



НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ  
УНИВЕРСИТЕТ

Факультет компьютерных наук  
Образовательная программа  
09.03.04 Программная инженерия  
Курсовой проект

# ПОИСКОВИК ПО КОДУ

Выполнил студент группы БПИ-192

Гудзикевич Максим Сергеевич

Научный руководитель:

Разработчик в компании JetBrains – преподаватель, Егор Геннадиевич  
Булычев



# ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

---

## Предметная область задачи

Предметная область задачи – поисковые системы (в нашем случае среди кода).

## Неформальная постановка задачи

Создание базы данных с кодом, в которой можно находить релевантные ответы через поисковые запросы.



# ОСНОВНЫЕ ПОНЯТИЯ, ОПРЕДЕЛЕНИЯ, ТЕРМИНЫ

---

**AST-дерево** – граф, который является деревом с внутренними вершинами (Node), которые являются операторами языка программирования, при этом листьями являются соответствующие им операнды.

**Node** – логическая единица AST-дерева, внутренняя вершина.

**Эмбединг** – отражение векторов в другое векторное пространство. То есть, например, превращение слова в векторное представление.

**CLI** – интерфейс командной строки.

**Датасет** – (в машинном обучении) набор данных для обучения.



# ЦЕЛИ И ЗАДАЧИ РАБОТЫ

---

## Цель работы

Целью работы является создание поисковой системы, которая позволила бы пользователю искать код через запросы на естественном языке.

## Задачи работы

- 1) Исследование области, конфигурация, поиск существующих решений
- 2) Создание MVP
- 3) Итеративное улучшение продукта



# АНАЛИЗ СУЩЕСТВУЮЩИХ РЕШЕНИЙ

---

На момент разработки было выявлено два популярных аналога – поиск внутри GitHub и поиск Google.

Однако у каждого из них есть свои недостатки, самый очевидный – примитивные анализаторы написанного текста.

Google выигрывает по количеству индексируемых страниц – становится очень легко найти нужный функционал, однако нет семантической близости и не все странички (например, приватные репозитории) обязательно будут обработаны.

У GitHub поиск очень примитивный, пара ошибок в запросе всё испортит.



# АНАЛИЗ СУЩЕСТВУЮЩИХ РЕШЕНИЙ

Результаты поиска по запросу “a plus b python example”:

## Python Program to Add Two Numbers

- `a = int(input("enter first number: "))`
- `b = int(input("enter second number: "))`
- `sum = a + b.`
- `print("sum:", sum)`


<https://www.thecrazyprogrammer.com> › 2017/04 › pytho...

## Python Program to Add Two Numbers - The Crazy Programmer

 xiechengin/supreme-xcb  
[xcb\\_ws/RoboMaster-SDK-master/lib/libmedia\\_codec/pybind11/docs/advanced/cast/functional.rst](https://github.com/xiechengin/supreme-xcb/blob/master/lib/libmedia_codec/pybind11/docs/advanced/cast/functional.rst)


```
202 >>> square_plus_1(4)
203 17L
204 >>> plus_1 = func_cpp()
205 >>> plus_1(number=43)
206 44L
207 $ python
208
209 from example import example
```

● reStructuredText Showing the top nine matches Last indexed on 4 Apr 2021

 NessajCN/dronetest  
[lib/libmedia\\_codec/pybind11/docs/advanced/cast/functional.rst](https://github.com/NessajCN/dronetest/blob/master/lib/libmedia_codec/pybind11/docs/advanced/cast/functional.rst)

```
202 >>> square_plus_1(4)
203 17L
204 >>> plus_1 = func_cpp()
205 >>> plus_1(number=43)
206 44L
207 $ python
208
209 from example import example
```

● reStructuredText Showing the top nine matches Last indexed on 20 Apr 2021

 dji-sdk/RoboMaster-SDK  
[lib/libmedia\\_codec/pybind11/docs/advanced/cast/functional.rst](https://github.com/dji-sdk/RoboMaster-SDK/blob/master/lib/libmedia_codec/pybind11/docs/advanced/cast/functional.rst)

```
202 >>> square_plus_1(4)
203 17L
204 >>> plus_1 = func_cpp()
205 >>> plus_1(number=43)
206 44L
207 $ python
208
209 from example import example
```

