CODE SEARCH IN THE IDE WITH QUERY REFORMULATION

by

Shihui Gao

Submitted in partial fulfillment of the requirements
for the degree of Bachelor of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
December 2023

Table of Contents

List of Tables

List of Figures

Abstract

Software developers spend about 20% of their programming time searching for relevant code. They often spend a lot of time to manually choose queries for their code search. Unfortunately, due to vocabulary mismatch problems, the accurate answer may not be always retrieved, which leads to numerous trials and errors. Furthermore, many answers might not contain the relevant code examples that the developers look for. In this thesis, we extend RACK, an existing solution for code search, and attempt to solve the code search problem effectively. First, we replicate RACK in Python language from its original implementation in Java. Second, we construct a token-API database by analyzing thousands of Python posts from Stack Overflow. Third, we determine the relevance between a natural language query and API classes using three co-occurrence based heuristics – KKC, KAC and KPAC. Then we return a list of relevant API classes against a natural language query. Finally, we integrate our RACK implementation into a VS-code plugin. The plug-in accepts a natural language query and retrieves relevant code examples from GitHub by leveraging its search API and the API classes from RACK. Software developers can use these code examples to solve their programming problems much faster.

Acknowledgements

First of all, I would like to express my sincere gratitude to Dr. Masud Rahman. Your valuable suggestions and guidance played a key role in my project. I can't imagine completing this project without your vast expertise and thoughtful guidance. I am also very grateful to my reader Dr. Srini Sampalli. Your time, attention and participation are priceless. I appreciate your consideration of the information presented and your interest in the content. I look forward to future opportunities to repay your support with even greater results.

Thanks to my parents for their selfless love and firm support that allowed me to maintain courage and confidence in the face of difficulties. Their company makes me feel indestructible, and I feel so lucky for that.

I would like to thank all of the members of the Intelligent Automation in Software EngineeRing (RAISE) Lab – Sigma Jahan, Ohiduzzaman Shuvo, Parvez Mahbub, Asif Samir, Riasat Mahbub, Usmi Mukherjee, Lareina Yang, Callum MacNeil and Mehil B Shah. Their help, kindness and encouragement gave me the strength to overcome difficulties. Looking forward to seeing them achieve greater success in their future academic careers.

Furthermore, I would like to thank Dalhousie University and the Faculty of Computer Science. Thanks to them for creating a good learning environment and resources for me.

Finally, I would like to thank my former teachers, classmates and friends. Over the past four years, their careful guidance and selfless help have benefited me a lot and have become the most valuable asset in my college career.

Thank you again for these support and concern. May the future be filled with more success and happiness.

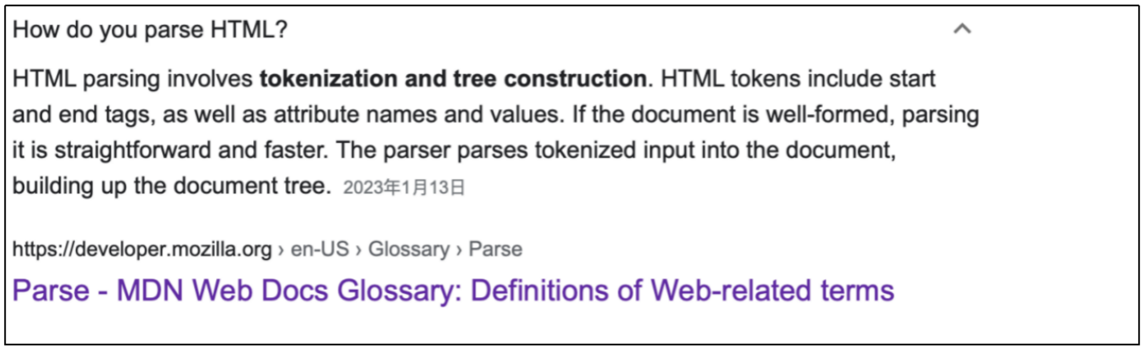
Chapter 1

Introduction

Software engineers often search on the web when they need relevant code examples to solve their programming problems. However, search results are not always guaranteed to have valid code examples. Software practitioners often use general-purpose search engines such as Google to find code examples. However, many of their results contain simple text-based answers (e.g., Figure 1.1 [1]), which might not be sufficient.

Developers often spend a lot of time manually designing search queries, which leads to numerous trials and errors with the search engines. Rahman et al [2] proposed a technique namely RACK that tackles the problem of code search. The original RACK has been used to search Java-based code examples against natural language queries. We replicated RACK in Python programming language to make it deliver Python code examples.

In this thesis, we undertake four major activities. First, we replicated RACK in Python programming language by carefully understanding its business logic and algorithms implemented in Java. Second, we constructed the token-API database by collecting the questions and Python code examples from the Stack Overflow Q&A site. Third, we determine the relevance between keywords in natural language and candidate API classes using three heuristics: Keyword-API Co-occurence (KAC), Keyword-Keyword Co-occurence (KKC) and Keyword Pair API Co-occurrence (KPAC). Then we use natural language queries to retrieve a list of related API classes. We collected 50 questions from four programming sites (freecodecamp.org, programiz.com, geeksforgeeks.org and realpython.com) to test the accuracy of the recommended API classes. (e.g., Figure 1.2) Finally, we designed a Visual Studio Code plug-in that encapsulates our solution. The plug-in accepts a natural language query, returns relevant code examples from GitHub, and then further refines them using the relevant API classes/methods from the RACK module. Software developers can use this RACK VS-code plug-in to reduce their hassle in code example searching.



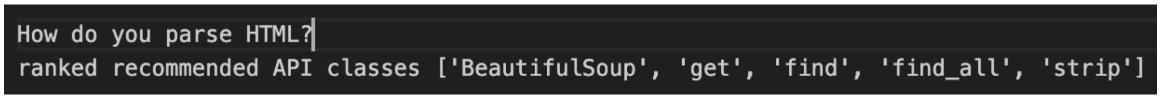
How do you parse HTML?

HTML parsing involves **tokenization and tree construction**. HTML tokens include start and end tags, as well as attribute names and values. If the document is well-formed, parsing it is straightforward and faster. The parser parses tokenized input into the document, building up the document tree. 2023年1月13日

https://developer.mozilla.org › en-US › Glossary › Parse

Parse - MDN Web Docs Glossary: Definitions of Web-related terms

Figure 1.1: Search result from Google: “How do you parse HTML?”



```
How do you parse HTML?
ranked recommended API classes ['BeautifulSoup', 'get', 'find', 'find_all', 'strip']
```

Figure 1.2: Question title and ranked recommended API classes

Chapter 2

Background

We introduce the required terminologies and concepts to follow the remaining of the thesis. We employ techniques such as Search Queries, HTML Parsing, GitHub Search API, AST Parsing, Jaccard Similarity, and the development of a VS-Code extension.

2.1 Search Query

A search query is words or phrases that we enter in the search box. There are three types of queries: Navigational search queries, Informational search queries and Transactional search queries. Navigational search queries are used to find specific places such as websites. For example, we use Google Scholar to find the academic articles we want. Information search queries allow us to find related resources based on the words or phrases we enter. For example, if we search for "Moon" in Google Chrome, we can get information about the moon such as the moon's Size, distance from the Earth and other information. Transactional search queries are mainly used for ads. This is usually where companies buy ads from the website. For example, if we search for facial cleansers, we will see websites selling facial cleansers to promote our transactions [3]. We mainly use navigation search queries in this thesis. We search query in VS-code IDE and return related top 5 code snippets.

2.2 HTML parsing

We need to use HTML parsing to extract the code content of the websites by Beautiful Soup. Beautiful Soup [1] is a popular Python library, which creates a parse tree for the parsed page that can be used to extract data from the HTML. So we can extract all the code parts from HTML files. We need to install Beautiful Soup first. Second, we need to import Beautiful Soup into our Python files. Third, we find HTML tags and

[1]url https://beautiful-soup-4.readthedocs.io/en/latest/

extract the full content from HTML tags. Then we use “find_all(’code’)” to extract all the code parts from HTML files. Finally, we need to iterate through all the code parts and use get_text() to like code can be printed.

2.3 GitHub Search API

We use the website ”https://api.github.com/search/” to search for APIs in GitHub. The GitHub search API supports different types of searches, such as code, repositories and issues. In the thesis, we need to find relevant code snippets, so we choose to set ”code search”. We can choose to search for answers in different programming languages. We chose Python in our search. We can search for URLs containing code. But this website contains other content besides code. So we need to convert github.com into a raw URL (raw.githubuserconten.com). For example: ”http://tinyurl.com/4bkme64u” needs to convert to ”http://tinyurl.com/3fj8axfr” . Then we use ast parsing to analyze and process these codes.

