

# Обработка датасета:

Запуск:

```
python main.py prepare-data
```

Необходимо было обработать датасет, выделить имена функций из кода при помощи python биндингов tree-sitter.

Я ограничился 1000 примерами. Единственной сложностью была очистка от комментариев/docstring. В результате в датасет были добавлены столбцы *extracted\_name*, *body\_no\_comments*, *body\_with\_comments*.

Примеры обработки:

Примеры

## Doc match: True

- Original name: `get_vid_from_url`
- Extracted name: `get_vid_from_url`
- Name match: True
- Code without comms:

```
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\?v=([^\&]+)')
```

- Original doc: Extracts video ID from URL.
- Extracted doc: Extracts video ID from URL.

## Doc match: False

- Original name: `ucas_download_single`
- Extracted name: `ucas_download_single`
- Name match: True
- Code without comms:

```
html = get_content(url)

resourceID = re.findall(
    r'resourceID":([0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12})', html)
```

```
) [0]
assert resourceId != '', 'Cannot find resourceId!'
```

- Original doc: video page
- Extracted doc:

## Overall результаты

- Total examples: 1000
- Name matches: 1000 (100.00%)
- Documentation matches: 979 (97.90%)

# Использование предобученной модели

---

Запуск:

```
python main.py predict-names -d ./prepared-dataset -m Salesforce/codet5p-
220m -t task_num
```

- task\_num - номер задания (1/2)

В качестве модели использовалась default модель Salesforce/codet5p-220m.

## Код без комментариев

Модель использует такой же токенайзер как *CodeT5* с **extra\_id\_i** токенами. После загрузки модели была идея генерировать на элементах датасета в формате **{extra\_id\_0} + input\_text**. Здесь сложность что модель не всегда в таком случае выдаст только имя функции, но может еще и все определение в формате **def function\_name**. Даже если выпаршивать имя, метрики были очень низкими (EM ~0.012). Поэтому использовал шаблон **def {extra\_id\_0} + input\_text**. Но все равно моделька иногда не оч справлялась и пришлось выпаршивать. Возможно, стоит генерировать с несколькими *extra\_id\_i* токенами.

Оценка происходила по метрикам EM/ROUGE-score; результаты генерации на предобработанном датасете получились следующие:

## Результаты генерации без комментариев

Metric	Score
Exact Match	0.135
ROUGE-1	0.372
ROUGE-2	0.190
ROUGE-L	0.371

Metric	Score
--------	-------

ROUGE-Lsum	0.371
------------	-------

- **0.135** EM находится около значения *0.145*, указанного в задании.
- **0.372** ROUGE-1 в окрестности *0.38*.

## Код с комментариями

Здесь были использованы данные *body\_with\_comments*. Оценка была по тем же метрикам:

### Результаты генерации с комментариями

Metric	Score
--------	-------

Exact Match	0.203
-------------	-------

ROUGE-1	0.465
---------	-------

ROUGE-2	0.273
---------	-------

ROUGE-L	0.462
---------	-------

ROUGE-Lsum	0.463
------------	-------

Как ожидалось метрики повысились, что логично, так как у модельки есть больше контекста для выводения имени функции.