Описание технической архитектуры
программного обеспечения
«Система поиска tvcAI КОЗ 1 Автономный поиск»

Москва, 2025

Термины и определения

Jetson	–	линейка компактных высокопроизводительных модулей для периферийных вычислений (edge computing) и искусственного интеллекта (AI), разработанных компанией NVIDIA. Эти модули сочетают GPU, CPU, память и интерфейсы на одной плате, обеспечивая энергоэффективную обработку данных в реальном времени.
CUDA	–	программно-аппаратная архитектура параллельных вычислений, которая позволяет существенно увеличить вычислительную производительность благодаря использованию графических процессоров фирмы Nvidia
JetPack	–	программный пакет разработчика (SDK) от NVIDIA, предназначенный для работы с модулями Jetson. Он включает операционную систему, драйверы, библиотеки и инструменты, оптимизированные для встроенных систем и искусственного интеллекта на платформе Jetson
Torch/Pytorch	–	фреймворк машинного обучения для языка Python с открытым исходным кодом, созданный на базе Torch. Используется для решения различных задач: компьютерное зрение, обработка естественного языка. Разрабатывается преимущественно группой искусственного интеллекта Facebook
ПО	–	программное обеспечение

1. Общие сведения

Настоящий документ содержит информацию, необходимую для описания технической архитектуры программного обеспечения «Система поиска tvcAI КОЗ 1 Автономный поиск». Исключительные права на программное обеспечение принадлежат Фонду НТИ (далее – Компания).

Настоящий документ подлежит размещению на официальном сайте Компании в сети Интернет по адресу: https://nti.fund/about/activity/information.php (далее – официальный сайт).

ПО «Система поиска tvcAI КОЗ 1 Автономный поиск» предназначено для автоматизированного поиска людей в природной местности.

ПО обеспечивает автоматическое распознавание людей на фотографиях, полученных с микрокомпьютера, установленный на беспилотном воздушном судне, на высоте от 25 до 90 м с перекрытием в не менее 50%, с применением технологий искусственного интеллекта.

ПО не имеет пользовательского графического интерфейса.

2. Описание архитектуры приложения

2.1. Диаграмма классов

Диаграмма классов программного обеспечения «Система поиска tvcAI КОЗ 1 Автономный поиск» приведена на рисунке 1.

1. **YOLO** (из библиотеки Ultralytics):
    - Отвечает за выполнение предсказаний
    - Содержит конфигурацию модели и логику инференса
2. **Predictor** :
    - Инкапсулирует логику обработки изображений
    - Поля:
        - **model**: Экземпляр YOLO-модели
        - **imgh**, **imgw**: Размеры изображения для обработки
    - Методы:
        - **process_image()**: Обрабатывает одно изображение и возвращает результаты
        - **predict()**: Обрабатывает список изображений
3. **BoxResult** :
    - DTO (Data Transfer Object) для хранения результатов детекции
    - Содержит нормализованные координаты bounding box и confidence
4. **ImageLoader** :
    - Отвечает за загрузку изображений из файловой системы
5. **Main** :
    - Точка входа приложения
    - Оркестрирует процесс: загрузка данных → предсказание → вывод результатов

.

