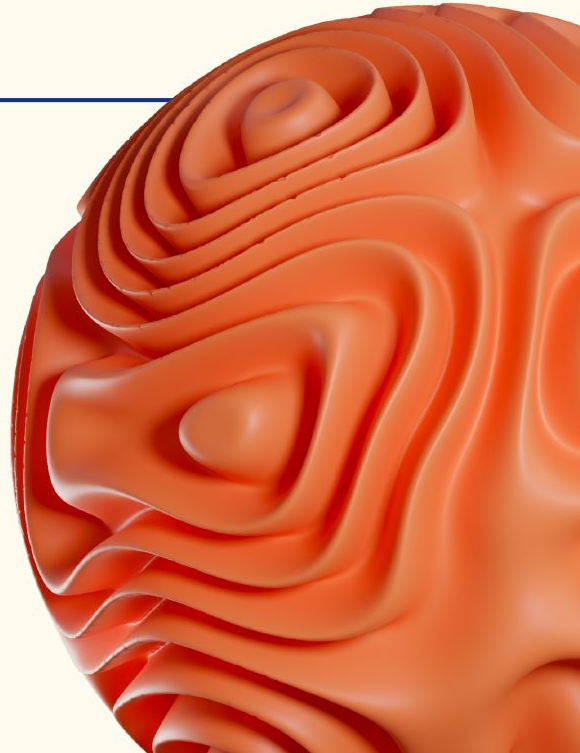
Модели ML в production

МФТИ × SKILLFACTORY

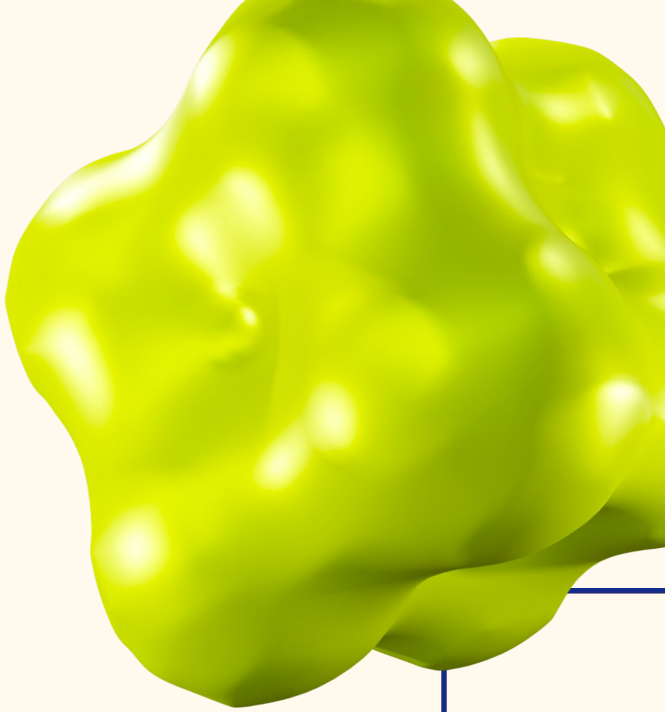
Лекция № 4
“Инференс модели на Flask + многопроцессность + ВО”



Жарова Мария Александровна
DS WildBerries, председатель дипломной комиссии на курсе по Data Science
t.me/data_easy

План занятия

1. Сервер на Flask с виртуальным клиентом
2. Параллелизм в Python:
    - Потоки и процессы
    - GIL
    - Parallel из joblib
    - ThreadPoolExecutor из concurrent
    - gunicorn
    - uswgi



