Продукт

# Задание

−Создать алгоритм на базе ИИ, который получает на вход два списка адресов с уникальными ID. Адреса в списках идентичны по смыслу, но различаются по написанию (например, "Невский проспект" и "Невский пр"). Алгоритм должен создать таблицу соответствия ID между этими списками.

−Разработать веб-приложение (фронтенд + бэкенд), реализующее нечёткий поиск по адресам. Система должна находить и предлагать варианты адресов при неполном вводе (например, "невс про 15") или при наличии опечаток.

Ожидаемый результат: Рабочий прототип с открытым API для интеграции, демонстрирующий оба компонента задачи.

# 

# Тех идея

Продукт представляет собой группу контейнеров, один из которых является поисковым движком с нечетким поиском на основе ИИ и с архитектурой подобной Elastic search. Предназначен для встраивания в другие проекты.

* Как и любой поисковый движок - этот продукт нуждается компактном хранилище информации доступной для поиска
* Соответственно продукт имеет функционал добавления, обновления, удаления данных. В первую очередь из внешнего источника: пока только csv. (теоретически может работать с БД пользователя, но пока что это не будет реализовано)

Расшифровка нечеткого запроса должна включать в себя такой элемент:

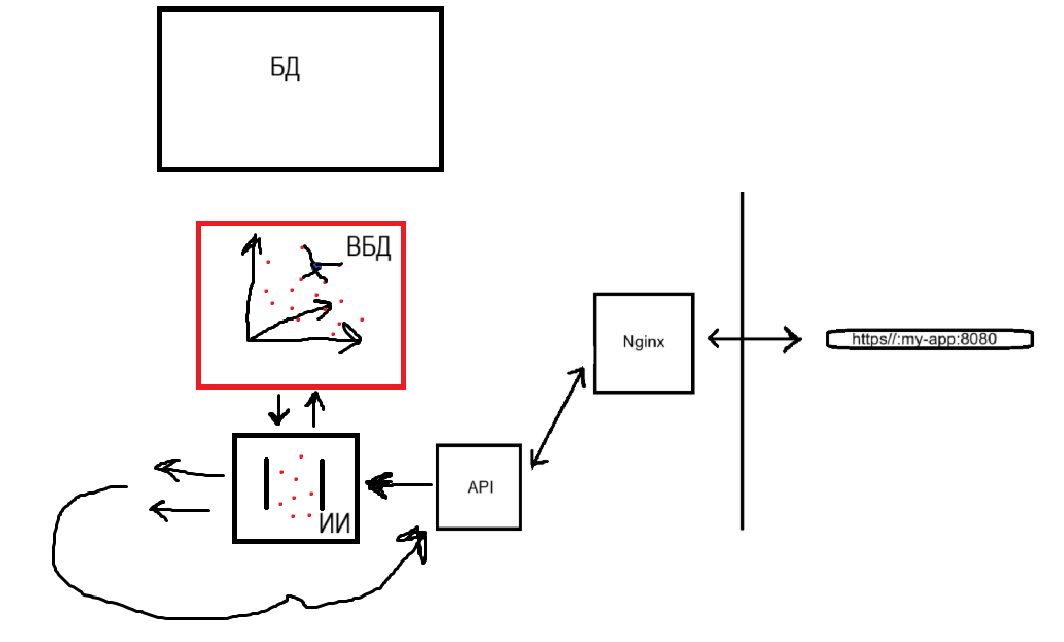
* получает на вход два списка адресов с уникальными ID. Адреса в списках идентичны по смыслу, но различаются по написанию (например, "Невский проспект" и "Невский пр"). Алгоритм должен создать таблицу соответствия ID между этими списками.

# Вопросы

1. Какая будет база данных. Теоретически можно попробовать реализовать этот сервис чтобы он был универсальный для баз данных
2. У меня есть идея, что можно будет перевести текущую базу данных в новую базу данных - векторную базу данных - тк в этой базе данных нейросеть может быстро и качественно осуществлять поиск
3. Как вы считаете допустимо ли это

# Архитектурные идеи

0)



1) ну самая логичная идея это создать векторную базу данных

проиндексировать все данные найти самые ближайшие записи

прогнать их через ИИ чтобы найти самую похожую.

единственное - это то, что у меня данные очень хорошо структурированы и имеют структуру реляционной базы данных (город, улица, дом). Как использовать эту структуру данных ?

2)

- можно извлечь структуру данных

- Построить реляционную БД.

- извлечь структуру запроса с помощью ИИ.

- для каждого поля запроса найти наиболее соответствующее поле из базы данных даже без ИИ.

