Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

ARTICLE TYPE

On Code Reuse from StackOverflow: An Exploratory Study on Jupyter Notebook

Mingke Yang[1] | Yuming Zhou[2] | Bixin Li[3] | Yutian Tang[4]

[1]School of Information Science and Technology, ShanghaiTech University, Shanghai, China
[2]Department of Computer Science and Technology, Nanjing University, Nanjing, China
[3]School of Computer Science and Engineering, Southeast University, Nanjing, China
[4]School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, China

Correspondence
Yutian Tang, Nanjing University of Science and Technology, China. Email: csytang@ieee.org

Abstract

Jupyter Notebook is a popular tool among data analysts and scientists for working with data. It provides a way to combine code, documentation, and visualizations in a single, interactive environment, facilitating code reuse. While code reuse can improve programming efficiency, it can also decrease readability, security, and overall performance. We conduct a large-scale exploratory study of code reuse practices in the Jupyter Notebook development community on the Stack Overflow platform to understand the potential negative impacts of code reuse. Our findings identified 1,097,470 Jupyter Notebook clone pairs that reuse Stack Overflow code snippets, and the average code snippet has 7.91 code quality violations. Through our research, we gain insight into the reasons behind Jupyter Notebook developers' decision to reuse code and the potential drawbacks of this practice.

KEYWORDS:
Jupyter Notebook, StackOverflow, Code Reuse

1 | INTRODUCTION

In the common paradigm of open source software development, the organization can be global and virtual, facilitated by the Internet and virtual communication among developers. The success of common open-source software, such as Linux and LibreOffice, relies not only on the efforts of developers but also on other publicly available software on the Internet. Code reuse is a widely adopted strategy for delivering high-quality software products efficiently. This approach allows developers to build upon existing work, rather than starting from scratch, to create new software solutions in a timely manner[1]. Code reuse can come from a variety of sources and in different forms, such as open source projects and software[2], Q& A platform (e.g., Stack Overflow)[3].

In recent years, the availability of large amounts of data and powerful computing resources led to the growth of fields such as data science and machine learning, where people in a wide range of fields, from healthcare and finance to social media and e-commerce are exposed to code[4]. These people who are new to programming or data analysis tend to solve their coding problems through code reuse with the following purposes: (1) save development time by reusing code to implement specific functionality; (2) improve code efficiency by finding best practices; and (3) fix bugs in existing code by reusing bug-fixing code snippets. Therefore, it is common for inexperienced developers to focus on whether a code snippet meets their specific needs and implements the expected features, rather than on the overall quality of the code. This can be a problem as they may reuse poorly written or unreliable code, leading to errors, bugs, and other issues.

State-of-art. Stack Overflow, the largest and most active online community for programmers and developers to ask and answer questions related to coding and software development, is a popular resource for code reuse. In the meantime, according to[5], Jupyter Notebook is the most widely used tool for analyzing data. Jupyter Notebook combines two components, a web application

and notebook documents[6]. The web application provides an interactive way to write notebook documents. Notebook documents contain codes, computational output, text and multimedia materials, it can serve as complete calculation records. As it provides a user-friendly interface and a powerful set of tools, it is particularly popular among inexperienced developers.Previous studies on Stack Overflow[7,8,9] and Jupyter Notebook[5,10] were conducted in isolation. Specifically, Zhang et al.[7]. designed ExampleCheck to detect potential API usage violations in Stack Overflow posts, and Ragkhitwetsagul et al.[8]. studied toxic code snippets in Stack Overflow. Fischer et al.[9]. studied security issues caused by stack overflow snippets in android apps. Wang et al.[5] experimentally demonstrated the existence of a large amount of poor-quality code in the Jupyter Notebook. Koenzen et al.[10] investigated code reuse between Jupyter Notebook.

Motivation. However, to the best of our knowledge, there is no study on code reuse in Jupyter Notebook. In this paper, we conducted an exploratory study to fill this gap. Specifically, we provide evidence of the prevalence of code reuse in Jupyter Notebook, summarize why developers reuse code in Jupyter Notebook, and the impact of code reuse on code quality.

Contribution. In this paper, we make the following contributions:

• First, to the best of our knowledge, we perform the *first* large-scale exploratory study on Jupyter Notebook to explore the code reuse practices from the Stack Overflow platform for the development of Jupyter Notebook.

• Second, we conduct a systematic study on 3,758,196 Jupyter Notebook and 4,204,891 Stack Overflow code snippets. We find 1,097,470 Jupyter Notebook clone pairs that reuse Stack Overflow code snippets. On average, a code snippet has 7.91 code quality violations.

• Third, we explore the reasons why Jupyter Notebook reuses Stack Overflow code snippets in terms of their Stack Overflow properties (e.g. whether they are accepted), developer experience, and developer purpose.

Skeleton. The rest of this paper is organized as follows: in Sec. 2, we present the background and basic concepts in Jupyter Notebook and code reuse. In Sec. 3, we present the research questions and describe the data collection process and the reuse detection methods. In Sec. 4, we present the results and findings of our exploratory study. We discuss the lessons learned and threats to the validity of our work in Sec. 5. In Sec. 6, we introduce the related work of our study. In Sec. 7, we conclude our study.

2 | BACKGROUND

In this section, we briefly introduce several key concepts related to Jupyter Notebook, Stack Overflow, and code reuse. Furthermore, we leverage a running example to illustrate the code reuse practice with Stack Overflow.

2.1 | Jupyter Notebook

A Jupyter Notebook contains a series of *cells*[11]. The type of a *cell* can be a *code* cell or a *markdown* cell. A *code* cell is a cell that contains executable source code. A *markdown* cell is a rich text cell that supports markdown language, which is normally used to describe the code snippets in *code* cells.

Fig. 1 illustrates a Jupyter Notebook example. It contains one markdown cell and two code cells. On the left of those cells, there are *execution counters*, which show the execution order of these cells. The execution results of code cells are shown as outputs, which are displayed later. Note that the execution order only indicates the order of cells rather than any logical relationship between them.

2.2 | Code on Stack Overflow

Stack Overflow is one of the largest online Q&A platforms for coding questions and answers[12]. In a Stack Overflow post, users can ask and answer questions and discuss the questions. Also, as shown in Fig. 2, askers can mark at most one answer as an “accepted” answer. Stack Overflow also allows developers to upvote and downvote answers. The number of votes for an answer is displayed as a score next to the answer. For example, the score of the answer in Fig. 2 is 6777, which indicates it is a high-quality solution to the question.

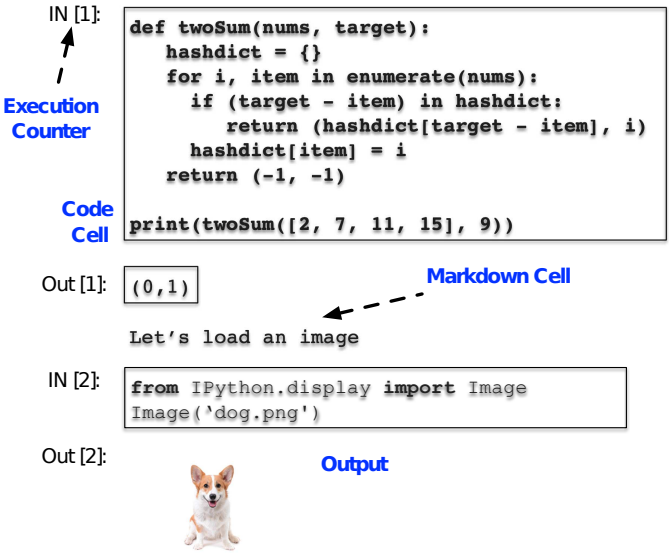

Figure 1 Jupyter Notebook Example

Figure 2 Stack Overflow Example

2.3 | A Motivating Example

Existing research shows that developers leverage Stack Overflow code snippets to build their own programs [13,8,1]. In this work, we intend to study how developers reuse code from Stack Overflow to create Jupyter Notebook.

In what follows, we show the motivation of our research by describing a real-world example in which developers reuse code snippets from Stack Overflow in building Jupyter Notebook.

The left-hand side of Fig. 3 shows a code snippet (e.g., function `fill_lower_diag(a)`) from an answer on a Stack Overflow post. The answer shows how to convert a one-dimensional array into a lower, zero diagonal matrix while keeping all the digits. The right-hand side of Fig. 3 shows a code snippet from a real-world Jupyter Notebook named `BigData_Linear_Correlation_Pairplot.ipynb`.

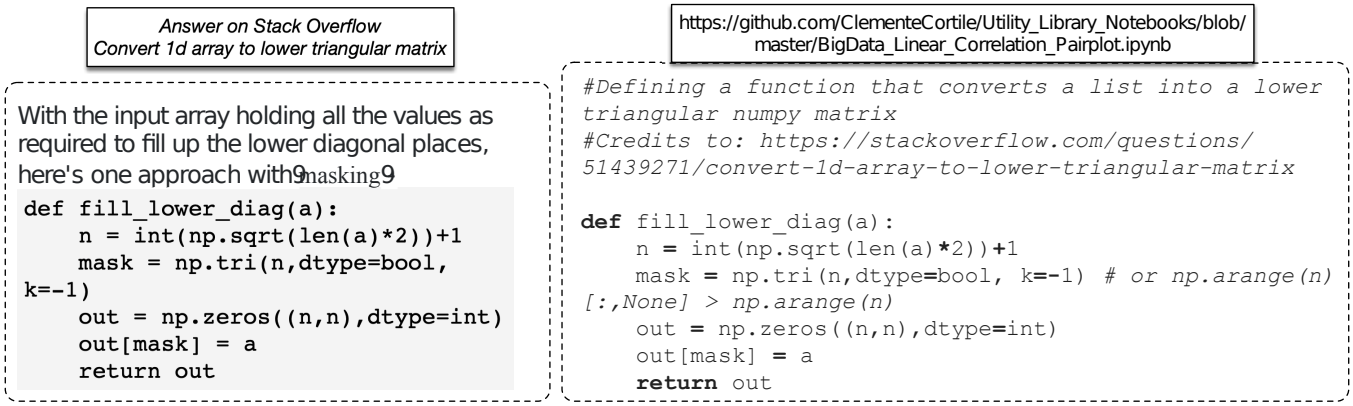

Figure 3 Source code from Stack Overflow is reused in building a Jupyter Notebook

By inspecting the timestamps of the commit and Stack Overflow, we can determine the code in the Jupyter Notebook reuses the code on the Stack Overflow post. In this Notebook, the function from Stack Overflow is directly reused by developers to convert a one-dimensional array into a lower, zero diagonal matrix. The general target of `BigData_Linear_Correlation_Pairplot.ipynb` is to offer a series of functions to build a utility library to check the linear correlation between all variables in an extremely large dataset. Developers reuse the method `fill_lower_diag(a)` to convert the one-dimensional array into a matrix, which is later used to build a heat map.

