Модуль № 1

Название: Программная инженерия для приложений искусственного интеллекта

Образовательные результаты:

* Студент знает особенность дисциплины Программной инженерии, ее структуру.
* Студент знает, как устроен жизненный цикл ИИ-приложения.
* Студент может использовать готовые модели для решения практических задач.
* Студент умеет разрабатывать программное обеспечение в команде.

# В этом модуле:

В настоящее время искусственный интеллект перестал казаться чем-то фантастическим. Мы постоянно сталкиваемся с системами искусственного интеллекта: от уже ставших привычных голосовых помощников и ассистентов, автоматических переводчиков, чат-ботов, до беспилотных такси, тестирование которых на дорогах Москвы Яндекс начинает осенью 2021 года, и системы оплаты проезда с использованием распознавания лиц, которая внедряется в Московском метро.

Большая часть систем искусственного интеллекта использует машинное обучение. Однако для успешной работы таких систем недостаточно просто обучить высокоточную модель машинного обучения. Необходимо разработать программное обеспечение, которое запускает модель на различных устройствах: компьютерах, смартфонах, серверах в облаке, встраиваемых системах и т.п. Такое программное обеспечение должно быть отказоустойчивым и высокопроизводительным, в противном случае его использование может привести к катастрофам с человеческими жертвами (например, в случае беспилотного автомобиля). Также программное обеспечение должно иметь возможность обслуживать тысячи или миллионы пользователей одновременно, например, в облачных системах автоматического перевода.

Этот модуль посвящен рассмотрению основ программной инженерии – дисциплины, изучающей приложение инженерного подхода к разработке программного обеспечения с целью создания надежных, высокопроизводительных и отказоустойчивых программных систем.

В модуле вы:

* Познакомитесь с предметной областью программной инженерии.
* Научитесь использовать готовые модели машинного обучения в программных системах.
* Изучите основы командной разработки.

# Модуль № 1. Юнит № 1. Организация курса

В курсе “Программная инженерия” вы познакомитесь с основными инструментами, которые применяются для создания крупных программных систем. Курс будет длиться два семестра. В первом семестре вы изучите базовые основы программной инженерии, необходимые для командной разработки приложений искусственного интеллекта, а во втором: продвинутые инструменты, которые помогут сделать вашу работу более эффективной.

Первый семестр курса включает 5 недель, каждая из которых посвящена изучению отдельного раздела программной инженерии.

В курсе будет использоваться язык Python и библиотеки для этого языка.

**Неделя 1. Программная инженерия для приложений искусственного интеллекта**

Вы узнаете, что представляет собой программная инженерия как дисциплина, какие особенности есть в проектах по созданию крупных приложений искусственного интеллекта. Также вы узнаете, как разрабатывать приложения искусственного интеллекта на основе готовых библиотек машинного обучения. Мы подробно рассмотрим популярные Python библиотеки Hugging Face и TensorFlow.

По настоящему большую систему искусственного интеллекта невозможно создать в одиночку. Поэтому на первой неделе мы рассмотрим командную разработку программного обеспечения и изучим наиболее популярный инструмент командной разработки – систему git и облачный сервис GitHub.

**Неделя 2. Архитектура приложений искусственного интеллекта**

Вторая неделя посвящена важной теме: как организованы приложения искусственного интеллекта. Вы изучите наиболее популярные подходы к архитектуре программного обеспечения и особенности их применения для приложений искусственного интеллекта.

Наибольшей популярностью сейчас пользуется микросервисная архитектура, в которой приложение строится из набора небольших сервисов, взаимодействующих по сети между собой. Каждый такой сервис предоставляет API (application programming interface) – набор функций, доступных внешним разработчикам. Мы рассмотрим инструменты работы с API, а также научимся использовать API популярных сетевых сервисов: социальные сети, крупные сетевые порталы и т.п.

**Неделя 3. Разработка API для приложений искусственного интеллекта**

На третьей неделе мы продолжим изучать архитектуру приложений искусственного интеллекта и API. Вы узнаете, как создать API для вашей модели машинного обучения и организовать к ней доступ по сети. В качестве инструмента для создания API мы будем использовать популярную систему FastAPI на Python.

**Неделя 4. Развертывание приложений искусственного интеллекта в облаке**

На четвертой неделе курса мы рассмотрим, как обеспечить доступ другим разработчикам к вашей модели машинного обучения по сети. Для этого нужно развернуть приложение, реализующее созданный вами API на каком-либо сервере, подключенном к сети. Сейчас для этой цели чаще всего используют облачные платформы.

Мы рассмотрим облачную платформу Heroku, научимся разворачивать на этой платформе приложение на Python, а также рассмотрим интеграцию Heroku с GitHub для автоматического развертывания приложений.

**Неделя 5. Тестирование программного обеспечения**

Тестирование – обязательная часть процесса промышленной разработки программного обеспечения. Приложения искусственного интеллекта также невозможно разработать без качественного тестирования. В противном случае пользователи вашего приложения будут получать ошибки и жаловаться вам. Если ошибок будет много, то пользователи могут совсем отказаться от вашего приложения.

Тестирование приложений на Python мы рассмотрим на пятой неделе. Также вы познакомитесь с концепцией программной инженерии Continuous Integration (CI, непрерывная интеграция) и узнаете, как она позволяет повысить качество программного обеспечения.

**Проект**

Обучение на курсе организовано на основе проекта, который вы будете делать командами из трех-четырех человек.

*Цель проекта:* развернуть в облаке модель машинного обучения и обеспечить к ней доступ через API. Модель вы можете выбрать сами. Мы рекомендуем использовать готовую модель из библиотек машинного обучения.

Проект мы будем реализовывать по шагам в течение всего курса. На первой неделе вы познакомитесь с готовыми библиотеками машинного обучения и сможете выбрать модель, для которой хотите реализовать приложение. Вы создадите репозиторий на GitHub, чтобы работать над приложением могли все участники вашей команды. На второй и третьей неделе вы создадите API для выбранной вами модели. На четвертой неделе настроите автоматическое разворачивание кода, реализующего API для вашей модели, из GitHub репозитория на облачную платформу Heroku. И, наконец, на пятой неделе вы разработаете тесты для своего приложения магинного обучения и настроите автоматический запуск тестов с помощью Continuous Integration на GitHub перед разворачиванием приложения на Heroku. Таким образом вы сможете быть уверены, что разворачиваемое приложение будет работать без ошибок.

Экзамен по курсу будет проходить в виде защиты проекта.

