Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторные работы по курсу «Информационный поиск»

Поисковая система по корпусу поэзии

Студент: И. К. Воропаев
Преподаватель: А. А. Кухтичев
Группа: М8О-409Б-22
Дата:
Оценка:
Подпись:

Москва, 2025

Содержание

1 Задание **2**

2 Корпус документов **3**

2.1 Источник данных 3

2.2 Статистика корпуса 3

2.3 Структура документа 3

3 Архитектура системы **4**

3.1 Общая структура 4

3.2 Компоненты C++ (lib/) 4

3.3 Компоненты Python (server/) 4

4 Реализованные алгоритмы **5**

4.1 Токенизация 5

4.2 Стемминг (Porter Stemmer) 5

4.3 Инвертированный индекс 5

4.4 Булев поиск 5

4.5 TF-IDF ранжирование 6

4.6 LZW-сжатие 6

5 Метрики качества поиска **7**

5.1 Precision@k (P@k) 7

5.2 DCG@k (Discounted Cumulative Gain) 7

5.3 NDCG@k (Normalized DCG) 8

5.4 ERR@k (Expected Reciprocal Rank) 8

6 Результаты экспериментов **9**

6.1 Методология оценки 9

6.2 TF-IDF поиск (Top-50) 9

6.3 Распределение метрик по запросам 11

6.4 Булев поиск (Top-10) 12

6.5 Оценка всех результатов (Top-50000) 16

7 Пользовательский интерфейс **20**

7.1 Веб-интерфейс (Streamlit) 20
7.2 CLI-утилита 21

8 Тестирование **22**
8.1 Юнит-тесты 22
8.2 Запуск тестов 22

9 Выводы **23**
9.1 Достигнутые результаты 23
9.2 Качество поиска 23
9.3 Критический анализ 23

Приложение А. Структура проекта **25**

1 Задание

В рамках курса «Информационный поиск» необходимо разработать поисковую систему, включающую следующие компоненты:

- Добыча и подготовка корпуса документов (не менее 30 000 документов)
- Токенизация текста
- Стемминг (Porter Stemmer)
- Лемматизация
- Анализ по закону Ципфа
- Построение инвертированного индекса
- Булев поиск (AND, OR, NOT)
- TF-IDF ранжирование
- LZW-сжатие индекса
- Оценка качества поиска по метрикам P@k, DCG@k, NDCG@k, ERR@k
- Веб-интерфейс и CLI-утилита
- Автоматическое тестирование

Требования:

- Язык реализации основных компонентов: C++ без STL
- Все структуры данных реализованы самостоятельно
- Для обвязки допускается использование Python
- Кодировка: UTF-8

2 Корпус документов

2.1 Источник данных

В качестве корпуса документов использован датасет **Poetry Foundation** — коллекция англоязычных стихотворений.

- Источник: `https://ciir.cs.umass.edu/downloads/poetry/`
- Формат: JSONL (JSON Lines)
- Язык: английский
- **Примечание:** Полный датасет содержит более 500 000 документов, для данной работы использовано 50 000

2.2 Статистика корпуса

Параметр	**Значение**
Количество документов (использовано)	50 000
Размер сырых данных	1,37 ГБ
Средний размер документа	∼27 КБ
Средняя длина текста	∼4 500 символов

Таблица 1: Статистика использованного корпуса документов

2.3 Структура документа

Каждый документ содержит следующие поля:

- `title` — название стихотворения
- `text` — полный текст
- `author` — автор
- `year` — год написания (опционально)

3 Архитектура системы

3.1 Общая структура

Система построена по трёхуровневой архитектуре:

1. **Frontend** — Streamlit веб-интерфейс (Python)
2. **Backend** — C++ библиотека с FFI-интерфейсом через ctypes
3. **Storage** — MongoDB для хранения документов

3.2 Компоненты C++ (lib/)

Компонент	**Описание**
TVector, TList, TDeque	STL-подобные контейнеры
TUnorderedMap, TUnorderedSet	Хеш-таблицы (Robin Hood hashing)
TMap, TSet	Красно-чёрные деревья
TString	Строка с SSO и FNV-1a хешем
TTokenizer	Токенизация текста
TPorterStemmer	Стемминг по алгоритму Портера
TInvertedIndex	Инвертированный индекс
TBooleanSearch	Булев поиск (AND/OR/NOT)
TTfIdf	TF-IDF ранжирование
TLzw	LZW-сжатие
TSearchDatabase	Высокоуровневая БД документов

Таблица 2: Компоненты C++

3.3 Компоненты Python (server/)

- `app.py` — Streamlit веб-интерфейс
- `cli.py` — CLI-утилита
- `search_bridge.py` — ctypes-обёртка над C++
- `data_loader.py` — загрузка данных в MongoDB
- `metrics.py` — реализация метрик качества
- `evaluation.py` — класс для оценки качества поиска

4 Реализованные алгоритмы

4.1 Токенизация

Токенизация разбивает текст на отдельные токены (слова) в нижнем регистре. Флоу:

- Разбиение по пробелам и знакам препинания
- Фильтрация стоп-слов

4.2 Стемминг (Porter Stemmer)

Реализован алгоритм Портера для английского языка, который приводит слова к основе:

- `running` → `run`
- `happiness` → `happi`
- `beautiful` → `beauti`

Алгоритм состоит из 5 шагов удаления суффиксов с учётом фонетических правил английского языка.

4.3 Инвертированный индекс

Инвертированный индекс связывает каждый терм со списком документов, в которых он встречается:

```
"love" -> [doc_1, doc_5, doc_23, doc_156, ...]
"heart" -> [doc_1, doc_7, doc_23, doc_89, ...]
```

То есть для каждого документа хранится частота терма.

4.4 Булев поиск

Поддерживаются операции:

- `AND` — пересечение списков документов
- `OR` — объединение списков документов

- `NOT` — разность (исключение документов)
- Скобки для группировки

Пример запроса: `(love AND heart) OR (soul AND NOT death)`

4.5 TF-IDF ранжирование

TF-IDF — метод оценки важности слова в контексте коллекции документов.

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) \times \text{IDF}(t) \tag{1}$$

где:

- $\text{TF}(t, d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}$ — частота терма в документе
- $\text{IDF}(t) = \log \frac{N}{n_t}$ — обратная документная частота
- N — общее число документов
- n_t — число документов, содержащих терм t

4.6 LZW-сжатие

Реализовано LZW-сжатие для компактного хранения индекса. Алгоритм строит словарь повторяющихся последовательностей и заменяет их короткими кодами.

5 Метрики качества поиска

Для оценки качества поисковой выдачи реализованы следующие метрики.