With the aforementioned running example, we can conclude that identifying reused code snippets from Stack Overflow benefits the following aspects:

• First, we can measure how much code reuse there is in Jupyter Notebook;

• Second, code on Stack Overflow can contain potential defects, which can have negative impacts on other programs;

• Third, understanding code reuse practice assists us in determining the potential motivation for code reuse and improving code reuse practices.

3 | RESEARCH QUESTIONS & METHODOLOGY

3.1 | Research Questions

We perform an empirical study of online code clones between Stack Overflow and Jupyter Notebook on GitHub to answer the following research questions (RQ):

- **RQ1: How many code clone happened in Juypter Notebook?** To understand the code clone practice in Jupyter Notebook, we quantitatively measure the number of code clones between Stack Overflow and Jupyter Notebook on GitHub to understand the scale of the problem;
- **RQ2:Which part of the Stack Overflow post is used?** Intuitively, accepted answers or answers with high scores are more useful and trustworthy. In this RQ, we intend to investigate whether developers reuse non-accepted or low-scored code snippets. Investigating this RQ can assist Q&A platforms in organizing answers;
- **RQ3: Why do developers reuse code from Stack Overflow?** Developers are more prone to reuse code snippets from Stack Overflow. In this RQ, we intend to answer why developers reuse code and what the motivations are.
- **RQ4: What is the quality of code snippets provided in answers on Stack Overflow?** When reusing code snippets from Q&A platforms like Stack Overflow, developers should consider the quality of the code snippets used. In this RQ, we intend to evaluate the quality of the code snippets from Stack Overflow.
- **RQ5: Who reuses code from Stack Overflow?** Previous research[14] demonstrated that reusing code can have negative effects on building software. This always associates with less experienced developers. Here, we intend to examine who reuses Stack Overflow code among developers.

3.2 | Methodology

In this section, we present the pipeline for processing block-level code reuse between Stack Overflow and Jupyter Notebook projects on GitHub.

3.2.1 | Pipeline

We first briefly outline the pipeline of our approach. As shown in Fig. 4, we extract code snippets from Stack Overflow posts and Jupyter Notebook files. Then, we use MD5 to identify the Type-1 code clone pair and SourcererCC for the Type-2,3 code clone pair. Next, we use timestamps to get the clone pairs of Jupyter Notebook reused Stack Overflow code snippets. Finally, we analyze the clone pairs using different methods.

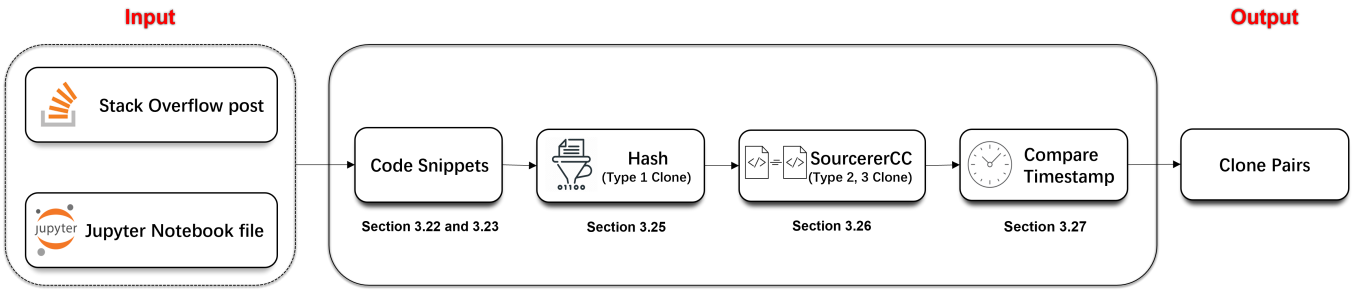

Figure 4 Pipeline of our study

3.2.2 | Extract Code Blocks from Jupyter Notebook

To understand how code reuse happened in real-world Jupyter Notebook, we collect Jupyter Notebook from GitHub. Here, we follow the approach presented in [15]. We collect Jupyter Notebook projects hosted on GitHub by specifying the identification language as "Jupyter Notebook" with the GitHub API. We collect Jupyter Notebook repositories built between 2013/01/01 and 2020/04/01. In summary, we obtain 188,302 repositories. For each Jupyter Notebook repository, we collect code snippets from each Notebook file (i.e., ends with `.ipynb`). The official Jupyter Notebook format is defined with the JSON schema.

The following JSON scheme is used to render a code cell in a Jupyter Notebook. The *cell_type* attribute is *code*, which represents the cell as a *code* cell. The value *source* attribute gives the source code in the cell. By parsing the Notebook files, we are able to extract the code snippets from the Notebook.

```
{
  "cell_type" : "code",
  "execution_count": 1, # integer or null
  "metadata" : {
       "collapsed" : True, # whether the output of the cell is collapsed
       "scrolled": False, # any of true, false or "auto"
  },
  "source" : "[some multi—line code]",
  "outputs": [{
       # list of output dicts (described below)
       "output_type": "stream",
       ...
  }],
}
```

Finally, we obtain 3,758,196 code snippets.

3.2.3 | Extracting Code Blocks from Stack Overflow Posts

We collect Stack Overflow posts from The SOTorrent dataset [16]. The SOTorrent dataset collects the Q&A posts from Stack Overflow. The latest version of SOTorrent contains the data updated to Dec,31,2020. We select Q&A posts with the tag "jupyter-notebook" or "python" for our purpose. The tag "jupyter-notebook" gives all questions and answers related to Jupyter Notebook.

As Python is the major programming language for Jupyter Notebook, we also collect posts with the tag "Python". In summary, we obtain 1,764,935 posts. The existing works[3,17] find that when the number of lines is greater than 5 it is more reasonable and meaningful. Thus, we adopt this setting to filter out code snippets within 5 lines. To be specific, for each post, we extract the code snippets from the post. The code snippets inside a post are embedded in the "code" tag. Then, we remove redundant code snippets by computing the MD5 of each code snippet. As a result, we obtain 4,204,891 code snippets.

3.2.4 | Type 1-4 Clones

In general, code clones have been categorized into four types of clones: Type-1 to Type-4[18]. Specifically, these four types of clones are:

- **Type-1 Clone:** This type of clone refers to direct copy, and the only differences are spaces and comments. This type of clone can be detected easily;
- **Type-2 Clone:** Same as the Type-1 clone, Type-2 clone but also allows renaming variables;
- **Type-3 Clone:** Same as the Type-1 and Type-2 clone, Type-3 clone but also allows adding/deleting some statements;
- **Type-4 Clone:** Semantically identical, but not necessarily the same syntax.

In this paper, we focus on Type 1-3 clones. The following reason makes us discard Type-4 clone: in this paper, we aim to discuss how developers reuse code from Stack Overflow. For the Type-4 clone, it is hard to claim that developers reuse the code snippet from Stack Overflow only based on semantic similarity.

3.2.5 | Type-1 Clone

To determine whether two code snippets are Type-1 clones, for two code snippets under comparison, we first remove all white spaces, including newlines and comments from the snippets. Second, we compute the MD5 hash for each code snippet. If two code snippets have the same MD5 hash, they are considered Type-1 clone pairs.

Example. Next, we leverage the following example to illustrate our Type-1 clone detection process. For the code snippets in List. 1 and 2, we first remove all white spaces, including newlines and comments from snippets. Next, we compute the MD5 hash for the code in the List. 1. The result is "00cee7ab3fa4839aad42c795415daa47". We get the same MD5 hash for the code in the List. 2. Thus, we consider two code snippets are Type-1 cloned.

Listing 1: Type-1 Clone Example(a)

```
def max(a, b):
    if a > b:
        return a
    else:
        return b
```

Listing 2: Type-1 Clone Example(b)

```
# Return the maximum of two numbers
def max(a, b):
    if a > b:
        return a # a is the maximum
    else:
        return b # b is the maximum
```

3.2.6 | Type-2 and Type-3 Clone

To determine Type-2 and Type-3 code clone, we leverage SourcererCC[19] to detect such clones. SourcererCC is a token-based clone detection tool that supports Type-1 to Type-3 clones. As we already have presented how to detect Type-1 clones, here, we mainly use SourcererCC to detect Type-2 and Type-3 clones.

SourcererCC performs clone detection with two phases: partial index creation and clone detection. In the first phase (partial index creation), SourcererCC parses the code blocks from the source files. Then, it tokenizes the code blocks with a simple

scanner, which is aware of the token and block semantics of the target programming language (i.e., Python in our context). Next, it builds inverted index mapping tokens to blocks that contain them. It leverages a filtering heuristic to build a partial index of only a subset of the tokens in each block instead of building indexes for all tokens. Here, the filtering heuristic is used to reduce the candidates for comparison.

In the second phase (clone detection), SourcererCC iterates all code blocks. For each code block, it retrieves its candidate clone blocks from the index built in the previous phase. Furthermore, it leverages another filtering heuristic to compute the upper- and lower-bound similarity scores between the current block and its clone candidates. If the upper-bound similarity score for a candidate is lower than the predefined threshold, the process is terminated. If the lower-bound similarity score is higher than the predefined threshold, a cloned candidate is found. The aforementioned process is repeated until all cloned pairs are located.

We use the SourcererCC's default similarity threshold for clone detection, which is 80%. That is, two code snippets are considered the clone pair if the similarity score is not less than 80%.

Example. Next, we leverage a running example to illustrate the use of SourcererCC.

Listing 3: Type-3 Clone Example

```
def max(a, b):
    if a > b:
        print()
        return a
    else:
        return b
```

Code snippets in List. 1 and List. 3 belong to the Type-3 clone, as the changes are added statements. The first step is tokenization. For each code snippet, the scanner parses the code snippet and outputs two files: *files_stats* and *files_tokens*. The *files_stats* file records statistics information of files, including file id, project id, project path, project URL, file hash, size bytes, lines, LOC (line of code), and SLOC (source line of code). The difference between LOC and SLOC in the SourererCC is that SLOC does not consider comment lines. The *files_tokens* file records lists of files together with various statistics and tokenized forms with the format: file id, project id, total tokens, unique tokens, token hash@ #@token1@@::@@frequency, token2@@::@@frequency... Here, the "token1@@::@@fre" refers that the frequency of "token1" is "fre". As a result, we obtain the following in the *files_stats* file:

Listing 4: files_stats file Example

```
1,1,"1.zip/1/max.py","list1/max.py","94621.(hash value)..",73,5,5,5
2,2,"2.zip/3/max.py","list3/max.py","28fa9.(hash value)..",90,6,6,6
```

We obtain the followings in the *files_tokens* file:

Listing 5: files_tokens file Example

```
1,1,12,7,e5d9...(hash value )..#def::1,max::1,a::3,b::3,if::1,return...
2,2,13,8,70a6...(hash value )..#def::1,max::1,a::3,b::3,if::1,...
```

Finally, SourcererCC outputs the indexes of cloned pairs.

