляет пространственный трекинг через пространство имен `UnityEngine.XR`. Мы можем запросить список активных XR-устройств (HMD и контроллеры) и считать их свойства. Ниже приведен фрагмент кода `MonoBehaviour`-компонента, который в методе `Update` будет извлекать координаты головы пользователя и выводить их в консоль:

```
using UnityEngine;
using UnityEngine.XR;
using System.Collections.Generic;

public class VRDataLogger : MonoBehaviour
{
    void Update()
    {
        // Получаем актуальный список всех XR-узлов (устройств)
        List<XRNodeState> nodeStates = new List<XRNodeState>();
        InputTracking.GetNodeStates(nodeStates);

        // Итерируем по узлам, находим узел типа «Голова»
        foreach (XRNodeState node in nodeStates)
        {
            if (node.nodeType == XRNode.Head)
            {
                Vector3 headPosition;
                Quaternion headRotation;

                // Пытаемся получить позицию и поворот текущего кадра
```


XrSpace для головы/контроллера. Но эти детали выходят за рамки данного пособия – важно понимать концепцию: **данные позиций поступают потоком и должны постоянно обрабатываться.**

Кроме позиций, через API доступны и остальные типы данных: нажатия кнопок (в Unity – класс `InputDevice` с методом `TryGetFeatureValue` для `CommonUsages.triggerButton` и др.), отслеживание рук (напрямую предоставляется, если гарнитура поддерживает – например, Oculus Quest дает набор точек скелета руки), камера passthrough (для AR) и т.п.

Стандарт OpenXR определяет унифицированные типы таких данных. Например, для рук существует расширение `XR_EXT_hand_tracking` – оно описывает 26 суставов руки и их позиции [17]. Разработчик, используя это расширение, может получить координаты каждого пальца. Поскольку OpenXR поддержан кросс-платформенно, одна и та же программа сможет, например, получать данные отслеживания рук на разных устройствах (HoloLens 2, Quest 2 и т.д.) без изменения кода – достаточно, чтобы runtime устройства поддерживал упомянутое расширение.

Еще одна интересная категория данных – **взаимодействие со сценой**. Под этим понимают события или параметры, характеризующие, как пользователь и объекты влияют друг на друга. К примеру:

- **Уровень двигательной активности** – сколько шагов прошел пользователь в виртуальной среде (если используется передвижение), как часто он наклоняется или приседает. Эти данные можно вычислять на основе трекинга позы: по траектории движения HMD вычислить пройденное расстояние, по изменению высоты – фиксировать приседания и т.п. В обучающих VR-приложениях по фитнесу или реабилитации такие показатели напрямую связаны с целями занятий.

- **Взаимодействие с объектами** – сколько предметов пользователь взял, какие действия выполнил (например, открыл дверь, нажал кнопку

панели управления, соединил два кабеля и т.п.). Здесь данные получают либо из скриптов (каждый раз, когда выполнено целевое действие, логируем событие), либо анализируя состояния объектов (например, через механизм событий: «OnPickUp», «OnDrop», «OnUseItem»).

- **Параметры виртуальной среды** – температура или освещение (если симулируются), уровень сложности (если сценарий адаптивный), текущее время симуляции, счетчик очков или прогресса и пр. Их тоже можно рассматривать как данные, подлежащие сбору – особенно если требуется анализировать сессии постфактум (например, инструктор потом просматривает лог: время прохождения уровня, сколько подсказок было запрошено и т.д.).

Во многих VR-системах, особенно обучающих, ведется **логирование всех ключевых параметров** в ходе сессии. Потом эти логи можно анализировать для оценки эффективности обучения или для улучшения дизайна системы. Например, лог может содержать: таймштампы всех действий пользователя, координаты его перемещений каждую секунду, объекты, с которыми он взаимодействовал, полученные им оценки. Далее эти данные могут быть обработаны с помощью методов анализа данных или ИИ: например, кластеризация путей движения всех пользователей, чтобы выявить типичные ошибки (места, куда все заходят не туда), либо обучение модели, предсказывающей успешность прохождения по ранним действиям (чтобы вовремя корректировать сценарий под отстающего пользователя).

Для эффективной разработки VR-приложений вместо создания всех систем с нуля рекомендуется использовать готовые фреймворки и инструменты. Они предоставляют проверенные, кросс-платформенные решения для стандартных задач, таких как взаимодействие с объектами, передвижение и рендеринг.

- **XR Interaction Toolkit: Стандарт от Unity.** XR Interaction Toolkit – это высокоуровневая, компонентная система взаимодействия для создания VR и AR-опытов, разработанная самой Unity. Его философия заключается в предоставлении строительных блоков (компонентов Interactor и Interactable), которые связываются через Interaction Manager. Это дает разработчику большую гибкость и контроль над логикой взаимодействий.

Типичный рабочий процесс:

1. *Установка:* добавляется через Package Manager в Unity.
2. *Быстрый старт:* в проект импортируются Starter Assets, которые содержат готовые префабы XR Origin (VR-камеры) и контроллеров, а также предварительно настроенные Input Actions для системы ввода.
3. *Создание взаимодействия:* чтобы сделать объект перемещаемым, на него добавляется компонент XR Grab Interactable. Контроллер, который может его схватить, помечается компонентом XR Direct Interactor или XR Ray Interactor.

Основные возможности:

1. *Взаимодействие с объектами:* захват, броски, наведение.
2. *Передвижение (Locomotion):* телепортация и мгновенные повороты.
3. *Взаимодействие с UI:* возможность взаимодействовать с элементами Canvas через контроллеры.
4. *Тактильная отдача:* простое подключение вибрации контроллеров.
5. *Кросс-платформенность:* работа с устройствами Meta Quest, HTC Vive и другими через OpenXR.

- **UltimateXR: Open-source решение.** UltimateXR – это бесплатный open-source фреймворк, ориентированный на предоставление готовых,

продвинутых и высококачественных механизмов взаимодействия. Его цель – позволить разработчикам быстро создавать богатый и immersive VR-контент, минимизируя рутинную работу.