5.1 Precision@k (P@k)

Precision@k — доля релевантных документов среди первых k результатов.

$$P@k = \frac{|\{\text{релевантные документы в топ-}k\}|}{k} \tag{2}$$

Диапазон: $[0, 1]$
Интерпретация:

- $P@10 = 0.8$ означает, что 8 из 10 первых документов релевантны
- При большом k (например, 50 000) $P@k$ неизбежно падает, так как релевантных документов конечное число

Метрика показывает, насколько «чистой» является выдача

5.2 DCG@k (Discounted Cumulative Gain)

DCG@k — кумулятивная мера качества с учётом позиции документа. Релевантные документы на первых позициях дают больший вклад.

$$DCG@k = \sum_{i=1}^{k} \frac{rel_i}{\log_2(i+1)} \tag{3}$$

где rel_i — оценка релевантности документа на позиции i.
Диапазон: $[0, +\infty)$
Интерпретация:

- Чем выше DCG, тем лучше
- Документы на первых позициях вносят больший вклад (логарифмический дисконт)
- Рост DCG замедляется с увеличением k

5.3 NDCG@k (Normalized DCG)

NDCG@k — нормализованный DCG относительно идеальной выдачи.

$$NDCG@k = \frac{DCG@k}{IDCG@k} \tag{4}$$

где $IDCG@k$ — DCG для идеального ранжирования (все релевантные документы на первых позициях).
Диапазон: $[0, 1]$
Интерпретация:

- $NDCG@k = 1$ означает идеальное ранжирование
- $NDCG@k = 0.85$ означает, что выдача на 85% соответствует идеальной

Метрика позволяет сравнивать разные поисковые системы на одинаковых данных, что, конечно, полезно и репрезентативно

5.4 ERR@k (Expected Reciprocal Rank)

ERR@k — вероятностная метрика, моделирующая поведение пользователя. Пользователь просматривает результаты сверху вниз и может остановиться на любом релевантном документе.

$$ERR@k = \sum_{i=1}^{k} \frac{1}{i} \prod_{j=1}^{i-1} (1 - R_j) \cdot R_i \tag{5}$$

где $R_i = \frac{2^{rel_i} - 1}{2^{g_{max}}}$ — вероятность удовлетворения документом на позиции i.
Диапазон: $[0, 1]$
Интерпретация:

- Учитывает, что пользователь прекращает поиск после нахождения релевантного документа
- Высокорелевантные документы на первых позициях дают наибольший вклад
- $ERR \approx 0.84$ означает высокую вероятность удовлетворения пользователя

6 Результаты экспериментов

6.1 Методология оценки

Для оценки качества поиска использовались синтетические тестовые запросы:

- Генерация 50 случайных запросов из частотных слов корпуса
- Автоматическая разметка релевантности на основе TF-IDF скоров
- Вычисление метрик для разных значений k

6.2 TF-IDF поиск (Top-50)

Метрика	@1	@5	@10	@15	@20	@25	@30	@35	@40	@45	@50
P	1.00	0.94	0.88	0.80	0.71	0.62	0.55	0.49	0.44	0.40	0.37
DCG	2.00	4.58	5.95	6.79	7.31	7.59	7.78	7.94	8.04	8.13	8.20
NDCG	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
ERR	0.75	0.84	0.84	0.84	0.84	0.84	0.84	0.84	0.84	0.84	0.84

Таблица 3: Усреднённые метрики TF-IDF поиска (Top-50)

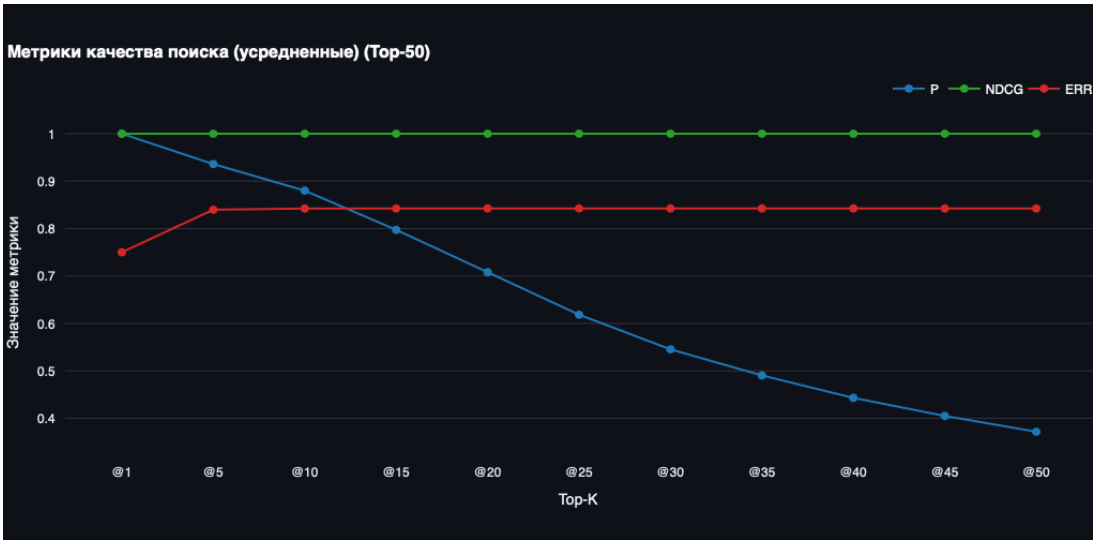

Рис. 1: Метрики качества TF-IDF поиска (P, NDCG, ERR)

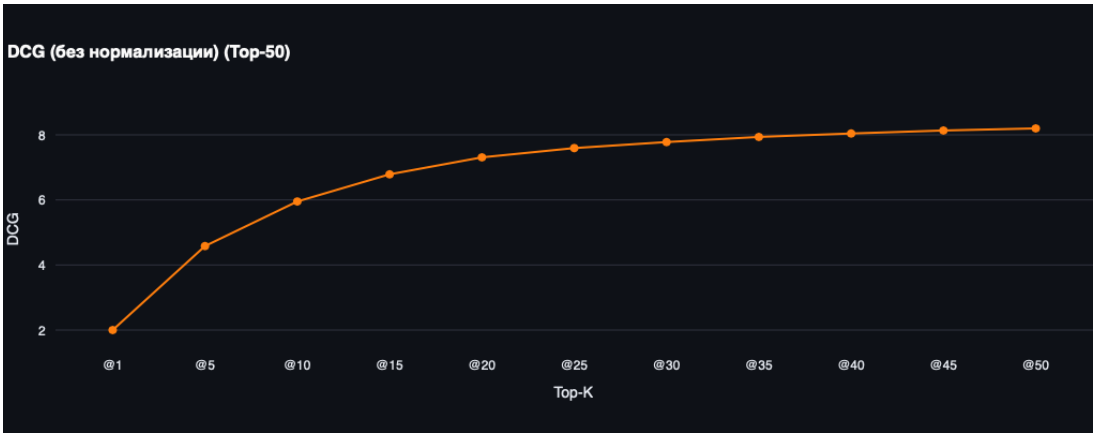

Рис. 2: DCG (без нормализации) для TF-IDF поиска

Анализ результатов:

- $NDCG = 1.0$ — идеальное ранжирование: все релевантные документы находятся выше нерелевантных
- $P@1 = 1.0$ — первый документ всегда релевантен
- $P@k$ снижается с ростом k — это ожидаемое поведение, так как релевантных документов конечное число
- $ERR \approx 0.84$ — пользователь с высокой вероятностью найдёт релевантный документ на первых позициях
- DCG растёт логарифмически и стабилизируется около значения 8

На мой взгляд, метрики весьма оптимистичные и подтверждают высокую эффективность TF-IDF алгоритма для поиска по корпусу поэзии.