● reStructuredText Showing the top nine matches Last indexed on 22 Apr 2021



# ФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ

---

## Состав функциональных требований

- 1) Конфигурация;
- 2) Работа с данными;
- 3) Создание и работа с индексом, его настройка;
- 4) Создание CLI;
- 5) Поддержка качества программы;
- 6) Снятие метрик для оценки релевантности.





# СОЗДАНИЕ MVP

---

## Основа для поиска

Для создания MVP был выбран поисковый движок Elasticsearch, в котором можно создавать индексы (аналогично таблице в БД) и выполнять запросы. Для начала было взято 10000 самых популярных репозиторий на Python, информация о которых доставалась с помощью preprocess (библиотека от JetBrains).

Для упрощения работы было создано несколько CLI команд, а также прописаны команды в Makefile для локальной разработки (в самом начале у нас не было удалённой машины).





# СОЗДАНИЕ MVP

## Работа с данными

Перед складыванием в индекс репозитории скачивались с помощью PyGithub, а затем каждая функция рекурсивно обходилась по её AST-дереву, из Nodes которого на каждом уровне забирались необходимые поля (например название функции, docstring или identifiers).

```
1 def a_plus_b(a, b):  
2     return a + b
```

Tree 0.8 ms

```
module [0, 0] - [2, 0]  
  function_definition [0, 0] - [1, 16]  
    name: identifier [0, 4] - [0, 12]  
    parameters: parameters [0, 12] - [0, 18]  
      identifier [0, 13] - [0, 14]  
      identifier [0, 16] - [0, 17]  
    body: block [1, 4] - [1, 16]  
      return_statement [1, 4] - [1, 16]  
        binary_operator [1, 11] - [1, 16]  
          left: identifier [1, 11] - [1, 12]  
          right: identifier [1, 15] - [1, 16]
```

function\_body: “def a\_plus\_b(a, b):\n return a + b”

function\_name: “a\_plus\_b”

docstring: “”



## Конструктор запросов и обёртка над Elasticsearch

Для того, чтобы можно было отправлять более сложные запросы к индексу, был создан класс, который конструирует готовые запросы на основе запроса пользователя. Также нами был создан класс-обёртка над Elasticsearch, который упрощает работу с индексом. Например, он подгружает схемы при создании индекса из отдельной директории, где схемы прописаны в json формате.

```
{'query': 'get columns of dataset',  
'from': 0,  
'size': 5,  
'filters': {'language': ['Python', 'C++'], 'stargazers_count': {'from': 50}}}
```



```
{'query': {'bool': {'must': {'multi_match': {'query': 'get columns of dataset',  
    'fields': ['identifiers^1',  
    'split_identifiers^2',  
    'function_body^2',  
    'docstring^12',  
    'location^3',  
    'function_name^5'],  
    'type': 'most_fields',  
    'fuzziness': 'AUTO',  
    'prefix_length': 3}},  
    'filter': {'bool': {'must': [{'terms': {'language': ['Python', 'C++']}},  
    {'range': {'stargazers_count': {'gte': 50, 'lte': 10000000}}}]}}}},  
'from': 0,  
'size': 5}}
```



# УЛУЧШЕНИЕ MVP

## Оценка поиска

После создания MVP было не очень понятно, как оценить релевантность поиска. Для начала мы создали функции, которые объясняют, почему Elasticsearch присваивет конкретный score.

Однако стало не очень понятно, как оценить, что мы действительно можем найти что-то полезное среди нашей кодовой базы.

В связи с этим был создан функционал по подсчёту метрики top\_n.

