

# オープンソースソフトウェアでの ChatGPT 提案コードの再利用動向分析

宮永 翔多

京都工芸繊維大学

大学院工芸科学研究科 情報工学専攻

s-miyana@se.is.kit.ac.jp

崔 恩潯, 西浦 生成, 水野 修

京都工芸繊維大学

情報工学・人間科学系

{echoi, k-nishiura, o-mizuno}@kit.ac.jp

## 要旨

ChatGPTのような大規模な言語モデルの登場は、ソフトウェア開発の状況を一変させた。ソフトウェア開発における ChatGPT の利用は普及しており、GitHub 上のオープンソースプロジェクトにおいては、ChatGPT を使用して提案されたコードを再利用している場合がある。

本研究では、ソフトウェア開発における ChatGPT の利用動向を明らかにする。具体的には、まず、ChatGPT が提案したコードは開発者によってどのように再利用されているかを、次に、開発者は ChatGPT が提案したコードを再利用するまでにどの程度の回数 ChatGPT と会話しているかを調査した。

調査の結果、ChatGPT が提案したコードは開発者によって再利用される際、修正せず再利用されることが最も多く、次に、コードの機能的な修正をして再利用されることが多いとわかった。また、開発者は ChatGPT が提案したコードを再利用するまでに、複数回 ChatGPT と会話していることが多いとわかった。さらに、その会話回数は、修正せず再利用するとき、少なくなる傾向にあり、機能的な修正をして再利用するとき、多くなる傾向にあるとわかった。

## 1. 緒言

ChatGPT は、人間のような会話をする能力、ユーザのフィードバックから継続的に学習する能力を持ち合わせており、さらに、アクセスのしやすさもあり、絶大な人気を誇る。ChatGPT は、様々な分野で利用されており、ソフトウェア開発の分野も例外ではない。GitHub 上に公開されたオープンソースプロジェクトのコミット

には、ChatGPT を使用して提案されたコードを再利用している場合がある。しかし、ChatGPT が実際にソフトウェア開発者にどのように使用されているかについては、ほとんどわかっていない。

本研究では、開発者による ChatGPT を使用したソースコードの再利用タスクに着目し、ソフトウェア開発における ChatGPT の利用動向を明らかにする。具体的には、まず、ChatGPT が提案したコードは開発者によってどのように再利用されているかを調査した。次に、開発者は ChatGPT が提案したコードを再利用するまでにどの程度の回数 ChatGPT と会話しているかを調査した。ChatGPT が使用されたコミットにおけるソースコードの再利用動向を調査し、その特徴を明らかにした。

調査の結果、ChatGPT が提案したコードは開発者によって再利用される際、修正せずにそのまま再利用されることが最も多く、次に、コードの機能的な修正をして再利用されることが多いとわかった。また、開発者は ChatGPT が提案したコードを再利用するまでに、複数回 ChatGPT と会話していることが多いとわかった。さらに、修正せず再利用するとき、会話回数が少なくなる傾向にあり、機能的な修正をして再利用するとき、多くなる傾向にあるとわかった。

## 2. 背景

### 2.1. ChatGPT

ChatGPT<sup>1</sup>は GPT (Generative Pre-trained Transformer) 言語モデルを搭載した AI チャットボットで、2022年11月に OpenAI によってリリースされた。ChatGPT はプロンプトの指示に従い詳細な返答をするよう

<sup>1</sup><https://openai.com/blog/chatgpt>

に訓練されており [10], ユーザとの対話を通じて, 自然言語での質問応答を行うことができる. プロンプトとは, ChatGPT に対して提供される命令のことでカスタマイズしたり, 改良したりすることで, ChatGPT の応答を操作できる [15]. また, ChatGPT はインターネット上の膨大なテキストデータで学習されているため, 幅広いタスクをこなすことができる. 例えば, エッセイを書いたり, 楽曲の作詞をしたり, コードを書いたり, さらにデバッグしたりできる [13]. しかし, 時にもっともらしく聞こえるが正しくない, または無意味な応答を提供する [10]. そのため, 最近の研究では, 事実の矛盾を検証する手段を持たないエンドユーザにリスクをもたらすことが懸念されている [1,4,8,9]. ユーザを簡単に惑わせる誤った情報が蔓延していることから, Stack Overflow は ChatGPT で生成された回答の投稿を禁止している [11]. ChatGPT については, 生成される応答の品質, 様々な効果的なプロンプト技術, プログラミングコンテストでの比較性能など, 多くの研究が行われている [7,12,15,18]. しかし, ChatGPT が実際にソフトウェア開発者にどのように使われているかについては, ほとんどわかっていない.

