Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

НА ТЕМУ:

«Метод построения поисковых индексов в реляционной базе данных на основе глубоких нейронных сетей»

Студент: ИУ7-83Б _____ М. Д. Маслова
(группа) (подпись, дата) (И. О. Фамилия)

Руководитель: _____ А. А. Оленев
(подпись, дата) (И. О. Фамилия)

Нормоконтролер: _____ _____
(подпись, дата) (И. О. Фамилия)

2023 г.

СОДЕРЖАНИЕ

1 Конструкторская часть

1.1 Требования и ограничения метода

Метод построения поисковых индексов в реляционной базе данных на основе глубоких нейронных сетей (далее – метод построения индексов) должен:

1. получать из таблицы реляционной базы данных набор ключей и набор соответствующих указателей на записи в индексируемой таблице реляционной базы данных или иных значений, выполняющих роль указателей;
2. выполнять предварительную обработку полученных наборов, такую, как их совместную сортировку по значениям ключей, получение позиций ключей в отсортированном виде и нормализацию ключей и позиций;
3. обучать модель нейронной сети на подготовленных набора ключей и позиций;
4. сохранять параметры обученной модели для каждой таблицы с целью возможности выполнять запросы поиска без переобучения;
5. обеспечивать поиск записи (диапазона записей) таблицы по ключу (диапазону ключей) с использованием обученной модели;
6. обеспечивать корректность операции поиска после вставки/удаления новых записей путем переобучения модели;

На разрабатываемый метод накладываются следующие ограничения:

— в качестве ключей на вход принимаются целые числа для исключения решения дополнительной задачи преобразования входных данных;
— ключи во входном наборе уникальны.

1.2 Особенности метода построения индекса

1.2.1 Общее описание метода построения индекса

Основные этапы метода построения индекса приведены на функциональной декомпозиции метода на рисунке 1.1.

На вход методу подается набор уникальных целочисленных ключей, которые перед обучением модели глубокой нейронной сети проходят предварительную обработку по определенным правилам, описанным далее. Отдельным

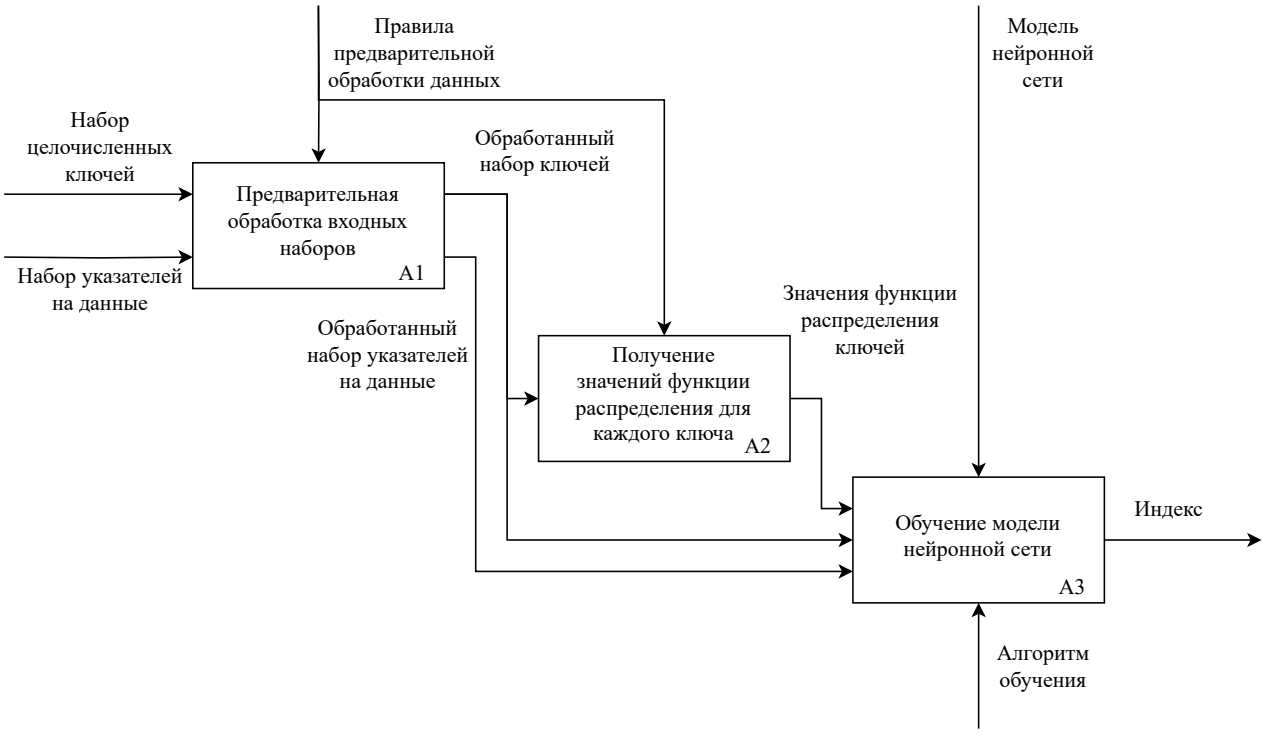

Рисунок 1.1 – Функциональная схема метода построения индекса

этапом выделено получение значений функций распределения для каждого ключа, относящееся к предварительной обработке, но представляющее собой ее ключевой момент. Полученные после первых двух этапов обработанные ключи и соответствующие значения функций используются для обучение модели глубокой нейронной сети в качестве признаков и меток соответственно.