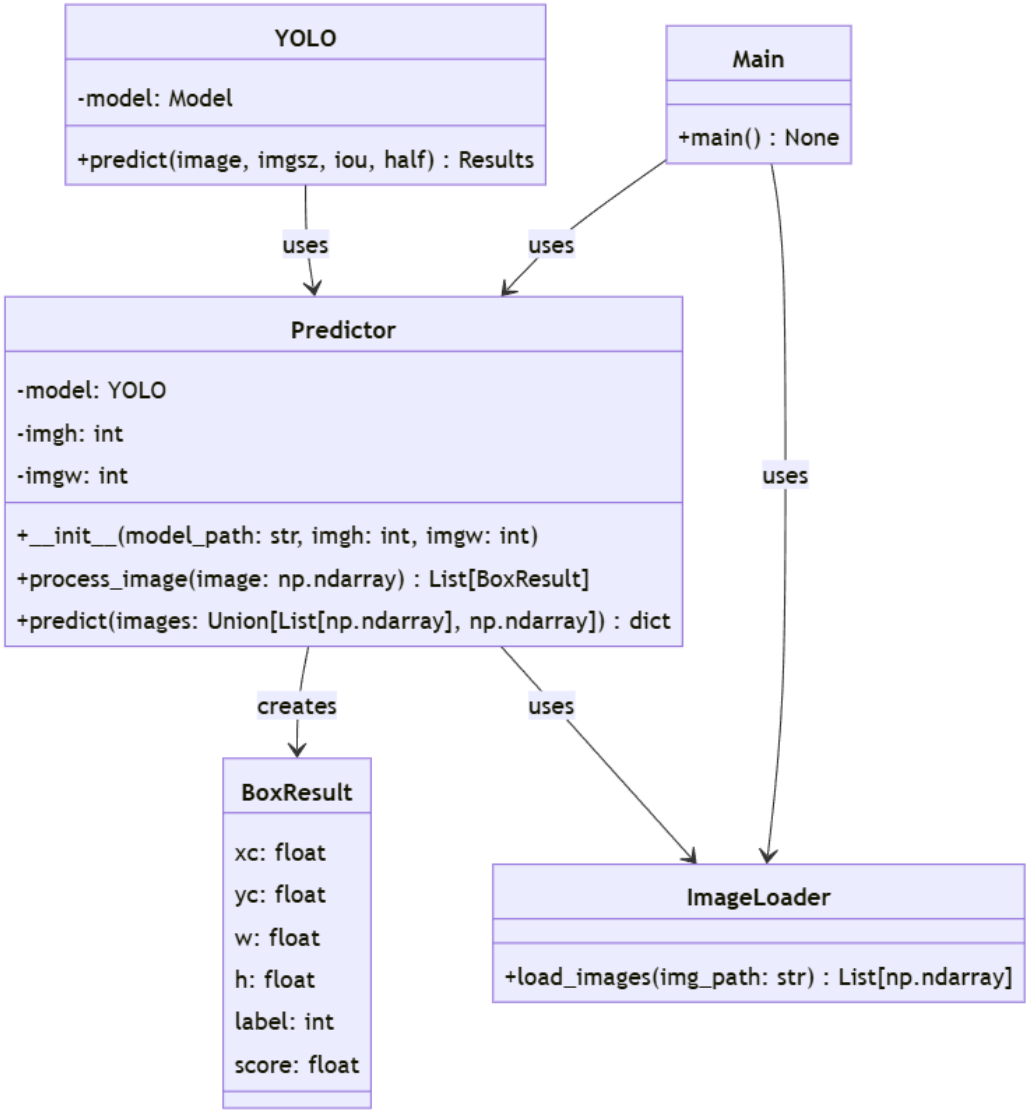

Рисунок 1 – Диаграмма классов

Основные взаимодействия:

1. Main использует ImageLoader для загрузки изображений
2. Main вызывает метод predict() у экземпляра Predictor
3. Predictor использует модель YOLO для выполнения предсказаний
4. Результаты обрабатываются и возвращаются в виде списка объектов BoxResult

Примечание: В текущем коде часть логики реализована в виде отдельных функций. Диаграмма показывает, как можно структурировать код в объектно-ориентированном стиле для повышения модульности и удобства тестирования.

2.2. Диаграмма последовательностей

Диаграммы последовательностей программного обеспечения «Система поиска tvcAI КОЗ 1 Автономный поиск» приведена на рисунке 2:

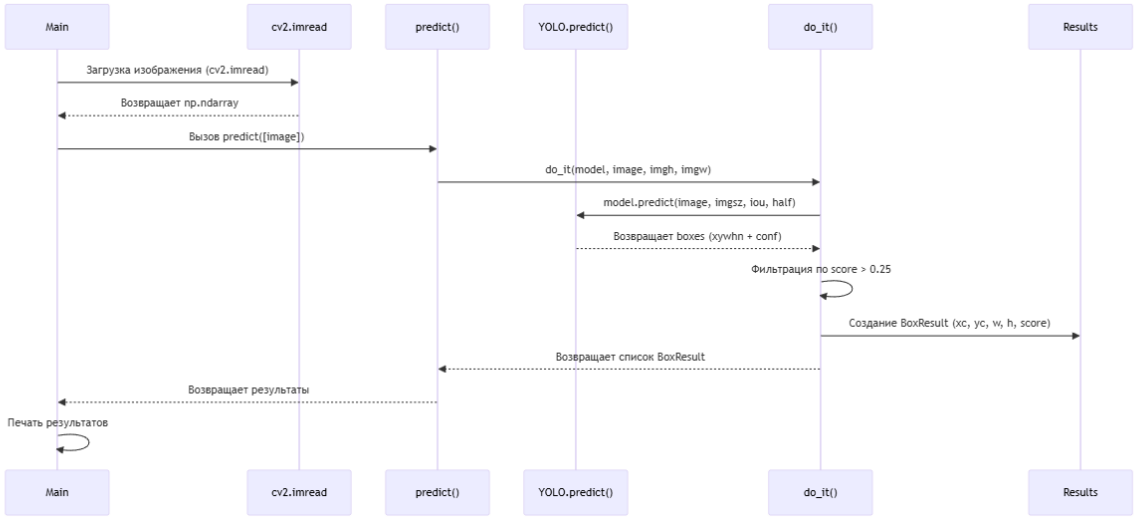

Рисунок 2 – Диаграмма последовательности

Основные шаги процесса:

1. **Загрузка изображения** :
    - **Main** использует **cv2.imread** для чтения изображения из файла
    - Изображение возвращается в виде NumPy массива
2. **Запуск предсказания** :
    - **Main** вызывает функцию **predict()**, передавая список изображений
    - В текущей реализации обрабатывается только первое изображение в списке
3. **Обработка в do_it()** :
    - Выполняется предсказание через YOLO модель
    - Модель возвращает нормализованные координаты bounding boxes (**xywhn**) и confidence
4. **Фильтрация и формирование результата** :