3)

* Построить реляционную БД но вместо текстовых значений хранить сразу эмбеддинги от BERT
* Извлечь структуру данных.
* Получить запрос от пользователя с ошибками.
* Извлечь структуру запроса с помощью ИИ
* Получить эмбеддинг каждого поля запроса
* В порядке иерархии, сравнивать поля запроса и данных с помощью string matcher- а и брать самые похожие
* Так до конечного поля
* Сконкатенировтаь весь путь бд и обратно преобразовать в слова
* Вернуть ответ пользователю

Идея/архитектура проекта заключается в том, чтобы использовать инструменты NLP - обработки естественного языка - для выделения нужной информации из запроса пользователя с ошибками и без четкой структуры

Обработка языка выполняется в два этапа:

1) Парсинг строки на словосочетания относящиеся к классам: улица, дом итд

(Задача: named entity recognition)

2) Для выявленных словосочетаний найти наиболее схожие варианты из базы данных

(Задача: entity matching/ string matching)

ресёрч

# 

# Google

## fuzzy address matching deep learning

Статья 1 9/10

<https://paperswithcode.com/paper/deezymatch-a-flexible-deep-learning-approach>

# Google scholar

## fuzzy matching deep learning

Статья 1 8/10

<https://scholar.google.com/scholar?hl=en&as_sdt=0%2C5&q=fuzzy+matching+deep+learning&btnG=#d=gs_qabs&t=1744876764797&u=%23p%3D5kpS4e6def8J>

## string fuzzy matching ai

# Youtube

[[2006.00575] Neural Entity Linking: A Survey of Models Based on Deep Learning](https://arxiv.org/abs/2006.00575)

Entity Linking

Entity matching

Ditto

DeepMatcher

<https://m.youtube.com/watch?v=423oWxmqVk4&pp=ygUVRGl0dG8gZW50aXR5IG1hdGNoaW5n>

NLP

1) Традиционные алгоритмы entity matching

2) Мл и глубокое обучение (DeepMatch)

3) Лингвистические модели (ditto, bert, RoBERT)

4) сложные архитектуры с глубоким обучением (DeezyMatch)

# Google scholar 2 попытка

## ("fuzzy matching" OR "entity matching" OR "entity linking") AND ("rule-based" OR "heuristic") AND ("deep learning" OR "transformer models" OR "language models") AND ("survey" OR "review" OR "state of the art")

8/10

Neural networks for entity matching: A survey

<https://scholar.google.com/scholar?hl=en&as_sdt=0%2C5&q=%28%22fuzzy+matching%22+OR+%22entity+matching%22+OR+%22entity+linking%22%29+AND+%28%22rule-based%22+OR+%22heuristic%22%29+AND+%28%22deep+learning%22+OR+%22transformer+models%22+OR+%22language+models%22%29+AND+%28%22survey%22+OR+%22review%22+OR+%22state+of+the+art%22%29&btnG=#d=gs_qabs&t=1744958777109&u=%23p%3D5EvpTdx6e8AJ>

9/10

Neural entity linking: A survey of models based on deep learning

6.5/10

Entity matching using deep neural networks: From discriminative pre-trained language models to generative large language models

Про новые методы разработанные ими

7.5/10

Performance and Interpretability of Entity Matching with Deep Learning

7.4/10

Leveraging Large Language Models for Entity Matching

Deep entity matching: Challenges and opportunities

6.5/10

The fusion of fuzzy theories and natural language processing: A state-of-the-art survey

## ("fuzzy string matching" OR "text similarity" OR "record linkage" OR "entity resolution" OR "approximate string matching") AND ("noisy text" OR "misspellings" OR "typos" OR "abbreviations" OR "text normalization")

8/10

Hybridizing Fuzzy String Matching and Machine Learning for Improved Ontology Alignment

Что делают современные работы:

Гибридные подходы: объединяют несколько метрик (лексические + семантические)

- ML и word embedding (Random Forest + word2vec — [13])

- ML на комбинации метрик (Gradient Boosting, Naive Bayes — [29])

- CNN для структурных и лексических признаков ([30])

BERT-подходы:

- BERTMap ([40], [42]) — контекстные эмбеддинги + структуры

- BERT для профессиональных доменов ([43]) показали, что BERT хорошо захватывает контекст и семантику

Отличие текущего исследования:

- Использует комбинацию fuzzy string matching + BERT + ML-регрессоры (kNN, SVR, DTR)

- Выполняет как лексическое, так и семантическое выравнивание

- Оценивает не только точность (precision, recall, F1), но и время выполнения — чего не делали многие предыдущие работы

8.5/10

Adaptive fuzzy string matching: How to merge datasets with only one (messy) identifying field

String match с одним полем в записи

Авторы не применяют deep learning, но их подход выходит за рамки классических алгоритмов:

Комбинация метрик:

Используются классические метрики (Левенштейн, Jaro-Winkler, Jaccard, косинусное сходство).

Нет нейросетей: Вместо MLP/RNN/Transformer применяется ансамбль метрик через Random Forest.