2.4 AST Parsing

AST (Abstract Syntax Trees) parsing can analyze and return the code structure when we enter Python code. For example, in the fourth step of methodology (”Design of a VS-Code Plug-in for Code Search”) We use ast parsing to analyze and extract all definition functions by the nodes of ast. AST parsing has three steps: lexical analysis, syntax analysis and code generation [4]. First, we divide the content into different parts, such as punctuation numbers and nouns. Inthe second step, we will generate the syntax tree. The syntax tree includes different parts such as body content, value, sourceType and so on. Third, when we know enough information about the code structure, we can find the specific location based on this information, and then return the code.

2.5 Jaccard Similarity

The Jaccard similarity is one of the similarity measurement techniques between two sets that are widely used in natural language processing, recommendation systems and so on [5]. It is the size of the intersection of two sets divided by the size of their

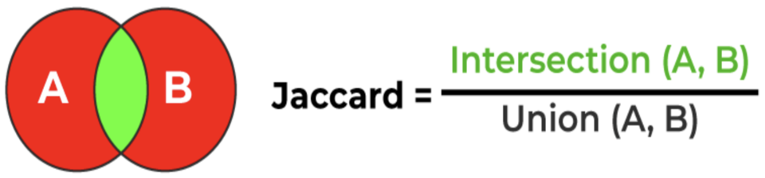


Figure 2.1: Jaccard similarity formula

union.(Figure 2.1) This article measures the similarity between recommended APIS and GitHub code snippets, which can help us to recommend relative code snippets with the 5 highest Jaccard similarity. The formula is the number of intersections between recommended API classes and defined function API classes divided by the number of unions between recommended API classes and defined function API classes.

2.6 VS-Code extension

We build a VS-Code extension IDE to let people use questions in natural language to search code snippets from it. We need to build a new VS-Code extension. First, we need to install Node.js and Git. Then we need to build a development-ready TypeScript project with a generator. After creating the extension with basic code, we can edit extension.ts to let it run our files in the src folder. Fourth, we press F5 in the extension.ts, the extension can be run in a new Extension Development Host window. Fifth, the extension command can be run from the Command Palette in the new window. Finally, we can see the result from the debug console. printed.

2.7 ChatGPT

People can use ChatGPT to find code samples for issues. ChatGPT will provide relevant codes based on the question. However, ChatGPT also has some problems. We could not find the original source of the code in ChatGPT. This may be affected by some software developers, especially students. Their work may be considered to be plagiarism without reference. Additionally, the results of ChatGPT can be integrated directly into the IDE. Therefore, using ChatGPT you need to switch context from your IDE to the browser before you can do this. This means that ChatGPT may not capture the full context of the IDE, so this is a limitation of ChatGPT.

2.8 Summary

In this chapter, we discussed several key concepts such as Search Query, HTML parsing, GitHub Search API, AST Parsing, Jaccard Similarity, VS-Code extension and ChatGPT. We need to understand what search query is. We use Beautiful Soup to perform HTML parsing to extract code from the stack overflow website to create a database. Because we want to find and return relevant code snippets in GitHub, we need to understand the GitHub search API. AST parsing is used to extract code snippets from the code part. Jaccard Similarity is used to calculate the correlation between code snippets and recommended API classes so that we can find more relevant and effective code snippets. We need to use VS Code to build a plug-in IDE so that we can return relevant codes based on our natural language search query in this IDE. ChatGPT can also search code based on natural language search queries, so we need to know what ChatGPT cannot do that our technology can.

Chapter 3

Methodology

We go through four steps to find relevant code snippets based on natural language search queries. First, we replicate RACK in Python language from its original implementation in Java language. Second, we construct a token-API database using Python posts from Stack Overflow. Third, we determine the relevance between natural language and API classes and return a list of related API classes. Finally, we design an IDE Plug-in for Code Search to get relevant code snippets.

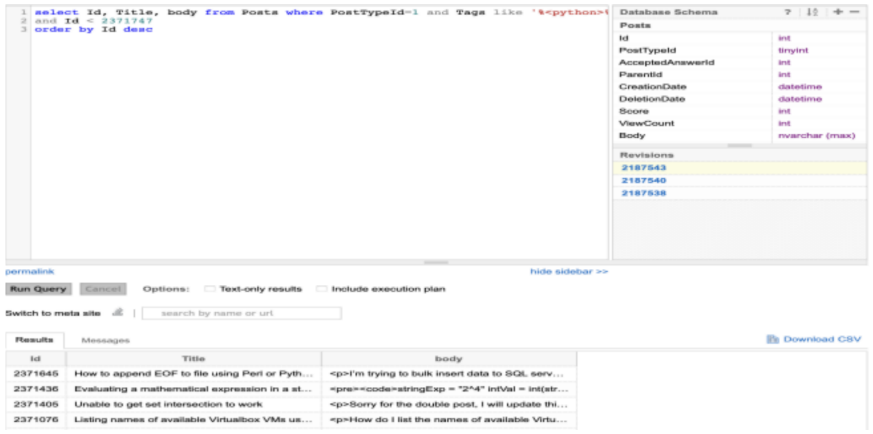
3.1 Replication of RACK in Python

First, we analyzed the original RACK to understand all required components including data, operating logic and their usage. Then we replicated RACK in Python language from its original implementation in Java language. Python and Java have different syntax and styles, which led to a significant learning curve. We thus migrate a lot of functions from Java to Python using W3School[1] [6].

3.2 Construction of token-API database for Python posts from Stack Overflow

In this step, we build a token-API database from Stack Overflow. We extracted the main items including post ID, question title and answer from the Python posts [7]. To collect the data from Stack Overflow, we execute relevant SQL queries at Stack Exchange Data Explorer (e.g., Figure 3.1). Since the question was for Python, we selected the question with "python" tags. Since the answer needs to include code segments, we select the answers containing the "code" tag(s) in their body. Since there were millions of posts, we downloaded them in batches where each batch contained up to fifty thousand posts. We downloaded all the post-related data and saved them

[1]url https://www.w3schools.com/



```
select Id, Title, body from Posts where PostTypeId=1 and Tags like '%<python>%
and Id < 2371747
order by Id desc
```

Id	Type
Id	int
PostTypeId	tinyint
AcceptedAnswerId	int
ParentId	int
CreationDate	datetime
DeletionDate	datetime
Score	int
ViewCount	int
Body	nvarchar (max)

2187543
2187540
2187538

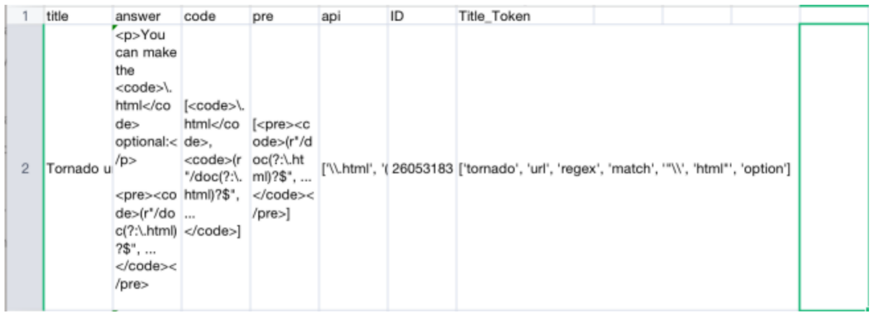
Id	Title	body
2371645	How to append EOF to file using Perl or Pyth...	<p>I'm trying to bulk insert data to SQL serv...
2371436	Evaluating a mathematical expression in a st...	<pre><code>stringExp = "2^4" intVal = int(str...
2371405	Unable to get set intersection to work	<p>Sorry for the double post, I will update thi...
2371076	Listing names of available Virtualbox VMs us...	<p>How do I list the names of available Virtu...

Figure 3.1: Stack overflow Python posts include ID, Title and the body content of posts

into one CSV file. To ensure data correctness, we first use the data of one hundred posts.

We used standard natural language pre-processing to extract the keywords and code elements from each question and answer respectively. For example, let us consider the question – "How do you use the argparse module in Python to parse command-line arguments?". First, we identify individual words by splitting the question against white space and punctuation marks. Second, we determine the parts of speech of each term and select verbs and nouns. Third, we remove stop words to get a list of keywords – ["argparse", "module", "Python", "parse", "command-line", "arguments"]. Finally, we use stemming to get the root form of each keyword as follows – ['argpars', 'modul', 'python', 'pars', 'command', 'argument']. Once keywords are extracted from a question, we use Beautiful Soup [8] to extract the code-like elements from each answer. (e.g., Figure 3.2)

In particular, we extracted the API methods and class names from the code part of each answer. Then we establish the connection between keywords and API items based on their co-occurrence on Stack Overflow posts and store these connections in a database. Figure 3.3 [9] shows different steps of our token-API database construction, as done in the earlier work [2].



