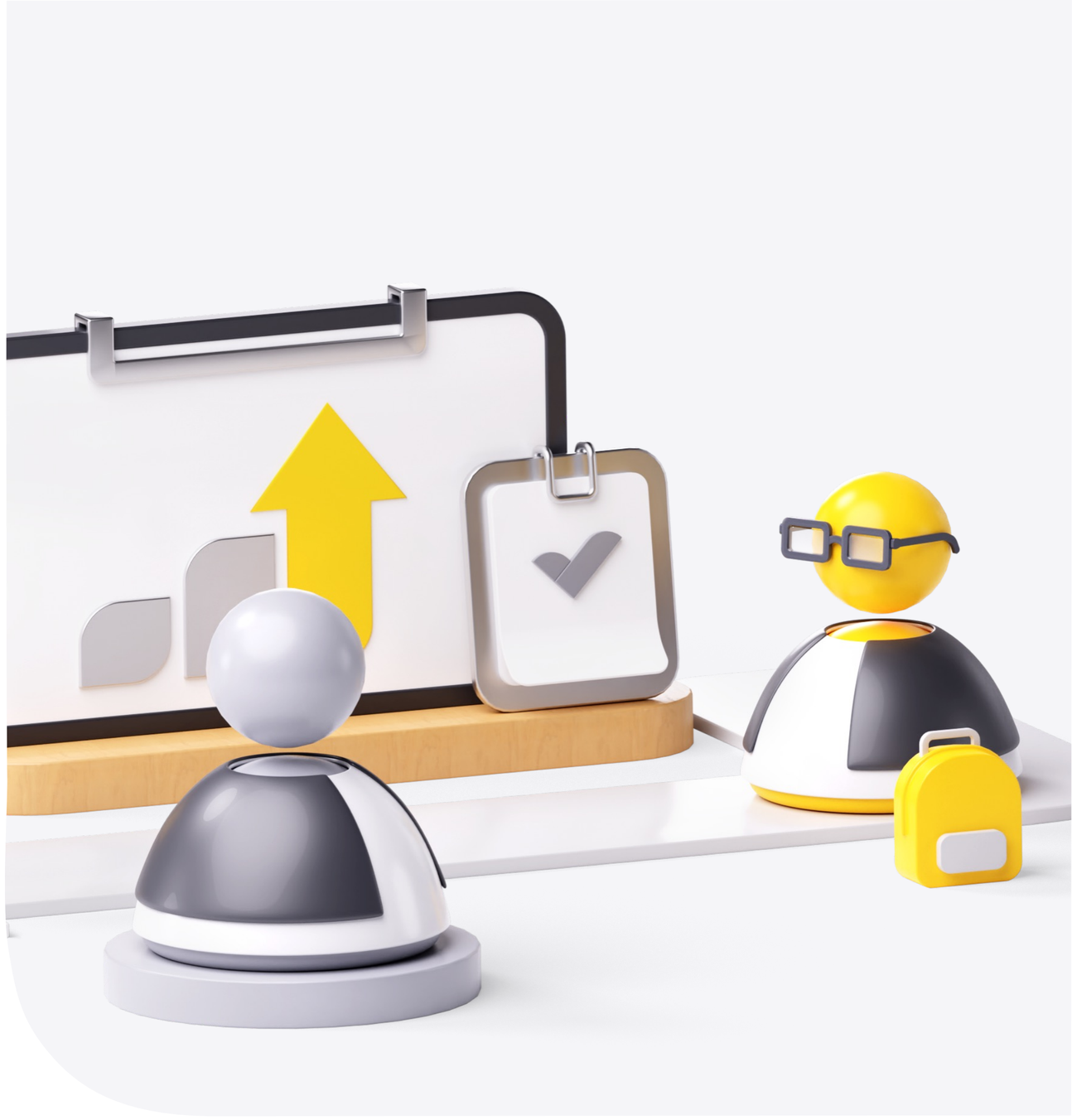
ТИНЬКОФФ

Model serving. Какой выбрать?

Егор Шестопалов



Егор Шестопалов

Работаю Computer Vision инженером в Тинькофф

4 года пилил сетки
в медицинском стартапе

Вывел не один проект в прод

Зачем этот доклад?

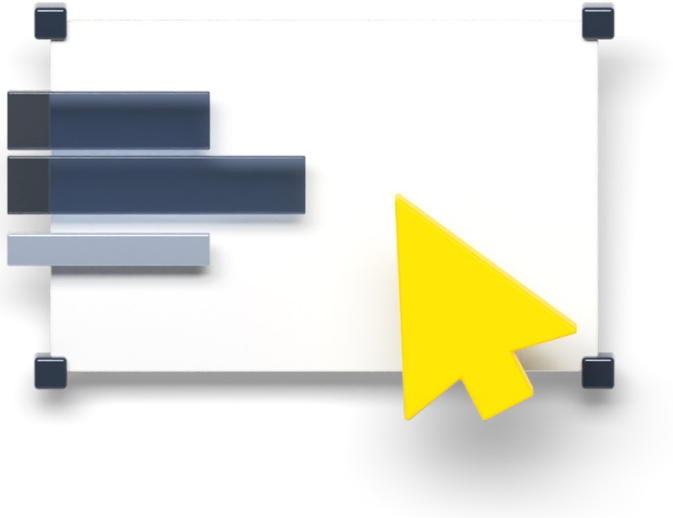
Помочь ответить на вопрос:

«Какой фреймворк для сервига выбрать?»

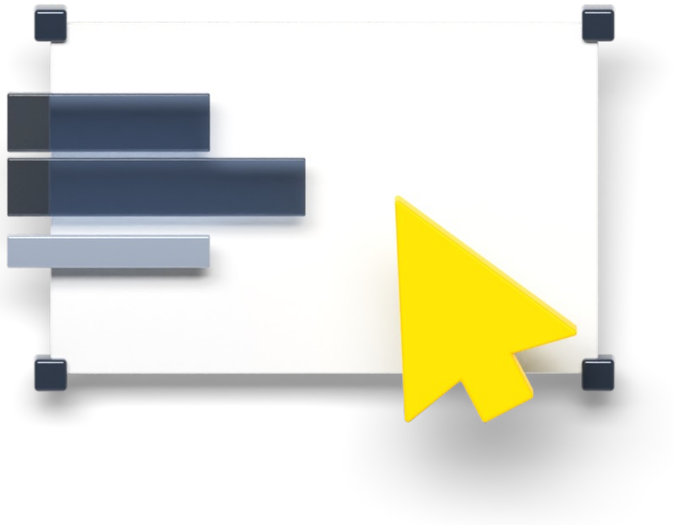
Зачем этот доклад?

Что будет?

- Что такое и зачем нужен serving engine?



Что будет?



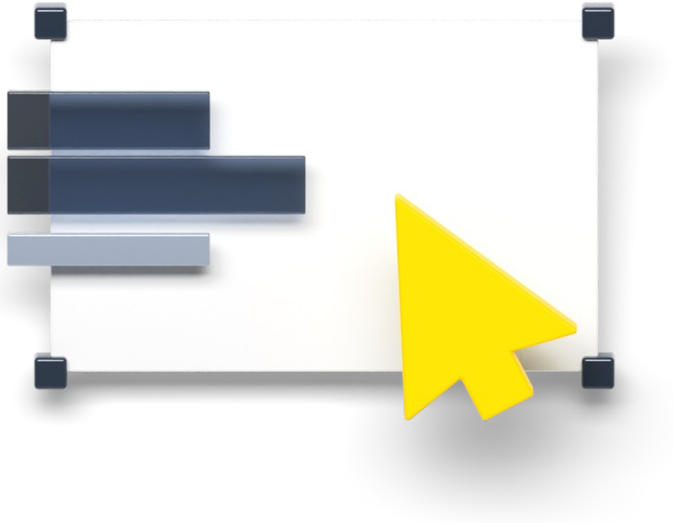
Что такое и зачем нужен serving engine?

Сравним фреймворки

- Парадигмы фреймворков
- Поддерживаемые форматы моделей
- Объединение моделей
- Деплой в k8s
- Сетевое взаимодействие
- Управление моделями
- Метрики
- Производительность
- Порог входа

Что будет?

- Что такое и зачем нужен serving engine?
- Сравним фреймворки
- Какой выбрать?



ТИНЬКОФФ

Что такое Serving Engine?

Зачем?

Serving Engine —

это технология, которая крутит модели, чтобы их можно было использовать

Зачем?

Serving Engine — это технология, которая крутит модели, чтобы их можно было использовать

Но зачем мне Serving Engine, когда есть я, могу сделать сам?

Зачем?

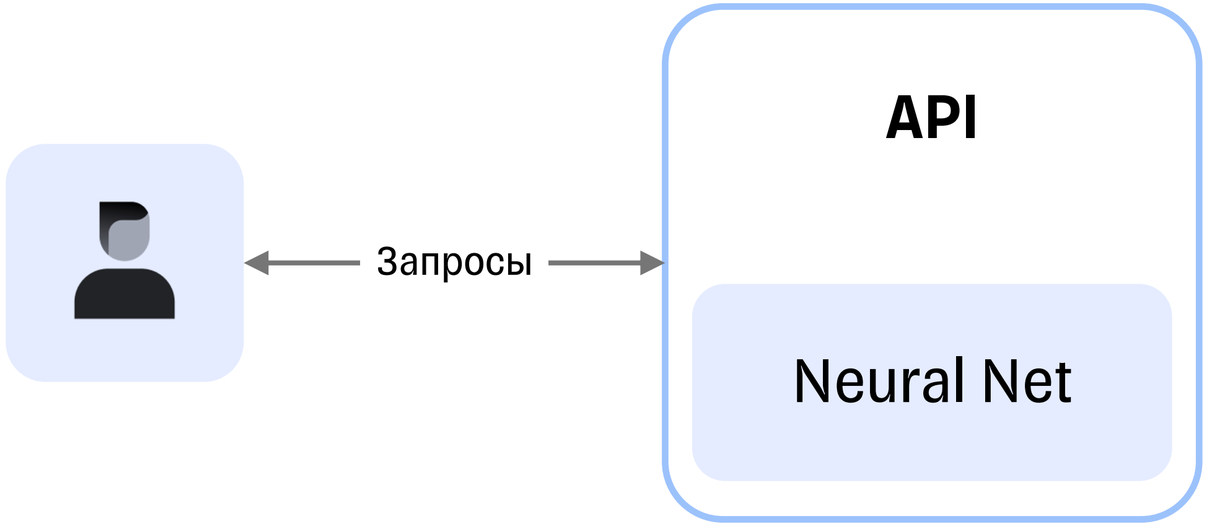
Serving Engine — это технология, которая крутит модели, чтобы их можно было использовать

Но зачем мне Serving Engine, когда есть я, могу сделать сам?