3.2.7 | Compare Timestamp

After obtaining the clone pair, we use the Stack Overflow API and Git command to acquire the timestamp of the code snippet's first appearance on Stack Overflow and GitHub. Then, we select clone pairs whose code snippets appeared on Stack Overflow earlier than Jupyter Notebook as target clone pairs. Specifically, for Stack Overflow, we use https://api.stackexchange.com/2.3/posts/{POST_ID}/revisions?site=stackoverflow&filter=!6M2o(oKM-oyhS to get edit revisions of the target Stack Overflow post. Then we use the clone code snippet to search its earliest appearance in the edit revision body and get the revision timestamp. For Jupyter Notebook, we first use `git log` to get all the commit ids and their corresponding commit dates. Then, we use `git checkout COMMIT_ID` to update files in the specified commit. We search for the clone code snippet in the file where the clone is detected and get its earliest commit date.

Example. For the clone pair in Fig. 3, we first query the Stack Overflow API, and the result is shown in List. 6. By comparing the timestamps, we find that the revision with timestamp 1532079237 is the earliest, which is Jul 20, 2018. Then, we use `git log` to get the commits of the Jupyter Notebook project, the result is shown in List. 7. Next, we use the `git checkout COMMIT_ID` to get the files in the specified commit. Searching for cloned code snippets, we find that Jul 13, 2019, is the matched commit date.

By comparing the timestamps, we can determine the Jupyter Notebook project reused the code snippet on the Stack Overflow post.

Listing 6: Stack Overflow API query result

```
{
    items: [
     {
          "creation_date":1532080688,
          "body": "...def fill_lower_diag(a):...",
     },
     {
          "creation_date": 1532079237,
          "body": "...def fill_lower_diag(a):...",
     }
    ]
...
}
```

Listing 7: Git command result

```
...
commit 55940698106310f8f5d8750060cf73b740c13e15
Date: Sat Jul 13 14:34:54 2019 +0200
...
commit e8dd1d684b8122480d6bf100ee013dac156e17ec
Date: Sat Jul 13 14:34:34 2019 +0200
...
```

4 | RESEARCH QUESTIONS AND DISCUSSION

4.1 | RQ1: How many code clone happened in Juypter Notebook?

Motivation. To understand the code clone practice in Jupyter Notebook, in this RQ, we quantitatively measure the number of code clones between Stack Overflow and Jupyter Notebook on GitHub to understand the scale of the problem.

Table 1 Result of Clone Detection for Clone Pairs.

	Clone Pairs	
Clone Type	**Without Timestamp Comparison**	**With Timestamp Comparison**
Type-1	71,851	33,448
Type-2,3	1,923,943	1,064,022
All Clone Type	1,995,794	1,097,470

Table 2 Result of Jupyter Code Snippets in Clone pairs.

	Jupyter Code snippets	
Clone Type	**Without Timestamp Comparison**	**With Timestamp Comparison**
Type-1	30,138	19,638
Type-2,3	272,560	184,727
All Cloned Snippets	285,545	193,248

Methodology. The methodology for this RQ is presented in Sec. 3.2.

Results. The result of our clone pair detection is shown in the Table. 1. We find 1,995,794 clone pairs containing 83,747 stack overflow code snippets and 285,545 Jupyter Notebook code snippets. The number of code snippets is less than the number of clone pairs because there is a situation where a code snippet is reused multiple times. These clone pairs also relate to 71,627 Stack Overflow posts and 59,942 Github repositories, representing 4.06% of collected posts and 31.83% of collected repositories. We summarize the number of Jupyter Notebook code snippets in cloned pairs, shown in the Table. 2.

For **Type-1 Code Clone**, we find 58,446 Type-1 clone pairs. We use the hash value to filter out identical code snippets and then count the number of distinct code snippets. Interestingly, there are only 5,472 distinct Stack Overflow code snippets, much smaller than the total number of clone pairs we have found. On average, every code snippet is reused 10.68 times. Using the timestamp of the code snippet first shown in the Jupyter Notebook file and Stack Overflow, we find 33,448 code clone pairs are Jupyter Notebook code snippets that reused code from Stack Overflow posts.

Next, we present the top five code snippets commonly reused with Type-1 code clone. To focus on code that implements specific function. We ignore meaningless code snippets such as importing libraries and defining initial data. The most commonly reused code snippet is for setting up Jupyter Notebook to hide input code and only show outputs and markdown, shown in List. 8. This code snippet was reused 273 times. When a Jupyter Notebook developer shares the notebook with others but only wants to share the results and text rather than the code. This code is reused and placed at the beginning of the notebook.

Listing 8: Most Type-1 reused code

```
from IPython.display import HTML

HTML('''<script>
code_show=true;
function code_toggle() {
 if (code_show){
 $('div.input').hide();
 } else {
 $('div.input').show();
 }
 code_show = !code_show
}
$( document ).ready(code_toggle);
</script>
<form action="javascript:code_toggle()">
<input type="submit"
value="Click here to toggle on/off the raw code."></form>''')
```

As shown in List.9, the second most reused code(243 times) loads detection graph from the external directory. When training a TensorFlow model, it is common to save it as a .pb (protocol buffers) file. Protocol Buffers is a method of serializing data[20], in this case for saving graph definitions and model weights. This code is reused when developers want to use the model defined in the protobuf file.

Listing 9: Second most Type-1 reused code

```
detection_graph = tf.Graph()
with detection_graph.as_default():
    od_graph_def = tf.GraphDef()
    with tf.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
        serialized_graph = fid.read()
        od_graph_def.ParseFromString(serialized_graph)
        tf.import_graph_def(od_graph_def, name='')
```

The third most commonly reused code snippet(227 times) is for adding custom CSS style for notebook, is shown in List. 10. This code is reused when the developer is not satisfied with the default style of Jupyter Notebook. This code is often placed at the end of the notebook to isolate it from the data analyze code.

Listing 10: Third most Type-1 reused code

```
#Apply styles
from IPython.core.display import HTML

def css_styling():
    styles = open("styles/custom.css", "r").read()
    return HTML(styles)
css_styling()
```

The fourth most commonly reused code snippet is to get the files uploaded to Colab and print it, as shown in List. 11. This code snippet reused 227 times. Google Colab is a free Jupyter notebook environment that runs in the cloud. The code snippet is used to query which file has been uploaded (usually data files), and will be used later.

Listing 11: Forth most Type-1 reused code

```
from google.colab import files
uploaded = files.upload()
for fn in uploaded.keys():
  print('User uploaded file "{name}" with length {length} bytes'.format(
      name=fn, length=len(uploaded[fn])))
```

Last, the fifth most reused code snippet(222 times) is shown in the List. 12. This code snippet is used `StandardScale` to scale the dataset to unit variance. After loading, the data may need to be preprocessed using normalization. This code snippet is reused when data needs to be scaled to the standard normal distribution.

Listing 12: fifth most Type-1 reused code

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

After introducing the 5 most common Type-1 reuse code snippets, we find that the reason for reuse can be divided into two categories:

- Customise Jupyter Notebook: List.8 sets the visibility of code cells. List.10 using external CSS to custom Jupyter Notebook theme. As the functionality of these code snippets is not related to data analysis, unskilled developers may lack knowledge about this part, thus they reuse code snippets from Stack Overflow.
- Prepare for data analysis: List.9 loads the trained model. List.11 queries the files that have been uploaded to the cloud. List.12 normalizes the data for preprocessing. They are off-the-shelf implementations of these functions on Stack Overflow, so developers reuse these code snippets.

For **Type-2,3**, compared to Type-1 code clone, the detection criteria of Type-2,3 code clone are less stringent. We find 1,937,348 pairs of Type-2,3 clones. This is about 33.15 times the number of Type-1 clone pairs and corresponds to 97.07% of the clone pairs. Similarly, we find 1,064,022 clone pairs are Stack Overflow posts code snippets reused by Jupyter Notebook from the Type-2,3 Code Clone pairs. These clone pairs consist of 65,811 Stack Overflow code snippets and 184,727 Jupyter Notebook code snippets, representing 1.60% of the Stack Overflow code snippets and 4.91% of the Jupyter Notebook code snippets we collected. It is remarkable that 1,097,470 clone pairs are the Jupyter Notebook reuse Stack Overflow posts.

Similarly, we present the top five code snippets commonly reused with Type-2,3 code clone. The most common code reuse snippets for Type 2,3 is shown in List.13. This code snippet reused 3601 times. It is used to set up the notebook so that figures are displayed inline in the notebook, and initialize the plotting settings. This code is reused when developers need to set up `matplotlib`. In Type 2,3 code reuse, developers usually change the `pyplot`'s runtime configuration options(rcParams).

Listing 13: Most Type-2,3 reused code snippets

```
import random
import numpy as np
from cs231n.data_utils import load_CIFAR10
import matplotlib.pyplot as plt
%matplotlib inline
plt.rcParams['figure.figsize'] = (10.0, 8.0) # set default size of plots
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'
```

In supervised machine learning, to prevent overfitting, the dataset is usually divided into two parts: the training set and the test set. The second most frequently reused code (1845 times) snippet is to load such dataset and print the shape of the dataset, as shown in the List. 14. Supervised learning is the machine learning task of inferring functions from a labeled dataset. When developers need to import the dataset, this code is reused. The sanity check is performed by printing the size of the dataset. A common change to use this code snippet is to modify the dataset path or dataset name.

Listing 14: Second most Type-2,3 reused code snippets

```
cifar10_dir = 'cs231n/datasets/cifar-10-batches-py'
X_train, y_train, X_test, y_test = load_CIFAR10(cifar10_dir)

print 'Training data shape: ', X_train.shape
print 'Training labels shape: ', y_train.shape
print 'Test data shape: ', X_test.shape
print 'Test labels shape: ', y_test.shape
```

After importing data, a common next step is to reshape the data into a form that facilitates the rest of the data analysis. As shown in List.15, the third most reused code (1285 times) is formatting the dataset shape and printing size of the dataset. Similar to the List 14, this code snippet performs reshape on multiple datasets, training the model with supervised learning. This code snippet is used to pre-process the data. Then the dataset size is printed for sanity check. A common way to reuse Type 2,3 for this code snippet is to change the parameters in `reshape`.

