

Are Human Rules Necessary? Generating Reusable APIs with CoT Reasoning and In-Context Learning

YUBO MAI, The State Key Laboratory of Blockchain and Data Security, Zhejiang University, China

ZHIPENG GAO*, Shanghai Institute for Advanced Study of Zhejiang University, China

XING HU, The State Key Laboratory of Blockchain and Data Security, Zhejiang University, China

LINGFENG BAO, The State Key Laboratory of Blockchain and Data Security, Zhejiang University, China

YU LIU, The State Key Laboratory of Blockchain and Data Security, Zhejiang University, China

JIANLING SUN, The State Key Laboratory of Blockchain and Data Security, Zhejiang University, China

Proceedings of the ACM on Software Engineering, Volume 1, Issue FSE
Article No.: 104, Pages 2355 - 2377

発表者：松尾 和輝

コードの再利用

- 開発者らが技術的な問題に直面した際は、ソフトウェアQ&Aサイトに解決策を求める傾向にある

復元抽出と非復元抽出を切り替えて実行できる関数を定義したい

質問日 3 日前 更新 今日 閲覧数 84件

▲ targetに格納されている値について、population内からこれに近似した値とindexを出力したいです。

0 抽出方法は復元抽出と非復元で、抽出関数の中で方法を切り替えたいと思っています。

▼ 以下のコードは、復元抽出だけ可能です。

1件の回答

並べ替え: スコア順 (デフォルト)