Но есть нюанс

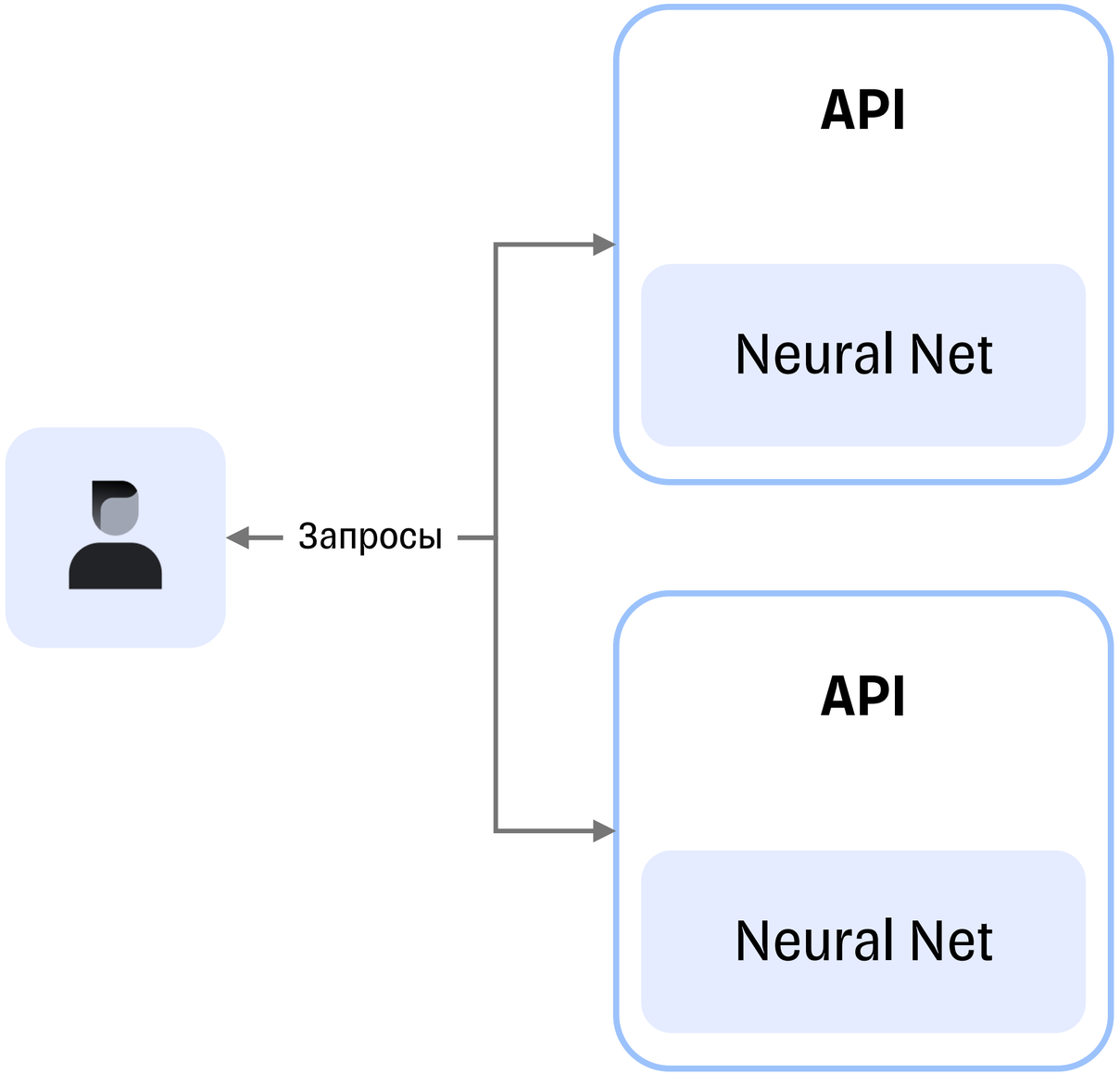
Детектируем номера авто

Добавил нейронку непосредственно в API, обернул в докер



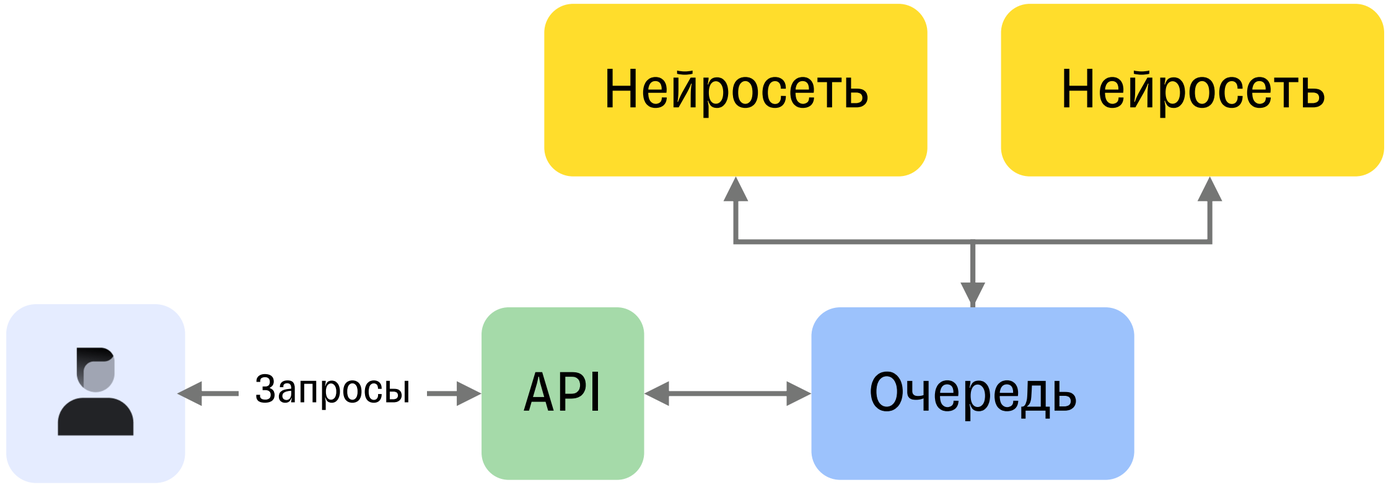
Возрос трафик

Сделал несколько копий



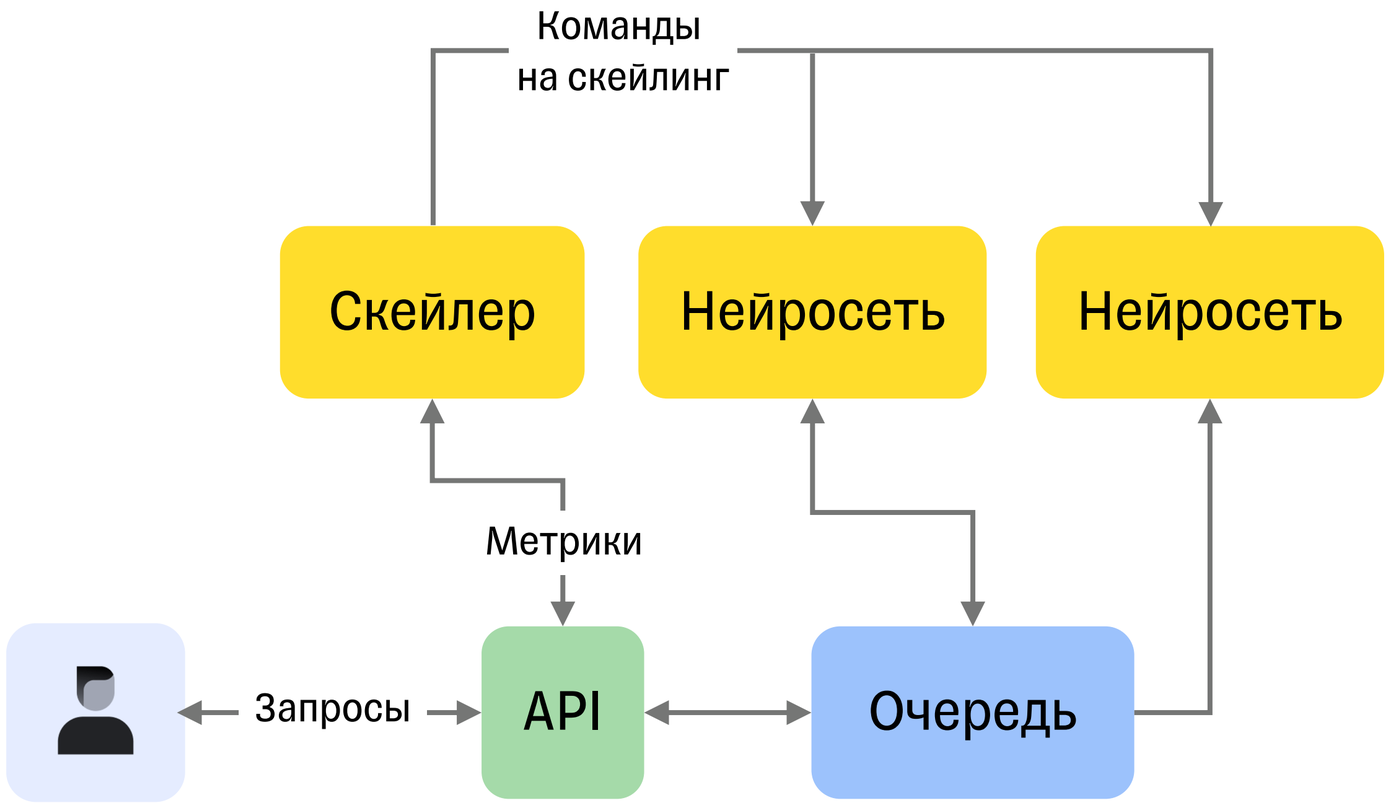
Неэффективно скейлим

Разнес нейронку с апи и сделал общение через очередь



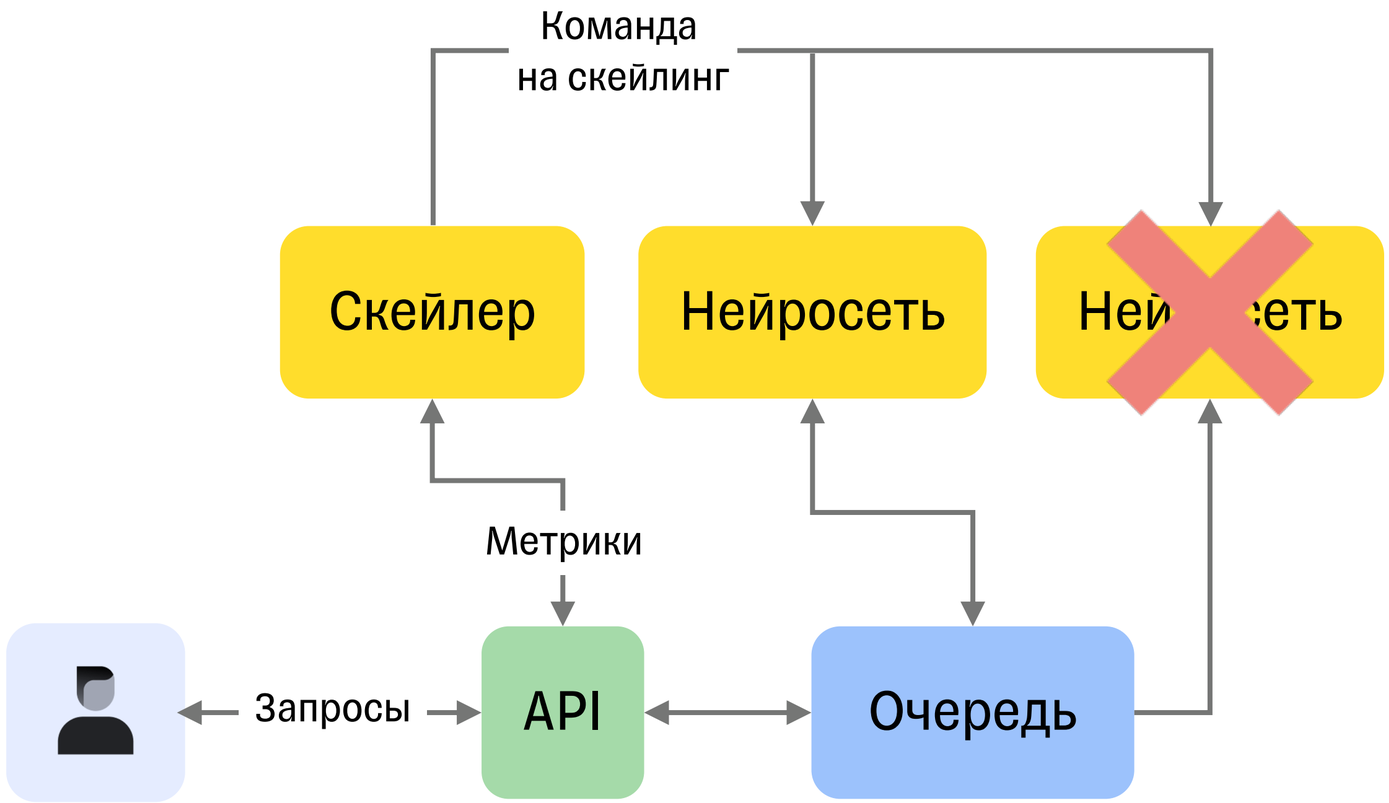
Трафик меняется в течение дня

Добавил метрики и настроил изменение количества реплик



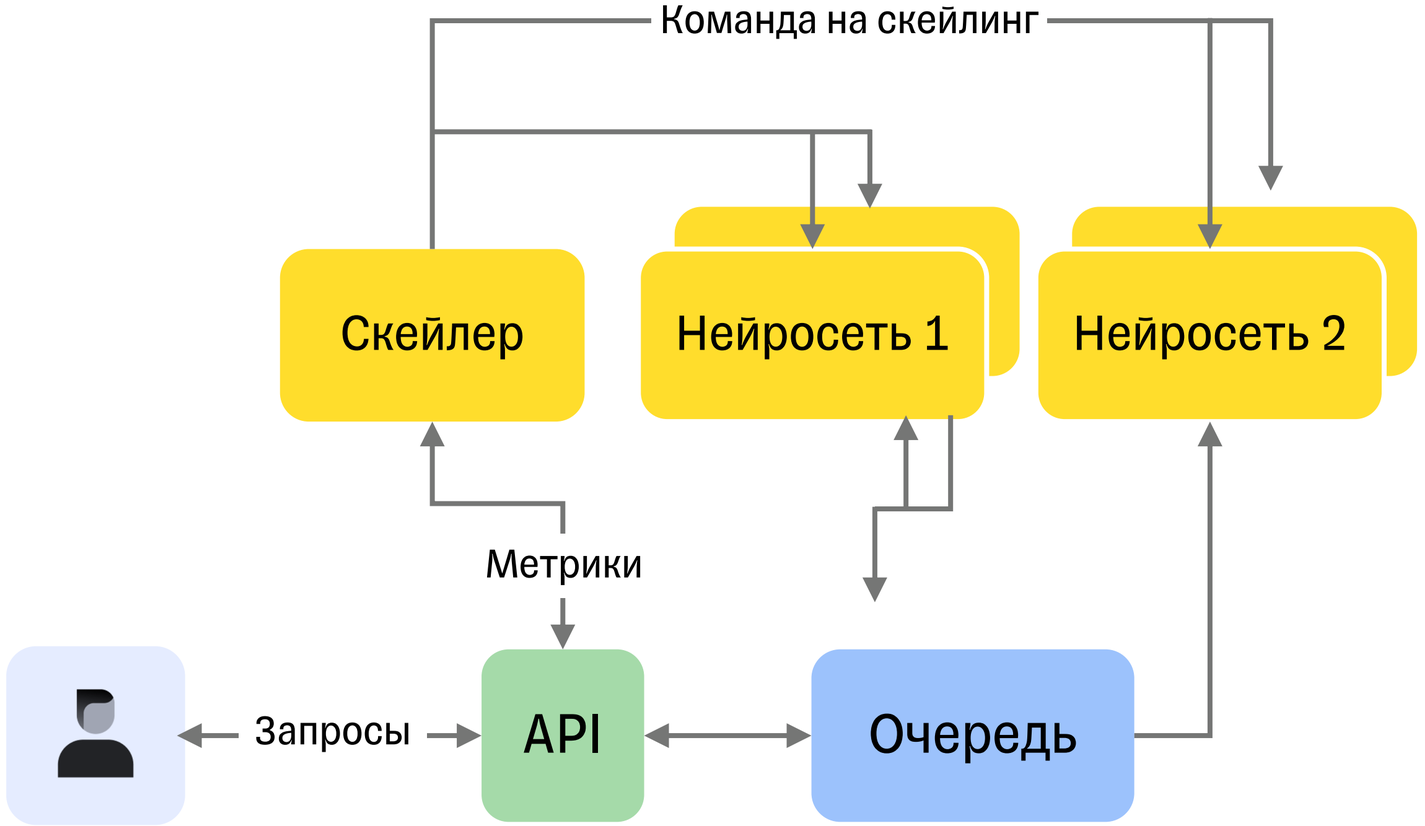
Во время обработки может что-то упасть

Добавил обработку падений



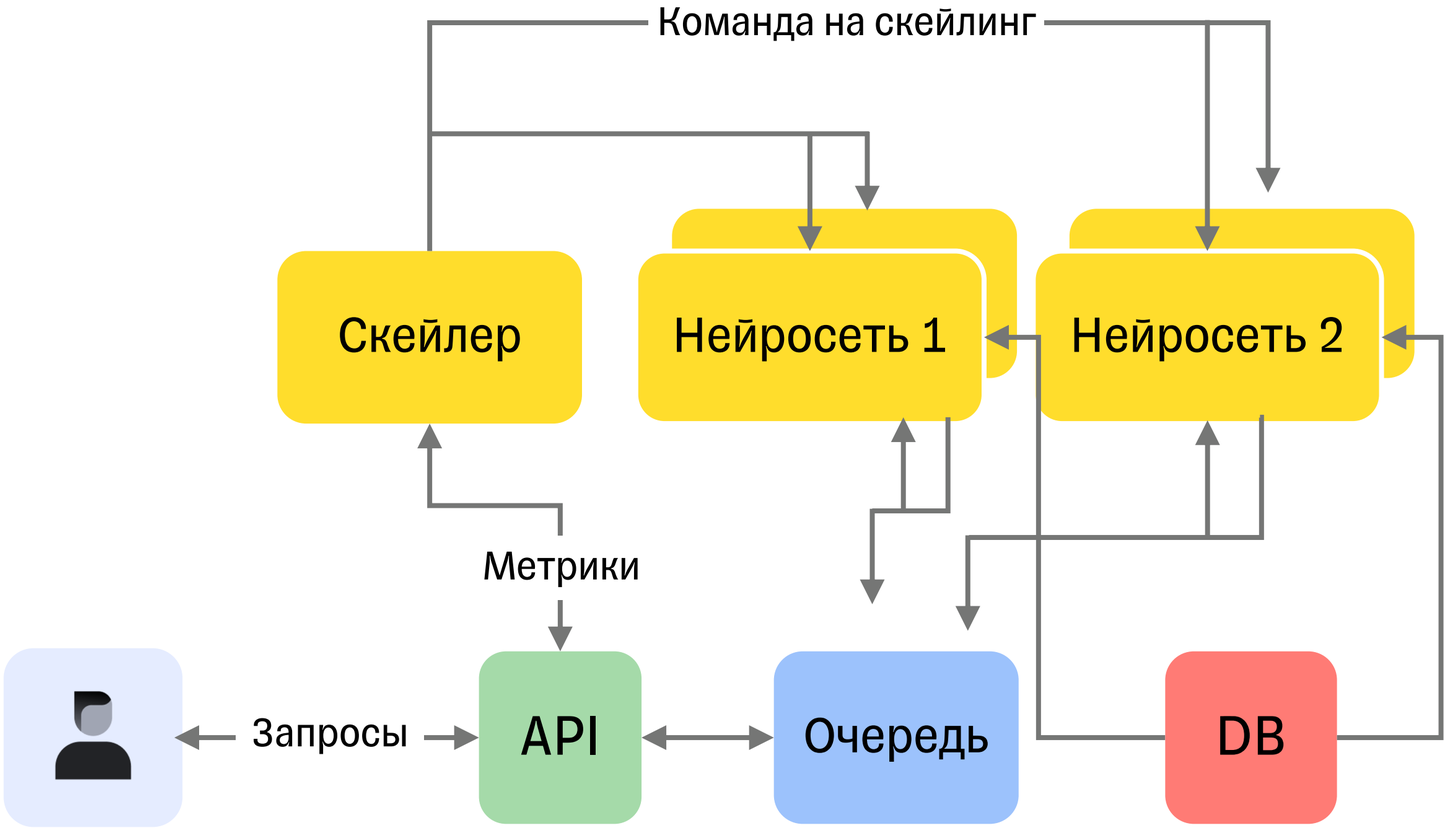
Нужно больше нейронок

Добавил еще нейронки. Связал их через очередь



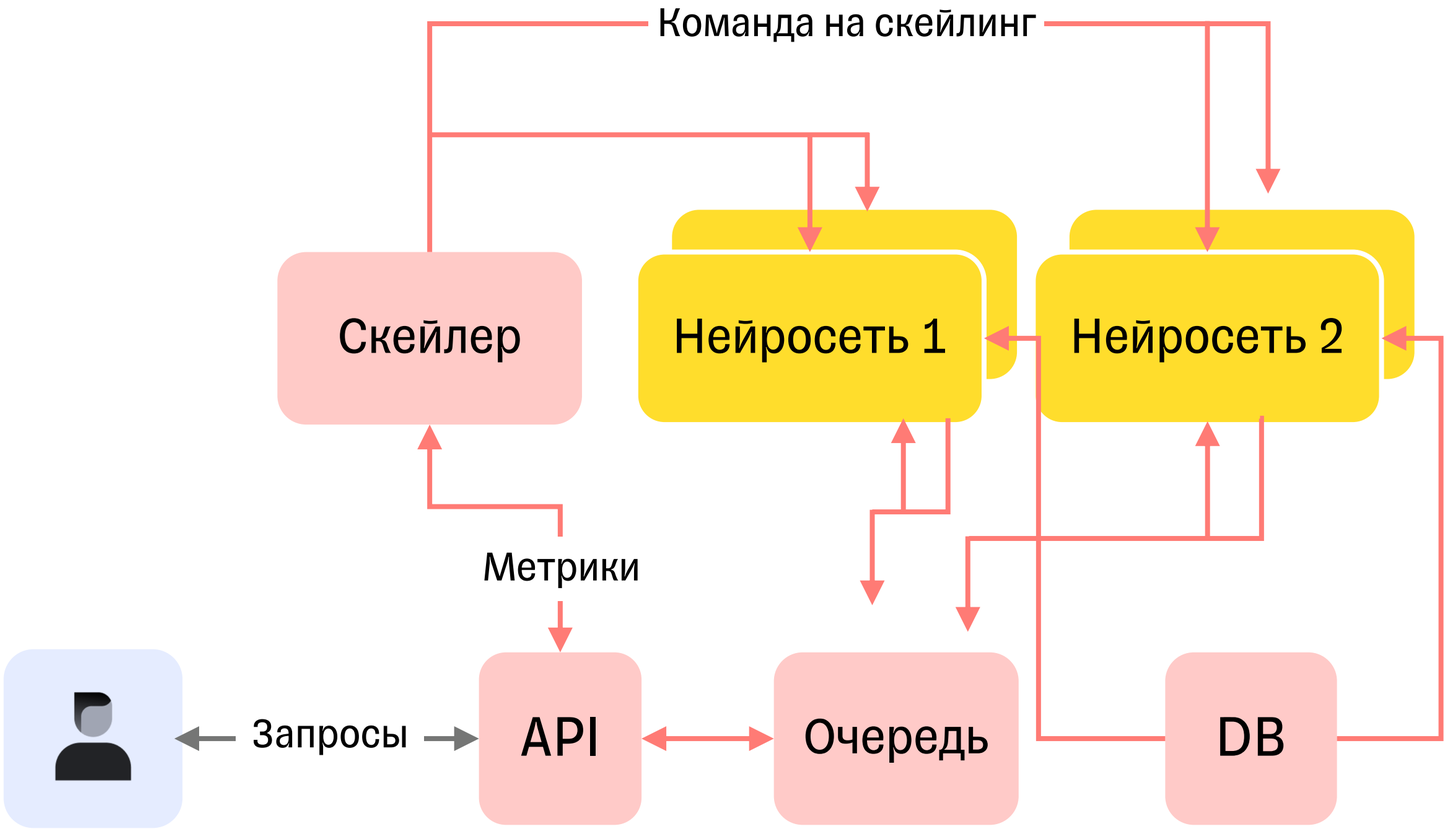
Нужно передавать большие ответы

Добавил передачу через внешнюю БД



Секундочку...

Внимательно присмотрелся
к проделанной работе...



Секундочку...

Внимательно присмотрелся
к проделанной работе...

Зачем?

- Serving Engine — это штука, которая крутит модели, чтобы мы могли ими пользоваться

