Final Year Project Interim Report

Source Code Search Engine: CodEX

Xingyu Pan	14207156
Mengya Zheng	14207169
Ke Xu	14207177
Ciarán Connolly	14445308

A thesis submitted in part fulfilment of the degree of

BSc. in Software Engineering

Supervisor: David Lillis

UCD School of Computer Science

Beijing-Dublin International College

University College Dublin

08/04/18

1. Introduction

As the typical representations of the textual search engine, Google, Bing and Yahoo achieved great success in this area. However, those search engines do not work very well when used for searching source code. Moreover, specialised code search engines such as Search Code, Ohloh Code, and Krugle only support keyword-based searching, which is not as smart as textual search engines. In this project, we aim to combine the code searching ability of code search engines with the ingenuity of textual search engines to provide users with a more intuitive code searching experience. We call our search engine CodEX (Code Explorer).

We have amassed a collection of code from GitHub and Stack Overflow which we plan to grow. Our corpus currently consists of over 250Gb of uncompressed projects from GitHub and 60Gb of questions and code snippets from Stack Overflow. At present, our project is split into two parts: a keyword-based code search engine and a code-snippets searching tool supporting Python and Java. Users can search our code base by entering one or more keywords, and all relevant results will be returned. In addition to this, users can search using code snippets or entire programs, and they will be presented with similar code blocks from other programs in a ranked list. The code snippet searching tool can be used to detect plagiarism and look for usage of query code. Thanks to the combination of our indexing and search methods along with our distributed framework implementation, CodEX can efficiently search large corpora.

Over the coming weeks, we plan to add further features such as natural language queries and recommended results as well as improving upon existing features. We also plan to improve the backend of the code and thoroughly evaluate the performance of CodEX.

2. Related Works

2.1 Natural Language Processing & Stopwords

There has been much research into summarising code and allowing it to be queried using natural language. Many of the papers provided us with ideas and inspiration on how we might go about allowing natural language queries from users.

Some papers explore the idea of summarising source code. An algorithm is developed in [1] to "extract and generate natural language phrases from source code". Similarly, Wang et al. [2] looks at summarising code but through the use of a graph. Iyer et al. [3] discuss an interesting method of summarising code snippets from Stack Overflow using question titles and code snippets from the answer. Using summaries of the code could be used for natural language queries by combining it with other tools such as a query parser.

A number of papers we have read investigate ways of querying code using natural language. Panchenko et al. [4] and Kimming et al. [5] rely heavily on third-party tools for indexing and searching code which is something we wanted to avoid. Raghothaman et al. [6] present a method of finding code snippets based on API-related queries. However, unlike the other two papers, it describes algorithms for indexing and searching code which is more closely related to what we originally set out to achieve.

Carvalho et al. [7] explore a method of "converting program identifiers to sets of full natural language (including domain specific) terms". Such a solution could be useful in natural language summaries and queries.

2.2 Code snippet searching (Plagiarism Detection)

Regarding code snippet searching, Greenan [8] introduced a method-level code searching algorithm applied for plagiarism detection in which programs are transformed into Abstract Syntax Trees (AST). In this way, the program structures are clearly recorded while unimportant symbols (e.g. colons) and text (e.g. strings, comments) are ignored. Subtrees of methods are transformed into string sequences and then compared line by line to get intersections. Once the number of entities in intersects exceeds a prescribed threshold, these methods are clone candidates. However, despite the advantages brought about by code intersection and AST, the method used in this article is still limited in method level because string sequences cannot represent the hierarchical structure of a whole program. Another drawback of this method exists in the line-by-line matching process on the original code, which is sensitive to the identity name changing and insertion/deletion modification.

Similar to Greenan's method, Chilowicz et al. [9] also used an AST to represent the programs by splitting them into multiple subtrees and recorded them as digested forms called "fingerprints". These "fingerprints" are also used by some other code snippet searching system and show high efficiency. For example, Moss [10, 11], Sourcerer [12] and Nilsson's plagiarism detection system [13]. These "fingerprints" reduce the demand on memory for corpus indexing and enable depth-first searching through hashed subtrees. The disadvantage of Chilowicz's system, however, is that it detects plagiarism only when hash values are the same, making detection sensitive to some tricky plagiarism techniques such as independent statements reordering and code insertion/deletion.

Considering the advantages and disadvantages of approaches talked about in these papers, some efficient techniques including ASTs, fingerprints, and intersection thresholds are applied with some modifications to the plagiarism detection module of CodEX.

