МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**
Кафедра математической кибернетики и компьютерных наук

ПОСТРОЕНИЕ СИСТЕМЫ ПОЛНОТЕКСТОВОГО ПОИСКА ПО ARXIV

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 4 курса 451 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Чернигина Михаила Андреевича

Научный руководитель
доцент _____ Б. А. Филиппов

Заведующий кафедрой
доцент, к. ф.-м. н. _____ С. В. Миронов

Саратов 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 3
1 Устройство поисковых систем 4
2 Оптимизация поисковых систем 5
3 Общее описание архитектуры реализуемой поисковой системы 6
3.1 Сбор данных 6
3.1.1 Клиент-серверная архитектура 6
3.2 Сервис вычисления эмбеддингов 7
4 Реализация сбора данных 8
4.1 Реализация сервера управления задачами сбора данных 8
4.2 Реализация клиентов сбора данных 8
4.2.1 Извлечение текста из pdf 8
5 Построение поискового индекса 9
6 Веб интерфейс 11
7 Оценка результатов 12
ЗАКЛЮЧЕНИЕ 13

ВВЕДЕНИЕ

Со временем количество накопленных человечеством знаний в виде научных работ не перестаёт расти, а наоборот, только набирает темп. Раньше для хранения этих знаний создавались огромные библиотеки, но с развитием информационных технологий и появлением интернета знания стало намного удобнее хранить в цифровой форме.

Умение ориентироваться в таких огромных массивах информации и находить нужное, стало для нас крайне важным аспектом жизни. Для решения этой задачи люди начали создавать поисковые системы и по итогу ими пользуется большая часть человечества.

Одним из самых крупных и известных хранилищ препринтов и научных работ является сервис arXiv.org, который на момент написания этой работы содержит около 2,4 миллионов публикаций в области физики, математики, компьютерных наук, биологии, экономики, статистики и не только. arXiv предоставляет бесплатный и свободный доступ статьям и позволяет осуществлять поиск по их заголовкам. Однако, arXiv не позволяет искать по аннотациям или всему содержанию.

Целью данной работы является реализация полнотекстового и семантического поиска по arXiv. Достижение этой цели включает в себя решение следующих задач:

- сбор всех опубликованных статей вместе с метаданными;
- построение индекса поиска по названию, аннотациям и текстам работ;
- построение семантического индекса поиска;
- реализация поисковой системы;
- создание веб интерфейса пользователя.

Структура и объём работы. Бакалаврская работа состоит из введения, двенадцати разделов, заключения, списка использованных источников и двух приложений. Общий объем работы — 52 страницы, из них 46 страниц — основное содержание, список использованных источников информации — 21 наименование.

1 Устройство поисковых систем

Архитектура поисковой системы включает несколько основных компонентов, каждый из которых играет важную роль в обеспечении эффективного поиска.

1. Программа сборки информации: отвечает за сбор информации с веб-страниц. Она посещает страницы, извлекает содержимое и сохраняет его для дальнейшей обработки.
2. Индексация: процесс, в ходе которого собранная информация организуется и сохраняется в структуре, оптимизированной для быстрого поиска.
3. Поисковый механизм: обрабатывает запросы пользователей, сопоставляет их с индексом и возвращает релевантные результаты.
4. Ранжирование: определяет порядок отображения результатов поиска, основываясь на релевантности найденных документов.

Индексация является ключевым процессом в работе поисковых систем, позволяющим эффективно организовать и хранить информацию для быстрого поиска. Этот процесс начинается с веб-краулеров, которые посещают веб-страницы, извлекают их содержимое и передают его для дальнейшей обработки. Содержимое страниц подвергается токенизации, то есть разбиению текста на отдельные слова или фразы, называемые токенами. Затем происходит нормализация текста, включающая лемматизацию или стемминг, что позволяет привести слова к их базовой или корневой форме.

Обратный индекс — это основная структура данных, используемая в поисковых системах для быстрого и эффективного поиска информации. Он представляет собой словарь, в котором каждому терму (словом или фразе) соответствует список документов, содержащих этот терм. Обратный индекс позволяет значительно ускорить процесс поиска, так как вместо полного перебора всех документов система может быстро найти те, которые содержат искомые термы.

Ранжирование является одним из самых критически важных аспектов в работе поисковых систем, поскольку оно определяет порядок отображения результатов поиска. Основная цель ранжирования — предоставить пользователю наиболее релевантные и полезные результаты в ответ на его запрос. Для этого используются сложные алгоритмы и модели, которые анализируют множество факторов, чтобы оценить релевантность каждого документа.