Зачем?

- Serving Engine — это штука, которая крутит модели, чтобы мы могли ими пользоваться

- Serving Engine — это технология, которая экономит время, ведь вам не нужно изобретать велосипед

Зачем

Типичный Serving Engine:

- Крутит модели - запросы и отдает ответы
- Объединяет запросы в батчи
- Отдает метрики
- Не ломается от ошибок в моделях
- Умеет выгружать текущие и подгружать новые модели
- Другое

ТИНЬКОФФ

Что и как сравнивал

Что сравнивал

Nvidia Triton Inference Server

BentoML

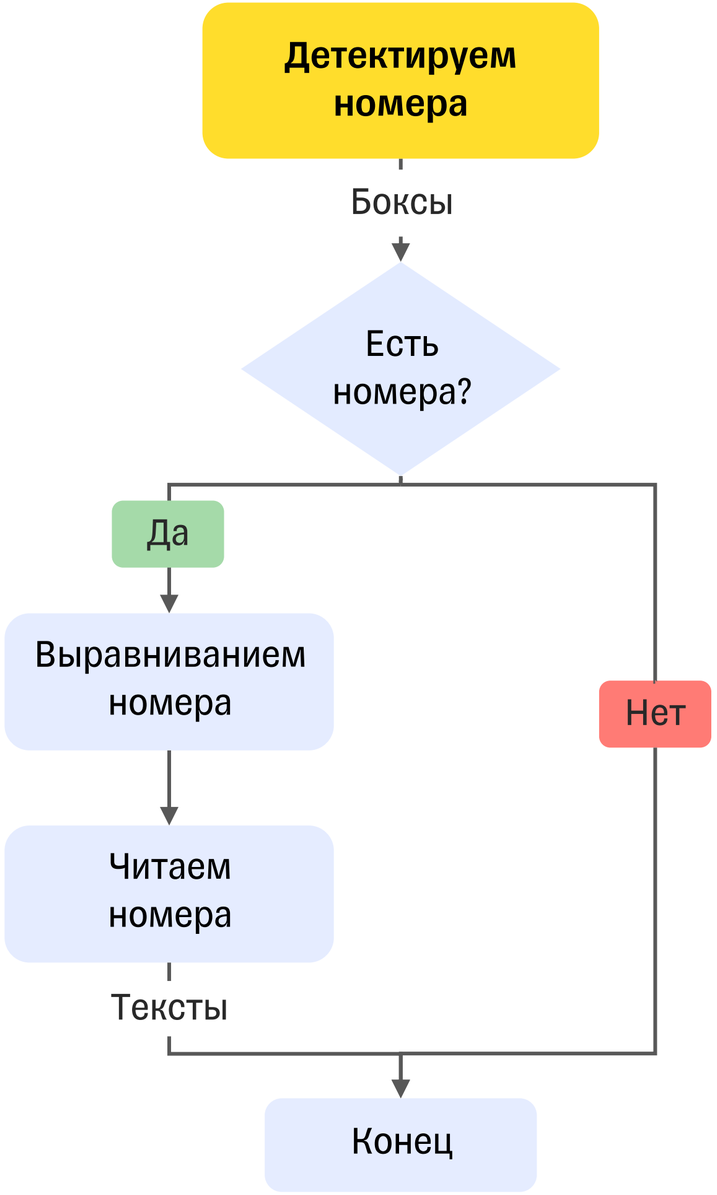
Torch Serve

Ray Serve

Как сравнивал

Реализовал руками одну и ту же задачу

- Несколько моделей
- Условие в логике
- Одна из моделей не сконвертирована — python модель



ТИНЬКОФФ

Парадигмы фреймворков

Парадигма TorchServe

Фреймворк для сервинга моделей в экосистеме PyTorch

Парадигма TorchServe

Для чего сделан:

Сервинг на одной машине

Парадигма Triton

Фреймворк для сервинга моделей

Парадигма Triton

Для чего сделан:

Сервинг на одной машине

Парадигма BentoML

Чуть больше, чем фреймворк
для сервинга моделей
с прицелом на облака

Парадигма BentoML

Чуть больше, чем фреймворк
для сервинга моделей
с прицелом на облака

Yatai — фреймворк, чтобы
использовать BentoML в k8s

Парадигма BentoML

Парадигма BentoML

Для чего сделан:

Сервинг моделей в контейнерах

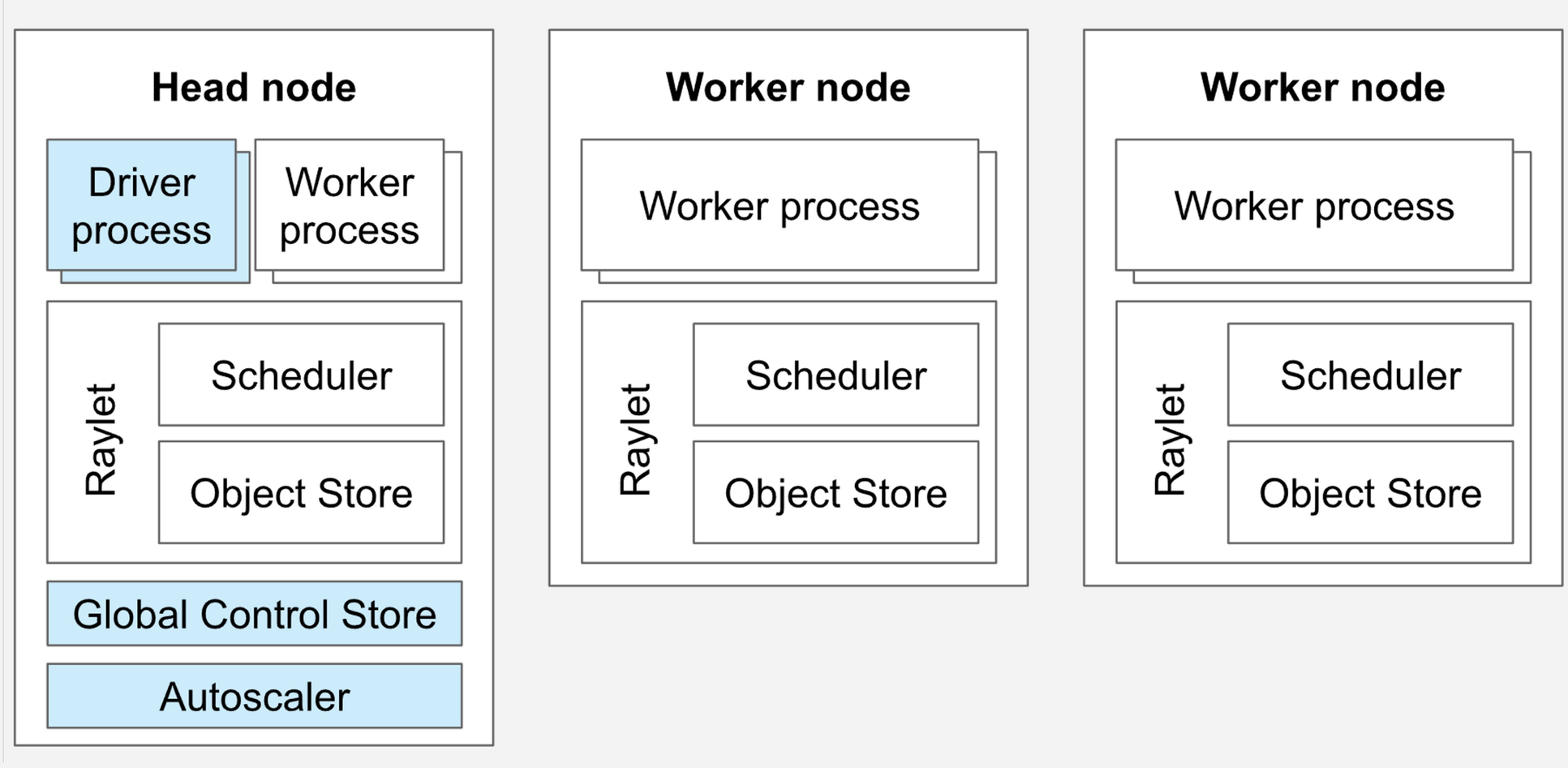
(и чуть-чуть еще)