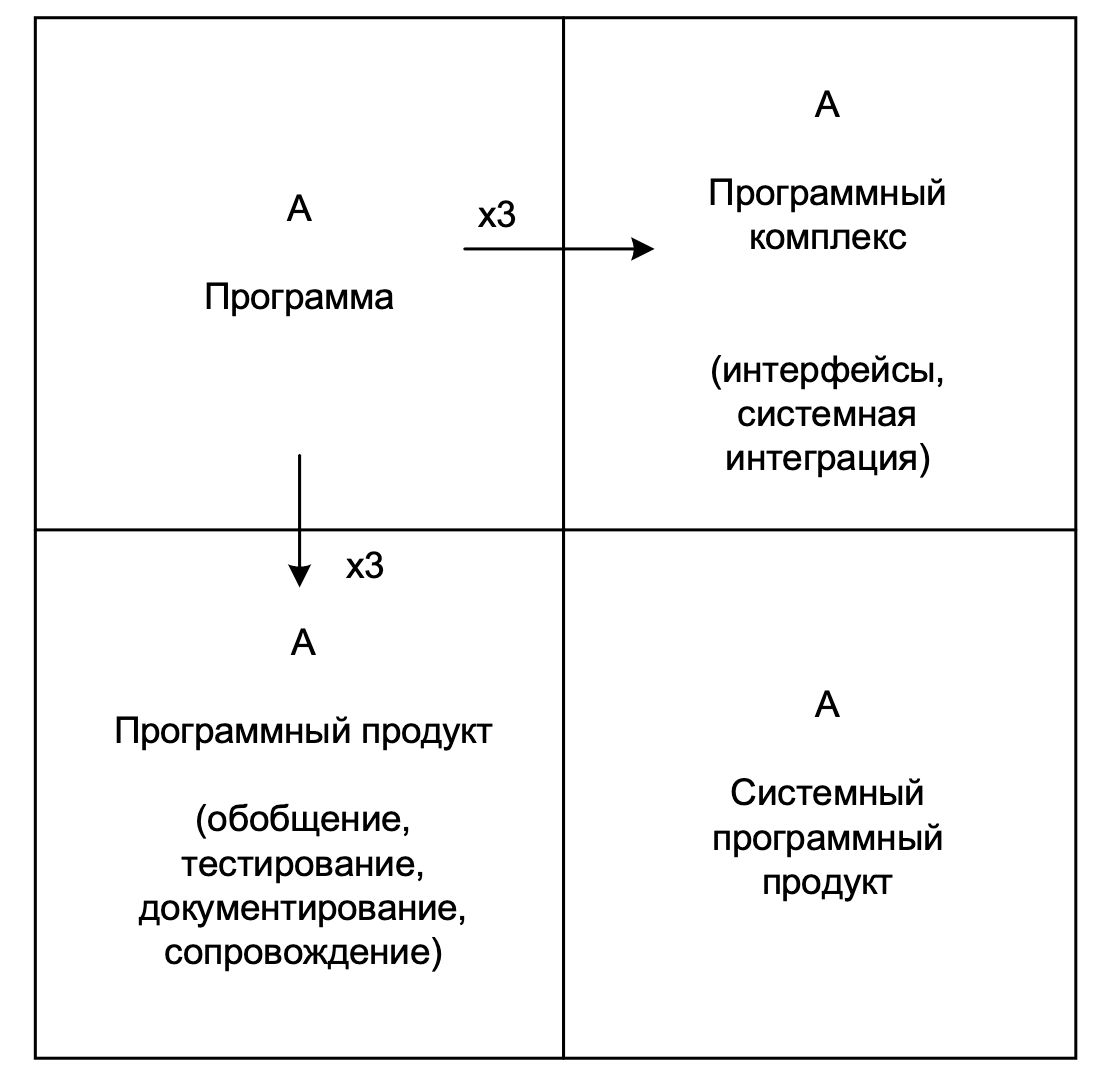
**Итоги**

* Курс состоит из пяти недель, во время которых вы будете изучать различные инструменты программной инженерии, помогающие создавать крупные приложения искусственного интеллекта.
* Обучение на курсе организовано на основе проекта, который вы будете выполнять по шагам на каждой неделе курса.
* Для получения оценки за курс необходимо реализовать и защитить проект.

# Модуль № 1. Юнит № 2. Программная инженерия как дисциплина

Приложения искусственного интеллекта, такие как система FacePay оплаты проезда с использованием распознавания лиц в Московском метро, голосовой помощник Алиса от Яндекса или автоматический переводчик от компании Google – это крупные программные системы, которыми пользуются тысячи или миллионы людей. Создают такие системы десятки или сотни разработчиков. Как показывает опыт, разработка крупных программных систем сильно отличается от разработки небольших программных продуктов, таких как мобильные приложения или сайты. В результате, крупные программные системы создаются долго, стоят дорого и часто работают нестабильно. Более того, некоторые крупные программные системы вообще не удается довести до работоспособного состояния.

Объяснение такой ситуации с разработкой крупных программных систем дает Фредерик Брукс в книге “Мифический человеко-месяц”. Разработку программной системы он представляет на схеме в виде квадрата, разделенного на четыре части.



Системный программный продукт по Фредерику Бруксу.

В левой части квадрата находится программа – реализация приложения, которое работает на компьютере разработчика и использовать которое может только его создатель. Например, разработчик может обучить модель на основе нейронной сети определять эмоциональную окраску текста: положительную или отрицательную. Как правило, для создания таких программ используется Jupyter Notebook. Код в ноутбуке обычно содержит обучение модели и несколько примеров ее применения для определения тональности текстовых вариантов текстов. Однако использовать такой код и обученную с его помощью модель вряд ли сможет кто-то, кроме авторов.

**Программный продукт**

Одна из осей изменений на схеме – это создание программного продукта на основе программы. Ключевое отличие программного продукта – это то, что его могут использовать сторонние пользователи, а не только авторы программы. Чтобы такое стало возможным, необходимо создать обобщенную версию программы, которая работает на разных платформах и может обрабатывать данные в разных форматах.

Программу на пути к программному продукту необходимо тщательно протестировать, опять же на различных платформах и с различными форматами данных, а также для различных вариантов применения, которые нужны пользователям.

Разработчики программы знают, как ее использовать, но сторонние пользователи не имеют об этом никакого представления. Поэтому для программного продукта необходимо разработать документацию. Чем выше качество документации, тем больше пользователей будет у вашего продукта. И наоборот, плохая документация способна оттолкнуть пользователей даже от хороших программ.

Программный продукт нуждается в сопровождении и технической поддержке. К сожалению, ошибки в программном обеспечении неизбежны и не всегда их удается найти с помощью тестирования. Пользователи программного продукта будут сталкиваться с ошибками и разработчикам необходимо их исправлять.

Чтобы программа с моделью машинного обучения стала программным продуктом, необходимо реализовать приложение, использующее такую модель. Например, на основе модели определения эмоциональной окраски текста может быть создано мобильное приложение, приложение в облаке или на другой платформе. Для приложения каждого типа необходимо написать документацию, протестировать и обеспечить поддержку.

По оценке Фредерика Брукса, создание программного продукта требует минимум в три раза больше времени и денег, чем программы.

**Программный комплекс**

Второе направление изменений – это включение программы в программный комплекс: множество программ, которые взаимодействуют между собой для решения какой-либо задачи.

Рассмотренная ранее нами программа определения эмоциональной окраски текста сама по себе вряд ли принесет большую пользу. Но ее можно применить в составе программного комплекса крупной компании, которая использует информацию из социальных сетей в процессе создания новых продуктов. Первая часть такого комплекса собирает в социальных сетях отзывы на существующие и создаваемые продукты компании. Второй частью может стать наша программ определения эмоциональной окраски отзывов. И третья часть – система аналитики, которая использует собранную информацию об эмоциональной окраске для принятия решений о направлениях разработки продуктов.

Однако, чтобы несколько программ могли работать совместно в составе комплекса, необходимо провести работу по их интеграции: определиться с форматами данных которые передаются от одной программы к другой, задать интерфейсы программирования API, определить протоколы, по которым будут передаваться данные.

Создание программного комплекса, также как и программного продукта, по оценке Фредерика Брукса требует минимум в три раза больше времени и денег, чем создание программы.

**Системный программный продукт**

Наиболее совершенной программной системой, согласно Фредерику Бруксу, является системный программный продукт. Он представляет собой программный комплекс, каждой частью которого является программный продукт: обобщенные, протестированные и документированные программы, для которых реализуется сопровождение.

Система проектирования продуктов с использованием аналитики отзывов пользователей в социальных сетях вполне может быть системным программным продуктом, если каждый ее компонент не разрабатывается самостоятельно, а используется готовый программный продукт.

Создание системного программного продукта требует минимум в 9 раз больших затрат времени и денег, чем разработка программы. Однако именно системные программные продукты наиболее полезны пользователям, т.к. они позволяют решать их задачи в комплексе. Большая часть практических приложений искусственного интеллекта, которые создаются сейчас и будут создаваться в ближайшие несколько лет являются системными программными продуктами.