Listing 15: Third Most Type-2,3 reused code snippets

```
def reformat(dataset, labels):
  dataset = dataset.reshape((-1, image_size * image_size)).astype(np.float32)
  labels = (np.arange(num_labels) == labels[:,None]).astype(np.float32)
  return dataset, labels
train_dataset, train_labels = reformat(train_dataset, train_labels)
valid_dataset, valid_labels = reformat(valid_dataset, valid_labels)
test_dataset, test_labels = reformat(test_dataset, test_labels)
print('Training set', train_dataset.shape, train_labels.shape)
print('Validation set', valid_dataset.shape, valid_labels.shape)
print('Test set', test_dataset.shape, test_labels.shape)
```

The fourth most commonly reused code snippet(1150 times) is for plotting the confusion matrix, as shown in List. 16. This code snippet is reused when users need to draw the confusion matrix. The common reused method of Type-2,3 is to change the plot drawing options without changing the data processing part.

Listing 16: Fifth Most Type-2,3 reused code snippets

```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, round(cm[i, j],4)*100,
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

Last, the fifth most commonly reused code snippet (1008 times) is shown in the List. 17. This code snippet is the same as List. 10, change the style of the notebook by reusing this code snippet. Changing the parameters in the `open` function is a common way to reuse it in Type-2,3 Clone.

Listing 17: Fifth most Type-2,3 reused code snippets

```
from IPython.core.display import HTML
def css_styling():
    styles = open("./example.css", "r").read()
    return HTML(styles)
css_styling()
```

After introducing the 5 most common Type-2,3 reuse code snippets, we find that the reason for reuse can be divided into three categories:

- Customise Jupyter Notebook: List. 17 loads external CSS to modify Jupyter Notebook's style settings. The reason for reusing this code snippet is the same as Type-1 code reuse.
- Prepare for data analysis: List. 14 loads the dataset and prints the dataset size. List. 15 reshapes the dataset. The reason for reusing those code snippets is the same as Type-1 code reuse.
- Plot setting: List. 13 sets the `pyplot` runtime configuration options. List. 16 plots the confusion matrix by setting `pyplot`. Plotting the data is an important part of data analysis. However, there are some common parameters that need to be set when plotting, such as `figsize` or `title`. Developers look for help from Stack Overflow to set those parameters. Since different developers have different ways of presenting data, they may have different settings for the plot. Therefore this part of the function is often used with Type-2, 3 code reuse instead of Type-1 code reuse.

ANSWER TO RQ1 (CODE REUSE PRACTICE)

By investigating 3,758,196 Jupyter Notebook and 4,204,891 Stack Overflow code snippets, we find 1,995,794 clone pairs containing 83,747 stack overflow code snippets and 285,545 Jupyter Notebook code snippets. The ratio of Type-1 to Type-2,3 code clones is about 1:33.15. Among all clone pairs, 33,448 Type-1 and 1,064,022 Type-2,3 clone code pair is the code snippets from the Stack Overflow post and reused by Jupyter Notebook code snippets. We find that customizing Jupyter Notebook and preparing for data analysis are the reasons for most Type-1 reusing code snippets. The reason most reused code snippets are reused by Type-2,3 is the same as Type-1 code reuse but with adding settings for the plot.

4.2 | RQ2: Which part of the Stack Overflow post is used?

Motivation. Intuitively, we suppose that accepted solutions or answered solutions with high votes are more useful and trustworthy. In this RQ, we intend to investigate whether developers use non-accepted answers or answers with low scores. Meanwhile, we also intend to investigate whether exists relationships between the types of clones and the quality of answers. For example, a high-quality solution may be reused without any modification (i.e., Type-1 clone).
Methodology. To cope with this, in this RQ, we check which parts (e.g., accepted answer, non-accepted answer, questions) of the Stack Overflow post are reused by developers. We check whether the reused code snippet is with the highest score (a.k.a. votes). Note that vote can be a positive value or a negative value. A negative value indicates the number of negative votes is larger than the number of positive votes. While a positive value indicates, the number of positive votes is larger than the number of negative votes. Finally, we ask two authors of this paper to categorize the motivations for reusing those parts of code snippets. If there exist discrepancies, another author of this paper is involved in the discussion until a consensus is reached.
Results. We count the number of code snippets of reused code snippets in different post sections. The result is shown in Table. 3.

•**Reuse question section code snippets** We observe 2485 (3.55%) code snippets reused by Type-1 from the question section and 37,339 (53.28%) for Type-2,3 code reuse. There are more reused code snippets from the question section than the answer section.

By manually checking for reused code snippets in Question section. we summarize the reasons that developers reuse them.

- Demo/Tutorial code snippets: These code snippets come from other sources, such as sample code from official documentation, libraries, or books. From the questioner's point of view, they often ask questions about how it works. From the developers' point of view, the code snippet meets their needs;

Table 3 Number of reused code snippets from different sections in Stack Overflow.

	Frequency		Average Vote	
Source	**Type-1**	**Type-2,3**	**Type-1**	**Type-2,3**
Question	2,485 (3.55%)	37,339 (53.28%)	17.89	12.53
Accepted answer	631 (0.90%)	10,756 (15.35%)	35.56	12.03
Non-accepted answer	1,148 (1.64%)	17,716 (25.28%)	11.25	5.33

- Reusing test cases: This code snippet is test code written by the questioner, and other developers also reuse this code snippet to test specific features;
- Fixing buggy code: The questioner demonstrated a buggy code snippet in the question section. The developers reuse it by fix the buggy codes. For example, as shown in List. 18, a buggy code is provided in the question section, due to Python's If statement selects the branch which first satisfies the condition. So when the input is 120, the output is `HOT` instead of `REALLY HOT!`. The bug in the code snippet has been fixed in Jupyter Notebook by reordering the If branches, shown in the List. 19.

Listing 18: Buggy code snippet in question section

```
temp = 120
if temp > 85:
    print("Hot")
elif temp > 100:
    print("REALLY HOT!")
elif temp > 60:
    print("Comfortable")
else:
    print("Cold")
```

Listing 19: Reused code snippet in Jupyter Notebook

```
temp = 120
if temp > 100:
    print "REALLY HOT!"
elif temp > 85:
    print "Hot"
elif temp > 60:
    print "Comfortable"
else:
    print "Cold"
```

The average votes for code snippets from the Question section are 17.89 for Type-1 and 12.53 for Type-2,3. However, the average vote for the entire question section we collected is 9.28.

•**Reuse accepted answer code snippets** For reused code snippets in the accepted answer question, we find 631 (0.90%) Type-1 code reuse and 10,756 (15.35%) Type 2,3 code reuse. Code snippets from the accepted answers represent 37.84% of all answers. The average acceptance rate of all the answers in our data is 34.57%, which shows that developers do not always reuse code from accepted answers.

We summarize the reasons that developers reuse the accept answer section code snippet.

- Provide developer requirement features: These code snippets implement the functionality mentioned in the question;
- Fix the question section's code bug: The accepted answer fixes the buggy code in the question section. The developer reuses this code to fix the same bug; For example, as shown in List. 20, The questioner uses `str()` to convert from Unicode to UTF-8 text, which throws an UnicodeEncodeError. The accepted answer shows that the questioner should use `.encode()` to encode the string, as shown in List. 21.

Listing 20: Buggy code in question section

```
p.agent_info = str(agent_contact + ' ' + agent_telno).strip()
```

Listing 21: Correct code snippet in accepted answer section

```
p.agent_info = u' '.join((agent_contact, agent_telno))\
                    .encode('utf-8').strip()
```

- API Usage: It demonstrates how to use the library's APIs. Developers reuse these code snippets to invoke the same API;

We collect an average of votes for all code snippets from answer sections 2.97. Answers containing reused code have an average of votes higher than that, regardless of the reuse Type of code snippet.

•**Reuse non-accepted answer code snippets** For code snippets in non-accepted answer, there are 1,148 (1.64%) reused by Type-1 and 17,716 (25.28%) reused by Type 2,3.

The reasons developers reuse code snippets in the non-accepted answer section are summarized below.

- Different code version: The version of the accepted answer's code does not match the developer's needs. For example, the code snippet in the accept answer is written in Python 2. Developers have to reuse the code segment in Python 3 from a non-accept answer;

- More generic code: The accepted answer to this question fit the questioner's requirements. However, this answer is specific to the questioner's problem. It does not easily transfer to other problems, but the non-accepted code snippet gives a generic way to solve the problem. For example, in the question section, the questioner asked how to concatenate `listone = [1, 2, 3]`, `listtwo = [4, 5, 6]` the two lists in the python. The accepted answer shown List.22 use + operator to solve this problem. However, the non-accept answer provides a function to combine multiple lists, shown in the List. 23.

Listing 22: Combine two lists

```
listone = [1,2,3]
listtwo = [4,5,6]

listthree = listone + listtwo
```

Listing 23: Combine multiple lists

```
def merge(*lists):
    rslt = [""]
    for idx in range(len(lists[0])):

        r = []
        for s in rslt:
            for l in lists:
                r.append(s + l[idx])
        rslt = r
    return rslt
```

- Code with more details: The accepted answer invokes library APIs to implement the corresponding functionality, but the non-accepted code snippet implements the functionality itself. It has more implementation details, the developer turn to reuse it. For example, in the accepted answer, python's `sort` function is invoked to sort a list, while a quick sort algorithm is implemented to sort the list in the non-accept answer; and

- More efficient code: The non-accepted code snippet is modified on the accepted code snippet to make it more efficient. For example, in the accepted answer, the Fibonacci sequence is calculated using recursion as shown in the List. 24. But in the unaccepted answer, a matrix is used for the calculation shown in the List. 25;

Listing 24: Calculate Fibonacci sequence by recursion

```
def fib(n):
if n <= 1:
    return n
else:
    return fib(n - 1) + fib(n - 2)
```

Listing 25: Calculate Fibonacci sequence using matrix

