Эффективные сервисы ML Inference нейросетей в Яндекс рекламе

Дмитрий Ульянин

Сентябрь 2024

Overview

1. О нас
2. Что такое Inference Server
3. Какую задачу мы решаем в рекламе
   - Наши масштабы
   - Open source и почему решили делать своё решение
4. Как устроен наш сервис
5. Оптимизации
   - Сплит архитектура моделей
   - Кэширование
   - Батчевание
   - GPU Inference
6. Итоги
7. Планы

Yandex

О нас

Yandex

Наша команда

Мы — команда разработки сервисов моделей в рекламе

- Отвечаем за внедрение ML моделей в продакшн рекламы:
  - Процесс внедрения
  - Сервисы применения нейросетей
- Делаем из нашего сервиса базовую технологию, чтобы
  - Внедрять модели быстрее в рекламе
  - Помогать в этом остальной части Яндекса

Yandex

Реклама

Реклама — огромный рекомендательный сервис.

Действующие лица:

- Пользователь
- Рекламодатель
- Площадка
- Яндекс

Задача отбора рекламы — максимизировать счастье каждого актора.

Yandex

Нейросети в ML в сервисах Яндекса

- Кандидатогенерация:
  - Отбирает релевантные пользователю и запросу баннера
  - Например, KNN индекс баннеров (HNSW)
  - Фильтрация
- Двухстадийное ранжирование баннеров:
  - Фильтрация на нижней стадии
  - Финальное ранжирование (кликовые, конверсионные прогнозы)
- Автобюджет:
  Расчёт ставок для эффективных трат рекламной компании в определённый срок
- Предсказание стоимости показа баннера на определённой позиции
- Реклама в телеграмме
- etc.

Yandex

Что такое Inference Server

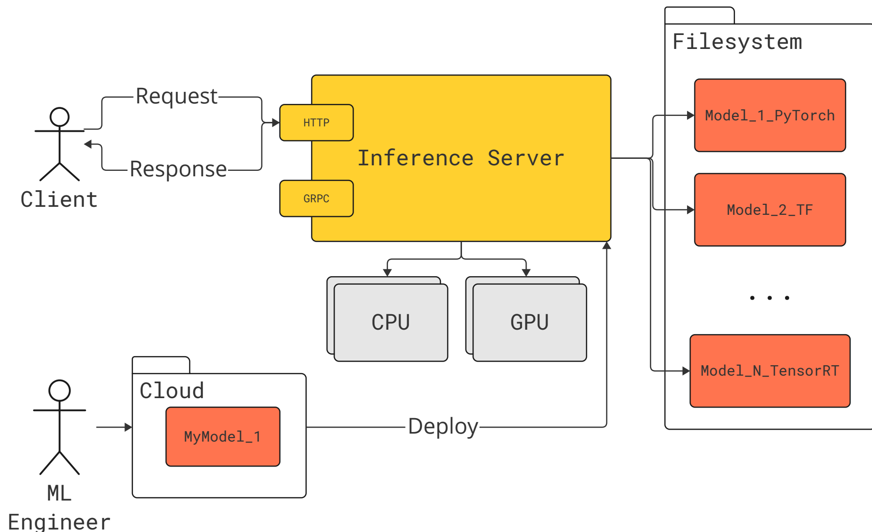
Yandex

Глоссарий

- **Inference** ML модели/нейросети — применение на конкретных входах/запросе
- **Вход на Inference** модели — именованный набор тензоров
- **Inference Server** — сервис с некоторым API, который позволяет деплоить ML модели и отвечать на запросы Inference от пользователя
- **Деплой ML модели**:
  - Развернуть новый Inference сервис в облаке с конкретной моделью
  - Выкатить модель под уже существующий Inference сервис
- **Фреймворк инференса / Inference Backend** — формат модели и код применения.
  Примеры: PyTorch/TensorFlow/ONNX/TensorRT

Yandex

Схема Inference Server



Yandex

Какую задачу мы решаем в рекламе

Yandex

Какую задачу мы решаем в рекламе

Разрабатываем ML инфаструктуру, в частности сервисы ML Inference.
Наши сервисы должны:

- Быть **эффективными**
- Иметь хороший time-to-prod
- Быть надёжными

disclamer

В этом докладе в **не** будем рассматривать MLOps процессы, а сосредоточимся на **оптимизациях**, которые мы внедряли.