**Программная инженерия**

Программная инженерия рассматривает применения инженерных методов для создания сложных программных систем. Фредерик Брукс в “Мифическом человеко-месяце” утверждает, что программирование занимает всего ⅙ часть времени создания сложной программной системы. Остальное время занимают другие активности, которые как раз и изучает программная инженерия.

Международная организация Институт инженеров электротехники и электроники (IEEE, Institute of Electrical and Electronics Engineers) разработала рекомендации по набору знаний специалистов по программной инженерии: Software Engineering Body of Knowledge (SWEBOK) (<https://www.computer.org/education/bodies-of-knowledge/software-engineering>). Основные разделы SWEBOK, которые необходимо изучать программным инженерам:

* Требования к программному обеспечению
* Проектирование программного обеспечения
* Тестирование программного обеспечения
* Поддержка программного обеспечения
* Управление конфигурацией программного обеспечения
* Менеджмент в программной инженерии
* Процессы в программной инженерии
* Модели и методы программной инженерии
* Качество программного обеспечения
* Профессиональные практики программной инженерии
* Экономика программной инженерии

Системное применение инструментов программной инженерии позволит вам успешно создавать, выводить в продуктивную эксплуатация и поддерживать сложные системы искусственного интеллекта.

В этом курсе мы рассмотрим вопросы проектирования и тестирования программного обеспечения, процессы в программной инженерии, качество программного обеспечения и некоторые профессиональные практики программной инженерии. Управление конфигурацией программного обеспечения с использованием подхода Infrastructure as a Code будет рассмотрено в курсе “Автоматизация администрирования DevOps”. Менеджмент и экономика программной инженерии будут рассмотрены в курсе “Управление проектами машинного обучения”.

**Итоги**

* Прикладные системы искусственного интеллекта являются системными программными комплексами – сложными программными системами.
* Затраты времени и денег на создание системных программных комплексов минимум в 9 раз превышают затраты на создание простых программ.
* Программирование в процессе создания программного комплекса занимает ориентировочно ⅙ времени. Остальное время уходит на проектирование, тестирование, документирование, интеграцию, сопровождение и другие активности.
* Программная инженерия изучает применение инструментов инженерии для создания крупных программных систем, таких как системные программные продукты.

**Тест**

Чем программный продукт отличается от программы

1. Программный продукт может использовать только его разработчик.
2. **Программный продукт протестирован, документирован, может работать на разных платформах с данными разных типов.**
3. Программный продукт использует согласованные с другими программами типы данных и протоколы.
4. Разработка программного продукта в три раза дешевле, чем разработка программы.

Во сколько раз затраты на разработку системного программного продукта превышают затраты на разработку программы

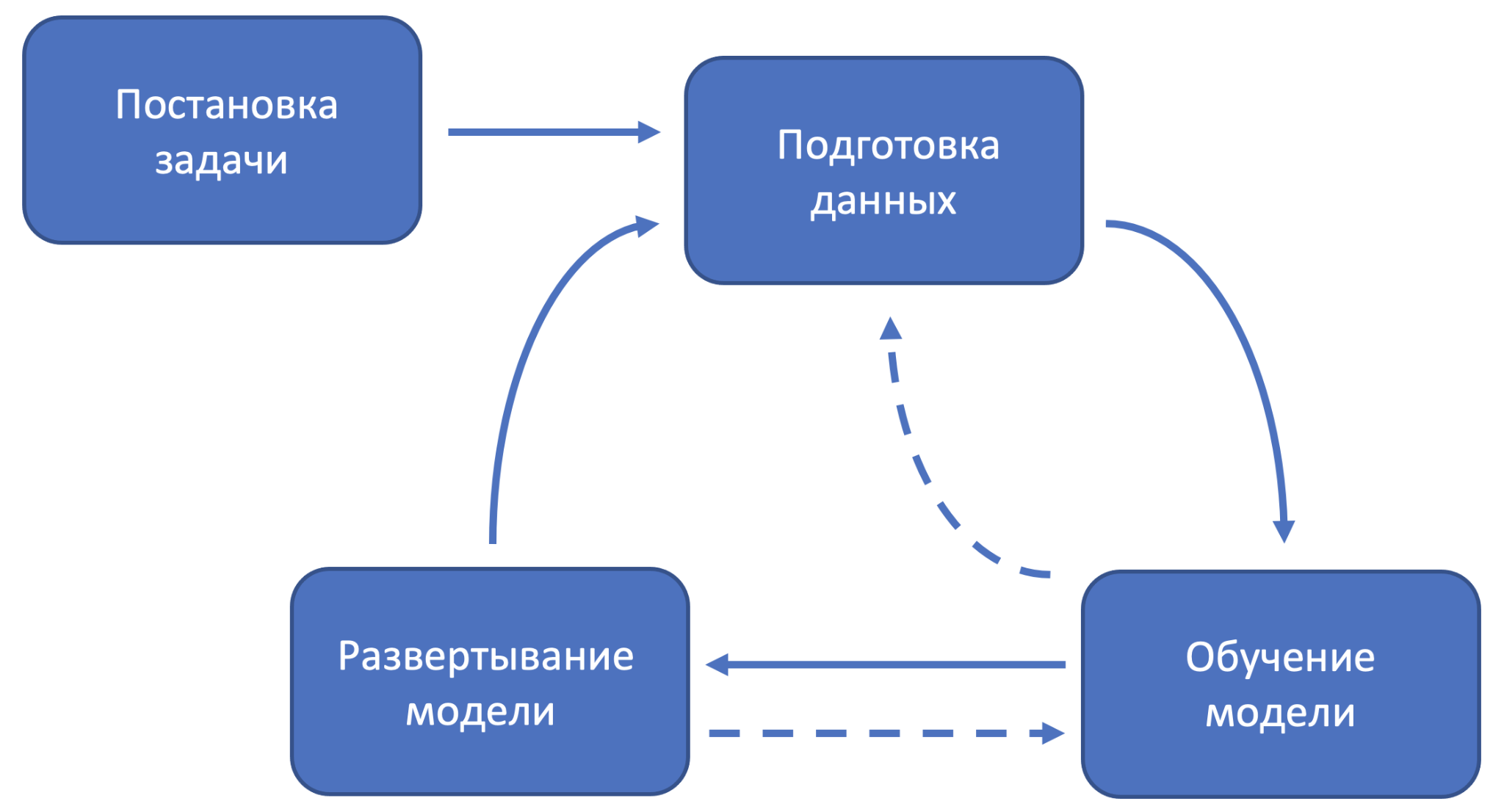
1. Минимум в 3 раза.
2. Минимум в ⅙ раза.
3. **Минимум в 9 раз.**
4. Более чем в ⅚ раза.

Что изучает программная инженерия

1. **Применения инженерных методов для создания сложных программных систем.**
2. Создание приложений искусственного интеллекта с помощью системного подхода.
3. Математические основы сложности алгоритмов программного обеспечения.
4. Алгоритмы машинного обучения.

# Модуль № 1. Юнит № 3. Жизненный цикл разработки приложений машинного обучения

Создание систем машинного обучения – это итеративный процесс, который состоит из нескольких этапов. Эндрю Ын (Andrew Ng), один из основателей Coursera, признанный специалист в области машинного обучения, предлагает следующий жизненный цикл разработки систем машинного обучения (https://www.coursera.org/specializations/machine-learning-engineering-for-production-mlops):



Жизненный цикл разработки приложения машинного обучения от Эндрю Ын

**Постановка задачи**

Работа над созданием системы машинного обучения начинается с постановки задачи. На этом этапе определяется цель проекта, над которым предстоит работать например, создание системы автоматического перевода. Задаются архитектурные требования к системе: должно ли это быть мобильное приложение, Web-приложение, или API, предназначенный для вызова другими приложениями. Необходимо определить и содержательные требования к приложению: продолжая рассматривать пример системы автоматического перевода мы можем решить, что наша система будет переводить с английского языка на русский и наоборот.