## 2.2. ソースコードの再利用

ソースコードの再利用は一般的に観測されるタスクであり, 開発者はソースコードを再利用することで, 開発時間を節約し, コード品質を向上させる目的で行う [2]. しかし, ソースコードを再利用することは容易でない [3]. 再利用タスクは, ソースコードの探索と統合の2つのステップに分けられるが [2], ソースコードの再利用は, (1) 開発者が必要とするソースコードを見つけること, (2) 見つけたソースコードを手元のシステムに統合する, という2点で困難である [2,14]. まず1点目は, 開発者は開発しているシステムの背景を意識して必要とするものをキーワードや制約という形で正確に指定することが難しい [14]. この問題に対して, 開発者にとってより関連性のあるソースコードを見つけられるようにするコード検索技術など, 多くの研究が行われてきた [14]. 2点目は, ニーズを満たすソースコードを見つけられたとしても, そのコードを手元のシステムに統合するには, 膨大なコード片に対して変数の参照の変換など些細な変更に注力することが必要となるという点で, この作業は面倒でミスを引き起こしやすく, 開発者にとって手作業で

コードを統合することは難しい可能性がある [2]. この問題に対処するため, 変数名の自動変換など様々な手法が提案されてきた [2].

本論文では, 再利用元のシステムと手元のシステム間の類似ソースコード片を検索することで, 開発者のソースコード再利用の動向を調査する. 類似ソースコード片とは, ソースコード中に存在する互いに一致, または類似したものを指す. 類似ソースコード片を検索するツールの1つとして, NCDSearch がある [5]. NCDSearch は, ソースコードの類似度を計測するためのツールであり, Normalized Compression Distance (NCD) を含む, いくつかの計測方法が用意されている. 本論文の調査では, NCDSearch の類似性メトリクスとして Lempel-Ziv Jaccard Distance (LZJD) を用いた [6]. NCD は, 2つのソースコード片がデータ圧縮アルゴリズムにより高度に圧縮されている場合, それらを類似しているものとみなす. LZJD は, NCD の代替として導入されたメトリクスで, NCD と同じく圧縮から着想を得ているが, 実際のデータ圧縮結果ではなく, データ圧縮のための辞書を比較する [6].

また, 開発者のソースコード再利用方法について, Stack Overflow<sup>2</sup>からのソースコード再利用を調査した研究では, その利用方法は, 「完全コピー」, 「表面的な修正」, 「表面的でない修正」, 「アイデアの変換」, 「情報の提供」の5つのタイプに分類される [16].

## 3. 調査概要

### 3.1. 調査目的

本研究の目的は, ソフトウェア開発における ChatGPT の利用動向を明らかにすることである. 本研究では, 開発者による ChatGPT を使用したソースコードの再利用タスクに焦点を当てたアプローチを取る. そこで, 以下の研究設問を設定し, ChatGPT が提案したコードの再利用動向について調査した. この調査により, ソフトウェア開発において ChatGPT を活用したソースコード再利用のアプローチを容易にする方法について洞察が得られると期待する.

**RQ1** ChatGPT が提案したコードは開発者によってどのように再利用されているか?

<sup>2</sup><https://stackoverflow.com>

**RQ2** 開発者は ChatGPT が提案したコードを再利用するまでにどの程度の回数 ChatGPT と会話しているか？

### 3.2. 調査対象のデータセットの説明

調査対象としてコードスニペットを含む ChatGPT の応答とプロンプトのデータセットである、DevGPT を選択した [17]. このデータセットは、ソフトウェア開発の中間生成物とそれに対応する開発者と ChatGPT の会話ログを含む。中間生成物は、ソースコード、コミット、イシュー、プルリクエストからディスカッションや Hacker News のスレッドである。このデータを収集することで、開発者と ChatGPT のやり取りの包括的な分析を可能にすることを目的に作成された。

このデータセットがどのように作成されたのかを説明する。2023年5月下旬に OpenAI が導入した、ユーザが専用のリンクを通じて ChatGPT とのやり取りを共有できる機能を活用して、2023年7月27日から毎週 GitHub と Hacker News での ChatGPT 共有リンクのメンションを追跡した。2023年8月31日に追跡したデータには、2237個の GitHub または Hacker News の参照から 2891個の ChatGPT 共有リンクが含まれる。これらの情報がデータセットとして、提供されている [17].