Ключевым моментом метода является представление в отсортированном (по ключам) виде наборов ключей и набора соответвующих указателей на данные. Именно отсортированный вид позволяет использовать закономерность распределения ключей по позициям для обучения модели, предсказывать позиции ключей и уточнять их.

Результатом работы метода является структура данных, предствляющая собой индекс на основе глубокой нейронной сети и имеющая следующие поля:

— отсортированный массив ключей, поданных на вход;
— отсортированный по значениям ключей массив указателей на данные, соответствующие ключам;
— модель обученной глубокой нейронной сети, с помощью которой будет предсказываться положение ключа в отсортированном массиве;
— средняя и максимальная абсолютные ошибки предсказания позиции

ключа, для ее уточнения и возврата верного указателя на данные.

Краткое описания индекса, являющегося результатом работы метода, как структуры данных представлено на рисунке 1.2.

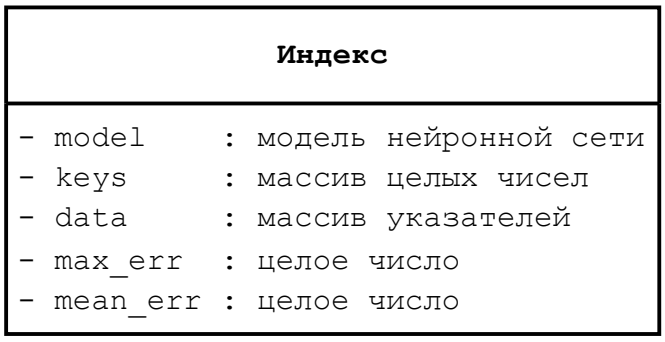

Рисунок 1.2 – Индекс как структура данных

Подробное описание каждого этапа приведено в следующих пунктах данного подраздела.

1.2.2 Предварительная обработка данных

Разрабатываемый метод построения индекса предполагает предварительную обработку набора целочисленных ключей, схема алгоритма которой представлена на рисунке 1.3.

На вход подаются согласованные массивы ключей и указателей на данные, то есть считается, что ключ, стоящем на первой позиции в массиве ключей, идентифицирует данные по указателю, стоящем на первой позиции в массиве указателей; ключ, стоящей на второй, — указатель, стоящий на второй, и так далее. С учетом этого происходит согласованная сортировка двух массивов по значениям ключей.

Далее для последующего обучения модели глубокой нейронной сети производится нормализация ключей, выступающих в качестве входных данных сети, в дапазон $[0, 1]$, для чего используется метод минимакс-нормализации, при котором нормализованное значение вычисляется по формуле:

$$x_{\text{норм}} = \frac{x - x_{\min}}{x_{\max} - x_{\min}}, \tag{1.1}$$

где $x_{\text{норм}}$ — нормализованное значение ключа;

x — натуральное значение ключа;

$x_{\min}$, $x_{\max}$ — минимальное и максимальное возможное значение ключа в

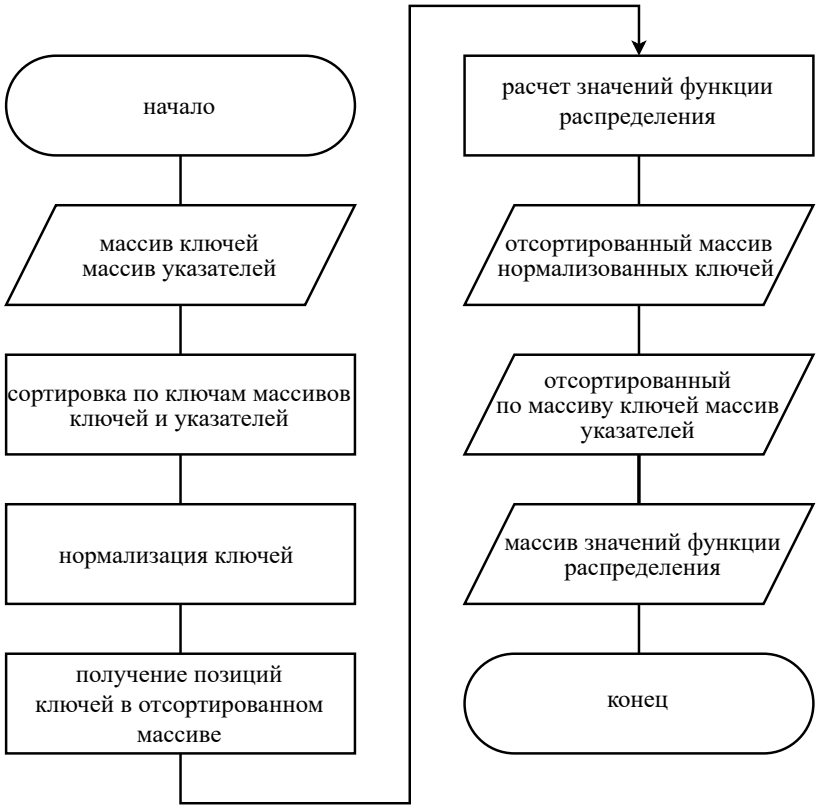

Рисунок 1.3 – Схема алгоритма предварительной обработки данных

наборе соответственно.