2 Оптимизация поисковых систем

Исправление опечаток является важной функцией современных поисковых движков, поскольку позволяет пользователям получать релевантные результаты даже при наличии ошибок в запросах. Ошибки в запросах могут возникать по разным причинам, включая типографические ошибки, неправильное написание слов или использование синонимов и жаргона. Исправление опечаток повышает качество поиска, улучшает пользовательский опыт и помогает найти нужную информацию быстрее и точнее.

Использование словаря синонимов является важной техникой, применяемой в поисковых системах для улучшения качества и релевантности результатов поиска. Словарь синонимов, или тезаурус, представляет собой коллекцию слов и фраз, которые имеют схожие или взаимозаменяемые значения. Включение синонимов в процесс поиска позволяет системе учитывать разнообразие языка и увеличивает вероятность нахождения релевантных документов, даже если пользователь использует разные слова для описания одного и того же понятия.

Семантический поиск представляет собой продвинутую технологию, используемую в современных поисковых системах для улучшения понимания намерений пользователя и контекста его запросов. В отличие от традиционного поиска, который основывается на точном совпадении ключевых слов, семантический поиск стремится понять смысл и отношения между словами в запросе, а также их значение в контексте.

3 Общее описание архитектуры реализуемой поисковой системы

3.1 Сбор данных

На arXiv размещено около 2,4 миллиона работ, которые предстоит загрузить в полном объёме вместе со всеми метаданными. Для этого необходимо реализовать систему, способную сделать это за разумное время. Но для начала нужно понять, как конкретно можно загружать статьи с arXiv.

Единственным местом, дающим возможность посмотреть все статьи на arXiv является поиск, который позволяет получать любое подмножество работ. Однако, если искать по пустому запросу, указав явно все доступные темы, то в качестве результата можно получить все 2,4 миллиона работ.

2,4 миллиона статей накладывают множество ограничений на разрабатываемую поисковую систему. Приходится думать в значительной степени об оптимизации процесса с точки зрения

— дискового пространства;
— скорости загрузки публикаций.

3.1.1 Клиент-серверная архитектура

Предложенная ранее архитектура подразумевает очень простое горизонтальное масштабирование путём добавления большего числа виртуальных серверов, на которых будет запущен скрапер. В виду отсутствия большой нагрузки во время скрапинга на сервер, расположенный на домашнем компьютере, допускается одновременный запуск большого числа скраперов на большом числе виртуальных серверов, что будет линейно понижать время загрузки данных с arXiv.

Для дальнейшего удобства назовём сервер *archivist*, а клиентов — *scrapers*.

Кроме того, чтобы просто хранить базу данных, archivist должен выполнять следующие задачи:

— отслеживать состояния дней — собраны ли уже работы опубликованные в конкретный день;
— предоставлять клиентам возможность получения следующего задания;
— получать от клиентов результаты выполнения задания и сохранение полученных данных в базу.

В это время задачи scraper включают в себя именно в такой последовательности:

— запрос у archivist следующего задания;
— получения от `export.arxiv.org` текстов и метаданных о работах, опубликованных в этот день;
— извлечение текста из pdf файлов;
— отправка полученного текста назад archivist.