このデータセットに含まれるソフトウェア開発者と ChatGPT の会話において、使用された ChatGPT のバージョンには、GPT-3.5 と GPT-4 が含まれる。また、データセットには行単位から、関数単位、プログラム単位まで様々な粒度の ChatGPT の提案したソースコードが含まれている。

## 4. RQ1: ChatGPT が提案したコードは開発者によってどのように再利用されているか？

### 4.1. 動機

ソフトウェア開発における ChatGPT の利用動向として、開発者による ChatGPT が提案したコードの再利用方法に着目する。ソフトウェア開発において再利用タスクは定期的に行われる。ChatGPT の提案したコードが開発者によってどのように再利用されたかは、ソフトウェア開発における ChatGPT の利用動向の1つの指標となると考えられる。

### 4.2. 方法

RQ1 を答えるために、ChatGPT が提案したコードとコミットにおいて修正されたソースコードを比較した。調査方法は、以下の手順で進めた。この手順を図 1 に示す。

手順1 データセットから、コミットとそれに対応する開発者と ChatGPT の会話ログを取得する。

手順2 開発者と ChatGPT の会話ログから、ChatGPT 提案コードを JavaScript, Python, Java で記述されたコードに絞る。

手順3 NCDSearch を使用して ChatGPT 提案コードに対して、対応するコミットから類似ソースコード片を検索する。

手順4 検索されたソースコード片から、再利用したソースコード片を手作業で識別し、そのコードの利用方法を分類する。

手順1では、DevGPT から、コミットとそれに対応する開発者と ChatGPT の会話ログを取得する。DevGPT からは、GitHub リポジトリへの URL、コミットハッシュを取得でき、この情報を用いて、GitHub から該当するソースファイルを取得できる。この時、該当するコミットのソースファイルが取得できないデータについては調査対象から除外した。

手順2では、コミットに対応する開発者と ChatGPT の会話ログの中で、ChatGPT が提案したコードを JavaScript, Python, Java の3つのプログラミング言語で記述されたコードに絞る。全てのプログラミング言語を対象にデータを分析するには時間がかかってしまうため、プログラミング言語を限定したデータに対して分析することにした。ここで、JavaScript, Python, Java を選択した理由は、コミットに対応する開発者と ChatGPT の会話ログ内の、ChatGPT 提案コードの中で、シェルスクリプト言語 (sh, bash) を除いた上位3つのプログラミング言語だからである。データセットにおいて、JavaScript, Python, Java のコードはそれぞれ、13.77%, 4.78%, 4.45%の割合で含まれている。

手順3では、NCDSearch を用いて ChatGPT が提案したコードに対して、対応するコミットのソースコードから類似ソースコード片を検索する。NCDSearch が類似ソースコード片の検索に使用する計測方法には LZJD を

