# Introduction

As described in the project proposal, we set out to compare the performance of Generative Entity Retrieval (GENRE) and Elasticsearch-based full-text search methods. We use Python documentation as the knowledge base, documentation sections as documents, and StackOverflow-like questions as queries.

We decided to use the StackOverflow dataset as a source of ground truth query-document mappings. Whenever there's an accepted answer on StackOverflow with a direct link to a specific section of Python documentation, it is safe to assume that the query (question) can be answered with the respective document (linked documentation).

We assume that the StackOverflow answer is a summarization of the embedded link. From that perspective, we posit that search engines can extract the same link as the StackOverflow answer against the same StackOverflow question if they have the entire Python documentation indexed.

# System components

The following are the steps we took to implement our project work.

1. Crawl data from StackOverflow.
2. Crawl data from the Python documentation.
3. Implement a retrieval engine based on ElasticSearch.
4. Feed questions from StackOverflow into ElasticSearch and retrieve the reciprocal rank of the ground truth documentation URL acting as the answer.
5. Implement a retrieval engine based on GENRE.
6. Feed questions from StackOverflow into GENRE and retrieve the reciprocal rank of the ground truth documentation URL acting as the answer.

Note that there are a lot of system components and data processing steps involved in this project. **If you simply want to see resulting data that we used for the project,** inspect `code/reference_doc_import.zip` and `code/entries.json.zip`.

Ilya Andreev was responsible for component (1).

Seungill Kim was responsible for components (2), (3), and (4).

Raj Krishnan was responsible for components (5) and (6).

## **code/pyoracle.py** Crawl data from StackOverflow

We wrote a scraper that uses StackOverflow API to find asked questions with tags python, python3.x, and python3 through the years 2008-2021. The scraper iteratively finds such questions, filters out those without an accepted answer, and then filters out those for which the answer does not link directly to Python documentation. The scraper extracts the context within which the URL was mentioned and successfully handles cases when multiple links to Python documentation are used in a single answer.

We ran this scraper throughout a week to retrieve > 30,000 training and testing examples of question-answer pairs, subject to StackOverflow API rate limiting. These examples are used for GENRE training and both GENRE and ElasticSearch evaluations.

Run this script with `./code/pyoracle.py --start-date=1625252003`

You can optionally provide a `--stackoverflow-key=` flag if you have an API key.

## **code/scraper-reference.py** Crawl data from Python documentation

Step 1: Collect links from the Python documentation Table of Contents.

Step 2: Analyse document structure of the collected links and parse anchor and header links.

Step 3: Crawl text from the links and build JSON entries that include the title, document URL, and text content object.

We only indexed anchor links such as https://docs.python.org/3/library/stdtypes.html#bytearray.join, ignoring top-level links such as https://docs.python.org/3/library/stdtypes.html.

Run this script with `./code/scraper-reference.py`

## **code/ElasticSearch_import.py** Beautify data for ElasticSearch

Before indexing ElasticSearch on Python documentation, we applied some postprocessing to retrieved links. To meet ElasticSearch requirements, we added indexes to each entry and removed new lines from text objects. Below is an example of the resulting JSON file:

```
{"index": {"_index": "reference", "_id": 9370}}
{"text": "This exception is raised when the server unexpectedly disconnects, or when an attempt is made to use
the SMTP instance before connecting it to a server. ", "title": "debugging cgi scripts", "url":
"https://docs.python.org/3/library/smtplib.html#smtplib.SMTPServerDisconnected"}
```

Once you crawl the documentation as in the previous step, you can run this script with `./code/ElasticSearch_import.py`

We set up an ElasticSearch instance on Windows Azure (13.82.107.120:9200). We used Kibana for development and testing even though it was not needed for final evaluation. We configured ElasticSearch to use its default text retrieval function.