最終的にPandasをループで回さず、numpy化して処理しました。

```
def match(target, population, method='復元'):
    result_df = pd.DataFrame()

    if method == '復元':
        neigh = NearestNeighbors(n_neighbors=1, metric='euclidean')
        neigh.fit(np.array(population).reshape(-1, 1))

        # ユークリッド距離と探索先のインデックス
        distances, indexes = neigh.kneighbors(np.array(target).reshape(-1, 1))

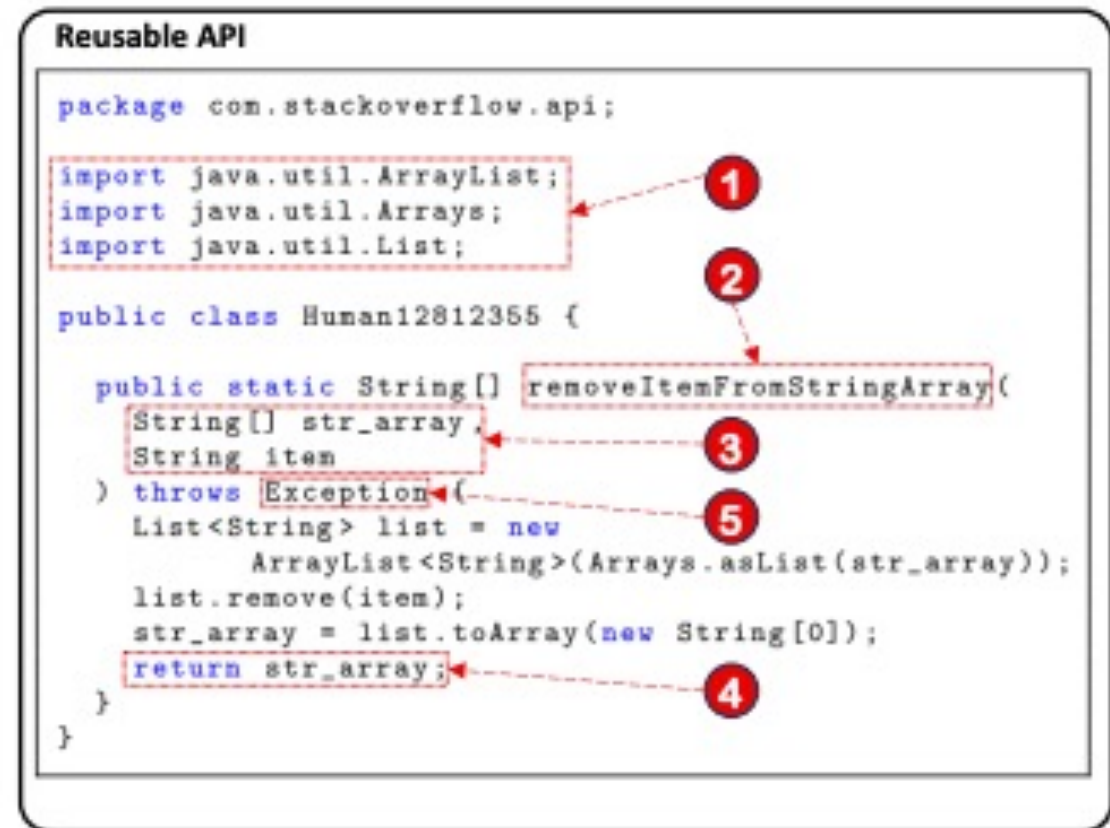
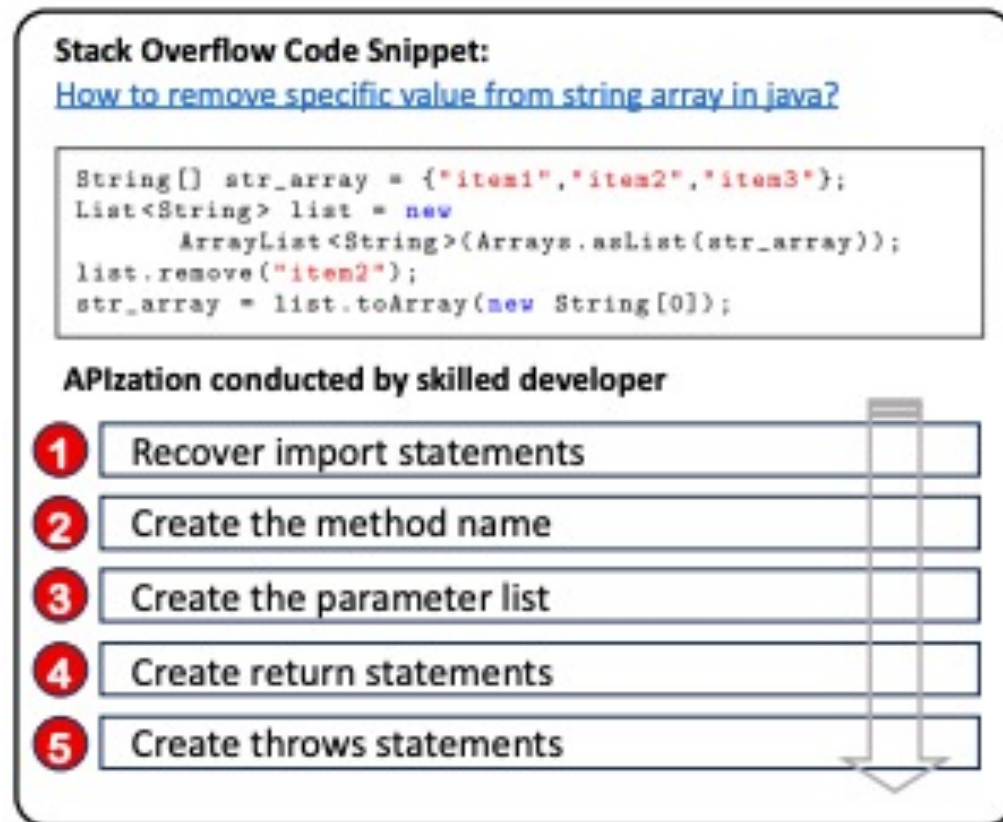
        result_df = pd.DataFrame({'match_indexes': indexes.reshape(-1),
                                  'match_distances': distances.reshape(-1)})
        result_df = pd.merge(target, result_df, left_index=True, right_index=True)
```

Stack Overflow (SO) における質問 (左) と回答 (右)

- 過去の質問が蓄積されており、開発者が直面した課題に類似する質問から、コードを再利用して課題を解決している

コードの再利用における課題

- SO上のコードは直接再利用できない場合が多い[45, 51, 64]
 - 再利用の目的で回答されておらず, Import文などが欠落しているため[32]



- コードを自動でAPI化（一般化）する研究の実施

[32] Seyed Mehdi Nasehi, Jonathan Sillito, Frank Maurer, and Chris Burns. 2012. What makes a good code example?: A study of programming Q&A in StackOverflow. In 2012 28th IEEE International Conference on Software Maintenance (ICSM). IEEE, 25–34.