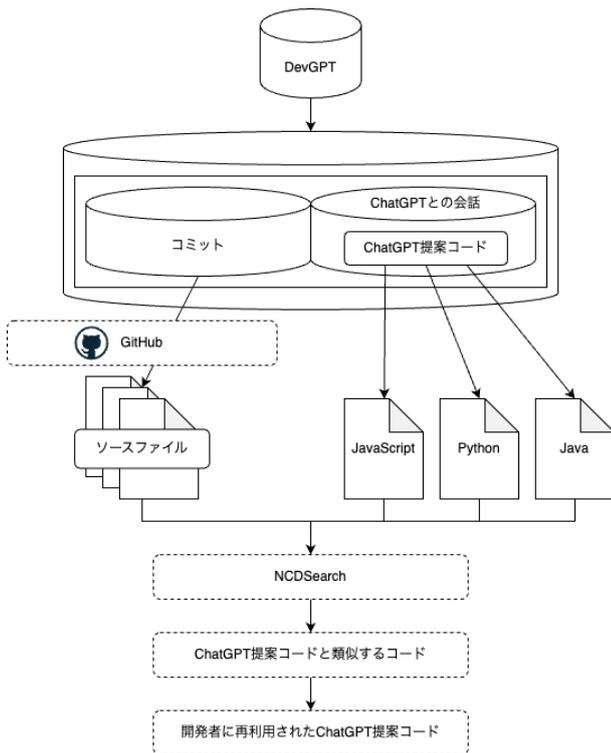


図 1. RQ1 における調査方法の概要図

利用する。またこの時、検索された類似ソースコード片から、対応するコミットで修正されたソースコードと関連性のないソースコード片は検索結果から除外した。

手順 4 では、検索された類似ソースコード片から、開発者が ChatGPT 提案コードを再利用したソースコード片を手作業で識別する。そして、開発者が ChatGPT 提案コードを再利用したソースコード片について、その利用方法を分類する。ここでは、検索された類似ソースコード片を全て開発者が ChatGPT 提案コードから再利用したソースコード片であると決定しなかった。その理由は、開発者が再利用したソースコードの候補となる ChatGPT 提案コードが複数存在するためである。ChatGPT は、開発者のニーズに応えられる提案を 1 回で生成しないこともあるため、会話中に開発者の再利用候補となるコードが複数存在する。そのため、本論文では、その中から再利用されたコードを識別するために手作業で識別することにした。開発者が、ChatGPT 提案

表 1. ソースコードの再利用タイプ分類

ID	名称	定義
C1	完全コピー	開発者が全く修正せず、ChatGPT 提案コードをコピー＆ペースト
C2	表面的な修正	開発者がコードの機能を変更せず、ChatGPT 提案コードをコピー＆ペースト
C3	機能的な修正	開発者がコードの機能を変更し、ChatGPT 提案コードをコピー＆ペースト
C4	アイデアの活用	開発者が ChatGPT 提案コードをコピー＆ペーストしない代わりに、提案を受け入れゼロからソースコードを実装

コードをどのように再利用しているか、表 1 に示す 4 タイプへ分類した。このタイプ分類は、Stack Overflow からのソースコードの再利用に関する研究に基づいて、作成した [16]。

#### 4.3. 結果

Python, Java, JavaScript で記述された ChatGPT の提案コードについて、開発者によって再利用されたコードの利用方法を分類した結果を表 2 に示す。

表 2 に示すように、再利用されたコードの 83.5% が修

表 2. ChatGPT 提案コードの再利用タイプ分類結果

ID	名称	合計	割合
C1	完全コピー	76	83.5%
C2	表面的な修正	1	1.1%
C3	機能的な修正	13	14.3%
C4	アイデアの活用	1	1.1%

```

1 + import express from 'express';
2 + import cors from 'cors';
3 + import { createPrompt } from
  ↪ './prompt/createPrompt.js';
4 + import { saveAndSendPrompt } from
  ↪ './interactiveSession/saveAndSendPrompt.js';
5 + import { api } from './config.js'; // Import your
  ↪ api object
6 +
7 + const app = express();
8 +
9 + // Enable CORS for all routes
10 + app.use(cors());
11 +
12 + app.use(express.json());
13 +
14 + app.post('/generate', async (req, res) => {
15 +   const { notes } = req.body;
16 +   const { prompt, saveto } = await
  ↪ createPrompt(notes);
17 +   const result = await saveAndSendPrompt(prompt,
  ↪ saveto, null, api); // pass the api object here
18 +   res.json({ prompt: result });
19 + });
20 +
21 + app.listen(3000, () => {
22 +   console.log('Server is running on port 3000');
23 + });

```

コード 1: 開発者による ChatGPT 提案コードの再利用例 (再利用タイプ C1)

正されずにそのまま再利用されている。これは、開発者は ChatGPT が提案したコードを再利用する際、修正せずにそのまま再利用するのが多いことを示唆している。また、表 2 より、開発者によって再利用された ChatGPT の提案コードのうち、14.3%がコードの機能的な修正をして再利用されている。これは、開発者は ChatGPT が提案したコードを再利用する際、修正せず再利用する場合に次いで、コードの機能的な修正をして再利用することが多いとわかる。

開発者によって再利用されたコードの 83.5%において、開発者は ChatGPT が提案したコードを修正せずにコピー&ペーストしている。コード 1 と図 2 にその例を示す。コード 1 は、コミットに含まれる 1 ファイルの変更差分<sup>3</sup>を示したものであり、緑色が追加の行を示している。図 2 は、再利用された ChatGPT の提案コード<sup>4</sup>である。これらを比較すると、開発者は ChatGPT の提案コードをそのまま再利用していることがわかる。