Далее полученный набор ключей размечается путем вычисления для каждого ключа K значения функции распределения F по его позиции P в отсортированном массиве и количества индексируемых ключей N с помощью формулы:

$$F(K) = \frac{P}{N} \tag{1.2}$$

На этом предварительная обработка завершается и полученные отсортированные массивы ключей и указателей, а также соответствующие значения функции распределения передаются в качестве входных данных на этап обучения модели глубокой нейронной сети.

Полное описание алгоритма предварительной обработки представлено на листинге 1.1.

Листинг 1.1 – Предварительная обработка данных

Вход:
$keys$: массив целочисленных ключей;
$data$: массив указателей на данные, соответствующие ключам;
N : длина массивов.
Выход:
$keys$: отсортированный массив нормализованных ключей;
$data$: отсортированный по массиву ключей массив указателей;
cdf : массив значений функции распределения.

```
1 begin
2     сортировать keys и data по keys;
      ▷ здесь и далее под операцией к вектору (массиву) и числу понимается
      ▷ применение данной операции с данным числом к каждому элементу вектора
3     keys ← (keys − keys[0]) / (keys[N−1] − keys[0]);
4     positions ← [0, 1, ..., N − 1];
5     cdf ← positions / (N − 1);
6     return keys, data, cdf;
7 end
```

1.2.3 Разработка архитектуры глубокой нейронной сети

Полученные на этапе предварительной обработки массивы ключей и соответствующих им значений функций распределения используются для обучения глубокой нейронной сети. Массив данных, хранимый в индексе, используется для возврата нужных данных при поиске, но не для обучения.

Задача построения индекса на основе нейронной сети сводится к задаче аппроксимации функции распределения, для решения которой подходят полносвязные нейронные сети.

За основу архитектуры глубокой нейронной сети принята архитектура из исследования [**1D2D**], которая представлена на рисунке 1.4. Это полносвязная нейронная сеть с двумя скрытыми слоями по 32 нейрона. В качестве фунцкции активации в каждом нейроне скрытых слоев используется линейный выпрямитель или ReLU (*Rectified Linear Unit*), значение которой вычисляется по формуле 1.3, чему соответствует график, представленный на рисунке 1.5.

$$f(x) = \max(0, x). \tag{1.3}$$

Активационная функция выходного слоя является линейной.

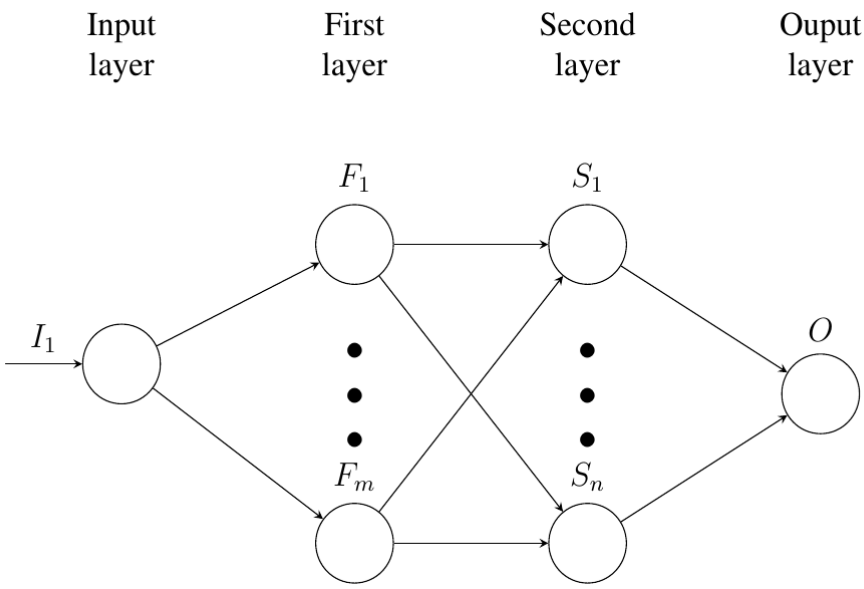

Рисунок 1.4 – Полносвязная нейронная сеть с двумя скрытыми слоями

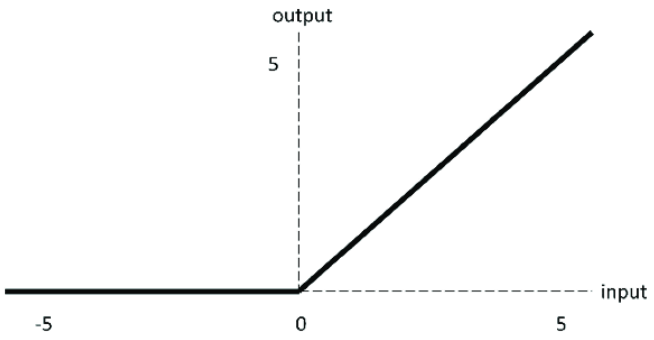

Рисунок 1.5 – График функции ReLU

Для исследования возможности увеличения точности предсказания, и как следствие уменьшение времени поиска, в качестве модели глубокой нейронной сети, представляющей основу индекса, используется полносвязная нейронная сеть с тремя слоями, представленная на рисунке 1.6. Число нейронов в слоях и активационные функции приняты такими же, как в случае глубокой нейронной сети с двумя скрытыми слоями.