Парадигма RayServe

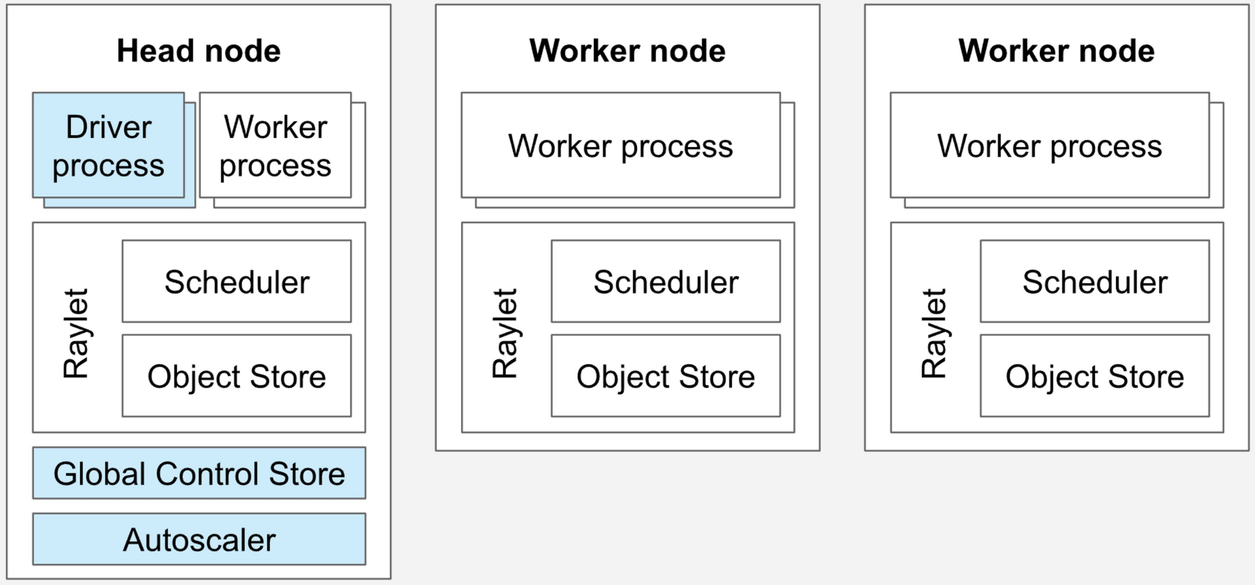
RayServe — фреймворк

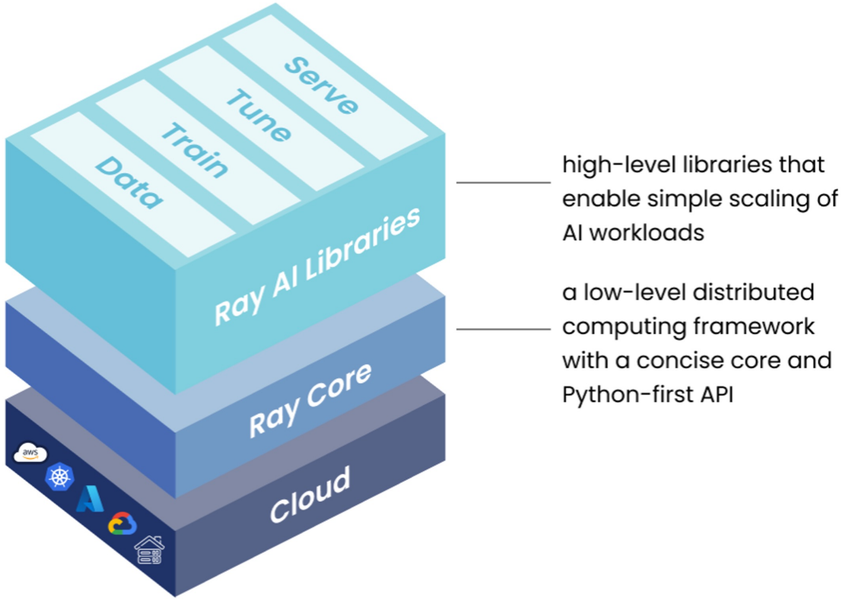
для сервинга на основе Ray

Парадигма RayServe



Парадигма RayServe



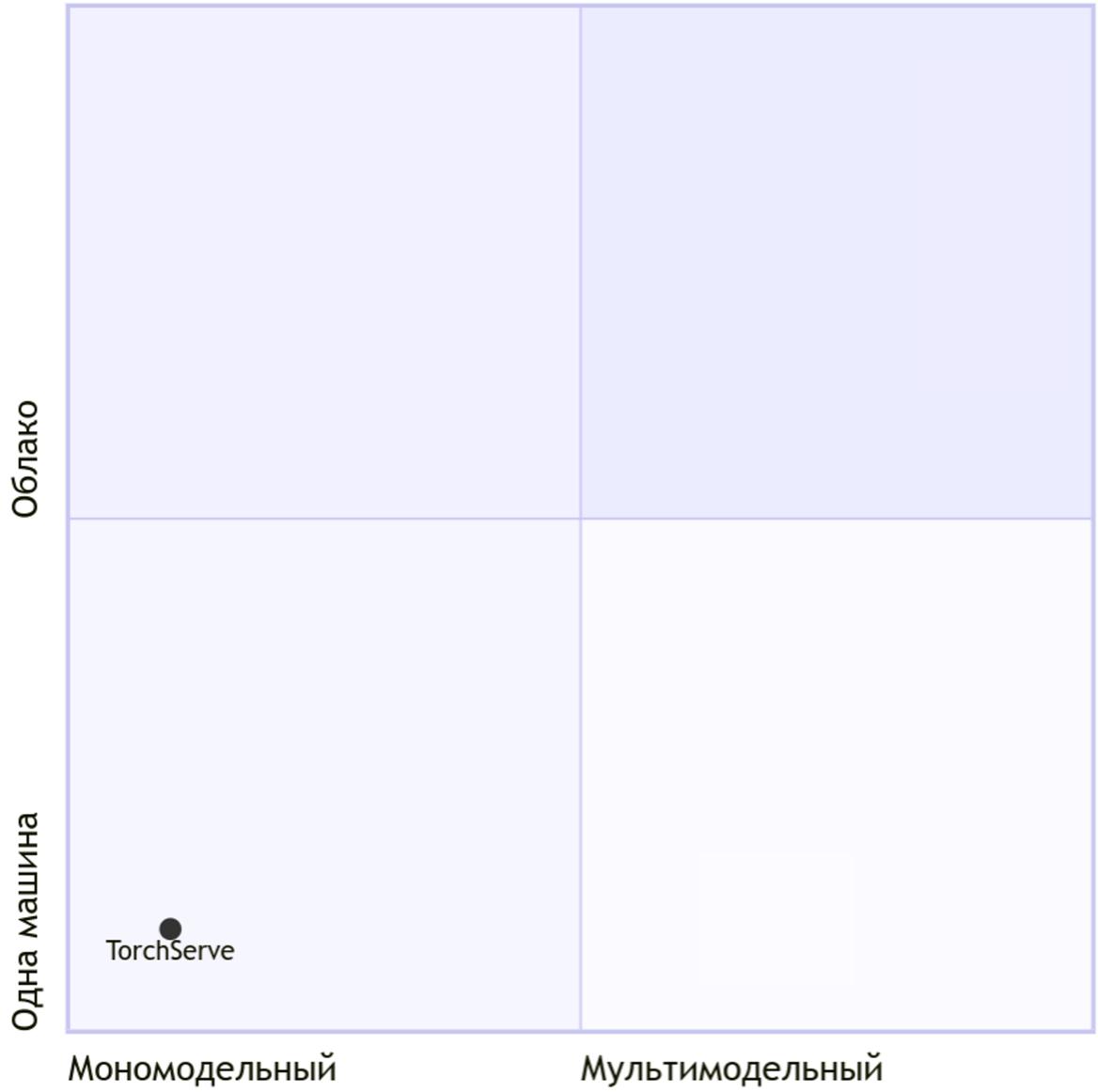


Парадигма RayServe

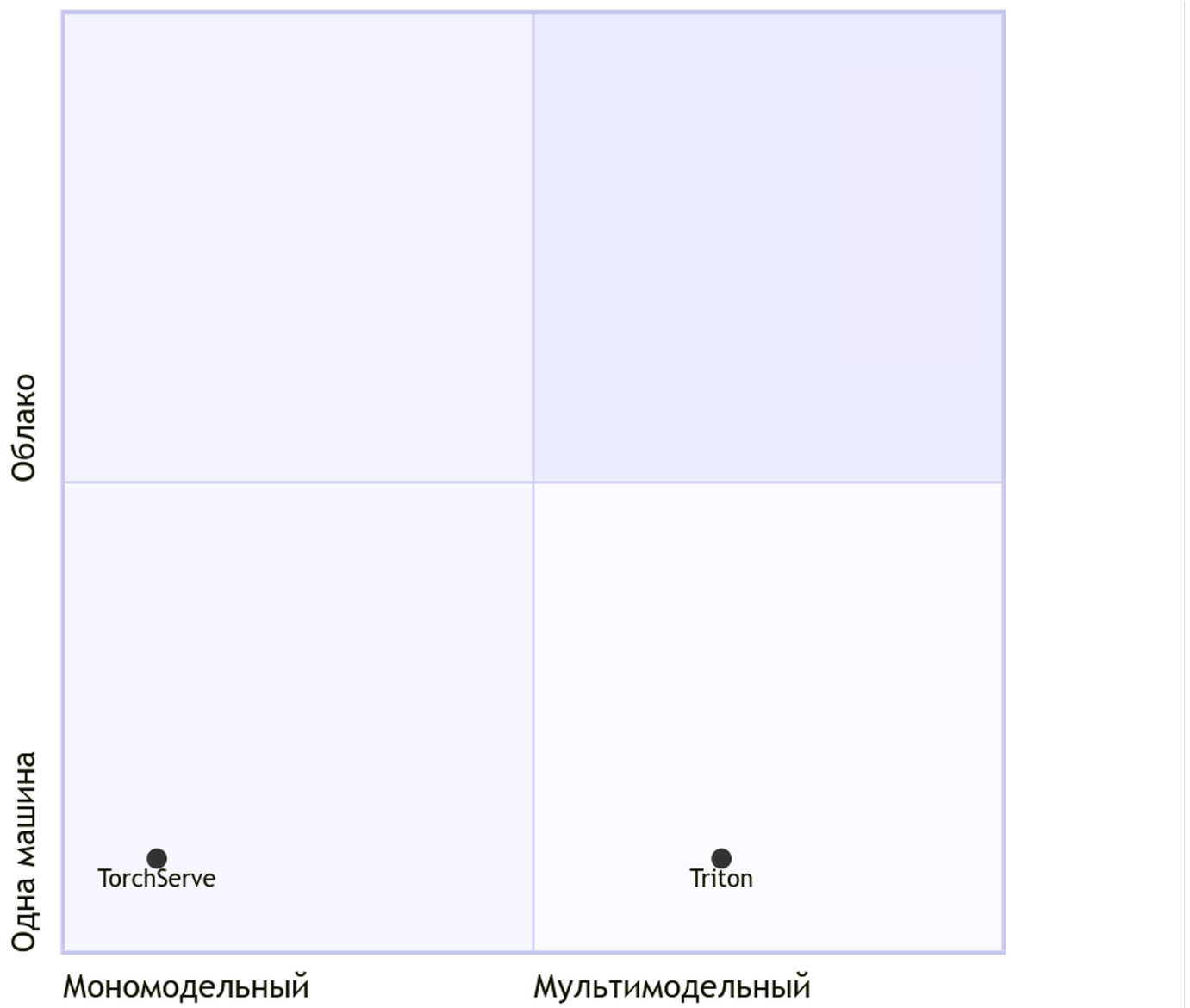
Для чего сделан:

Распределенного сервинга с автоскейлингом из коробки

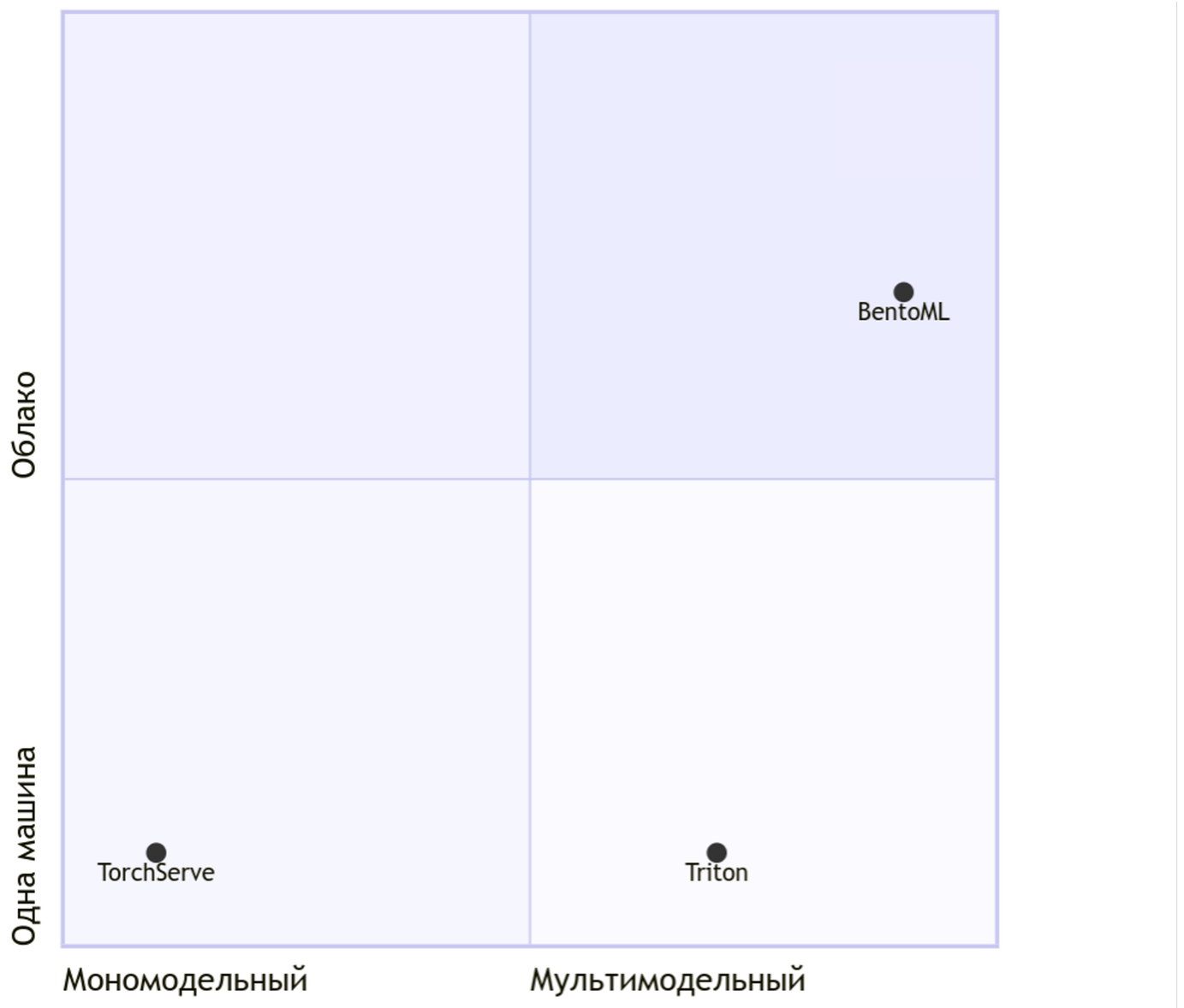
Парадигмы


Облако
Одна машина
TorchServe
Мономодельный
Мультимодельный

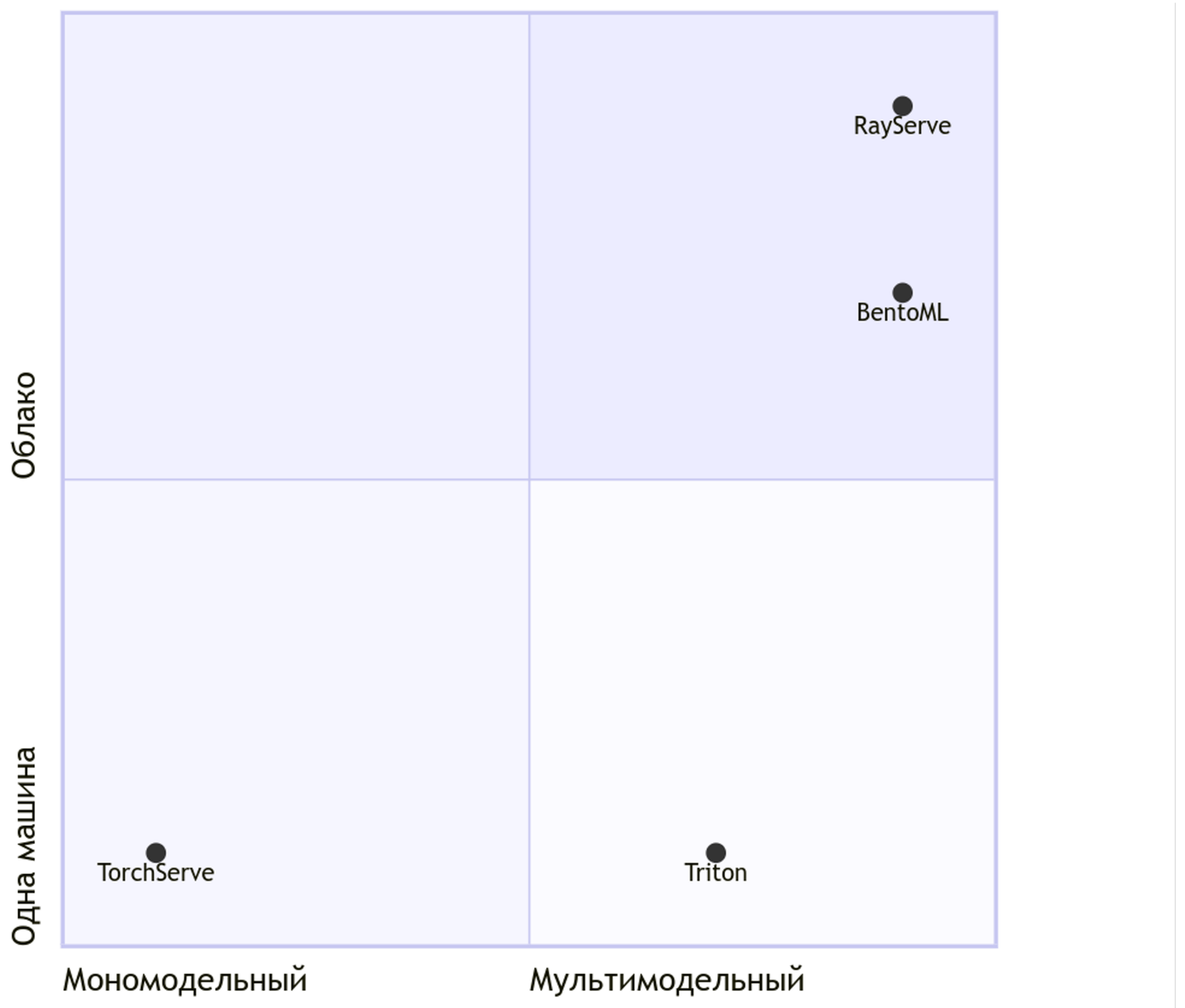
Парадигмы


Облако
Одна машина
TorchServe
Triton
Мономодельный
Мультимодельный

Парадигмы


Облако
Одна машина
BentoML
TorchServe
Triton
Мономодельный
Мультимодельный

Парадигмы


RayServe
BentoML
TorchServe
Triton
Облако
Одна машина
Мономодельный
Мультимодельный

ТИНЬКОФФ

Поддерживаемые форматы моделей

Форматы моделей. TorchServe

- TorchScript
- TensorRT
- ONNX
- Python

Форматы моделей. TorchServe

- TorchScript
- TensorRT
- Onnx

```python
from ts.torch_handler.base_handler import BaseHandler


class StnHandler(BaseHandler, ABC):

    def preprocess(self, data):
        ...
        return inputs

    def postprocess(self, data):
        ...
        return outputs
```

Форматы моделей. TorchServe

- Python

```python
from ts.torch_handler.base_handler import BaseHandler


class StnHandler(BaseHandler, ABC):

    def preprocess(self, data):
        return inputs

    def postprocess(self, data):
        return outputs

    def initialize(self, ctx):
        self.model = load_yolo(...)

    def inference(self, inputs:
        return self.model(inputs)
```