Какую задачу мы решаем в рекламе:

Наши масштабы

Yandex

Масштабы

- Поисковая реклама:
  - ∼ Десятки тысяч RPS
  - ∼ Десять тысяч CPU
  - 4 сервиса
- Рекламная сеть Яндекса:
  - ∼ **Сотни тысяч RPS**
  - ∼ **Десятки тысяч** CPU
  - 4 сервиса
- Контент система рекламы:
  - Десятки миллиардов баннеров в день ∼ 10 – 100 тысяч RPS
  - ∼ Десятки тысяч CPU
  - 5 сервисов
- Другое (реклама в телеграме, генерация баннеров, ...)
- **Много** GPU
- Время генерации ответа рекламы — 200 ms Q99

Yandex

Какую задачу мы решаем в рекламе:

Open source и почему решили делать своё решение

Yandex

Triton Inference Server

- https://github.com/triton-inference-server
- Даёт только C++ библиотеку:
  - Можно развернуть сервис в kubernetes со статическим набором моделей
  - Систему деплоя моделей в сервис и многое другое нужно настраивать самому

Плюсы:

- Готовое достаточно эффективное решение
- Поддерживает большинство фреймворков инференса (TensorRT, ONNX runtime, ...)
- Широко используется известными компаниями
- Поддерживает переподгрузку моделей без рестарта сервиса

Минусы:

- Недостаточно эффективный на большом количестве маленьких входов
- Налаживать инфраструктуру нужно самому (шардирование моделей по подам, autoscaling):
  - Шардирование моделей по подам
  - Autoscaling
  - Hot reload: для переподгрузки моделей без рестарта есть только API, нужно разработка для полной поддержки

Yandex

Другие

Другие:

1. BentoML
   Написан на питоне. Можно использовать как самостоятельно, так и как надстройку над triton.
   Плюсы: Гибкий, простой
   Минусы:
   - Не подходит из-за производительности
   - Всё ещё требует интеграции в яндекс инфру
2. Seldon core, Ray, ...
   Имеют примерно те же плюсы и минусы, что BentoML
3. AMD inference server/kserve/torch serve/... — являются очень похожими по смыслу на triton.
   - Как правило, ничем не лучше triton
   - Поддерживают меньше фреймворков инференса
   - Имеют те же недостатки, что triton
4. **TLDR; Посмотрите замечательный доклад:**
   PyterPy 2023, «Model serving. Какой фреймворк выбрать?»
   `https://piterpy.com/archive/2023/talks/f0472db53d144a81af68e952783ee756/`

Yandex

Почему решили развивать свой Inference Server

Итог по open source технологиям: Вероятно одно из самых лучших решений — взять triton и строить вокруг него свою инфру.

Почему свой:

1. Легче разрабатывать **свои модули** к сервису
2. Проще и быстрее делать **интеграцию в инфрастуктуру** Яндекса
3. Верим, что можем сделать **оптимальнее**
4. Широко используем **ragged тензоры**.
   Пример: история тайтлов баннеров пользователя.
   issue на гитхабе про поддержку нескольких ragged размерностей висит с декабря 2022

Yandex

Как устроен наш сервис

Yandex

Компоненты

- **InferenceBackend.**
  Позволяет подключать разные форматы модели к сервису.
  Примеры: Pytorch, TensorRT, TensorFlowLight, ONNX Runtime
- **ModelInstance.**
  Метод Infer принимает на вход и возвращает именованный набор тензоров:
  $Infer :: \left[(InputName, Tensor)\right] \rightarrow \left[(InputName, Tensor)\right]$
- **Runtime.**
  Вся бизнес логика
- **Server.**
  Предоставляет API (Http/Grpc/WebSocket) Server