1	title	answer	code	pre	api	ID	Title_Token
2	Tornado u	<p>You can make the <code>\\.html</code> optional:</p> <pre><code>(r"/doc(?:\\.html)?$", ... </code></pre>	[<code>\\.html</code>, <code>(r"/doc(?:\\.html)?$", ... </code>]	[<pre><code>(r"/doc(?:\\.html)?$", ... </code></pre>]	['\\\\.html', '(	26053183	['tornado', 'url', 'regex', 'match', '"\\\\', 'html"', 'option']

Figure 3.2: One example about one Python stack overflow post

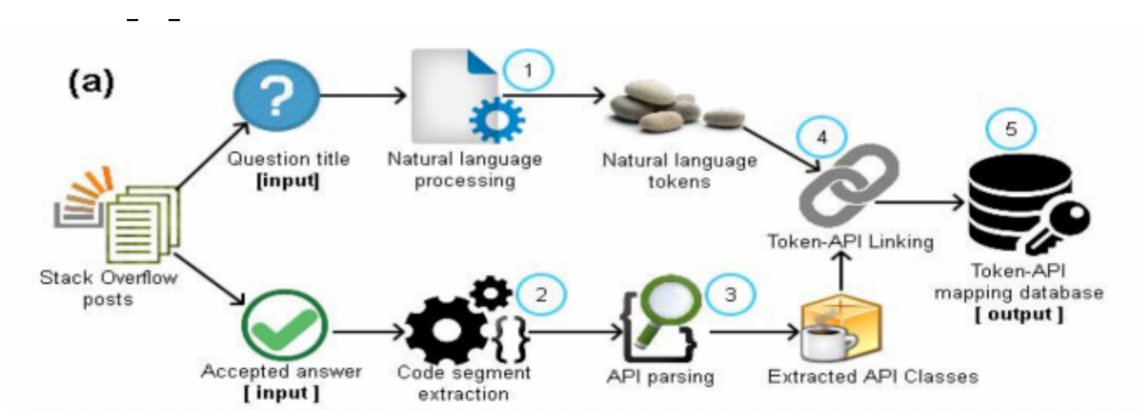


Figure 3.3: Construction of token-API database using Python posts from Stack Overflow

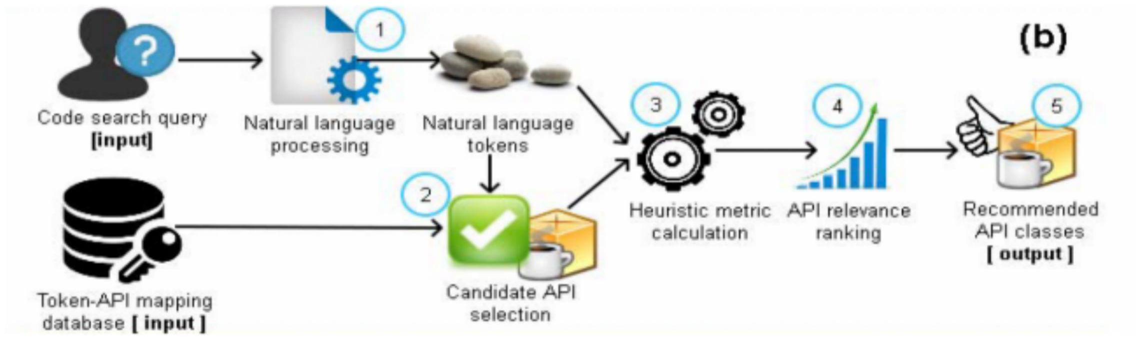

(b)
1
2
3
4
5
Code search query [input]
Natural language processing
Natural language tokens
Token-API mapping database [input]
Candidate API selection
Heuristic metric calculation
API relevance ranking
Recommended API classes [output]

Figure 3.4: Determining the relevance between a natural language query and API classes and return a list of related API classes

3.3 Determine the relevance between natural language and API classes and return a list of related API classes

In the third step, we have some code search queries and use natural language processing to get natural language tokens. We have the token API mapping database that can be used. The list is a natural language token to find the candidate API class sections from the token API as my mapping database. Then we use the heuristic metrics calculation to rank the API classes. (Figure 3.4 [9])

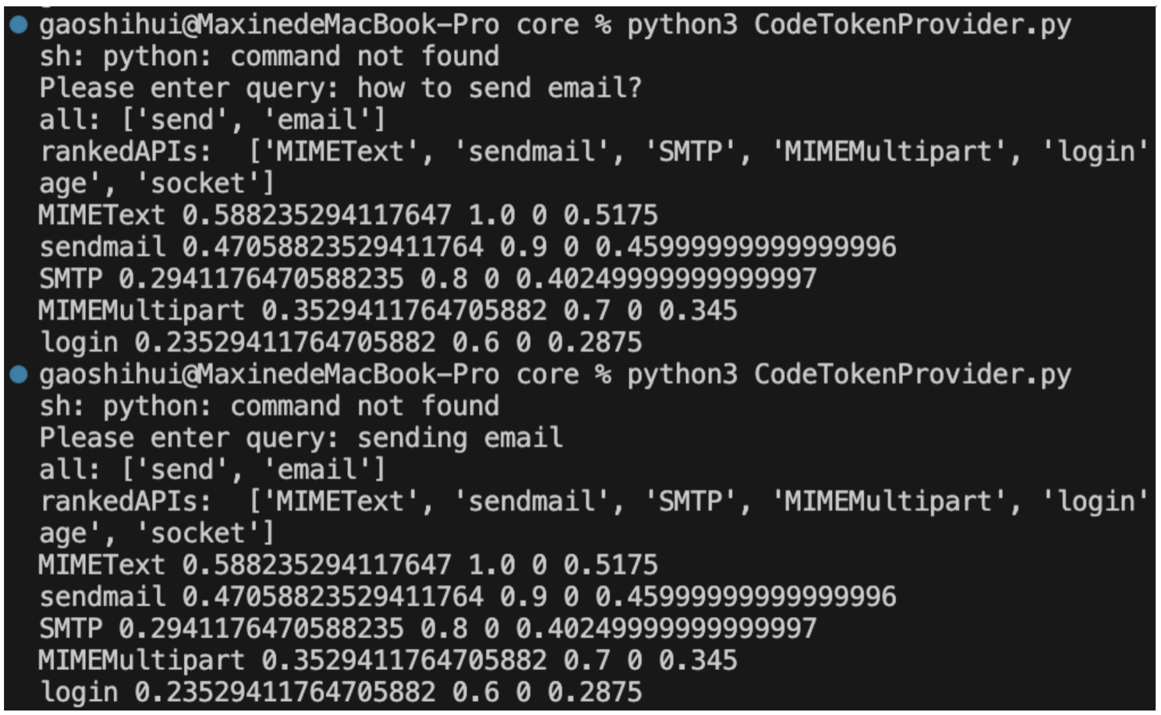
For RACK in Python, we use three heuristics –Keyword-API Co-occurence(KAC), Keyword-Keyword Co-occurence(KKC) and Keyword Pair API Co-occurrence(KPAC) to find the relevant API classes and methods against a natural language query [2]. KAC refers to the fact that some keywords may appear frequently with a specific API or may be associated with multiple API classes in various programming solutions. There is a potential semantic connection between the keywords and the API. KKC identifies that candidate API should not only be related to multiple keywords, but also should be mutually consistent or compatible with each other. A set of compatible API classes/methods are more likely to implement a programming solution. KPAC computes the co-occurrence score of an API against a pair of keywords. The API is likely to be relevant for queries containing these two keywords [10]. KAC, KKC and pair against an API [10]. In RACK, KAC, KKC and KPAC are combined and used to recommend relevant API classes against a given query [9]. (Figure 3.5 [9])

Algorithm 1 API Relevance Ranking Algorithm

```
procedure RACK($Q$)                                ▷ $Q$: code search query
    $R \leftarrow \{\}$                            ▷ list of relevant APIs
    ▷ collecting keywords from the search query
    $K \leftarrow$ collectKeywords($Q$)
    ▷ collecting candidate APIs
    $L \leftarrow$ getKACList($K$)
    $L_{coh} \leftarrow$ getKKCList($K$)
    ▷ estimating relevance of the candidate APIs
    for Keyword $K_i \in K$ do
        $sortedAPIs \leftarrow$ sortByFreq($L[K_i]$)
        for APIClass $A_j \in sortedAPIs$ do
            ▷ likelihood score of an API
            $S_{KAC} \leftarrow$ getKACScore($A_j, sortedAPIs$)
            $R[A_j].score \leftarrow R[A_j].score + S_{KAC}$
        end for
    end for
    for Keyword $K_i, K_j \in K$ do
        $C_i \leftarrow$ getContextList($K_i$)
        $C_j \leftarrow$ getContextList($K_j$)
        ▷ relevance of an API with multiple keywords
        $S_{KKC} \leftarrow$ getKKCScore($C_i, C_j$)
        for APIClass $A_j \in L_{coh}[K_i, K_j]$ do
            $R[A_j].score \leftarrow R[A_j].score + S_{KKC}$
        end for
    end for
    ▷ ranking of the APIs
    $rankedAPIs \leftarrow$ sortByScore($R$)
    return $rankedAPIs$
end procedure
```

