Отчет по проделанной работе.

#### Введение.

В данной работе необходимо выполнить задание по подготовке, оптимизации и развертыванию модели машинного обучения с использованием современных инструментов. Цель проекта – освоить процессы обучения, конвертации, оптимизации и интеграции моделей в продакшн-среду с применением Trition Inference Server, Docker микросервисной архитектуры.

#### Разработка и обучение модели.

Для обучения нейросетевой модели использовался датасет MNIST, содержащий изображения рукописных цифр. Архитектура модели была реализована с использованием фреймворка PyTorch. Процесс обучения выполнялся на видеокарте NVIDIA RTX 3080 с задействованием вычислений на CUDA-ядрах для ускорения.

По завершении обучения веса модели были экспортированы в форматы TorchScript и ONNX для дальнейшего использования и оптимизации в среде Triton Inference Server.

#### Оптимизация Torch.

Для повышения эффективности инференса на CPU была проведена **оптимизация модели на основе PyTorch** с использованием встроенных средств квантизации. Метод статической квантизации позволяет заменить вычисления с плавающей точкой (FP32) на целочисленные (INT8), что существенно снижает требования к памяти и ускоряет выполнение модели без значительной потери точности.

#### Оптимизация ONNX.

После экспорта модели в формат ONNX была проведена оптимизация с использованием **инструментов ONNX Runtime.**

#### ****Разработка микросервиса предобработки данных.****

Для подготовки входных изображений перед инференсом модели был реализован микросервис предобработки данных на базе фреймворка FastAPI. Микросервис выполняет нормализацию и преобразование изображений в формат, совместимый с входными требованиями нейронной сети, развернутой в Triton Inference Server.

Функциональность микросервиса:

* Приём изображений по HTTP:  
  Используется POST-метод /preprocess, принимающий изображение в формате multipart/form-data.
* Обработка изображений:  
  Загруженное изображение преобразуется в оттенки серого (L), масштабируется до размера 28×28 пикселей, нормализуется в диапазон [0, 1], и приводится к тензору с формой (1, 1, 28, 28) (batch size = 1, 1 канал).
* Ответ в формате JSON:  
  Обработанные данные возвращаются в виде списка чисел (тензор, сериализованный в JSON), пригодного для последующей отправки в Triton Inference Server.

#### Развернутое моделей с помощью Triton Inference Server.

Для обеспечения высокопроизводительного инференса нейронных сетей была использована платформа **Triton Inference Server**, разработанная NVIDIA. Triton обеспечивает унифицированный способ подачи данных и выполнения моделей в различных форматах, включая PyTorch, TensorFlow и ONNX.

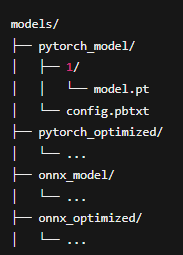


Рисунок 1. Структура Triton Inference Server

#### Мониторинг метрик.

#### Для получения информации о работе модели и микросервиса в реальном времени была настроена система мониторинга на базе **Prometheus** и **Grafana**. Она позволила отслеживать ключевые метрики, такие как количество запросов, время инференса и использование ресурсов.

#### **Prometheus** используется в качестве основного инструмента для сбора и хранения метрик. Он осуществляет периодические запросы (scraping) к сервисам, поддерживающим экспорт метрик в формате /metrics.

Для визуализации данных используется Grafana. Prometheus был добавлен в качестве источника данных, после чего был импортирован дашборд, содержащий ключевые панели:

* Количество инференсов в секунду (Triton)
* Средняя длительность инференса (µs)
* Время ответа FastAPI-сервиса (p95)
* Загрузка CPU и потребление памяти микросервисом