2.3 Code Searching

There are several code search engines [14] developed based on Latent Semantic Indexing (LSI) [15], which frequently considers co-occurring tokens as being similar or related and, thus, is useful in solving synonym problems. This method also reflects the most significant features of corpora through dimension reduction on term-document matrices. This method can be applied to search through documents and source code based on matching keywords. According to the LSI, matrices transform algorithms introduced in [15], the big term-document matrices can be simplified to a 2-d LSI subspace where documents are represented as vectors.

3. Completed Work

The following section gives an overview of the work we have completed so far and is accompanied by an architecture diagram which can be found in (Appendix B: Architecture Diagram) of this document. This diagram will be explained in following parts.

3.1 Environment Preparation

We rented three servers from Tencent Cloud [16]. Each one with one core,1 MB per second bandwidth, 2GB RAM and 50GB storage, installed Ubuntu 16.0.4, Python 3.5 and corresponding modules.

3.2 Distributed Framework

To balance the load of our servers, we designed a mechanism to allocate the searching process among three servers, which involves using a distributed framework. The framework was made by using Python socket and multiple threads. As seen in (Appendix A: Deployment Diagram), clients send queries directly to the main server. The main server then transmits these queries to the two slave servers which search within their own corpora in parallel with the master one. All the results will be sent back to the main server. By applying this mechanism, our search engine could deal with bigger corpora with higher efficiency.

3.3 GitHub Crawler

The GitHub Crawler is a customized web crawler developed with python. As we found that GitHub.com is the most popular website which is used to host open-source software projects and provides many user-friendly and comprehensive APIs [17], we decided to build a highly customized GitHub Crawler by making API calls instead of using a generalized crawler framework. Specifically, the library "requests" [18] is used to handle internet requests and file downloading.

The first step is to call the GitHub Search API in order to get a list of repositories, which are ranked by their user ratings, or stars, in a specific programming language. By using the highest rated projects, we can help to guarantee that the code snippets we obtain are of high quality. Then, the URLs of the repositories that we wish to download can be retrieved from the JSON data of the Search API results, and the content of each repository can be downloaded as a single zip file. At the same time, a JSON file which contains valuable information about the repository, such as the project name, the author name and the last updated time, etc., is generated and stored accompanied with the zip file. In addition, the downloading records are stored in a JSON file in order to maintain the crawling request queue and facilitate the regular update check.

During these steps, we also need to consider the GitHub API rate limit issue. According to the API documentation, an authorized account can make up to 30 API requests per minute. Thus, we control the speed of making API search requests so as not to exceed the rate limit and, if necessary, we can register multiple accounts to break through this rate limit and enable the crawler to handle multiple search and downloading requests at the same time. However, due to the bottleneck of our server's storage space, we only choose to crawl a limited number of repositories from GitHub.com.

Robustness of the GitHub Crawler is achieved through different ways. Due to the instability of the network condition of our servers and the fluctuation of the servers used by GitHub.com, we need to build our crawler in a robust way. We have come across many exceptions which may occur, such as HTTP connection failures and incomplete downloaded files. Therefore, the timeout parameter is set to 15 seconds for a reasonable API calls or download requests. If the connection breaks down during downloading, the crawler will restart the download request again and guarantee all the repositories we want can be collected successfully. In addition to this, the integrity of the zip files will be tested again after the downloading has finished to further ensure their robustness.

Furthermore, to realise downloading progress visualisation, the library "tqdm" [19] is integrated into our crawler and enables the downloading progress bar displayed in CLI for each download request.

3.4 Data Cleaning

After all the zipped projects and their corresponding JSON files have been saved to the servers the next step is to unzip and clean them. Each server contains a “clean” and “unclean” directory. Within the unclean directory are all the unzipped projects. Each project is unzipped individually and saved in the clean directory under the same name.

Many of the files in these projects are not source code files and, thus, are of little use to us. As such, the next step is to delete all these unwanted files. This allows us to save some valuable storage on the servers and reduce the possibility of one of these files from causing issues later.

Finally, the projects in the clean and unclean directories are compared to ensure that all projects have been unzipped and to record any that couldn’t be unzipped.

3.5 Generate Formatted JSON Files

The next stage is to create a JSON file for each source file in the projects. The first step is to create a dictionary with the project names as the keys and their corresponding JSON data as the values. Once we have recorded all the projects in the dictionary, we can begin to create new JSON files for each individual source file.

Each project is recursively searched for source code files and, when a file is discovered, we create a FormattedCodeInterface (FCI) object. An FCI object stores information about the file such as the project author, the contents of the file, and the project URL.

Finally, once an FCI object is created, it is converted to a JSON file and saved to a directory containing all JSON files. Additionally, if the JSON file was created on one of the slave servers, a copy will also be saved to the master server. As such the master server will store all JSON files created on all servers.