Yandex

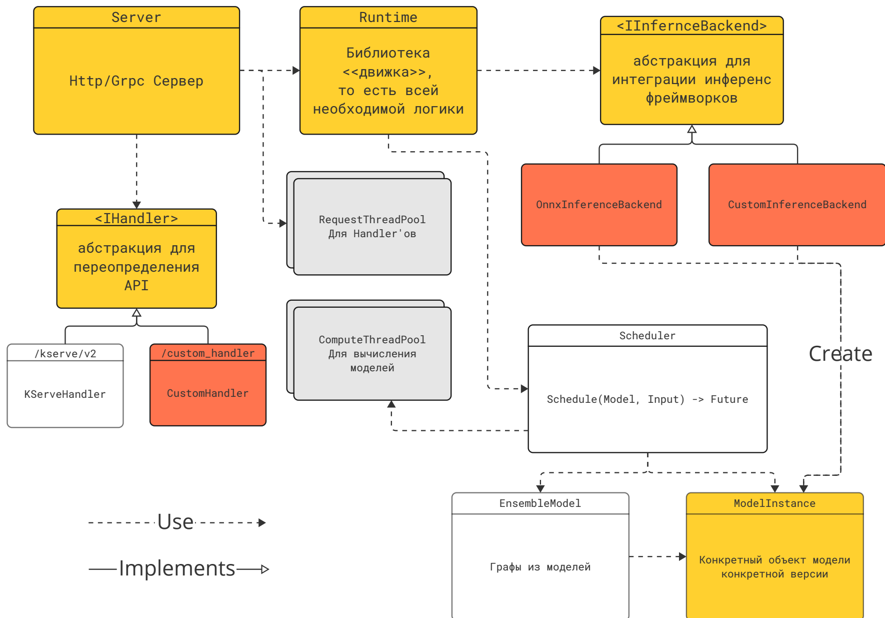


Figure: Схема основных компонент сервиса

Примеры конфигов нашего сервиса

application.txt.pb

```
ProgramConfig: {
    ServerConfig: {
        IoServerThreads: 2
        RequestThreads: 2
        Port: 80
    }

    TracingConfig {
        ServiceName: "my_gpu_inference"
        SampleRatio: 0.1
    }
}

RuntimeConfig: {
    InferencePoolSize: 20
    SharedBatcherConfig: {
        BatchingThreads: 2
    }
    ModelRepositoryConfig: {
        Path: "/workloads/ml_inference/repo"
    }
}

ManagedByReloader: true
```

my_model.yaml

```
backend_config:
  backend: unitednn
  custom_params:
    - key: lock_memory
      value: true
    - key: subnetwork
      value: _head
    - key: gpu
      value: '0'
runtime_config:
  batching_config:
    max_batch_size: 8
    batch_timeout_mcs: 2000
```

Yandex

Оптимизации

Yandex

Оптимизации:

Сплит архитектура моделей

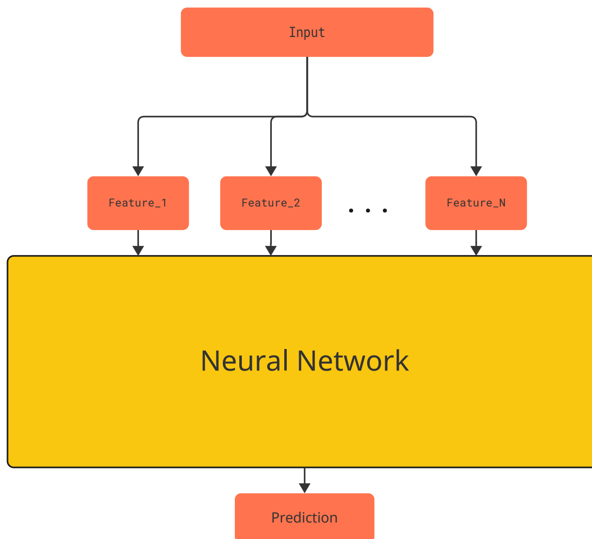
Yandex

Проблема 1: Количество кандидатов

1. Возьмём Рекламную Сеть Яндекса
2. 300 000 RPS
3. Несколько сотен баннеров-кандидатов на каждый запрос
4. $\implies$ 60 миллионов документов в секунду
5. Считать нейросети на таком потоке *очень* дорого
6. Поэтому большинство моделей имеют архитектуру позднего связывания (или split)