Также на этапе постановки задачи необходимо сформулировать ожидаемые требования к качеству работы системы машинного обучения. Для оценки качества перевода можно использовать метрику BLEU (bilingual evaluation understudy), желаемые значения должны быть в диапазоне от 50 до 60 (что соответствует уровню fluent translation) (https://cloud.google.com/translate/automl/docs/evaluate).

**Подготовка данных**

Следующий этап – это подготовка данных, на котором необходимо сформировать набор данных и произвести его разметку. Для системы автоматического перевода набор данных будет включать предложения на русском языке и соответствующий им перевод предложения на английский язык. Набор данных можно подготовить самостоятельно или использовать один из существующих.

**Обучение модели**

На третьем этапе жизненного цикла производится работа с моделью машинного обучения. Нужно выбрать тип модели, который будет использоваться, например, глубокие нейронные сети или какая-то из классических моделей машинного обучения. Затем выполняется обучение модели с использованием подготовленного на предыдущем этапе набора данных. Последний шаг – оценка качества работы обученной модели. Для системы автоматического перевода мы можем выбрать в качестве модели машинного обучения глубокую нейронную сеть на основе BERT, дообучить ее на подготовленном наборе данных с русскими и английскими предложениями и затем оценить качество работы обученной сети с помощью метрики BLEU.

Если не получится добиться минимального значения качества перевода (50 по метрике BLEU), то возможна еще одна итерация с выбором модели нового типа, ее последующим обучением и оценкой качества работы. Также возможен возврат на предыдущий этап к работе с набором данных, т.к. во многом качество обучения модели зависит от качества набора данных, который используется для обучения.

**Развертывание модели**

Четвертый этап – это развертывание модели для практического применения. Именно на этом этапе создается приложение, которое использует модель машинного обучения, созданную на предыдущем этапе, и организуется доступ пользователей к приложению.

Вопреки распространенному мнению, на этапе развертывания разработка системы машинного обучения не заканчивается. Часто бывает, что данные от пользователей существенно отличаются от тех, которые мы применяли для обучения модели. Например, пользователи могут попытаться автоматически переводить сообщения из соцсетей, в то время как наша система обучалась на литературных текстах или статьях с Wikipedia. В результате у системы могут возникнуть проблемы при переводе слэнга, предложений с плохой грамматикой и эмоджи. Такое явление называется сдвиг данных (data shift).

Проблема сдвига данных часто встречается на практике. Компания Яндекс сейчас проводит соревнования Shifts Challenge (https://research.yandex.com/shifts), в рамках которых открыла доступ к трем большим набором данных: обучения систем управления автомобилей без водителя, автоматического перевода и прогноза погоды. Цель этих соревнований: разработать методы и технологии обеспечения качественной работы моделей даже в условиях сдвига данных.

После развертывания и запуска в продуктивное использование важно организовать мониторинг качества работы модели с данными от пользователей. Если качество работы начинает снижаться, то необходимо переобучение модели на новых данных. Для этого запускается еще одна итерация жизненного цикла: данные от пользователей добавляются в подготовленный ранее набор данных (этап подготовки данных), затем модель обучается на новых данных (этап обучения модели), и новая модель развертывается. Так как данные от пользователей постоянно меняются, то и в жизненном цикле большинства приложений машинного обучения этапы подготовки набора данных, обучения модели, развертывание модели для продуктивного использования будут повторятся несколько раз.

Иногда бывает, что после запуска модели в продуктивное использование приходится не просто переобучать модель на новых данных, а полностью менять модель на более подходящую к данным пользователей. В этом случае с этапа развертывания происходит возврат к этапу обучения модели.

**Итоги**

* Жизненный цикл разработки приложений машинного обучения включает этапы постановки задачи, подготовки данных, обучение модели и развертывание модели для практического использования.
* Жизненный цикл разработки приложений машинного обучения – итеративный процесс. С каждого этапа возможен возврат на предыдущие.
* Качество работы развернутой модели со временем падает из-за сдвига данных. Поэтому цикл подготовка данных–обучение модели–развертывание модели необходимо повторять через определенные промежутки времени (от нескольких недель до нескольких месяцев).

Тест

Какие шаги включает жизненный цикл разработки приложений машинного обучения?

1. Проектирование, разработка, тестирование, развертывание.
2. **Постановка задачи, подготовка данных, обучение модели, развертывание модели.**
3. Сбор данных, очистка данных, обучение модели, оценка качества модели.
4. Проектирование, разработка, сопровождение, вывод из эксплуатации.

Что такое сдвиг данных

1. Запись данных в неправильные столбцы таблицы.
2. Переиспользование данных при обучении и тестировании модели.
3. **Отличие данных, которые поступают от пользователей от данных, на которых обучалась модель.**
4. Подготовка данных для модели машинного обучения нового типа.

Для какой цели используется мониторинг качества работы модели на данных пользователей.

1. Чтобы обнаружить выбросы в данных и заблокировать их.
2. Чтобы подобрать алгоритмы правильной подготовки данных для модели.
3. Чтобы обнаружить повышение качества работы модели в связи с поступлением новых данных от пользователей.
4. **Чтобы обнаружить снижение качества работы модели из-за сдвига данных.**

# Модуль № 1. Юнит № 4. Определение тональности текстов

Давайте рассмотрим, как можно разрабатывать приложения искусственного интеллекта на примере системы определения тональности отзывов на тексты.

Полный жизненный цикл создания такой системы будет включать следующие этапы:

1. **Постановка задачи.** Предположим, что наша цель: научиться определять тональность отзывов в социальных сетях на продукты, которые производит наша компания. Эту информацию мы планируем использовать дальше для проектирования продуктов компании. В качестве метрики качества работы модели будем использовать долю правильных ответов, желаемая величина: минимум 95%
2. **Подготовка набора данных.** На этом этапе нам необходимо собрать в социальных сетях тексты отзывов на продукты нашей компании, или похожие по смыслу и эмоциональной окраске тексты. Затем нужно выполнить разметку: для каждого отзыва указать, положительный он, или отрицательный.
3. **Обучение модели**. Лучшие результаты анализа текстов сейчас дают нейронные сети. Поэтому нам необходимо обучить нейронную сеть с помощью набора данных, который мы подготовили на предыдущем этапе.
4. **Развертывание модели.** На этом этапе создается приложение, использующее полученную модель для определения эмоциональной окраски текстов, и разворачивается, например, в облаке.

**Использование готовых моделей машинного обучения**

Однако на практике не всегда необходимо создавать модели машинного обучения с нуля. Существует большое количество библиотек машинного обучения, содержащих готовые к использованию модели машинного обучения, обученные на популярных наборах данных. В этом случае можно пропустить этапы подготовки набора данных и обучения модели, и сразу перейти к разработке приложения, предназначенного для развертывания.

Для определения тональности текстов удобнее всего использовать библиотеку Hugging Face, т.к. она автоматизирует большое количество этапов обработки текста и позволяет создавать программы небольшого размера. Вот пример программы определения тональности текста с помощью библиотеки Hugging Face:

| **from** transformers **import** pipeline  classifier = pipeline("sentiment-analysis",  "blanchefort/rubert-base-cased-sentiment")  classifier("Я обожаю инженерию машинного обучения!") |
| --- |

В первой строке мы подключаем класс pipeline из модуля transformers библиотеки Hugging Face. Пайплайны в машинном обучении служат для автоматизации процесса обработки данных. Пайплайны в Hugging Face позволяют применять модели машинного обучения для обработки текстов с использованием нескольких строк кода.