3.6 Remote Connection to Servers

To enable communication between the servers we used a python module named Paramiko. “Paramiko is a Python (2.7, 3.4+) implementation of the SSHv2 protocol, providing both client and server functionality” [20]. This module enables connection to a remote machine via SSH and provides much functionality including the ability to execute commands remotely and copying files from one machine to another.

3.7 Generate Stopwords

As a small part of a search engine, stop words are always ignored. However, accuracy would increase a lot if proper stop words were to be applied. We can easily find existing natural language stop words from the internet, but no code stop words are given. Therefore, we built our own stop word list using statistical methods.

3.8 Keyword-Based Code/Text Searching:

The keyword-based searching has been finished using Latent Semantic Indexing (LSI) algorithm and merged with the front end. As Figure 2 shows, the corpus will be represented as a term-document matrix after indexing. These matrices are stored in a “pickle file”, a file processed by a Python Standard Library called “Pickle” [21] that supports the storage of standard data structures in Python. Next, through matrix transformation on the stored term-document matrix, accepted queries and documents are represented as vectors with the same number of dimensions. Finally, cosine similarities between query vectors and document vectors are worked out and sorted into a ranked list where the document with the highest similarity ranks on the top. For every relative document, we also have recorded the matching lines and highlighted them on the web pages. This module has been tested, and a problem on the single value decomposition is found out, which will be fixed in the future.

3.9 AST Based Code Snippet Searching (Python):

This module was developed based on Abstract Syntax Trees (AST) and now is capable of code snippet searching in Python. By now, as Figure 2 illustrates, the noise cleaning has been done, in which the identifier names, strings, output statements and meaningless symbols are removed from the ASTs of documents. After this, in the indexing phase, cleaned AST is turned to standard Python data structures, called fingerprints, without ignoring their hierarchies. In these data structures, the weights, start line number and end line number of subtrees are recorded in a pickle file. The last phase is the retrieval phase, in which we accept queries from users, preprocess them in the same way we used on documents and, finally, find matching subtrees from the pickle file. Based on the weights of these matching subtrees we worked out a similarity proportion for every document, and if this proportion exceeds a prescribed threshold, then this document is defined to be similar to the query so that users can judge the plagiarism cases themselves. In merge phase, the ranked lists of these documents are returned to the front end along with matching line numbers. Therefore users can see the plagiarized blocks highlighted in both the query code and the code in matching documents.

In the module of LSI based searching and AST based plagiarism detection, we have applied Redis server, an in-memory database supporting key-value storage with self-defined durability, to cache data in an efficient manner. All the results sent to the front end have been encapsulated as an object designed by ourselves.

3.10 Web Application

Compared with traditional Client-Server architecture, web applications provide users with less of a storage burden. Nothing to download, easy to use web applications are also common within large companies. Therefore, we chose to construct a web application to represent CodEX. In the beginning, Java was chosen as the development language. We also involved Struts 2 [22], Spring [23], and Hibernate [24] together as the framework. However, we found it consumed too much time connecting a Python searching module to a Java Web Application during the first iteration, so we turned to build our web application by using Django [25] module in Python. The application can be divided into two parts, code searching and plagiarism detection.

3.11 Log Tool and Interfaces

CodEX is a large project which is deployed onto three different cloud servers, so it is necessary to write logs to keep track of the system status. We wrote a tool script to help us output a different kind of logs into log files. Furthermore, we also made some interfaces to transfer information among different parts of the system. FormattedCodeInterface (FCI) and FCIConverter are the most important interfaces in the system. Those interfaces are used to store cleaned code into JSON files and save them to their corresponding location.

3.12 Stop Words Generating

Stop words are some words that exist in the majority of documents. Those words are not helpful for searching engine to locate the documents that users want. Therefore, we need to find out those stop words and get rid of them. We applied statistic method to find stop words list. Words which occurs most frequently (top 5%) would be put on our list.

4. Future Works

This section will illustrate the work we plan on completing over the next six weeks. Every task below will have somebody take charge of it. There is also a Gantt Chart in the appendix (Appendix C: Gantt Chart) which gives a clear overview of every member's tasks and time frame for completion.

4.1 Natural Language Query

4.1.1 Overview

In the next stage of the project, Ciarán and Kirk will work on allowing users to query the code using natural language. For example, a user may search for "methods with a variable number of parameters in python" and be presented with a list of code snippets matching their query. We have yet to decide on how to proceed, however, we have researched numerous methods, which we will discuss in the following section, and will select one based on time, complexity, and suitability.