True label	Top-3 Predicted labels	Correct
Cat	<b>Cat</b> , Lion, Dog	✓
Dog	Giraffe, Lion, <b>Cat</b>	×
Lion	Cat, <b>Lion</b> , Dog	✓
Giraffe	<b>Giraffe</b> , Dog, Cat	✓
Dolphin	<b>Dolphin</b> , Cat, Giraffe	✓



# УЛУЧШЕНИЕ MVP

## Оптимизация параметров поиска

Для того, чтобы оптимизировать метрику, необходимо было сделать перебор по некоторой сетке параметров. Эта проблема решалась с помощью hyperopt – библиотека работает с моделью как с чёрным ящиком. Теперь пользователь может прописывать сетку параметров, среди которых выбираются лучшие с точки зрения метрики.

```
{'identifiers_weight': {'from': 1, 'to': 15, 'step': 1},  
'split_identifiers_weight': {'from': 1, 'to': 15, 'step': 1},  
'function_body_weight': {'from': 1, 'to': 15, 'step': 1},  
'location_weight': {'from': 1, 'to': 15, 'step': 1},  
'function_name_weight': {'from': 1, 'to': 15, 'step': 1},  
'prefix_length': {'from': 1, 'to': 15, 'step': 1},  
'match_type': ['most_fields', 'best_fields']}
```



```
{'identifiers_weight': {'from': 1, 'to': 2, 'step': 1},  
'split_identifiers_weight': {'from': 2, 'to': 3, 'step': 1},  
'function_body_weight': {'from': 2, 'to': 3, 'step': 1},  
'location_weight': {'from': 3, 'to': 4, 'step': 1},  
'function_name_weight': {'from': 5, 'to': 6, 'step': 1},  
'prefix_length': {'from': 3, 'to': 4, 'step': 1},  
'match_type': ['most_fields', 'best_fields']}
```



## Разметка данных

Чтобы подбирать оптимальные параметры нужен датасет. Размечать руками нашу кодовую базу очень долго, так что я взял данные с соревнования CodeSearchNet. Данные проходят предобработку, дополняются полями, которые лежат в нашем индексе и гипотетически влияют на score. Далее создаётся аналогичный нашему индекс и оптимизация происходит уже на нём.

code	code_tokens	docstring
<pre>def get_vid_from_url(url):\n    """Extracts video ID from URL\n    return match1(url, r'youtu\\.be/([\\^/?]+)') or\n    match1(url, r'youtube\\.com/embed/([\\^/?]+)') or\n    match1(url, r'youtube\\.com/v/([\\^/?]+)') or\n    match1(url, r'youtube\\.com/watch/([\\^/?]+)') or\n    parse_query_param(url, 'v') or\n    parse_query_param(url, 'list')</pre>	<pre>[def, get_vid_from_url, (, url, ), :,   return, match1, (, url, ,,     r'youtu\\.be/([\\^/?]+)', ), or,     match1, (, url, ,,       r'youtube\\.com/embed/([\\^/?]+)',     ), or, match1, (, url, ,,       r'youtube\\.com/v/([\\^/?]+)', ), or,     match1, (, url, ,,       r'youtube\\.com/watch/([\\^/?]+)',     ), or, parse_query_param, (, url, ,,       'v',     )]</pre>	Extracts video ID from URL.
<pre>def sina_xml_to_url_list(xml_data):\n    """str-&gt;list\n    Convert XML to URL List.\n    From Biligrab.\n    """\n    rawurl = []\n    dom = parseString(xml_data)\n    for node in dom.getElementsByTagName('durl'):\n        url = node.getElementsByTagName('url')[0].childNodes[0].data\n        rawurl.append(url)\n    return rawurl</pre>	<pre>[def, sina_xml_to_url_list, (,   xml_data, ), :, rawurl, =, [, ], dom,   =, parseString, (, xml_data, ), for,     node, in, dom, ,,   getElementsByTagName, (, 'durl',     ), :, url, =, node, ,,   getElementsByTagName, (, 'url',     ), [, 0, ], rawurl, ,, append, (, url, ,,     childNodes, [, 0, ], ,, data, ),   return, r...</pre>	str->list\n    Convert XML to URL List.\n    From Biligrab.

dataset\_to\_elastic()





# УЛУЧШЕНИЕ MVP

## Разметка данных

Во всём этом есть загвоздка – **откуда брать запросы?** Очевидное решение – из docstring функции. Но тогда будет сильное совпадение по этому полю, такой поиск будет далёк от реальности. То есть нельзя напрямую в feature (в данном случае часть docstring) складывать target (сущность, у которой одно из полей – docstring).