Ключевые уникальные особенности:

- **Интегрированный редактор поз рук:** позволяет создавать и настраивать анимацию хвата и жестов прямо в редакторе Unity, не прибегая к внешним 3D-редакторам. Система Blend Poses позволяет одним набором анимаций охватывать объекты разного размера, динамически регулируя «схват» кисти.
- **Универсальные механики:** включает системы манипулирования объектами, модульную систему оружия, механику магнетической перчатки, смену аватаров (включая частичное или полное тело) и многое другое.
- **Типичный рабочий процесс:** объекты, с которыми можно взаимодействовать, наследуются от базового класса `UxrGrabbableObject`. Их поведение (например, тип хвата, двух-handed manipulation) настраивается в специальном редакторе UltimateXR, что ускоряет итерации.

3.3. ОБРАБОТКА ДАННЫХ В VR-ТРЕНАЖЕРАХ И СИСТЕМАХ

После сбора данных важно их правильно **обработать и использовать** внутри приложения. В реальном времени обработка часто сводится к прямой реакции: данные трекинга идут в обновление позиции камеры и объектов, нажатие кнопки вызывает выполнение привязанной функции (выстрел, teleport и т.п.), столкновение с опасной зоной вызывает отображение предупреждения на экране. То есть значительная часть обработки – это **детерминированные алгоритмы, прописанные разработчиком**.

Однако все чаще VR-системы дополняются интеллектуальной обработкой. Примеры таких случаев:

- **Анализ траектории и позы пользователя.** Допустим, VR-тренажер по пожарной эвакуации записывает траекторию движения человека. После упражнения система анализирует: насколько оптимальным был маршрут? Не проходил ли человек слишком медленно или опасно близко от «очага пожара»? Такая аналитика может быть реализована с помощью алгоритмов: сравнить путь пользователя с эталонным (вычислить отклонение в метрах), рассчитать время простоя, построить тепловую карту посещенных областей. На основе этого система дает рекомендации. Это уже обработка данных – по сути, классическая задача анализа временных рядов (координаты от времени) и геометрических данных.

- **Распознавание действий или жестов.** Некоторые VR-приложения пытаются автоматически распознать, что делает пользователь, помимо простых нажатий. Например, в медицинском VR-симуляторе важно понять, правильно ли обучаемый выполняет **последовательность шагов** процедуры. Система может сравнивать последовательность его действий (взял инструмент А, потом В, потом сделал Х) с заложенным скриптом, а может использовать модель машинного обучения, обученную на данных эксперта, чтобы классифицировать «правильность» действий. Другой пример – распознавание жестов рук: по данным отслеживания суставов руки (например, используя 26 точек skeleton hand tracking) алгоритм классифицирует жест (напр. «указание пальцем», «поднята рука», «жест стоп»). Это тоже задача классификации, только данные поступают в реальном времени и решение должно быть мгновенным.

- **Фильтрация и сглаживание сенсорных данных.** Часто сырые данные трекинга содержат шум (особенно если используются внешние камеры или инерциальные датчики). Для плавности визуализации применяются фильтры (например, комплементарный фильтр или фильтр Калмана) – они сглаживают небольшие скачки в положении, предсказыва-

ют следующий кадр позиции, если сигнал пропал на мгновение. Такая обработка улучшает пользовательский опыт (меньше дрожания объектов) и часто встроена в SDK оборудования, но разработчик тоже может ее настраивать.

- **Синхронизация и временные метки.** Данные могут приходить с разных частот (например, данные трекинга 90 Гц, а логика игры работает с фиксированной частотой 60 Гц). Нужно правильно синхронизовать их: либо использовать последний доступный трекинг на момент расчета кадра, либо интерполировать/экстраполировать положения. Это техническая задача обработки данных времени.

- **Использование внешних данных.** Иногда VR-приложение интегрировано с внешними системами. Например, тренажер оператора крана может получать **реальные данные ветра** или других условий с датчиков, чтобы симуляция соответствовала реальности. Это значит, приложение должно получать данные по сети, обрабатывать их (скажем, порывы ветра конвертировать в силу воздействия на поднимаемый груз) и влиять на физическую модель. Здесь обработка предполагает преобразование внешних сигналов в параметры виртуальной физики.

- **OpenXR: Слои и расширения.** С точки зрения обработки стоит упомянуть возможность OpenXR использовать **слои API** – т.е. плагины, которые перехватывают данные и могут их модифицировать. Например, слой от Ultraleap добавляет данные отслеживания рук для любого устройства. Такие слои фактически делают обработку на уровне API: они могут объединять данные от нескольких устройств, фильтровать их. Разработчик может подключать/отключать слои для расширения функциональности своего приложения (например, слой записи реплея – перехватывает все позы и сохраняет, а потом может воспроизвести).

В итоге обработка данных в VR-системах включает как **реактивную компоненту** (мгновенно отработать поступившее событие), так и **аналитическую компоненту** (собирать статистику и делать выводы). Современные системы виртуальной реальности все чаще сохраняют данные сессий для последующего анализа методами Data Mining. Например, исследователи могут применять алгоритмы последовательного анализа действий, чтобы найти типичные ошибки новичков и на основе этого улучшить программу обучения. Также анализируют поведение успешных и неуспешных обучаемых, сравнивая последовательности их действий (методики *Differential Sequence Mining*, *Epistemic Network Analysis* и др. были применены для анализа поведения учащихся в VR).

Таким образом, роль данных в VR-приложениях двояка: с одной стороны, без мгновенной обработки потоковых данных невозможна сама работа VR (рендер сцены в соответствии с движениями пользователя), с другой – накопленные данные открывают возможности для улучшения и интеллектуализации VR-систем. Разработчики должны уметь работать с обеими сторонами: и оптимизировать обработку в реальном времени, и организовывать сбор/хранение данных для офлайн-анализа (возможно, с помощью методов из раздела 2).