[45] Siddharth Subramanian and Reid Holmes. 2013. Making sense of online code snippets. In 2013 10th Working Conference on Mining Software Repositories (MSR). IEEE, 85–88.

[51] Valerio Terragni, Yepang Liu, and Shing-Chi Cheung. 2016. CSNIPPEX: automated synthesis of compilable code snippets from Q&A sites. In Proceedings of the 25th international symposium on software testing and analysis. 118–129.

[64] Di Yang, Aftab Hussain, and Cristina Videira Lopes. 2016. From query to usable code: an analysis of stack overflow code snippets. In Proceedings of the 13th International Conference on Mining Software Repositories. 391–402.

先行研究

- APIzator^[52]
 - ルールベース手法
 - メソッド名生成にPOSタグ付けを採用
 - 質問から適切な単語を抽出しメソッド名にしている
- 課題
 - 生成されるAPIの90%は誤解を招くメソッド名である
 - Javaのコードでしか利用できない

Motivating Example 1

● [How to get the last Sunday before current date?](#)

```
public class APIzator12783806 {  
    public static int get() throws Exception {  
        Calendar cal = Calendar.getInstance();  
        cal.add(Calendar.DAY_OF_WEEK,  
            -(cal.get(Calendar.DAY_OF_WEEK) - 1));  
        return cal.get(Calendar.DATE);  
    }  
}
```

Motivating Example 2

● [How to remove leading zeros from alphanumeric text?](#)

```
public class APIzator2800839 {  
    public static void removeZero(String[] in) throws  
        Exception {  
        for (String s : in) {  
            System.out.println("[ " +  
                s.replaceFirst("^0+(?!$)", "") + " ]");  
        }  
    }  
}
```

[52] Valerio Terragni and Pasquale Salza. 2021. APIzation: Generating reusable APIs from StackOverflow code snippets. In 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, 542–554.

本研究の動機・目的

- 動機

- ChatGPT^[34]は人間の質問の意図を捉え、適切な回答を出力することが可能
- ChatGPTを用いることで、コードの内容を捉えた適切なメソッド名を含むAPIを出力することが可能なのでは？