Обучение обоих моделей глубокой нейронной сети начинается с инициализации весов случайно сгенерированными значениями по распределению $U(-\frac{1}{\sqrt{N}}, \frac{1}{\sqrt{N}})$. Собственно обучение проводится методом стохастического градиентного спуска с оптимизацией в качестве функции потерь среднеквадратической ошибки (*MSE — mean squared error*). Описание алгоритма обучения приведено на листинге 1.2.

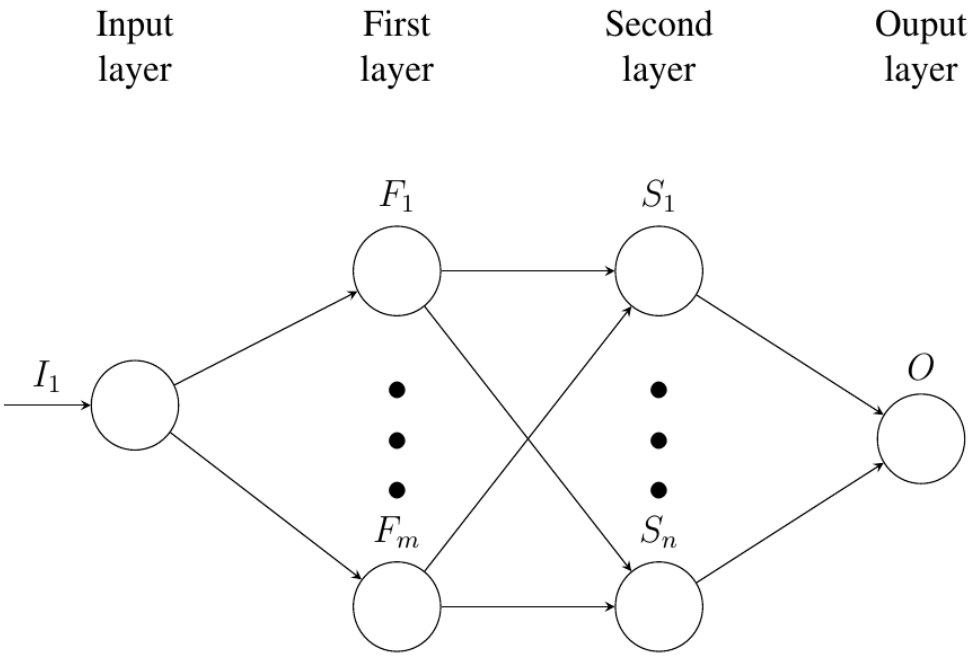

Рисунок 1.6 – Полносвязная нейронная сеть с тремя скрытыми слоями

Листинг 1.2 – Алгоритм обучения глубокой нейронной сети на основе градиентного спуска

Вход:

$keys$: массив нормализованных ключей;
cdf : массив значений функции распределения для каждого ключа;
N : длина массивов;
$epochs$: количество эпох;
$alpha$: скорость обучения.

Выход:

$model$: обученная модель.

begin
  $model \leftarrow$ структура модели;
  ▷ вектор смещений включей в матрицу весов
  $model.weights \leftarrow U(-\frac{1}{\sqrt{N}}, \frac{1}{\sqrt{N}})$; ▷ случайные значения из распределения
  for $epoch \leftarrow 1$ **to** $epochs$ **do**
    согласованно перемешать $keys$ и cdf;
    for $i \leftarrow 0$ **to** $N - 1$ **do**
      $\hat{y} = model.predict(key[0])$; ▷ прямой проход
      $mse = (\hat{y} - cdf[i])^2$;
      $gradients \leftarrow backward\_pass(model, mse)$; ▷ обратный проход
      $model.weights \leftarrow model.weights - alpha \cdot gradients$
    end
  end
end

Так как в случае поискового индекса глубокая нейронная сеть будет предсказывать положения только тех ключей, на которых она обучалась, явления переобучения нейронной сети является положительным, поэтому в качестве одного батча выступает одна пара (ключ; значение функции распределения), что также отражено на листинге выше.

1.3 Разработка алгоритмов поиска и вставки

Основной операцией, выполняемой с помощью индекса, является поиск, функциональная схема выполнения которого представлена на рисунках 1.7-1.8.

