АО «АЛМАТИНСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет инженерных и информационных технологий

Кафедра компьютерной инженерии

ПРОЕКТИРОВАНИЕ СИСТЕМ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

ЛЕКЦИЯ № 13

MLOps и управление жизненным циклом моделей

Теоретико-практический конспект

Составитель: Алтыбай А.

Кафедра компьютерной инженерии

Алматы, 2026

Аннотация и назначение лекции

Лекция 13 раскрывает тему «MLOps и управление жизненным циклом моделей» в контексте разработки промышленной AI-системы. Материал опирается на тематический план дисциплины и соединяет определения, принципы проектирования, математические оценки, проектные артефакты, практический код и аргументированный анализ рисков. Вопросы построены вокруг решений, которые предстоит принимать инженеру, а не только вокруг названий алгоритмов. [2, 5]

План занятия

1. Задачи MLOps
2. Experiment tracking
3. Model registry
4. Data and code versioning
5. Training pipeline
6. CI/CD/CT
7. Data validation gates
8. Model acceptance gates
9. Практические сценарии, контроль качества и проектные задания.

Ожидаемые результаты обучения

По завершении студент способен:

1. объяснить и аргументированно применить принцип «Задачи MLOps» к инженерному решению;
2. объяснить и аргументированно применить принцип «Experiment tracking» к инженерному решению;

3. объяснить и аргументированно применить принцип «Model registry» к инженерному решению;
4. объяснить и аргументированно применить принцип «Data and code versioning» к инженерному решению;
5. объяснить и аргументированно применить принцип «Training pipeline» к инженерному решению;
6. объяснить и аргументированно применить принцип «CI/CD/CT» к инженерному решению;
7. реализовать упрощённый проверяемый прототип и сформулировать критерии его приемки.
8. указать угрозы достоверности результатов и предложить проверки до эксплуатации.

Сквозной кейс. В течение лекции рассматриваем платформу повторного обучения. Существенные условия: публикацию худшего кандидата, потерю lineage. Основная проверяемая характеристика: количество успешных воспроизводимых сборок. Числа в расчётных примерах учебные и не являются результатами реальных измерений.

1. Базовые теоретические положения из конспекта курса

Ниже представлены исходные положения учебно-методического конспекта, связанные с данной темой. Следующие разделы раскрывают их с инженерными деталями и проверяемыми примерами. [2, 5] Цель лекции. сформировать понимание MLOps как набора процессов и инструментов для воспроизводимых экспериментов, версионирования, регистрации, автоматизации обучения и безопасного продвижения моделей.

Ключевые понятия. MLOps, experiment tracking, model registry, pipeline, CI/CD/CT, reproducibility, artifact, lineage, approval workflow, champion/challenger.

1.1 Зачем нужен MLOps

ML-проект производит больше изменяемых объектов, чем обычный софт: код, данные, признаки, конфигурации, модели, метрики и окружение. MLOps связывает их в управляемый процесс, чтобы ответить: «какая версия данных и кода породила модель, работающую сейчас в production?»

Главная цель — не максимальная автоматизация, а надежная повторяемость, трассируемость и безопасная скорость изменений.

1.2 Experiment tracking

Experiment tracking сохраняет параметры запуска, метрики, артефакты и ссылки на код/данные. Это предотвращает потерю успешной конфигурации и позволяет сравнивать trials. Хороший эксперимент имеет уникальный ID и минимальный набор контекста, достаточный для повторения.

MLflow, Weights & Biases и другие системы помогают организовать журнал экспериментов, но дисциплина команды важнее инструмента.

1.3 Model registry

Model registry хранит версии моделей и их статус: candidate, staging, production, archived. Registry должен связывать артефакт с evaluation report и правилами допуска. В критических системах публикация может требовать ручного approval.

Champion/challenger подход сохраняет текущую production-модель как champion и сравнивает новую версию challenger до замены.

1.4 CI/CD/CT для ML

CI проверяет код, тесты и качество pipeline. CD автоматизирует доставку проверенного артефакта. CT (continuous training) автоматизирует переобучение по условию или расписанию. В ML automation должна включать проверки данных и метрик, а не только unit tests.

Автоматический retraining без gates опасен: новая модель может оказаться хуже из-за плохих данных или изменения разметки.

1.5 Тестирование ML-систем

Помимо unit/integration tests используются schema tests, data quality tests, feature tests, prediction invariants, regression tests на эталонном наборе, latency tests и smoke tests после deployment.