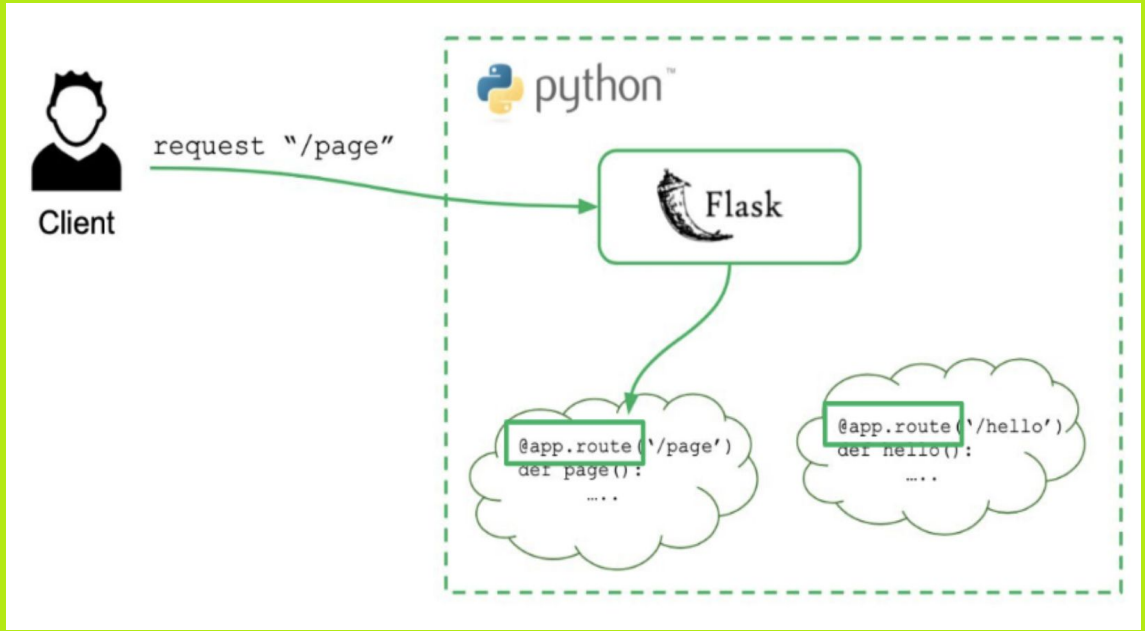
1. Сервер на Flask с виртуальным клиентом

Flask

МФТИ × SKILLFACTORY

Пользователя можно использовать:

- **реального** => веб-интерфейс, приложение
- **имитировать** при помощи отдельного кода



2. Параллелизм в Python

Процессы и потоки

Процесс — это экземпляр выполняемой программы со своим адресным пространством в памяти.
Поток — это единица выполнения внутри процесса, использующая его память и ресурсы.

Важные факты:

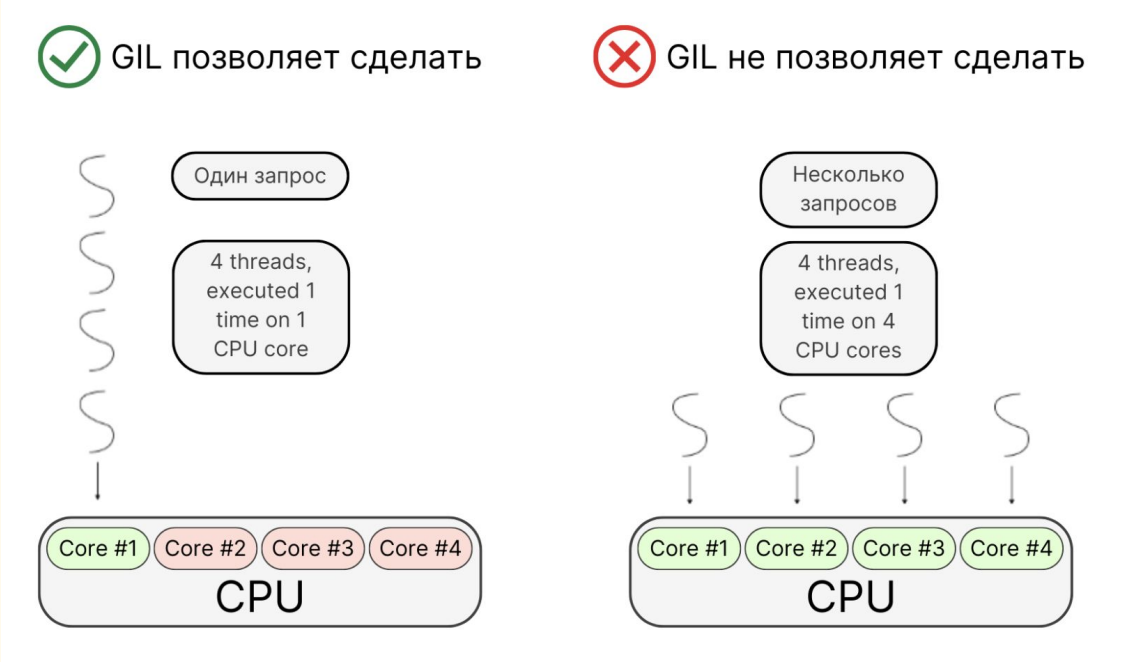
- 1 ядро в процессоре != 1 процесс (возможен механизм многозадачности scheduling).
- Если запускаете программу несколько раз — это будет одна программа, но несколько процессов, которые являются её экземплярами
- У программы может быть один или несколько процессов
- У процесса может быть один или несколько потоков