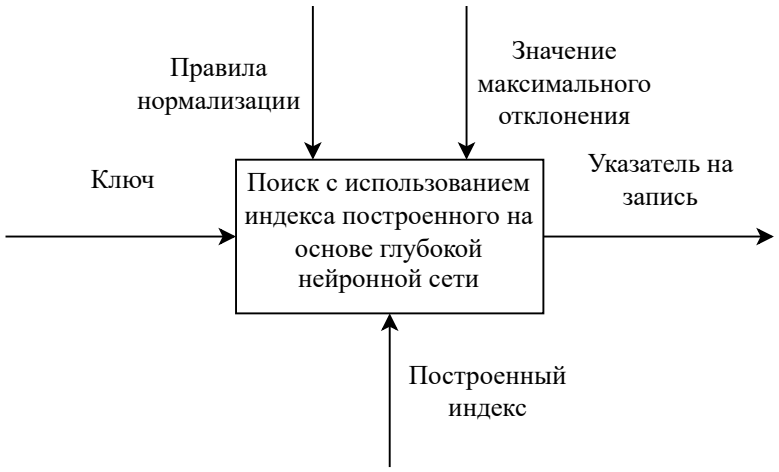

Рисунок 1.7 – Функциональная схема нулевого уровня поиска

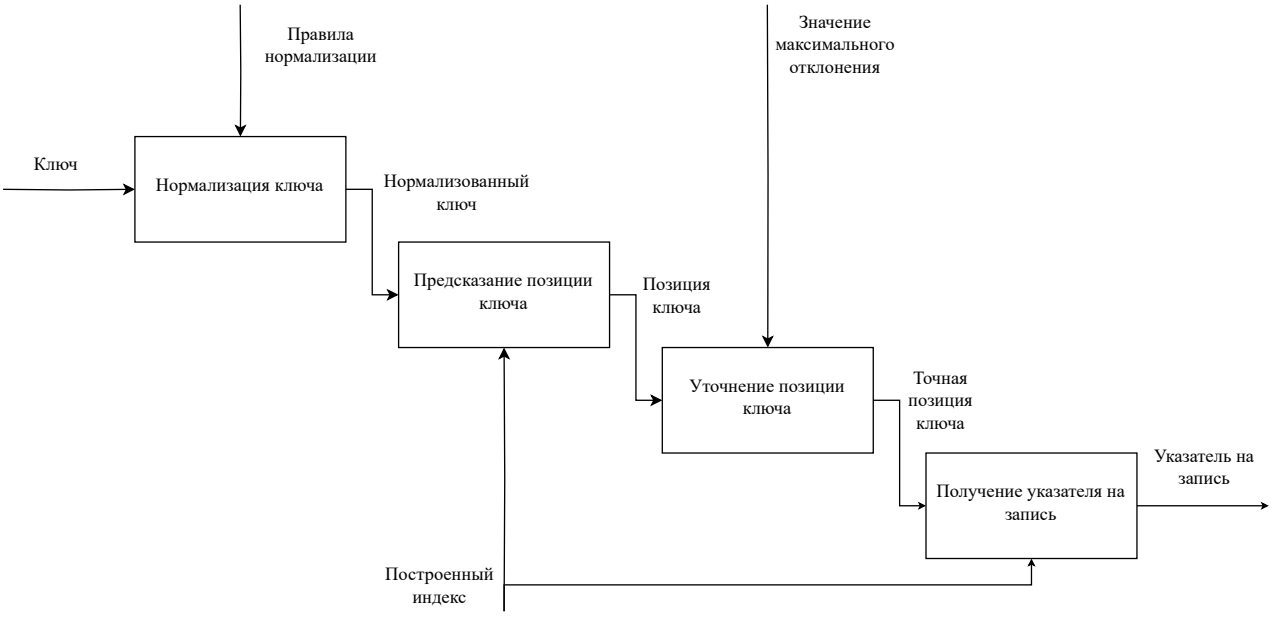

Рисунок 1.8 – Функциональная схема первого уровня поиска

Получаемая позиция требует уточнения, происходящего за счет получаемого в результате обучения модели максимального отклонения от истинного расположения. (Алгоритм уточнения???)

Для реализации вставки происходит добавление новых значений ключа и указателя в существующие массивы, и повторятся алгоритм построения индекса (??? не сначала, а со значений параметров уже обученной модели ???).

1.4 Данные для обучения и тестирования индекса

Так как в основе индекса на основе глубоких нейронных сетей лежит аппроксимация функции распределения ключей, работу разработанного метода необходимо протестировать на различных законах, перечисленных далее.

— Равномерный закон $R[a, b]$, фукнция распределения которого описывается формулой 1.4.

$$F(x) = \begin{cases} 0, & \text{если } x < a \\ \frac{x-a}{b-a}, & \text{если } a \leq x \leq b \\ 1, & \text{если } x > b. \end{cases} \tag{1.4}$$

Нормализованные ключей лежат в диапазоне $[0, 1]$, значения функции распределения за пределами этого диапазона не представляют интереса для построения индекса, поэтому можно считать, что функция имеет вид, представленный формулой 1.5.

$$F(x) = x, \ x \in [0, 1]. \tag{1.5}$$

— Нормальный закон $N(\mu, \sigma^2)$, функция распределения которого описывается формулой 1.6.

$$F(x) = \int_{-\infty}^{x} \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(t-\mu)^2}{2\sigma^2}\right) dt \tag{1.6}$$

Для тестирования метода построения по заданным законам генерируются значения ключей, по совокупности которых формируется эмпирическая функция распределения, как это было описано выше.

Также для проверки работы метода требуется оценить его работоспособность на реальных данных, в качестве которых выбраны уникальные идентифи-

каторы элементов из открытого набора географических данных *OpenStreetMap*, или *OSM* [**osm**], функция распределения которых имеет более сложный вид, чем основные законы распределения.

Графики функций распределения каждого из набора входных ключей, представлены на рисунке **??**.

2 Технологическая часть

2.1 Выбор средств программной реализации

Для реализации метода построения индекса в реляционной базе данных на основе глубоких нейроных сетей в качестве языка программирования выбран Python 3.10 [**python**], так как предоставляет широкий выбор библиотек для глубокого обучения и визуализации его результатов. В качестве библиотеки глубокого обучения выбран TensorFlow 2.11.0 [**tf**] и работающий поверх нее высокоуровневый программный интерфейс Keras 2.11.0 [**keras**]. Для работы с массивами данных выбрана библиотека numpy [**numpy**].