КОНТРОЛЬНЫЕ ВОПРОСЫ К РАЗДЕЛУ 3

1. Какие типы данных в наибольших объемах генерируются при работе VR-приложения? Почему они требуют высокочастотной обработки?
2. Что такое 6DoF-трекинг и какие данные он предоставляет приложению?
3. Перечислите основные устройства ввода, используемые в VR. Какие данные от них нужно обрабатывать помимо положения в пространстве?
4. В чем суть стандарта OpenXR и как он облегчает работу с данными от различных VR-устройств?
5. Как в игровом движке (Unity или Unreal) можно получить позицию VR-гарнитуры в пространстве? Опишите общий подход или приведите псевдокод.
6. Зачем VR-приложению может потребоваться логирование действий пользователя? Какие преимущества дает анализ таких логов?
7. Приведите пример того, как данные траектории движения пользователя в VR могут быть использованы для улучшения тренажера.
8. Какие методы машинного обучения могли бы применяться к данным, собираемым в VR (траектории, последовательности действий, события) для их анализа? Назовите хотя бы два.
9. В каких случаях в VR-системах задействуются фильтры (например, Калмана) для обработки данных, и что они позволяют улучшить?
10. Если в VR-тренажере используется трекинг рук (без контроллеров), какие данные получает система и какие задачи ИИ можно решать на основе этих данных?
11. Каковы основные challenges (проблемы) при синхронизации данных, поступающих от разных источников в VR (например, трекинг, аудио, биометрия)?

12. Как стандарт OpenXR упрощает процесс сбора данных от контроллеров и гарнитуры? Приведите пример использования OpenXR API.

13. Что такое «Motion-to-Photon Latency» и почему это критически важный параметр для обработки данных в VR?

14. Какие методы фильтрации данных трекинга (например, фильтр Калмана) применяются для снижения дрожания (джиттера) и почему они важны для пользовательского комфорта?

15. Как данные трекинга глаз (Eye-Tracking) могут быть использованы для адаптивного рендеринга (foveated rendering) и повышения производительности?

16. Опишите, как можно использовать логирование действий пользователя в VR-тренажере для последующего анализа и улучшения методики обучения.

17. Какие специфические требования к данным предъявляют системы трекинга всего тела (Full-Body Tracking) по сравнению с трекингом только головы и рук?

18. В чем разница между обработкой дискретных событий (нажатие кнопки) и непрерывных потоков данных (позиция) в реальном времени внутри игрового движка?

19. Как можно использовать машинное обучение для анализа биометрических данных (например, ЭЭГ, ЧСС) пользователя в реальном времени для адаптации VR-сценария?

20. Какие архитектурные решения позволяют эффективно обрабатывать и передавать большие объемы данных в многопользовательских VR-приложениях?

4. РЕШЕНИЕ ЗАДАЧ УПРАВЛЕНИЯ В ПРИЛОЖЕНИЯХ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА И ВИРТУАЛЬНОЙ РЕАЛЬНОСТИ

Разработка приложений с использованием ИИ и VR часто сопряжена с необходимостью **управлять** сложными процессами или агентами в виртуальной среде. Речь может идти как об управлении персонажем (или роботом) на основе нейронной сети, так и об **автоматическом выборе оптимальных действий** системой для достижения поставленной цели. Также под задачами управления подразумевается поддержание определенного состояния системы или обеспечение реагирования на изменения – по сути, это близко к классической теории управления, но с привлечением методов искусственного интеллекта. В данном разделе мы рассмотрим, как нейронные сети применяются для задач управления и мониторинга, особое внимание уделив **обучению с подкреплением** как методике создания интеллектуальных агентов, а также обсудим примеры задач управления в виртуальных тренажерах.

4.1. НЕЙРОННЫЕ СЕТИ ДЛЯ УПРАВЛЕНИЯ И МОНИТОРИНГА

Нейронные сети способны не только распознавать образы, но и вырабатывать **управляющие воздействия**. Если подходить формально, задача управления – это задача выбора таких действий, которые переводят систему (или агента) в желаемое состояние или позволяют достичь цели. Классический пример – управление роботом: нужно генерировать команды двигателям (напряжения, усилия) так, чтобы робот прошел по траектории и не упал. В виртуальной реальности аналог – управление виртуальным персонажем (NPC) или объектом на основе данных окружения (рис. 4.1).