Human-in-the-Loop (HITL):

Человек проверяет предложенные моделью совпадения, что адаптирует алгоритм под конкретные данные.

Какая то хуета

8.5/10

Using fuzzy string matching for automated assessment of listener transcripts in speech intelligibility studies

[Using fuzzy string matching for automated assessment of listener transcripts in speech intelligibility studies | Behavior Research Methods](https://link.springer.com/article/10.3758/s13428-021-01542-4)

Классические методы

Token Sort Ratio (TSR) — лучший метод из статьи

Расстояние Левенштейна (Levenshtein Distance)

Расстояние Джаро (Jaro Distance)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Если вы используете методы, основанные на расстоянии Левенштейна или Джаро, то сложность сравнения двух наборов строк действительно будет квадратичной относительно количества строк (O(N2 )), умноженной на сложность самого метода сравнения (O(m⋅n)).

Token Sort Ratio также имеет сложность O(m⋅n), так как он основан на других методах (например, Левенштейне).

Ответ:

Да, общая сложность выполнения будет O(N2 ⋅m⋅n) , где N — количество строк в наборах, а m и n — средние длины строк.

9/10

Fuzzy String Matching with a Deep Neural Network

[Full article: Fuzzy String Matching with a Deep Neural Network](https://www.tandfonline.com/doi/full/10.1080/08839514.2018.1448137#abstract)

Классические методы

FCNN и бомжарские эмбеддинги

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

объясни алгоритм с нейросетью

нейросеть берет слово, делает форвард пасс и получает правильный вариант, то есть за один проход набора с ошибками?

но при условии что нейросеть идеально обучена и возвращает только правильные ответы

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

7/10

A Comparative Analysis of Fuzzy String Matching Algorithms for Content-Based Ontology Alignment

[A Comparative Analysis of Fuzzy String Matching Algorithms for Content-Based Ontology Alignment | SpringerLink](https://link.springer.com/chapter/10.1007/978-3-031-85363-0_9)

Рассматриваются только классические методы

6/10

Learning string distance with smoothing for OCR spelling correction

[Learning string distance with smoothing for OCR spelling correction | Multimedia Tools and Applications](https://link.springer.com/article/10.1007/s11042-016-4185-5)

Параметризованное расстояние строк (learning string distance),

Языковые модели (n-gram),

Алгоритм Витерби для поиска оптимальной последовательности исправлений.

### Итог

Я понял, что все эти и многие алгоритмы просто берут два набора данных и возвращают таблицу соответсвий

Эта тема (string matching, spell correction) является более какой то маргинальной и как бы ущербной по сравнению с предыдущей. Предыдущая тема более абстрактная и включает в себя эту.

В этой теме доминируют классические способы оценки схожести строк

Наиболее технологическая статья Hybridizing Fuzzy String Matching and Machine Learning for Improved Ontology Alignment использует BERT для создания эмбеддингов и просто последовательно O(N2) оценивается схожесть эмбеддингов через косинусное расстояние.

## Deezy Match

<https://paperswithcode.com/paper/deezymatch-a-flexible-deep-learning-approach>

# \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Инфо

## Поиск в пространстве эмбеддингов

О, да — ты поднимаешь прямо **ключевую тонкость**, которую важно понимать, когда речь идёт о **поиске ближайших соседей в высоких размерностях**.  
 Давай аккуратно и по сути.

## **📌 Почему в высоких размерностях поиск становится проблемой**

Это называется **проклятие размерности** (curse of dimensionality):  
 Когда размерность пространства (количество признаков в векторе) растёт:

* **Расстояния между всеми точками становятся примерно одинаковыми**
* Вся "геометрия" теряется: ближайший сосед почти так же далёк, как самый дальний
* Обычные структуры типа k-d дерева или R-tree перестают работать эффективно

📉 Пример:  
 В 3D найти ближайшие точки просто.  
 В 10D — уже сложнее.  
 В 500D (эмбеддинги BERT или TF-IDF) — классические алгоритмы вообще разваливаются.

## **📌 Как работают Annoy / HNSW / LSH в больших размерностях**

Ты прав, **точный поиск** ближайших соседей становится медленным.  
 **Вот что делают Annoy, Faiss, HNSW:**

### **🟢 Approximate Nearest Neighbor (ANN)**

Они не ищут **абсолютно ближайшего**.  
 Они строят приближённые структуры:

* **Annoy (Angular/Euclidean trees)**
  + Строит много случайных деревьев с разбиением по гиперплоскостям
  + За счёт случайности и многократности (n деревьев) повышает вероятность найти достаточно близких соседей
  + Работает **быстро и эффективно на средних размерностях (100–300)**
  + Линейно по времени построения, **приближённо log(n) на запрос**
  + В размерностях 1000+ тоже работает, но эффективность падает, и требует больше деревьев