Во второй строке мы создаем пайплайн, использую конструктор класса pipeline. При создании пайплайна в Hugging Face указывается:

* Тип пайплайна – название задачи машинного обучения, реализация которой автоматизируется. В нашем примере название задачи: sentiment-analysis, анализ эмоциональной окраски текстов.
* Модель машинного обучения, которая будет использоваться в пайплайне для решения нужной нам задачи. Название модели: blanchefort/rubert-base-cased-sentiment, описание модели можно посмотреть на сайте библиотеки Hugging Face по ссылке – <https://huggingface.co/blanchefort/rubert-base-cased-sentiment>. Модель использует одну из наиболее популярных в настоящее время архитектур нейросетей для обработки текстов BERT. Модель дообучена определять тональность текстов на русском языке на составном объеме данных, включающем 351.797 текстов.

Созданный пайплайн записывается в переменную classifier, которую можно использовать для определения тональности текста.

В третьей строке мы используем пайплайн в переменной classifier, чтобы определить тональность текста "Я обожаю инженерию машинного обучения!". После запуска получаем следующий результат:

| [{'label': 'POSITIVE', 'score': 0.9807393550872803}] |
| --- |

Это список, который содержит столько же элементов, сколько текстов было передано в пайплайн (в нашем случае один элемент). Каждый элемент списка на выходе из пайплайна содержит словарь. Элементы в словаре зависят от того, для какой задачи применяется пайплайн. В задаче определения тональности словарь на выходе из пайплайна содержит два элемента:

* 'label' – метка результата распознавания тональности. Чаще всего используются значения 'POSITIVE' – позитивная тональность и 'NEGATIVE' – негативная тональность. Иногда могут встречаться другие метки, например, 'NEUTRAL' – нейтральная тональность.
* 'score' – значение на выходе из модели. Для задачи определения тональности на выходе из модели выдается число в диапазоне от 0 до 1. Значение 0 означает максимальную уверенность, что тональность отрицательная, значение 1 – максимальную уверенность, что тональность положительная.

В нашем примере элемента 'label' содержит значение 'POSITIVE' – эмоциональная окраска текста положительная. Значение элемента 'score' 0.9807393550872803 близко к 1, значит, модель имеет высокую уверенность в положительной эмоциональной окраске.

Таким образом, с помощью трех строк кода нам удалось решить достаточно сложную задачу машинного обучения: определения эмоциональной окраски текстов на русском языке. Это стало возможным благодаря наличию готовых библиотек машинного обучения.

Библиотека Hugging Face является примером системного программного продукта, в котором интегрируется несколько программных продуктов: фреймворки машинного обучения TensorFlow и PyTorch, библиотеки обработки данных на Python (в первую очередь текстов), предварительно обученные модели машинного обучения и т.п. Создание системного программного продукта является трудозатратным процессом, но после создания на его основе можно быстро разрабатывать приложения. Что мы и сделали на примере приложения для определения эмоциональной окраски текста.

**Библиотеки машинного обучения с предварительно обученными моделями**

Hugging Face не единственная библиотека машинного обучения, которая содержит готовые к использованию модели. Вот еще несколько библиотек:

* TensorFlow Hub (<https://www.tensorflow.org/hub>) – Репозиторий с обученными моделями для библиотеки TensorFlow.
* PyTorch Hub (<https://pytorch.org/hub/>) – Репозиторий с обученными моделями для библиотеки PyTorch.
* Keras Applications (<https://keras.io/api/applications/>) – Предварительно обученные модели в библиотеке Keras. Преимущественно модели компьютерного зрения.

**Итоги**

* Использование предварительно обученных моделей из готовых библиотек машинного обучения позволяет существенно ускорить разработку приложений искусственного интеллекта.
* Пайплайны в машинном обучении применяются для автоматизации комплексных процессов обработки данных, состоящих из нескольких этапов.
* Библиотека Hugging Face – пример системного программного продукта, интегрирующего различные программные продукты: библиотеки машинного обучения, обработки данных, обученные модели машинного обучения и т.п.
* Рекомендуется изучить и использовать в реализации проекта одну из следующих библиотек: Hugging Face, TensorFlow Hub, PyTorch Hub, Keras Applications.

**Практическое задание**

Поэкспериментируйте с определением тональности текстов на русском языке с помощью библиотеки Hugging Face с использованием Colab-ноутбука – <https://colab.research.google.com/drive/136RKDIVzzLkMsiO8H6VRMYO1h0ZsdIgv?usp=sharing>

# Модуль № 1. Юнит № 5. Выбор модели машинного обучения для решения задачи

Видео – <https://www.dropbox.com/s/rbp4arlkuavn859/hf_models.mp4?dl=0>

**Практическое задание**

1. Поэкспериментируйте с ноутбуками из видео:

* Автоматический перевод – <https://colab.research.google.com/drive/1MvqqKTOFqMepC1VVEiCxOuhggHQx2a4q?usp=sharing>
* Генерация аннотаций новостей – <https://colab.research.google.com/drive/1P1tAK15CIYFd7WFvGN6ImBD-sQZXUuRi?usp=sharing>

1. Найдите на сайте Hugging Face модель для решения задачи по вашему выбору и создайте приложение, использующее эту модель.  
   Используйте документацию на Pipeline в библиотеке Hugging Face – <https://huggingface.co/transformers/main_classes/pipelines.html>

# Модуль № 1. Юнит № 6. Командная разработка

Крупные программные системы, использующие искусственный интеллект, практически невозможно создать одному человеку. Поэтому важно организовать процесс командной разработки, при котором над кодом программной системы работают несколько человек. В этом разделе вы узнаете, как это делается.

**Системы контроля версий**

Основной инструмент, который используется для организации командной разработки – это система контроля версий. Она нужна для того, чтобы можно было хранить разные версии файлов и удобно работать с ними. Чаще всего системы контроля версий используются для хранения программного кода, но могут применяться для файлов любых типов, в том числе наборов данных для обучения моделей.

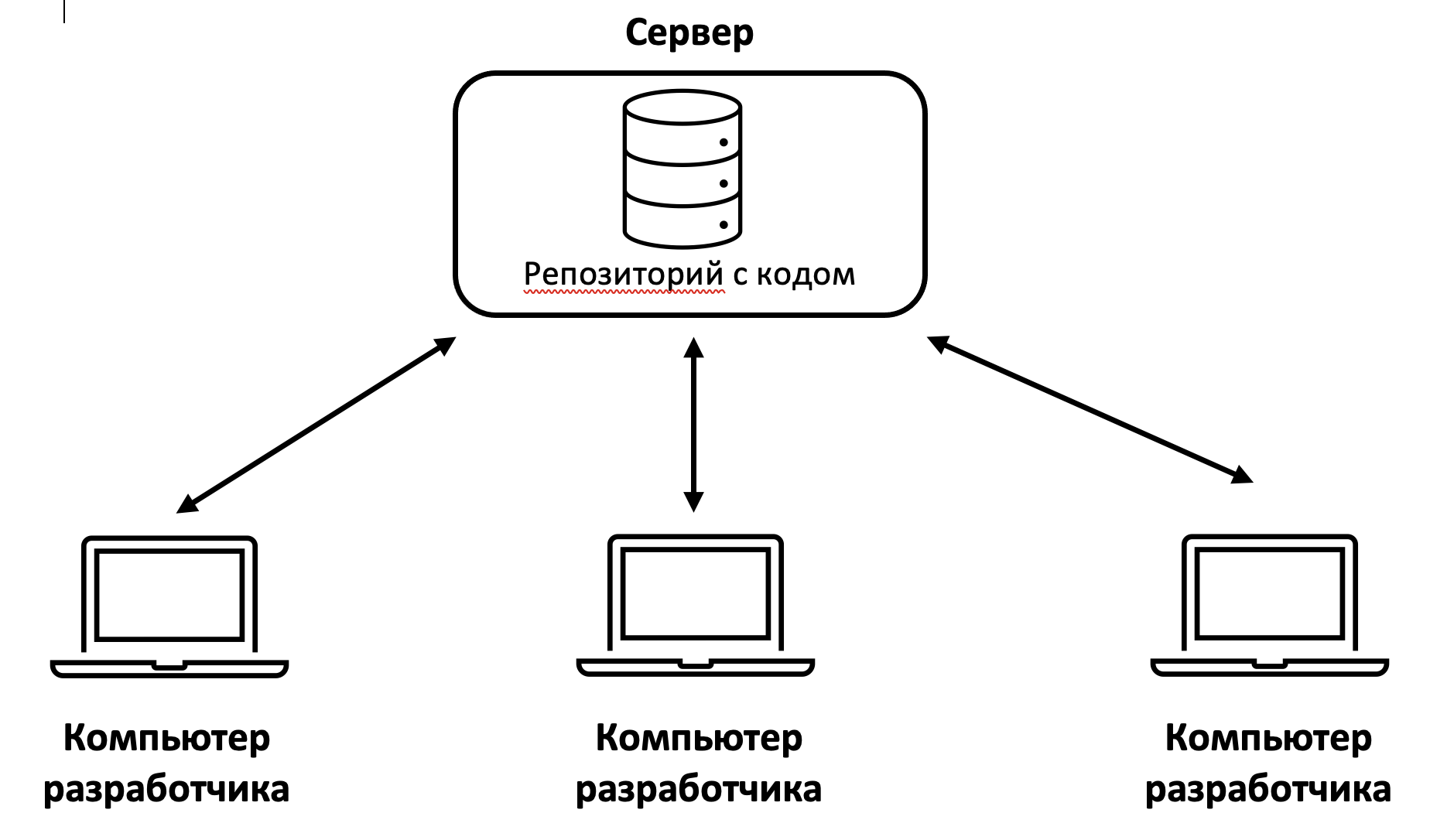
Чем полезны системы контроля версий в командной разработке? Представьте, что один из разработчиков в команде внес изменения в код, но в этих изменениях есть ошибка. В результате программа перестала работать. Решить ситуацию можно двумя способами:

* Найти ошибку в коде и исправить ее. Однако этот процесс может потребовать длительного времени, если ошибка не тривиальная. Особенно сложно искать ошибки в коде, который написал другой человек.
* Использовать одну из предыдущих версий кода, в которой нет ошибки. Такой подход называется откатом изменений и его можно реализовать достаточно быстро. Все, что нужно – это найти версию без ошибки и использовать ее в качестве действующей.

Самый простой вариант системы контроля версий – это каталог с файлами, причем к имени файла дописывается номер версии. Такую систему просто организовать, но с ней неудобно работать. Во-первых, когда файлов в проекте становится много, и у каждого файла несколько версий, может возникнуть путкница. Во-вторых, сложно найти, в какой именно версии файла были произведены интересующие изменения. Например, если мы обнаружили критическую ошибку в коде и хотим сделать откат к предыдущей версии без такой ошибки, то как нам узнать, в каком именно файле такой ошибки нет? Для этого придется последовательно просматривать все версии файла, что долго и не удобно. В третьих, в такой системе сложно организовать командную работу. Конечно, можно сделать так, чтобы каталог был доступен всем разработчикам по сети, но это не очень удобно. Поэтому для командной разработки были созданы специализированные системы контроля версий.

**Централизованные системы контроля версий**

Первые системы контроля версий, такие как CVS (Concurrent Versions System) и SVN (Subversion), использовали централизованный подход. В этом случае код программной системы храниться в одном месте – в репозитории на сервере. Разработчики подключаются со своих компьютеров к репозиторию и копируют часть данных себе. Такая копия данных из репозитория называется рабочей копией.



Централизованная система контроля версий

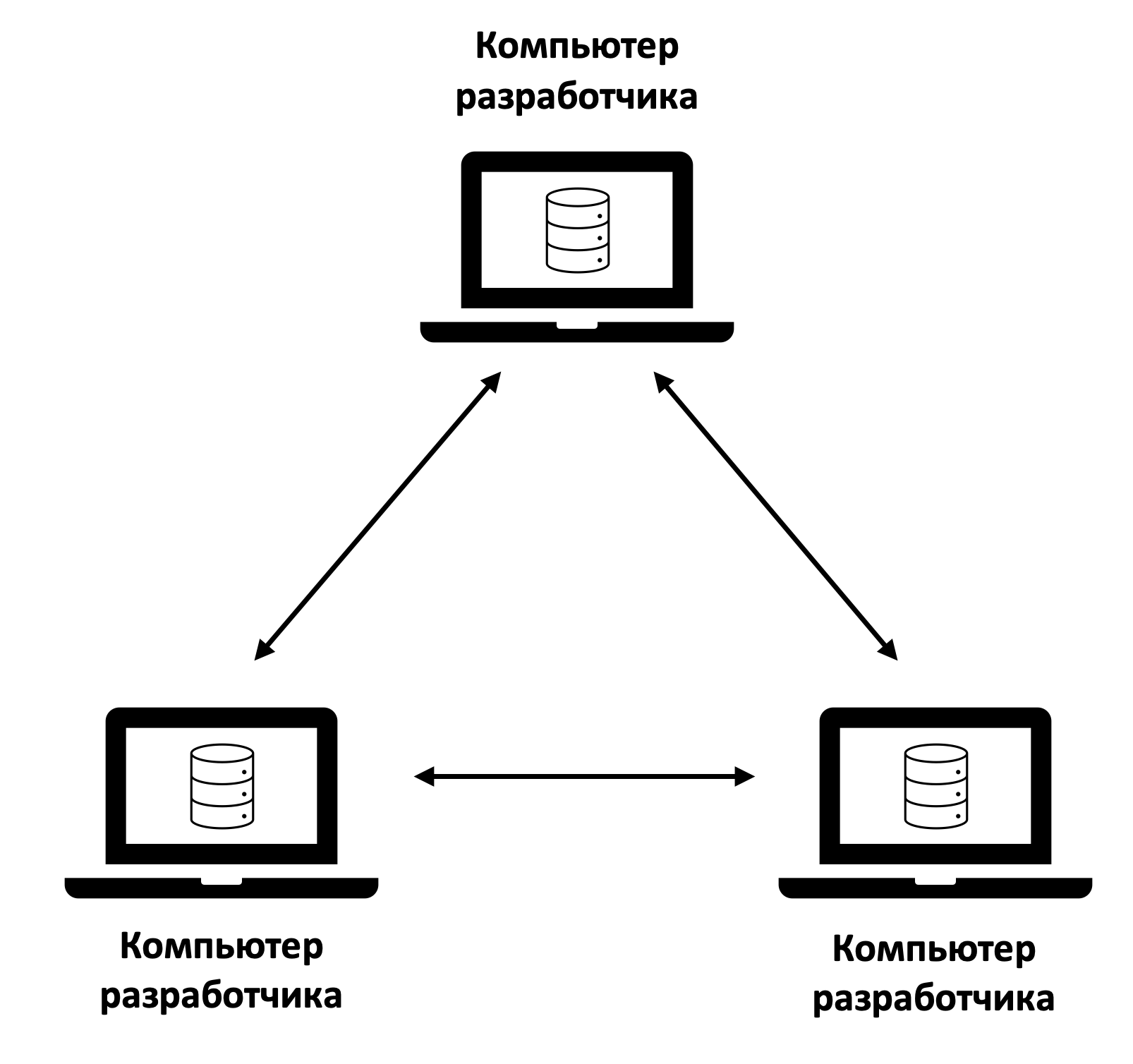
Если в команде несколько разработчиков, то каждый создают рабочую копию на своем компьютере. Изменения в код вносятся в локальные рабочии копии компьютеров разработчиков. Поэтому несколько человек могут работать с кодом независимо друг от друга.

После того, как разработчик реализовал законченную часть изменений в коде и проверил ее работоспособность, изменения необходимо перенести в централизованный репозиторий. Для этого используется функция фиксации изменений (по английски commit) системы контроля версий).

