Model Deployment & Serving

Page • 1 backlink • uni

Варианты деплоя

- cloud
- on premise – закрытый контур
- on device – клиентский контур
- report

Стадии

- обучение
- упаковка
- раскатка и обслуживание
- мониторинг
- обновление

Сложности

- масштабирование
- низкая задержка
- стабильность
- возобновляемость

Задачи

- доставка запросов и возврат ответов
- доставка данных
- отгрузка логов
- мониторинг работы
- оптимизация
- среда (и отсутствие жесткой завязки на нее)

Еще до деплоя нужно понимать, что

- мы точно катим то, что нужно?
- у нас достаточно железа для раскатки? (с учетом ожидаемого трафика от пользователей)

- у нас достаточно железа для обучения и дообучения?
- как мы поймем, что модель до сих пор крутится?
- можем ли воспроизвести обучение модели?
- учли ограничения от СИБ?
- как будут интегрироваться клиенты?
- не поломаем ли архитектуру сервиса?

Дашборд или ноутбук в Jupyer тоже считается деплоем, но мы говорим о деплое в ключе реализации некоторого сервиса

- cloud computing (как правило, голосовые помощники)
    - batch prediction – генерация до прихода запроса, предсказания сохраняются и достаются по запросам, асинхронная работа

        > В качестве примера, Яндекс.Погода с некоторой периодичностью использует Catboost для обсчета погодных условий на карте, расбитой по сетке (легко паралеллизовать), на разные временные промежутки вперед – если пайплайн дообсчета упал, ничего страшного, в течение некоторого времени будет устаревшая погода

    - online prediction – генерация после прихода запроса, предсказания возвращаются в качестве ответов, синхронизация посредством REST/gRPC
    - гибридная схема: предложения YouTube
- edge computing (оффлайн-переводчики и прочее)
    - плюсы: работает оффлайн, не беспокоимся о задержке сети, меньше забот о конфиденциальности, дешевле
    - сложности: нагрузка на устройство конечного пользователя, необходимость поддерживать разные GPU, трудность обновления

Способы взаимодействия с моделью

- FastAPI – удобен для быстрого прототипирования (но часто остается в продакшене)
- REST – перекладывание JSON-ов
- gRPC – де-факто текущий стандарт

Варианты

- унарный – запрос-ответ

- поток от клиента
- поток от сервера
- двунаправленный поток *(например, распознавание речи)*

DVC – аналог Git для data science, идея использования: общий реестр, куда DS складывает модель, а ML-инженер достает и внедряет (что-то упало – идем в реестр, ищем коммит с ошибкой, исправляем ошибку)

Контейниризация (Docker и прочие механизмы) используется вместо изжившей себя идеи сбора кода в бинарь – такой подход был полезен для статического кода, например, на C++, но современный динамический код на Python удобнее контейнеризовать и иметь возможность запустить его на любой машине с совместимым железом

Балансировка – клиент обращается только к балансировщику, который распределяет нагрузку по инстансам (если инстанс упал, информация отправляется в реестр, доступ к которому есть у балансировщику)

- L3-level – network-level balancing, прячем множество инстансов под IP, проблем мало, но зачастую такой балансировки недостаточно
- L7-level – application-level balancing (с помощью Envoy), знаем про используемый протокол и в зависимости от типа запроса направляем его на определенный инстанс + там же авторизация и анти-DDOS

Балансер знает про все инстансы, нужна более сложная логика на балансере

Паковка моделей на кластере

- плотная
- VM per service

Ресурсы

- CPU
- GPU
- RAM
- VRAM (более общо, GPU RAM)
- network

Распределять модели по машинкам с учетом требуемых ими ресурсов можно хоть руками

Необходимо пропускать трафик через легковесный контур при падении основного (или не выдерживании им нагрузки), чтобы сервис не превратился в тыкву

Стратегии деплоя

- rolling deployment – раскатываем на машины по возможности, но в один момент времени может получиться, что одна модель работает со старой моделью, а на другой уже крутится новая
- blue / green deployment – раскатываем на машины новую модель параллельно со старой и атомарно переключаемся при успешной загрузке на все машины (нужно двойное количество памяти на каждой машине)
- canary – перенаправление трафика на новую модель с итеративным увеличением доли (rolling – внутри сервиса переключается модель, parallel – переключаемся на новые инстансы с моделью)

    > Похоже на A/B, но при AB-тестировании нам интересны метрики и определение наличия эффекта, а canary – это скорее smoke test на то, что приложение/сервис не упадет

    - обращение через разный API: (?)
    - обращение через один API: (?)

Мониторинг

- падений/перезапусков/неответов
- нагрузки железа, задержки
- модель (про это дальше)