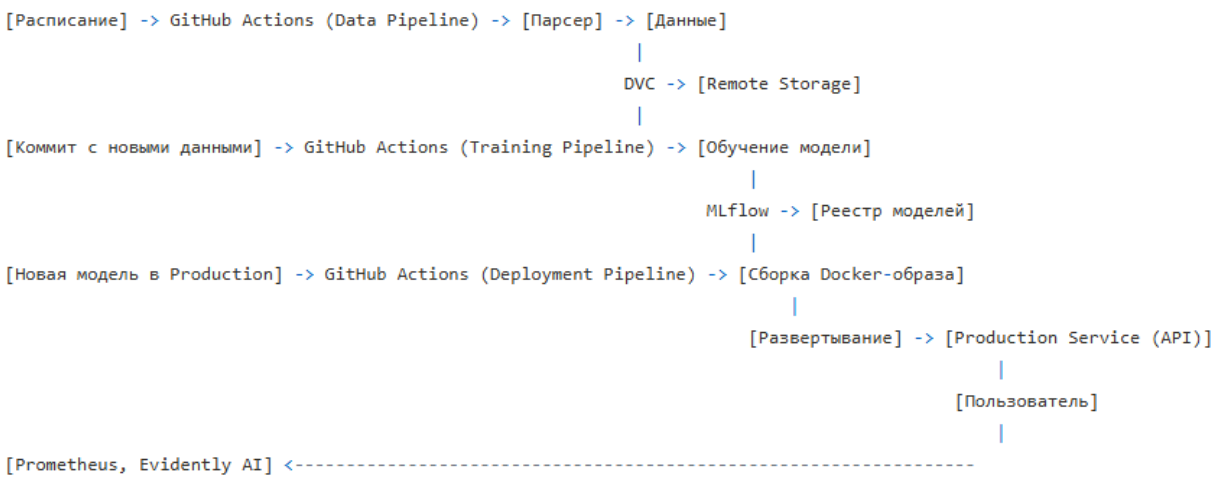
Антонян Давид
5140904/50201
Управление конфигурацией ПО

Тема курсовой работы

1. Предлагаемая тема – Разработка и внедрение MLOps-конвейера для сервиса прогнозирования стоимости недвижимости.
2. Автоматизированный MLOps-конвейер для предиктивных моделей на рынке недвижимости
3. Целью проекта является преобразование прототипа сервиса по предсказанию стоимости недвижимости в автоматизированную MLOps-систему. Проект должен включать в себя весь цикл модели машинного обучения от автоматического сбора свежих данных с сайта-источника до непрерывного обучения, версионирования, развертывания и мониторинга модели в среде, приближенной к продуктивной. Итогом работы должен стать воспроизводимый пайплайн (конвейер), который позволяет модели адаптироваться к динамичным изменениям на рынке недвижимости без ручного вмешательства.
4. Git (Контроль версий), DVC, MLFlow (управление циклом модели МО), Docker (контейнеризация), Flask (API, сервер для запуска приложения, взаимодействие с веб-интерфейсом), scikit-learn (модель МО и обучение), pandas (анализ данных), PostgreSQL (бд), Evidently AI (Мониторинг качества модели МО)

5. Компоненты проекта:
    a. Источник данных (например, сайт ЦИАН).
    b. Репозиторий (например, GitHub) - Центральное место для хранения всего кода, конфигураций DVC и пайплайнов GitHub Actions.
    c. Хранилище данных и моделей (DVC Remote): Внешнее хранилище (например, Google Drive, S3-совместимое) для версионируемых датасетов и артефактов моделей.
    d. Пайплайн: Workflow в GitHub Actions, который по расписанию запускает парсер, сохраняет новые данные и версионирует их с помощью DVC.
    e. Конвейер обучения: Workflow, который запускается после обновления данных. Он извлекает актуальный датасет, обучает модель, логирует результаты в MLflow Tracking Server и, если метрики удовлетворительные, регистрирует новую версию модели в MLflow Model Registry.
    f. Конвейер развертывания: Workflow, который срабатывает при появлении новой модели в статусе "Production" в реестре. Он упаковывает сервис с новой моделью в Docker-образ, отправляет его в реестр контейнеров и разворачивает на целевом сервере.
    g. Продуктивный сервис: Docker-контейнер с Flask-приложением, которое предоставляет API для получения прогнозов.
    h. Система мониторинга: Отслеживает работоспособность сервиса и качество предсказаний модели.

