

Описание структуры проекта.

Задача #15.

Команда go_hack x Yarrу.

План.

- 1) Описание структуры проекта;
- 2) Перечень инструментов/языков программирования/библиотек/open-source решений, использованных в работе;
- 3) Скорость работы системы в разрезе двух подзадач: вставка нового элемента в базу данных/индекс; ответы на текстовый запрос пользователя.
- 4) Идеи и гипотезы об улучшении/доработках системы, которые не успели сделать в рамках Хакатона.
- 5) Схема логики взаимодействия всех модулей внутри проекта.
- 6) Указание уникальности решения.

Команда:

Александр Валясин [tg](#) - Data Engineer, Data Scientist

Кирилл Богатырёв [tg](#) - Backend developer

Богдан Онищенко [tg](#) - Data Scientist

Никита Молчанов [tg](#) - Data Scientist

Денис Самаркин [tg](#) - Data Engineer, Data Scientist

1. Описание структуры проекта

Проект представляет собой веб-сервис, обеспечивающий поиск и индексацию видеоконтента по его содержанию на основе текстовых запросов. Структура проекта включает в себя следующие компоненты:

Основные файлы и директории:

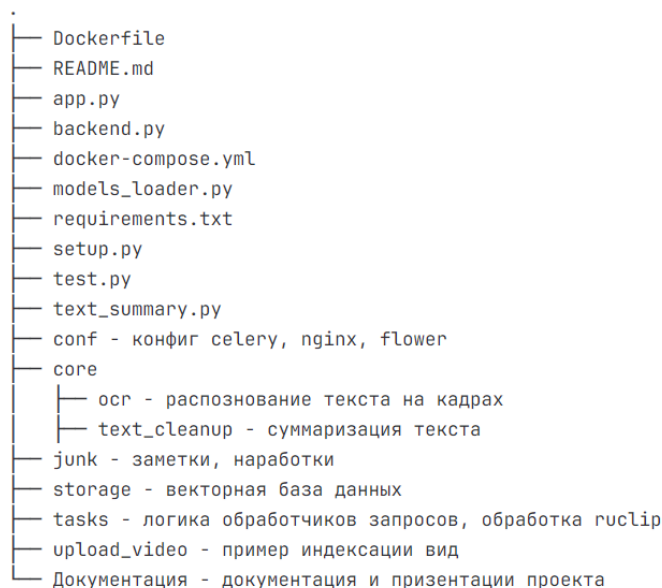
- **Dockerfile**: файл для создания Docker-образа проекта.
- **README.md**: документация проекта, включающая инструкции по установке и использованию.
- **app.py**: основной файл приложения, отвечающий за обработку запросов и взаимодействие с пользователями.
- **backend.py**: файл, содержащий логику бэкенда, включая обработку и индексацию видео.
- **docker-compose.yml**: файл для оркестрации Docker-контейнеров.
- **models_loader.py**: скрипт для загрузки и инициализации моделей машинного обучения.
- **requirements.txt**: файл с перечнем зависимостей проекта.
- **setup.py**: файл для установки проекта.
- **test.py**: тесты для проверки функциональности проекта.
- **text_summary.py**: модуль для суммаризации текста.
- **conf**: директория с конфигурационными файлами для Celery, Nginx и Flower.
- **core**: основная логика проекта.
 - **ocr**: модуль для распознавания текста на кадрах видео.
 - **text_cleanup**: модуль для очистки и суммаризации текста.
- **junk**: директория с заметками и набросками.
- **storage**: директория с векторной базой данных.
- **tasks**: логика обработчиков запросов, включая обработку RuCLIP.
- **upload_video**: пример индексации видео.
- **Документация**: директория с документацией и презентациями проекта.

Основные компоненты проекта:

- **Видеоиндексация**: модуль для индексации видеоконтента, включающий распознавание текста и аудио, генерацию эмбеддингов и сохранение их в базу данных.
- **Поиск по видео**: модуль для выполнения поиска по видео на основе текстовых запросов пользователей.
- **Векторная база данных**: используется для хранения и быстрого поиска по эмбеддингам видео. Технология: LanceDB.
- **OCR и суммаризация текста**: модули для распознавания и обработки текста, полученного из видео. Технологии: EasyOCR и mT5.

Проект также включает различные конфигурационные файлы и скрипты для настройки и развертывания сервисов, а также для обеспечения масштабируемости и производительности системы.

Project structure



```

.
├── Dockerfile
├── README.md
├── app.py
├── backend.py
├── docker-compose.yml
├── models_loader.py
├── requirements.txt
├── setup.py
├── test.py
├── text_summary.py
├── conf - конфиг celery, nginx, flower
├── core
│   ├── ocr - распознавание текста на кадрах
│   └── text_cleanup - суммаризация текста
├── junk - заметки, наброски
├── storage - векторная база данных
├── tasks - логика обработчиков запросов, обработка ruclip
├── upload_video - пример индексации вид
└── Документация - документация и презентации проекта

```

Репозиторий: <https://gitlab.com/fizzzgen/video-indexer> (открытый)

2. Перечень инструментов, языков программирования, библиотек и open-source решений, использованных в работе

Языки программирования

- **Python:** Основной язык для разработки backend и ML-алгоритмов, обеспечивающий гибкость и множество библиотек для машинного обучения и обработки данных.