Например, можно проверять, что вероятность лежит в [0,1], список классов не изменился, p95 latency не превышает SLO и новая модель не хуже champion на критическом сегменте.

1.6 Выводы

- MLOps связывает код, данные, модель и инфраструктуру в трассируемый жизненный цикл.
- Experiment tracking и registry отвечают на разные задачи.
- CI/CD для ML должен включать проверки данных и метрик.
- Continuous training требует quality gates и rollback.

1.7 Контрольные вопросы

- Какие объекты нужно версионировать в ML-проекте?
- Чем model registry отличается от хранения файлов модели?
- Что означает champion/challenger?
- Какие проверки должны входить в CI для ML?
- Почему полностью автоматическое переобучение может быть рискованным?

2. Систематическое изложение темы и инженерные решения

2.1 Задачи MLOps

MLOps систематизирует путь от идеи до контролируемого изменения production-модели. Автоматизация полезна только при известном экспериментальном протоколе. [2]

Проверяемый пример. Свяжите задачу в трекере с экспериментом, артефактом и deployment-версией.

Вопрос для обсуждения. Как проверить выполнение такого требования: Свяжите задачу в трекере с экспериментом, артефактом и deployment-версией.

2.2 Experiment tracking

Каждый запуск хранит параметры, метрики, версии кода и артефакты. Сравнивать разные данные без указания версий нельзя. [3]

Проверяемый пример. Зарегистрируйте два запуска с разным max_depth и одинаковым test split.

Вопрос для обсуждения. Как проверить выполнение такого требования: Зарегистрируйте два запуска с разным max_depth и одинаковым test split.

2.3 Model registry

Registry представляет жизненный цикл версий и стадии утверждения; он не просто папка весов. Неизменяемые артефакты облегчают rollback. [4]

Проверяемый пример. Сохраните модель с run_id, signature и результатом acceptance tests.

Вопрос для обсуждения. Как проверить выполнение такого требования: Сохраните модель с run_id, signature и результатом acceptance tests.

2.4 Data and code versioning

Воспроизведение требует как данных, так и кода преобразования, зависимостей и конфигурации. Номер версии библиотеки не восстанавливает исчезнувший датасет. [5]

Проверяемый пример. Проверьте возможность повторного запуска через фиксированную дату и data hash.

Вопрос для обсуждения. Как проверить выполнение такого требования: Проверьте возможность повторного запуска через фиксированную дату и data hash.

2.5 Training pipeline

Этапы extract, validate, transform, train, evaluate и register должны формировать артефакты и четкие ошибки. Ненадежная очистка не должна скрыто пропускать дефекты. [6]

Проверяемый пример. Создайте DAG с контрольным gate на минимальную долю валидных записей.

Вопрос для обсуждения. Как проверить выполнение такого требования: Создайте DAG с контрольным gate на минимальную долю валидных записей.

2.6 CI/CD/CT

CI проверяет код и схемы, CD доставляет сервис, CT запускает повторное обучение по одобренному условию. Автоматическое переобучение не означает автоматический rollout. [1]

Проверяемый пример. Разделите workflow для unit tests, training и manual production approval.

Вопрос для обсуждения. Как проверить выполнение такого требования: Разделите workflow для unit tests, training и manual production approval.

2.7 Data validation gates

Каждый pipeline должен останавливать сборку при отсутствии критических признаков, некорректной метке или резком изменении распределения. [2]

Проверяемый пример. Сымитируйте ошибочную категорию и продемонстрируйте отказ стадии validate.

Вопрос для обсуждения. Как проверить выполнение такого требования: Сымитируйте ошибочную категорию и продемонстрируйте отказ стадии validate.

2.8 Model acceptance gates

Кандидат не публикуют, если качество хуже допустимого, метрики по подгруппам недопустимы или latency превышает порог. [3]

Проверяемый пример. Сравните кандидата с champion на фиксированном holdout.

Вопрос для обсуждения. Как проверить выполнение такого требования: Сравните кандидата с champion на фиксированном holdout.

2.9 Artifact lineage

Происхождение артефакта связывает версии dataset, code, preprocessing и обученных весов. В случае инцидента должно быть возможно найти источник изменения. [4]

Проверяемый пример. Проследите model_v7 до upstream dataset snapshot и commit.

Вопрос для обсуждения. Как проверить выполнение такого требования: Проследите model_v7 до upstream dataset snapshot и commit.