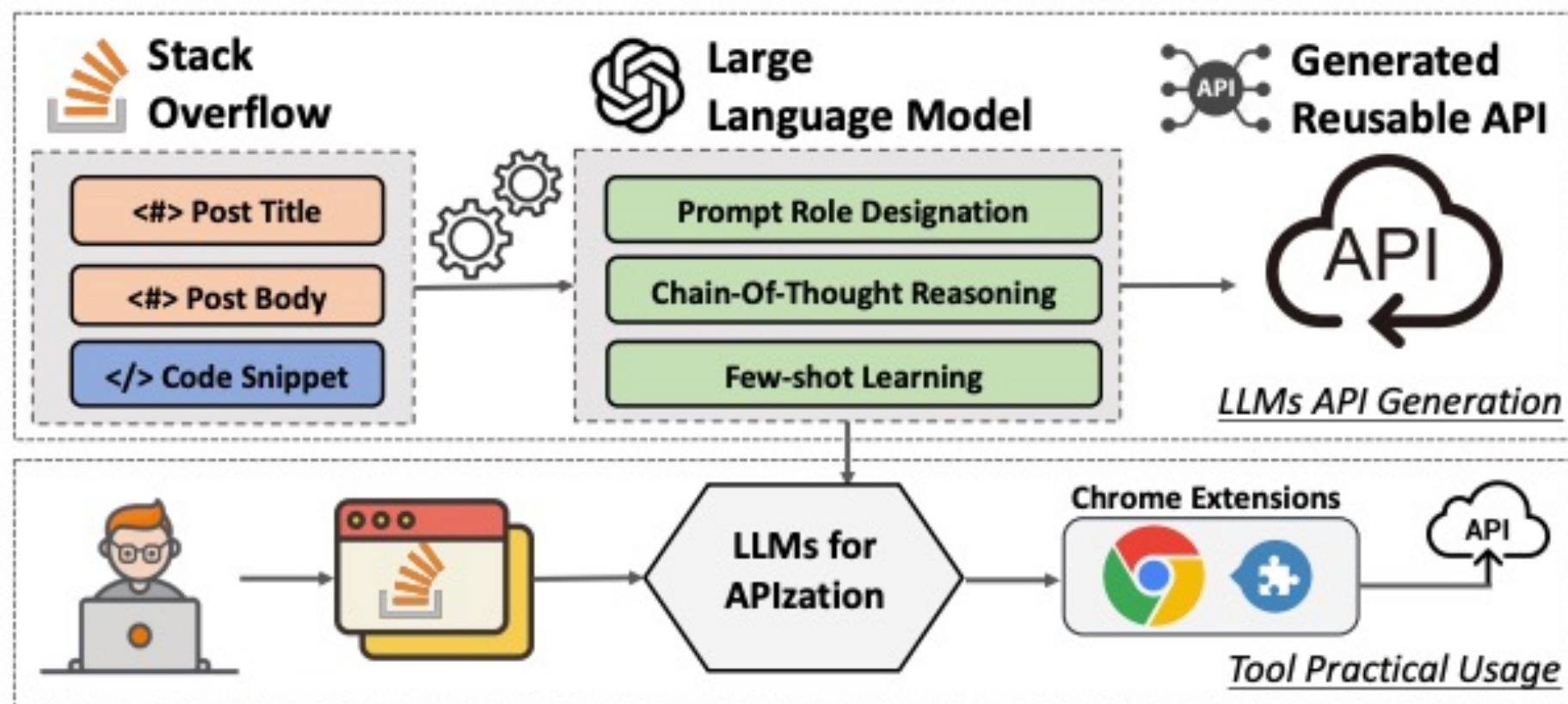
- 目的

- プロンプトエンジニアリングを活用した新しいAPI化手法「Code2API」の提案

[34] OpenAI. 2022. Introducing ChatGPT. OpenAI Blog (November 2022)

Code2API | ワークフロー

- 入力 | コードと質問タイトル, 質問内容
- 出力 | API化されたコード



- LLMには, GPT-3.5-turboを採用

プロンプトエンジニアリング

- プロンプトの役割指定
 - LLMが特定のタスクを解決するための役割を指定
 - タスクのコンテキストの理解を助け、より正確で適切な応答を導くことが可能となる
- Chain-of-Thought (CoT) 推論
 - 複雑なタスクを段階に分けて出力させる手法
 - 本タスクは自明なタスクではないため、誘導が必要
 - 2名の開発者にAPI化のプロセスを言語化してもらい、プロンプトに落とし込む
 - 明確な指示, 構造化された出力, わかりやすい区切り文字 等

プロンプトエンジニアリング

- Few-shot学習

- 例を複数プロンプトに記載し，出力させる手法
- LLMは本タスクに学習されているわけではないため，例がある方が望ましい

例の抽出

- APIzatorデータセットからランダムに100件抽出し，以下の条件でフィルタリング
 - CoTプロンプトのプロセスをカバーできるコード
 - 3～10行程度のコード
 - ユニークなコード
- 5件のコードを採用

プロンプト (1/2)

Step1 : import文復元※
Step2 : メソッドの定義
Step3 : 修飾子定義
Step4 : メソッド名作成
Step5 : パラメータ推論※
Step6 : return文推論※
Step7 : throws文推論※
Step8 : コード出力