```
import numpy as np
def fib_matrix(n):
    Matrix = np.matrix([[0, 1], [1, 1]])
    vec = np.array([[0], [1]])
    return np.matmul(Matrix ** n, vec)
```

The average number of votes for Type 1 reused code snippets from the non-accept answer section is 11.25, while Type 2,3 is 5.33. From the number of votes for reused code snippets, we can observe that the higher the vote count, the more likely the code snippets are to be reused. It is worth noting that regardless of which part of the Stack Overflow post code snippets appear in, if it is reused by Type-1, then its average vote number is significantly greater than reused by Type-2,3. This phenomenon shows that code snippets with high votes are of higher quality and are more likely to be reused without modification. In contrast, code snippets with low votes are more likely to need to be modified before they are reused.

Answer to RQ2 (Which part reused?)

• For code reuse from the question section, we find that there are more reused code snippets from the question section than the answer section. We summarize three reasons by manually examining why these code snippets are being reused;
• For code reuse from the accepted answer section, we find that code snippets reused from the accepted answers represent 37.84% of all answers. The average acceptance rate of all the answers in our data is 34.57%. Similarly, we summarize three reasons why the accepted answer's code snippets are reused;
• For code reuse from the non-accepted answer section, we find that there are 1,148 (1.64%) code snippets reused by Type-1 and 17,716 (25.28%) reused by Type 2,3. There are four common reasons why developers reuse code from non-accept answer sections.
In summary, we find that when a post's vote is high, the code snippets are more likely to be Type-1 reused; low votes code snippets are more like to be reused by Type-2,3.

4.3 | RQ3: Why do developers reuse code from Stack Overflow?

Motivation. In RQ1, we find that developers are more prone to reuse code snippets from Stack Overflow. Following that research question, we need to explore the motivation of code reuse on Stack Overflow. By answering this RQ, we are able to provide insights into how developers benefit from reusing code snippets from Stack Overflow.
Methodology. To answer this RQ, we perform a qualitative analysis. Specifically, we ask two authors of this paper to manually inspect the motivations for reusing code snippets from Stack Overflow. To have a confidence level of 99%, 1000 clone pairs were randomly selected from Type-1 and Type-2,3 clone pairs for qualitative analysis. One clone pair includes the related Stack Overflow post and the related Jupyter Notebook file. Second, we ask two authors of this paper to manually analyze the clone code pairs to obtain the following information, including context, description, and the functionality of the code snippets, which are used to categorize the code snippets. If discrepancies exist, another author of this paper is involved in the discussion until a consensus is reached. We use Cohen's Kappa coefficient[21] to measure the agreement of two raters.
Results. After manual classification, we group the reuse reasons into six different categories. The result is shown in Table. 4. We observe that Jupyter Notebook developers often reuse code snippets from Stack Overflow for different purposes. The two common reasons for reuse are adding new features and using APIs. That adds up to 615 and 530 for Type-1 and Type-2,3 code reuse, accounting for more than half of the reasons given. This phenomenon is consistent with our intuition that most developers reuse code snippets from Stack overflow when they have trouble implementing a feature or encountering an unfamiliar API. It is important to note that the import libraries is also part of the reuse reason and cannot be ignored. The most commonly reused code snippets are shown in List 26, which shows numerical calculation, data processing, and the graphing module is the most frequently used by Jupyter Notebook developers in our data collection.

Listing 26: Most reused snippets of imported libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

When comparing the difference between code reuse causes of Type-1 and Type-2,3, we can observe that the cause of module import in Type-1 is 289. The number of Type-2,3 is 1.2 times higher than Type-1, with 334. This is because, unlike Type-2,3 code reuse, Type-1 has strict requirements on the order of statements, and Type-2,3 code reuse is more concerned with semantic similarity. Since the order of importing modules in code snippets does not affect the semantics, there are more causes in Type-2,3 code reuse caused by importing modules. Among the reasons for reusing the initialization data, Type-2,3 code reuse is twice as high as Type-1 code. One possible explanation is that when developers process data, although the data itself is the same, the way of processing is different. Therefore, there is a lot of code considered to reuse the initialization data for Type 2,3 reuse. We use Cohen's Kappa coefficient[21] to measure the agreement between two raters. The result is +0.85, showing good agreement between the two raters.

Answer to RQ3 (Reasons for Code Reuse)

Jupyter Notebook developers reuse StackOverflow code for the following reasons: (1) adding new features; (2) using the APIs; (3) importing libraries; (4) reusing the initialization data; and (5) reusing test cases. The most common reason for reusing StackOverflow code snippets is to add new features.

4.4 | RQ4: What is the quality of code snippets provided in answers on Stack Overflow?

Motivation. According to the results in RQ1, we find that existing Jupyter Notebook developers reuse code snippets from Stack Overflow. Furthermore, existing work[13] finds that there exist low-quality code snippets on Stack Overflow. Thus, reusing code snippets from Stack Overflow can introduce security risks to the Jupyter Notebook. In this RQ, we intend to evaluate the quality of the code snippets provided in answers on Stack Overflow.

Methodology. To answer this RQ, we evaluate the quality of reused code snippets in Jupyter Notebook from four dimensions: (1) reliability and conformance to programming rules, (2) readability, (3) performance, and (4) security. The details of these four dimensions are presented in Table. 5.

Specifically, evaluating the four dimensions for the quality of the code snippets. For reliability and conformance, we leverage *PyLint*[22] for the task. Specifically, PyLint checks errors in the code and looks for code smells in the code. For readability, we leverage *pycodestyle*[23] to evaluate the code snippets reused from Stack Overflow. *Pycodestyle* checks Python code against the style conventions in PEP 8[24], which is the style guide for Python code. For performance, we leverage *PyLint*[22] to check potential performance-related issues. In *PyLint*, it supports different types of checkers to find potential defects in the code snippets. We select all performance related checkers in PyLint and leverage them to detect potential defects in code snippets. PyLint supports 50 checkers, such as Basic checker, Refactoring checker, and Type checker. For each check, it checks several possible defects in the code. We select all related defects from all 50 checkers. As shown in Table. 6, in summary, we obtained 15 issues from these checkers related to performance.

For security, we leverage the *bandit* tool[25] to find common security issues in reused Python snippets.

Results. We scan all Jupyter Notebook code snippets reused from Stack Overflow posts and used the tools noted in the methodology. After scanning 193,248 code snippets, we find 1,528,844 violations. That means, on average, a code snippet has 7.91 code quality violations. The general distribution of violations across the four quality attributes is summarized in Table 7. Note that we

Table 4 Reason for code reuse.

	Frequency	
Category	**Type-1**	**Type-2,3**
Adding new features	368	223
API usage	247	307
Importing libraries	289	334
Data initialization	48	98
Testing	29	26
Other	19	12

Table 5 Description of the code snippet's quality four dimensions

Quality Dimension	Description
Reliability and Conformance	Code snippets should be able to compile and contain no bugs and errors. Furthermore, the code snippets should also conform to accepted programming rules.
Readability	Code snippets should follow standard Python readability conventions to ensure they can be easily understood and maintained.
Performance	Performance and efficiency should be considered when reusing code snippets. For example, has the code snippet offered in an answer improve the performance (e.g., saving processing steps)
Security	Reusing code snippets should consider the security issues of the snippets.

have removed the violations that are unrelated to Jupyter Notebook code snippets. For example, the code snippet shown in the List 27, after scanning with pylint, this code snippet is reported as having a pointless-statement violation. However, the "df_a" variable defined in the code snippet is used later, and we argue that this code snippet is not pointless-statement and removed from the violations. Next, we discuss the violation of four quality attributes in more detail.

Listing 27: Example of violations not related to Jupyter Notebook code snippet