2.10 Рабочие окружения

Development, staging и production разделяют секреты, нагрузки и доступ. Равные версии зависимостей уменьшают неопределенность, но не исключают среды GPU. [5]

Проверяемый пример. Опишите тестовый стенд для staging с синтетическими персональными данными.

Вопрос для обсуждения. Как проверить выполнение такого требования: Опишите тестовый стенд для staging с синтетическими персональными данными.

2.11 Релизная стратегия

Shadow, canary и blue-green rollout уменьшают риск; выбор зависит от стоимости двойной инфраструктуры и характера ошибок. [6]

Проверяемый пример. Проверьте новую модель на зеркальных запросах без влияния на пользовательские решения.

Вопрос для обсуждения. Как проверить выполнение такого требования: Проверьте новую модель на зеркальных запросах без влияния на пользовательские решения.

2.12 Ответственность после релиза

Команда определяет owners, on-call, журнал изменений, политику rollback и сроки пересмотра модели. Закрыть задачу на deployment недостаточно.

[1]

Проверяемый пример. Разработайте checklist передачи модели эксплуатации.

Вопрос для обсуждения. Как проверить выполнение такого требования: Разработайте checklist передачи модели эксплуатации.

3. Математические инструменты и количественные критерии

Для перехода от неформального решения к проверяемой постановке введём следующие величины.

$$\mathrm{Run} = \big(\mathrm{data_hash}, \mathrm{code_commit}, \mathrm{config}, \mathrm{environment}, \mathrm{metrics}\big) \tag{1}$$

$$\mathrm{Gate} = \mathbf{1}\big[F_1 \geq F_{1,\min} \ \wedge \ p95 \leq T_{\max}\big] \tag{2}$$

Пороговый шлюз предотвращает публикацию кандидата, который проходит тест качества, но нарушает инженерное требование по задержке.

При выборе числового критерия нужно отличать обучающую функцию потерь, продуктовую ценность и эксплуатационные ограничения. Первую можно оптимизировать дифференцированием, вторая определяется процессом принятия решения, а третьи проверяются нагрузочными испытаниями, политиками безопасности и мониторингом.

3.1 Пошаговый расчёт на учебных данных

Рассмотрим две конфигурации одной системы. Вариант A имеет условную оценку качества $q_A = 0.86$ и задержку $t_A = 90$ мс; вариант B — $q_B = 0.89$ и $t_B = 280$ мс. Предположим, что требование заказчика задаёт $q \geq 0.85$ и $p95 \leq 150$ мс. Тогда A удовлетворяет обоим условиям, тогда как B нарушает второе. Следовательно, выбор B только потому, что его offline-метрика выше, не является обоснованным. Числа иллюстративны; реальный вывод требует измерений при одинаковой нагрузке и одинаковом наборе данных.

$$\mathrm{accept}(A) = \mathbf{1}\{0.86 \geq 0.85 \wedge 90 \leq 150\} = 1,$$
$$\mathrm{accept}(B) = \mathbf{1}\{0.89 \geq 0.85 \wedge 280 \leq 150\} = 0.$$

3.2 Как интерпретировать число, а не только вычислить его

Сначала следует проверить единицы измерения, состав выборки и смысл агрегирования. Например, p95 на спокойном стенде нельзя переносить на рабочую нагрузку с иным числом одновременных запросов. Затем исследуют чувствительность к изменениям входных данных и неопределённость оценок. Наконец, фиксируют последствия порога: какую ошибку он предотвращает, а какую может увеличить. Этот подход делает количественный критерий частью инженерной аргументации, а не украшением отчёта. [2, 5]

4. Практическое программирование на Python

Пример предназначен для демонстрации ключевого инженерного принципа. Он не претендует на обработку реального production-трафика. Запускать код следует в изолированном окружении и фиксировать версии библиотек, если они используются. Вывод зависит от указанных входных данных; для стохастических алгоритмов он может меняться.

4.1 Основной воспроизводимый пример

```
import mlflow
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

X, y = load_iris(return_X_y=True)
with mlflow.start_run():
    model = DecisionTreeClassifier(max_depth=3, random_state=17)
    model.fit(X[:120], y[:120])
    acc = accuracy_score(y[120:], model.predict(X[120:]))
    mlflow.log_param("max_depth", 3)
    mlflow.log_metric("holdout_accuracy", float(acc))
    mlflow.sklearn.log_model(model, name="iris_tree")
# Use a stratified split for a real evaluation; ordered split is
# intentionally used here only to show tracking commands.
```

Проверка результатов. Сначала убедитесь, что код завершился без исключений; проверьте формы массивов, схему результата и отсутствие недопустимых значений. Если используются внешние данные, включите в отчёт способ их получения и идентификатор версии. Измените один входной параметр и объясните, почему наблюдаемый результат изменился или остался прежним.