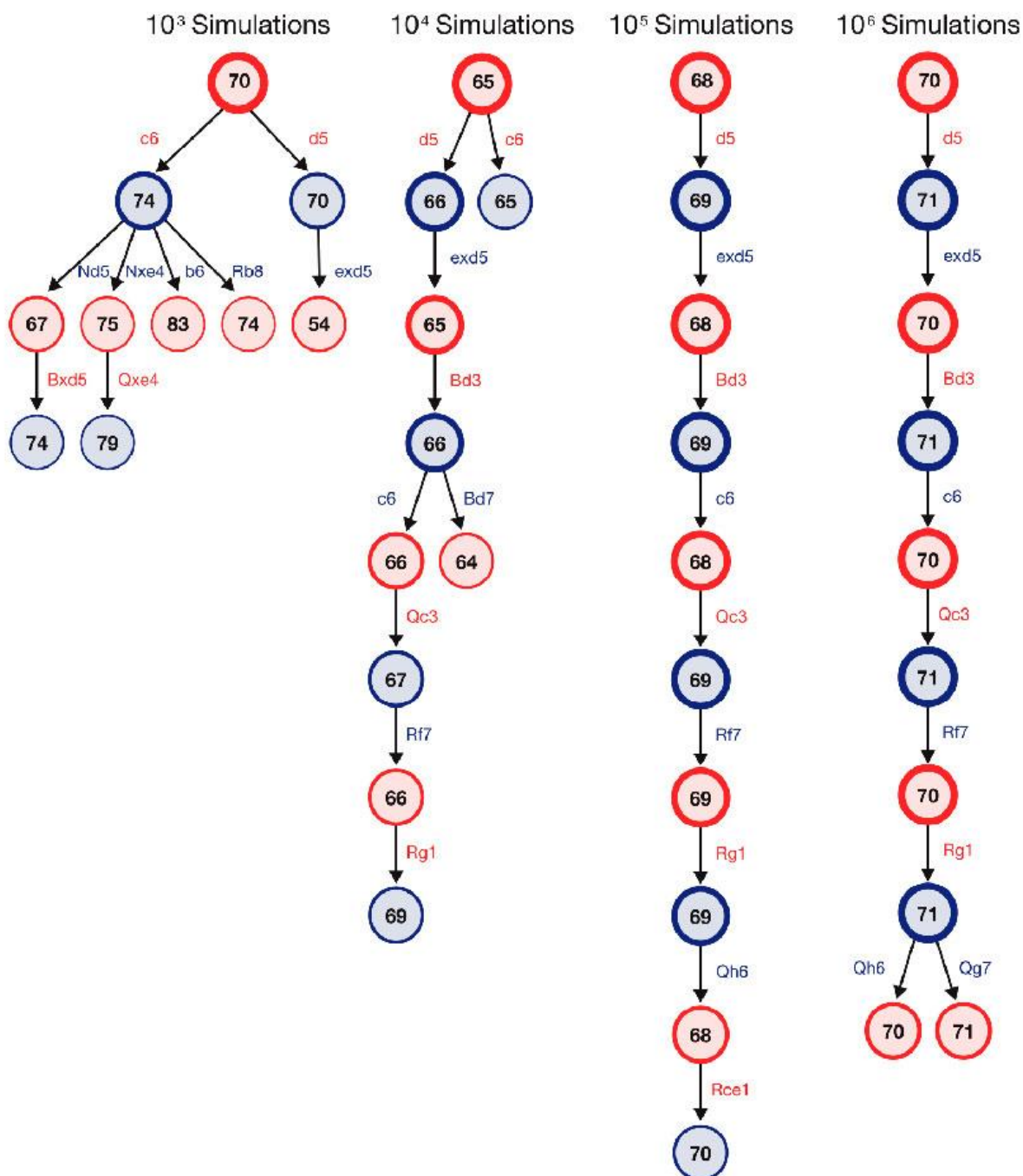
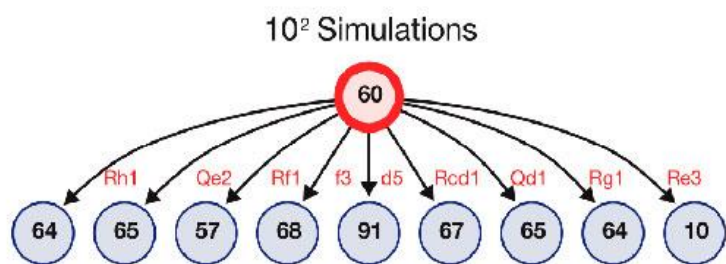
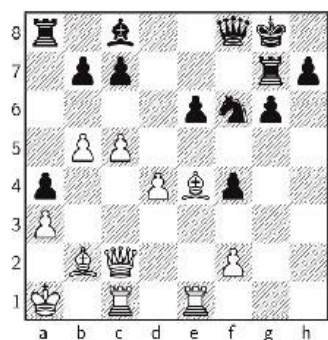


**Рис. 4.1. Пример поиска хода нейросетью AlphaZero
(шахматный движок)**

Нейронные сети в управлении могут использоваться двумя основными способами:

- **Как аппроксиматоры сложных функций в системах автоматического управления.** В традиционной теории управления есть понятие регулятора, который на основе отклонения системы от заданного состояния рассчитывает управляющее воздействие. Нейронная сеть может выступать таким регулятором, особенно если объект управления сложный и не описывается легко математическими уравнениями. Например, нейросеть может принимать на вход вектор состояния (ошибки по каждому параметру) и выдавать управляющий сигнал, пытаясь имитировать оптимального оператора. Обучить такую сеть можно по данным: либо из экспериментов (обучение с учителем – когда есть записи действий человека-оператора или оптимального алгоритма), либо с использованием методов оптимизации (обучение с подкреплением, о нем далее). В системах мониторинга нейронные сети могут обрабатывать потоки данных датчиков и «решать», все ли в норме. Например, сеть-аномалий может по телеметрии с VR-устройства (температуре, нагрузке) предсказать скорый отказ или необходимость калибровки – тем самым осуществляя управление в виде предупреждения оператора.

- **Как интеллектуальные агенты, принимающие решения.** Здесь мы приближаемся к теме обучения с подкреплением: нейронная сеть становится «мозгом» агента, который взаимодействует со средой и выбирает действия. Например, в игре (или VR-симуляции) нейросеть может управлять поведением NPC, делающим среду реалистичней (вместо скриптового поведения враг обучается и адаптируется). Или в виртуальном тренажере водительского мастерства можно реализовать автомобиль, управляемый нейронным «автопилотом», который реагирует на действия обучаемого (например, корректно проезжает перекрестки, тем самым создавая реалистичную обстановку на дороге). В этих случаях сеть обычно

получает на вход **состояние среды** (например, для автопилота – лидарная карта окружения, позиции других машин, состояние светофора), и выдает **действие** (поворот руля, нажатие газа/тормоза). Такой подход активно используется, например, в современных играх – нейросеть может управлять персонажем, делая решения более непредсказуемыми и интересными для игрока.

Для мониторинга нейросети применяют и в задачах анализа поведения пользователя. Например, анализируя траекторию взгляда и движения в VR, обученная модель может определять, устал ли пользователь, не требуется ли ему перерыв (по косвенным признакам). Это форма «управления» состоянием пользователя – система решает, когда вмешаться (например, остановить тренировку, если видит, что человек вымотан). Подобные интеллектуальные системы мониторинга часто строятся на архитектурах рекуррентных сетей или современных трансформеров, когда важно учесть временную динамику сигналов.