```
raw_data = {
        'first_name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
        'last_name': ['Anderson', 'Ackerman', 'Ali', 'Aoni', 'Atiches']}
df_a = pd.DataFrame(raw_data, columns = ['first_name', 'last_name'])
df_a
```

4.4.1 | Reliability and Conformance

There are 507,298 violations relate to reliability and conformance, accounting for 33.18% of all violations. Of the code snippets analyzed, 144,486 (78.22%) have violations. The average number of violations per snippet is 2.74. The maximum number of violations per snippet is 481, indicating the prevalence of reliability and conformance violations in the reused snippets.

Furthermore, we explore the distribution of the number of violations in code snippets. 123,079 (85.18%) snippets contain 1-5 violations, 11,788 (8.16%) snippets contain 6-10 violations, 4,223 snippets (2.92%) have 11-15 violations, 2267 snippets (1.57%) have 16-20 violations. The number of code snippets with more than 20 violations is 3129 (2.17%).

The top ten reasons for the most common code snippet violation are shown in the Tab 8. The `line-too-long` is the most common reason for violations, with 113,364 (22.35%). PEP 8 suggests for flowing long blocks of text with fewer structural restrictions (docstrings or comments), the line length should be limited to 72 characters.[26]. These ten violations can be divided into 5 categories: violations related to code style (trailing-whitespace, bad-indentation, line-too-long, redefine-outer-name), violations against best practices (consider-using- f-string), wrong import violations (wrong-import-position, wrong-import-order), missing documentation violations (missing-function-docstring, missing-class-docstring) and syntax errors (syntax-error). Violations related to code style are the category containing the most violations, with 321,047 representing 63.28% of all violations.

4.4.2 | Readability

Readability-related violations is the highest of the four quality dimensions, with 1,010,342 (66.09%). Of the code snippets analyzed, 129,075 (69.87%) have violations, with an average of 5.23 violations per snippet, and the snippet with the most readability violations had 1,097 violations. Compared with reliability and conformance, there are more readability violations but fewer code snippets involved.

Table 6 Supported Performance related Violation type in PyLint

Violation Type	Description
self-assigning-variable	A statement in the code that assigns a variable's value to itself. For example, a statement like "foo = foo".
comparison-with-itself	A statement that compares a variable with itself, for example, "if foo > foo: ...."
simplifiable-condition	A conditional statements that can be simplified, for example "if 2 > 1 and a > b: ... ", can be simplified to "if a > b: ....".
condition-evals-to-constant	A conditional statements that can be simplified to constants, a special case of R1726. For example "if 2 > 1:...".
consider-using-in	Checking if a variable is equal to one of many values, it is better to combine those values into a tuple or set and use the "in" keyword instead of comparing variables to values one by one. For example, you should use "a in (1, 2)" instead of "a == 1 or a == 2".
consider-merging-isinstance	When using the `isinstance` function consecutively, it can be merged into one. For example, "isinstance(value, int) or isinstance(value, float)" can be merged to "isinstance(value, (int, float))".
consider-using-generator	If a container is large, using a generator will bring better performance. For example "list([0 for y in list(range(10))])" can be refactored to "list(0 for y in list(range(10))) # using generator"
use-a-generator	It is more efficient to use a generator instead of comprehension when invoke "any", "all", "max", "min", "sum" functions. For example, for "all([randint(-5, 5) > 0 for _ in range(10)])" statement, it is better to use generator like "all(randint(-5, 5) > 0 for _ in range(10))", because it can cut the execution tree and exit directly at the first element that is `False`.
consider-using-join	When concatenating strings, "str.join(sequence)" should be used instead of using for-loop iteration. For example, """.join(["a", "b"])" is more efficient than "s = '' for c in ["a", "b"]: s += a".
unnecessary-dict-index-lookup	When enumeration is performed on the dict, the value and its index can be directly obtained, and there is no need to use the index to obtain the value. For example, you should use "d = {'a': 1, 'b': 2} for key, value in enumerate(d): print(value)" rather than "d = {'a': 1, 'b': 2} for key, value in enumerate(d): print(d[key])".
unnecessary-list-index-lookup	When enumeration is performed on the list, the value and its index can be directly obtained, and there is no need to use the index to obtain the value. For example, "l = ['a', 'b'] for index, c in enumerate(l): print(c)" is better than "l = ['a', 'b'] for index, c in enumerate(l): print(letters[index])".
use-sequence-for-iteration	When iterating over values, sequence types (e.g., lists, tuples, ranges) are more efficient than sets. For example, "l = [1, 2] for i in enumerate(l): print(i)" is efficient than "l = {1, 2} for i in enumerate(l): print(i)"
use-list-literal	When creating a new list, it is faster to use [] instead of list(). Because it avoids an additional function call
consider-using-tuple	Consider using an in-place tuple() instead of list(). Due to optimizations by CPython, there is no performance benefit from it.

Table 7 Summary of violations for the four quality dimensions.

Number of violations	Reliability and conformance	Readability	Performance	Security
Median	2	4	0	0
Average	2.63	5.23	0.02	0.04
Maximum	481	1,097	11	17
Total	507,298	1,010,342	4,165	7,039

Table 8 Most Common Reliability and Conformance Violations.

Violation Type	Number of Violations
line-too-long	113,364
trailing-whitespace	98,522
bad-indentation	97,383
missing-function-docstring	51,681
syntax-error	50,297
consider-using-f-string	25,921
wrong-import-order	14,292
redefined-outer-name	11,778
missing-class-docstring	7,906
wrong-import-position	5,740

Table 9 Most Common Readability Violations.

Violation Type	Number of Violations
missing-whitespace	399,767
line-too-long	113,364
trailing-whitespace	98,522
indentation-not-multiple-of-four	90,573
unexpected-spaces	85,894
blank-line-contains-whitespace	83,157
operators-missing-whitespace	79,315
module-import-should-be-top	50,987
under-indented	37,628
comment-missing-whitespace	36,831

Similarly, the number of violations among code snippets is calculated; 84,116 (65.17%) code snippets have 1 to 5 violations, 22,424 (17.37%) code snippets have 6 to 10 violations, and 7704 (5.97%) code snippets have 11 to 15 violations. There are 3930 (3.04%) code snippets with 16 to 20 violations and 10901 (8.44%) code snippets with more than 20 violations.

To figure out why code snippets violate readability, we compile the top ten most common reasons for violations, as shown in the Table. 9. We observe that missing-whitespace is the most common cause of style violation, with a total of 399,767 (39.57%). Among the top ten causes of violations, Whitespace caused the most violations (missing-whitespace, trailing-whitespace, unexpected-spaces, blank-line-contains-whitespace, missing- operators-whitespace, missing-comment-whitespace) with 783,486 (77.54%); Line length related violations (line-too-long) with 113,364 (11.22%); Indentation caused violations are indentation-not-multiple-of-four and under-indented, which add up to 128,201 (12.69%); violations related to import (module-import-should-be-top) have a 50987(5.04%).

Table 10 Number of performance violations.

Violation Type	Number of Violations
consider-using-tuple	1,621
self-assigning-variable	1,357
use-list-literal	811
consider-using-generator	393
consider-using-in	381
unnecessary-list-index-lookup	56
consider-merging-isinstance	34
comparison-with-itself	14
condition-evals-to-constant	12
simplifiable-condition	2
consider-using-join	2
unnecessary-dict-index-lookup	2
use-sequence-for-iteration	2
use-a-generator	0

4.4.3 | Performance

Pylint finds 4,687 performance violations, with an average of one violation per 0.02 code snippets. Performance violation is the least of our four quality dimensions, showing that Jupyter Notebook developers care more about the performance of reused code than other quality dimensions, such as code style. Additionally, there are 2,182 (0.14%) code snippets with performance violations, the largest number of violations being 11. There are 2050 (93.95%) code snippets with a violation count between 1 and 5, 130 (5.96%) code snippets with a violation count between 6 and 10, and only 2 (0.09%) code snippets with a violation count greater than 10.

We count the number of all causes of performance violations, which are shown in Table. 10. The most performance violations we observed (1,621 or 34.59%) are due to consider-using-tuple violations, with self-assigning-variable in second place with 1357 (28.95%) and use-list-literal with 811 (17.3%). The rest of the violations are below 10%. We recommend that Jupyter Notebook developers make more efficient use of built-in *list* types when reusing code snippets and avoid assigning a variable's value to itself.

4.4.4 | Security

We find 7,039 (0.46%) security violations using bandit, an average of one security violation for every 0.04 code snippets. The remaining 27 snippets (0.65%) had more than 5 violations.

We count the top ten most common reasons for security violations, which are shown in the Table. 11. Unlike the previous three quality categories, the bandit has graded each violation by severity, and it can be observed that among the most common reasons for violations, 5171 have a low severity, accounting for 73.46% of all violations, while the medium has 1254 or 17.82%. Violations with high severity are not among the top ten common violations, with 66 accounting for 0.94% of all security violations. These data suggest that the problem of harmful reuse of code in Jupyter Notebook is not severe. The top ten most common reasons for violation can be divided into 3 categories, invokes blacklists function(use-standard-pseudo-random-generators, unexpected-url-open-parameter, use-insecure-function), import blacklists module(use-pickle -module, use-subprocess-module, use-xml-parse-untrusted-data) and misc violation (assert-used, insecure-use-temp-file, try-except-pass-found, possible-hardcoded-password). The highest number of misc violations is 2740, invokes blacklists function is 2036, and the lowest is the import blacklists module is 1649.

Table 11 Most Common Security Violations.

Violation Type	Number of Violations	Severity
assert-used	1705	low
use-pickle-module	1464	low
use-standard-pseudo-random-generators	1130	low
unexpected-url-open-parameter	545	medium
use-insecure-function	361	medium
insecure-use-temp-file	348	medium
possible-hardcoded-password	347	low
try-except-pass-found	340	low
use-subprocess-module	103	low
use-xml-parse-untrusted-data	82	low

Table 12 The number of code violations in different reuse Types.

Quality Dimension	Type-1 Violations	Violations Per Snippet	Type-2,3 Violations	Violations Per Snippet
Reliability	38,908	1.98	490,629	2.65
Readability	63,608	3.24	978,559	5.30
Performance	237	0.01	4,034	0.02
Security	745	0.04	6,662	0.04

4.4.5 | Type-1 and Type-2,3 reused code snippets' quality

The result for the number of violations for different code reuse in four quality dimensions is shown in Table. 12. We find that Type-1 reuse code snippets have 38,908 reliability violations, with an average of 1.68 violations per code snippet.

Type-2,3 reuse code snippets have 490,629 reliability violations. There are 2.65 reliability violations per code snippet. On average, each Type-2,3 reuse code snippet has 33.84% more reliability violations than Type-1 reuse code snippet. In the readability quality dimension, there are 63,608 violations for Type-1 code reuse snippets and 978,559 violations for Type-2,3 code reuse snippets.

On average, Type-2,3 reused code snippets contain 2.06 more readability violations than Type-1 reused code snippets. For the number of average performance violations, Type-2,3 reused code snippets are 0.01 larger than that in Type-1 code snippets. The Type-1 and Type-2,3 reused code snippet's average violations are the same on the security dimension. We observe that Type-2,3 reused code snippets are more likely to have reliability and readability violations. On the performance and security dimensions, there is no significant difference in the number of average violations per code snippet for different types of code clones.

Answer to RQ4 (Code Quality)