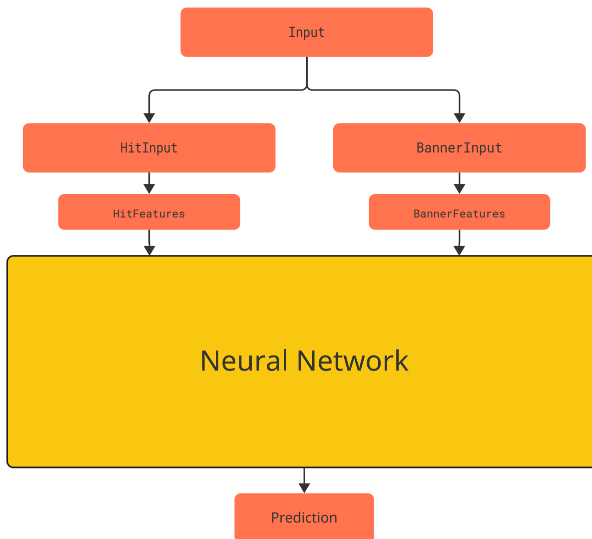
Yandex

Нейросетевая модель в общем случае



Yandex

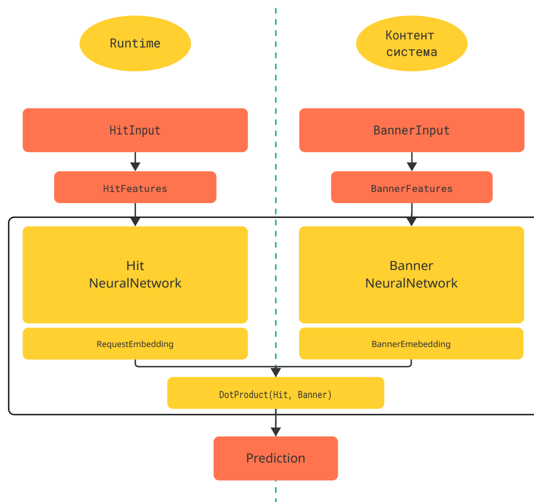
Нейросетевая модель в рекламе



Yandex

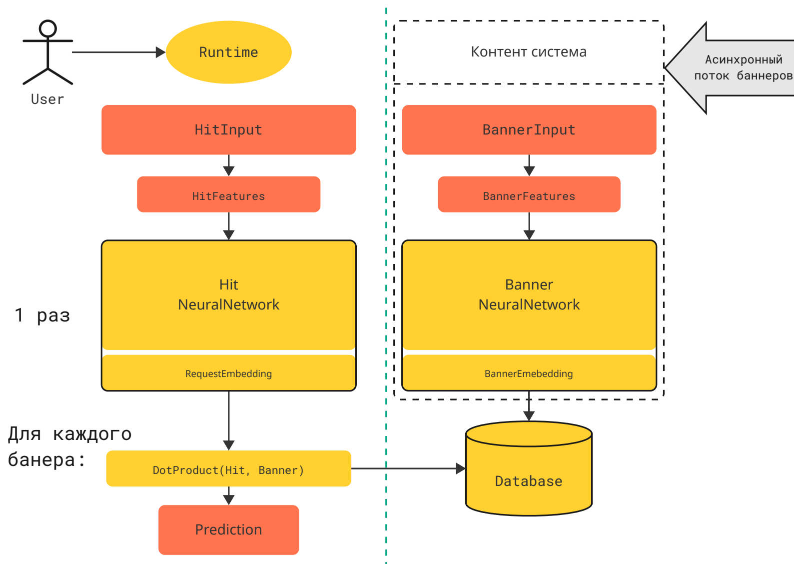
Нейросетевая split модель в рекламе

- Учим модель специальной архитектуры (похоже на DSSM)
- Hit и Banner слои связываем в конце через скалярное произведение
- После обучения делим модель на две чисти
- Hit часть считаем в рантайме
- Banner часть считаем в контент системе
- В рантайме достаём эмбеддинг баннер части из базы
- Predict считаем как DotProduct