We used the command below to import Python documentation into ElasticSearch.

```
curl -X POST "13.82.107.120:9200/reference/_bulk?pretty" -H "Content-Type: application/json" --data-binary
@reference_doc_import.json
```

## **code/ElasticSearch_query.py** Evaluate ElasticSearch

This code queries ElasticSearch with StackOverflow questions and finds the reciprocal rank of the ground truth URL from the accepted StackOverflow answer. To access ElasticSearch from Python, we used the ElasticSearch Client Python library.

In the code, we set the limit on the valid reciprocal rank. We purposely only get up to 200 results from the ElasticSearch.

Run this code with `./code/ElasticSearch_query.py`

## ElasticSearch performance

We tested with 1172 StackOverflow questions, and ElasticSearch matched 320 answers, with approximately 27.3% accuracy.

As seen in the table and the histogram below,

1. The distribution is skewed to the right.
2. Ranks 1-20 constitute 33% of the entire result. The other 67% have a gentle slope.

| **Rank** | **Count** | **Accuracy** | **Cumulative Accuracy** |
|---|---:|---:|---:|
| 1-20 | 105 | 33% | 33% |
| 21-40 | 52 | 16% | 49% |
| 41-60 | 43 | 13% | 63% |
| 61-80 | 21 | 7% | 69% |
| 81-100 | 26 | 8% | 77% |
| 101-120 | 18 | 6% | 83% |
| 121-140 | 14 | 4% | 87% |
| 141-160 | 12 | 4% | 91% |
| 161-180 | 15 | 5% | 96% |
| 181-200 | 14 | 4% | 100% |

**Table 1. ElasticSearch performance overview**

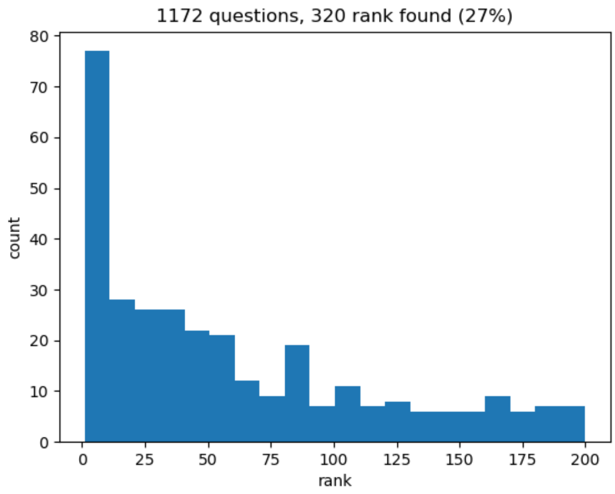


**Figure 1. ElasticSearch ranking breakdown**

## Implement a retrieval engine based on GENRE

### Converting the training data to a compatible format

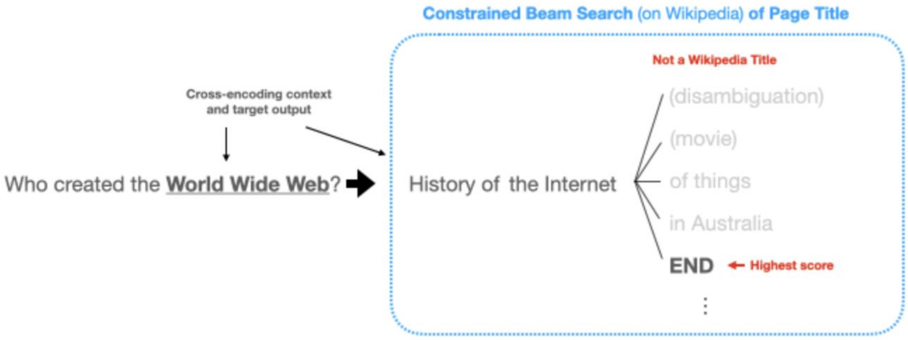
Our training dataset had a format which needed some updates for use with GENRE's training system, which uses fairseq to manage the bulk of the training under the hood. We performed two tasks as a part of this process:

- Split each entry in the training dataset into multiple output entries, with each output entry corresponding to one link in the input data. This allows us to extract the maximum value from the training dataset, and increases the amount of data from 27831 entries to 35670 entries. The data is then assembled into two files, corresponding to the source answer (with entity tags around the hyperlinked text), and the target URL, which can then be passed on to GENRE's training architecture.
- We also convert the URLs to a format that is more amenable to sequence generation in English. We do this as our base model is based on BART, picking the model specializing in English language generation. For example, the URL https://docs.python.org/3/library/constants.html#True is converted to library page constants section True, which can easily be converted to the source URL as a post-processing step.