Процессы и потоки

Критерий сравнения	**Процесс**	**Поток**
Совместное использование памяти	Память не разделяется между процессами	Память разделяется между потоками внутри одного процесса
Объем потребляемой памяти	Большой	Маленький
Для каких задач	Изолированные задачи	Задачи, привязанные к вводу/выводу
Время начала работы	Медленнее, чем у потока	Быстрее, чем у процесса
Прерываемость	Дочерние процессы можно прервать	Потоки прервать можно, но сложнее

GIL

GIL (Global Interpreter Lock) — это глобальная блокировка интерпретатора, которая позволяет одновременно исполнять **только один поток Python-кода в рамках одного процесса**.



Как решить проблему?

Если не можем запустить несколько потоков в одном процессе - запустим несколько процессов:)

Принцип работы:

- запускаем команду *старта* веб-сервера
- передаём в неё в качестве параметра *указание на скрипт*, в котором написано приложение
- команда запустит *несколько рабочих процессов* (копий главного скрипта)
- также будем *прослушивать* входящие запросы и *передавать их на обработку* в один из запущенных процессов.

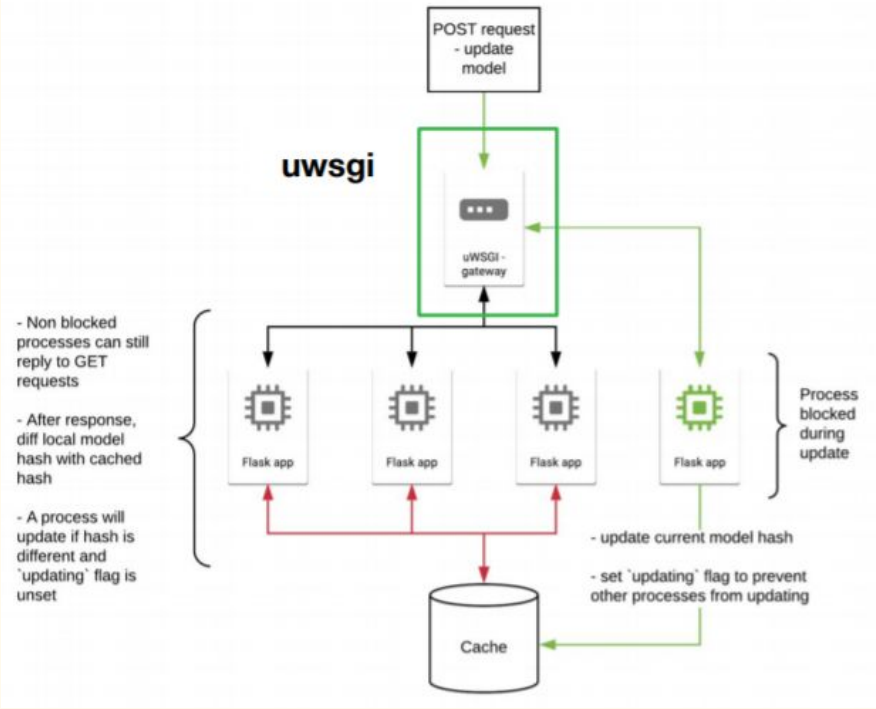
Как решить проблему?