Инструменты и библиотеки

1. **FastAPI:** Используется для создания веб-фреймворка, обеспечивающего API для взаимодействия с пользователем и обработки запросов на поиск и индексацию видео.
2. **Celery:** Для управления задачами в очереди и их асинхронного выполнения, что повышает эффективность обработки данных.
3. **Redis:** Используется для кэширования эмбедингов запросов и хранения промежуточных данных, что снижает нагрузку на основную базу данных и ускоряет ответы на запросы из-за кэша.
4. **Docker:** Контейнеризация приложения для обеспечения изоляции, легкости развертывания и масштабируемости.
5. **ruCLIP:** Модель для генерации векторов из изображений и текста, используемая для индексации и поиска видео по визуальному и текстовому содержанию.

6. **Hugging Face Transformers**: для работы с предобученными моделями NLP.
7. **EasyOCR**: Библиотека для распознавания текста на изображениях, применяемая для извлечения текстовой информации из видеофреймов.
8. **LanceDB**: Векторная база данных, используемая для быстрого и эффективного поиска по видео на основе векторных представлений.
9. **NGINX**: Веб-сервер, используемый для обработки и маршрутизации запросов, балансировки нагрузки и обеспечения безопасности.
10. **FFmpeg**: Инструмент для обработки аудио и видео, используемый для экстракции фреймов из видео и предварительной обработки мультимедийных данных.

Open-source решения

1. **Ruclip-vit-base-patch16-384**: Открытая модель, разработанная для создания векторных представлений текстов и изображений, что позволяет использовать ее для мультимодальных задач.
2. **Whisper** - Модель для автоматического распознавания речи в текст. Одна модель для распознавания, перевода, определения языка. Также работаем в основном на RU+EN.
3. **EasyOCR cyrillic_g2 и CRAFT**: Open-source библиотека для оптического распознавания текста, доступная под лицензией Apache-2.0.
4. **mT5_multilingual_XLSum**: Open-source модель для суммаризации текста. Используем для генерации эмбеддингов из текстов и изображений с одинаковым размером. Из-за широкого применения модели, одновременно не перегружаем архитектуру и получаем когерентные эмбеддинги.
5. **LanceDB**: Open-source векторная база данных, обеспечивающая быструю индексацию и поиск, поддерживающая работу с GPU для ускорения обработки.

Датасет:

- **Yappy**: датасет, содержащий 400к видео и их описаний.

Программные решения и утилиты

- **GitLab**: Платформа для управления исходным кодом, CI/CD и совместной разработки, использованная для контроля версий и автоматизации процессов сборки и развертывания.
- **Docker Compose**: Инструмент для определения и управления многоконтейнерными Docker-приложениями, обеспечивающий удобство развертывания и тестирования.
- **JupyterLab**: для проведения тестирования.

Эти инструменты и библиотеки выбраны для обеспечения высокой производительности, масштабируемости и гибкости системы, что позволяет эффективно решать задачи поиска и индексации видеоконтента.

3. Скорость работы системы в разрезе двух подзадач: вставка нового элемента в базу данных/индекс; ответы на текстовый запрос пользователя

Машина:

- 32 vCPU
- SSD
- 64 Gb RAM

Индексация: 35 sec

Поиск: 0.3 sec

1. Поиск видео по текстовому запросу:

GET <http://176.123.161.67/api/search>

Python3

```
import requests
payload = {'query': 'dogs'}
response = requests.get('http://176.123.161.67/api/search', params=payload)
```