Форматы моделей. TorchServe

Выход — архив
с расширением .mar

Форматы моделей

Triton

TorchScript

TensorFlow

TensorRT

ONNX

OpenVINO

DALI

FIL

Python

Форматы моделей

Triton

TorchScript

TensorFlow

TensorRT

ONNX

OpenVINO

DALI

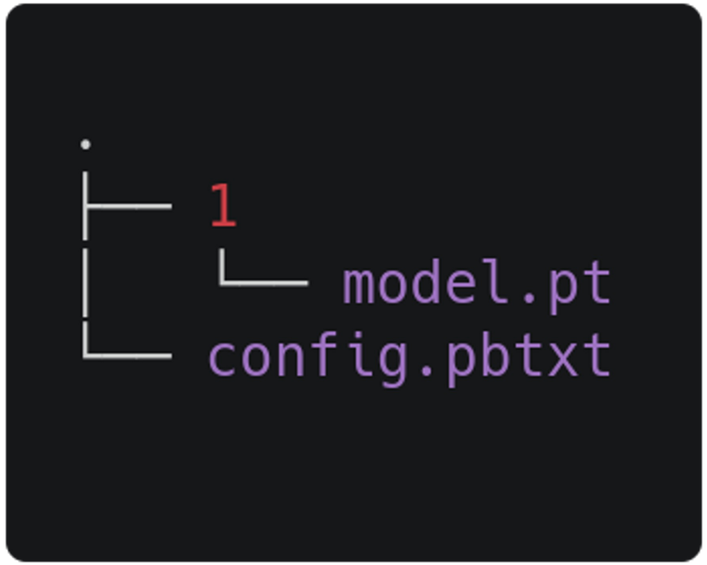
FIL 

Python

Форматы моделей

Triton

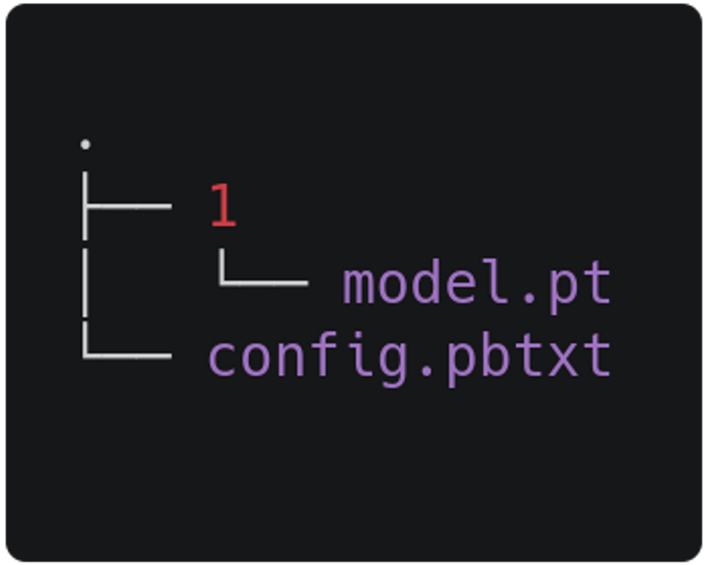
- TorchScript
- TensorFlow
- TensorRT
- ONNX
- OpenVINO
- DALI
- FIL



Форматы моделей

Triton

- TorchScript
- TensorFlow
- TensorRT
- ONNX
- OpenVINO
- DALI
- FIL

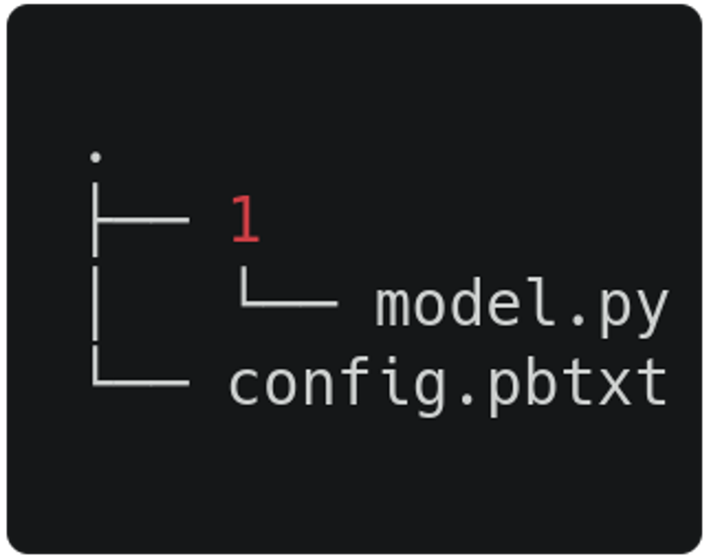



```
backend: "pytorch"
max_batch_size: 32
input [
    {
        name: "input__0"
        data_type: TYPE_FP32
        dims: [ 3, 24, 94 ]
    }
]
output [
    {
        name: "output__0"
        data_type: TYPE_FP32
        dims: [ 3, 24, 94 ]
    }
]
```

Форматы моделей

Triton

Python




```
backend: "python"
max_batch_size: 0
input [
    {
        name: "input__0"
        data_type: TYPE_UINT8
        dims: [ -1, -1, 3 ]
    }
]
output [
    {
        name: "output__0"
        data_type: TYPE_FP32
        dims: [ -1, 3, 24, 94 ]
    },
    {
        name: "output__1"
        data_type: TYPE_FP32
        dims: [ -1, 4 ]
    }
]
```

Форматы моделей

Triton

Python

```python
from nn.models import load_yolo

class TritonPythonModel:
    def initialize(self, args):
        self.model = load_yolo()

    def execute(self, requests):
        responses = []
        for request in requests:
            image = get_input_tensor_by_name(request, "input__0")

            detection = self.model(image)

            responses.append(detection)
        return responses
```

Форматы моделей

BentoML

- CatBoost
- fast.ai
- LightGBM
- ONNX
- PyTorch lightninig
- TensorFlow
- XGBoost
- EasyOCR
- Diffusers
- Keras
- PyTorch
- Scikit-Learn
- Transformers
- Detectron
- Python

Форматы моделей

BentoML

- CatBoost
- fast.ai
- LightGBM
- ONNX
- Lightninig
- TensorFlow
- XGBoost
- EasyOCR
- Diffusers
- Keras
- PyTorch
- Scikit-Learn
- Transformers
- Detectron
- Python

```
bentoml.pytorch.save_model("stn", model)
```

```
stn_runner = bentoml.pytorch.get("stn:latest").to_runner()
```

Форматы моделей

BentoML

- CatBoost
- fast.ai
- LightGBM
- ONNX
- Lightninig
- TensorFlow
- XGBoost
- EasyOCR
- Diffusers
- Keras
- PyTorch
- Scikit-Learn
- Transformers
- Detectron
- Python

```python
class YoloRunnable(bentoml.Runnable):
    SUPPORTED_RESOURCES = ("nvidia.com/gpu", "cpu")
    SUPPORTS_CPU_MULTI_THREADING = True

    def __init__(self, model_file):
        self.model = load_yolo()

    @bentoml.Runnable.method(batchable=False)
    def predict(self, inputs: np.ndarray):
        detection = self.model(image)
        return detection
```

```python
yolo_runner = bentoml.Runner(
    YoloRunnable,
    name="yolo"
)
```

Форматы моделей

BentoML

- CatBoost
- fast.ai
- LightGBM
- ONNX
- Lightninig
- TensorFlow
- XGBoost
- EasyOCR
- Diffusers
- Keras
- PyTorch
- Scikit-Learn
- Transformers
- Detectron
- Python

```
model: torch.ScriptModule = torch.jit.load(
    weight_file,
    map_location=device_id,
    _extra_files=_extra_files,
)
```

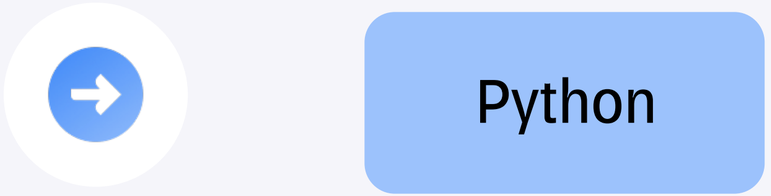
```
with tf.device(device_name):
    tf_model = tf.saved_model.load(bento_model.path)
    return tf_model
```

Форматы моделей

BentoML

- CatBoost
- fast.ai
- LightGBM
- ONNX
- Lightninig
- TensorFlow
- XGBoost

- EasyOCR
- Diffusers
- Keras
- PyTorch
- Scikit-Learn
- Transformers
- Detectron
- Python



Форматы моделей

Ray Serve

Python

```python
class YoloModel:
    def __init__(self):
        self.model = load_yolo()

    def predict(self, inputs: np.ndarray):
        detection = self.model()
        return detection
```

```python
YoloDeployment = serve.deployment(YoloModel, "yolo")
```

Форматы моделей

TorchServe	Triton	BentoML	RayServe
			

ТИНЬКОФФ

Объединение моделей

Объединение моделей

TorchServe

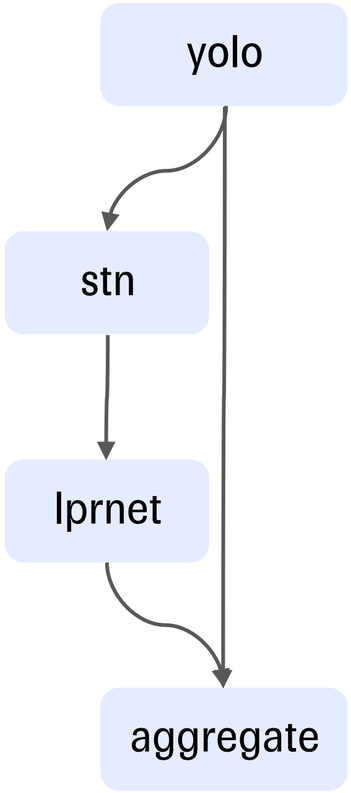
Workflow

Позволяет создавать цепочки и ветвления

Объединение моделей

TorchServe

```
dag:
  yolo: [stn, aggregate]
  stn: [lprnet]
  lprnet: [aggregate]
```



Объединение моделей

TorchServe

```
class StnHandler(BaseHandler, ABC):

    def handle(self, data, context):
        request = self.parse_request(data)

        if len(request["data"]) == 0:
            return [request]

        return super().handle(data, context)
```

Объединение моделей

Triton

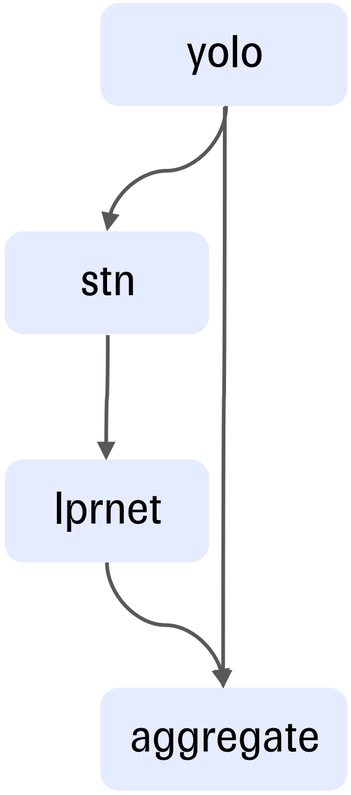
Ensembling

Позволяет создавать цепочки и ветвления

Объединение моделей

Triton

```
ensemble_scheduling {
  step [{
      model_name: "stn"
      input_map {
        key: "input__0"
        value: "yolo_output"
      }
      output_map {
        key: "output__0"
        value: "stn_output"
      }
    },
    {
      model_name: "lprnet"
      input_map {
        key: "input__0"
        value: "stn_output"
      }
      output_map {
        key: "output__0"
        value: "lprnet_output"
      }
    }
  ]
}
```



Объединение моделей

Triton

Ensembling

Позволяет создавать цепочки и ветвления

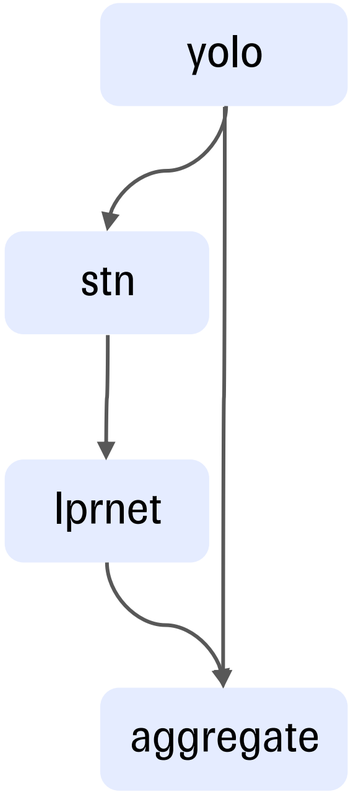
Business logic scripting (BLS)

Цепочки любой сложности

Объединение моделей

Triton

```python
class TritonPythonModel:
    def execute(self, requests):
        cropped_images, coordinates = self.predict(
            model_name="yolo"
        )

        if num_plates == 0:
            ...
            responses.append(inference_response)
            return responses

        plate_features = self.predict(
            model_name="stn",
        )
        text_features = self.predict(
            model_name="lprnet"
        )

        predicted_plates = postprocess(text_features)
        ...
        responses.append(inference_response)

        return responses
```



Объединение моделей

BentoML

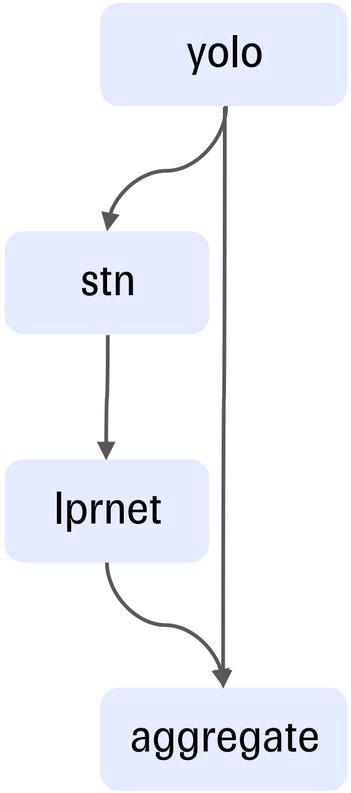
Python

Цепочки любой сложности

Объединение моделей

BentoML

```python
def recognize(
    inputs: np.ndarray,
    yolo_runner: YoloRunnable,
    stn_runner: Runnable,
    lprnet_runner: Runnable,
) -> PlatePrediction:
    plates, coordinates = yolo_runner.run(inputs)

    num_plates = plates.shape[0]
    if num_plates == 0:
        return PlatePrediction([], [])

    plate_features = stn_runner.run(plates)
    text_features = lprnet_runner.run(plate_features)

    predicted_plates = postprocess(text_features)

    return PlatePrediction(coordinates, predicted_plates)
```