**Решение:** во-первых, берётся лишь начало docstring (условно первые 30 символов). Далее каждое слово в строке зашумляется с прописанной пользователем вероятностью по такому алгоритму: слово заменяется на случайный синоним из библиотеки nltk, если синонимов нет – случайный символ зашумляется на #. Сам список синонимов просчитывается заранее.

'Andrew Russell Garfield is an  
English and American actor.'

corrupt\_text(probability=0.15)



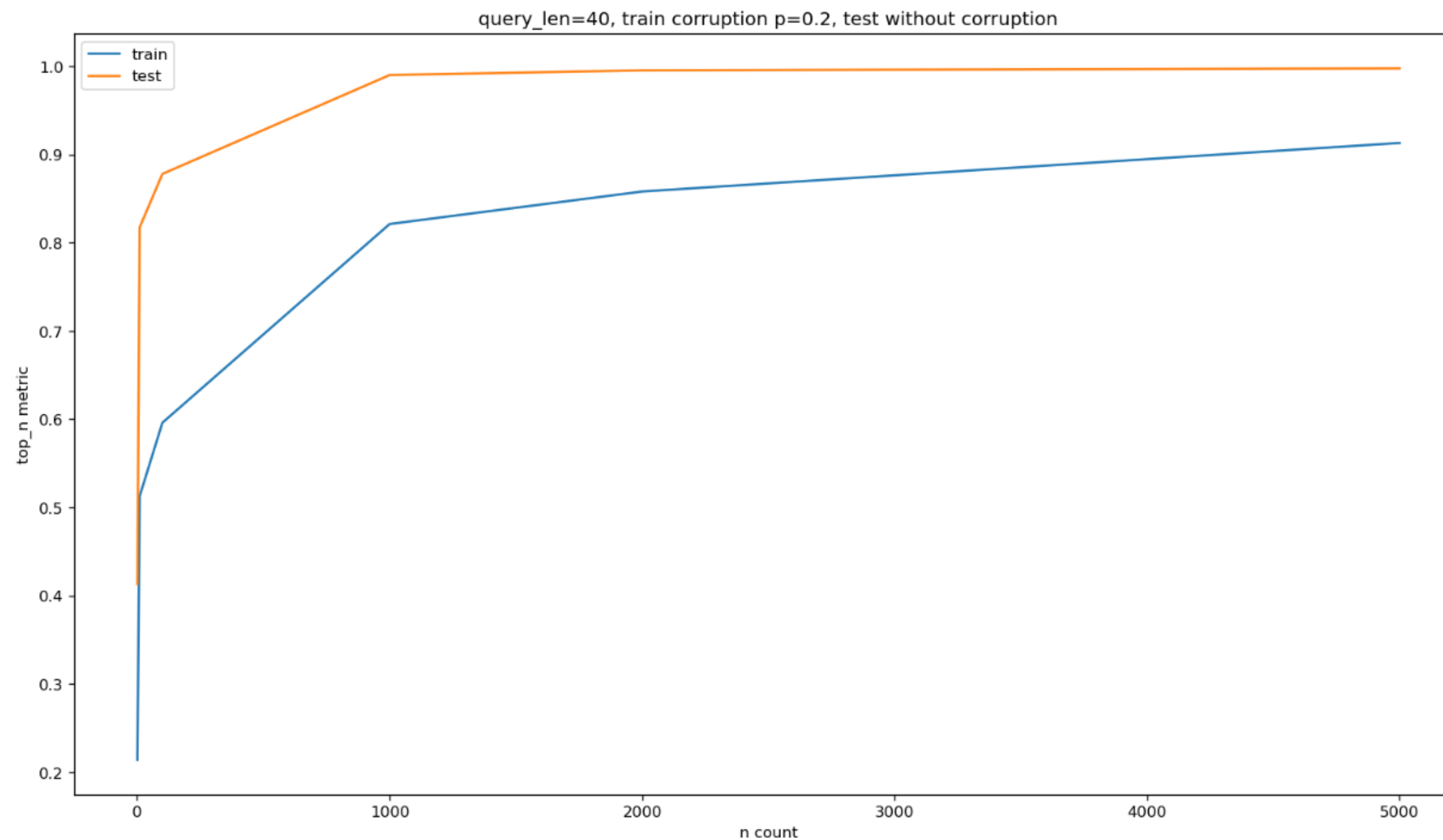
'Andrew Rus#ell Garfield is an  
#nglish and American **histrion**.'