Важный класс – **системы с обучением с подкреплением (reinforcement learning, RL)**. В них нейронная сеть учится управлять, взаимодействуя с миром через пробы и ошибки, получая вознаграждение за успешные действия. Это мощный подход, заслуживающий отдельного рассмотрения.

4.2. ОБУЧЕНИЕ С ПОДКРЕПЛЕНИЕМ: ОСНОВЫ И ПРИМЕНЕНИЕ В VR/AI

Обучение с подкреплением (RL) – метод машинного обучения, при котором агент самостоятельно учится на основе сигналов вознаграждения из среды. Формально агент находится в некотором **состоянии** среды (state s), выбирает **действие** a , среда переходит в новое состояние s' и выдает агенту **вознаграждение** r . Цель агента – максимизировать

суммарное полученное вознаграждение (может быть дисконтировано во времени). Этот цикл повторяется многократно, агент улучшает свою стратегию выбора действий (политику) на основе опыта.

Важно понимать, что в RL нет «правильных ответов» как в обучении с учителем – агент сам пробует действия и учится на отклике среды. Поэтому RL хорошо подходит для задач, где трудно собрать размеченные данные, но можно симулировать процесс взаимодействия. Многие задачи управления являются именно такими.

Пример из VR: допустим, мы хотим обучить виртуального персонажа ходить. У нас есть симулированный человечек (с физическим скелетом), наша цель – он должен научиться самостоятельно двигать суставами, чтобы передвигаться вперед. Мы можем задать вознаграждение r как пройденное расстояние по прямой (чем дальше прошел – тем больше награда) и штрафовать за падение. Агент – это «мозг» персонажа, он наблюдает, например, наклоны суставов, скорость, контакт с землей (состояние s) и может управлять усилиями в суставах (действия a – значения для сервомоторов). Путем проб и ошибок, получая положительную награду за движение вперед и отрицательную при падении, со временем агент научится ходить. Такой подход продемонстрирован в Unity ML-Agents: четырехногий или двуногий персонаж может самостоятельно научиться ходить и даже бегать.

С формальной точки зрения, среду часто моделируют как **марковский процесс принятия решений (MDP)**, где есть множество состояний, действий, функция вознаграждения и (возможно) случайности переходов. RL-алгоритмы пытаются приблизить **оптимальную политику** или **функцию ценности** состояний. Многие современные подходы используют нейросети для представления политики или ценности, поскольку пространство состояний/действий может быть огромным (например, пиксельные изображения как состояние).

Простое определение: «Обучение с подкреплением – метод машинного обучения, при котором система (агент) обучается, взаимодействуя с некоторой средой. Среда откликается на действия агента сигналами подкрепления (вознаграждения или наказания) [7]. По сути, агент учится, какие действия приводят к награде, а какие – нет, подобно тому, как дрессируют животных или как человек учится на своих ошибках.

Алгоритмы RL включают такие, как **Q-learning**, **SARSA**, **DQN** (глубокое Q-обучение), методы **Policy Gradient** (**REINFORCE**), **Actor-Critic**, **DDPG**, **PPO**, **SAC** и многие другие. Без углубления в них, отметим общую схему процесса обучения агента:

Псевдокод обучения с подкреплением

for episode in range(num_episodes):

state = env.reset() # начать новый эпизод, получить начальное состояние

total_reward = 0

while True:

action = agent.choose_action(state) # выбрать действие
(по текущей политике)

next_state, reward, done, info = env.step(action) # выполнить действие в среде

agent.learn(state, action, reward, next_state, done) # обучить агента на полученном опыте

state = next_state

total_reward += reward

if done: # если эпизод завершен (например, достигнута цель или провал)

break

print(f»Episode {episode}: total reward = {total_reward}»)

В этом цикле агент может обновлять свою стратегию после каждого шага или по завершении эпизода (в зависимости от алгоритма). В случае нейросетевого агента, внутри `agent.learn` происходит обновление весов нейросети (например, через градиентный спуск) на основе ошибки между предсказанной ценностью действия и фактически полученным вознаграждением + будущей ценностью.

Применения RL уже нашли место и в VR, и в реальном мире:

- **Игровые AI.** Компания DeepMind показала, что RL-агенты могут превзойти человека в сложных видеоиграх. В контексте VR это означает, что можно создавать виртуальных противников или партнеров, обученных побеждать/помогать пользователю оптимальным образом.

- **Робототехника.** Существует направление *Sim2Real* – обучение роботов в виртуальной симуляции (где можно падать сколько угодно) с последующим переносом стратегии в реальный мир. VR-тренажеры техники могут использовать тот же подход: обучить модель управления машиной в симуляторе, а потом использовать на реальной машине. Это снижает риски и затраты. Например, в виртуальном складе можно обучить AI-агента оптимально управлять погрузчиком, прежде чем доверить управление настоящим погрузчиком.

- **Управление сложными системами.** Нейросети с RL применяются для задач управления трафиком (светофоры учатся оптимизировать потоки машин), управления ресурсами (умные grids). В VR эти же принципы могут использоваться для управления виртуальными толпами (*crowd simulation*), где агент решает, как должна группа людей эвакуироваться, или для адаптивного управления сложностью сценария (агент-RL подстраивает интенсивность вызовов исходя из того, как хорошо справляется пользователь, максимизируя его вовлеченность).

- **Мониторинг и исправление ошибок пользователя.** Интересное применение – агент следит за действиями обучаемого в VR и вмешивается,

если видит ошибку. Например, в хирургическом тренажере AI может в реальном времени оценивать, правильно ли стажер выполняет операцию, и давать подсказку или перехватывать управление инструментом в случае опасности (как автокорректор). Такое агентское поведение тоже можно задавать через RL: агент получает отрицательное подкрепление, если стажер совершил критическую ошибку и не был исправлен вовремя, поэтому агент учится вовремя вмешиваться оптимально.