4.2 Компактная проверка контракта или метрики

```
import hashlib, json
```

```
config = {"max_depth": 3, "seed": 17, "split": "v1"}
config_hash = hashlib.sha256(json.dumps(config, sort_keys=True).encode()).hexdigest()
print("config hash:", config_hash[:12])
```

При самостоятельном повторении предложите минимум один негативный тест: некорректное значение, пустой вход, несовместимую версию схемы либо нарушение SLO. Переход от успешного демонстрационного запуска к надежной системе требует именно таких проверок.

5. Разбор трёх инженерных ситуаций

5.1 Ситуация 1: Задачи MLOps

Заказчик использует платформу повторного обучения. Новое требование заставляет рассмотреть принцип «Задачи MLOps». Основная неопределённость состоит в том, как одновременно обеспечить количество успешных воспроизводимых сборок, не допустить потерю lineage и сохранить возможность изменения модели.

Постановка. MLOps систематизирует путь от идеи до контролируемого изменения production-модели. Автоматизация полезна только при известном экспериментальном протоколе.

Действие инженера. Свяжите задачу в трекере с экспериментом, артефактом и deployment-версией.

Альтернативы. Сравните три подхода: (A) простое правило без ML, (B) модульное ML-решение с отдельной проверкой входных данных и (C) интегрированный сервис со сложной моделью. Для каждого укажите потребность в данных, ожидаемые инженерные риски, возможность тестирования и минимальную инфраструктуру. Таблица ниже задаёт форму отчёта, а не готовый рейтинг.

Вариант	Данные и зависимости	Проверка	Основной риск
Правила	Фиксированные условия	Набор тест-кейсов	Ограниченная гибкость
ML-модуль	Версионированные признаки	Offline и API tests	Drift и skew
Сложный сервис	Несколько компонентов	End-to-end и нагрузка	Каскадный отказ

Проверка решения. Подготовьте две позитивные и две негативные пробы; задайте метрику успеха, базовый сценарий и критерий остановки внедрения. Обсудите, какие допущения останутся непроверенными даже после удачного теста.

5.2 Ситуация 2: CI/CD/CT

Заказчик использует платформу повторного обучения. Новое требование заставляет рассмотреть принцип «CI/CD/CT». Основная неопределённость состоит в том, как одновременно обеспечить количество успешных воспроизводимых сборок, не допустить потерю lineage и сохранить возможность изменения модели.

Постановка. CI проверяет код и схемы, CD доставляет сервис, CT запускает повторное обучение по одобренному условию. Автоматическое переобучение не означает автоматический rollout.

Действие инженера. Разделите workflow для unit tests, training и manual production approval.

Альтернативы. Сравните три подхода: (A) простое правило без ML, (B) модульное ML-решение с отдельной проверкой входных данных и (C) интегрированный сервис со сложной моделью. Для каждого укажите потребность в данных, ожидаемые инженерные риски, возможность тестирования и минимальную инфраструктуру. Таблица ниже задаёт форму отчёта, а не готовый рейтинг.

Вариант	Данные и зависимости	Проверка	Основной риск
Правила	Фиксированные условия	Набор тест-кейсов	Ограниченная гибкость
ML-модуль	Версионированные признаки	Offline и API tests	Drift и skew
Сложный сервис	Несколько компонентов	End-to-end и нагрузка	Каскадный отказ

Проверка решения. Подготовьте две позитивные и две негативные пробы; задайте метрику успеха, базовый сценарий и критерий остановки внедрения. Обсудите, какие допущения останутся непроверенными даже после удачного теста.

5.3 Ситуация 3: Релизная стратегия

Заказчик использует платформу повторного обучения. Новое требование заставляет рассмотреть принцип «Релизная стратегия». Основная неопределённость состоит в том, как одновременно обеспечить количество успешных воспроизводимых сборок, не допустить потерю lineage и сохранить возможность изменения модели.

Постановка. Shadow, canary и blue-green rollout уменьшают риск; выбор зависит от стоимости двойной инфраструктуры и характера ошибок.

Действие инженера. Проверьте новую модель на зеркальных запросах без влияния на пользовательские решения.