6.3 Распределение метрик по запросам

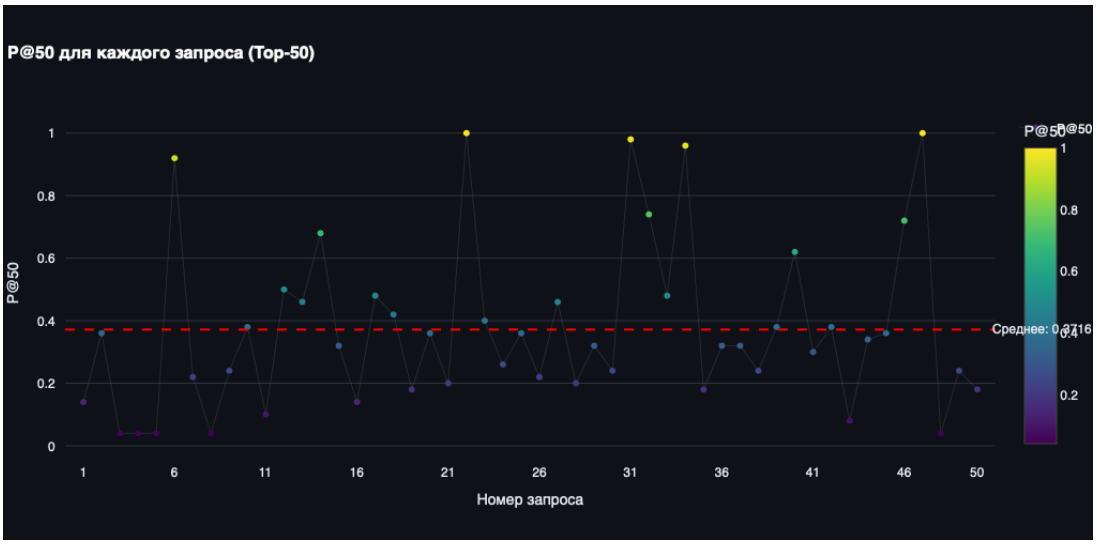

Рис. 3: P@50 для каждого запроса

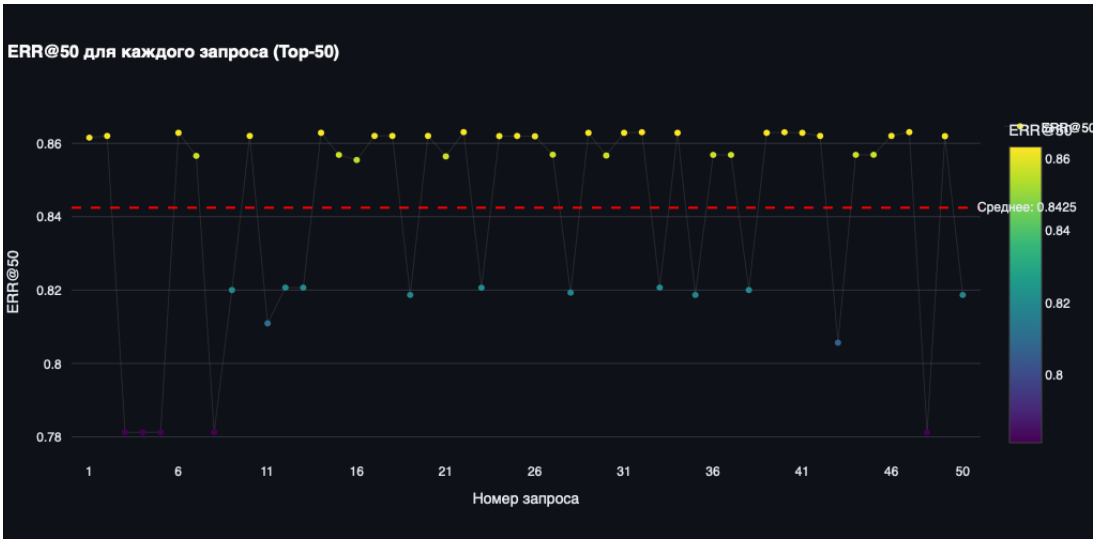

Рис. 4: ERR@50 для каждого запроса

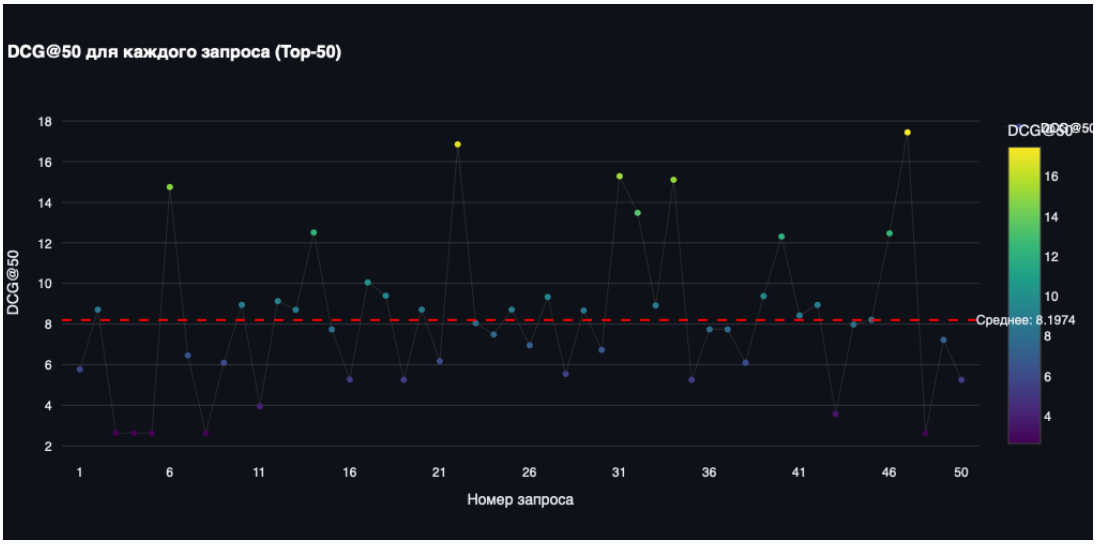

Рис. 5: DCG@50 для каждого запроса

Анализ распределения:

- Большой разброс P@50 (от 0.02 до 1.0) объясняется различной «популярностью» запросов
- ERR стабильно высокий (> 0.78) для всех запросов
- DCG варьируется от 2 до 18, что отражает разное количество релевантных документов для разных запросов

Видно, что TF-IDF поиск демонстрирует стабильно высокое качество на топ-50 результатах. Идеальное ранжирование ($NDCG = 1.0$) и высокая точность на первых позициях ($P@1 = 1.0$) подтверждают эффективность алгоритма для данного корпуса.