※必要があれば出力

Prompt	Instantiation
Role Designation	Give you a context including a question title, a question post and an answer post, your task is to transform the Java code snippet within the answer post into Java method based on the context.
Chain of Thought	To solve the problem, do the following: Step 1 - Recover import statements based on the code snippet. If necessary, it can be none. Step 2 - Define a public class Chatgpt that will be used to wrap the method. Step 3 - Create "public static" modifier for the method. Step 4 - Create the method name based on the context or the code snippet itself. Step 5 - Infer parameter list based on the code snippet. If necessary, it can be none. Step 6 - Infer return statements based on the code snippet. If necessary, it can be none. Step 7 - Infer throws statements based on the code snippet. If necessary, it can be none. Step 8 - Output the complete code based on the above results.
Example input	Here are some examples: Example one: Question title: How to convert <code>int[]</code> into <code>List<Integer></code> in Java? Question post: How do I convert <code>int[]</code> into <code>List<Integer></code> in Java? ..., I'll pick that one as the best to show the fact that this functionality is not part of Java. Answer post: There is no shortcut for converting from <code>int[]</code> to <code>List<Integer></code> ..., you have to make a utility method. <the code snippet in this answer> Code snippet in the answer post: <pre>int[] ints = {1, 2, 3}; List<Integer> intList = new ArrayList<Integer>(ints.length); for (int i : ints){ intList.add(i); }</pre>

プロンプト (2/2)

Step1 : import文復元※
Step2 : メソッドの定義
Step3 : 修飾子定義
Step4 : メソッド名作成
Step5 : パラメータ推論※
Step6 : return文推論※
Step7 : throws文推論※
Step8 : コード出力

※必要があれば出力

Example output	Specific steps: Step 1: <code>import java.util.ArrayList;import java.util.List;</code> Step 2: <code>public class Chatgpt {}</code> Step 3: <code>public static</code> Step 4: <code>convertIntArrayToList</code> Step 5: <code>(int[] arr)</code> Step 6: <code>return intList;</code> Step 7: <code>// None</code> Complete code: <code>import java.util.ArrayList; import java.util.List; public class Chatgpt { public static List<Integer> convertIntArrayToList(int[] arr){ List<Integer> intList = new ArrayList<Integer>(arr.length); for (int i : ints){ intList.add(i); } return intList; } }</code>
Test Input	Now, give you the following context: Question title: <Question title here>; Question post: <Question post here>; Answer post: <Answer post here>; Code snippet in the answer post: <Code snippet here>
Format Constraints	Please output the results in the following format: Specific steps: <the results of step 1-7> Complete code: <the result of step 8>

リサーチクエスチョン

- RQ1 | メソッドパラメータとreturn文の予測精度
- RQ2 | 意味のあるメソッド名を生成できるか
- RQ3 | CoT推論とFew-shot学習の効果
- RQ4 | 他のプログラミング言語への一般化
- RQ5 | コンパイル可能なAPIを生成できるか

RQ1 | パラメータ, リターン文精度

- 方法

- APlzatorデータセットから200件のデータを取得
- 人間がAPI化したコードとツールがAPI化したコードを比較
- パラメータ, リターン文, 実装が一致するかを目視

- 結果

		メソッド実装 の一致率	パラメータ の一致率	リターン文 の一致率	パラメータ or リターン文 の一致率
	Approach	M-Acc	P-Acc	R-Acc	PR-Acc
元論文の結果	APlzator (Ori)	31.5%	56.5%	57.5%	81.5%
再現実験の結果	APlzator (Cur)	32.5%	56.5%	57.5%	81.5%
	Code2API	43.5%	65.0%	66.0%	86.5%

- 先行研究よりも平均15%~16.5%精度高
- LLMにより, 柔軟な出力ができるようになったため

RQ2 | メソッド名の品質

- 方法

- APIzatorデータセットから200件のデータを取得
 - 人間がAPI化したコードとツールがAPI化したコードを含む
- 開発経験のある18人にアンケートを実施
 - データセットと参加者をそれぞれ6グループに分けて、主観評価を実施