Bash

```
curl "http://176.123.161.67/api/search?query=dogs"
```

2. Индексация видео

POST <http://176.123.161.67/api/index>

Python3

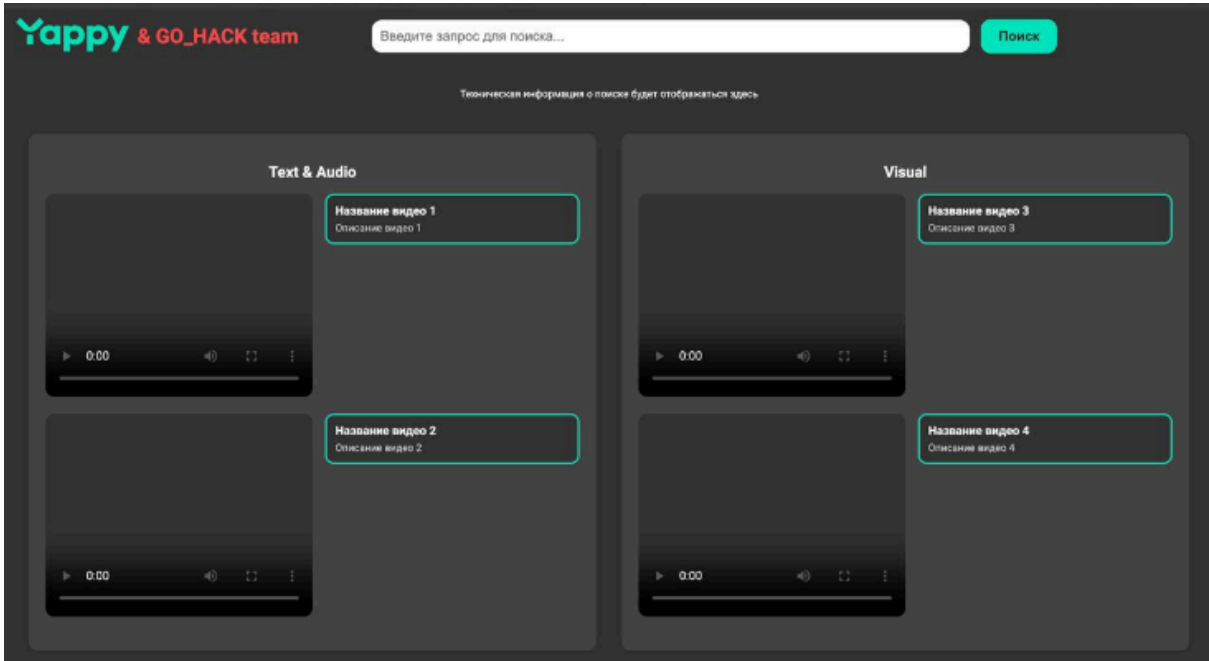
```
import requests
payload = {'link': url, 'description': description}
response = requests.post('http://176.123.161.67/api/index', json=payload)
```

Bash

```
curl --header "Content-Type: application/json" \
--request POST \
--data '{"link":"url","description":"description"}' \
"http://176.123.161.67/api/index"
```

Статус индексации GET <http://176.123.161.67/workers>

Фронт:



Flower (посмотреть статус задач, очередь и тд):

Воркеры

C Flower Workers Tasks Broker Documentation							
Show 15 workers		Search:					
Worker	Status	Active	Processed	Failed	Succeeded	Retried	Load Average
worker@6c1d0733d29d	Online	0	0	0	0	0	0, 0.01, 0
Total		0	0	0	0	0	
Showing 1 to 1 of 1 workers							Previous 1 Next

Очередь

redis://redis:6379//

Queue	Messages	Unacked	Ready	Consumers	Idle since
celery	0	N/A	N/A	N/A	N/A

4. Идеи и гипотезы об улучшении/доработках системы, которые не успели сделать в рамках Хакатона

Идеи и гипотезы

1. Внедрение модели для классификации запросов:

- Отсутствие шаффла между визуальными и текстовыми эмбедингами. Мы предлагаем добавить в пайплайн еще одну модель в начало для классификации запроса - визуального или текстового. Потенциально это может ускорить сервис.
- Похожим решением будет добавить модель в самый конец пайплайна для шаффла результатов.

2. Использование S3 для хранения данных:

- **Идея:** В LanceDB оптимизирована для работы на диске и без сервера. Предлагаем использовать S3 для хранения и на операции write. Для операций read предлагаем использовать реплику LanceDB в файловой системе. LanceDB легко интегрируется в облако, но нет собственных блокировок.

1. S3 / GCS / Azure Blob Storage

Lowest cost, highest latency

- **Latency** ⇒ Has the highest latency. p95 latency is also substantially worse than p50. In general you get results in the order of several hundred milliseconds
- **Scalability** ⇒ Infinite on storage, however, QPS will be limited by S3 concurrency limits
- **Cost** ⇒ Lowest (order of magnitude cheaper than other options)
- **Reliability/Availability** ⇒ Highly available, as blob storage like S3 are critical infrastructure that form the backbone of the internet.

Another important point to note is that LanceDB is designed to separate storage from compute, and the underlying Lance format stores the data in numerous immutable fragments. Due to these factors, LanceDB is a great storage option that addresses the $N + 1$ query problem. i.e., when a high query throughput is required, query processes can run in a stateless manner and be scaled up and down as needed.

Подробнее: <https://lancedb.github.io/lancedb/concepts/storage/>

3. **Разработка "тяжелых" мультимодальных моделей для сложного контента:**

- Мы берем 1 кадр из содержания видео и 7 для OCR. "Тяжелые" мультимодальные модели выдадут лучший результат для видео, в котором много разнородных эвентов и они все важны для понимания происходящего на видео

4. **Оптимизация векторного поиска с использованием GPU:**

- **Идея:** Перенести все модели и индекс IVF-PQ на GPU.
- **Гипотеза:** Это значительно ускорит процесс поиска и индексации.
- **Ожидаемые улучшения:** Существенное сокращение времени обработки запросов и повышение производительности системы.