Figure 3.5: API Relevance Ranking Algorithm

We accumulate these metrics for each candidate API, rank the API classes based on their relevance, and recommend the top five API classes. For example, for the query – “generate MD5 hash from a string” RACK recommends these API classes/methods – hexdigest, md5, encode, update, hashlib. These API classes will be used to find relevant code from GitHub later. (e.g., Figure 3.6)



```
● gaoshihui@MaxinedeMacBook-Pro core % python3 CodeTokenProvider.py
  sh: python: command not found
  Please enter query: how to send email?
  all: ['send', 'email']
  rankedAPIs:  ['MIMEText', 'sendmail', 'SMTP', 'MIMEMultipart', 'login'
  age', 'socket']
  MIMEText 0.588235294117647 1.0 0 0.5175
  sendmail 0.47058823529411764 0.9 0 0.45999999999999996
  SMTP 0.2941176470588235 0.8 0 0.40249999999999997
  MIMEMultipart 0.3529411764705882 0.7 0 0.345
  login 0.23529411764705882 0.6 0 0.2875
● gaoshihui@MaxinedeMacBook-Pro core % python3 CodeTokenProvider.py
  sh: python: command not found
  Please enter query: sending email
  all: ['send', 'email']
  rankedAPIs:  ['MIMEText', 'sendmail', 'SMTP', 'MIMEMultipart', 'login'
  age', 'socket']
  MIMEText 0.588235294117647 1.0 0 0.5175
  sendmail 0.47058823529411764 0.9 0 0.45999999999999996
  SMTP 0.2941176470588235 0.8 0 0.40249999999999997
  MIMEMultipart 0.3529411764705882 0.7 0 0.345
  login 0.23529411764705882 0.6 0 0.2875
```

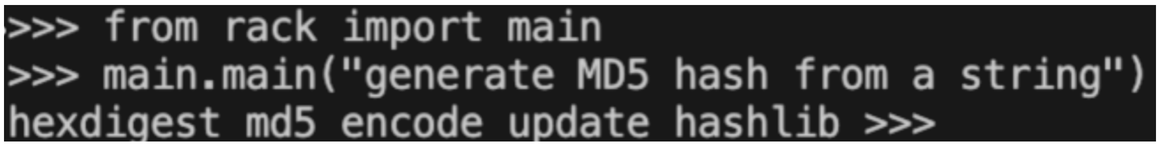
Figure 3.6: Search for complete sentences and phrases with the same meaning

3.4 Design of a VS-Code Plug-in for Code Search

Once RACK in Python is functional, we build a VS-code plug-in for our solution. (Figure 3.10 [9]) We package all related files of RACK (e.g., .py files) into a wheel file, and make a Python library. Then we can use pip/pip3 to install RACK successfully on the local machine. (It can be installed on other machines as well). In this way, we can call it anywhere from the terminal. (e.g., Figure 3.7) Then we build a new typescript-based extension for the VS Code IDE. This extension not only can load our RACK solution but also can respond to our query. For example, when we enter our query in natural language, the plug-in is able to return the relevant API classes in the Debug Console. (e.g., Figure 3.8)

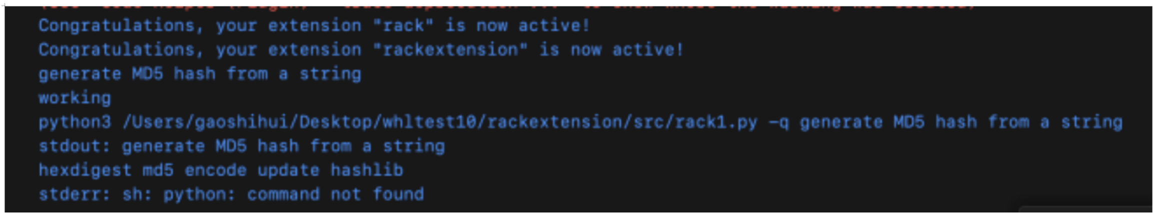
When we get the recommended API classes, we leverage them to search for relevant code snippets from GitHub. Toward that goal, we can develop a REST API client for GitHub and execute our NL query against the GitHub search [x].

From the GitHub search API, we extract the results and download the source code files programmatically. We also parse each source code file using AST parsing and extract the method bodies. Our idea was to prioritize the method bodies containing



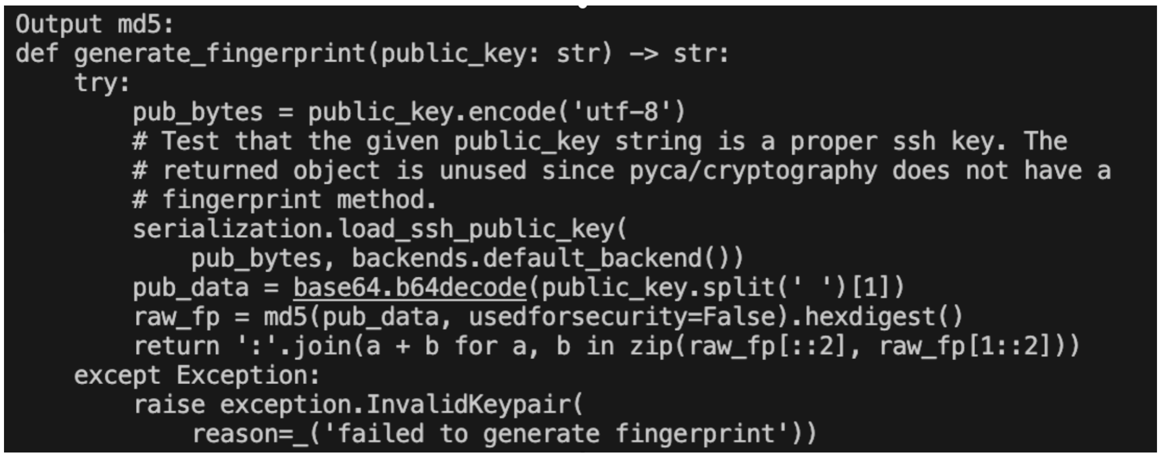
```
>>> from rack import main
>>> main.main("generate MD5 hash from a string")
hexdigest md5 encode update hashlib >>>
```

Figure 3.7: Using RACK in terminal



```
Congratulations, your extension "rack" is now active!
Congratulations, your extension "rackextension" is now active!
generate MD5 hash from a string
working
python3 /Users/gaoshihui/Desktop/whltest10/rackextension/src/rack1.py -q generate MD5 hash from a string
stdout: generate MD5 hash from a string
hexdigest md5 encode update hashlib
stderr: sh: python: command not found
```

Figure 3.8: Running rackextension



```
Output md5:
def generate_fingerprint(public_key: str) -> str:
    try:
        pub_bytes = public_key.encode('utf-8')
        # Test that the given public_key string is a proper ssh key. The
        # returned object is unused since pyca/cryptography does not have a
        # fingerprint method.
        serialization.load_ssh_public_key(
            pub_bytes, backends.default_backend())
        pub_data = base64.b64decode(public_key.split(' ')[1])
        raw_fp = md5(pub_data, usedforsecurity=False).hexdigest()
        return ':'.join(a + b for a, b in zip(raw_fp[::2], raw_fp[1::2]))
    except Exception:
        raise exception.InvalidKeypair(
            reason=_('failed to generate fingerprint'))
```

Figure 3.9: The relevant code snippet of API classes (md5)

the relevant API classes, recommended by RACK. We thus calculate the relative percentage. We use the Set operation to determine the overlap between the ranked API classes and the API classes of the candidate code. Then based on their Jaccard similarity (intersection(recommended API classes, definition function API classes) / union(recommended API classes, definition function API classes)), we rank the top-5 code examples against each query. For example, Figure 3.9 shows the top relevant code example for our query – "generate MD5 hash from a string".