4.1.2 A Basic Approach - Mapping the Text Around the Source Code

Like some previous ways of searching images and videos, people map the text appeared around the non-text content so that the natural language query can retrieve the non-text content successfully [26]. However, the accuracy of relevant text mapping based information retrieval mostly depends on the quality of the relevant text. We are trying to find some ways to realise how to handle natural language queries for searching source code using this approach in a reasonably accurate way.

Initially, we plan to use only code snippets from StackOverflow for two main reasons. Firstly, it is easier to retrieve a code snippet and its question title from Stackoverflow than a code snippet and its matching comment from a GitHub project. Secondly, in general, we expect a question title to give a more concise and comprehensive description of a code snippet than a comment in a project. However, we hope to expand our work by generalising our approach to work with projects from GitHub.

4.1.2.1 Preprocessing

The preprocessing method used by S. Iyer et al. [3] provides us with an idea as to how to apply such a method to content from StackOverflow. From archive.org, we can easily collect all the questions and answers posted on StackOverflow. However, question and answer descriptions tend to be "domain-specific and verbose, mixed with details that are irrelevant for our tasks". As such, Kirk will adapt the method from [3] to collect question title, code snippet pairs from those questions which only contain one code snippet in their accepted answer.

4.1.2.2 Natural Language Processing Keyword Retrieval

The next step is to extract key information from natural language text. This text will consist of users' search engine queries and question titles from StackOverflow. We propose a general technique to achieve such an idea by using NLP libraries and tools, such as the Natural Language Toolkit (NLTK) [27] and Snips NLU [28]. Ciarán will process text, such as those mentioned above, to extract information from them.

Using a query parser we can also process natural language queries and extract key information from them. We can train the dataset to handle common user queries. For example, if a user searches for "How to create a HashMap in Java?" the keywords "HashMap" and "Java" will be identified and used to find code snippets matching the query. Similarly, we can use NLTK to extract information to extract information from StackOverflow question titles.

4.1.2.3 Information Retrieval

The last step is to retrieve the code snippet by natural language queries. As we have already linked the code snippet with the corresponding keywords in our preprocessing part, an IR algorithm can be used to conduct the code retrieval. The search results are based on the matching result between the well-structured natural language queries and the keywords around the code snippet. This piece of work will be finished by Kirk.

4.1.2.4 Generalizing Our Approach

This method can also be generalised to the GitHub repositories in which a large number of comments could be used to conduct the mapping technique. However, we need to come up with a new technique for mapping the many lines of codes in one projects with their corresponding limited numbers of comments. If we complete the previous sections, then Ciarán and Kirk will work together to realise this goal.

4.1.3 Real NLP - Summarizing the Meaning of the Source Code

As discussed in the 'Related Work' section, previous papers have presented methods of summarising code in natural language or through a graph structure. By combining such summaries with our information retrieval approach from the previous section, it may allow for more accurate results to natural language queries.

We have explored using the method presented in [3] to allow us to summarise code snippets from Stack Overflow. Using this summary combined with methods from other papers or through the use of the Natural Language Toolkit we could allow users to query code using natural language.

This work will also be taken up by Ciarán and Kirk.

4.2 Map Reduce & Data fusion

Although we have already implemented a distributed framework for our current version, some mechanism like resource lock has yet to be achieved. However, MapReduce mechanism by Dean and S. Ghemawat [29] will assign tasks and merge result easily. "Map" represents a mapping function, used for task assignment while "Reduce" takes charge of result merging. Xingyu will apply MapReduce for process synchronisation between 7th and 8th week. The next task is to merge those result lists computed by two slave servers. The rank of files should be considered most in this part.

4.3 Code Snippet Searching Tool

The tasks in this section will be fulfilled by Zheng Mengya.

4.3.1 Improve the Code Snippet Searching Tool Supporting Python

Applied to plagiarism detection scenarios, our existing code snippet searching tool is capable of detecting most categories of plagiarism techniques, but it is not developed for some more complex plagiarism avoidance techniques. Therefore, we aim to improve this tool to detect them. Some of these techniques, and methods which can be used to combat them include:

- Combining several code blocks copied from multiple documents. To detect this kind of plagiarism technique, we will conduct a global search to find and record the matching subtrees between the query code and the whole corpus.
- Inserting meaningless code lines which do not affect the program function into plagiarised code blocks. To combat this technique we wish to define a definition of what constitutes code whose only purpose is to help avoid plagiarism detection. For example, creating new variables which are not used within the program or code which is never reached. After these specific definitions are determined, this type

of code can be ignored from documents and queries in the preprocessing phases. However, because of the difficulties in creating these definitions, this task might be discarded at the end.