- 評価指標

- メソッド名の表現力
 - メソッド名と実装の関連度合いを評価
 - スコア1～スコア4の四段階で、高いほど良い
- 使用意欲
 - 各コードのうちどれが使いやすいかを1つ選択
 - 1データあたり3人が投票するため、多数決を実施

RQ2 | メソッド名の品質

- 結果

	メソッド名 と実装が 無関係	メソッド名 と実装が 低い	メソッド名 と実装が 高い	メソッド名 と実装が 一致		使用意欲	
MNE SCORE	1	2	3	4	Avg	P-value	WILLINGNESS
Human	25 (4.2%)	77 (12.8%)	138 (23.0%)	360 (60.0%)	3.39	1.27e-8	96 (48.0%)
APIzator	136 (22.7%)	248 (41.3%)	156 (26.0%)	60 (10.0%)	2.23	1.85e-122	3 (1.5%)
CODE2API	14 (2.3%)	29 (4.8%)	111 (18.5%)	446 (74.3%)	3.65	-	101 (50.5%)

- Code2APIのメソッド表現力はAPIzatorを大差で上回った
 - ・ 人間が書いたコードよりも良いという結果になった
- CoT推論により，開発者と同じ思考プロセスでAPI化ができたため

RQ3 | アブレーションスタディ

- 方法

- 以下パターンで比較

- Few-shot学習を使用しない
 - CoT推論を使用しない
 - Few-shot学習とCoT推論を使用しない

- 結果

Prompt	メソッド実装 の一致率		パラメータ の一致率		パラメータ or リターン文 の一致率	
	M-Acc	P-Acc	R-Acc	PR-Acc		
w/o few-shot	22.5%	36.0%	55.0%	68.5%		
w/o CoT	30.5%	41.0%	53.5%	64.0%		
w/o both	6.5%	8.0%	10.5%	12.0%		
CODE2API	43.5%	65.0%	66.0%	86.5%		

- メソッド実装, パラメータ予測ではFew-shot学習が重要
- リターン文予測ではCoT推論が重要

RQ4 | 他プログラミング言語への一般化

- 方法

- SOからPythonに関する質問を100件収集
- 手作業でAPI化したコードと, Code2APIでAPI化したコードを比較
- RQ1と同じ評価指標を使用

- 結果

	メソッド実装 の一致率	パラメータ の一致率	リターン文 の一致率	パラメータ or リターン文 の一致率
Language	M-Acc	P-Acc	R-Acc	PR-Acc
Java	43.5%	65.0%	66.0%	86.5%
Python	57.0%	69.0%	80.0%	92.0%

- Javaのコードよりも精度が高い
- Pythonは型を記述する必要がなく, 一致させやすいため

RQ5 | コンパイル率

- 方法

- 各手法でAPI化したコードがコンパイルに成功するか
 - Original Code Snippet : SOの回答にあるコードそのまま
 - PostFinder^[41] : テキストマッチングにより, コンパイル可能なコードにラップする手法
 - APIzator : ルールベースの手法

- 結果

Method	Original Code Snippet	PostFinder	APIzator	Code2API
Compilation Rate	14.5%	46.5%	99.5%	95%

- APIzatorとCode2APIのコンパイル率はほぼ同程度
- 意味のあるメソッド名を出力できる点を考慮すると, Code2APIの方が優れている

[41] Riccardo Rubei, Claudio Di Sipio, Phuong T Nguyen, Juri Di Rocco, and Davide Di Ruscio. 2020. PostFinder: Mining Stack Overflow posts to support software developers. Information and Software Technology 127 (2020), 106367.

まとめと所感

- まとめ

- プロンプトエンジニアリングを活用した新しいAPI化手法「Code2API」の提案
- 先行研究より意味のあるメソッド名を出力することが可能
- 正しい振る舞いをするコードを出力することが可能

- 所感

- CoTの書き方とかは参考になるかも？
- Few-shot学習時の例の数とか，Pythonで検証した時の例をどうしているのかとかの説明が書かれていなくて，ちょっともやもやする