Объединение моделей

Ray Serve

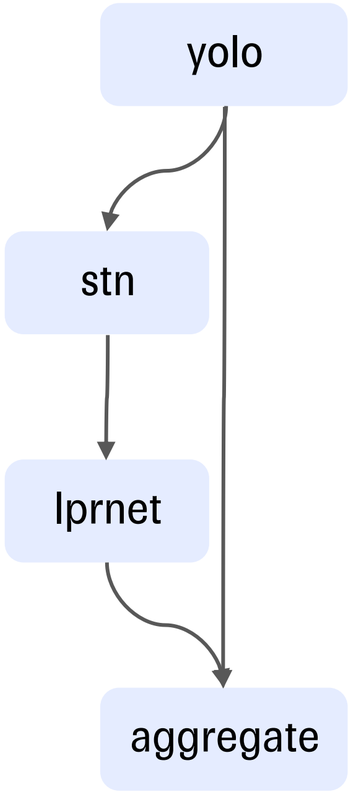
Python

Цепочки любой сложности

Объединение моделей

Ray Serve

```python
async def __call__(self, http_request):
    image = np.array(await http_request.json())

    ref = await self.yolo.predict.remote(image)
    plates, coordinates = await ref

    num_plates = plates.shape[0]
    if num_plates == 0:
        return PlatePrediction(coordinates=[], texts=[])

    ref = await self.stn.predict.remote(plates)
    plate_features = await ref

    ref = await self.lprnet.predict.remote(plate_features)
    text_features = await ref

    predicted_plates = postprocess(text_features)

    return PlatePrediction(coordinates, predicted_plates)
```



Объединение моделей

TorchServe	Triton	BentoML	RayServe
			

ТИНЬКОФФ

Деплой в k8s

Сетевое взаимодействие

Triton	TorchServe	BentoML	RayServe
Руками	Руками	Yatai	KubeRay

Объединение моделей

TorchServe	Triton	BentoML	RayServe
			

ТИНЬКОФФ

Сетевое взаимодействие

Сетевое взаимодействие

Triton	TorchServe	BentoML	RayServe
▪ REST ▪ gRPC	▪ REST ▪ gRPC	▪ REST ▪ gRPC	▪ REST ▪ gRPC

Сетевое взаимодействие

Triton

- REST
- gRPC

RayServe

- REST
- gRPC

Сетевое взаимодействие

TorchServe	Triton	BentoML	RayServe
			

ТИНЬКОФФ

Управление моделями

Управление моделями

TorchServe, Triton

Предоставляют Management API

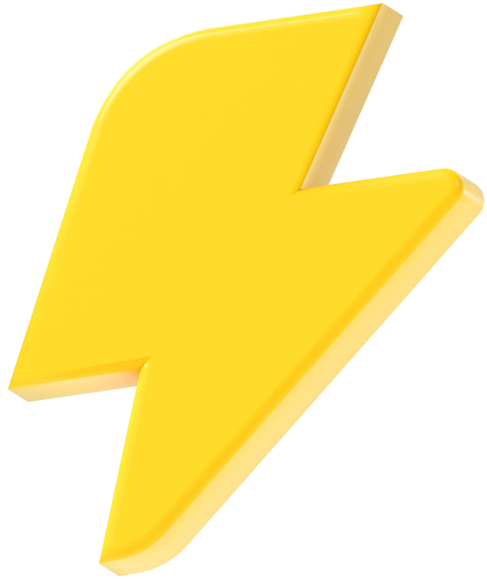
```
# register a model
curl -X POST  "http://localhost:8081/models?
url=https://torchserve.pytorch.org/mar_files/squeezenet1_1.mar"
```

```
# register a model
curl -X POST  "http://localhost:8081/v2/repository/models/${MODEL_NAME}/load"
```

Управление моделями

BentoML

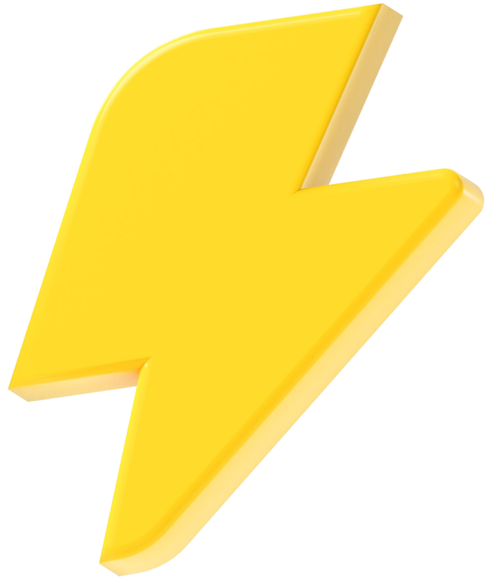
- Не имеет Managment API
- Управление передано в Docker, Yatai