In this RQ, we measure the quality of the reused code in Jupyter Notebook from four dimensions. On average, a code snippet has 7.91 code quality violations. We find that reused code snippets have the highest number of violations on readability with an average of 3.39 violations per code snippet. In the reliability and conformance quality dimension, reused code snippets have the second-highest number of violations with an average of 2.74 violations per code snippet. There are only 0.04 Security violations and 0.02 Performance violations per code snippet. We find that Type-2,3 reused code snippets are more likely to have reliability and readability violations.

4.5 | RQ5: Who reuses code from Stack Overflow?

Motivation. Previous research[14] demonstrated that reusing code can have negative effects on building software. This always associates with less experienced developers. For example, developers with less experience may adopt the Type-1 clone. Here, we

intend to examine who reuses Stack Overflow code among developers. Furthermore, we also aim to investigate how experienced developers cope with code reuse.

Methodology. To answer this RQ, we adopt the same methodology presented in[3]. Specifically, we first measure the experience of developers. Similar to previous works[27,28], we measure the number of commits from the start of the project to the time of code reuse. This helps us evaluate the experience of developers. One developer commits more often than another developer, he/she is more familiar with the project and is more experience. We normalize the experience of a developer as a percentage of the total number of commits he/she made to the project.

Results. We present the distribution of experience in projects by authors who reuse Stack Overflow code snippets, as shown in Figure 5. We observe the majority of developers who perform code reuse have an experience of 1 (81.72% and 77.17% for Type-1 and Type-2,3 code reuse, respectively), while there is also a partial distribution of data in the 0.4 to 0.6 range (4.25% and 4.48% for Type-1 and Type-2,3 code reuse respectively), and the developer's experience doesn't correlate significantly with the way of reuse. By inspecting the total number of developers in Jupyter Notebook projects where code reuse exists, we find the following possible reason for the distribution of developer experience with code reuse. First, many Jupyter Notebook projects have only one developer, which results in a large number of reusers with experience of 1. Second, in projects with more than one developer, there are also more projects with only two developers, so the developer experience tends to be distributed around 0.5.

For more detail, We obtain 149,541 Jupyter Notebook code snippets with developers' experience of 1. The number of one-author code snippets accounted for 98.14% of them. We also count 43,707 code snippets with developer experience of less than 1, and projects with two developers accounted for 19.85% of them. Meanwhile, there are 5,850 code snippets in which the developer is two, and the developer experience is in the 0.4 to 0.6 range. This represents 13.38% of the clone pairs with developer experience of less than 1 and 67.41% of the code snippets with developer experience in the range of 0.4 to 0.6.

To focus on the distribution of developer experience in medium and large projects, we filter out small Jupyter Notebook projects by the total number of developers greater than 10. The results are shown in Fig. 6. We find that less experienced developers (less than 0.1 experience) are more likely to reuse code with the Type-1 clone, accounting for 83.28% of all Type-1 code clones. The proportion of developers with Type 2,3 code reuse experience below 0.1 is 52.46%. This result suggests that developers with less development experience on medium or large projects tend to resort to Type-1 code cloning for code reuse. Experienced developers turn to choose Type-2,3 code clone.

Answer to RQ5 (Who Reuse Code?)

When it comes to code reuse, most Jupyter Notebook developers have an experience of 1, and there is no significant difference in the reuse method because most of the Jupyter Notebook projects we collected are small personal projects. Furthermore, we find that the less experienced developers intend to reuse code using Type-1 code clone. In contrast, more experienced developers intend to reuse code with Type-2,3 code clones.

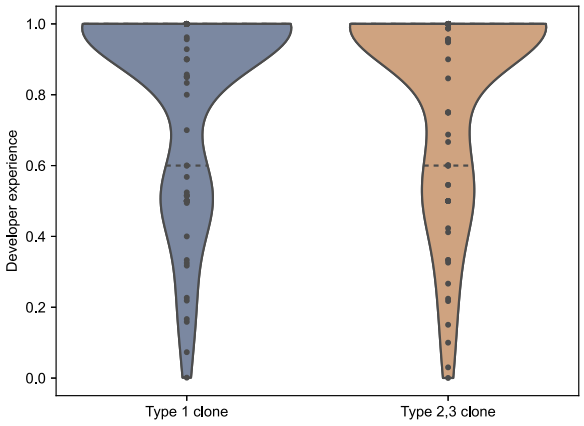

Figure 5 Distribution of the developers' experience for developers who reuse code from StackOverflow

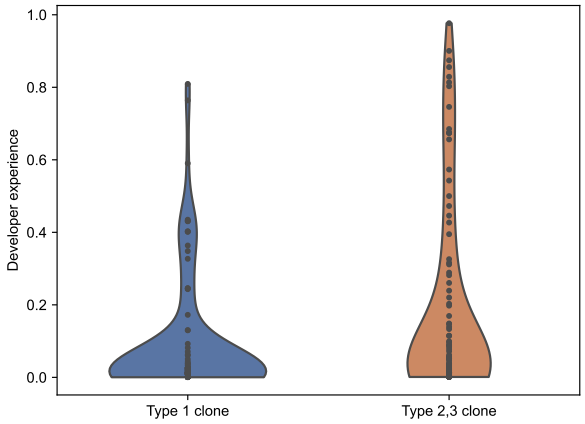

Figure 6 Distribution of the developers' experience on teamwork project

5 | LESSONS LEARNT AND THREATS TO VALIDITY

In this section, we summarize our findings from this study and provide suggestions for developers. Then we present the threats to the validity of our work.

5.1 | Lessons Learnt

We find that 59,942 of the 188,302 (31.83%) Github repositories we collected contain code reuse. This shows that code reuse is widespread in Jupyter Notebook. To make it easier for Jupyter Notebook programmers to reuse code on Stack Overflow, we have summarised our findings in the following three aspects.

(1) Reasons for reuse code Based on our manual analysis, we find that adding new features, importing modules ,and using APIs are the three most common reasons for reuse. This suggests that developers turn to Stack Overflow for help when they are faced with a need for knowledge they are not familiar with. We also find that developers do not always reuse code snippets from the accepted answer, as the question section or unaccepted answers may have more efficient implementations or code that is easier to reuse. So we recommend that when looking for code to reuse on Stack Overflow, developers should not focus too much on the accepted answer, as there may be more suitable code in the question or unaccepted answer section.

(2) Reuse code quality Our experiments on code quality show that for Jupyter Notebook code snippets in reused pairs. On average, there are 7.91 code quality violations per code snippet. This indicates that code quality violations are common in Jupyter Notebook reuse code snippets. After reusing a Stack Overflow code snippet, we recommend that developers use code quality checkers to check the code. Avoid the situation where reusing external code degrades the quality of the program.

(3) Code reuser's experience: We find that more experienced developers tend adopt Type-2,3 code clone in medium and large projects. Our experiments have shown that Type-2,3 code snippets are more likely to have reliability and readability violations. Thus, for large-size projects, developers should carefully inspect code reliability and readability violations for reused type 2,3 code snippets.

5.2 | Threats to Validity

Threats to internal validity. We use SourcererCC to detect Type-2,3 code clones. However, the accuracy of the SourcererCC detection limits our identification of Type-2,3 clone pairs. In some cases, SourcerCC detected code snippet pairs that did not have Type-2,3 clones as code clones. Furthermore, SourcerCC may miss code clone pairs for Type-2,3. To mitigate this problem, we

manually sampled several code snippet pairs reported by SourcerCC and code snippets not reported as clones by SourcerCC. In all cases, the reported clones were actual clones, and there was no underreporting.

When looking at why developers reuse Stack Overflow, we use manual classification as this task is hard or even impossible to automate. To avoid errors associated with manual classification, we ask two authors to classify independently and then discuss the results to reach a consensus on the classification. We use Cohen's Kappa to calculate an inter-rater agreement of 0.85, which indicates a high degree of agreement between those two.

Threats to external validity. There is a possibility that the results of this study are not generalize.

• First, we focus on reusing code snippets from Stack Overflow, which is only one of many Q&A websites, so there is a possibility that the results cannot be generalized to all Q&A websites.

• Second, although Jupyter Notebook Note is most commonly used as a tool for analyzing data[10,15], there is a possibility that our findings may not be generalizable to other applications (e.g. Polynote or R Notebook).

6 | RELATED WORK

We discuss the related work from three aspects: studies on StackOverflow, studies on Jupyter Notebook, and studies related to code reuse and clone detection.

Works related to StackOverflow. An et al.[29] studied whether app developers respect license terms when reusing code from StackOverflow posts by inspecting 399 Android apps. Uddin et al.[30] proposed a framework named Opiner to mine API usage patterns from StackOverflow posts, which can be used to assist developers in programming tasks and reusing code segments from StackOverflow posts. Ponzanelli et al.[31] developed an Eclipse plugin to generate queries for StackOverflow according to the programming context in the IDE. Some works care about the code quality of code segments on StackOverflow posts. Specifically, Wu et al.[1] discussed how developers utilize code segments from StackOverflow. They conduct an exploratory study on 289 files from 182 open-source projects, which contain source code that has an explicit reference to a StackOverflow post. They found the top 3 barriers that make it difficult for developers to reuse code from StackOverflow. Ragkhitwetsagul et al.[8] conducted a large-scale empirical study for online code clones between Stack Overflow and 111 Java open source projects to understand: (1) code clone practices; (2) code clone patterns; (3) out-dated code clones; and (4) software licensing violations. Meldrum et al.[13] conducted a large-scale study on code snippet quality for code snippets from StackOverflow. Rahman et al.[27] discussed the insecure code practices for Python code on StackOverflow. Other works focus on the code smell detection for code segments on StackOverflow[28,32,33].

Works related to Jupyter Notebook Pimentel et al. conducted a large-scale study on the quality and reproducibility of Jupyter notebooks[15]. They studied 1.4 million notebooks from GitHub, showing that only 24.11% of Jupyter notebooks can be executed without exception, and only around 4% of notebooks can be reproduced with the same results. Koenzen et al.[10] explored the way of code duplications in Jupyter notebooks and identified the potential barriers to code reuse. Besides, Wang et al.[34] first studied whether existing notebooks can be executed successfully (i.e., reproducibility). Then, they proposed a prototype named Osiris, which takes a notebook as an input and outputs the possible execution schemes to reproduce the notebook. Wang et al.[35] developed SnifferDog to restore the execution environments for executing Jupyter notebooks. Specifically, SnifferDog first collects the APIs of Python packages to build the database and then analyzes the notebooks to determine the candidate packages and versions. Wang et al. 's work[5] conducted a preliminary study on code quality for Jupyter notebooks. They found that the existing notes are with poor quality codes, which requires quality control on Jupyter notebooks. Other work is developing tools to improve the readability or efficiency of Jupyter Notebook[36,37].

Works related to clone detection and reuse. Roy et al.[38] developed NiCAD, which leverages a text-based approach to detect Type-1 and Type-2 code clones. Saini et al.[39] developed a code clone tool Oreo by leveraging machine learning, information retrieval, and software metrics. Lopes et al.[40] studied code clones on a corpus of 4.5 million non-fork projects on GitHub. These projects represent over 428 million lines of code in Java, C++, Python, and JavaScript. Code reuse has been widely studied[41,42,3,43,44,45,46,47,48,49,50]. Specifically, Ossher et al.[51] developed a line-level code clone detection tool, which is used upon the Sourcerer Repository. Among over 13,000 projects in Sourcerer Repository, they found that over 10% of files are cloned. Gharehyazie et al.[41] detected cross-project clones among 8,599 projects on GitHub for Type-1 and 2 clone. Yang et al.[42] presented a large-scale study on 909k non-fork Python projects on GitHub, and 1.9 million Python snippets captured in Stack Overflow to learn the code clone on these projects. Abdalkareem et al.[3] studied the code reuse practice by studying how developers reuse code snippets from Stack Overflow when building Android apps. Their study provides the potential impact of code reuse from

StackOverflow on building apps. Ahmad et al.[43] measured the impact of the code snippets from Stack Overflow in GitHub projects during the evolution, including prior to the addition of the snippet, immediately after the addition of the snippet and a longer time after the addition of the snippet. They found that almost 70% of the cases where the copied snippet affected the cohesion of the project.

7 | CONCLUSION