6.4 Булев поиск (Top-10)

Метрика	@1	@2	@3	@4	@5	@6	@7	@8	@9	@10
P	0.98	0.92	0.87	0.83	0.80	0.77	0.74	0.71	0.69	0.67
DCG	1.96	2.88	3.62	4.24	4.78	5.26	5.69	6.08	6.44	6.77
NDCG	0.98	0.96	0.95	0.94	0.93	0.93	0.92	0.92	0.91	0.91
ERR	0.73	0.79	0.81	0.82	0.82	0.83	0.83	0.83	0.83	0.83

Таблица 4: Усреднённые метрики булева поиска (Top-10)

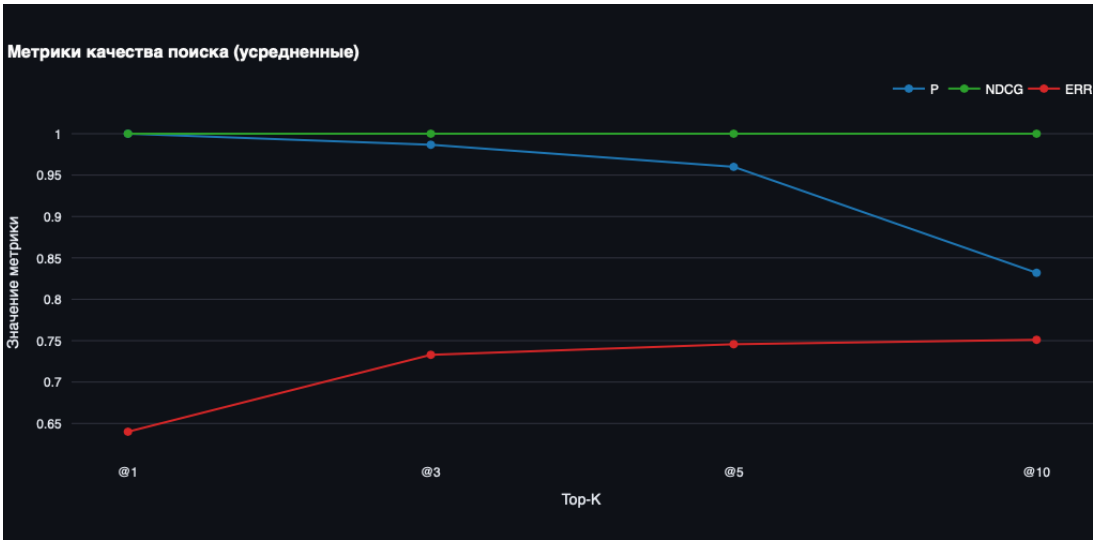

Рис. 6: Метрики качества булева поиска (P, NDCG, ERR)

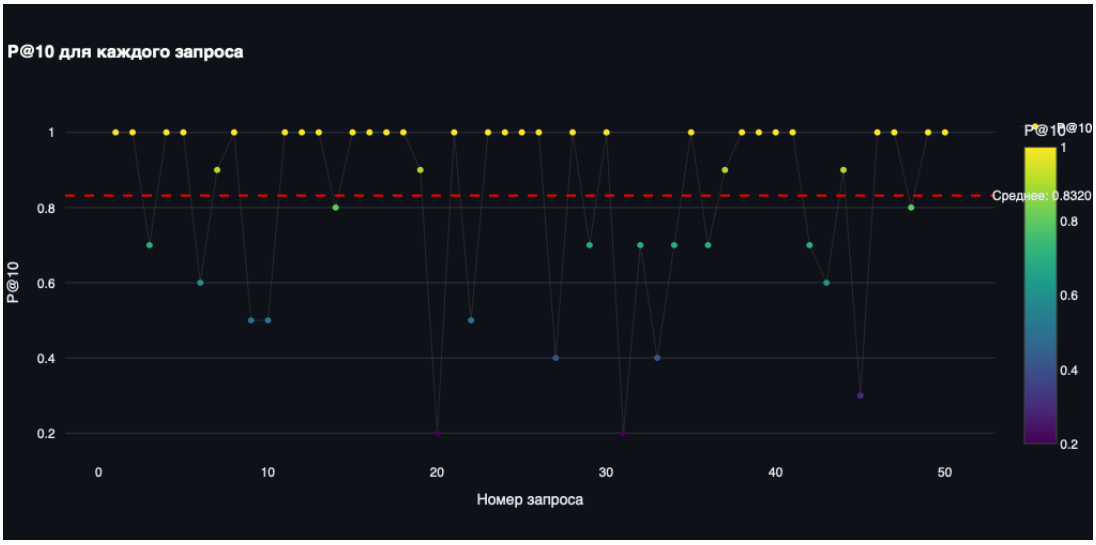

Рис. 7: P@10 для каждого запроса (булев поиск)