5. **Интеграция дополнительных языковых моделей:**

- **Идея:** Интегрировать модели, которые хорошо работают с различными языками, помимо русского и английского.
- **Гипотеза:** Это расширит возможности системы и позволит обслуживать пользователей из разных регионов.
- **Ожидаемые улучшения:** Повышение доступности и универсальности системы для пользователей, говорящих на разных языках.

6. **Внедрение шаффла между визуальными и текстовыми эмбедингами:**

- **Идея:** Размывается смысл в итоговом векторе, если складывать AI-текст и содержимое воедино. Веса не помогли. Предлагаем оставить в продакшене FTS поиск по описаниям и включить его в шаффл-модель/модель классификации.

7. **Убрать дубликацию в видео.**

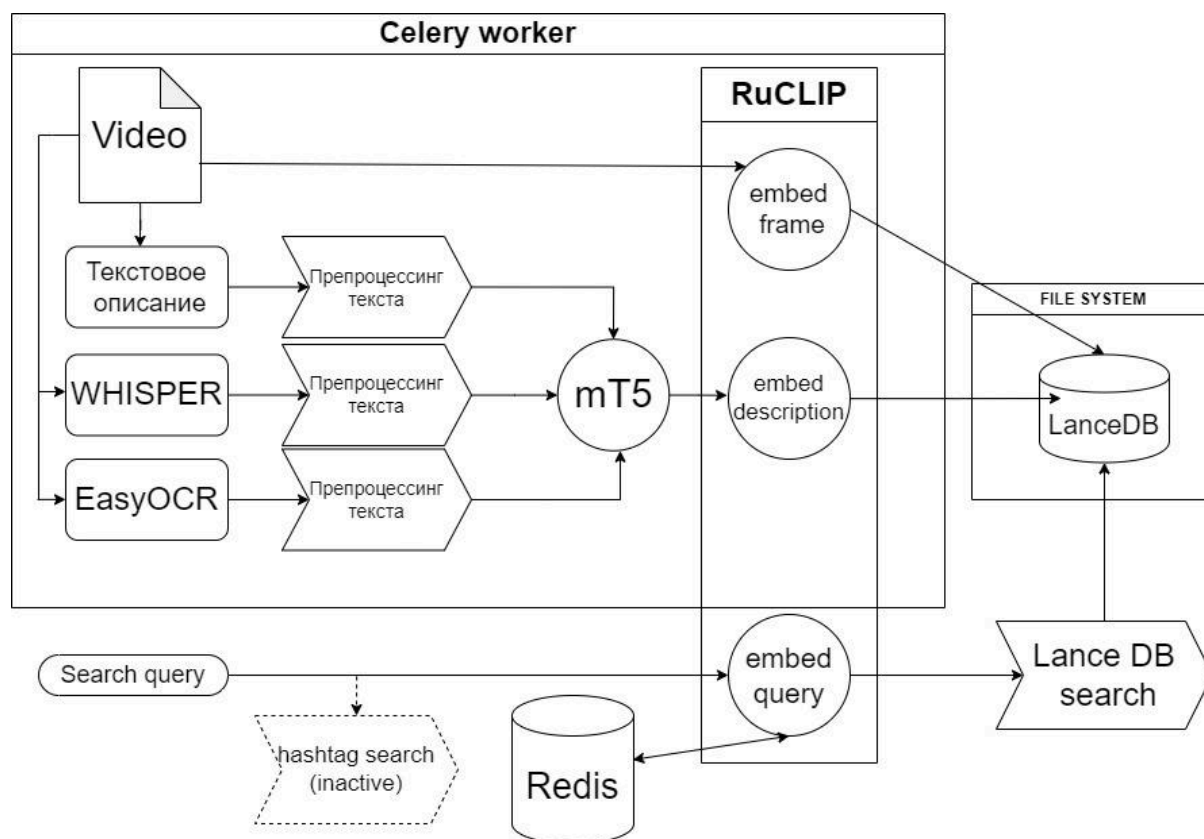
Также смотри презентацию в <https://gitlab.com/fizzzgen/video-indexer/> в папке **/Документация**

5. Схема логики взаимодействия всех модулей внутри проекта

Машина:

- 32 vCPU
- SSD
- 64 Gb RAM

Схема взаимодействий веб-сервиса:



Решение **GO_HACK** комбинирует визуальный и AI-текстовый поиск:

- Визуальный поиск по самому часто встречающейся сцене на видео.
- Текстовый — ищет по ключевым словам в описании, транскрибации аудио, по надписям/субтитрам и символам в видео.

Это позволяет быстро находить наиболее релевантные видео по запросу.

Визуальный поиск - поиск, основанный на содержащимся в видео (ruClip).
AI-текстовый - поиск, основанный на транскрибации аудио (Whisper), описании и EasyOCR. В LanceDB у нас два разных атрибута - два вектора AI-текстовый и визуальный. Далее из-этих результатов собирается результат из ТОП10 векторов.

Данное решение выбрано из-за разнородности видео в датасете и неудачных результатов взвешенной конкатенации данных видео из-за "размытия" смысла.