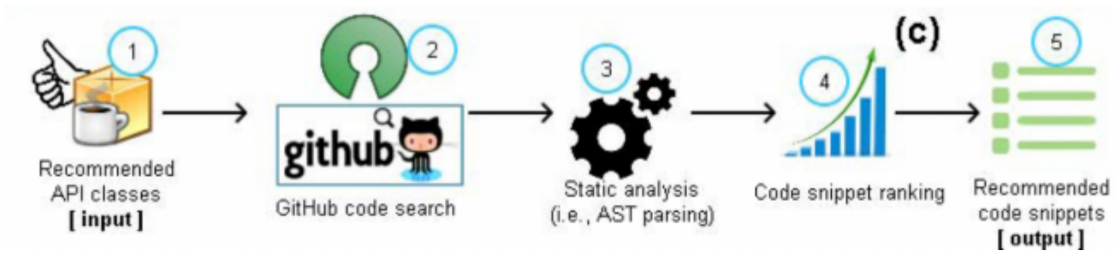

1
Recommended API classes [input]
2
github
GitHub code search
3
Static analysis (i.e., AST parsing)
4
Code snippet ranking
(c)
5
Recommended code snippets [output]

Figure 3.10: Code snippet search in GitHub

3.5 Summary

We analyze and understand all required components from the original implementation in Java programming language. We replicate RACK in Python language by migrating the syntax and styles between Java and Python. Then we construct a token-API database by combining the same posts' natural language token that the title of Stack Overflow deals with natural language processing, and the API classes that the function names and method name from the code of Stack Overflow. Third, we use all heuristics and three heuristics (Keyword-API Co-occurrence(KAC), Keyword-Keyword Co-occurrence(KKC), and Keyword Pair API Co-occurrence(KPAC)) to rank and return recommended API classes based on natural language search queries. Finally, Rack is encapsulated in a VS-Code plug-in to get recommended API classes and related code snippets that are from GitHub.

Chapter 4

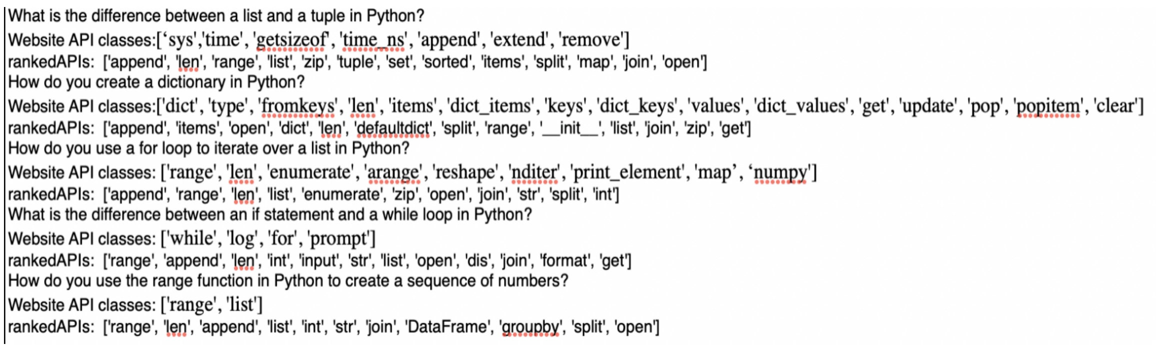
Empirical findings & discussions

In this chapter, we collected 50 Python questions from four open source sites to evulate the accuracy of the recommendation API classes technology of RACK by three performance metrics (Precision (P) , Recall (R) and F1-score (F)). We also answer two research questions.

4.1 Evaluation & Performance Metrics

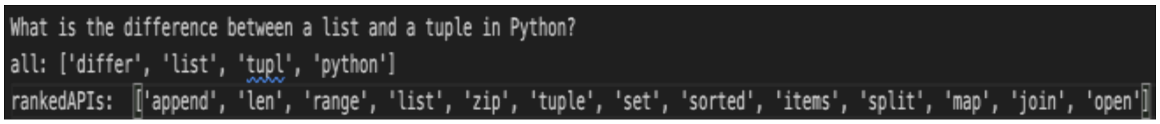
In the experiments, We collected 50 Python questions from four programming sites (freecodecamp.org, programiz.com, geeksforgeeks.org and realpython.com).

We calculate Mean P(precision), Mean R(recall) and Mean F percent of Keyword-API Co-occurence (KAC), Keyword-Keyword Co-occurence (KKC), Keyword Pair API Co-occurrence (KPAC) and all (KAC, KKC and KPAC) to validate the 50 questions. KAC uses contextual information in Stack Overflow question titles to enhance code search queries by considering co-occurring words and identifying relevant API classes based on multiple query keywords. KKC uses the complexity of code search



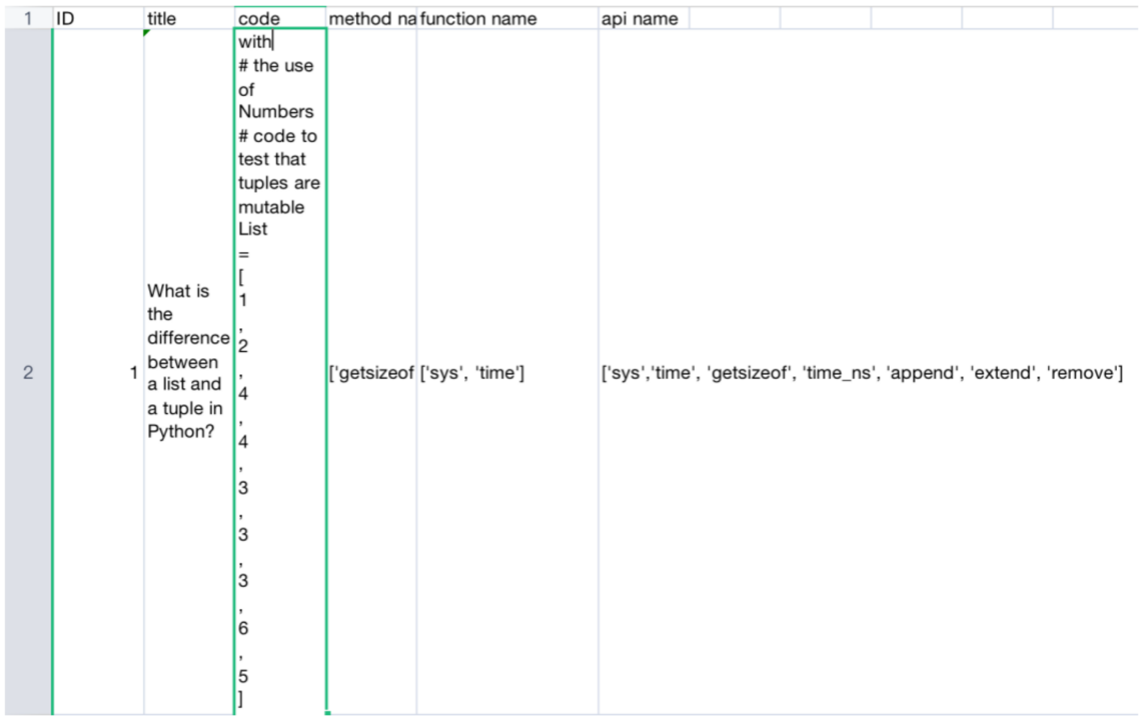
```
What is the difference between a list and a tuple in Python?
Website API classes:['sys','time', 'getsizeof', 'time_ns', 'append', 'extend', 'remove']
rankedAPIs:  ['append', 'len', 'range', 'list', 'zip', 'tuple', 'set', 'sorted', 'items', 'split', 'map', 'join', 'open']
How do you create a dictionary in Python?
Website API classes:['dict', 'type', 'fromkeys', 'len', 'items', 'dict_items', 'keys', 'dict_keys', 'values', 'dict_values', 'get', 'update', 'pop', 'popitem', 'clear']
rankedAPIs:  ['append', 'items', 'open', 'dict', 'len', 'defaultdict', 'split', 'range', '__init__', 'list', 'join', 'zip', 'get']
How do you use a for loop to iterate over a list in Python?
Website API classes: ['range', 'len', 'enumerate', 'arange', 'reshape', 'nditer', 'print_element', 'map', 'numpy']
rankedAPIs:  ['append', 'range', 'len', 'list', 'enumerate', 'zip', 'open', 'join', 'str', 'split', 'int']
What is the difference between an if statement and a while loop in Python?
Website API classes: ['while', 'log', 'for', 'prompt']
rankedAPIs:  ['range', 'append', 'len', 'int', 'input', 'str', 'list', 'open', 'dis', 'join', 'format', 'get']
How do you use the range function in Python to create a sequence of numbers?
Website API classes: ['range', 'list']
rankedAPIs:  ['range', 'len', 'append', 'list', 'int', 'str', 'join', 'DataFrame', 'groupby', 'split', 'open']
```

Figure 4.1: Recommended API classes and Ground-truth API classes and methods for the questions



```
What is the difference between a list and a tuple in Python?
all: ['differ', 'list', 'tupl', 'python']
rankedAPIs: ['append', 'len', 'range', 'list', 'zip', 'tuple', 'set', 'sorted', 'items', 'split', 'map', 'join', 'open']
```