Управление моделями

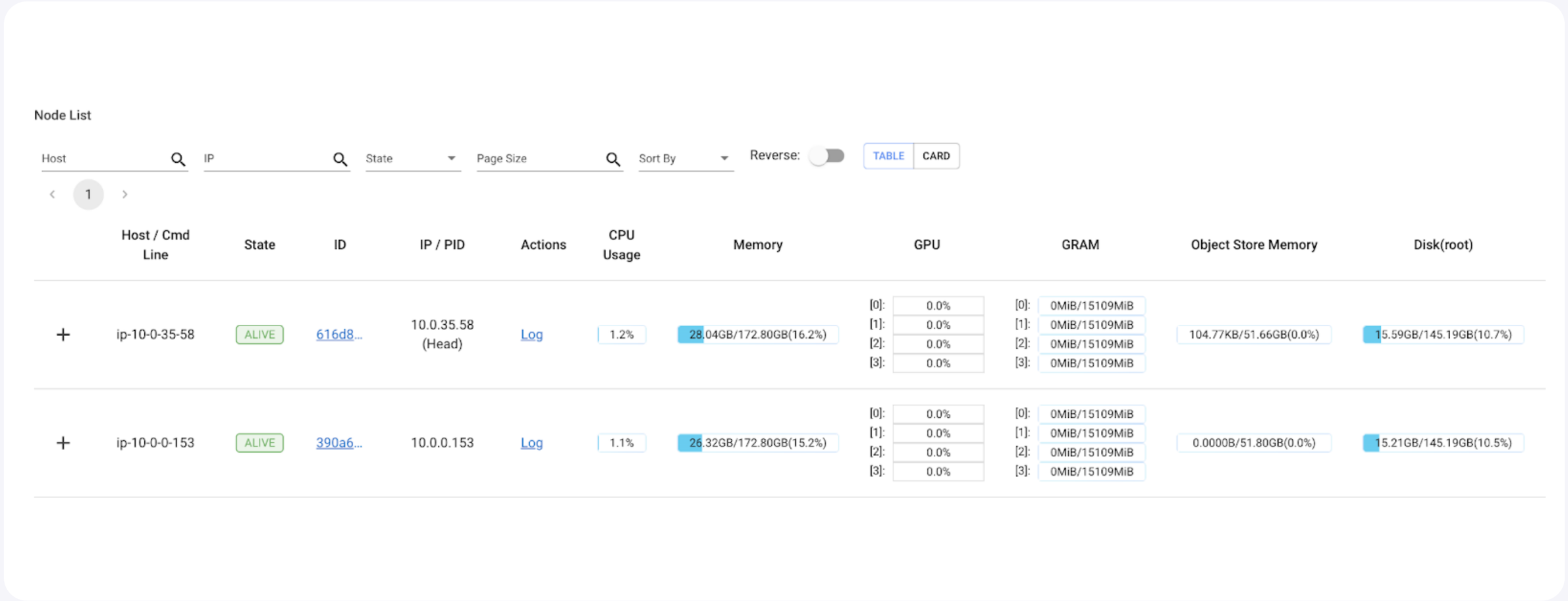
Ray Serve

- Управляется из дашборда
- Также имеет автоскейлинг из коробки



Управление моделями

Ray Serve



Управление моделями

TorchServe	Triton	BentoML	RayServe
			

ТИНЬКОФФ

Метрики

TorchServe

- Совместим с Prometheus

BentoML

- Совместим с Prometheus

Triton

- Совместим с Prometheus

Ray Serve

- Совместим с Prometheus

TorchServe

- Совместим … rometheus

BentoML

Triton

- Совместим … rometheus

TorchServe

- Совместим с Prometheus
- **Можно реализовать свои на python**

BentoML

- Совместим с Prometheus
- **Можно реализовать свои на python**

Triton

- Совместим с Prometheus
- **Можно реализовать свои на C**

Ray Serve

- Совместим с Prometheus
- **Можно реализовать свои на python**

Метрики

TorchServe	Triton	BentoML	RayServe
			

ТИНЬКОФФ

Производительность

Батчинг

Triton	TorchServe	BentoML	RayServe
Есть	Есть	Есть	Есть

Батчинг

Triton

Есть

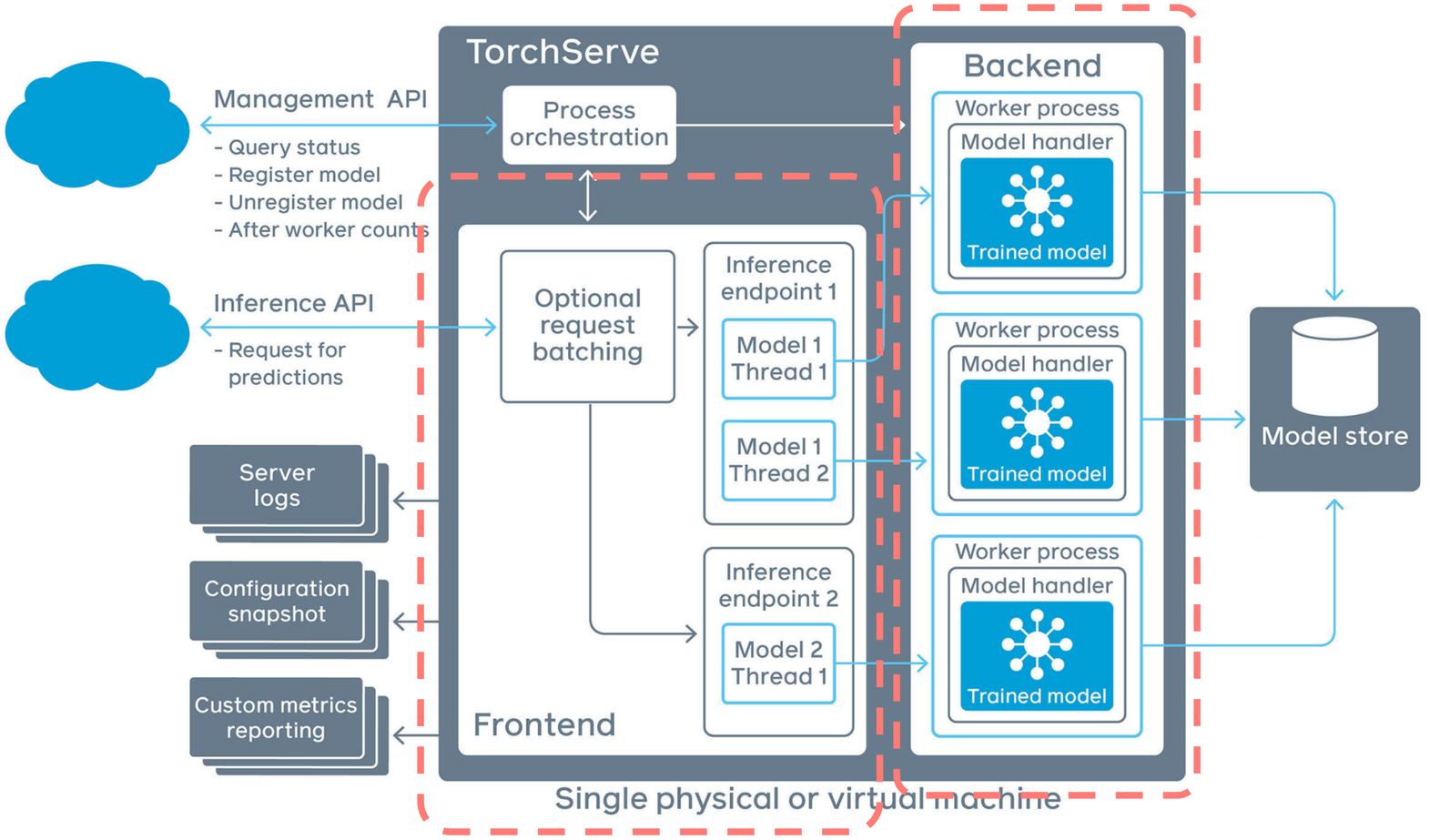
RayServe

Есть

TorchServe

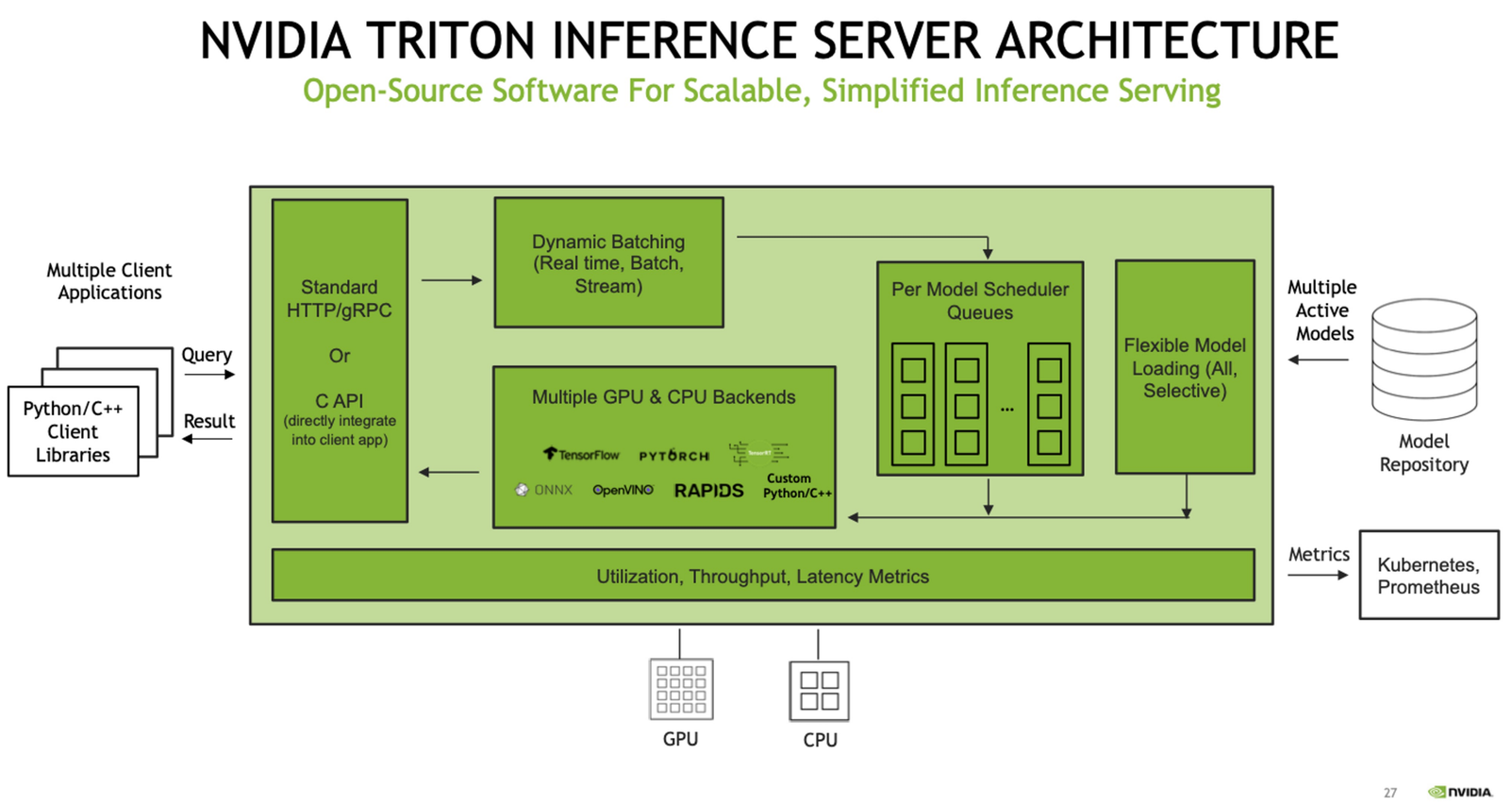
Java

Python



Triton

C++


NVIDIA TRITON INFERENCE SERVER ARCHITECTURE
Open-Source Software For Scalable, Simplified Inference Serving
Multiple Client Applications
Python/C++ Client Libraries
Query
Result
Standard HTTP/gRPC
Or
C API
(directly integrate into client app)
Dynamic Batching (Real time, Batch, Stream)
Multiple GPU & CPU Backends
TensorFlow
PYTORCH
ONNX
OpenVINO
RAPIDS
Custom Python/C++
Per Model Scheduler Queues
Flexible Model Loading (All, Selective)
Multiple Active Models
Model Repository
Utilization, Throughput, Latency Metrics
Metrics
Kubernetes, Prometheus
GPU
CPU
27
NVIDIA

Triton

Оптимизирует модели

Triton

Держит данные на GPU (dlpack)

```
from torch.utils.dlpack import from_dlpack
import triton_python_backend_utils as pb_utils

class TritonPythonModel:

  def execute(self, requests):
    ...
    input0 = pb_utils.get_input_tensor_by_name(request, "INPUT0")

    # We have converted a Python backend tensor to a PyTorch
    # tensor without making any copies.
    pytorch_tensor = from_dlpack(input0.to_dlpack())
```

Triton

Оптимизирует модели

Умеет кешировать ответы из коробки

Держит данные на GPU (dlpack)

Triton

Оптимизирует модели

Умеет кешировать ответы из коробки

Держит данные на GPU (dlpack)

Можно реализовать передачу данных через Shared Memory

BentoML

Python

Ray Serve

Python

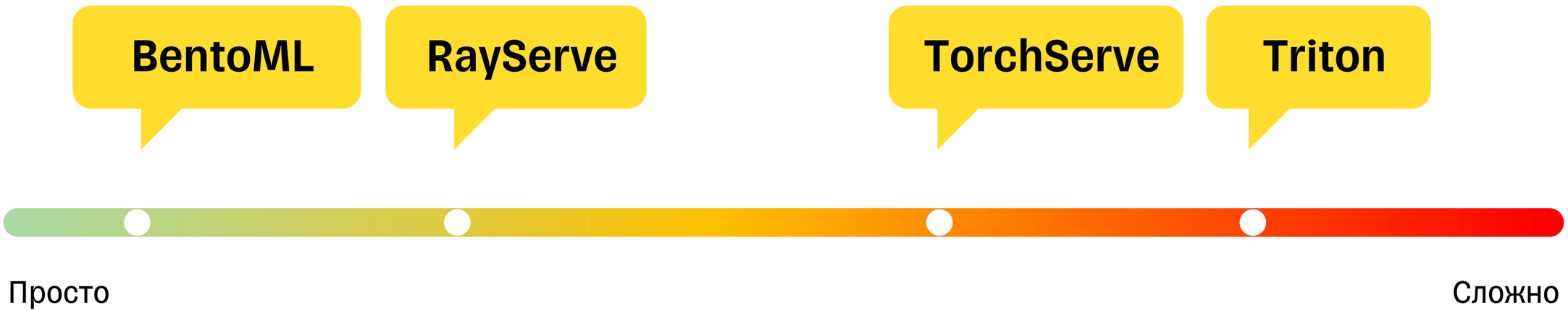
Производительность

TorchServe	Triton	BentoML	RayServe
			

ТИНЬКОФФ

Порог входа

Порог входа



Порог входа

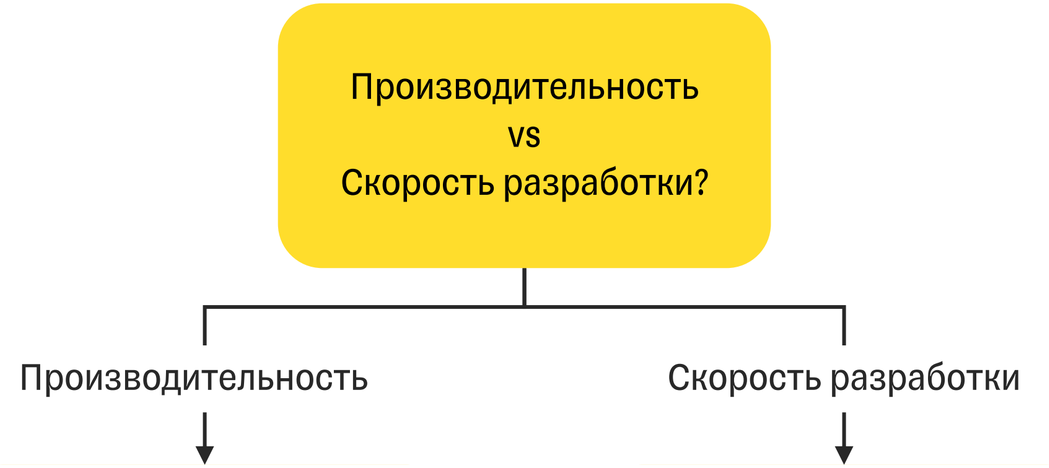
TorchServe	Triton	BentoML	RayServe
			

ТИНЬКОФФ

Что выбрать?

	TorchServe	Triton	BentoML	RayServe
Форматы моделей	🥲	😌	🤩	😌
Объединение моделей	😌	🤩	🤩	🤩
Деплой в k8s	😌	😌	🤩	🤩
Сетевое взаимодействие	🤩	🤩	🤩	😌
Управление моделями	🤩	🤩	🥲	😌
Метрики	🤩	😌	🤩	🤩
Производительность	😌	🤩	🥲	🥲
Порог входа	😌	🥲	🤩	🤩

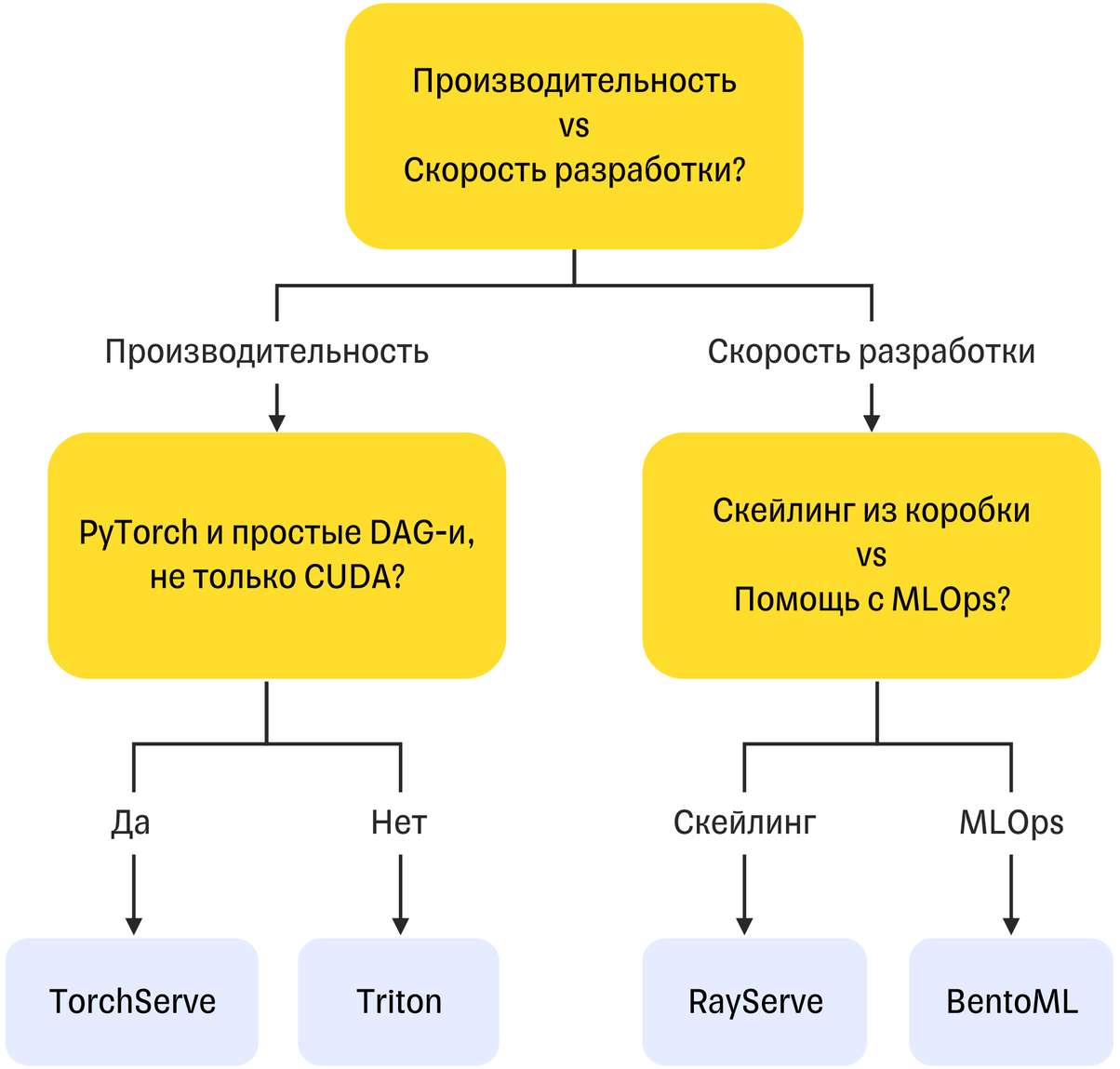
Что выбрать?


Производительность
vs
Скорость разработки?
Производительность
Скорость разработки

Что выбрать?


Производительность
vs
Скорость разработки?
Производительность
Скорость разработки
PyTorch и простые DAG-и,
не только CUDA?
Да
Нет
TorchServe
Triton

Что выбрать?


Производительность
vs
Скорость разработки?
Производительность
Скорость разработки
PyTorch и простые DAG-и,
не только CUDA?
Скейлинг из коробки
vs
Помощь с MLOps?
Да
Нет
Скейлинг
MLOps
TorchServe
Triton
RayServe
BentoML

Спасибо!

Код

Мой канал