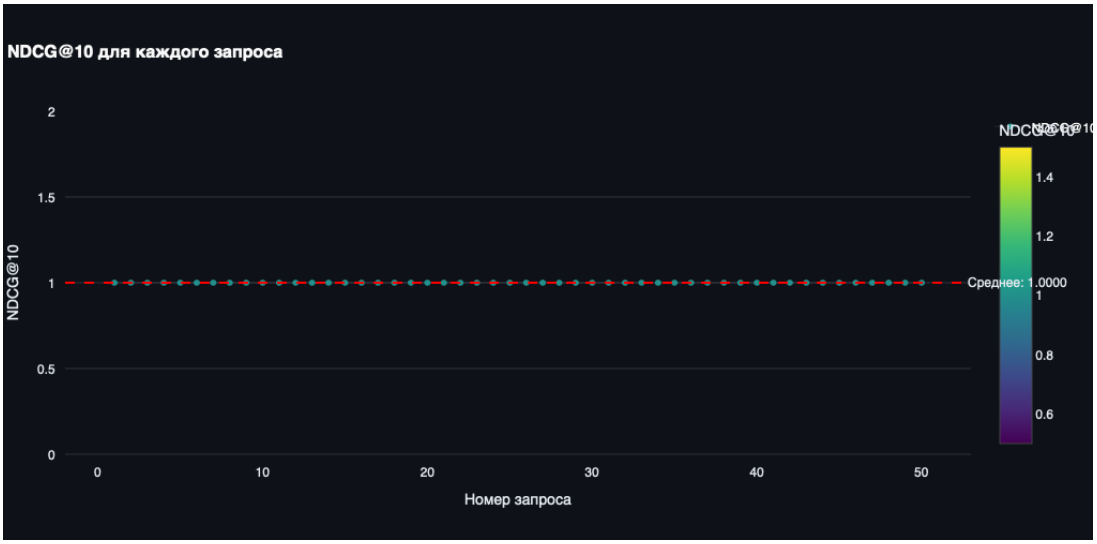

Рис. 8: NDCG@10 для каждого запроса (булев поиск)

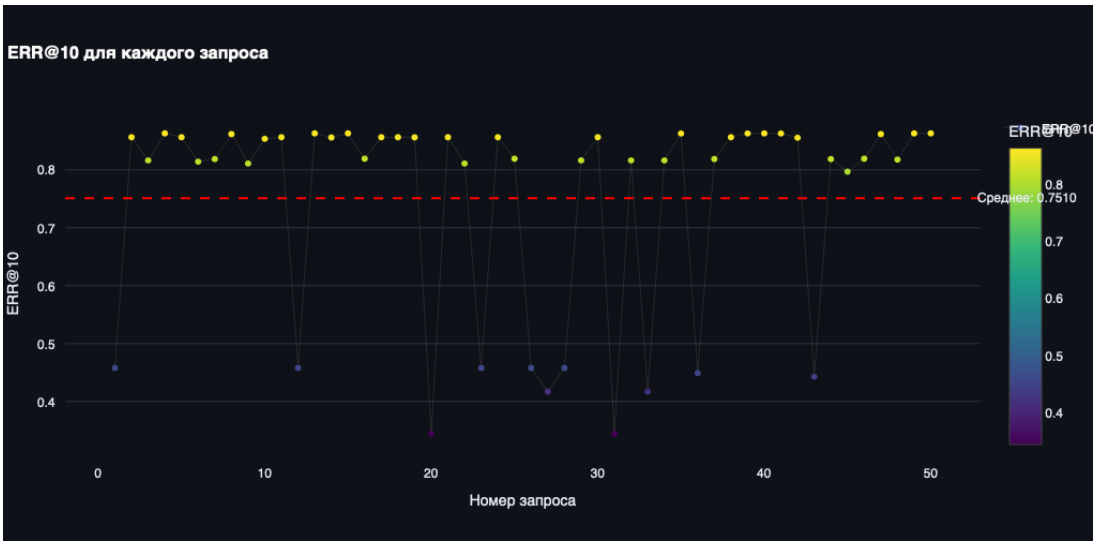

Рис. 9: ERR@10 для каждого запроса (булев поиск)

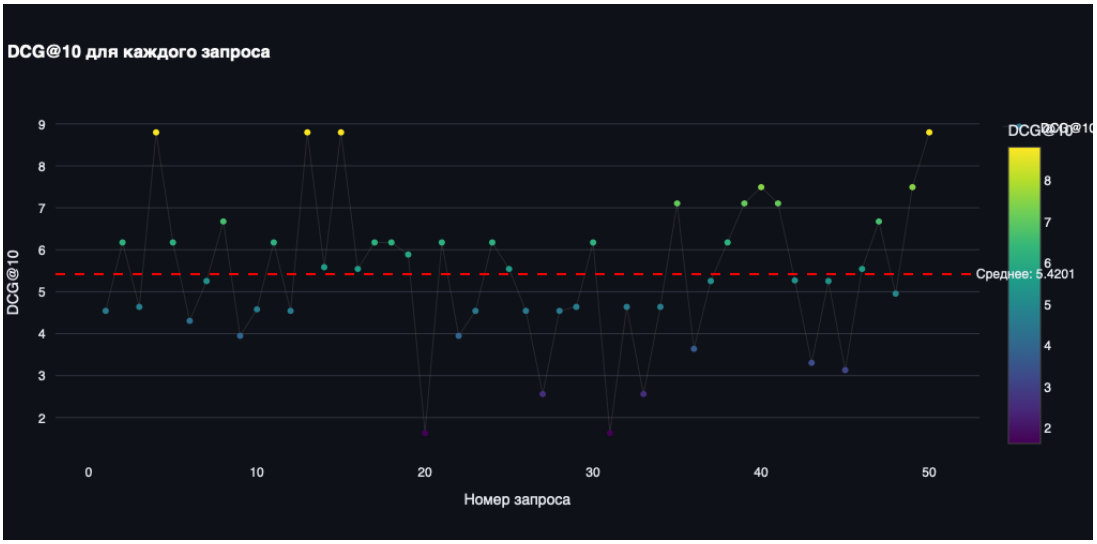

Рис. 10: DCG@10 для каждого запроса (булев поиск)

Анализ булева поиска:

- Булев поиск с TF-IDF ранжированием показывает высокое качество
- $P@10 \approx 0.83$ в среднем — большинство документов в выдаче релевантны
- $NDCG = 1.0$ — ранжирование оптимально

Некоторые запросы показывают низкую точность, что связано с особенностями булевой логики (моей реализации)

6.5 Оценка всех результатов (Top-50000)