We also plan to tune thresholds to give more accuracy and better performance. These thresholds are:

- The similarity threshold: if the similarity percentage between a query and a code snippet exceeds the chosen threshold, then it is suspected to have been plagiarised.
- The weight threshold: if the weight of a subtree's node is less than the chosen threshold it will be pruned from the original tree. This will help increase the performance of the system as it removes the need to compare unimportant subtrees.

These tasks will be done in week 7 and week 8.

4.3.2 Testing on Code Snippet Searching Tool in Python

To test the function of plagiarism detection, different categories of plagiarised programs will be created as testing cases to test what kinds of plagiarism techniques our system can detect. In this process, thresholds can be tuned based on testing results to get the highest performance. And this task is predicted to be fulfilled in week 9.

4.3.3 Implement Code Snippet Searching Tool Supporting Java

To search with a code snippet, code parsing is necessary but is difficult and time-consuming. Due to the time limit, we are unable to implement a code snippet searching tool for every language. We choose to implement this tool in Python and Java. Since this tool has been implemented in Python based on AST API and there is also an AST API in Java called "Package org.eclipse.jdt.core.dom" [30], the implementation of the Java version should be similar except for the code cleaning phase because the output statements in Java are different from that in Python. This part will be finished before week 9.

4.4 Fix the Problem in the Keyword-Based Searching Tool

We have encountered some problems in keyword-based code searching system, and the real reason has not been ascertained yet. This will be researched and addressed by Zheng Mengya in week 10.

4.5 Update CodEX to Handle Other Languages

Currently, CodEX only uses a code base of python projects. This means that users can only search with code written in python. We plan to expand our project by enabling it to parse, index and search Java code. This will require some changes in existing code.

4.5.1 Code Cleaning

When deleting unwanted files from the unzipped projects, we check that each file is not a python file and delete it if so. This will need to be updated to not delete java files from a project written in Java.

4.5.2 Generating JSON Files

Part of creating a JSON file for each source file in a project involves separating the code from the comments. This is done by using a regular expression. However, a regular expression used to detect comments will be different for every language. As such, a new regular expression will have to be written to handle Java comments.

4.6 Web Application

The web application is the presentation of CodEX. During week 7th and week 8th, Xingyu will revise some pages of web application and add more details of code snippets. By the end of 8th week, our CodEX will have strong robustness and be user-friendly.

4.7 Evaluation

Evaluation is a crucial way to make sure our code search engine works well and find out weaknesses in it. During week 9th and week 10th, Xingyu will apply Cranfield paradigm to evaluate our search engine. Furthermore, some volunteers might be invited to achieve a better result.

4.8 Final Report

In the last two weeks (11th and 12th week), all of us will start to write our final report. The report will illustrate what we will have done individually. Moreover, potential bugs or problems will also be fixed during those two weeks.

5. Conclusion

In this report, we have presented the current implementation of our source code search engine called CodEX. We also discussed some related material as well as our future plans CodEX combines the ability of source code search engines with the ingenuity of search engines. It currently comprises of a keyword search engine and a plagiarism detection tool. However, we aim to expand upon the current implementation by adding a natural language query capability and a recommendation system. It achieves respectable speed and accuracy thanks to the indexing and search techniques we have used. On top of this, our distributed framework provides CodEX with the ability to search large corpora in an efficient manner. This allows users to search throughout a larger code base and get more relevant results.

6. References

[1] E. Hill, L. Pollock and K. Vijay-Shanker, “Automatically capturing source code context of NL-queries for software maintenance and reuse,” In Proceedings of the 2009 IEEE 31st International Conference on Software Engineering, Vancouver, BC, 2009, pp. 232-242.

[2] S. Wang, D. Lo, and L. Jiang, “Code Search via Topic-Enriched Dependence Graph Matching,” In Proceedings of the 18th Working Conference on Reverse Engineering, Limerick, 2011, pp. 119-123.

[3] S. Iyer, I. Konstas, A. Cheung and L. Zettlemoyer, “Summarizing Source Code using a Neural Attention Model,” In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), 2016, pp. 2073-2083.

[4] O. Panchenko, S. Müller, H. Plattner and A. Zeier, “Querying Source Code Using a Controlled Natural Language,” In Proceedings of the Sixth International Conference on Software Engineering Advances (ICSEA), 2011, pp. 369-373.

[5] M. Kimming, M. Monperrus, M. Mezini, “A Natural Language Interface for Code Search,” Ph. D. dissertation, TU Darmstadt, 2011.

[6] M. Raghothaman, Y. Wei and Y. Hamadi, "SWIM: Synthesizing What I Mean - Code Search and Idiomatic Snippet Synthesis," In Proceedings of the 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE), Austin, TX, 2016, pp. 357-367.

