OZONtech

Тритон-сервер - Архитектура и внутренняя кухня

Представил: Илья Жданов,
ML-инженер ОзонТех

ПЛАН ЛЕКЦИИ

Структура доклада:

- Зачем нужен тритон сервер?
- Конкуренция за ресурсы при мультипроцессинге
- Архитектура тритон сервера
- Подробнее про компоненты сервера:
  1) Triton’s Core
  2) Frameworks for models
  3) Scheduler
  4) Sequence batcher
  5) Dynamic batcher
  6) Model repository manager
  7) Metrics exporter

ЗАЧЕМ НУЖЕН ТРИТОН СЕРВЕР ?

Приложение должно быть быстрым у пользователя – и не всегда у него есть кластер из 16 видеокарточек.

Требования к приложению

- Быстрый отклик в рантайме
- Устойчивость к большому наплыву запросов
- Масштабируемость нашего решения
- Простая поддержка созданного приложения новыми сотрудниками компании
- Экономия используемых ресурсов GPU и CPU

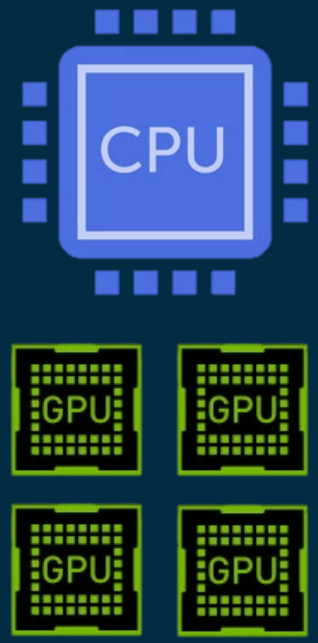
ЧЕМ ХОРОШ ТРИТОН СЕРВЕР ?

Open-source решение для инференса моделей от NVIDIA

1. **Разнообразие моделей**
   Поддерживаются различные фреймворки, такие как TensorRT, TensorFlow, PyTorch, ONNX, OpenVINO, Python, FIL и другие
2. **Работа с различными ресурсами**
   Можно развернуть и в облаке, и в дата-центре, и локально.
   Может работать как с GPU, так и в CPU-only формате.
3. **Способность выдерживать высокую нагрузку**
   Параллельный асинхронный инференс моделей.
   Возможность настраивать число инстансов одной модели.
4. **Возможность мониторинга**
   Model Analyzer и Performance Analyzer.
   Метрики с загрузкой GPU, временем ответа и т.д.
5. **Дополнительные возможности для инференса**
   Организация моделей в ансамбли.
   Различные инструменты для батчевания.