Стоит помнить, что RL в чистом виде требует **много взаимодействий** со средой (эпизодов), чтобы обучиться. Если среда – симуляция, это не проблема (можно ускорить время симуляции, запустить параллельно много копий). В VR в реальном времени обучать что-то сложно, но можно предварительно обучить в симуляторе, а потом уже интегрировать модель.

Например, Unity ML-Agents как раз позволяет натренировать агента (скажем, физический персонаж, стреляющий из лука по мишеням) внутри Unity-проекта, а затем заморозить его «мозг» и вставить в игру. Такой персонаж будет управляться нейросетью, которая обучилась метко стрелять (целью обучения было максимизировать попадания в центр, т.е. награда – обратная дистанция от центра мишени). Для игрока такой противник будет выглядеть как довольно умный лучник. Если бы разрабатывали классическим способом, пришлось бы прописывать логику, целясь и учитывая баллистику, а сеть научилась сама.

4.3. ЗАДАЧИ УПРАВЛЕНИЯ

В ВИРТУАЛЬНЫХ ТРЕНАЖЕРАХ: ПРИМЕРЫ РЕШЕНИЙ

Рассмотрим конкретнее, какие **задачи управления** возникают в VR-тренажерах и как они решаются:

- **Управление виртуальными объектами-симуляциями.** Например, тренажер автомобиля: помимо пользователя, в сцене движутся другие

машины. Можно пойти простым путем – задать для них фиксированные траектории или записанные маневры. Но это предсказуемо и не учит реагировать на неожиданные ситуации. Вместо этого можно внедрить AI-водителей для этих машин. Задача: управлять автомобилем (скорость, поворот) так, чтобы, например, соблюдая ПДД, добраться до точки назначения. Это типичная RL-задача (похожая на автономный автомобиль). Решение: обучить агента на симулированных дорожных ситуациях (светофоры, пешеходы), награда – быстрая и безопасная езда. В итоге AI-водитель будет адекватно реагировать на действия обучаемого: тормозить, если тот выскочил под нос, объезжать препятствия и т.п. Такое поведение нельзя получить скриптами на все случаи – а агент сам найдет оптимальный паттерн.

- **Адаптивное управление сценарием.** В тренажерах часто важно подстраивать сложность и условия под уровень ученика. Задача: если новичок – давать более простые задачи, если продвинутый – усложнять, поддерживать уровень вызова. Можно реализовать это как задачу управления: контролируемые параметры – например, скорость появления неожиданных событий, число одновременно действующих факторов. AI-агент (или просто алгоритм) наблюдает за успешностью пользователя (скорость выполнения, количество ошибок) и решает, увеличивать или уменьшать сложность. Простое правило можно закодировать вручную, но можно и обучить агента с подкреплением: награда будет максимальна, когда ученик находится «на грани» своих возможностей (например, выполняет задания с 70...80%-ным успехом – не слишком легко, но и не проваливает их полностью). Агент, пробуя повышать или снижать сложность, научится удерживать эту грань. В результате получаем **персонализированное обучение** – каждому пользователю система предъявляет оптимальные для него задачи.

- **Управление группой персонажей.** В имитационных VR может быть толпа людей (например, эвакуация при пожаре, где помимо обучаем-

мого присутствуют виртуальные люди). Управлять толпой сложно: это множество агентов, возможно взаимодействующих. Здесь могут применяться алгоритмы многоагентного обучения с подкреплением (Multi-agent RL), где агенты обучаются совместно. Либо можно использовать известные модели толпы (boids, социальные силы), но настроить их параметры с помощью оптимизации: например, генетический алгоритм подбирает такие параметры модели толпы, чтобы достичь нужной плотности потока и скорости (эволюционный подход тоже вид обучения без учителя).

- **Обратная связь и коррекция.** Еще задача: когда выдавать подсказку обучаемому? Постоянные подсказки мешают, а отсутствие – может привести к тому, что он застрянет. Это можно сформулировать как задачу оптимального управления уровнем помощи. Агент наблюдает, что делает пользователь, и имеет действие «дать подсказку» или «промолчать». Награда – допустим, успешное выполнение задания с минимальным числом подсказок. Тогда агент RL научится подавать подсказку, только если видит, что иначе задача будет провалена. В реальности пока чаще делают скриптовые триггеры (если пользователь стоит 30 с без действия – дать намек), но AI мог бы более гибко решать, когда именно и какую подсказку дать.

Мониторинг с помощью AI также можно считать задачей управления – управление безопасностью или качеством выполнения. В некоторых критических VR-тренажерах (например, военных или авиакосмических) в систему закладывают подсистему, которая следит за психофизиологическим состоянием обучаемого, как он реагирует на стресс, и может остановить сценарий, если видит риск. Такие подсистемы могут использовать ML (классификацию состояний по сенсорам). По сути, это автоматическое управление состоянием симуляции на основе состояния пользователя – тоже интересный аспект.

Подведем итог: **нейронные сети и методы ИИ расширяют возможности систем управления в VR и за ее пределами.** Обучение с подкреплением позволяет создавать агентов, которые управляют объектами или сценарием, приближаясь по эффективности к человеку или даже превосходя его в узких задачах. Это находит применение как для создания умных виртуальных напарников/противников, так и для адаптации обучения, оптимизации сложных процессов. Конечно, внедряя такие решения, разработчик должен учитывать и ограничения: нейросетевой агент – «черный ящик», его трудно напрямую верифицировать, необходимо обширное тестирование, чтобы убедиться, что он не сделает чего-то непредусмотренного (особенно критично, если мы доверяем ИИ управление реальным оборудованием после симуляции).

Тем не менее тенденция однозначна: VR и AI связываются все сильнее. Уже сейчас существуют системы, где **виртуальная реальность + искусственный интеллект** дают синергетический эффект: VR обеспечивает реализм среды, а AI – реализм поведения и интеллектуальную поддержку. Будущие тренажеры, игры, обучающие системы станут еще более интерактивными и «умными» благодаря этому тандему.