Важные факты про GIL:

- GIL не запрещает многопоточное программирование — просто Python-код исполняется по очереди между потоками.
- GIL не мешает работать с I/O (сеть, файлы, БД) — другие потоки могут выполнять такие операции, пока основной поток ждёт.
- GIL не действует на C-расширения (например, NumPy, SciPy, Numba, Pandas, FAISS). Если код использует векторные вычисления в C (без GIL), потоки могут работать параллельно.

Серьёзное решение

Хорошо для Flask: Gunicorn, связка NGINX + uWSGI



Подготовка проекта:

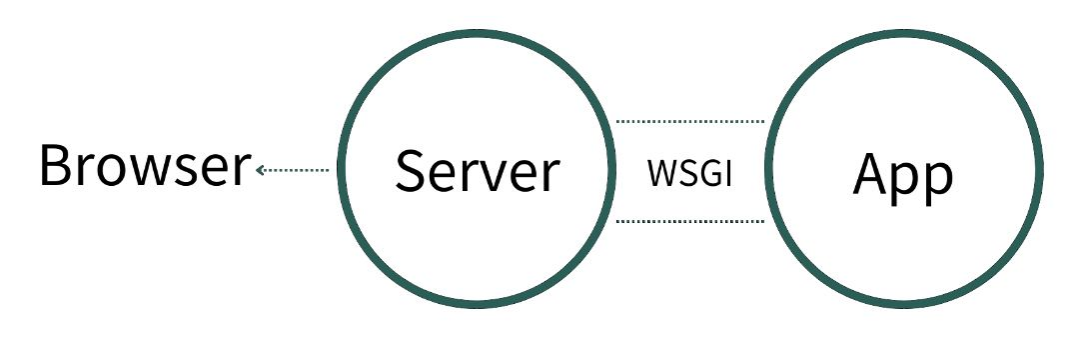
```
├─web
    ├─models
        └─model.pkl
    └─client.py
    └─server.py
```

WSGU

Нужен интерфейс, который будет:

- запускать приложение в нескольких процессах
- обрабатывать поступающие запросы
- возвращать пользователю ответ

WSGI (Web-Server Gateway Interface) — протокол, стандарт взаимодействия между Python-программами и веб-сервером.



Простое решение

```python
from joblib import Parallel, delayed

def square(x):
    return x ** 2

results = (
    Parallel(
        n_jobs=-1, # кол-во процессоров/потоков
        backend='threading', # мультипроцесс./многопот.
        verbose=1 # кол-во выводимых логов
    )
    (delayed(square)(i) for i in range(100))
)

print(results)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256,
5, 676, 729, 784, 841, 900, 961, 1024, 1089, 1156, 1225, 1296, 1369, 14
5, 2116, 2209, 2304, 2401, 2500, 2601, 2704, 2809, 2916, 3025, 3136, 32
6, 4225, 4356, 4489, 4624, 4761, 4900, 5041, 5184, 5329, 5476, 5625, 57
9, 7056, 7225, 7396, 7569, 7744, 7921, 8100, 8281, 8464, 8649, 8836, 90
```

```
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 8 concurrent
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    0.0s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:    0.0s finished
```

Parallel из Joblib: для ускорения инференса/расчётов, особенно пригодится для эффективного использования ресурсов видеокарт, батчевой обработки данных.

Для запуска нескольких Flask-серверов: можно, но не очень хорошо.

Шутка:)

```python
from joblib import Parallel, delayed
from flask import Flask

def run_server(port):
    app = Flask(__name__)

    @app.route("/")
    def hello():
        return f"Hello from port {port}!"

    app.run(port=port)

# Запуск серверов на портах 5000 и 5001
(
    Parallel(n_jobs=2)
    (delayed(run_server)(port)
     for port in [5000, 5001])
)
```

Сервер на Flask при помощи Parallel:
в качестве “итерируемого объекта” перебираем различные порты, на которых будут запущены сервера.

Лучше gunicorn:

```
gunicorn -w 4 app:app
```

здесь приложение app будет запущено на 4-ч параллельных “воркерах” (в 4-х параллельных процессах).

Спасибо
за внимание!

МФТИ × SKILLFACTORY