01 { Архитектура Тритон-сервера

КОНКУРЕНЦИЯ ЗА РЕСУРСЫ

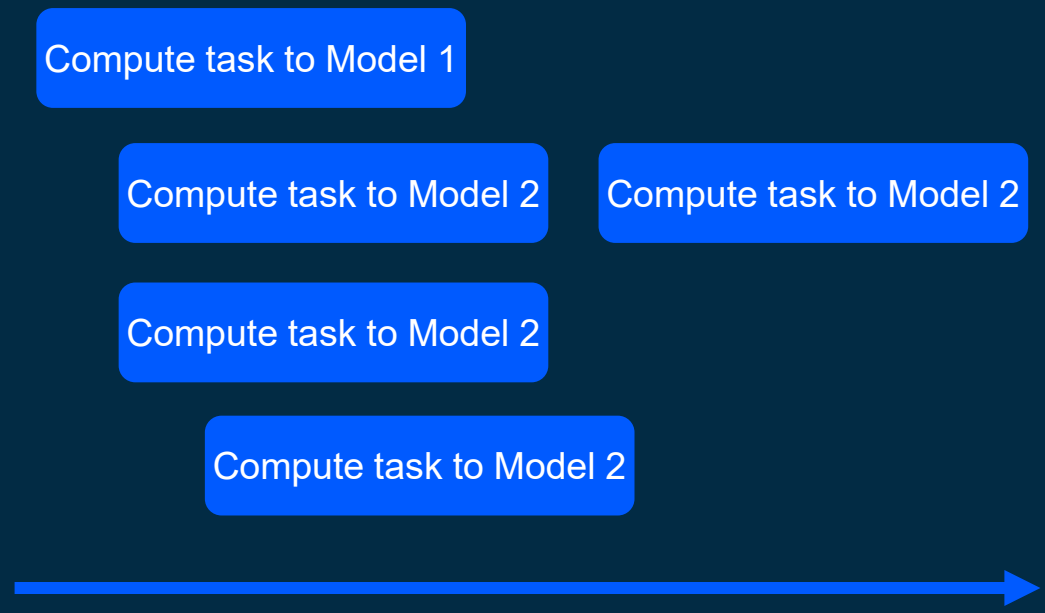


Ссылка на материалы NVIDIA

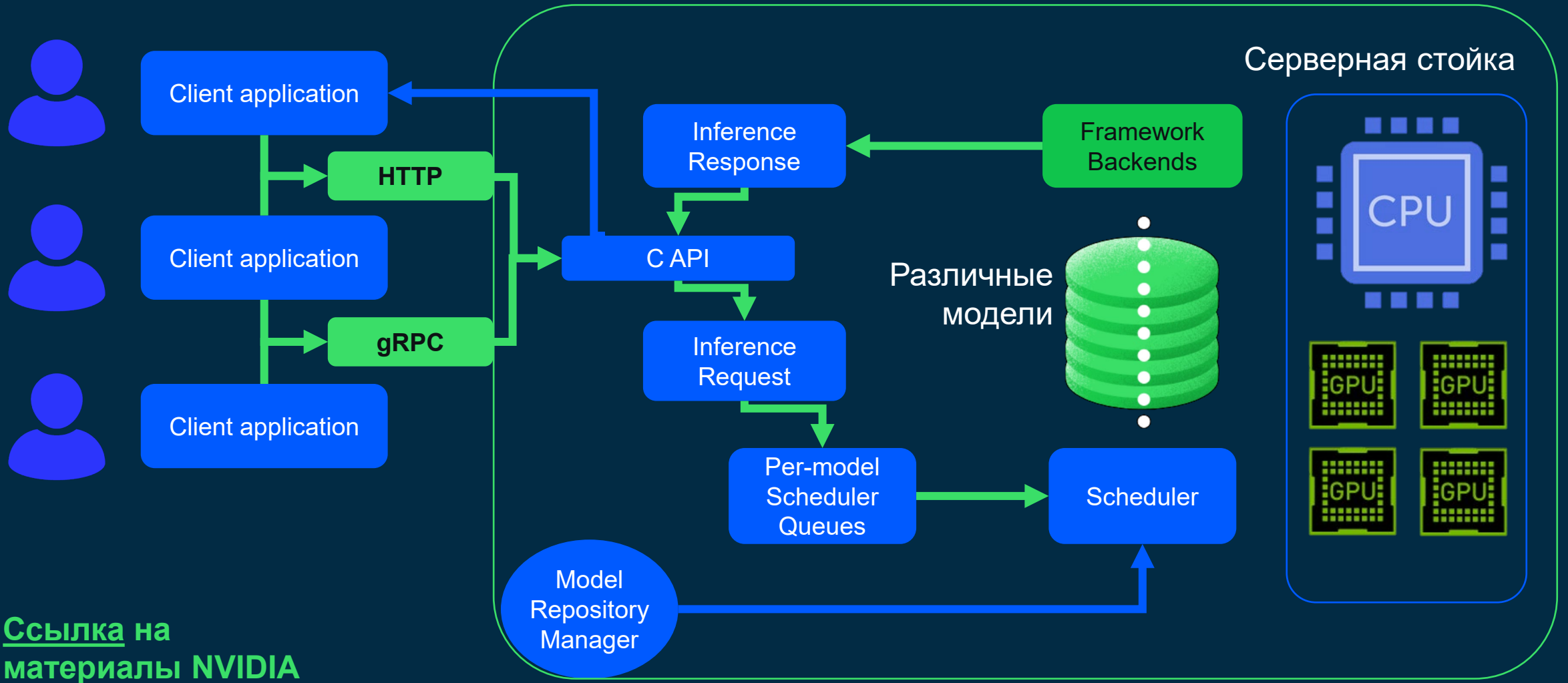
Ограничения серверной стойки:

1. **Центральный процессор (*CPU*)** – представляет из себя набор ядер, управляемых системой (*OS*). Доступ к конкретному ядру называется потоком (*thread*).
2. **Оперативная память (*RAM*)** – память с быстрым откликом для хранения промежуточных расчётов.
3. **Видеокарты (*GPUs*)** – каждая видеокарта это, по сути, набор ядер и памяти, специализирующихся на быстрых расчётах.
4. **Долговременная память (*HDD/ SSD*)** – дисковое пространство для хранения моделей, конфигов и прочего.