[7] N.R. Carvalho, J.J. Almeida, P.R. Henriques and M.J. Varanda, “From source code identifiers to natural language terms,” *Journal of Systems and Software*, vol. 100, no. 1, Feb., pp. 117-128, 2015.

[8] K. Greenan, “Method-level code clone detection on transformed abstract syntax trees using sequence matching algorithms,” Student Report, University of California-Santa Cruz, 2005.

[9] M. Chilowicz, É. Duris and G. Roussel, “Syntax tree fingerprinting: a foundation for source code similarity detection,” In Proceedings of IEEE 17th International Conference on Program Comprehension, 2011, pp. 243-247.

[10] Ada Resource Association, “Ada 95 Documents,” 2018. [Online]. Available: http://www.adaic.org/ada-resources/standards/ada-95-documents/. [Accessed Apr. 7, 2018].

[11] S. Schleimer, D.S. Wilkerson and A. Aiken, “Winnowing: Local algorithms for document fingerprinting,” In Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, 2003, pp. 76-85.

[12] S. Bajracharya, T. Ngo, E. Linstead, Y. Dou, P. Rigor, P. Baldi, and C. Lopes, “Sourcerer: a search engine for open source code supporting structure-based search,” In Proceedings of the Companion to the 21st ACM SIGPLAN Symposium on Object-oriented Programming Systems, Languages, and Applications, 2006, pp. 681-682.

[13] E. Nilsson, “Abstract Syntax Tree Analysis for Plagiarism Detection,” Dissertation, 2012.

[14] A. Marcus and J. I. Maletic, “Recovering documentation-to-source-code traceability links using latent semantic indexing,” In Proceedings of the 25th International Conference on Software Engineering, 2003, pp. 125-135.

[15] S. Deerwester, S.T. Dumais, G.W. Furnas, T.K. Landauer and R. Harshman, “Indexing by latent semantic analysis,” *Journal of the American society for information science*, vol. 41, iss. 6, Sep. 1st, pp. 391-407, 1990.
[16] Tencent Inc., “Tencent Cloud,” 2018. [Online]. Available: https://cloud.tencent.com/?lang=en. [Accessed Apr. 8, 2018].

[17] GitHub Inc., “GitHub REST API v3,” 2018. [Online]. Available: https://developer.github.com/v3/. [Accessed Apr. 8, 2018].

[18] K. Reitz, “Requests: HTTP for Humans,” 2018. [Online]. Available: http://www.python-requests.org/en/master/. [Accessed Apr. 8, 2018].

[19] Casper da Costa-Luis, Stephen L., Hadrien Mary, noamraph, Mikhail Korobov, Ivan Ivanov, Marcel Bargull, Guangshuo Chen, James, Matthew D. Pagel, Staffan Malmgren, Socialery, Jack McCracken, Fabian Dill, Daniel Panteleit, Alexander, Alex Rothberg, Yaroslav Halchenko, Tomas Ostasevicius, Shirish Pokharel, ReadmeCritic, Peter VandeHaar, Kuang-che Wu, jcea, Hugo, Ford Hurley, Edward Betts, David Bau, Arun Persaud, and Adnan Umer, “tqdm/tqdm: tqdm v4.21.0 stable,” Zenodo, Apr. 08, 2018. [Online]. Available: https://zenodo.org/record/1214899#.WsnF1IhuY2w. [Accessed Apr. 8, 2018].

[20] J. Forcier, “Paramiko,” 2018. [Online]. Available: http://www.paramiko.org/index.html. [Accessed Apr. 8, 2018].

[21] Python Software Foundation, “pickle — Python object serialization,” 2018. [Online]. Available: https://docs.python.org/3/library/pickle.html. [Accessed Apr. 8, 2018].

[22] The Apache Software Foundation, “Apache Struts,” 2018. [Online]. Available: https://struts.apache.org/. [Accessed Apr. 7, 2018].

[23] Pivotal Software, Inc., “Spring Framework,” 2018. [Online]. Available: https://spring.io/. [Accessed Apr. 7, 2018].

[24] Red Hat Inc., “Hibernate Framework,” 2018. [Online]. Available: http://hibernate.org/. [Accessed Apr. 7, 2018].

[25] Django Software Foundation and individual contributors, “Django,” 2018. [Online]. Available: https://www.djangoproject.com/. [Accessed Apr. 7, 2018].

[26] A. Vadivel, S. Sural and A. K. Majumdar, "Query Refinement for Internet Multimedia Information Retrieval Using Keywords and Low-level Features," In Proceedings of the International Conference on Computational Intelligence and Multimedia Applications (ICCIMA 2007), Sivakasi, Tamil Nadu, 2007, pp. 192-196.