В качестве реляционной системы управления базами данных выбрана SQLite [**sqlite**], предоставляющая программный интерфейс виртуальных таблиц, позволяющих релизовать пользовательский поисковый индекс. Виртуальные таблицы являются одним из видов расширений SQLite, программный интерфейс которых предоставляется на языке C [**c**], который и выбран в качестве языка программирования для взаимодействия с реляционной базой данных.

Для обеспечения взаимодействия между компонентом работы с базой данных и компонентом, непосредственно реализующий индекс, используются библиотеки языка C `Python.h` для работы с объектами языка Python и `numpy/arrayobject.h`, предоставляющая программный интерфейс для работы с numpy-массивами, которые являются основным типом данных, через который происходит взаимодействие модулей.

2.2 Реализация программного обеспечения

2.2.1 Форматы входных и выходных данных

Основой для построения индекса в качестве входных выступают данные из таблицы реляционной базы данных SQLite. Требованием к таблице является наличие атрибута целочисленного типа (`INTEGER`) с уникальными значениями (`UNIQUE`).

Для создания индекса в аргументах соответствующего запроса должен присутствовать идентификатор столбца, удовлетворяющего требованию описанному выше, и строковое имя модели индекса, на основе которой он строится (`FCNN2` — для модели с двумя скрытыми слоями, `FCNN3` — с тремя).

В качестве входных данных компонента, реализующего индекса, является набор значений столбца, идентификатор которого был указан в аргументах запроса на создание индекса, и идентификаторы строк `ROWID` индексируемой таблицы.

Выходным данных является указатель на объект, представляющий индекс на основе глубокой нейронной сети, обученный на предоставленных входных данных.

Формат входных и выходных данных операций с индексом (поиска и вставки) описан в следующем пункте.

2.2.2 Поддерживаемые виды запросов

Разработанное программное обеспечения предоставляет возможность работы только с запросами фильтрации, представленными оператором `WHERE` следующими со следующими условиями:

— `column operator value`,
  где `column` — имя проиндексированного столбца,
  `operator` — одна из операций сравнения: `=, <, >, <=, >=`,
  `value` — некоторое целочисленное значение.
— `column BETWEEN value1 AND value2`,
  где `column` — имя проиндексированного столбца,
  `value1, value2` — целочисленные значения, представляющие нижнюю и верхнюю границы диапазона.

Выходным значением из модуля на языке Python, реализующего индекс, по данным запросам является numpy-массив с соответствующими запросу значениями ROWID, а результатом работы программного обеспечения — набор записей таблицы с ROWID из представленного массива.

Для вставки поддерживается стандартный запрос `INSERT`, результатом которого является индекс с переобученной на новых данных моделью. Запросы удаления и изменения не поддерживаются, так как являются вторичными для оценки работы метода.

2.2.3 Программный интерфейс виртуальных таблиц

Виртуальные таблицы SQLite [**vtable**] — это объект базы данных, представляющий с точки зрения инструкций SQL обычную таблицу или представле-

ние, но обрабатывающий запросы посредством вызова функций программного интерфейса, которые реализуются пользователем.

Виртальные таблицы являются расширением SQLite, регистрация которых происходит с использованием макросов и фукнции инициализации, представленных на линстинге 2.1.

Листинг 2.1 – Инициализация расширения

```
#include <sqlite3ext.h>

SQLITE_EXTENSION_INIT1

int sqlite3_extension_init(sqlite3 *db,
                           char **pzErrMsg,
                           const sqlite3_api_routines *pApi)
{
  int rc = SQLITE_OK;
  SQLITE_EXTENSION_INIT2(pApi);

  /* инициализация расширения */

  return rc;
}
```

При инициализация расширения вирутальной таблицы должен быть зарегистрирован модуль, описывающийся структурой `sqlite3_module`, с помощью функции регистрации `sqlite3_create_module`, подробное описание которых представлено на листинге 2.2

Методы, сигнатуры которых представлены на листинге 2.2, можно разделить на две группы.

— Методы для взаимодействие с таблицей, как с некоторым объектом, к которым относятся:

  – `xCreate` — создание виртуальной таблицы, при выполнении соответствующего запроса, представленного на листинге 2.3;
  – `xConnect` — подключение к виртуальной таблице, вызывющийся при выполнении любого запроса к таблице, который является первым при повторном подключении к базе данных;
  – `xDestroy` — удаление вирутальной таблицы при выполнеивыполнении запроса, представленного на листинге 2.4;
  – `xDisconnect` — удаление поключения к виртуальной таблице.

Листинг 2.2 – Структура и функции для регистрации модуля виртуальной таблицы

```
struct sqlite3_module {
    int iVersion;
    int (*xCreate)(sqlite3*, void *pAux,
                int argc, char *const*argv,
                sqlite3_vtab **ppVTab,
                char **pzErr);
    int (*xConnect)(sqlite3*, void *pAux,
                int argc, char *const*argv,
                sqlite3_vtab **ppVTab,
                char **pzErr);
    int (*xBestIndex)(sqlite3_vtab *pVTab, sqlite3_index_info*);
    int (*xDisconnect)(sqlite3_vtab *pVTab);
    int (*xDestroy)(sqlite3_vtab *pVTab);
    int (*xOpen)(sqlite3_vtab *pVTab,
                sqlite3_vtab_cursor **ppCursor);
    int (*xClose)(sqlite3_vtab_cursor*);
    int (*xFilter)(sqlite3_vtab_cursor*,
                  int idxNum, const char *idxStr,
                  int argc, sqlite3_value **argv);
    int (*xNext)(sqlite3_vtab_cursor*);
    int (*xEof)(sqlite3_vtab_cursor*);
    int (*xColumn)(sqlite3_vtab_cursor*, sqlite3_context*, int);
    int (*xRowid)(sqlite3_vtab_cursor*, sqlite_int64 *pRowid);
    int (*xUpdate)(sqlite3_vtab *, int,
                  sqlite3_value **, sqlite_int64 *);
    /* представлены указатели на реализующиеся методы */
    /* при инициализации структуры, */
    /* остальные поля принимают значение 0 */
};

int sqlite3_create_module(
    sqlite3 *db,          /* соединение для регистрации модуля */
    const char *zName,      /* имя модуля */
    const sqlite3_module *, /* ссылка на структуру модуля */
    void *,                 /* данные для xCreate/xConnect*/
);
```