Скорость проведения расчётов ограничена железом серверной стойки.



АРХИТЕКТУРА ТРИТОНА



Ссылка на материалы NVIDIA

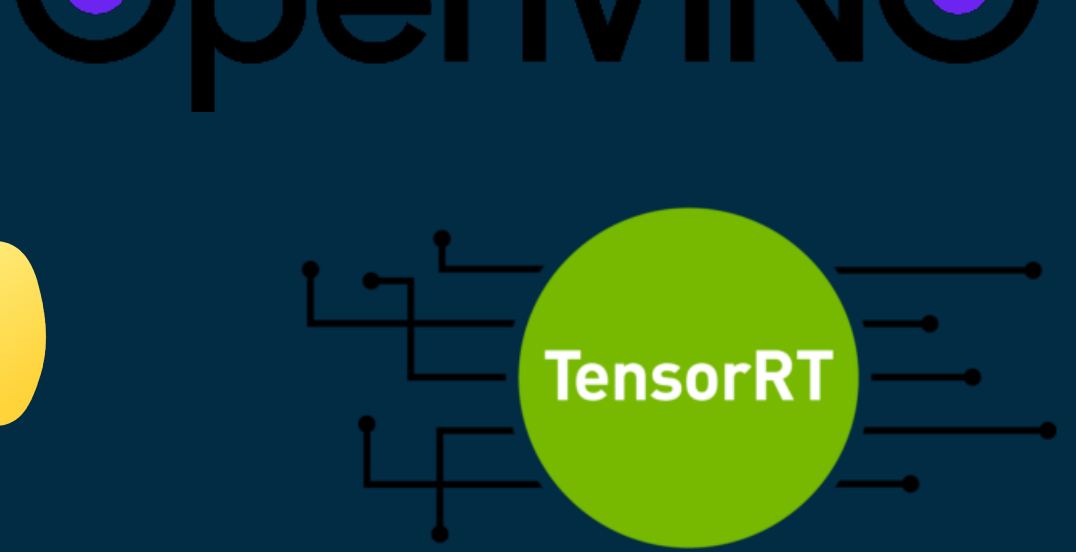
TRITON'S CORE

Основные компоненты ядра сервера:

1. **Backend:**
   - Фиксирует регионы временной памяти под модели, чтобы избежать её переполнения
   - Управляет реквестами в C API
   - Создаёт инстансы моделей
   - Scheduler and Batchers
2. **RepoAgent:**
   - Загрузка данных моделей в RAM
   - Работа с облачными хранилищами
3. **Server:**
   - Управление реквестами между моделями
   - Сбор метрик и логов внутри сервера
4. **Cache:**
   - Управление кэшами внутри тритона (Redis)
5. **Python:**
   - Написание моделей предобработки данных
   - Создание дерева ансамблей типо DAG'а
   - API для управления соседними моделями

Ссылка 1 на ядро сервера
Ссылка 2 на компоненты питона

FRAMEWORKS FOR MODELS



Ссылка 1 на поддерживаемые бекэнды

SCHEDULER

Состояния моделей:

Stateless

- Каждый инфер модели независим от других запросов к ней

Statefull

- Модель запоминает предыдущее состояние – или инициализирует заданное в конфиге

Ensemble

- Ансамбль содержит в себе очередность обращений к другим моделям

Ссылка 1 на материалы NVIDIA

Стратегии Scheduler'а:

Default

- Реквест не подвергается батчингу с другими

Direct

- Создаётся жёсткая привязка реквеста к слоту в батче

Oldest

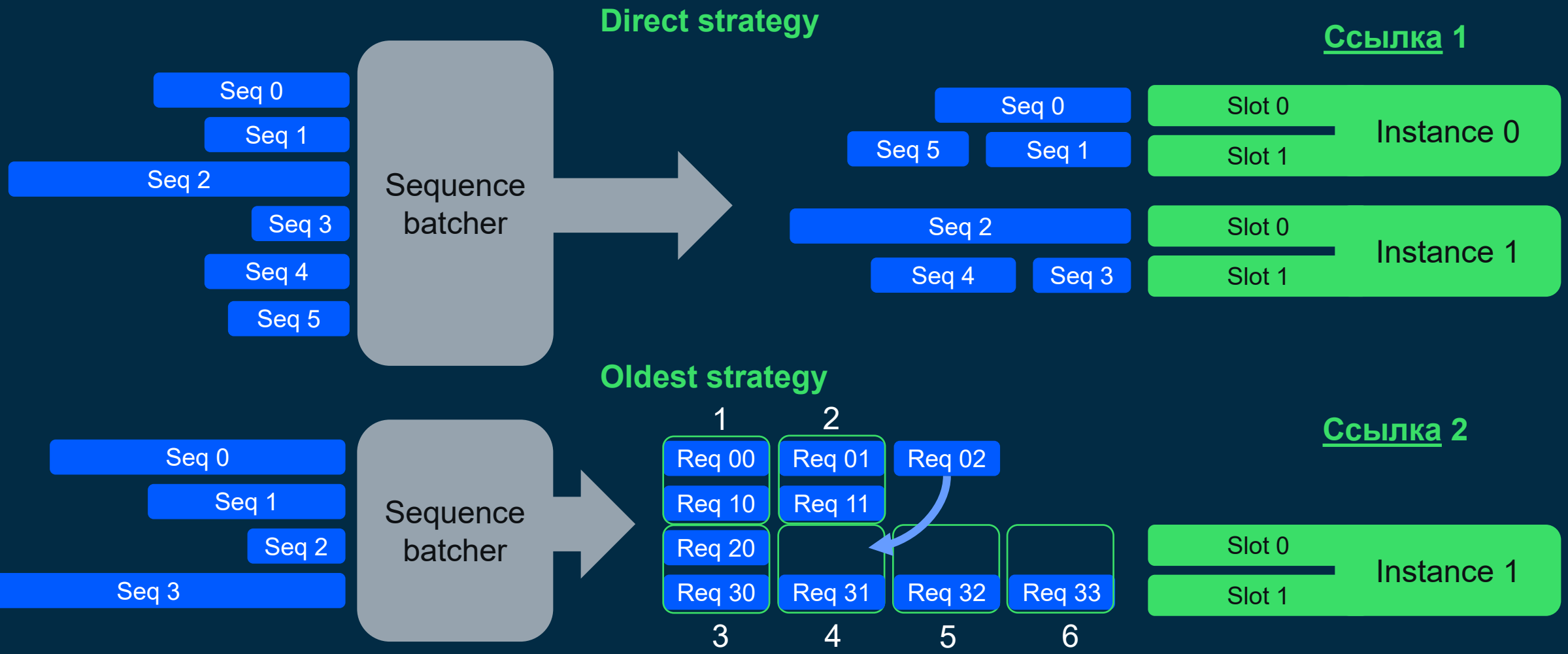
- Большой реквест делится на меньшие, которые комбинируются

Model Transaction policy:

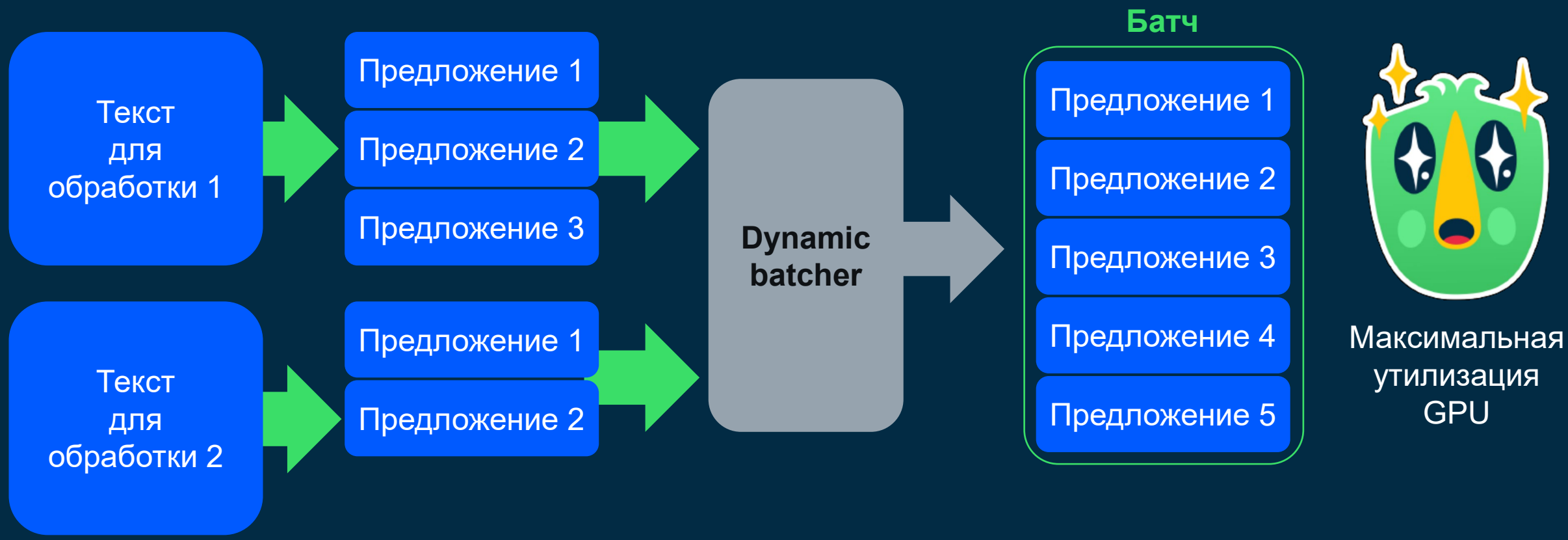
Decoupled

- Когда число респонсов не равно числу реквестов

SEQUENCE BATCHER



DYNAMIC BATCHER



Ссылка

MODEL REPOSITORY MANAGER

Model control mode

None
Тритон пытается загрузить всё, что находится в репозитории.
Но только один раз и игнорирует все внешние команды.

Poll
Также производится загрузка всех моделей при запуске тритон-сервера, однако все изменения моделей в репозитории детектируются раз в какое-то время, после чего тритон автоматически подгружает новые версии моделей взамен старых.

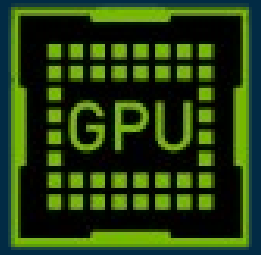
Explicit
Тритон не предпринимает никаких действий пока не придёт команда по загрузке модели из репозитория.

Model repository

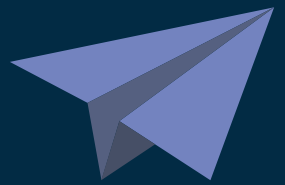
```
├── model name
│   ├── model version
│   │   └── model files
│   └── model configuration
└── text_recognition
    ├── 1
    │   └── model.py
    ├── conda_env.tar.gz
    └── config.pbtxt
```

METRICS EXPORTER

CPU
Собирается статистика утилизации и использованной памяти из /proc/meminfo.
Доступно только на Linux.



GPU
Замеряется утилизация каждой видеокарточки и кол-во использованной памяти.
Также собирается информация о потреблении электроэнергии.



Requests
Количество реквестов по статусам прохождения через модели внутри.
Также время их вычислений с разбиением на время ожидания в очереди, формирование тензоров на gpu, вычисление в модели и т.д.

Ссылка

Метрики живут по адресу: localhost:8002/metrics

РАЗЛИЧНЫЕ API ДЛЯ УПРАВЛЕНИЯ

With Frontend

KServe API

Полноценное управление тритон-сервером через его C API по gRPC/HTTP.

- SSL/TLS
- Compression
- KeepAlive

OpenAI API

Интеграция тритон-сервера с HuggingFace.

- vLLM
- TensorRT-LLM

Ссылка

Without Frontend

C++ API

Основное средство взаимодействия с ядром тритона через его сервер.