Весь текст очищается и препроцессится. Результат дополнительно суммаризируется mT5. EasyOCR берет 7 кадров, RuCLIP берет 1 кадр.
Репрезентативный адр для RuCLIP выбираем с помощью косинусного расстояния.

Querying an ANN Index

Querying vector indexes is done via the [search](#) function.

There are a couple of parameters that can be used to fine-tune the search:

- **limit** (default: 10): The amount of results that will be returned
- **nprobes** (default: 20): The number of probes used. A higher number makes search more accurate but also slower. Most of the time, setting nprobes to cover 5-10% of the dataset should achieve high recall with low latency. e.g., for 1M vectors divided up into 256 partitions, nprobes should be set to ~20-40. Note: nprobes is only applicable if an ANN index is present. If specified on a table without an ANN index, it is ignored.
- **refine_factor** (default: None): Refine the results by reading extra elements and re-ranking them in memory. A higher number makes search more accurate but also slower. If you find the recall is less than ideal, try refine_factor=10 to start. e.g., for 1M vectors divided into 256 partitions, if you're looking for top 20, then refine_factor=200 reranks the whole partition. Note: refine_factor is only applicable if an ANN index is present. If specified on a table without an ANN index, it is ignored.

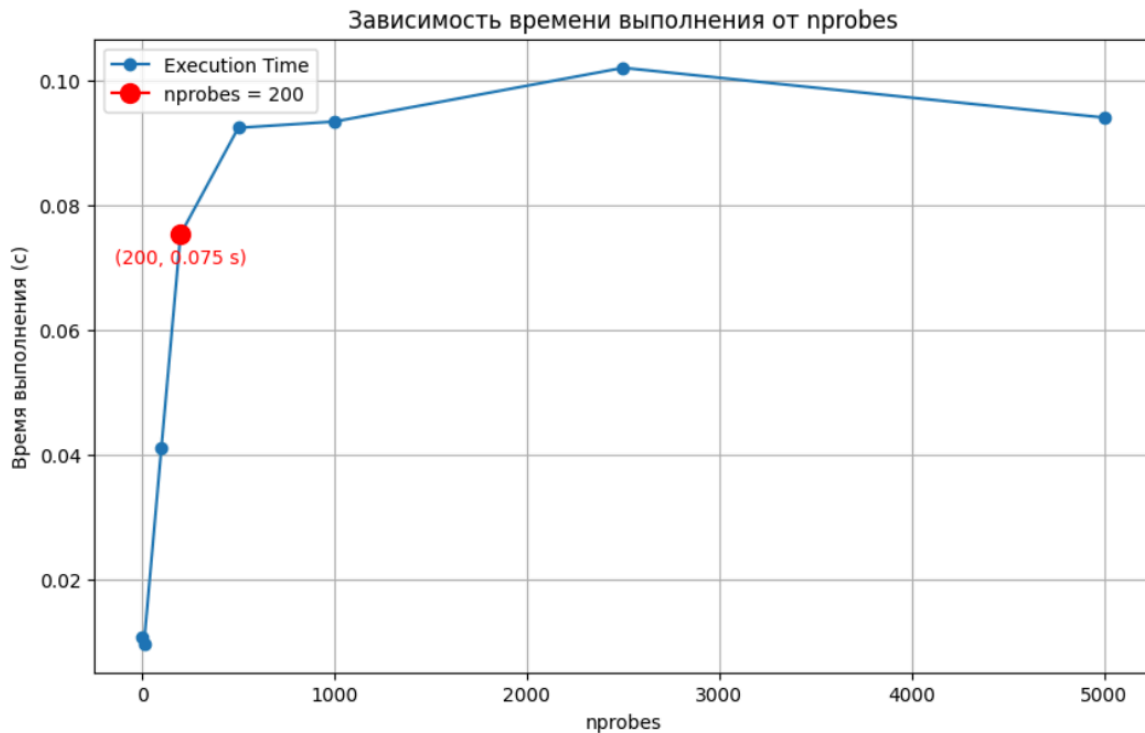
Python Typescript Rust

```
tbl.search(np.random.random((1536))) \
    .limit(2) \
    .nprobes(20) \
    .refine_factor(10) \
    .to_pandas()
```

	vector	item	_distance
0	[0.44949695, 0.84444449, 0.06281311, 0.23338133, ...]	item 1141	103.575333
1	[0.48587373, 0.269207, 0.15095535, 0.65531915, ...]	item 3953	100.393867

(nprobes, refine_factor - параметры в LanceDB, которые мы тюнили)

LanceDB хранится в файловой системе. С учетом SSD + zero-copy access + disk-based индекс/поиск получаем крайне высокую скорость обработки и индексации. Эта комбинация (zero-copy access + disk-based индекс/поиск) больше не встречается в open-source DB. Для ПРОДа рекомендуем облачные сервисы, как S3.



Redis используется для кэширования самых популярных эмбедингов.