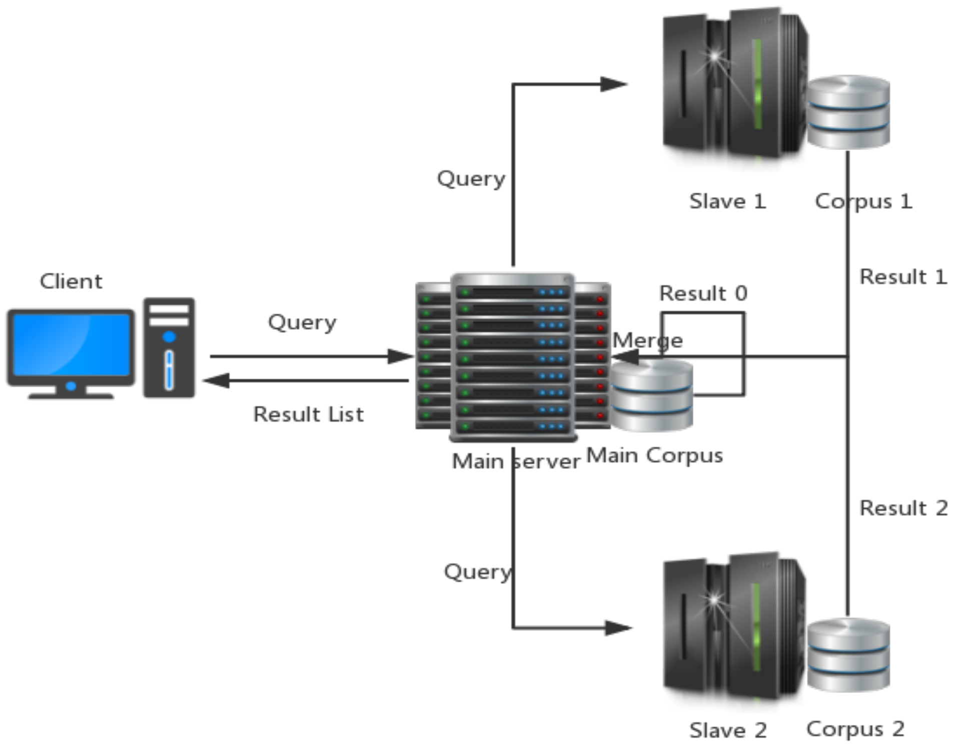
[27] NLTK Project, “Natural Language Toolkit,” 2018. [Online]. Available: https://www.nltk.org/. [Accessed Apr. 8, 2018].

[28] A. Ball, C. Doumouro, “Snips Natural Language Understanding,” 2018. [Online]. Available: https://snips-nlu.readthedocs.io/en/latest/. [Accessed Apr. 8, 2018].

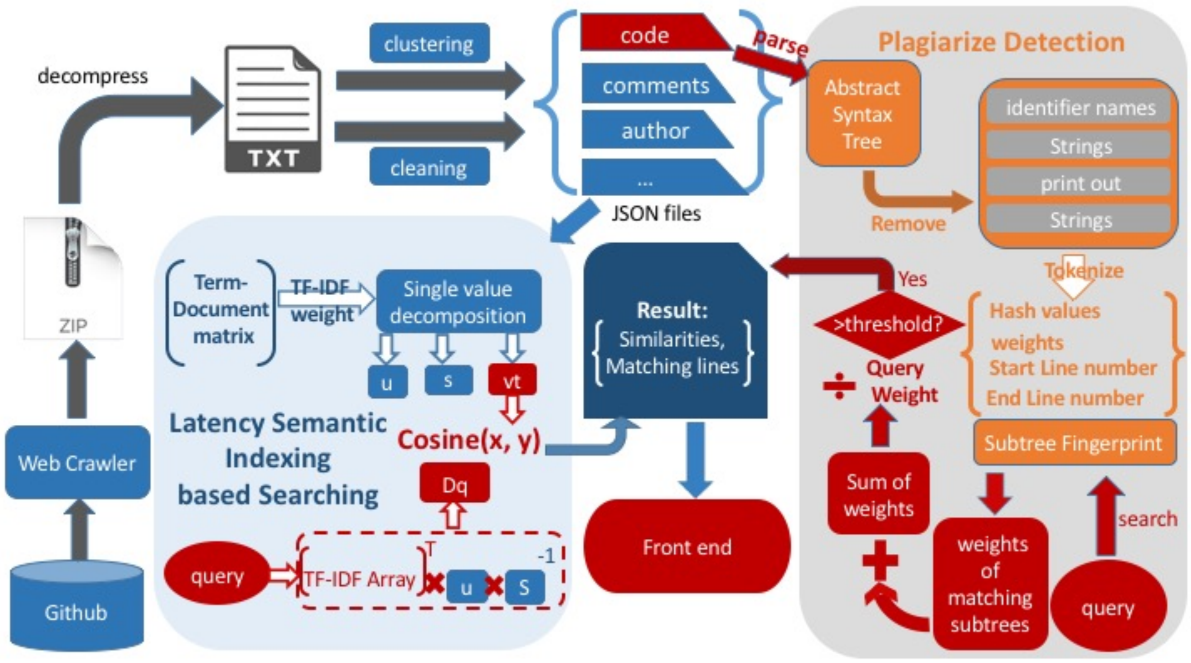
[29] J. Dean and S. Ghemawat, “MapReduce: simplified data processing on large clusters,” *Communications of the ACM*, vol. 51, no. 1, Jan., pp. 107-113, 2008.