КОНТРОЛЬНЫЕ ВОПРОСЫ К РАЗДЕЛУ 4

1. Что понимается под задачами управления в контексте приложений искусственного интеллекта и виртуальной реальности? Чем они отличаются от задач классификации/регрессии?
2. Приведите примеры, как нейронные сети могут использоваться для управления какими-либо объектами или процессами.
3. Какое основное отличие обучения с подкреплением (reinforcement learning) от обучения с учителем? Почему RL особенно подходит для задач управления агентами?
4. Опишите цикл взаимодействия агента и среды в задаче обучения с подкреплением. Какие элементы входят в этот цикл?
5. В чем заключается роль функции вознаграждения в обучении с подкреплением? Что может произойти, если задать неудачную (неполную или неточно отражающую цель) функцию вознаграждения?
6. Для задачи автономного вождения в виртуальном городе: какие параметры состояния могли бы быть поданы агенту, какие действия он должен выдавать и как можно определить вознаграждение?
7. Что такое метод обучения *Actor-Critic* или *Deep Q-Network (DQN)*? (Достаточно пояснить общее назначение этих методов в контексте RL.)
8. Задача: виртуальный персонаж должен научиться перепрыгивать через препятствия. Как можно организовать обучение с подкреплением для этой задачи (формулировка состояния, действие, вознаграждение)?
9. В чем вы видите преимущества применения обученных с подкреплением агентов в VR-играх или тренажерах по сравнению со скриптовыми (запрограммированными вручную) противниками или сценариями?
11. Какие риски и сложности связаны с использованием нейросетевых агентов для управления в критически важных приложениях (например, в тренажере самолета или автомобиля)? Как можно снизить эти риски? В чем заключаются ключевые отличия между управлением

на основе моделей (Model Predictive Control) и управлением с помощью обученных нейросетевых агентов?

12. Как можно формализовать задачу управления для NPC (неигрового персонажа) в виде марковского процесса принятия решений (MDP)? Опишите элементы MDP для конкретного примера.

13. Что такое «симуляционно-реальный разрыв» (Sim2Real Gap) и какие методы используются для его преодоления при переносе политики, обученной в VR, в реальный мир?

14. Какие преимущества и недостатки у подхода «обучение с учителем» для имитации человеческих действий по сравнению с «обучением с подкреплением» для задач управления?

15. Опишите, как работает архитектура Actor-Critic в обучении с подкреплением. Какую роль играют «актер» (actor) и «критик» (critic) в этом процессе?

16. Что такое «разведка-эксплуатация» (exploration-exploitation dilemma) в RL и почему это фундаментальная проблема для агентов, обучающихся в VR-средах?

17. Какие методы используются для обеспечения безопасности (Safe RL) при обучении агентов управлению в критически важных симуляциях, таких как авиатренажеры?

18. Как можно использовать обратное обучение с подкреплением (Inverse Reinforcement Learning) для того, чтобы агент научился управлению, наблюдая за действиями эксперта-человека в VR?

19. В чем состоят основные трудности применения многоагентного обучения с подкреплением (Multi-Agent RL) для управления группой объектов в VR (например, толпой или роем дронов)?

20. Какой вклад вносят нейросетевые модели управления (например, трансформеры) в решение задач управления, требующих учета долгосрочной временной зависимости и контекста?

ЗАКЛЮЧЕНИЕ

В рамках данного учебного пособия рассмотрен комплексный анализ ключевых аспектов разработки приложений, интегрирующих технологии искусственного интеллекта и виртуальной реальности. Рассмотрение вопросов системного анализа, обработки данных, решения задач управления, проектирования архитектуры и инструментария разработки позволило сформировать целостное представление о современных подходах к созданию AI/VR-систем.

Проведенное исследование демонстрирует, что успешная разработка AI/VR-приложений требует глубокого междисциплинарного подхода, сочетающего знания в области программной инженерии, машинного обучения, компьютерной графики и человеко-компьютерного взаимодействия. Особое значение приобретает системный анализ на начальных этапах проектирования, позволяющий формализовать требования и выбрать оптимальную архитектуру, обеспечивающую производительность, масштабируемость и надежность системы.

Ключевым выводом является осознание критической важности обработки данных в AI/VR-приложениях. Эффективный сбор, предварительная обработка и анализ данных как от VR-устройств, так и для обучения AI-моделей становятся определяющим фактором качества итогового решения. Рассмотренные методы классификации, регрессии и обучения с подкреплением открывают широкие возможности для создания интеллектуальных систем, способных адаптироваться к пользователям и решать сложные задачи управления в реальном времени.

Значительное внимание уделено архитектурным паттернам и инструментам разработки, поскольку правильный выбор технологического стека непосредственно влияет на эффективность процесса создания

AI/VR-приложений. Единые стандарты, такие как OpenXR, и современные игровые движки, такие как Unity и Unreal Engine, существенно упрощают разработку кросс-платформенных решений, в то время как специализированные фреймворки для ИИ обеспечивают seamless-интеграцию интеллектуальных компонентов.

Перспективы развития AI/VR-направления связаны с дальнейшей конвергенцией технологий, созданием более эффективных методов интеграции AI-моделей в реальном времени, развитием инструментов для совместной работы распределенных команд и повышением доступности технологий для широкого круга разработчиков. Особый потенциал заключается в применении рассмотренных подходов в образовательных, промышленных и медицинских тренажерах, где сочетание иммерсивности виртуальной реальности и адаптивности искусственного интеллекта способно обеспечить качественно новый уровень обучения и профессиональной подготовки.

Таким образом, материалы пособия создают прочный фундамент для дальнейшего изучения и практического применения технологий искусственного интеллекта и виртуальной реальности, вооружая специалистов необходимыми знаниями и методиками для создания инновационных решений на стыке этих перспективных направлений.