Альтернативы. Сравните три подхода: (A) простое правило без ML, (B) модульное ML-решение с отдельной проверкой входных данных и (C) интегрированный сервис со сложной моделью. Для каждого укажите потребность в данных, ожидаемые инженерные риски, возможность тестирования и минимальную инфраструктуру. Таблица ниже задаёт форму отчёта, а не готовый рейтинг.

Вариант	Данные и зависимости	Проверка	Основной риск
Правила	Фиксированные условия	Набор тест-кейсов	Ограниченная гибкость
ML-модуль	Версионированные признаки	Offline и API tests	Drift и skew
Сложный сервис	Несколько компонентов	End-to-end и нагрузка	Каскадный отказ

Проверка решения. Подготовьте две позитивные и две негативные пробы; задайте метрику успеха, базовый сценарий и критерий остановки внедрения. Обсудите, какие допущения останутся непроверенными даже после удачного теста.

6. Самостоятельная работа: индивидуальные проектные задания

Для каждого задания студент создаёт отдельный небольшой артефакт: схему, набор тестов, код либо таблицу сравнения, прикладывает наблюдаемый результат и объясняет ограничения.

1. **Задачи MLOps.** Свяжите задачу в трекере с экспериментом, артефактом и deployment-версией. Сформулируйте проверяемое требование, продемонстрируйте альтернативу и приложите протокол хотя бы одного эксперимента.

2. **Experiment tracking.** Зарегистрируйте два запуска с разным max_depth и одинаковым test split. Сформулируйте проверяемое требование, продемонстрируйте альтернативу и приложите протокол хотя бы одного эксперимента.

3. **Model registry.** Сохраните модель с run_id, signature и результатом acceptance tests. Сформулируйте проверяемое требование, продемонстрируйте альтернативу и приложите протокол хотя бы одного эксперимента.

4. **Data and code versioning.** Проверьте возможность повторного запуска через фиксированную дату и data hash. Сформулируйте проверяемое требование, продемонстрируйте альтернативу и приложите протокол хотя бы одного эксперимента.

5. **Training pipeline.** Создайте DAG с контрольным gate на минимальную долю валидных записей. Сформулируйте проверяемое требование, продемонстрируйте альтернативу и приложите протокол хотя бы одного эксперимента.

6. **CI/CD/CT.** Разделите workflow для unit tests, training и manual production approval. Сформулируйте проверяемое требование, продемонстрируйте альтернативу и приложите протокол хотя бы одного эксперимента.

7. **Data validation gates.** Сымитируйте ошибочную категорию и продемонстрируйте отказ стадии validate. Сформулируйте проверяемое тре-

бование, продемонстрируйте альтернативу и приложите протокол хотя бы одного эксперимента.

8. **Model acceptance gates.** Сравните кандидата с champion на фиксированном holdout. Сформулируйте проверяемое требование, продемонстрируйте альтернативу и приложите протокол хотя бы одного эксперимента.

9. **Artifact lineage.** Проследите model_v7 до upstream dataset snapshot и commit. Сформулируйте проверяемое требование, продемонстрируйте альтернативу и приложите протокол хотя бы одного эксперимента.

10. **Рабочие окружения.** Опишите тестовый стенд для staging с синтетическими персональными данными. Сформулируйте проверяемое требование, продемонстрируйте альтернативу и приложите протокол хотя бы одного эксперимента.

11. **Релизная стратегия.** Проверьте новую модель на зеркальных запросах без влияния на пользовательские решения. Сформулируйте проверяемое требование, продемонстрируйте альтернативу и приложите протокол хотя бы одного эксперимента.

12. **Ответственность после релиза.** Разработайте checklist передачи модели эксплуатации. Сформулируйте проверяемое требование, продемонстрируйте альтернативу и приложите протокол хотя бы одного эксперимента.

7. Вопросы для текущего и итогового контроля

1. В чём назначение и ограничения принципа «Задачи MLOps»?
2. Как проверить на практике утверждение: «Свяжите задачу в трекере с экспериментом, артефактом и deployment-версией.»?
3. В чём назначение и ограничения принципа «Experiment tracking»?
4. Как проверить на практике утверждение: «Зарегистрируйте два запуска с разным max_depth и одинаковым test split.»?