In this paper, we investigate the reuse of code from Stack Overflow in Jupyter Notebook in Stack Overflow. We find that 3.26% of code snippets are related to code reuse. Then, by analyzing the source of cloned code snippets, we find that code snippets come from the question more than the answer section. In addition, we summarise the three most common reasons for code reuse: adding new features, using APIs, and importing modules. By examining the quality of the reused code snippets, we find an average of 7.91 violations per snippet, with readability violations being the most frequent, accounting for 66.09% of the total violations. Finally, we learn that in medium to large projects, less experienced developers are more likely to adopt Type 1 code clones than more experienced developers. Our research provides insight into code reuse on Stack Overflow in Jupyter Notebook.

References

1. Wu Y, Wang S, Bezemer CP, Inoue K. How do developers utilize source code from stack overflow?. *Empirical Software Engineering* 2019; 24(2): 637–673.
2. Gharehyazie M, Ray B, Filkov V. Some from here, some from there: Cross-project code reuse in github. In: *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*; 2017: 291–301.
3. Abdalkareem R, Shihab E, Rilling J. On code reuse from stackoverflow: An exploratory study on android apps. *Information and Software Technology* 2017; 88: 148–158.
4. Kery MB, Radensky M, Arya M, John BE, Myers BA. The Story in the Notebook: Exploratory Data Science Using a Literate Programming Tool. In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*CHI '18. Association for Computing Machinery; 2018; New York, NY, USA: 1–11
5. Wang J, Li L, Zeller A. Better code, better sharing: on the need of analyzing jupyter notebooks. In: *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: New Ideas and Emerging Results*; 2020: 53–56.
6. Jupyter Notebook Documentation. https://jupyter-notebook.readthedocs.io/en/latest/notebook.html; 2015.
7. Zhang T, Upadhyaya G, Reinhardt A, Rajan H, Kim M. Are code examples on an online q&a forum reliable?: a study of api misuse on stack overflow. In: *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*; 2018: 886–896.
8. Ragkhitwetsagul C, Krinke J, Paixao M, Bianco G, Oliveto R. Toxic code snippets on stack overflow. *IEEE Transactions on Software Engineering* 2019; 47(3): 560–581.
9. Fischer F, Böttinger K, Xiao H, et al. Stack overflow considered harmful? the impact of copy&paste on android application security. In: *2017 IEEE Symposium on Security and Privacy (SP)*; 2017: 121–136.
10. Koenzen AP, Ernst NA, Storey MAD. Code duplication and reuse in Jupyter notebooks. In: *2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*; 2020: 1–9.
11. Jupyter. https://jupyter.org/; 2023.
12. Stack Overflow. https://stackoverflow.com/; 2023.
13. Meldrum S, Licorish SA, Owen CA, Savarimuthu BTR. Understanding stack overflow code quality: A recommendation of caution. *Science of Computer Programming* 2020; 199: 102516.

14. Barzilay O, Urquhart C. Understanding reuse of software examples: A case study of prejudice in a community of practice. *Information and Software Technology* 2014; 56(12): 1613-1628.

15. Pimentel JF, Murta L, Braganholo V, Freire J. A large-scale study about quality and reproducibility of jupyter notebooks. In: *2019 IEEE/ACM 16th international conference on mining software repositories (MSR)*; 2019: 507–517.

16. Baltes S, Dumani L, Treude C, Diehl S. Sotorrent: Reconstructing and analyzing the evolution of stack overflow posts. In: *Proceedings of the 15th international conference on mining software repositories*; 2018: 319–330.

17. Nasehi SM, Sillito J, Maurer F, Burns C. What makes a good code example?: A study of programming Q&A in StackOverflow. In: *2012 28th IEEE International Conference on Software Maintenance (ICSM)*; 2012: 25–34.

18. Roy CK, Cordy JR. A survey on software clone detection research. *Queen's School of computing TR* 2007; 541(115): 64–68.

19. Sajnani H, Saini V, Svajlenko J, Roy CK, Lopes CV. Sourcerercc: Scaling code clone detection to big-code. In: *Proceedings of the 38th International Conference on Software Engineering*; 2016: 1157–1168.

20. Google . Protocol Buffers | Google Developers. https://developers.google.com/protocol-buffers; 2023.

21. Cohen J. A coefficient of agreement for nominal scales. *Educational and psychological measurement* 1960; 20(1): 37–46.

22. PyLint . Pylint:Static analyzer for Python 2 and 3. https://pylint.pycqa.org/en/latest/; 2023.

23. pycodestyle . pycodestyle:Python style guide checker. https://github.com/PyCQA/pycodestyle; 2022.

24. PEP . PEP 08:Style Guide for Python Code. https://peps.python.org/pep-0008/; 2013.

25. bandit . Bandit:find common security issues in Python code.. https://github.com/PyCQA/bandit; 2023.

26. Python . Style Guide for Python Code. https://peps.python.org/pep-0008/#maximum-line-length; 2013.

27. Rahman A, Farhana E, Imtiaz N. Snakes in paradise?: Insecure python-related coding practices in stack overflow. In: *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*; 2019: 200–204.

28. Tahir A, Yamashita A, Licorish S, Dietrich J, Counsell S. Can you tell me if it smells? a study on how developers discuss code smells and anti-patterns in stack overflow. In: *Proceedings of EASE*; 2018: 68–78.

29. An L, Mlouki O, Khomh F, Antoniol G. Stack overflow: a code laundering platform?. In: *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*; 2017: 283–293.

30. Uddin G, Khomh F, Roy CK. Mining API usage scenarios from stack overflow. *Information and Software Technology* 2020; 122: 106277.

31. Ponzanelli L, Bacchelli A, Lanza M. Seahawk: Stack overflow in the ide. In: *2013 35th International Conference on Software Engineering (ICSE)*; 2013: 1295–1298.

32. Shcherban S, Liang P, Tahir A, Li X. Automatic identification of code smell discussions on stack overflow: A preliminary investigation. In: *Proceedings of the 14th ACM/IEEE international symposium on empirical software engineering and measurement (ESEM)*; 2020: 1–6.

33. Duijn M, Kucera A, Bacchelli A. Quality questions need quality code: Classifying code fragments on stack overflow. In: *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*; 2015: 410–413.

34. Wang J, Kuo Ty, Li L, Zeller A. Assessing and restoring reproducibility of Jupyter notebooks. In: *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*; 2020: 138–149.

35. Wang J, Li L, Zeller A. Restoring execution environments of jupyter notebooks. In: *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*; 2021: 1622–1633.

36. Titov S, Golubev Y, Bryksin T. ReSplit: Improving the Structure of Jupyter Notebooks by Re-Splitting Their Cells. In: *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*; 2022: 492-496.

37. Patra J, Pradel M. Nalin: Learning from Runtime Behavior to Find Name-Value Inconsistencies in Jupyter Notebooks. In: *Proceedings of the 44th International Conference on Software Engineering (ICSE)*; 2022: 1469–1481.

38. Roy CK, Cordy JR. NICAD: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization. In: *2008 16th IEEE international conference on program comprehension*; 2008: 172–181.

39. Saini V, Farmahinifarahani F, Lu Y, Baldi P, Lopes CV. Oreo: Detection of clones in the twilight zone. In: *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*; 2018: 354–365.

40. Lopes CV, Maj P, Martins P, et al. DéjàVu: a map of code duplicates on GitHub. *Proceedings of the ACM on Programming Languages* 2017; 1(OOPSLA): 1–28.

41. Gharehyazie M, Ray B, Keshani M, Zavosht MS, Heydarnoori A, Filkov V. Cross-project code clones in github. *Empirical Software Engineering* 2019; 24(3): 1538–1573.

42. Yang D, Martins P, Saini V, Lopes C. Stack overflow in github: any snippets there?. In: *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*; 2017: 280–290.

43. Ahmad M, Cinnéide MO. Impact of stack overflow code snippets on software cohesion: a preliminary study. In: *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*; 2019: 250–254.

44. Haefliger S, Von Krogh G, Spaeth S. Code reuse in open source software. *Management science* 2008; 54(1): 180–193.

45. Chen X, Liao P, Zhang Y, Huang Y, Zheng Z. Understanding Code Reuse in Smart Contracts. In: *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*; 2021: 470-479.

46. Feitosa D, Ampatzoglou A, Gkortzis A, Bibi S, Chatzigeorgiou A. Code reuse in practice: Benefiting or harming technical debt. *Journal of Systems and Software* 2020; 167: 110618.

47. Yang D, Hussain A, Lopes CV. From query to usable code: an analysis of stack overflow code snippets. In: *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*; 2016: 391–401.

48. Diamantopoulos T, Symeonidis A. Employing Source Code Information to Improve Question-Answering in Stack Overflow. In: *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*; 2015: 454-457.

49. Fischer F, Xiao H, Kao CY, et al. Stack Overflow Considered Helpful! Deep Learning Security Nudges Towards Stronger Cryptography. In: *28th USENIX Security Symposium (USENIX Security 19)*; 2019: 339–356.

50. Diamantopoulos T, Sifaki MI, Symeonidis A. Towards mining answer edits to extract evolution patterns in Stack Overflow. In: *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*; 2019: 215–219.

51. Ossher J, Sajnani H, Lopes C. File cloning in open source java projects: The good, the bad, and the ugly. In: *2011 27th IEEE International Conference on Software Maintenance (ICSM)*; 2011: 283–292.

52. Knuth DE. Literate Programming. *The Computer Journal* 1984; 27(2): 97-111.

53. Ponzanelli L, Bavota G, Di Penta M, Oliveto R, Lanza M. Mining stackoverflow to turn the ide into a self-confident programming prompter. In: *Proceedings of the 11th working conference on mining software repositories*; 2014: 102–111.

54. Zhu C, Saha RK, Prasad MR, Khurshid S. Restoring the Executability of Jupyter Notebooks by Automatic Upgrade of Deprecated APIs. In: *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*; 2021: 240–252.

55. Grotov K, Titov S, Sotnikov V, Golubev Y, Bryksin T. A Large-Scale Comparison of Python Code in Jupyter Notebooks and Scripts. *arXiv preprint arXiv:2203.16718* 2022.

56. Robinson D, Ernst NA, Vargas EL, Storey MAD. Error Identification Strategies for Python Jupyter Notebooks. *arXiv preprint arXiv:2203.16653* 2022.

57. Baltes S, Diehl S. Usage and attribution of Stack Overflow code snippets in GitHub projects. *Empirical Software Engineering* 2019; 24(3): 1259–1295.

58. Reinhardt A, Zhang T, Mathur M, Kim M. Augmenting stack overflow with API usage patterns mined from GitHub. In: *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*; 2018: 880–883.

59. Kamiya T, Kusumoto S, Inoue K. CCFinder: A multilinguistic token-based code clone detection system for large scale source code. *IEEE transactions on software engineering* 2002; 28(7): 654–670.

60. Jiang L, Misherghi G, Su Z, Glondu S. Deckard: Scalable and accurate tree-based detection of code clones. In: *29th International Conference on Software Engineering (ICSE'07)*; 2007: 96–105.

61. Komondoor R, Horwitz S. Using slicing to identify duplication in source code. In: *International static analysis symposium*; 2001: 40–56.

62. Frakes WB, Kang K. Software reuse research: Status and future. *IEEE transactions on Software Engineering* 2005; 31(7): 529–536.

63. Mohagheghi P, Conradi R. Quality, productivity and economic benefits of software reuse: a review of industrial studies. *Empirical Software Engineering* 2007; 12(5): 471–516.

64. Pearce H, Ahmad B, Tan B, Dolan-Gavitt B, Karri R. Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions. In: *2022 IEEE Symposium on Security and Privacy (SP)*; 2022: 754–768.

65. GitHub Copilot · Your AI pair programmer. https://copilot.github.com/; 2023.

66. Nguyen HA, Nguyen AT, Nguyen TT, Nguyen TN, Rajan H. A study of repetitiveness of code changes in software evolution. In: *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*; 2013: 180–190.

67. Ray B, Kim M. A case study of cross-system porting in forked projects. In: *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*; 2012: 1–11.

68. SOTorrent Dataset. https://zenodo.org/record/4415593; 2021.

69. Begel A, Zimmermann T. Analyze This! 145 Questions for Data Scientists in Software Engineering. In: *Proceedings of the 36th International Conference on Software Engineering*ICSE 2014. ; 2014; New York, NY, USA: 12–23

70. Huang Y, Xu F, Zhou H, Chen X, Zhou X, Wang T. Towards Exploring the Code Reuse from Stack Overflow during Software Development. *arXiv preprint arXiv:2204.12711* 2022.

71. Zhang H, Wang S, Chen TH, Zou Y, Hassan AE. An empirical study of obsolete answers on Stack Overflow. *IEEE Transactions on Software Engineering* 2019; 47(4): 850–862.

72. Tekieh R. *Understanding How Developers Reuse Stack Overflow Code in Their GitHub Projects*. PhD thesis. Carleton University, Ottawa, Canada; 2021.

How to cite this article: Mingke Y, Yuming Z, Bixin L, Yutian T, (2023), On Code Reuse from StackOverflow: An Exploratory Study on Jupyter Notebook., *Softw Pract Exper.2023*.