Figure 4.2: The title, keywords and recommended API classes of the question



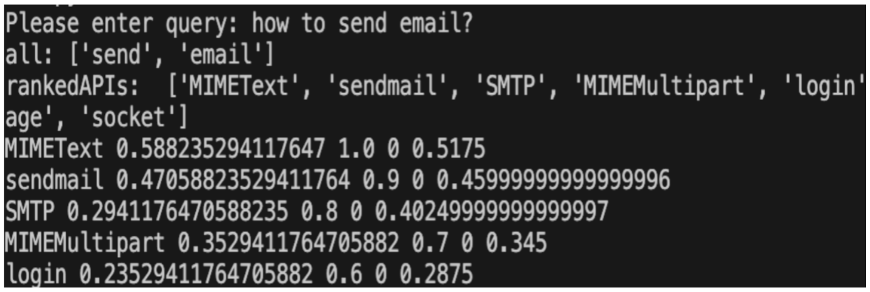
1	ID	title	code	method na	function name	api name
2	1	What is the difference between a list and a tuple in Python?	with # the use of Numbers # code to test that tuples are mutable List = [1 , 2 , 4 , 4 , 3 , 3 , 3 , 6 , 5]	['getsizeof	['sys', 'time']	['sys','time', 'getsizeof', 'time_ns', 'append', 'extend', 'remove']

Figure 4.3: The title, code, method names, function names and API class names (method names + function names) of the question

queries using multiple keywords by ensuring that candidate API lists exhibit consistent semantics, leveraging semantic similarity methods, and using consistent keyword pairs extracted from the Stack Overflow context. KPAC identifies relevant API classes by analyzing their co-occurrence with keyword pairs in searches, providing an approach to finding API classes that are closely related to specific keywords in a query, especially when multiple keywords are involved. (e.g., Figure 4.1, 4.2 and 4.3)

Precision (P): It is a measure of the accuracy of the positive predictions made by a model. There, it refers to the percentage of the retrieved API classes that are relevant. Formula: (GT n Ra) / Ra

Recall (R): It is a measure of the ability of a model to capture all the relevant instances of a particular class. There, it refers to the percentage of relevant API



```
Please enter query: how to send email?
all: ['send', 'email']
rankedAPIs:  ['MIMEText', 'sendmail', 'SMTP', 'MIMEMultipart', 'login'
age', 'socket']
MIMEText 0.588235294117647 1.0 0 0.5175
sendmail 0.47058823529411764 0.9 0 0.45999999999999996
SMTP 0.2941176470588235 0.8 0 0.40249999999999997
MIMEMultipart 0.3529411764705882 0.7 0 0.345
login 0.23529411764705882 0.6 0 0.2875
```

Figure 4.4: The recommended APIs by KAC, KPAC, KKC and all. Query: How to send email?

classes that are retrieved. Formula: (GT n Ra) / GT

F1-score (F): It provides a balanced measure. It is the harmonic mean of precision and recall. Formula: (2 * P * R) / (P + R)

GT: It is a shorthand for the term “ground truth”, all method names and function names of the website definition function

Ra: ranked recommended API classes

Harmonic mean: a type of average calculated by dividing the number of observations by the reciprocal of each number in the series and then taking the reciprocal of that result.

In addition, we answer two research questions:

In RQ1, What are the Precision, Recall and F1-Score of RACK in Python for all heuristics and individual heuristics?

In RQ2, Java VS Python – for which language does RACK perform better?

4.2 RQ1: What are the Precision, Recall and F1-Score of RACK in Python for all heuristics and individual heuristics?

To answer R1, RACK uses three heuristics Keyword-API Co-occurence (KAC), Keyword-Keyword Co-occurence (KKC) and Keyword Pair API Co-occurrence (KPAC) to calculate the accuracy of API recommendations.

To prove the effectiveness of Rack recommended API technology, RACK ranks API classes by using Keyword-API Co-occurence (KAC), Keyword-Keyword Co-occurence (KKC) and Keyword Pair API Co-occurrence (KPAC) to calculate the correlation of each API token with the title tokens in the title. (e.g., Figure 4.4)

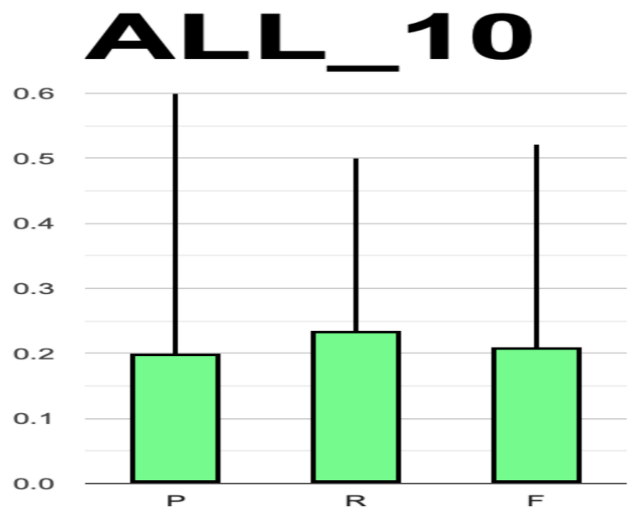


Figure 4.5: The P, R and F of all heuristics (KAC, KKC and KPAC) with the top 10 ranked API classes

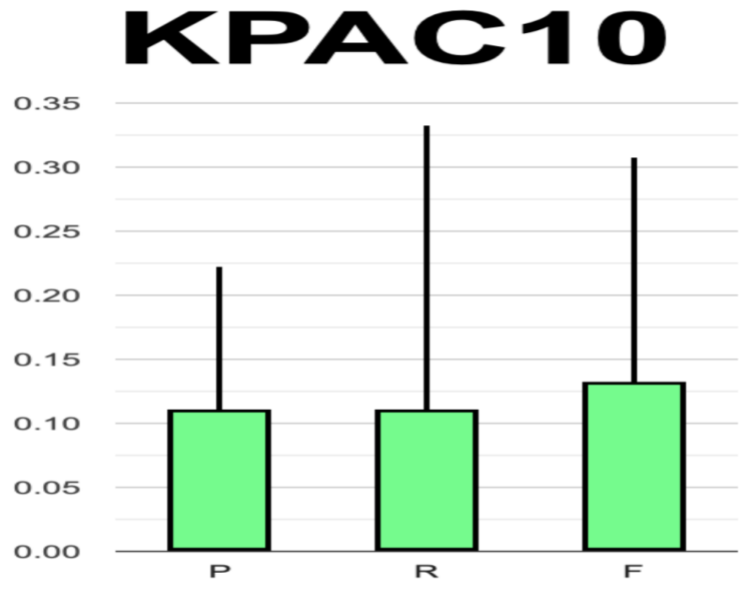


Figure 4.6: The P, R and F of KPAC heuristics with the top 10 ranked API classes

For the evaluation and validation of RACK, we should use a large number of examples. RACK recommends 10 relevant APIs for each query based on KKC, KPAC and KAC. We use 50 questions randomly collected from open source sites to validate