Централизованные системы контроля версий активно использовались начиная с 1990-х годов, но в 2016 году их популярность начала снижаться. Недостатком централизованных систем контроля версий является наличие единой точки отказа – сервера. В случае, если сервер перестает работать, то клиенты не смогут сохранять изменения и обмениваться этими изменениями с другими разработчиками из команды. А если сервер полностью выйдет из строя, то можно потерять весь репозиторий с кодом. Восстановить данные можно только из резервной копии (если, конечно, она у вас есть). Поэтому централизованные системы контроля версий постепенно были вытеснены распределенными системами контроля версий.

**Распределенные системы контроля версий**

В распределенных системах контроля версий полная копия репозитория с кодом хранится на каждом компьютере. Поэтому единой точки отказа нет. Если у разработчика пропадет связь с другими репозиториями, то он сможет продолжить работу со своей копией кода. Когда связь восстановится, изменения будут синхронизированы с другими участниками.

****

Распределенная система контроля версий

Потеря любого компьютера или сервера с репозиторием кода не страшна, т.к. существует несколько других полных копий репозитория.

Распределенная работа делает устройство системы контроля версий сложнее, чем централизованной. Изменения, внесенные на разных компьютерах, могут приводить к конфликтам. Например, два разработчика могут внести изменения в один и тот же файл. Разработчики делают это независимо друг от друга в копии репозитория на своих компьютерах. Но когда распределенная система контроля версий будет объединять изменения, она может не понять, как корректно обработать две разные версии одного файла от двух разработчиков. Иногда в процессе синхронизации изменений может потребоваться вмешательство человека.

В настоящее время самой популярной системой распределенного контроля версий является git (<https://git-scm.com/>). Именно ее мы и будем изучать в этом курсе, а также использовать в проектах в магистратуре.

**Итоги**

* Крупные программные системы, использующих искусственный интеллект, невозможно создать в одиночку. Необходима командная работа.
* Основной инструмент командной разработки программного обеспечения – системы контроля версий.
* Существует два типа систем контроля версий: централизованные и распределенные. Сейчас преимущественно используются распределенные системы контроля версий.
* Наиболее популярная в настоящее время распределенная система контроля версий git.

**Задания**

**Тест**

Какой тип систем контроля версий сейчас используется чаще всего

1. Локальные
2. **Распределенные**
3. Централизованные
4. Системные

**Задание на перетаскивание карточек**

Выберите характеристики, соответствующие типу системы контроля версий.

| Тип системы контроля версий | Характеристика |
| --- | --- |
| Централизованная | * Полный репозиторий с кодом хранится на сервере. * Сервер – единая точка отказа. * Работа без соединения с сервером невозможна. |
| Распределенная | * Полный репозиторий с кодом хранится на каждом компьютере. * Нет единой точки отказа. * Возможна работа без соединения с сервером и другими компьютерами. * Используются сложные алгоритмы синхронизации изменений между копиями репозитория. |

# Модуль № 1. Юнит № 7. Основы Git

В этом модуле вы познакомитесь с основами использования распределенной системы контроля версий git. Вы изучите команды git, научитесь создавать репозиторий git с кодом программы, отслеживать изменения в нем и синхронизировать изменения с удаленными репозиториями, чтобы ваш код стал доступен другим участникам вашей команды.

**Репозиторий git**

Репозиторий git – это хранилище кода, изменения в котором отслеживаются и сохраняются в виде версий. Технически репозиторий представляет собой каталог на компьютере. В этом каталоге хранится код программной системы, а также информация об изменениях в коде (версиях).

Работа с git начинается с создания репозитория. **Первый шаг** – это создание каталога в операционной системе, в котором вы хотите создать репозиторий. Например, это может быть каталог /home/user/ml\_project. **На втором шаге** вам нужно перейти в интересующий каталог и инициализировать репозиторий git с помощью команды:

| git init |
| --- |

При инициализации git создаст в каталоге подкаталог .git, в котором будет храниться служебная информация. (В Linux точка в начале имени файла говорит о том, что каталог является скрытым. Информация о нем не будет выдаваться в команде ls. Если вы хотите увидеть скрытые каталоги, используйте команду ls -la).

**Добавление файлов в репозиторий git**

Репозиторий создан, можно переходить к разработке кода программной системы. В качестве примера давайте создадим файл с кодом программы, определяющей тональность текста с помощью библиотеки Hugging Face. Для этого в каталоге репозитория создайте файл sentiment.py и запишите в него код, который мы рассматривали в разделе, посвященном определении тональности текстов:

| **from** transformers **import** pipeline  classifier = pipeline("sentiment-analysis",   "blanchefort/rubert-base-cased-sentiment")  classifier("Я обожаю инженерию машинного обучения!") |
| --- |

**Отслеживание изменений в git**

Git отслеживает любые изменения, выполненные в репозитории. Добавление файла sentiment.py тоже относится к таким изменениям. Получить информацию об изменениях в репозитории можно с помощью команды git status:

| ml\_project % git status On branch main  No commits yet  Untracked files:  (use "git add <file>..." to include **in** what will be committed)  sentiment.py  nothing added to commit but untracked files present (use "git add" to track) |
| --- |

В выводе команды git status достаточно много информации. Во-первых, git сообщает нам, что работа выполняется в ветке main (“On branch main”). Ветки – это механизм git для работы с большими изменениями в коде, которые могут привести к потенциальным конфликтам при синхронизации между всеми репозиториями. Механизм устроен достаточно сложно, поэтому его изучение мы отложим на второй семестр нашей магистерской программы. В первом семестре мы будем использовать только одну ветку с названием по умолчанию “main” (основная ветка). Поэтому в первом семестре ваша задача – отслеживать, что вы всегда работаете именно в ветке main.

Второе сообщение, “No commits yet”, говорит о том, что зафиксированных изменений (commit) в репозитории нет. Разработчикам редко удается написать работающий код с первой попытки. Обычно код нужно протестировать и по результатам тестирования внести изменения. Записывать в репозиторий имеет смысл только окончательную работоспособную версию, а не промежуточные результаты. Именно для этого служит механизм фиксации изменений: сохранении версии программы в репозитории выполняется после того, как закончено ее изменение. Для фиксации изменений необходимо ввести специальную команду git commit.

Третье сообщение говорит о том, что в репозитории существуют неотслеживаемые git файлы:

| Untracked files:  (use "git add <file>..." to include **in** what will be committed)  sentiment.py |
| --- |

Мы добавили в репозиторий файл с кодом sentiment.py, но git пока не отслеживает изменения в этом файле. Часто разработчик вносит изменения в несколько файлов, но зафиксировать изменения нужно только в некоторых из них. git выдает нам подсказку, что нужно сделать, чтобы подготовить файл для фиксации изменений (включить файл в commit): использовать команду git add.

Последняя строка говорит о том, что файлы для фиксации изменений не отмечены, но есть файлы, в которых произошли изменения.

| nothing added to commit but untracked files present (use "git add" to track) |
| --- |

**Фиксация изменений в git**

Давайте зафиксируем изменения в файле sentiment.py, который мы создали на предыдущем этапе. Для этого нам необходимо выполнить два действия:

1. Указываем, что изменения в файле sentiment.py будут зафиксированны, с помощью команды git add:

| git add sentiment.py |
| --- |

Просматриваем статус репозитория после выполнения команды:

| ml\_project % git status On branch main  No commits yet  Changes to be committed:  (use "git rm --cached <file>..." to unstage)  new file: sentiment.py |
| --- |

В выводе команды git status можно увидеть, что для фиксации изменений (“Changes to be committed:”) подготовлен новый файл sentiment.py (new file: sentiment.py). Теперь можно переходить к фиксации изменений.