Yandex

Нейросетевая split модель в рекламе



Pros & cons

Плюсы:

1. Кратно оптимизируется нагрузка (сотни раз)

Минусы:

1. Требует изменений архитектуры
2. Можно потерять в качестве.
   Эксперименты в рекламе не доказали этого (но и не опровергли)
3. Требует разработки:
   - Сервис в контент системе
   - Доставка эмбедингов в базу
   - Лукап в базе и подсчёт DotProduct

Yandex

Оптимизации:

Кэширование

Yandex

Кэширование слабого человека

Добавление кэширования выглядит просто:

- Пишем или берём готовый кэш
- Profit

На практике:

- Наивное кэширование по всем входам даёт очень маленький кэшхит
- Кэшируем по подмножеству входов
- Входы по `User` кэшируем по `UserId` на `TTL` (`TTL = 120`) секунд

Пример:

```
Input {
    UserProfile {
        UserID: 42, // участвует в ключе кэша
        ...         // не участвует в ключе кэша
    }
    HitContext {
        PageID: 1  // номер площадки рекломадателя, участвует в ключе кэша
        PageTitle: "Super Title" // Зависит от PageID, не участвует в ключе кэша
        ...
    }
}
```

Итог: 10-20% кешхит -> такая же экономия `CPU` .

Yandex

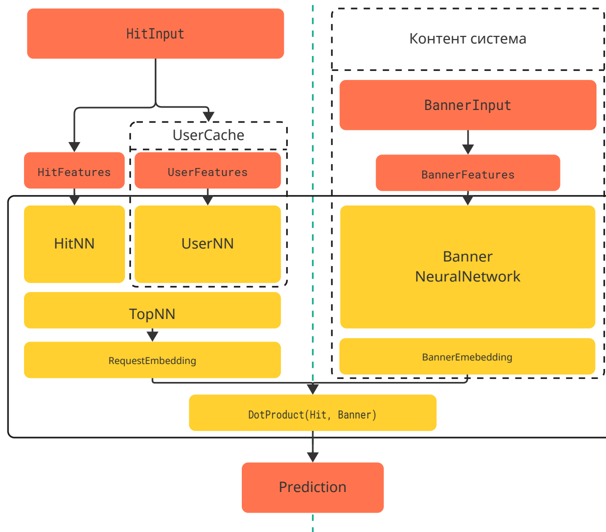
Кэширование сильного разработчика

Алгоритм:

- Выделим ещё кусок у модели — `UserNN`
- Сделаем по нему отдельный кеш

Профит:

- Кешхит по User равен 60%.
- Допустим `CPU (UserNN) = CPU (HitNN + TopNN)`
- Экономия $= 60\% \cdot 0.5 = \mathbf{30\%}$



Yandex

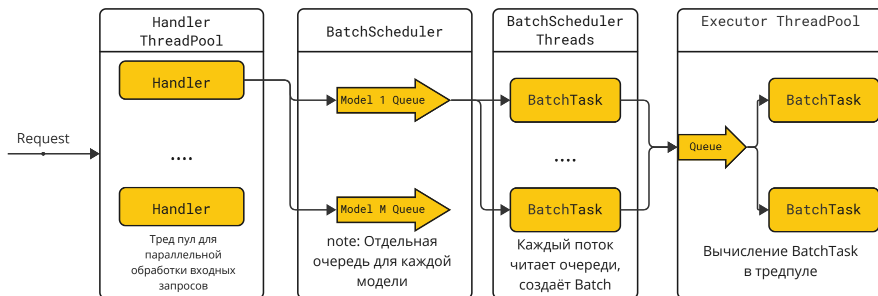
Оптимизации:

Батчевание

Yandex

Батчевание — схема

- Складываем запросы в очередь и отдельными тредами выгребаем элементы из очереди батчами:
  - Параметры: `BatchTimeout` и `MaxBatchSize`
  - `MaxBatchSize` — максимальный размер батча
  - Ждём накопления батча не более `BatchTimeout` времени
- must-have фича на GPU сервисах
- Оказывается, экономит много даже на CPU сервисах



Yandex

Почему экономится CPU ?