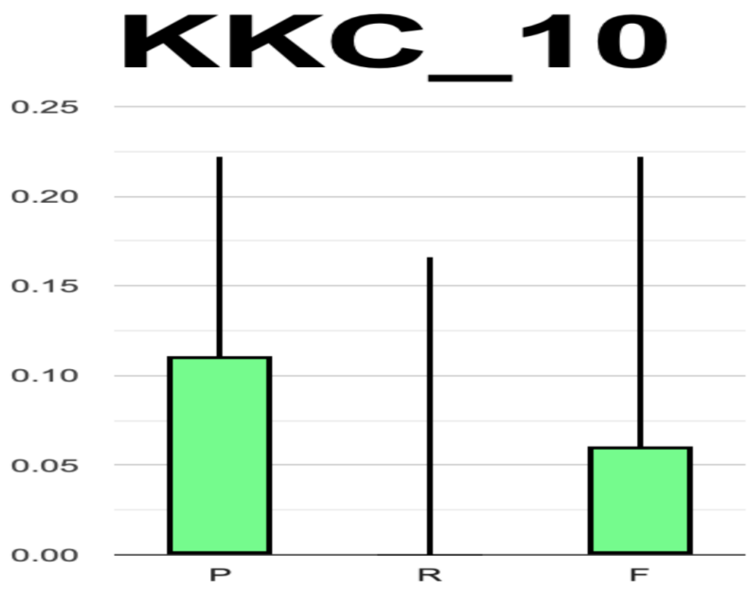


Figure 4.7: The P, R and F of KKC heuristics with the top 10 ranked API classes

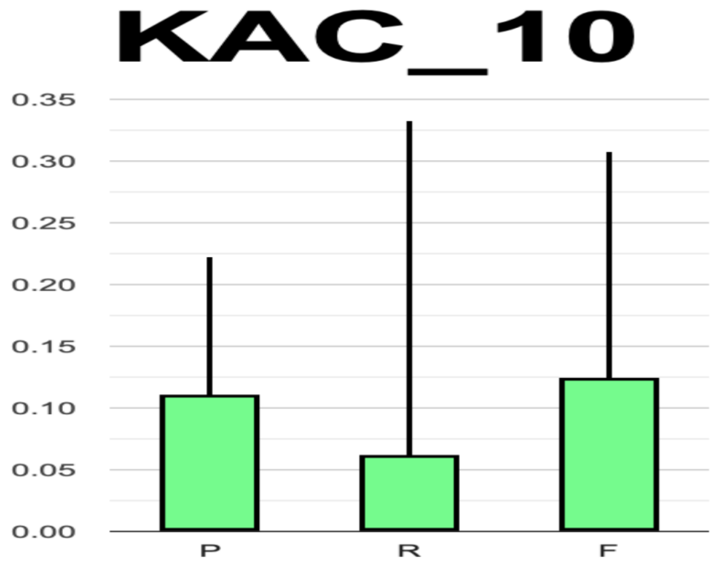


Figure 4.8: The P, R and F of KAC heuristics with the top 10 ranked API classes

API ranking and the accuracy of the relevance of API [2].

We calculate Mean P(precision), Mean R(recall) and Mean F (f1-score) percent of KAC, KKC, KPAC and all (KAC, KKC and KPAC).

Figure 4.5- Figure 4.8 are about the Mean P(precision), Mean R(recall) and Mean

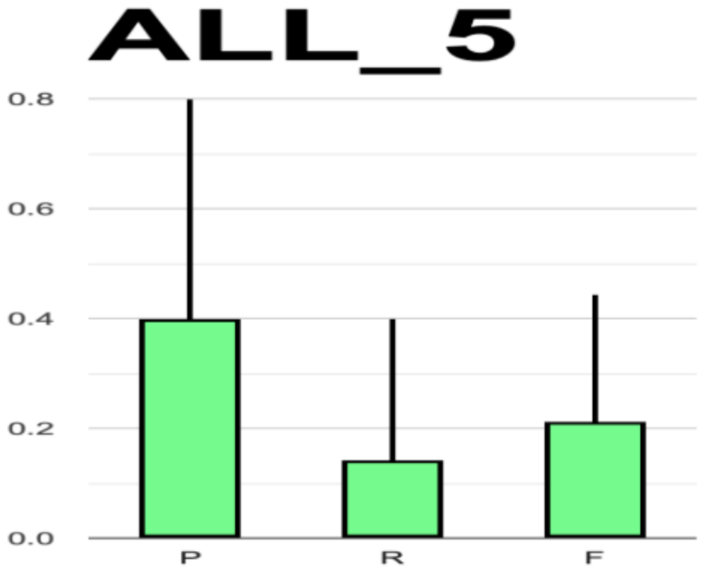


Figure 4.9: The P, R and F of all heuristics (KAC, KKC and KPAC) with the top 5 ranked API classes

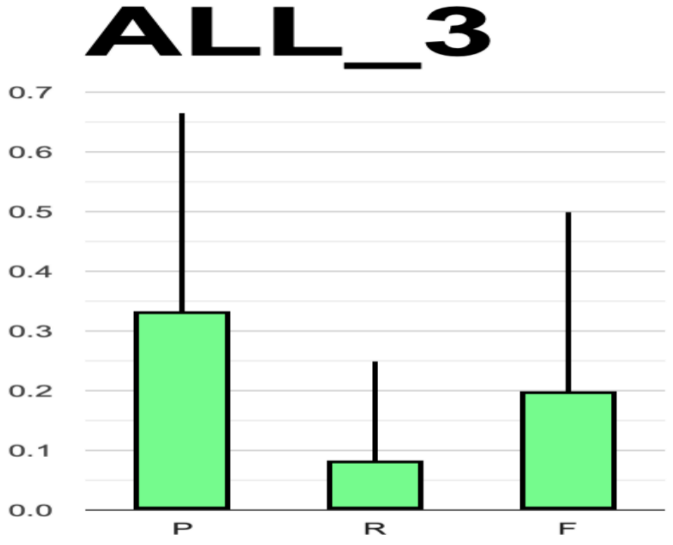


Figure 4.10: The P, R and F of all heuristics (KAC, KKC and KPAC) with the top 3 ranked API classes

F (F1-score) of ranked top 10 API classes for Keyword-API Co-occurence (KAC), Keyword-Keyword Co-occurence (KKC), Keyword Pair API Co-occurrence (KPAC) and all (KAC, KKC and KPAC). Comparing Figure 4.5- Figure 4.8 we can know when KKC, KAC and KPAC combine that they can provide the maximum performance of Mean P(precision), Mean R(recall) and Mean F, and the ranking combination of APIs is more reasonable. The Mean P(precision) of all of KKC, KPAC and KAC are about 11%. All heuristics have 20% in the Mean P(precision), which is the highest one. The highest Mean R(recall) is about 24% in all heuristics. KPAC has the second highest Mean R(recall), which is about 11%. All have about 20% Mean F (f1-score), which is the highest one. both of KAC and KPAC has the second highest Mean F (f1-score), which is about 13%. Hence, all heuristics is the most effective, and KPAC is the most effective of the three individual heuristics.

Figure 4.5, Figure 4.9 and Figure 4.10 are about the Mean P(precision), Mean R(recall) and Mean F (f1-score) percent of ranked top 3, 5 and 10 API classes for all (combines KAC KKC and KPAC). Comparing Figure 4.5, Figure 4.9 and Figure 4.10, we can know top 5 ranked API classes have the highest Mean P(precision) percentage, which is about 40%. The top 10 ranked API classes have the highest Mean R(recall) percentage, which is about 24%. Both of the ranked top 5 and 10 API classes have the similar Mean F (f1-score) percentage, which is about 20%.

4.3 RQ2: Java VS Python – for which language does RACK perform better?

To answer R2, we compared Top-3, 5 and 10 Accuracy, Mean Precision@K, and Mean Recall@K of KKC, KAC and ALL in Python (Table 4.1and Table 4.3) and Java (Table 4.2 [2] and Table 4.4 [2]).

From Table 4.1 and Table 4.2 [2], we can know Mean Average Precision@K for the top 3 and 5 all recommended API classes in JAVA(30.39% and 33.6%) are lower than those in Python (about 33.3% and 40%). However, the top 10 recommended API classes Mean Average Precision(34.92%) for all heuristics is higher than those in Python (20%). For Mean Recall @K of all heuristics for the top 3,5 and10 recommended API classes in Java (23.71%, 33.48%, 45.02%) is higher than those in Python (8.3%, 14.3%, 23.6%). Therefore, RACK in Java for all heuristics performs better than Python in Mean Recall @K.

From Table 4.3 and Table 4.4 [2], we can know Top-10 Mean Average Precision@K

Performance Metric	**Top-3**	**Top-5**	**Top-10**
Mean precision	33.3%	40%	20%
Mean Recall	8.3%	14.3%	23.6%

Table 4.1: Top - 3, 5 and 10 Mean Precision and Mean Recall of all heuristics in Python

Performance Metric	**Top-3**	**Top-5**	**Top-10**
Mean Average precision	30.39%	33.36%	34.92%
Mean Recall	23.71%	33.48%	45.02%

Table 4.2: Top - 3, 5 and 10 Mean Average Precision@K, and Mean Recall@K of all in JAVA

Heuristics	**Metric**	**Top-10**
Keyword-API Co-occurence (KAC)	Mean precision	11.1%
	Mean Recall	6.3%
Keyword-Keyword Co-occurence (KKC)	Mean precision	11.1%
	Mean Recall	0%

Table 4.3: Mean Precision, and Mean Recall of KKC, KAC and (KKC+KAC) in Python

Heuristics	**Metric**	**Top-10**
Keyword-API Co-occurence (KAC)	Mean Average precision	35.41%
	Mean Recall	44.8%
Keyword-Keyword Co-occurence (KKC)	Mean Average precision	24.11%
	Mean Recall	19.52%

Table 4.4: Top - 3, 5 and 10 Mean Average Precision@K, and Mean Recall@K of KKC, KAC and (KKC+KAC) in JAVA

of KKC and KAC heuristics in JAVA (35.41% and 24.11%) is higher than those in Python (11.1% and 11.1%). Top-10 Recall@K of KKC and KAC heuristics in JAVA (44.80% and 19.52%) is higher than them in Python (0% and6.3%). Therefore, RACK in Java performs better than Python in both KKC and KAC heuristics.