[30] Eclipse Contributors and others, “Package org.eclipse.jdt.core.dom,” 2013. [Online]. Available: http://help.eclipse.org/kepler/index.jsp?topic=%2Forg.eclipse.jdt.doc.isv%2Freference%2Fapi%2Forg%2Feclipse%2Fjdt%2Fcore%2Fdom%2Fpackage-summary.html. [Accessed Apr. 7, 2018].

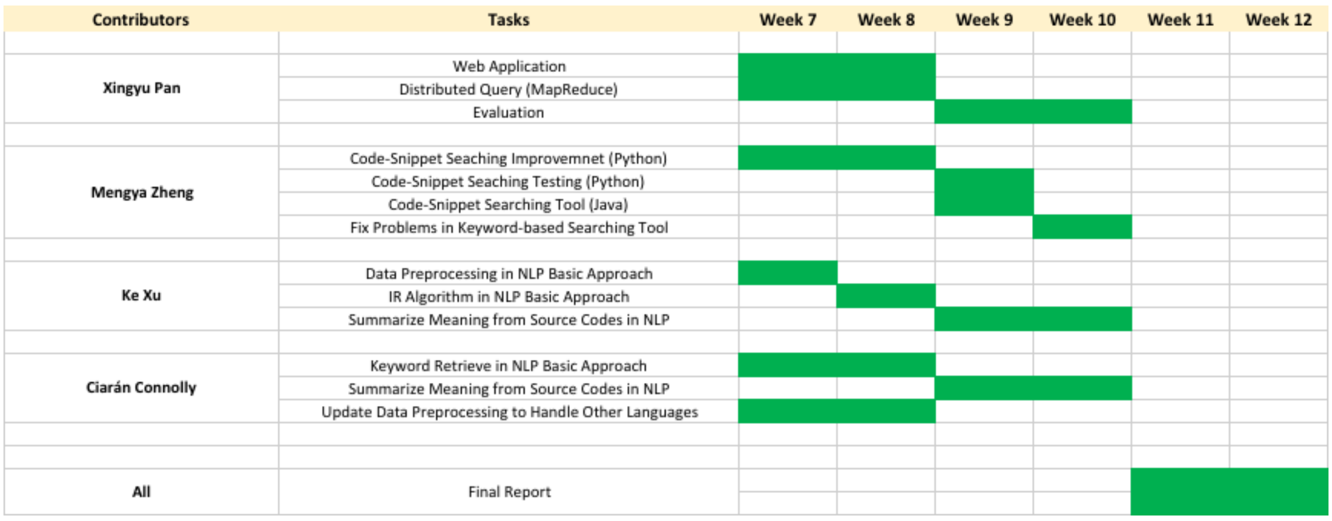
Appendix



Appendix A: Deployment Diagram



Appedix B: Architecture Diagram



Contributors	Tasks	Week 7	Week 8	Week 9	Week 10	Week 11	Week 12
Xingyu Pan	Web Application	🟩	🟩				
	Distributed Query (MapReduce)	🟩	🟩				
	Evaluation			🟩	🟩		
Mengya Zheng	Code-Snippet Seaching Improvemnet (Python)	🟩	🟩				
	Code-Snippet Seaching Testing (Python)			🟩			
	Code-Snippet Searching Tool (Java)			🟩			
	Fix Problems in Keyword-based Searching Tool				🟩		
Ke Xu	Data Preprocessing in NLP Basic Approach	🟩					
	IR Algorithm in NLP Basic Approach		🟩				
	Summarize Meaning from Source Codes in NLP			🟩	🟩		
Ciarán Connolly	Keyword Retrieve in NLP Basic Approach	🟩	🟩				
	Summarize Meaning from Source Codes in NLP			🟩	🟩		
	Update Data Preprocessing to Handle Other Languages	🟩	🟩				
All	Final Report					🟩	🟩

Appedix C: Gantt Chart