1. Истоники: thedataquarry.com/posts/vector-db-3/
2. thesequence.substack.com/p/guest-post-choosing-the-right-vector
3. github.com/prrao87/lancedb-study
4. И наш ресерч по скорости на датасете с 1 миллионом векторов:
gitlab.com/fizzzgen/video-indexer/-/blob/main/test/DB_testing.ipynb
5. Смотри презентацию и питч для бенчмарков и метрик.. Они выложены в репозитории.
6. https://lancedb.github.io/lancedb/ann_indexes/#use-gpu-to-build-vector-index

6. Указание уникальности решения

1. Комбинированный подход к анализу видео:

- Мы используем как визуальный, так и текстовый анализ для получения наиболее точных и релевантных результатов поиска.
- **Визуальный анализ** осуществляется с помощью модели ruCLIP, которая генерирует векторные представления для ключевых кадров видео.
- **Текстовый анализ** включает в себя распознавание текста с помощью EasyOCR, транскрипцию аудио с помощью модели Whisper, и суммаризацию текстовых описаний.

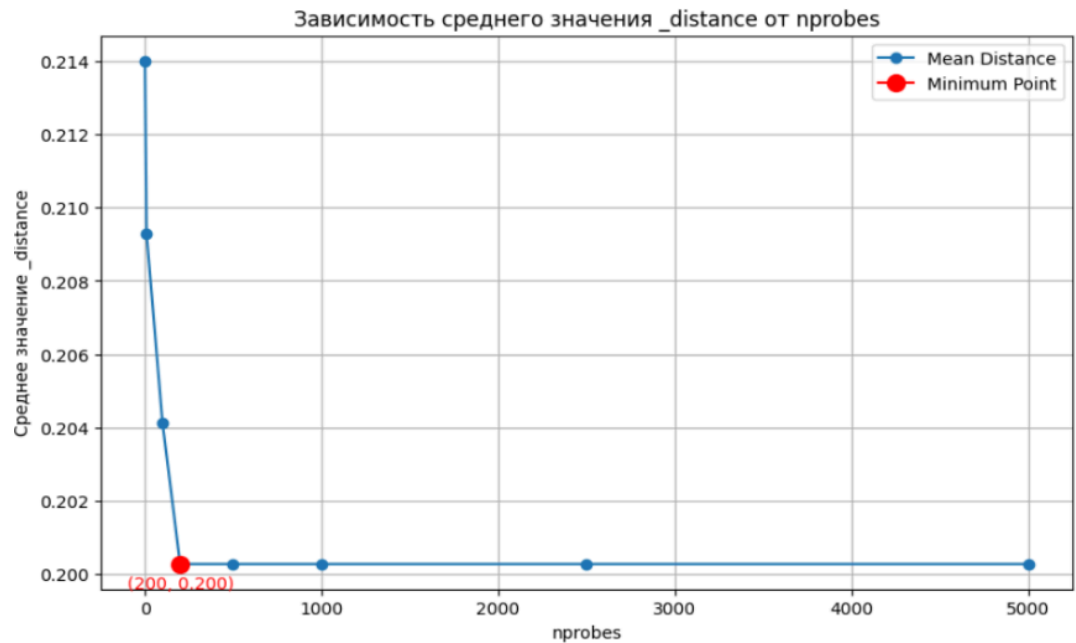
2. Эффективная векторная база данных (LanceDB):

- Использование базы данных LanceDB с индексом IVF-PQ позволяет значительно ускорить поиск по большим объемам данных.
- Среди всех конкурентов HNSW/RHNSW (PQ), IVF и FLAT IVF-PQ дает максимальный прирост скорости, но меньший recall при числе векторов.
- LanceDB оптимизирована для работы на диске и без сервера, что обеспечивает высокую производительность и минимальную настройку.
- Единственная БД с IVF-PQ.
- Единственная БД у которой все индексы disk-based (*) + zero-copy data access(**).