4.4 Summary

We evaluate the accuracy of the recommendation of RACK by collecting 50 Python questions from four open-source sites (freecodecamp.org, programiz.com, geeksforgeeks.org and realpython.com). Three performance metrics (Precision (P), Recall (R), and F1-score (F)) are used to evaluate the performance of all heuristics and three individual heuristics (KKC, KPAC and KAC). In the research, we found ALL heuristics provide the best performance, with KPAC being the most effective individual heuristic. And Java has better performance in Mean Recall@K, and KKC and KAC heuristics.

Chapter 5

Threats to validity

We also face several limitations and threats. We extracted API from Stack overflow, but some issues in Stack overflow may have incorrect labels. When asked some less common questions, the correlation between keywords and API is very low. We have demonstrated that RACK can be used in Python and Java, but it can also be used in other programming languages such as JavaScript, C and C++.

Less common questions: When we search less common questions on RACK, the database may not be able to give accurate answers, and the accuracy of the recommendation API could be very low. We can use web scraping to search for relevant questions and answers across multiple platforms to mitigate such issues.

Diversity of programming languages: We have only two programming languages (Python and Java) available in RACK, and currently it does not support other programming languages such as C, Javascript and C++. In future, we will try more types of programming language from Stack Overflow to test the generalizability of RACK.

Unable to return code: When we use GitHub to find related code repositories, we should only proceed with the next steps when the web response is 200, otherwise, errors occur. If the status codes are different such as 404 (not found), we need to handle the error accordingly. One of the common non-response situations is that we did not give the correct developer token or the original token has expired so the response is not successful. We need to check and update tokens regularly to prevent expiration issues. There are other reasons such as poor network connection. In addition, if the code snippets have syntactic errors, our program can not parse the code snippets.

Chapter 6

Related work

Rack is a method for recommending relevant API classes based on natural language search queries. At the same time, we found other methods that can recommend API classes: ChatGPT, NLP2API, and Thung et al. (API methods are recommended based on feature requests).

6.1 ChatGPT

ChatGPT can use natural language queries to find relevant API classes. ChatGPT parses the search query to find the keywords and understand the query intent. Then the keywords extracted from the query are mapped to the parameters of the API classes to find the relevant ones. Since ChatGPT has learned from millions of web pages and code examples from the Internet, it can effectively provide users with relevant API classes based on their natural language search query.

6.2 NLP2API

NLP2API is one of the methods to find relevant API classes based on natural language search queries. It uses crowdsourced knowledge to recommend the relevant API classes. NLP2API uses natural language pre-processing, word embeddings using fastText and semantic proximity evaluation to find related APIs. NLP2API inputs natural language queries in the an plug-in IDE called NLP2API-runner and can return a ranking list of related API classes [11][12].

6.3 Thung et al.

Thung et al. automatically recommend API methods based on feature requests. It takes natural language queries as input. Thung et al. process queries by extracting relevant information such as abstracts and descriptions. Leveraging a history-based recommendation approach, Thung et al. compare queries with historical feature requests by considering various criteria such as summary, description, component, reporter, and priority to get a historical similarity score (SimHISTORY). The description-based recommendation process calculates a similarity score (RecScore-DESCRIPTION) between queries and API methods based on textual descriptions. The two recommendation scores are combined to produce a final recommendation score (RecScore) for API classes. Thung et al.use weighting adjustments to optimize the contribution of historical and description-based recommendations. The technique provides developers with a ranked list of related API classes that combine historical context and textual similarity to guide effective feature implementation [13].

6.4 Summary

ChatGPT, NLP2API, and Thung et al. can return relevant API classes for natural language search queries. ChatGPT recommends API classes by parsing keywords, mapping them to API class parameters and employing error handling to provide effective suggestions. NLP2API uses crowdsourced knowledge, fastText-based word embeddings, and semantic evaluation to return a ranked recommendation list and display it in the NLP2API-runner. Thung et al. combine historical, description-based recommendations with historical context and textual relevance to calculate similarity scores to recommended API classes.

Chapter 7

Conclusion & Future work

7.1 Conclusion

Software developers spend a lot of time searching for code examples to solve problems. We use RACK for automated code search to enhance the efficiency and accuracy of code search. RACK is used for searching natural language questions to get related-code answers by three individual heuristics — Keyword-API Co-occurence (KAC), Keyword-Keyword Co-occurence (KKC), Keyword Pair API Co-occurrence (KPAC) and all heuristics (KAC, KKC and KPAC). And we use it to make API database mapping. We use 50 examples from four programming sites to test RACK in Python. programming language. We use Precision, Recall and F1-Score to evaluate RACK effectiveness. We built an RACK plug-in tool to let software developers easily search questions and get more accurate answers with code snippets. In total, RACK is a type of effective API class recommendation technology. We designed a VS-Code tool to make the RACK available for Python code search in natural language. We will find 50 questions from four programming sites (freecodecamp.org, programiz.com, geeksforgeeks.org and realpython.com) to test ranked API classes to make sure it's accurate and widely usable [2].

RACK VS-Code plugin can find the keyword token of the problem by analyzing the natural language query. According to the keyword tokens, we get the relevant recommended API classes. We find the code snippets with these API classes in GitHub based on the relevant recommended API classes. Then we arrange them according to the relevance percent (the intersection of ranked API classes with all code's API classes divided by the union of ranked API classes with all code's API classes) and select the top five GitHub websites and return those code snippets. So software developers can quickly find suitable and accurate code snippet answers.

Github link: https://github.com/shihuiMaxine/RACK_2023.git [1]

[1]https://github.com/shihuiMaxine/RACK_2023.git

7.2 Future work

In future work, we will adapt RACK to other programming languages such as C, C++ and PHP. We will try more programming languages in RACK. We use three heuristics, KKC, KAC and KPAC, to query API classes. However, we may try to find more heuristics to replace KPAC to better help software developers find related API classes. We only use Jaccard similarity to calculate the matching between recommended and ground-truth API classes and return code snippets. We may try to use more evaluation methods. In addition, we can let recommended API classes be closer to natural language tokens. We can try using existing state-of-the-art API recommendation techniques to combine with RACK to look for their advantage to work and recommend API classes together.

We collect relevant code snippets from GitHub. We will also ask and answer several questions like these. If we use some other open source websites such as Bitbucket, SourceForge and Codeberg, can we find valid relevant code snippets? If we merge content from multiple open source sites together, we will see if we can find a more efficient and accurate answer with code. We will try different open source sites in the future to find the answer.

Bibliography

[1] MozDevNet, *Parse - mdn web docs glossary: Definitions of web-related terms: Mdn.* [Online]. Available: https://developer.mozilla.org/en-US/docs/Glossary/Parse.

[2] M. M. Rahman, C. K. Roy, and D. Lo, "Rack: Automatic api recommendation using crowdsourced knowledge," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, IEEE, vol. 1, 2016, pp. 349–359.

[3] *The 3 types of search queries amp; how you should target them - wordstream.* [Online]. Available: https://www.wordstream.com/blog/ws/2012/12/10/three-types-of-search-queries.

[4] D. Kundel, *Introduction to abstract syntax trees*, Jun. 2020. [Online]. Available: https://www.twilio.com/blog/abstract-syntax-trees.

[5] Aug. 2023. [Online]. Available: https://www.geeksforgeeks.org/how-to-calculate-jaccard-similarity-in-python/.

[6] [Online]. Available: https://www.w3schools.com/.

[7] *Stack exchange data explorer.* [Online]. Available: https://data.stackexchange.com/stackoverflow/query/new.

[8] [Online]. Available: https://beautiful-soup-4.readthedocs.io/en/latest/.

[9] M. M. Rahman, C. K. Roy, and D. Lo, "Rack: Code search in the ide using crowdsourced knowledge," in *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, IEEE, 2017, pp. 51–54.

[10] M. M. Rahman, C. K. Roy, and D. Lo, "Automatic query reformulation for code search using crowdsourced knowledge," *Empirical Software Engineering*, vol. 24, pp. 1869–1924, 2019.

[11] R. F. Silva, C. K. Roy, M. M. Rahman, K. A. Schneider, K. Paixao, and M. de Almeida Maia, “Recommending comprehensive solutions for programming tasks by mining crowd knowledge,” in *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*, IEEE, 2019, pp. 358–368.

[12] M. M. Rahman and C. Roy, “Nlp2api: Query reformulation for code search using crowdsourced knowledge and extra-large data analytics,” in *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, IEEE, 2018, pp. 714–714.

[13] F. Thung, S. Wang, D. Lo, and J. Lawall, “Automatic recommendation of api methods from feature requests,” in *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, IEEE, 2013, pp. 290–300.