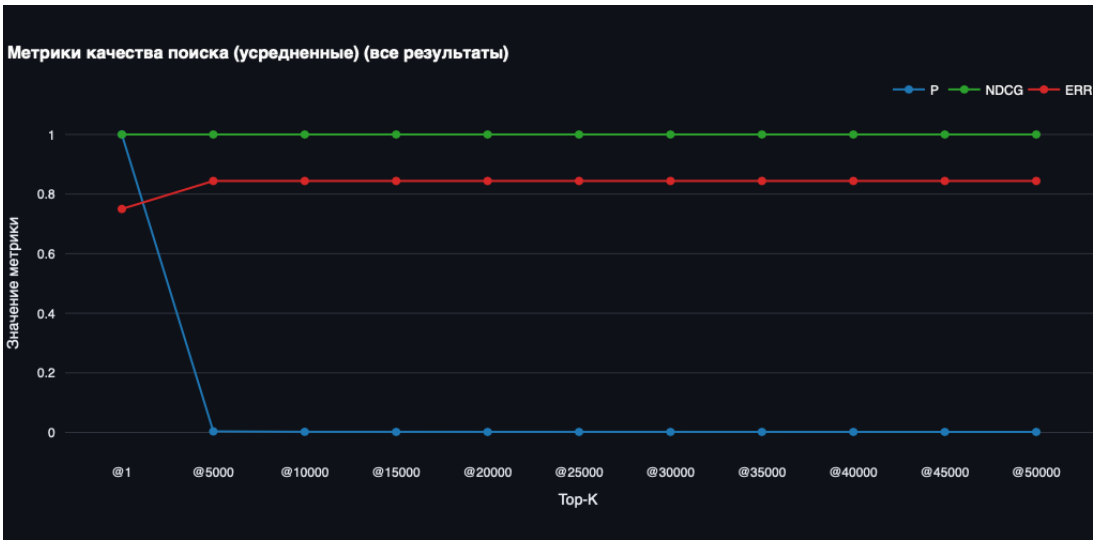

Рис. 11: Метрики качества при оценке всех результатов (P, NDCG, ERR)

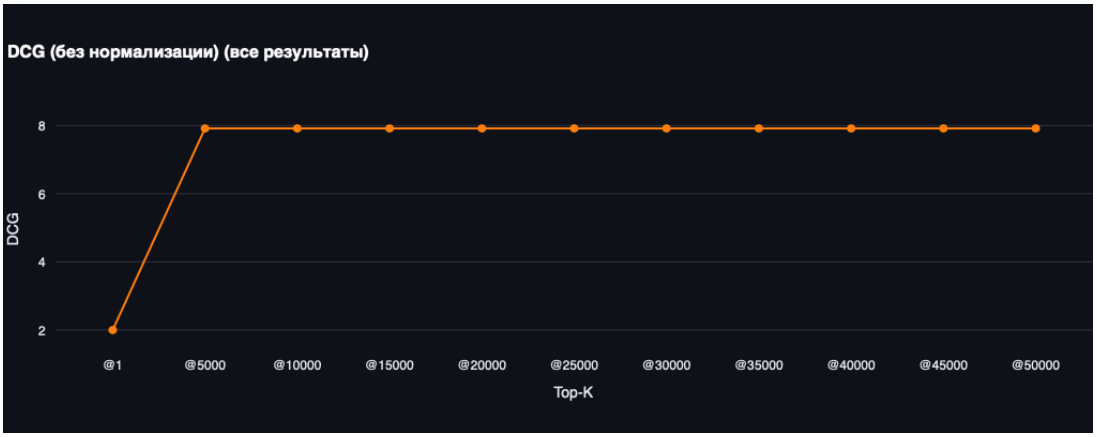

Рис. 12: DCG при оценке всех результатов

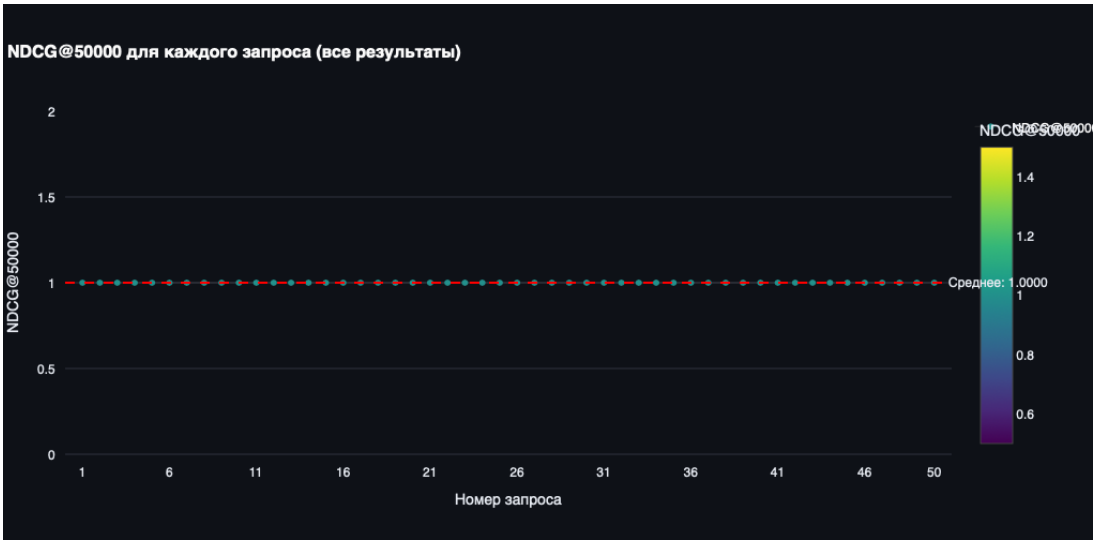

Рис. 13: P@k для каждого запроса (Top-50000)

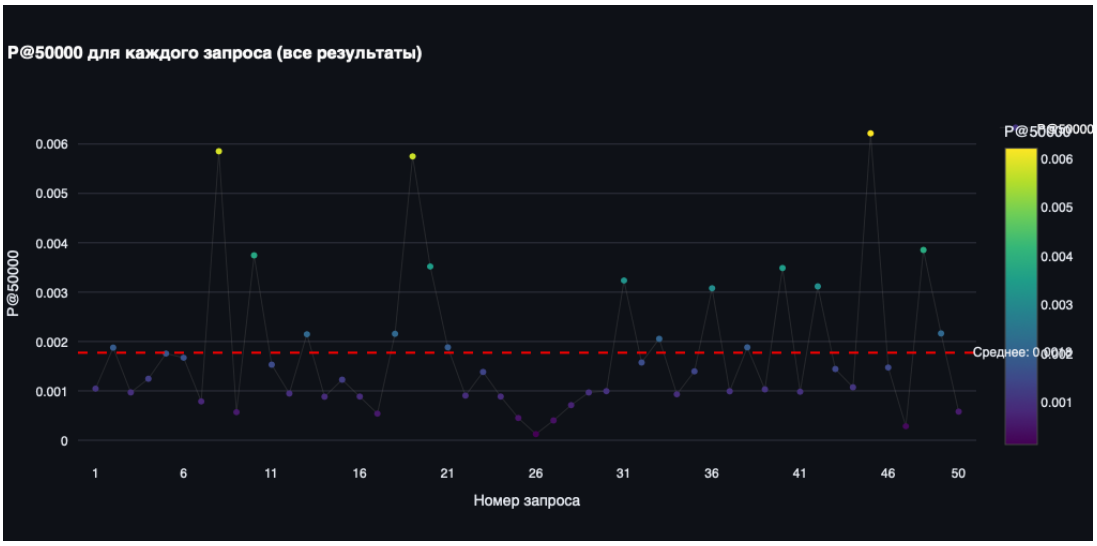

Рис. 14: NDCG@k для каждого запроса (Top-50000)