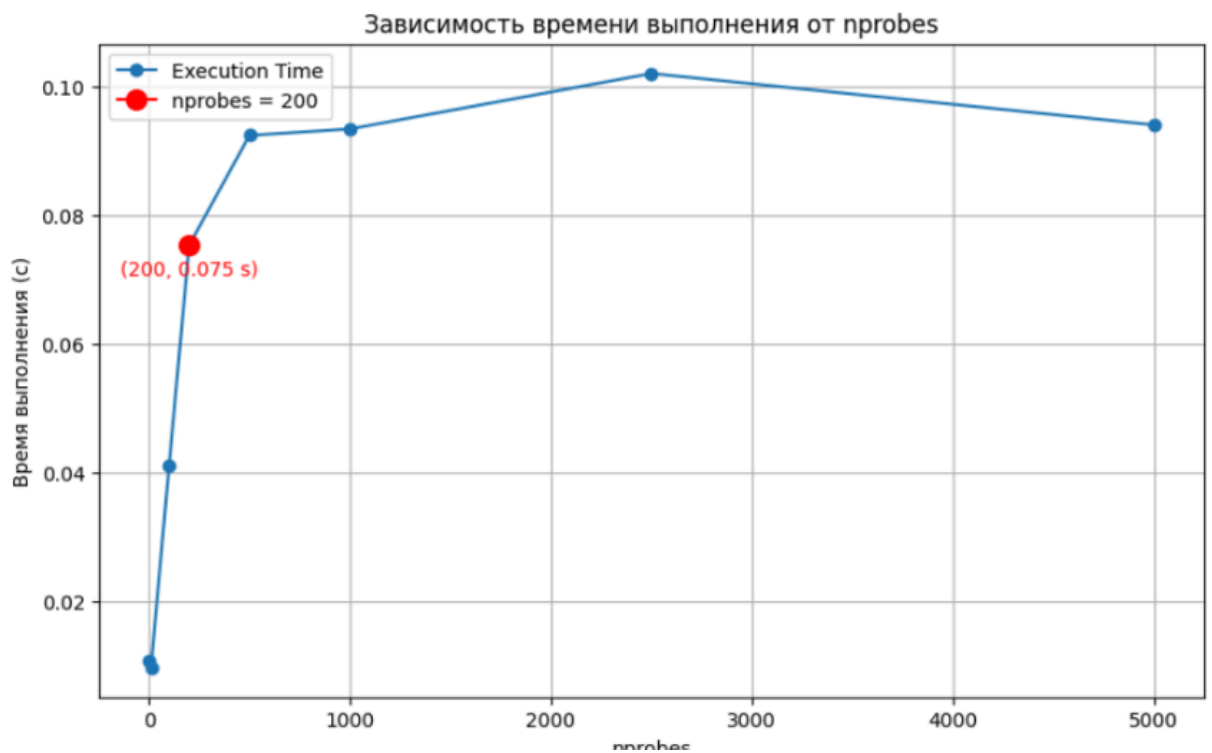
Источники: thedataquarry.com/posts/vector-db-3/ (*),
thesequence.substack.com/p/guest-post-choosing-the-right-vector (**)

3. Скорость и масштабируемость:

- Наше решение позволяет индексировать видео в ~10 раз быстрее и осуществлять поиск в ~2 раза быстрее по сравнению с регламентом, даже на посредственном оборудовании без GPU.
- Все модели и индекс IVF-PQ могут быть перенесены на GPU для дальнейшего увеличения скорости.
- На реальных данных (1 миллион записей), GPU и IVF-PQ индексе скорость поиска в БД будет от 0.075 sec при максимальной близости векторов (т.е. точности).



(Refine factor: 10; датасет: 1M; размерность: 512; T4 GPU; индекс: IVF-PQ.)



(Без индекса: 20 nprobes - 1.98s; с индексом: 200 nprobes - 0.075s;
создание индекса: 04 min 17 sec (CUDA))

4. Автономность и независимость от внешних API:

- Все модели установлены локально, что делает решение независимым от внешних API и обеспечивает высокую надежность.
- Использование Redis для кэширования эмбедингов запросов

дополнительно ускоряет обработку запросов.

5. **Когерентность эмбедингов:**

- Благодаря использованию одной CLIP-модели для всех типов данных (визуальных и текстовых), полученные эмбединги являются когерентными и обеспечивают высокую точность поиска.

6. **Разные языки:**

- Наше решение легко масштабируется и адаптируется под различные языки, включая русский и английский.
- Модели можно легко заменять на аналоги в пайплайне, что обеспечивает гибкость и возможность дальнейших улучшений.

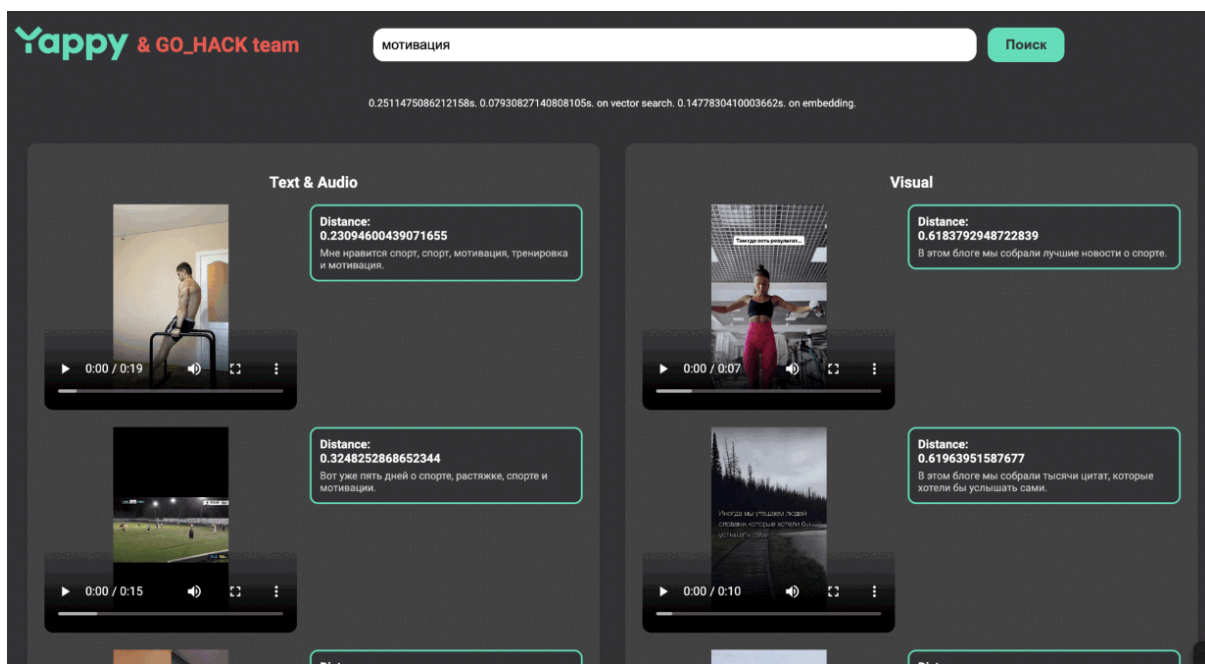
7. **Открытая лицензия и использование open-source решений:**