Пример схемы:



6. План:

1. Настройка инфраструктуры и окружения

    1. Создание репозитория на GitHub. Настройка Git
    2. Инициализация DVC в проекте и настройка удаленного хранилища (например, папка в Google Drive).
    3. Установка и локальный запуск MLflow Tracking Server.
    4. Первичное версионирование существующего кода и данных.

- Применение DevOps:
    - Весь код помещается в Git. Первоначальный набор данных версионируется с помощью DVC.
    - Infrastructure as Code (IaC): Создание файла requirements.txt или environment.yml для описания зависимостей. Написание скриптов для запуска локальных сервисов (например, MLflow).

2. Автоматизация конвейера данных
    1. Адаптация парсера для работы в автоматическом режиме (обработка ошибок, логирование).
    2. Создание первого workflow в GitHub Actions (data-pipeline.yml), который:
        - Запускается по расписанию (например, раз в неделю).
        - Выполняет скрипт парсера.
        - Добавляет новые данные под контроль DVC (dvc add).
        - Отправляет данные в удаленное хранилище (dvc push).
        - Создает коммит с обновленными DVC-файлами.

- Применение DevOps:
    - Automation: Полная автоматизация рутинной задачи сбора данных.
    - Continuous Integration (for data): Каждый запуск конвейера интегрирует новый набор данных в проект, делая его доступным для всех последующих этапов.

3. Автоматизация конвейера обучения
    1. Интеграция MLflow Tracking в скрипт обучения модели "случайный лес".
    2. Создание второго workflow в GitHub Actions (training-pipeline.yml), который:
        - Запускается при появлении нового коммита в основной ветке, затрагивающего данные.
        - Извлекает актуальную версию данных (dvc pull).

        - Запускает скрипт обучения.
        - Регистрирует обученную модель и ее метрики в MLflow Model Registry.
- Применение DevOps/MLOps:
    - Continuous Training (CT): Модель автоматически переобучается на свежих данных.
    - Experiment Tracking: Каждый запуск обучения логируется, обеспечивая воспроизводимость и возможность сравнения моделей.

4. Контейнеризация и непрерывное развертывание
    1. Разработка простого Flask API для загрузки модели и выдачи предсказаний.
    2. Написание Dockerfile для упаковки Flask-приложения и модели в единый контейнер.
    3. Создание третьего workflow (deployment-pipeline.yml), который:
        - Запускается вручную или по триггеру (например, по созданию тега Git).
        - Извлекает последнюю "Production" модель из MLflow Registry.
        - Собирает Docker-образ.
        - (Для курсовой) Выводит команду для ручного запуска контейнера или, при наличии сервера, разворачивает его.
- Применение DevOps/MLOps:
    - Containerization: Использование Docker для создания переносимого и изолированного артефакта развертывания.
    - Continuous Deployment (CD): Автоматизация процесса доставки готовой к работе модели пользователям.

5. Настройка мониторинга
    1. Добавление в Flask-приложение эндпоинта /metrics для Prometheus.
    2. Создание скрипта, который периодически (или в рамках конвейера) использует Evidently AI для сравнения свежих данных с референсным датасетом и генерирует HTML-отчет о дрейфе данных.
    3. Написание раздела в курсовой работе, описывающего, как эти отчеты и метрики помогают поддерживать качество модели.
- Применение DevOps/MLOps:
    - Monitoring & Logging: Внедрение инструментов для отслеживания состояния системы и выявления проблем до того, как они повлияют на пользователя. Это замыкает цикл MLOps, предоставляя обратную связь для дальнейшего улучшения модели.