Листинг 2.3 – Запрос на создание виртуальной таблицы

```
CREATE VIRTUAL TABLE <имя_таблицы> USING <имя_модуля>(arg1, ...);
```

Листинг 2.4 – Запрос на удаление виртуальной таблицы

```
DROP TABLE <имя_таблицы>;
```

Данные методы работают со структурой `sqlite3_vtab`, представленной на листинге 2.5. Для реализации нужных функциональностей указатель на данную структуру включается в пользовательскую, с которой уже работают представленные методы посредством преобразования типов. Это дает возможность передавать между методами вирутальной таблицы нужные данные.

Листинг 2.5 – Структура вирутальной таблицы

```
1 struct sqlite3_vtab {
2     const sqlite3_module *pModule; /* модуль таблицы */
3     int nRef; /* число ссылок, инициализирующееся ядром SQLite */
4     char *zErrMsg; /* для передачи сообщений об ошибках ядру */
5 };
```

— Методы прохода по записям таблицы, использующие для этого структуру курсора `sqlite3_vtab_cursor`, представленую на листинге 2.6, над которой также реализуют обертку для хранения необходимых для обработки переменных.

Листинг 2.6 – Структура курсора

```
1 struct sqlite3_vtab_cursor {
2   sqlite3_vtab *pVtab; /* указатель на виртуальную таблицу */
3 };
```

Данная группа представлена методом обработки вставки, удаления и изменения записи (последний) и методами для прохода по записям таблицы при поиске:

- `xOpen` — создание и инициализации структуры курсора;
- `xBestIndex` — получение параметров фильтрации и выбор лучшего индекса для обработки запроса;
- `xFilter` — получение соответствующих параметрам фильтрации записей, установка курсора на первую из них;
- `xEof` — проверка окончания списка выбранных записей;
- `xNext` — переход к следующей записи;
- `xColumn` — обработка столбца записи;
- `xClose` — удаление структуры курсора;
- `xUpdate` — реализация запросов вставки, удаления и изменения.

Для реализации метода построения индекса используются оберточные структуры для виртуальной таблицы и курсора, представленные на листинге 2.7.

Листинг 2.7 – Пользовательские структуры виртуальной таблицы и курсора

```
1  typedef struct lindex_vtab {
2      sqlite3_vtab base;  /* основа виртуальной таблицы */
3      sqlite3_stmt *stmt; /* инструкция доступа к записи по ROWID*/
4      PyObject *lindex;   /* собственно объект индекса */
5  } lindex_vtab;
6
7  typedef struct lindex_cursor {
8      sqlite3_vtab_cursor base; /* базовая структура курсора */
9      PyObject *rowids;         /* массив выбранных ROWID */
10     PyArrayIterObject *iter;  /* итератор по массиву ROWID */
11 } lindex_cursor;
```

На листинге 2.8 представлена реализация метода создания индекса, реализованного в качестве `xCreate`. Код инициализации и запуска обучение индекса в Python через программиный интерфейс приведен на листинге 2.9.

Обработка параметров фильтрации приведена на листинге 2.10. Получение массива подходящих строк и проход для их вывода приведены на листингах 2.11, 2.12 соответственно.

2.2.4 Реализация индекса

2.3 Сборка программного обеспечения

2.4 Взаимодействие с программным обеспечением

Взаимодействие с программным обеспечением происходит через командную строку sqlite3. Пример работы представлен на листинге **??**.

2.5 Результаты тестирования

???

Листинг 2.8 – Создание индекса

```
int lindexCreate(sqlite3 *db,
                 void *pAux,
                 const int argc,
                 const char *const *argv,
                 sqlite3_vtab **ppVtab,
                 char **errMsg)
{
    lindex_vtab *vtab = sqlite3_malloc(sizeof(lindex_vtab));

    if (!vtab)
        return SQLITE_NOMEM;

    memset(vtab, 0, sizeof(*vtab));
    *ppVtab = &vtab->base;

    char *sql_template = get_create_table_query_by_args(argc,
       argv);

    const char *vTableName = argv[2];
    const char *rTableName = sqlite3_mprintf("r%s", vTableName);

    char *vSqlQuery = sqlite3_mprintf(sql_template, vTableName);
    char *rSqlQuery = sqlite3_mprintf(sql_template, rTableName);

    int rc = sqlite3_declare_vtab(db, vSqlQuery);

    if (!rc)
        rc = sqlite3_exec(db, rSqlQuery, NULL, NULL, errMsg);

    sqlite3_free(sql_template);
    sqlite3_free(vSqlQuery);
    sqlite3_free(rSqlQuery);

    rc = initPythonIndex(db, rTableName, "fcnn2", vtab);

    char* result_query = sqlite3_mprintf("SELECT * FROM %s WHERE
       ROWID = ?;", rTableName);
    sqlite3_prepare_v2(db, result_query, -1, &vtab->stmt, NULL);

    return rc;
}
```