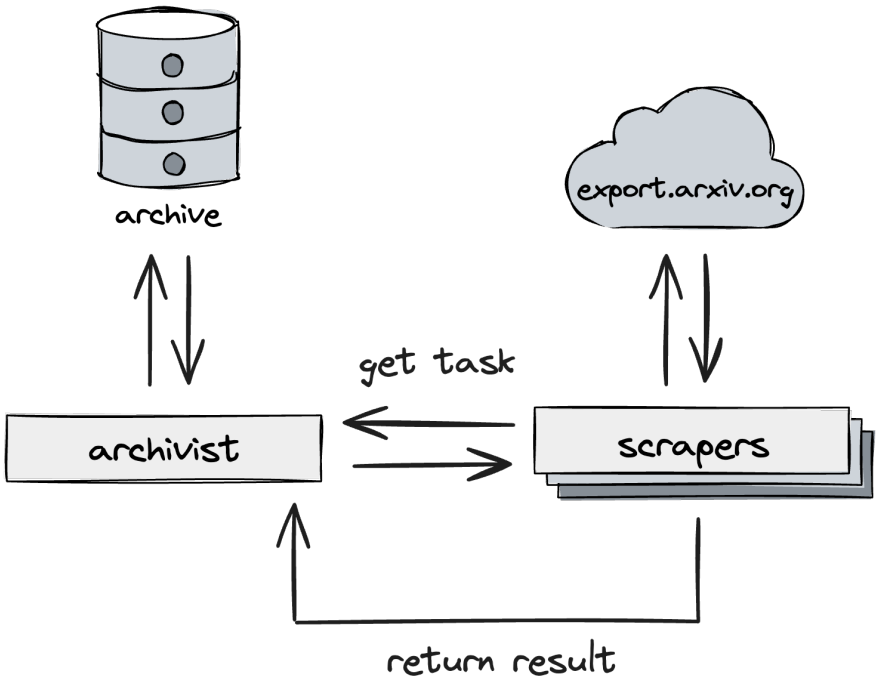

Рисунок 3.1 – Визуализация процесса сбора данных

3.2 Сервис вычисления эмбеддингов

Для реализации семантического поиска по статьям, необходимо выбрать подходящую модель машинного обучения. В качестве главного обоснования для выбора использовалась рейтинговая таблица сравнения моделей машинного обучения для получения эмбеддингов для задачи переранжирования, первые 4 модели из которой отображены в таблице.

Выбор пал именно на модель Mistral, как на самую лучшую модель, так ещё и модель с отрытыми весами, свободными для использования.

4 Реализация сбора данных

Для реализации сбора данных необходимо создать два сервиса: archivist и scraper.

4.1 Реализация сервера управления задачами сбора данных

Archivist — сервер в клиент-серверной архитектуры сбора данных. Он должен предоставлять REST API для:

- управления архивом:
  - получение статистики о количестве статей в базе данных и занимаемом пространстве;
  - получение всех публикаций за конкретный день;
- управления очередью заданий:
  - получение статистики о количестве доступных, занятых и выполненных задач;
  - получение следующей задачи;
  - сдачи результатов выполнения задачи;
  - добавление новых задач в очередь.

4.2 Реализация клиентов сбора данных

Реализация отдельных клиентов несколько проще сервера и не требует использование веб фреймворка, так как клиент общается с сервером исключительно по средствам его запросов к серверу.

Главная идея реализации клиента — бесконечно спрашивать новые задачи и решать их.

4.2.1 Извлечение текста из pdf

Для извлечения текста из текстов была выбрана библиотека `libpoppler` с pdf драйвером, которая обычно используется для создание просмотрщиков pdf. Именно эта библиотека показала себя наилучшим образов в задаче извлечения текста даже в сравнении со специализированными для это инструментами.

Единственная возникшая проблема — необходимость самостоятельно исправлять переносы строк. Для этого использовалось простое регулярное выражение.

5 Построение поискового индекса

В этой работе поисковая система строится на базе библиотеки Tantivy для Rust.

Для удобной работы с Tantivy используется структура `SearchEngine`, которая будет содержать схему с описанием всех индексируемых и хранимых полей для осуществления поиска.

```
pub struct SearchEngine {
    schema: tantivy::schema::Schema,
    searcher: Searcher,
    index: Index,
    query_parser: QueryParser,
}
```

В методе `new`, выполняющем роль конструктора, заполним схему нужными полями.

```
...
let mut schema_builder = Schema::builder();
let _id = schema_builder.add_u64_field("id", STORED);
let _url = schema_builder.add_text_field("url", STORED);
let _embediding = schema_builder.add_bytes_field("embedding", STORED);
let title = schema_builder.add_text_field("title",
↪ options.clone().set_stored());
let authors = schema_builder.add_text_field("authors",
↪ options.clone().set_stored());
let description = schema_builder.add_text_field("description",
↪ options.clone());
let body = schema_builder.add_text_field("body", options.clone());
let schema = schema_builder.build();
...
```

После чего выполняется построение индекса по данным из базы данных, получаемых через archivist. Индекс достаточно большой, чтобы помещаться в оперативную память, поэтому директория с индексом открывается через `mmap`, при этом сразу применяя Zstd сжатие.

Tantivy позволяет легко добавить функционал в токенизатор:

— приведение к нижнему регистру;

— использование словаря стоп слов, то есть слов, которые должны быть проигнорированы про поиске;

— добавление стеммера для поиска по однокоренным словам.

Для исправления опечаток в поисковых запросах можно воспользоваться библиотекой symspell, которая уже предоставляет словарь слов для английского языка.

Будем считать слово напечатанным с ошибкой, если в словаре нет этого слова, но есть слова с дистанцией равной 1. В случае большей дистанции, будем считать, что слово просто неизвестно словарю.