СПИСОК ЛИТЕРАТУРЫ

1. Разработка VR и AR приложений – что это? : [веб-сайт] / Portal VR. – URL : <https://portal-vr.ru/razrabotka-vr-i-ar-prilozhenij-chto-eto/> (дата обращения: 15.05.2025).
2. Обухов, А. Д. Микросервисная архитектура виртуальных тренажерных комплексов / А. Д. Обухов, А. А. Волков, А. О. Назарова // Информатика и автоматизация. – 2022. – Т. 21, № 6. – С. 1265 – 1289.
3. Предварительная обработка данных для машинного обучения : [веб-сайт] / sun_lil // Хабр. – URL : <https://habr.com/ru/articles/800973/> (дата обращения: 15.05.2025).
4. Алексеев, Г. Введение в машинное обучение : [веб-сайт] / Г. Алексеев // Хабр. – URL : <https://habr.com/ru/articles/448892/> (дата обращения: 15.05.2025).
5. Focus on OpenXR – What You Need to Know : [веб-сайт] / Ultralep Documentation. – URL : <https://docs.ultraleap.com/ultralab/openxr-blog.html> (дата обращения: 15.05.2025).
6. Components of a VR system : [веб-сайт] // ResearchGate. – URL : https://www.researchgate.net/figure/Components-of-a-VR-system_fig2_317244129 (дата обращения: 15.05.2025).
7. Обучение с подкреплением : [веб-сайт] // Википедия. – URL : https://ru.wikipedia.org/wiki/Обучение_с_подкреплением (дата обращения: 15.05.2025).
8. Unity – User Manual (Version 2022.3) [Электронный ресурс] / Unity Technologies. – URL : <https://docs.unity3d.com/Manual/index.html> (дата обращения: 15.05.2025).

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ	3
1. СИСТЕМНЫЙ АНАЛИЗ ПРИЛОЖЕНИЙ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА И ВИРТУАЛЬНОЙ РЕАЛЬНОСТИ	5
1.1. МЕТОДОЛОГИИ РАЗРАБОТКИ И ЖИЗНЕННЫЙ ЦИКЛ AI/VR-СИСТЕМ	5
1.2. ФОРМАЛИЗАЦИЯ ТРЕБОВАНИЙ И ПРОЕКТИРОВАНИЕ: СХЕМЫ И АЛГОРИТМЫ.....	10
1.3. ОСНОВНЫЕ ТЕХНОЛОГИИ: ИГРОВЫЕ ДВИЖКИ, ФРЕЙМВОРКИ И БИБЛИОТЕКИ ИИ	14
КОНТРОЛЬНЫЕ ВОПРОСЫ К РАЗДЕЛУ 1	22
2. ОБРАБОТКА ДАННЫХ В ПРИЛОЖЕНИЯХ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА	24
2.1. ПРЕДВАРИТЕЛЬНАЯ ОБРАБОТКА ДАННЫХ ДЛЯ МОДЕЛЕЙ МАШИННОГО ОБУЧЕНИЯ	24
2.2. ЗАДАЧИ КЛАССИФИКАЦИИ И РЕГРЕССИИ: ПОСТАНОВКА И РЕШЕНИЕ.....	32
КОНТРОЛЬНЫЕ ВОПРОСЫ К РАЗДЕЛУ 2	44
3. ОБРАБОТКА ДАННЫХ В ПРИЛОЖЕНИЯХ ВИРТУАЛЬНОЙ РЕАЛЬНОСТИ	46
3.1. КЛАССИФИКАЦИЯ И ИСТОЧНИКИ ДАННЫХ В VR.....	47
3.2. СБОР ДАННЫХ: API И ИНСТРУМЕНТЫ (ПРИМЕР OPENXR)	51
3.3. ОБРАБОТКА ДАННЫХ В VR-ТРЕНАЖЕРАХ И СИСТЕМАХ.....	57
КОНТРОЛЬНЫЕ ВОПРОСЫ К РАЗДЕЛУ 3	61
4. РЕШЕНИЕ ЗАДАЧ УПРАВЛЕНИЯ В ПРИЛОЖЕНИЯХ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА И ВИРТУАЛЬНОЙ РЕАЛЬНОСТИ	63
4.1. НЕЙРОННЫЕ СЕТИ ДЛЯ УПРАВЛЕНИЯ И МОНИТОРИНГА	63
4.2. ОБУЧЕНИЕ С ПОДКРЕПЛЕНИЕМ: ОСНОВЫ И ПРИМЕНЕНИЕ В VR/AI	66
4.3. ЗАДАЧИ УПРАВЛЕНИЯ В ВИРТУАЛЬНЫХ ТРЕНАЖЕРАХ: ПРИМЕРЫ РЕШЕНИЙ.....	70
КОНТРОЛЬНЫЕ ВОПРОСЫ К РАЗДЕЛУ 4	74
ЗАКЛЮЧЕНИЕ	76
СПИСОК ЛИТЕРАТУРЫ	78

Учебное электронное издание

ОБУХОВ Артем Дмитриевич
ВОЛКОВ Андрей Андреевич
ТЕСЕЛКИН Даниил Вячеславович
НИКИТНИКОВ Юрий Витальевич
ШИЛЬЦЫН Максим Александрович

СИСТЕМНЫЙ АНАЛИЗ,
ОБРАБОТКА ДАННЫХ И УПРАВЛЕНИЕ
В ПРИЛОЖЕНИЯХ ИСКУССТВЕННОГО
ИНТЕЛЛЕКТА И ВИРТУАЛЬНОЙ РЕАЛЬНОСТИ

Учебное пособие

Редактор Л. В. Комбарова
Графический и мультимедийный дизайнер Т. Ю. Зотова
Обложка, упаковка, тиражирование Л. В. Комбаровой

ISBN 978-5-8265-3030-6



Подписано к использованию 13.05.2026.
Тираж 50 шт. Заказ № 66

Издательский центр ФГБОУ ВО «ТГТУ»
392000, г. Тамбов, ул. Советская,
д. 106/5, пом. 2, к. 14
Телефон 8(4752)63-81-08.
E-mail: izdatelstvo@tstu.ru