#### 

Рисунок 2. Ключевые метрики Prometheus

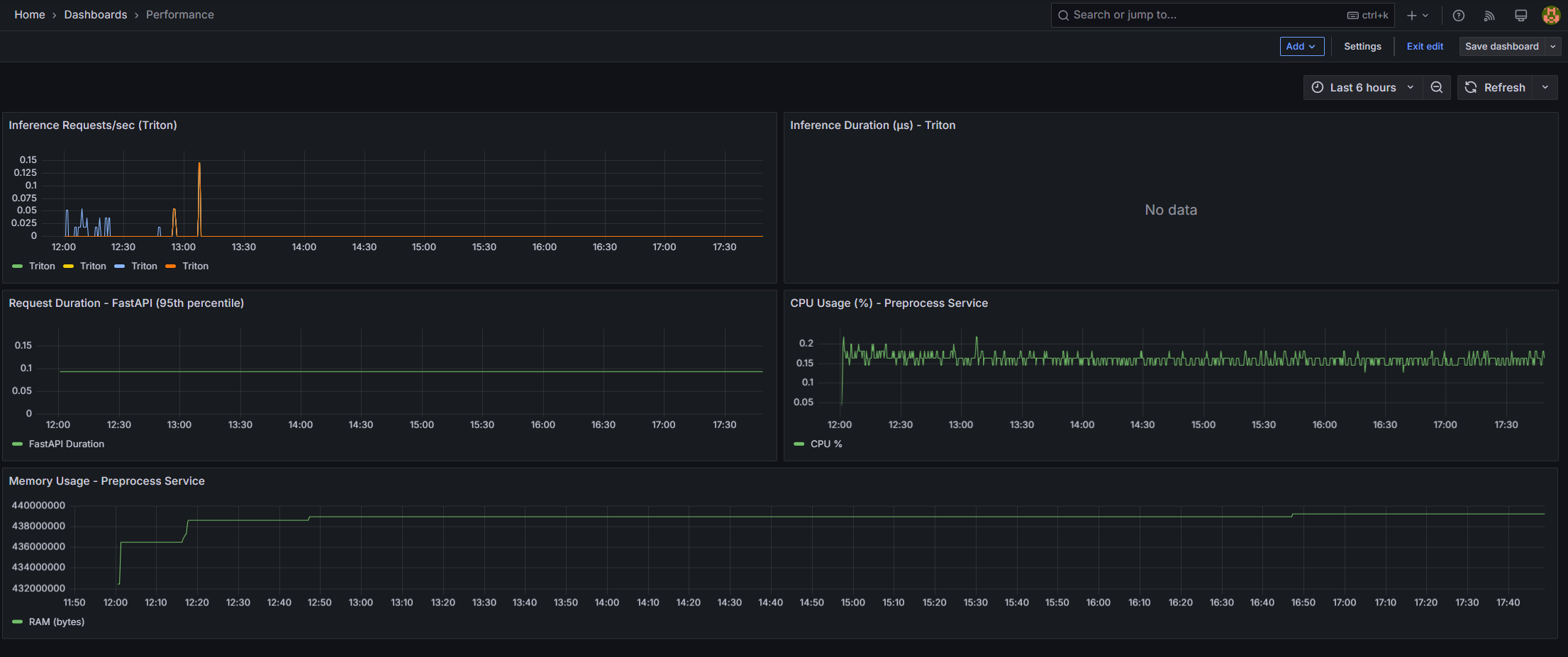


Рисунок 3. Дашборд анализза производительности.

#### Оркестрация сервисов с помощью docker-compose

#### Для удобства запуска, масштабирования и взаимодействия между компонентами системы была реализована оркестрация сервисов с использованием инструмента ****Docker Compose****. Это позволило объединить все элементы решения — микросервис предобработки данных, Triton Inference Server, систему мониторинга (Prometheus и Grafana) — в единую, управляемую среду.

|  |  |
| --- | --- |
| Сервис | Назначение |
| Preprocess | FastAPI микросервис предобработки изображений |
| Triton | Triton Inference Server для запуска моделей |
| Prometheus | Сбор метрик от Triton и FastAPI |
| Grafana | Визуализация метрик и создание дашбордов |

Использование Docker Compose позволило:

* быстро разворачивать всю систему в изолированной среде;
* обеспечить воспроизводимость при переносе проекта между машинами;
* упростить запуск, отладку и демонстрацию решения.

Таким образом, была достигнута полная контейнеризация и изоляция компонентов, необходимая для стабильного запуска моделей и сопутствующих сервисов в составе распределённой ML-инфраструктуры.

#### Тестирование

#### После развертывания всех версий модели в Triton Inference Server была проведена серия тестов, направленных на сравнение **времениотклика** и **точностипредсказаний** для каждой модели. Были протестированы четыре версии:

1. PyTorch — оригинальная модель на PyTorch
2. ONNX — экспортированная ONNX-модель
3. ONNX Optimized — оптимизированная ONNX-модель
4. PyTorch Optimized — квантованная модель на PyTorch

Для финальной оценки эффективности различных версий модели были рассчитаны средние времена инференса на основе нескольких тестовых изображений, которых не было в обучающей выборке. Полученные значения отражают усреднённую задержку выполнения одного запроса на предсказание.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Модель |  |  |  |  |
| PyTorch | Предсказание: 2  Время: 5.41 мс | Предсказание: 5  Время: 7.85 мс | Предсказание: 6  Время: 5.81 мс | Предсказание: 8  Время: 4.79 мс |
| ONNX | Предсказание: 2  Время: 3.8 мс | Предсказание: 5  Время: 4.13 мс | Предсказание: 6  Время: 3.17 мс | Предсказание: 8  Время: 7.28 мс |
| ONNX Optimized | Предсказание: 2  Время: 5.53 мс | Предсказание: 5  Время: 4.55 мс | Предсказание: 6  Время: 4.71 мс | Предсказание: 8  Время: 5.02 мс |
| PyTorch Optimized | Предсказание: 2  Время: 6.30 мс | Предсказание: 5  Время: 4.20 мс | Предсказание: 6  Время: 4.30 мс | Предсказание: 8  Время: 4.43 мс |

|  |  |
| --- | --- |
| Модель | Среднее время обработки |
| PyTorch | 5.965 мс |
| ONNX | 4,595 мс |
| ONNX Optimized | 4,953 мс |
| PyTorch Optimized | 4,808 мс |

#### Вывод

По результатам тестирования можно сделать следующие выводы:

* Все модели корректно выполняют предсказание с одинаковой точностью.
* Использование ONNX-формата приводит к заметному снижению времени отклика.
* Дополнительная оптимизация (например, через ONNX Runtime или Torch Quantization) также положительно влияет на производительность, особенно на CPU.
* Для задач, где критична задержка, рекомендуется использовать экспорт в ONNX с последующей оптимизацией.

Таким образом, применение ONNX-модели является предпочтительным вариантом с точки зрения балансаскоростиисовместимости, особенно при развёртывании в системах инференса типа TritonInferenceServer.