- Поддерживает все KServe extensions

Java API

Аналог C++ API, работающий на JAVA CPP

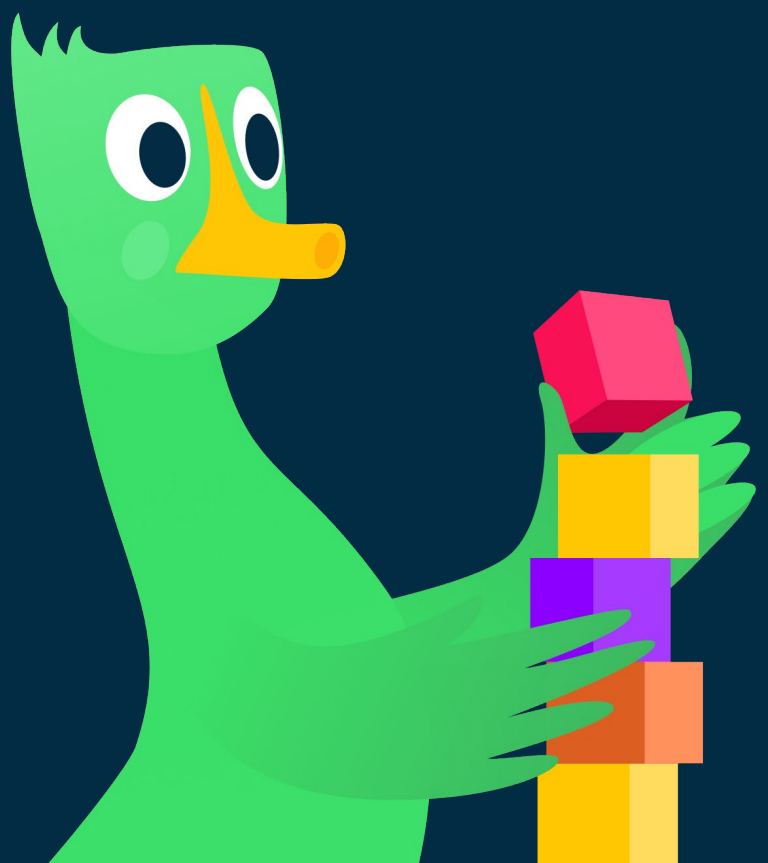
Python API

Это python shell, взаимодействующий с C API.

- Kafka I/O
- Rayserve

02 {

Семинар



ДАВАЙТЕ ЗАПУСТИМ !

Model repository

```
├── add_sub
├── 1
│   └── model.py
└── config.pbtxt
```

model.py

```
import triton_python_backend_utils as pb_utils

class TritonPythonModel:
    def initialize(self, args):
        pass
    def execute(self, requests):
        responses = []
        for request in requests:
            in_0 = pb_utils.get_input_tensor_by_name(
request, "INPUT0")
            out_0 = in_0.as_numpy()**2
            out_tensor_0 = pb_utils.Tensor("OUTPUT0", out_0)
            inference_response = pb_utils.InferenceResponse(
                output_tensors=[out_tensor_0]
            )
            responses.append(inference_response)
        return responses

    def finalize(self):
        print("Cleaning up...")
```

config.pbtxt

```
name: "add_sub"
backend: "python"

input [{
    name: "INPUT0"
    data_type: TYPE_FP32
    dims: [ 4 ]}]
output [{
    name: "OUTPUT0"
    data_type: TYPE_FP32
    dims: [ 4 ]}]

instance_group [{ kind: KIND_CPU }]
```

Ссылка

Домашнее задание 1

1. Необходимо взять любой чекпоинт модели-классификатора текстов или картинок с Hugging Face.
2. Чекпоинт необходимо обернуть в тритон-модель на базе Python Backend, результатом является model repository со всеми необходимыми компонентами.
   Предобработка данных, не считая приведения данных к нужному формату (например, байтам), должна производиться на стороне тритон-модели.
3. Помимо модели необходимо создать jupyter noteebook, с примерами обращения к вашей модели (не менее 5 примеров).
4. Результатом вашей работы должен стать zip архив с model repository и jupyter notebook из пунктов 2 и 3, соответственно.
5. Помимо качества решения оцениваются также качество и читаемость вашего кода.

Скачать контейнер с тритон-сервером:

Ссылка

Скачать модель классификатор с HuggingFace:

Ссылка

OZON tech

Группа разработки Генеративный дизайн и языковые модели

Илья Жданов
izhdanov@ozon.ru