- Фильтрация детекций с confidence > 0.25
- Создание структур **BoxResult** с округлением значений до 4 знаков

5. **Возврат результатов** :
    - Результаты возвращаются в **Main** и выводятся в консоль

Особенности реализации:

- **Прогрев модели** : В начале кода выполняется 10 тестовых прогонов модели на пустом тензоре для инициализации
- **Фиксированные размеры** : Размеры изображения (2752x4096) жестко заданы в коде
- **Ограничение** : В текущей реализации обрабатывается только первое изображение из переданного списка

2.3. Диаграмма прецедентов

Диаграмма прецедентов программного обеспечения «Система поиска tvcAI КОЗ 1 Автономный поиск» приведена на рисунке 3.

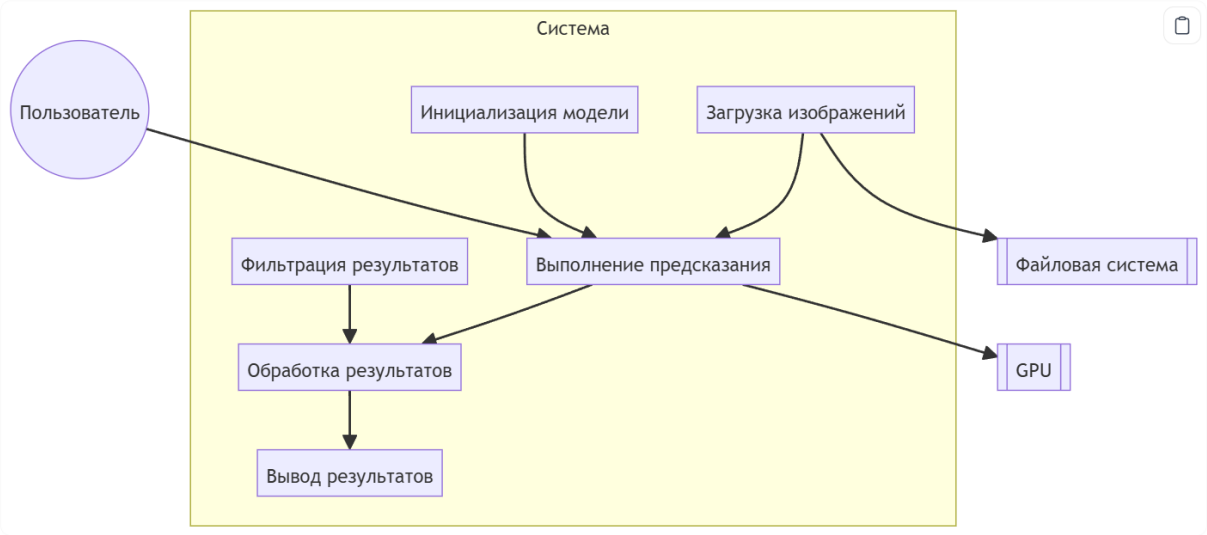

Рисунок 3 – Диаграмма прецедентов

Основные элементы:

1. **Акторы** :
    - Пользователь (инициирует процесс)
    - Файловая система (источник данных)
    - GPU (аппаратный ускоритель)
2. **Прецеденты** :
    - Загрузка изображений
    - Выполнение предсказания
    - Обработка результатов
    - Вывод результатов
    - Инициализация модели
    - Фильтрация результатов

3. Описание архитектурного стиля

При выборе code style использовался только PEP8.

Архитектура представленного кода сочетает несколько стилей. Основу составляет процедурный подход с линейным выполнением операций, глобальными переменными (model, imgh, imgw) и отсутствием объектно-ориентированной структуры, что видно по функциям predict() и do_it(), а также блоку if __name__ == "__main__". Данные обрабатываются по конвейерной схеме (Pipe-Filter): изображения последовательно проходят этапы загрузки → предсказание модели → фильтрацию результатов → вывод, что формирует чёткую цепочку преобразований.

Код активно использует клиент-сервисные взаимодействия через внешние библиотеки: ultralytics.YOLO для инференса, cv2.imread для загрузки данных и numpy для работы с массивами. Несмотря на отсутствие явной event-driven логики, обработка каждого изображения в цикле for img_name in img_names напоминает архитектуру на основе событий, где каждая итерация — отдельный "событийный" цикл.

Особое внимание уделено аппаратной оптимизации: задействован GPU (device='cuda'), half-precision вычисления (half=True) и прогрев CUDA-кэша (10 тестовых прогонов). Конфигурационные параметры (imgh = 2752, iou=0.4, порог score > 0.25) жёстко вшиты в код, что характерно для конфигурационно-ориентированного стиля . Данные обрабатываются пакетно, а результаты формируются в структурированном виде (списки словарей), что отражает Data-Driven подход.

Всё это формирует монолитную архитектуру с тесной связью компонентов и отсутствием модульности.

Для связи с разработчиками писать на почту ntifundsoft@nti.fund