The code for these steps is available in build_genre_datasets.py.

### Building a Trie to Constrain Output URLs

One of the main ideas proposed in GENRE is the use of a trie, which provides a list of valid URLs to ensure that the output of the model points to a website. GENRE works by generating one token at a time, and assembling the output. An example of this can be seen below (from GENRE's docs), where we have one a set of suggestions by the underlying model, which is constrained to valid Wikipedia page names.


Constrained Beam Search (on Wikipedia) of Page Title
Cross-encoding context and target output
Who created the World Wide Web?
History of the Internet
Not a Wikipedia Title
(disambiguation)
(movie)
of things
in Australia
END
Highest score
⋮

The valid page names are then scored and used to create the next token in the generated sequence.

Each entry in the trie is an id, which corresponds to the encoded value of the token, to allow for faster retrieval of valid tokens (no interconversion between token ids and strings), and minimizing memory usage (as we no longer need to store the whole string in the trie).

The code for these steps is available in build_trie.py.

### Training the Model

We take these two components, and then feed it into GENRE, train the sequence generation model for our task of URL generation. We used UIUC's NCSA cluster to perform the training, which outputs a pytorch model which can subsequently be imported and run on any system where GENRE and pytorch are installed.

## Roadblocks with GENRE

We unfortunately ran into some large blockers with GENRE though, due to the limitations of the underlying training architecture due to which we cannot retrieve valid tokens from the model.

BART, GENRE's base model, encodes data such that an integer maps to every word in the vocabulary. This encoding can be extended to map new words, which might come up in the relevant corpus. This requires us to perform an additional action, which would modify the underlying pytorch models to work with the updated vocabulary.

We ran into a blocker due to GENRE's training architecture at this stage, which used Fairseq to handle the training. Fairseq does not support the addition of new tokens at this stage, as compared to transformers, which allows us to update the vocabulary with a simple function call. Fairseq suggests that we substitute words in the pretrained vocabulary, and use it with our new tokens. This does introduce a source of errors, but is still a reasonable option if we have a limited number of tokens.

We unfortunately need a sizable number of tokens to represent all edge cases in python's documentation links, with over 10k tokens required for a naïve linking approach, which handles the conversion of hyphens and underscores to placeholder tokens. We would hence need to remove tokens which appear less than 17,000 times in the pretraining data.

This introduces significant issues in our model, as we have limited training data, and expect to see these tokens a limited number of times in our training data. The inherent data due to the pretraining data would hence dominate the generation, and break the results that can be obtained.

Due to these issues, we skipped testing the GENRE model, as we cannot run a benchmark between the two approaches, when placing one approach at such a handicap. We would suggest the following next steps when comparing these approaches:

- In case fairseq has not resolved the issue with the addition of new tokens, we should start with migrating the training architecture to use transformers.
- At the same time, we should explore more complex remapping techniques for links, to minimize the number of tokens.
- Expand the number of stack overflow answers available, or switch to a documentation base with a smaller number of tokens, to limit the impact of a small training dataset.

## Acknowledgements

*This work utilizes resources supported by the National Science Foundation's Major Research Instrumentation program, grant #1725729, as well as the University of Illinois at Urbana-Champaign.*

## References

- Autoregressive Entity Retrieval: https://arxiv.org/abs/2010.00904
- GENRE Implementation: https://github.com/facebookresearch/GENRE