### **🟢 HNSW (Hierarchical Navigable Small World Graph)**

* Строит граф из точек, соединённых короткими и длинными рёбрами
* Обеспечивает быстрый спуск по иерархии  
   → **Работает стабильно лучше других ANN-структур вплоть до 1000-2000 измерений** → Быстро отбрасывает бесполезные участки → log(n) поиск приближённого соседа

### **🟢 LSH (Locality Sensitive Hashing)**

* Хеширует векторы так, что **похожие попадают в один или соседние бакеты**
* Поиск за **почти O(1)** в одном бакете
* Но для этого нужно много хешей при высокой размерности
* Хорошо подходит для 100–500D, хуже на 1000+

## **📌 Почему в совсем больших размерностях всё плохо**

Когда размерность:

* **5000+** (например, TF-IDF по всей Википедии)
* Или **768** (BERT sentence embeddings)

ANN-структуры начинают:

* Требовать очень много памяти (много деревьев или сложных графов)
* Терять точность (приближённый ближайший сосед может быть далеко)
* Замедляться из-за большого числа пересечений и проверок

Поэтому:

* **Векторы обычно уменьшают (через PCA, UMAP, AutoEncoder) до 50–300D**
* Только после этого строят Annoy, HNSW или LSH

## **📌 Как делают на практике**

▶️ **Шаг 1:** Векторизуют строки в 300D (например, Word2Vec, FastText, Sentence-BERT)  
 ▶️ **Шаг 2:** При необходимости сжимают до 50–100D (PCA, UMAP)  
 ▶️ **Шаг 3:** Строят Annoy/HNSW  
 ▶️ **Шаг 4:** Ищут приближённо ближайших за **логарифмическое или константное время**

Если этого не сделать — даже ANN будет работать медленно или плохо.

## **📌 Вывод**

Ты абсолютно прав —  
 👉 **В пространствах большой размерности (500+ D) прямой поиск ближайших соседей — почти всегда долго** 👉 **ANN работает за счёт приближённости и снижения размерности** 👉 **Без фильтрации или редукции размерности — эффективность любой структуры резко падает**

Если хочешь — могу:

* Показать рабочий пример с Annoy на Python
* Или объяснить как PCA или UMAP ужимают размерность  
   Скажешь, что интереснее?

Address parsing

Область:   
named entity recognition (NER)

Adress parsing

# Google

## address parsing using NLP

### Источник1

medium: Address Extraction and Parser with NLP

<https://khadkechetan.medium.com/address-extraction-and-parser-with-nlp-4d3db7b9535d>

Два фрейморка, которые предоставляют возможности для парсинга адресов из естественной речи с ошибками

Flair

<https://flairnlp.github.io/>

- это пайплайновая надстройка над pytorch

Simple Transformer

- надстройка над hugging face transformers

Hugging face transformers

# NER

NER на русском

<https://github.com/deeppavlov/ner/blob/master/example.ipynb>

эксперименты

# Создание Entity Matcher

Статья с несколькими алгоритмами entity matching - а

<https://arxiv.org/pdf/2010.11075>

SEMINT [58, 59]

SMDD [107]

Nin and Torra [69]

Pixton and Giraud-Carrier [77]

Wilson [101]

Tran et al. [95]

NNSM [109]

Gottapu et al. [32]

Reyes-Galaviz et al. [83].

Kooli et al. [51]

DeepMatcher [65]

Wolcott et al. [104]

DeepER [24]

MPM [28]

Kasai et al. [46]

Seq2SeqMatcher [68]

Nozaki et al. [70]

AutoBlock [110]

Hi-EM [111]

Brunner and Stockinger [11]

Ditto [60]

# Поиск

Ход работы

entity matcher - это всегда модель которая принимает 2 набора данных и строит таблицу соответствий

И наша задача разработать или использовать чужой entity matcher

Тогда логично что нужно разбить общую задачу на 2 этапа:

1) Поиск и подбор набора записей (правильных) из БД и создать набор записей с ошибками ( 1 или несколько записей ) от пользователя.

2) подача из в разработанный entity matcher и получение результатов

# Address parsing

То есть задача по сути сводится к обучению простого трансформера на классификацию слов в предложении

# Поиск

Поиск будет осуществляться по соответствию каждого поля запроса ближайшему из всех полей датасета

после нахождения всех соответствий формируется полный адрес

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Я понял, что задачу можно сформулировать так:

1) Классификацию на поля (улица, дом,...) по паттернам, а не по конкретным словам

2) Для каждого поля сделать умный fuzzy match для строчек с ошибками уже из конкретных примеров датасета.

1) тест кв.1

2) тест отдельных чисел

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Address parsing

Формулировка задачу  
  
классификация словосочетаний в предложении на классы: улица, дом, город итд.