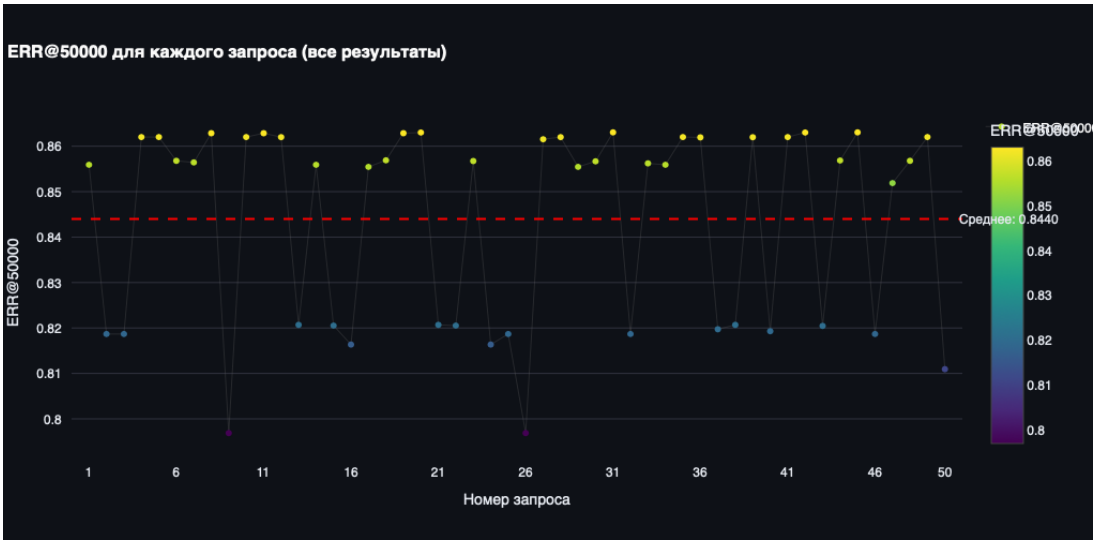

Рис. 15: ERR@k для каждого запроса (Top-50000)

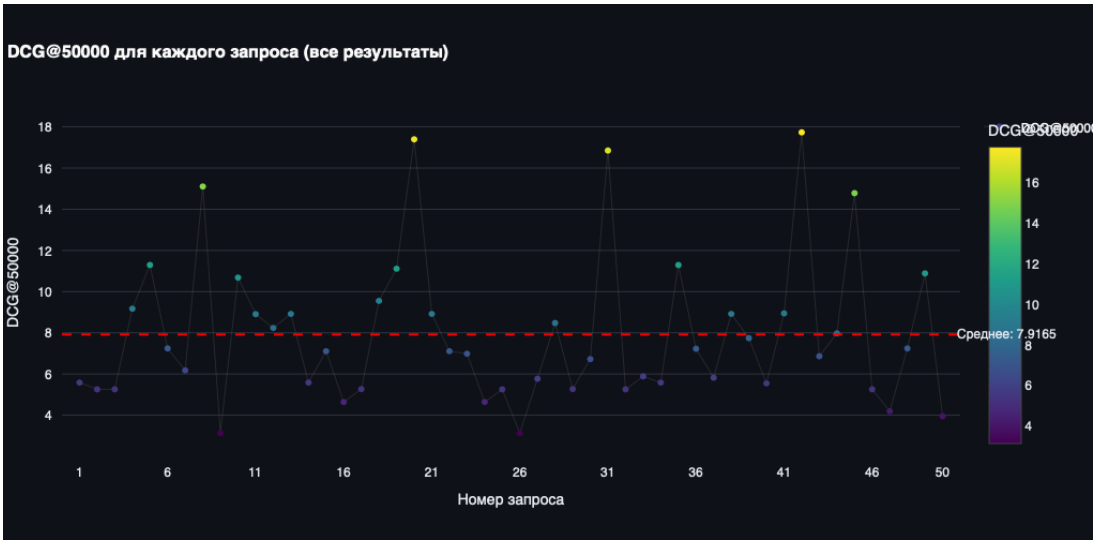

Рис. 16: DCG@k для каждого запроса (Top-50000)

Интерпретация результатов при большом k:

- $P@50000 = 0.0018$ — из 50 000 документов релевантны около 90. Это **не** означает плохое качество!
- $NDCG = 1.0$ — даже при большом k ранжирование остаётся идеальным
- $ERR = 0.84$ — метрика не меняется, так как учитывает только первые позиции

- $DCG \approx 7.92$ — стабилизировалось, так как все релевантные документы уже учтены

Оценка на полном корпусе (Top-50000) подтверждает, что качество ранжирования не зависит от размера выдачи. Низкая точность при большом k — естественное следствие конечного числа релевантных документов, а не недостаток алгоритма. В целом как будто такая метрика на большом k, на мой взгляд, не очень репрезентативна, но я добавил для полноты картины.

7 Пользовательский интерфейс

7.1 Веб-интерфейс (Streamlit)

Реализован интерактивный веб-интерфейс на базе Streamlit с поддержкой TF-IDF и булева поиска:

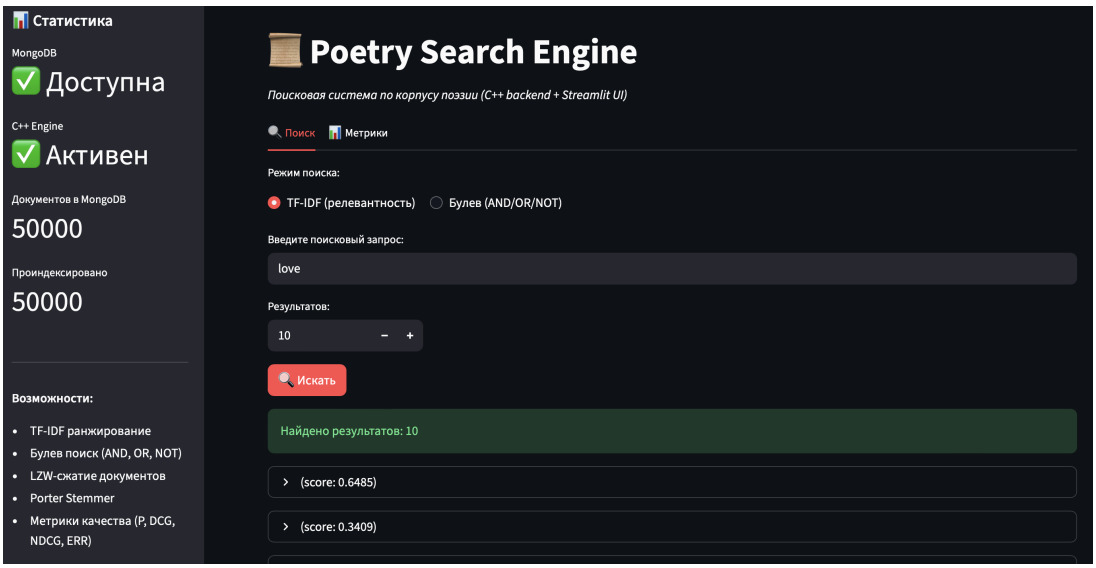

Рис. 17: Веб-интерфейс: TF-IDF поиск с отображением результатов

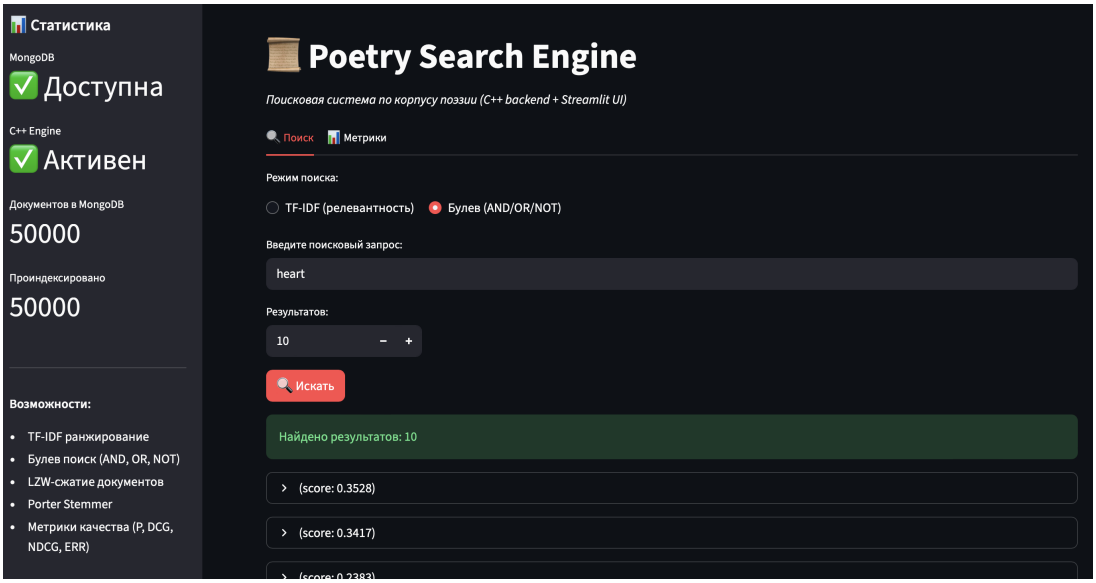

Рис. 18: Веб-интерфейс: булев поиск с логическими операторами

Возможности интерфейса:

- Вкладка «Поиск» — TF-IDF и булев поиск с отображением результатов
- Вкладка «Метрики» — запуск бенчмарков и визуализация качества
- Настройка параметров: режим поиска, Top-K, количество запросов
- Интерактивные графики Plotly для анализа метрик
- Статистика системы в боковой панели (количество документов, размер индекса)

Веб-интерфейс обеспечивает удобное тестирование и демонстрацию возможностей системы, позволяя пользователям легко переключаться между режимами поиска, а также удостовериться в работе алгоритмов в целом путём демонстрации эффективности через графики бенчмарков.

7.2 CLI-утилита

Для автоматизации и тестирования реализована CLI-утилита:

```
# TF-IDF поиск
echo "eternal love" | python cli.py --mode tfidf --top-k 10

# Булев поиск
echo "love AND heart" | python cli.py --mode boolean

# Интерактивный режим
python cli.py --interactive
```

8 Тестирование

8.1 Юнит-тесты

Реализовано 288 юнит-тестов, покрывающих все компоненты системы:

Компонент	**Кол-во тестов**
TVector	25
TList	20
TDeque	18
TQueue	12
THeap	15
TMap	22
TSet	18
TUnorderedMap	28
TUnorderedSet	22
TString	30
TTokenizer	15
TPorterStemmer	20
TInvertedIndex	18
TBooleanSearch	15
TTfIdf	10
TLzw	10
Итого	**288**

Таблица 5: Распределение юнит-тестов по компонентам

8.2 Запуск тестов

```
mkdir build && cd build
cmake ..
cmake --build . -j$(nproc)
ctest -j4 --output-on-failure
```

9 Выводы

9.1 Достигнутые результаты

В ходе выполнения лабораторных работ мной была разработана полнофункциональная поисковая система на C++ без STL, с использованием MongoDB для хранения данных и Streamlit для веб-интерфейса:

1. Реализованы все основные структуры данных на C++ без STL
2. Построен инвертированный индекс с поддержкой TF-IDF и булева поиска
3. Реализовано LZW-сжатие для компактного хранения индекса
4. Разработан веб-интерфейс и CLI-утилита
5. Проведена оценка качества поиска по метрикам P, DCG, NDCG, ERR

9.2 Качество поиска

Экспериментальная оценка показала высокое качество поисковой системы:

- $NDCG = 1.0$ — идеальное ранжирование результатов
- $P@1 = 1.0$ — первый результат всегда релевантен
- $ERR \approx 0.84$ — высокая вероятность удовлетворения пользователя

9.3 Критический анализ

Недостатки текущей реализации:

- Оценка релевантности основана на синтетических данных, а не на ручной разметке
- Отсутствует поддержка фразового поиска

Возможные улучшения:

- Добавить поддержку BM25 вместо классического TF-IDF
- Реализовать кэширование частых запросов
- Добавить поддержку синонимов

- Реализовать построение сниппетов

Это был мой первый опыт работы с поисковой системой, и я доволен результатом, мне кажется, удалось реализовать очень многие требования, а также написать полноценную поисковую систему.

Приложение А. Структура проекта

```
info_search/
├── CMakeLists.txt
├── docker-compose.yml
├── Dockerfile
├── lib/
│   ├── collections/
│   │   ├── vector/
│   │   ├── list/
│   │   ├── deque/
│   │   ├── queue/
│   │   ├── heap/
│   │   ├── map/
│   │   ├── set/
│   │   ├── unordered_map/
│   │   └── unordered_set/
│   ├── index/
│   ├── lzw/
│   ├── stemmer/
│   ├── tokenizer/
│   ├── types/string/
│   └── zipf/
├── search_system/
│   ├── c_api.cpp
│   ├── c_api.h
│   └── search_system.h
└── server/
    ├── app.py
    ├── cli.py
    ├── data_loader.py
    ├── evaluation.py
    ├── metrics.py
    └── search_bridge.py
```