Листинг 2.9 – Инициализация индекса

```
int initPythonIndex(sqlite3 *db,
                    const char *const tableName,
                    const char *const modelName,
                    lindex_vtab *vTab) {
    char* query = sqlite3_mprintf("SELECT ROWID, * FROM %s",
       tableName);

    sqlite3_stmt* stmt;
    sqlite3_prepare_v2(db, query, -1, &stmt, NULL);
    sqlite3_free(query);

    PyObject* builderModule = PyImport_ImportModule("indexes.
       builder");
    PyObject* builderClassName = PyObject_GetAttrString(
       builderModule, "LindexBuilder");
    PyObject* pyModelName = PyTuple_Pack(1, PyUnicode_FromString(
       modelName));
    PyObject* builder = PyObject_CallObject(builderClassName,
       pyModelName);
    PyObject* lindex = PyObject_CallMethod(builder, "build", NULL
       );
    PyObject* keys = PyList_New(0);
    PyObject* rows = PyList_New(0);

    int i = 0;
    while (sqlite3_step(stmt) == SQLITE_ROW) {
        int key = sqlite3_column_int(stmt, 1);
        int64_t rowid = sqlite3_column_int64(stmt, 0);

        PyList_Append(keys, PyLong_FromLong(key));
        PyList_Append(rows, PyLong_FromLong(rowid));

        i++;
    }

    if (i) {
        PyObject* train = PyUnicode_FromString("train");
        PyObject* check = PyObject_CallMethodObjArgs(lindex,
           train, keys, rows, NULL);
        Py_DECREF(check);
        Py_DECREF(train);
    }

    vTab->lindex = lindex;

    Py_DECREF(keys);
    /* ... */
    sqlite3_finalize(stmt);

    return SQLITE_OK;
}
```

Листинг 2.10 – Обработка параметров фильтрации запроса

```
int lindexBestIndex(sqlite3_vtab *tab,
                    sqlite3_index_info *pIndexInfo)
{
    if (pIndexInfo->nConstraint > 0) {
        for (int i = 0; i < pIndexInfo->nConstraint; i++) {
            if (pIndexInfo->aConstraint[i].usable
                && pIndexInfo->aConstraint[i].op ==
                   SQLITE_INDEX_CONSTRAINT_EQ) {
                pIndexInfo->aConstraintUsage[i].argvIndex = i+1;
                pIndexInfo->aConstraintUsage[i].omit = 1;
            }
        }
    }
    return SQLITE_OK;
}
```

Листинг 2.11 – Выбор строк, удовлетворяющих фильтру

```
int lindexFilter(sqlite3_vtab_cursor *cur,
                 int idxNum,
                 const char *idxStr,
                 int argc,
                 sqlite3_value **argv)
{
    import_array()
    lindex_vtab *lTab = (lindex_vtab*)cur->pVtab;

    PyObject* keys = PyList_New(0);
    PyList_Append(keys, PyLong_FromLong(sqlite3_value_int(argv[i
       ])));

    PyObject* find = PyUnicode_FromString("find");
    PyObject* rowids = PyObject_CallMethodObjArgs(lTab->lindex,
       find, keys, NULL);
    npy_intp size = PyArray_SIZE(rowids);
    PyArrayIterObject *iter = (PyArrayIterObject *)
       PyArray_IterNew(rowids);

    lindex_cursor *pCur = (lindex_cursor*)cur;
    pCur->rowids = rowids;
    pCur->iter = iter;

    int64_t rowid = *(int64_t *)PyArray_ITER_DATA(pCur->iter);
    sqlite3_bind_int64(lTab->stmt, 1, rowid);
    sqlite3_step(lTab->stmt);

    return SQLITE_OK;
}
```

Листинг 2.12 – Реализация работы курсора

```
int lindexNext(sqlite3_vtab_cursor *cur)
{
    lindex_cursor *pCur = (lindex_cursor*)cur;
    lindex_vtab *lTab = (lindex_vtab*)cur->pVtab;

    sqlite3_reset(lTab->stmt);
    sqlite3_clear_bindings(lTab->stmt);
    int64_t rowid = *(int64_t *)PyArray_ITER_DATA(pCur->iter);
    sqlite3_bind_int64(lTab->stmt, 1, rowid);
    sqlite3_step(lTab->stmt);

    PyArray_ITER_NEXT(pCur->iter);

    return SQLITE_OK;
}

int lindexColumn(sqlite3_vtab_cursor *cur,
                 sqlite3_context *ctx,
                 int i)
{
    lindex_vtab *lTab = (lindex_vtab*)cur->pVtab;
    int columnValue = sqlite3_column_int(lTab->stmt, i);
    sqlite3_result_int(ctx, columnValue);

    return SQLITE_OK;
}

int lindexEof(sqlite3_vtab_cursor *cur)
{
    lindex_cursor *pCur = (lindex_cursor*)cur;
    return !PyArray_ITER_NOTDONE(pCur->iter);
}
```