### 1. Задание 1. Подготовка модели и создание конфигурационного файла для Triton

- Разработайте простую модель для задачи классификации (например, на PyTorch или TensorFlow).
- Экспортируйте модель в нужном формате. Для демонстрации работы с разными фреймворками подготовьте:
  - Модель в формате **ONNX** (например, экспортируйте модель из PyTorch в ONNX).
  - Модель, оптимизированную с помощью **TensorRT** (создайте TensorRT план, используя инструмент `trtexec` или аналогичный).
- Создайте структуру директории для модели в репозитории Triton:

```
model_repository/
└── my_model/
    ├── 1/
    │   └── <файлы модели>
    └── config.pbtxt
```

- В файле **config.pbtxt** пропишите параметры модели, указав:
  - Имя модели и соответствующую платформу (например, `"onnxruntime_onnx"` для модели ONNX или `"tensorrt_plan"` для модели TensorRT).
  - Максимальный размер пакета (`max_batch_size`).
  - Описание входных и выходных данных.
  - (Опционально) Настройки динамического пакетирования.
- Проверьте корректность структуры репозитория и валидность конфигурационного файла.

---

### 2. Задание 2. Развёртывание Triton Inference Server с использованием Docker

- Скачайте официальный образ Triton Inference Server:

```
docker pull nvcr.io/nvidia/tritonserver:23.02-py3
```

- Создайте локальную директорию для репозитория моделей и разместите туда подготовленную модель из задания 1.
- Запустите контейнер Triton, смонтировав репозиторий моделей:

```
docker run --gpus=all --rm -p8000:8000 -p8001:8001 -p8002:8002 \
  -v /path/to/model_repository:/models \
  nvcr.io/nvidia/tritonserver:23.02-py3 tritonserver --model-
repository=/models
```

- Проверьте работоспособность сервера, обратившись к эндпоинту `/v2/health/ready` или `/v2/health/live`.

---

### 3. Задание 3. Тестирование инференса через REST и gRPC API

- Напишите простой скрипт на Python, который:
  - Отправляет POST-запрос с данными для инференса через REST API Triton.
  - Принимает и анализирует ответ сервера.
- Используйте входные данные, соответствующие конфигурации вашей модели.
- (Опционально) Реализуйте аналогичный клиент для gRPC API, используя библиотеку `grpc` и предоставленные Triton протоколы.
- Сравните результаты инференса, полученные через REST и gRPC, и обсудите их.

---

### 4. Задание 4. Работа с моделью в формате ONNX

- Выберите или создайте модель в PyTorch и экспортируйте её в формат ONNX:

```
import torch
from my_model import MyModel

model = MyModel()
model.eval()
dummy_input = torch.randn(1, 3, 224, 224)
torch.onnx.export(model, dummy_input, "my_model.onnx", opset_version=11)
```

- Поместите полученный файл **my_model.onnx** в соответствующую директорию модели (например, `model_repository/onnx_model/1/`).
- Создайте файл **config.pbtxt** для ONNX модели с использованием платформы `"onnxruntime_onnx"`, указав:
  - Имя модели.
  - Максимальный размер пакета.
  - Описание входов и выходов.
- Запустите Triton Inference Server с новым репозиторием и протестируйте инференс для модели ONNX.

---

### 5. Задание 5. Работа с оптимизированными моделями с использованием TensorRT

- Возьмите ту же модель (или другую) и оптимизируйте её с помощью TensorRT, получив план (файл с расширением `.plan`). Для этого можно воспользоваться инструментом `trtexec`, например:

```
trtexec --onnx=my_model.onnx --saveEngine=my_model.plan --fp16
```

- Разместите полученный файл **my_model.plan** в директорию модели (например, `model_repository/trt_model/1/`).
- Создайте файл **config.pbtxt** для модели TensorRT, указав:
  - Имя модели.
  - Платформу `"tensorrt_plan"`.

    - Максимальный размер пакета.
    - Описание входных и выходных данных.
- Разверните Triton Inference Server с моделью TensorRT и проведите инференс, сравнив показатели задержки и пропускную способность с моделью в формате ONNX.