Представим, что:

- Вход модели – вектор размера m
- Модель — умножение вектора на матрицу
- В батче n элементов

Соображения:

- Умножать матрицу $n \times m$ на матрицу можно асимптотически быстрее, чем n раз вектор на матрицу
  Модели используют суперумную библиотеку `Eigen` для работы с матрицами
- При умножении батча входов на матрицу вы меньше ходите в память (за весами матрицы)
- Скорее всего, помогают кеши процессора на обращения в память

Yandex

Батчевание — профит

Профит:

- Экономия не менее 20+% CPU сервиса
- Пример: Потребление CPU одного из сервисов в момент включения батчевания: 6800 -> 5150 ядер

Минусы:

- Требует настройки параметров батчевания
- Ухудшает latency (время ответа)

Оптимизации:

GPU Inference

Yandex

Схема

GpuServiceHandler

- IScheduler Scheduler;

```
modelInput = Preprocess(request);
modelResult = scheduler.Schedule(modelName, modelInput);
modelResult.Wait();
return modelResult.Get();
```

Код сервиса:

- Сделали некоторый Preprocessing — CPU работа
- Пошедулили вычисление модели на GPU и дождались
- Вернули результат

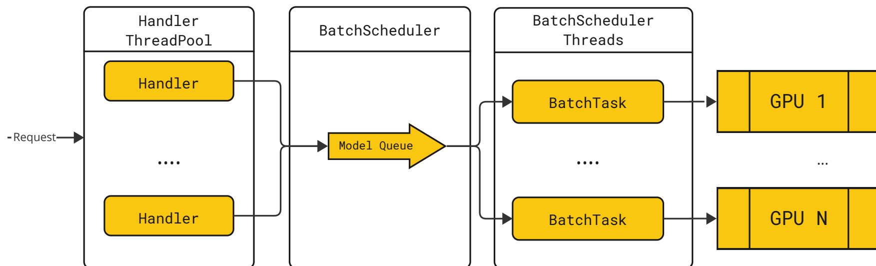


Figure: Схема запроса в GPU сервисе с одной моделью

Yandex

Узкое место

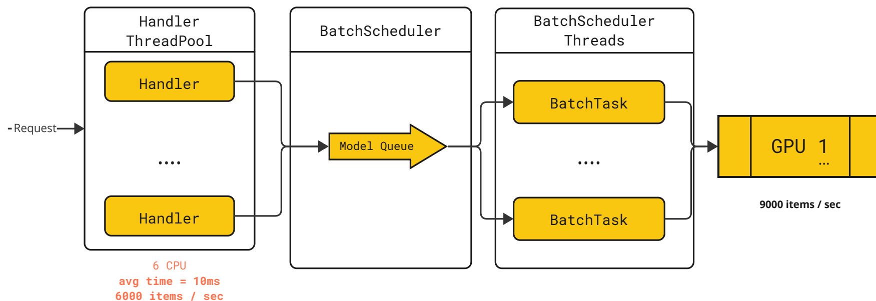
```
GpuServiceHandler
- IScheduler Scheduler;

modelInput = Preprocess(request);
modelResult = scheduler.Schedule(modelName, modelInput);
modelResult.Wait();
return modelResult.Get();
```

Карточка процессит 9K запросов в секунду.