# ОЦЕНКА УЛУЧШЕНИЙ

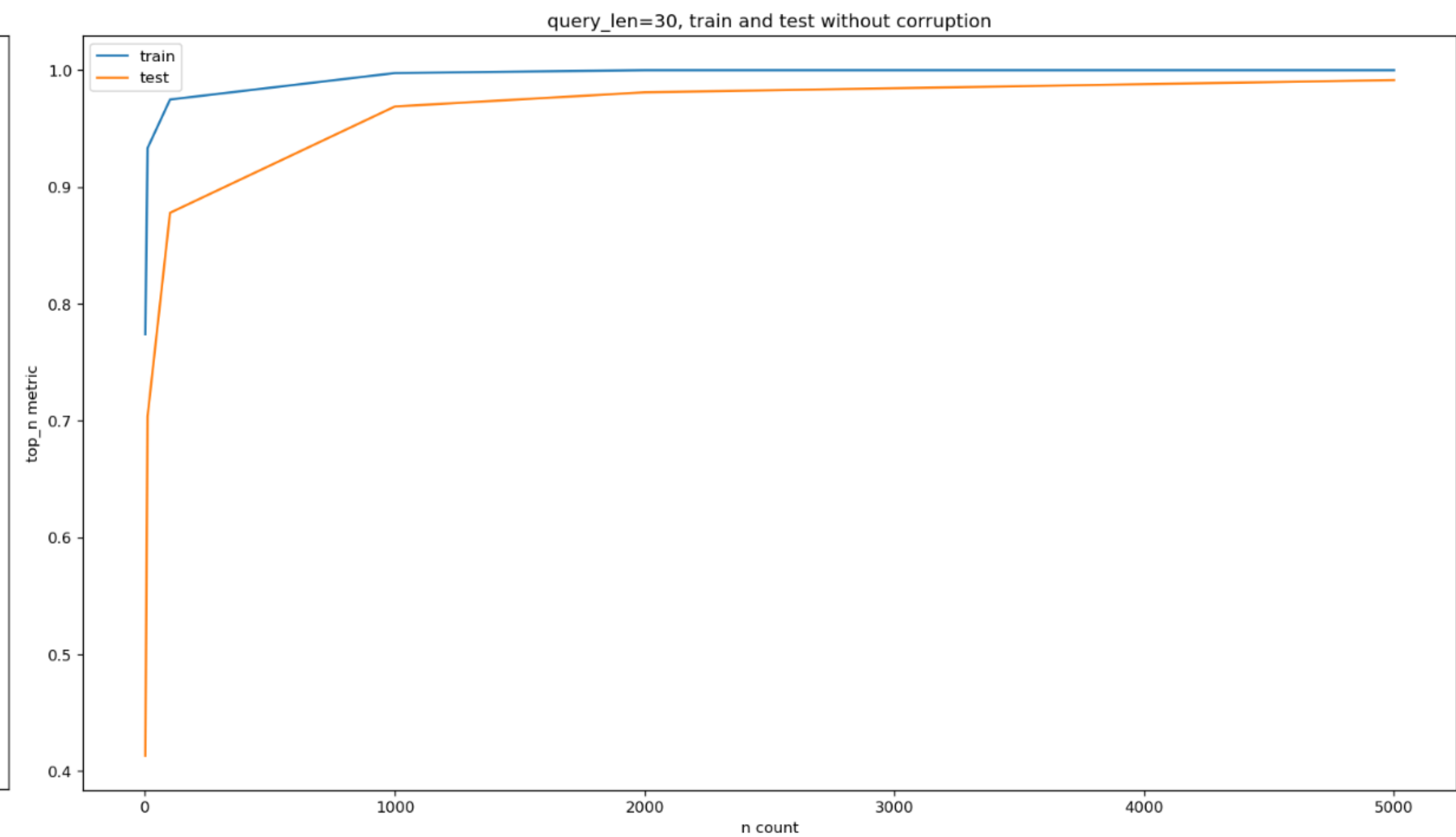
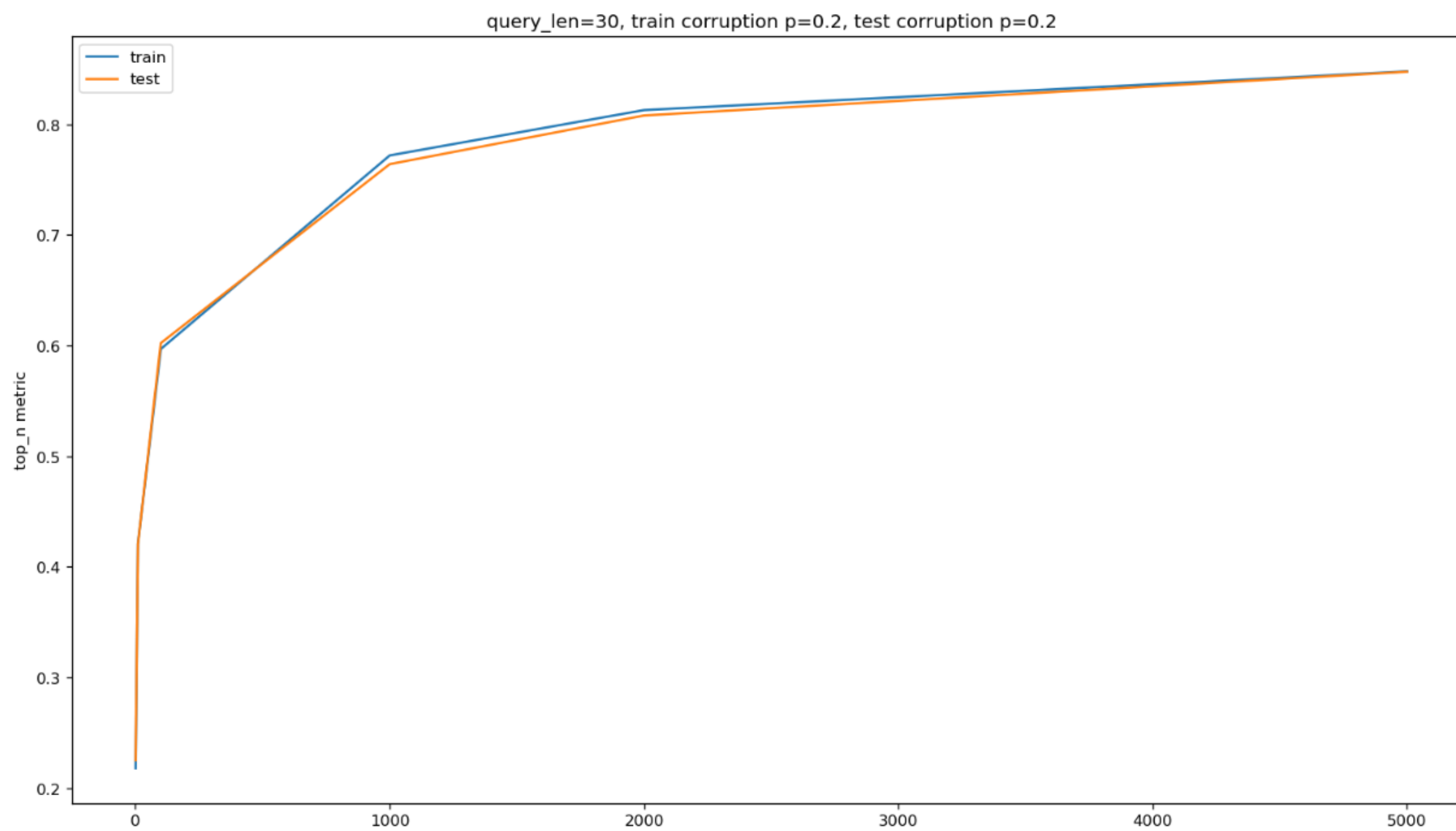
Собирая воедино весь пайплайн можно получить метрики, при которых поиск наиболее оптимален с точки зрения метрики `top_n`:







# ОЦЕНКА УЛУЧШЕНИЙ

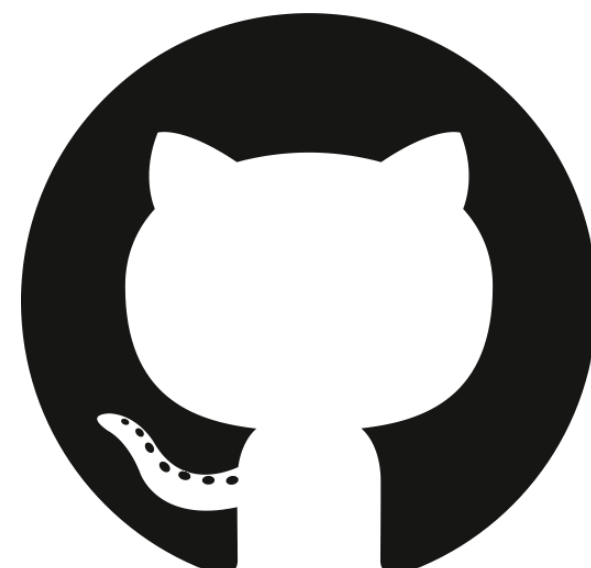




# ТЕХНОЛОГИИ И ИНСТРУМЕНТЫ РЕАЛИЗАЦИИ

## Состав использованных технологий и инструментов

- 1) Python 3.8
- 2) Pycharm 2021.2.3
- 3) GitHub
- 4) WSL 2.0
- 5) Tmux
- 6) Elasticsearch
- 7) Tree Sitter и др.





# ПУТИ ДАЛЬНЕЙШЕГО РАЗВИТИЯ

- 1) Добавление Transformer модели для построения эмбеддингов вводимого запроса и кода (уже в процессе)
- 2) Расширение кодовой базы
- 3) Упрощение индексации нового репозитория
- 4) Тонкая настройка поиска (удаление из топа выдачи сомнительных результатов)
- 5) Встраивание системы внутрь организации или же создание публичного API

```
location: https://github.com/MathInspector/MathInspector/blob/2e2978c7374
language: Python
function_name: Add
function_body:
def Add(a,b):
    """
    a + b : a plus b
    """
    return a + b
```

Результаты поиска по запросу “a plus b example” в Code Search –  
третий по релевантности результат



# ДЕМОНСТРАЦИЯ РАБОТЫ

---



# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

---

- 1) Поиск в кодовой базе GitHub [Электронный ресурс]: GitHub. Режим доступа: <https://github.com> свободный (дата обращения: 17.04.2022).
- 2) Поиск с помощью Google [Электронный ресурс]: Google. Режим доступа: <https://www.google.com>, свободный (дата обращения: 17.04.2022).
- 3) Соревнование CodeSearchNet [Электронный ресурс]. Режим доступа: <https://github.com/github/CodeSearchNet>, свободный (дата обращения: 17.04.2022).



НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ  
УНИВЕРСИТЕТ

СПАСИБО ЗА ВНИМАНИЕ,  
ГОТОВ ОТВЕТИТЬ НА ВАШИ ВОПРОСЫ