1. Фиксируем изменения с помощью команды git commit:

| ml\_project % git commit -m "Added sentiment.py" [master (root-commit) 9cfc2c4] Added sentiment.py  1 file changed, 6 insertions(+)  create mode 100644 sentiment.py |
| --- |

С помощью параметра -m необходимо указать строку с описанием внесенных изменений. В нашем случае строка "Added sentiment.py" (добавлен файл sentiment.py).

Фиксация изменений завершена, давайте снова проверим статус репозитория:

| ml\_project % git status  On branch main nothing to commit, working tree clean |
| --- |

Изменений для фиксации нет.

**Передача зафиксированных изменений в удаленный репозиторий**

С помощью команды git commit мы зафиксировали изменения в репозитории на локальном компьютере. Однако этого недостаточно для командной разработки. Чтобы ваши изменения стали доступны другим разработчикам в команде, их нужно передать на удаленный репозиторий (или несколько репозиториев). Для этого используется команда git push:

| ml\_project % git push origin main |
| --- |

В команде git push необходимо указать, какую ветку мы хотим передать. Так как мы пока работаем только с одной веткой, то указываем ее название – main. Также нужно указать, на какой компьютер нужно передать изменения. Это делается с помощью обозначения компьютера – origin.

**Итоги**

* Для создания репозитория git используется команда git init
* Изменения в репозитории git выполняются в следующей последовательности:

1. Редактирование файлов в репозитории или создание новых файлов.
2. Отметка файлов для включения в фиксацию изменений с помощью команды git add
3. Фиксация изменений с помощью команды git commit.
4. Отправка изменений на удаленный репозиторий с помощью команды git push.

* Просмотр состояния репозитория git выполняется с помощью команды git status.

**Карточки**

Выберите действие, которое выполняет команда git

| Команда git | Действие команды |
| --- | --- |
| git init | Создание репозитория git |
| git status | Просмотр статуса репозитория git |
| git commit | Фиксация изменений в репозитории |
| git add | Выбор файлов для фиксации изменений |
| git push | Отправка изменений на удаленный репозиторий для синхронизации |

# Модуль № 1. Юнит № 8. Сервис GitHub

Систему контроля версий git для командной разработки можно установить и настроить самостоятельно, а можно воспользоваться одним из существующих сервисов для создания репозиториев git. Сейчас популярны следующие сервисы:

* GitHub – <https://github.com/>
* BitBucket – <https://bitbucket.org/>
* GitLab – <https://gitlab.com/>

Мы будем использовать сервис GitHub, т.к. он является лидером в своей области. На GitHub размещается исходный код многих популярных библиотек машинного обучения: TebsorFlow (<https://github.com/tensorflow/tensorflow>), PyTorch (https://github.com/pytorch/pytorch), scikit-learn (https://github.com/scikit-learn/scikit-learn), Hugging Face (<https://github.com/huggingface>) и др.

Видео: Создание репозитория на GitHub – <https://www.dropbox.com/s/m50b22itd5ndbl5/github.mp4?dl=0>

Видео: Настройка ключа SSH на GitHub – <https://www.dropbox.com/s/f5pj7i5i9gjz873/ssh_github.mp4?dl=0>

**Итоги**

* GitHub – самый популярный сервис для создания и хранения репозиториев git.
* GitHub использует SSH ключи для проверки прав доступа пользователей при обновлении репозитория.
* Команды git для работы с удаленным репозиторием:
  + git clone – клонирование удаленного репозитория на локальный компьютер.
  + git push – отправка изменений с локального компьютера на удаленный репозиторий.
  + git pull – получение изменений с удаленного репозитория на локальный компьютер.

**Тест**

Какая команда git используется для клонирования удаленного репозитория на локальный компьютер?

1. git checkout
2. git copy
3. **git clone**
4. git download

Для чего используется ключ SSH на GitHub?

1. Для проверки прав доступа пользователя к Web порталу GitHub.
2. **Для проверки прав доступа пользователя при обновлении репозитория GitHub с локального компьютера командой git push.**
3. GitHub не использует ключи SSH.
4. Для организации доступа без ввода пароля к виртуальной машине Linux в GitHub.

Какая команда git используется для получения изменений с удаленного репозитория на локальный компьютер?

1. **git pull**
2. git push
3. git clone
4. git sync

# Практическое задание

Цель задания: научиться в команде создавать приложения, использующие готовые библиотеки машинного обучения.

Задание:

1. Сформируйте команду из 2-3 человек.
2. Создайте репозиторий на GitHub, который будет использоваться для командной разработки приложения.
3. Изучите возможности готовых библиотек машинного обучения.
4. Сформулируйте задачу, которую вы хотите решить с помощью машинного обучения.
5. Реализуйте решение выбранной вами задачи в коде с использованием готовой библиотеки машинного обучения.
6. Отправьте реализованное решение в репозиторий на GitHub.
7. Оформите документацию на ваше решение в репозитории.

Рекомендуется использовать одну из следующих библиотек машинного обучения: Hugging Face (https://huggingface.co/), TensorFlow Hub (<https://www.tensorflow.org/hub>), PyTorch Hub (<https://pytorch.org/hub/>), Keras Applications (<https://keras.io/api/applications/>).

Пример репозитория – <https://github.com/sozykin/ml_sentiment>

# Термины

Все потенциально-незнакомые и новые понятия выдели отдельно и “положи” сюда.

# Список источников

* Фредерик Брукс: Мифический человеко-месяц, или Как создаются программные системы. 1975.
* Giannakis, M., Dubey, R., Yan, S. et al. Social media and sensemaking patterns in new product development: demystifying the customer sentiment. Ann Oper Res (2020). <https://doi.org/10.1007/s10479-020-03775-6>
* Software Engineering Body of Knowledge (SWEBOK) (рекомендации IEEE к области знаний «Программная инженерия») –

https://www.computer.org/education/bodies-of-knowledge/software-engineering

* Andrew Ng. Специализация Machine Learning Engineering for Production (MLOps) – https://www.coursera.org/specializations/machine-learning-engineering-for-production-mlops
* Hugging Face Course – https://huggingface.co/course/
* Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding – <https://arxiv.org/abs/1810.04805>
* Hugging Face Models – <https://huggingface.co/models>
* Система контроля версий git – <https://git-scm.com/>
* Scott Chacon, Ben Straub. Pro Git. – <https://git-scm.com/book/ru/v2>

# Дополнительные материалы

* Hugging Face Course – https://huggingface.co/course/
* Ноутбук в Colab с определением тональности текстов на русском языке с помощью библиотеки Hugging Face – <https://colab.research.google.com/drive/136RKDIVzzLkMsiO8H6VRMYO1h0ZsdIgv?usp=sharing>
* Ноутбук в Colab с автоматическим переводом с русского на английский язык с помощью библиотеки Hugging Face – <https://colab.research.google.com/drive/1MvqqKTOFqMepC1VVEiCxOuhggHQx2a4q?usp=sharing>
* Ноутбук в Colab с генерацией аннотаций новостей с помощью библиотеки Hugging Face – <https://colab.research.google.com/drive/1P1tAK15CIYFd7WFvGN6ImBD-sQZXUuRi?usp=sharing>
* BERT — state-of-the-art языковая модель для 104 языков. Туториал по запуску BERT локально и на Google Colab – <https://habr.com/ru/post/436878/>
* Документация на Pipeline в библиотеке Hugging Face – <https://huggingface.co/transformers/main_classes/pipelines.html>
* Готовые модели машинного обучения в библиотеке Hugging Face – <https://huggingface.co/models>)
* Готовые модели машинного обучения в библиотеке TensorFlow – (<https://www.tensorflow.org/hub>),
* Готовые модели машинного обучения в библиотеке PyTorch – <https://pytorch.org/hub/>
* Готовые модели машинного обучения в библиотеке Keras – <https://keras.io/api/applications/>

# 