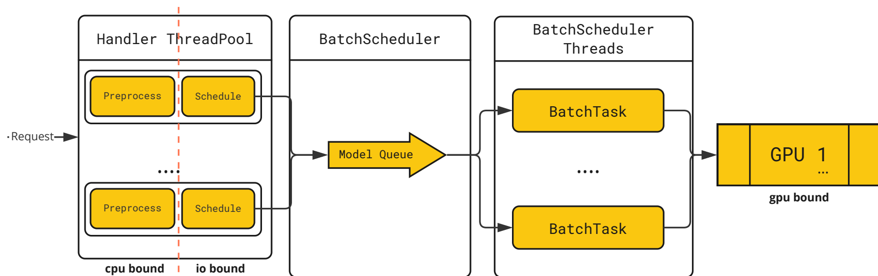
Конфигурация GPU хоста:

- 56 ядер
- 8 GPU карт



Yandex

Заметим что...



Yandex

Разделяй и получай профит

Отмасштабируем!

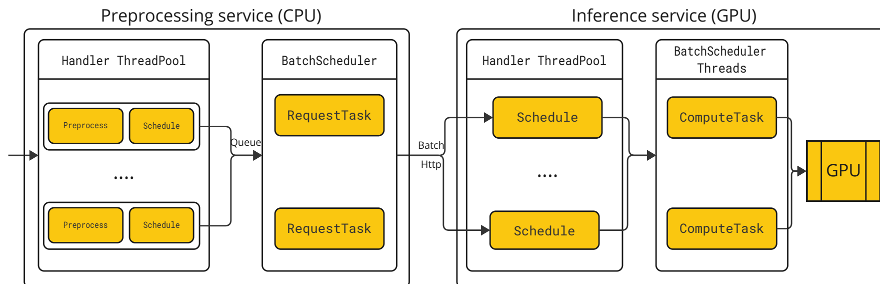


Figure: Схема разделённого по CPU only / GPU подам сервиса

Итог:

- Сэкономили **30%** GPU
- Потратили $\times 1.5 - 2$ CPU на сервис и перекладывание байт не дорого, по сравнению с экономией GPU

Yandex

Итоги

Yandex

Посчитаем

1. Сплит модели (Hit/Banner)
   Экономия: **в сотни раз** CPU **и** GPU
2. Кэширование v1
   Экономия: **10-20%** CPU
3. Кэширование v2 (сплит User/Hit)
   Экономия: 10-60% CPU , **30%** CPU в среднем
4. Батчевание
   Экономия GPU : в десятки-сотни раз.
   Экономия CPU : $\geq$ **20%**.
5. GPU split на CPU preprocessing и GPU inference
   Экономия: **30%** GPU

Yandex

Выводы

- Оптимизировать можно и нужно
- Оптимизации требуют разработки и часто уместны только начиная с какого-то объёма мощностей
- Большинство крупных оптимизаций продуктовые.
  Но низкоуровневыми оптимизациями заниматься тоже нужно
- Оптимизации — это tradeoff latency, throughput и качества модели

Мой совет:

- Ищите продуктовые оптимизации

Yandex

Планы

Yandex

Что есть

Что у нас уже есть:

- Inference Server с основной функциональностью
- Набор инструментов для интеграции в систему деплоя и мониторинга
- Поддержка нескольких InferenceBackend, использующихся в моделях рекламы
- Толпа сервисов в рекламе с немного разными функциональностями и кодом
- Сервисы в маркете и медиасервисах на нашей технологии

Yandex

Планы

Развиваем и обобщаем нашу технологию:

- Продолжаем оптимизировать и разрабатывать фичи:
  - Кэши
  - Динамическая балансировка (учитывает throughput пода)
  - Разные алгоритмы шедулинга вычисления моделей на GPU
  - etc.
- Инфрастуктура
  - IAC (Infrastructure As a Code)
  - Автоскейлинг & автошардирование
  - Inference As A Service
- Инструменты для замера performance моделей

Внедряем:

- Унифицируем сервисы и ускоряем внедрение ML в рекламе:
  - Амбиция — time-to-exp $\sim$ 2 дня
- Web Поиск
- Различные рекомендательные сервисы Яндекса
- Весь Яндекс

Мы нанимаем ;)

Yandex