Для обработки синонимов использовался один из самых больших словарей такого рода — `WordnetSynonyms`. Синонимы также добавляются в запрос с меньшим приоритетом чем оригинальное слово, но в данном случае разница приоритета ещё более разительная.

6 Веб интерфейс

Для пользования поисковой системой, она должна предоставлять некий пользовательский интерфейс. Самым удобным вариантом такого интерфейса будет веб интерфейс.

В рамках этого проекта был реализован интерфейс с использованием библиотеки React.js вместе с мета-фреймворком Next.js.

Рисунок 6.1 – Главная страница поиска

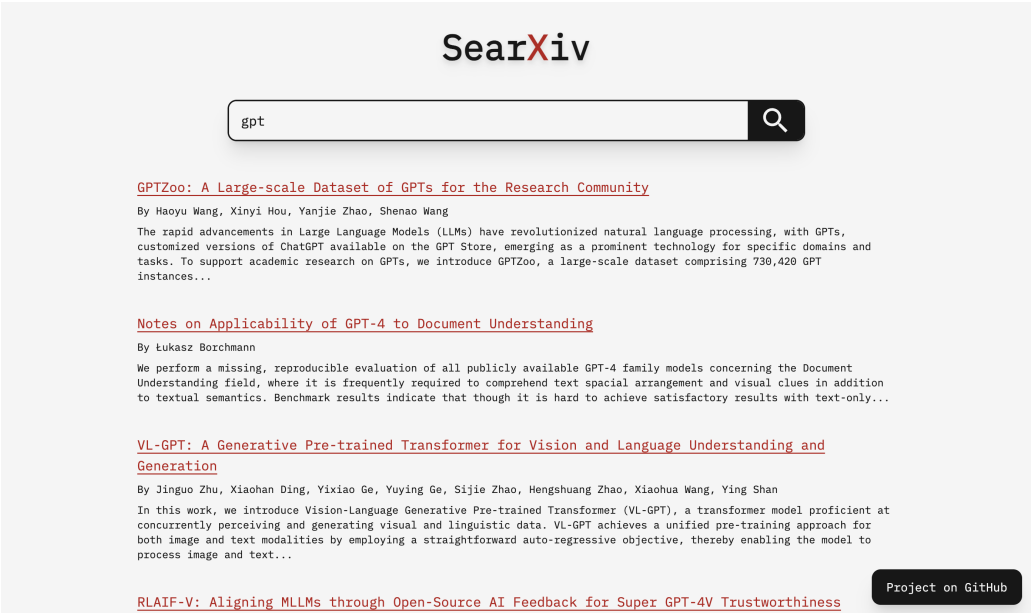

Рисунок 6.2 – Результаты поиска

7 Оценка результатов

Для оценки качества работы поисковой системе принято использовать $nDCG$ метрику, описываемую формулой:

$$nDCG_p = \frac{DCG}{IDCG}, \text{ где} \tag{7.1}$$

$$DCG_p = \sum_{i=1}^{p} \frac{rel_i}{log_2(i+1)}, \tag{7.2}$$

$$IDCG_p = \sum_{i=1}^{|REL|} \frac{2^{rel_i} - 1}{log_2(i+1)} \tag{7.3}$$

Для оценки качества делались запросы и оценивалась релевантность первых 10 результатов в поисковой выдаче. Значения релевантности ранжировались от 0 до 2. Соответственно,

- 0 — результат не является подходящим для запроса;
- 1 — результат является косвенно подходящим для запроса;
- 2 — результат является подходящим для запроса.

Полный список всех запросов и их результатов можно найти в приложении. По результатам тестирования, $nDCG$ в среднем составил ≈ 0.87, что можно считать достаточно хорошим показателем.

ЗАКЛЮЧЕНИЕ

В данной работе рассматривался пример построения системы полнотекстового поиска по arXiv, хранилищу препринтов научных публикаций и статей. В результате получилось успешно

- построить систему постоянного сбора новых публикаций с arXiv;
- реализовать вычисление эмбеддингов для семантического поиска;
- построить поисковой индекс по полученным данным;
- реализовать веб интерфейс для использования поиска.

Кроме того, получилось реализовать масштабируемую отказоустойчевую систему для всех 5 терабайт статей на arXiv, которая предусматривает автоматическое пополнение индекса новыми статьями на ежедневной основе.

Это решение — первый полнотекстовый поиск по arXiv с открытым исходным кодом, написанный на языке Rust с использованием Tantivy. Это показало, что Tantivy является подходящим выбором для построения крупных поисковых систем наряду с Lucene.