5. В чём назначение и ограничения принципа «Model registry»?
6. Как проверить на практике утверждение: «Сохраните модель с run_id, signature и результатом acceptance tests.»?
7. В чём назначение и ограничения принципа «Data and code versioning»?
8. Как проверить на практике утверждение: «Проверьте возможность повторного запуска через фиксированную дату и data hash.»?
9. В чём назначение и ограничения принципа «Training pipeline»?
10. Как проверить на практике утверждение: «Создайте DAG с контрольным gate на минимальную долю валидных записей.»?
11. В чём назначение и ограничения принципа «CI/CD/CT»?
12. Как проверить на практике утверждение: «Разделите workflow для unit tests, training и manual production approval.»?
13. В чём назначение и ограничения принципа «Data validation gates»?
14. Как проверить на практике утверждение: «Сымитируйте ошибочную категорию и продемонстрируйте отказ стадии validate.»?
15. В чём назначение и ограничения принципа «Model acceptance gates»?
16. Как проверить на практике утверждение: «Сравните кандидата с champion на фиксированном holdout.»?
17. В чём назначение и ограничения принципа «Artifact lineage»?
18. Как проверить на практике утверждение: «Проследите model_v7 до upstream dataset snapshot и commit.»?
19. В чём назначение и ограничения принципа «Рабочие окружения»?
20. Как проверить на практике утверждение: «Опишите тестовый стенд для staging с синтетическими персональными данными.»?
21. В чём назначение и ограничения принципа «Релизная стратегия»?
22. Как проверить на практике утверждение: «Проверьте новую модель на зеркальных запросах без влияния на пользовательские решения.»?

23. В чём назначение и ограничения принципа «Ответственность после релиза»?
24. Как проверить на практике утверждение: «Разработайте checklist передачи модели эксплуатации.»?
25. Почему высокое offline-качество не гарантирует надежность промышленной AI-системы?
26. Какие артефакты нужны для воспроизводимости и анализа инцидента?

8. Глоссарий ключевых понятий

Задачи MLOps

MLOps систематизирует путь от идеи до контролируемого изменения production-модели.

Experiment tracking

Каждый запуск хранит параметры, метрики, версии кода и артефакты.

Model registry

Registry представляет жизненный цикл версий и стадии утверждения; он не просто папка весов.

Data and code versioning

Воспроизведение требует как данных, так и кода преобразования, зависимостей и конфигурации.

Training pipeline

Этапы extract, validate, transform, train, evaluate и register должны формировать артефакты и четкие ошибки.

CI/CD/CT

CI проверяет код и схемы, CD доставляет сервис, CT запускает повторное обучение по одобренному условию.

Data validation gates

Каждый pipeline должен останавливать сборку при отсутствии критических признаков, некорректной метке или резком изменении распределения.

Model acceptance gates

Кандидат не публикуют, если качество хуже допустимого, метрики по подгруппам недопустимы или latency превышает порог.

Artifact lineage

Происхождение артефакта связывает версии dataset, code, preprocessing и обученных весов.

Рабочие окружения

Development, staging и production разделяют секреты, нагрузки и доступ.

Релизная стратегия

Shadow, canary и blue-green rollout уменьшают риск; выбор зависит от стоимости двойной инфраструктуры и характера ошибок.

Ответственность после релиза

Команда определяет owners, on-call, журнал изменений, политику rollback и сроки пересмотра модели.

9. Итоги и связь со следующей темой

Представленные методы имеют смысл как части целостного управляемого жизненного цикла. Ключевой результат инженерного проектирования состоит не в максимальном числе компонентов или сложности модели, а в убедительной связи требований, проектных решений, измерений и механизмов сопровождения. Дальнейшее изучение курса развивает эти связи на следующем этапе жизненного цикла.

Список литературы

[1] Huyen C. Designing Machine Learning Systems. O'Reilly Media, 2022. https://www.oreilly.com/library/view/designing-machine-learning/9781098107956/.

[2] MLflow. Official documentation: Tracking, Models, Registry. https://mlflow.org/docs/latest/index.html.

[3] Breck E. et al. The ML Test Score: A Rubric for ML Production Readiness and Technical Debt Reduction. 2017. https://research.google/pubs/the-ml-test-score-a-rubric-for-ml-production-readiness-and-technical-debt-reduction/.

[4] Google Developers. Rules of Machine Learning: Best Practices for ML Engineering. https://developers.google.com/machine-learning/guides/rules-of-ml.

[5] Sculley D. et al. Hidden Technical Debt in Machine Learning Systems. NeurIPS, 2015. https://research.google/pubs/hidden-technical-debt-in-machine-learning-systems/.

[6] Kleppmann M. Designing Data-Intensive Applications. O'Reilly Media, 2017. https://dataintensive.net/.