- Мы используем популярные и поддерживаемые open-source решения, такие как EasyOCR и LanceDB, что делает наше решение доступным и легко интегрируемым в различные системы.

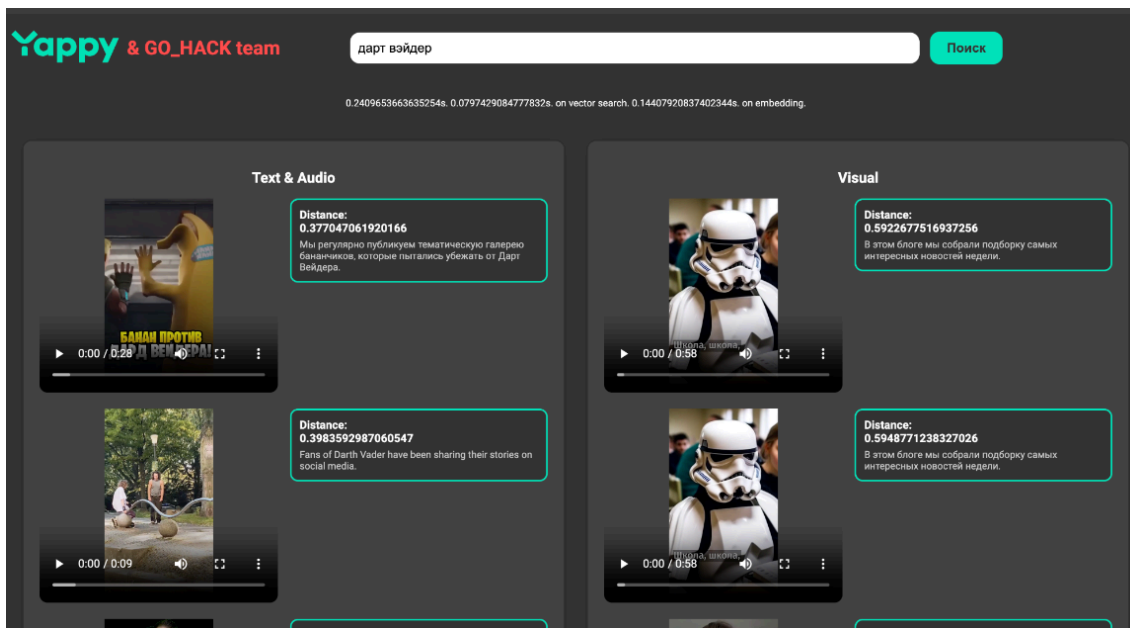
8. **Поддержка мультимодальных данных:**

- Решение эффективно работает с мультимодальными данными, комбинируя визуальные и текстовые признаки для достижения наилучших результатов.

9. **Справляемся со сложными запросами** - например, художественными персонажами и эмоциями.



Пример: Мотивация.



Пример: Дарт Вейдер.

Преимущества перед конкурентами

1. **Сравнение с ElasticSearch:**
 - ElasticSearch не подошел из-за более медленной обработки запросов для наших задач.

Сводка результатов по 10 000 случайных запросов.

Case	Elasticsearch (QPS)	LanceDB (QPS)
FTS: Serial (неактуален для решения)	399.8	<u>468.9</u>
FTS: Concurrent (неактуален для решения)	<u>1539.0</u>	528.9
Vector search: Serial	11.9	<u>54.0</u>
Vector search: Concurrent	50.7	<u>71.6</u>

Источник: github.com/prrao87/lancedb-study

2. **Сравнение с Milvus DB:**
 - Milvus DB не поддерживает IVF-PQ
3. **Сравнение с облачными решениями (QDRANT, Google Cloud Vision OCR, AWS Textract):**

- Облачные решения требуют платной подписки и зависят от сторонних API, что делает их менее привлекательными с точки зрения автономности и контроля.
- Наше решение автономно и не требует подключения к внешним сервисам и API, что делает его более надежным и экономически эффективным.