また、再利用されたコードの 1.1%において、開発者は ChatGPT が提案したコードをコピー&ペーストして、

<sup>3</sup><https://github.com/tisztamo/Junior/commit/0f9939#diff-79f4c7>

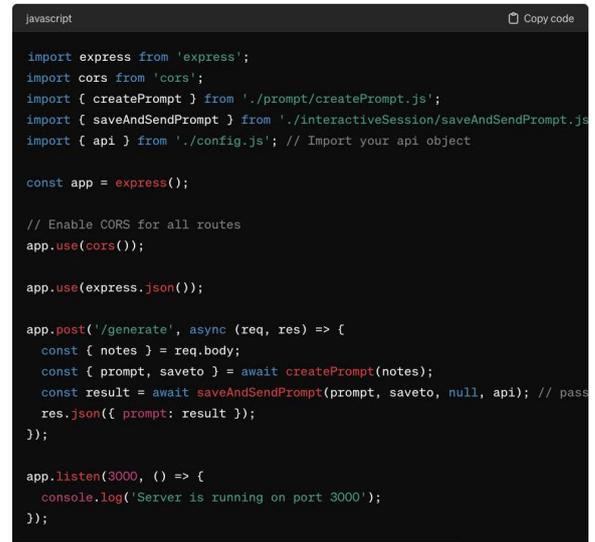
<sup>4</sup><https://chat.openai.com/share/bbf03be2-dc7b-4451-a614-1a542af33712>

#### ChatGPT

Thank you for providing the `config.js`.

From the provided file, I see that you export `api` object which you can import in `server.js` and pass it as an argument to the `saveAndSendPrompt` function.

Here is the corrected `server.js`:



```

javascript
import express from 'express';
import cors from 'cors';
import { createPrompt } from './prompt/createPrompt.js';
import { saveAndSendPrompt } from './interactiveSession/saveAndSendPrompt.js';
import { api } from './config.js'; // Import your api object

const app = express();

// Enable CORS for all routes
app.use(cors());

app.use(express.json());

app.post('/generate', async (req, res) => {
  const { notes } = req.body;
  const { prompt, saveto } = await createPrompt(notes);
  const result = await saveAndSendPrompt(prompt, saveto, null, api); // pass
  res.json({ prompt: result });
});

app.listen(3000, () => {
  console.log('Server is running on port 3000');
});

```

Please replace `import { api } from './config.js';` with the actual import statement that corresponds to your project structure.

If you are still getting the error, you might need to double-check how the `api` object is initialized in the `config.js` file. Make sure that the `OPENAI\_API\_KEY` environment variable is correctly set and the `ChatGPTAPI` is correctly initialized.

図 2. 開発者が修正せず再利用した ChatGPT 提案コード

開発者のシステムを動作させるために必要ではないと考えられる修正を加えている。

コード 2 にその例を示す。コード 2 は、ChatGPT が提案したコードと、開発者が実際にコミットしたコード<sup>5</sup>の差分を示したものである。このうち、緑色の行は開発者が新たに追加したコードを、赤色の行は ChatGPT の提案コードから開発者が削除したコードを示している。開発者は ChatGPT の提案コードをコピーし、コード 2 の 1 行目にある、開発者にとって不必要な「path」モジュールの import 文を削除している。

さらに、再利用されたコードの 14.3%において、開発者は ChatGPT が提案したコードをコピー&ペーストして、開発者のシステムを動作させるために必要な修正を加えている。コード 3 にその例を示す。

<sup>5</sup><https://github.com/tisztamo/Junior/commit/59b38e#>

```

1 - import path from 'path';
1 import { processPath } from './filesystem.js';
2 import { processInterfaceSection } from
  ↳ './processInterfaceSection.js';
3 import { printFolderStructure } from
  ↳ './printFolderStructure.js';
4
5 export const readAttention = async (attentionArray
  ↳ = [], attentionRootDir = '.') => {
6   try {
  @@ -9,6 +9,8 @@
9     if (trimmedLine.endsWith(' iface')) {
10      const filePath = trimmedLine.slice(0,
  ↳ -6).trim();
11      return
  ↳ processInterfaceSection(attentionRootDir,
  ↳ filePath);
12    } else if (trimmedLine.endsWith('/')) {
13      return
  ↳ printFolderStructure(attentionRootDir,
  ↳ trimmedLine.slice(0, -1).trim());
14    } else {
15      return processPath(attentionRootDir,
  ↳ trimmedLine);
16    }

```

コード 2: 開発者による ChatGPT 提案コードの再利用時の変更例 (再利用タイプ C2)

このコード 3 も、コード 2 と同様に、ChatGPT が提案したコードと、開発者が実際にコミットしたコード<sup>6</sup> の差分を示したものである。開発者は ChatGPT の提案コードをコピーし、コード 3 の 17 行目のように、「"Translate" というテキストの「Button」に、ボタンが押下されたのを見えるように機能させる「bind\_return\_key=True」を追加している。同様に、39 行目では、イベントのトリガとして機能する「'\r」を削除している。

そして、再利用されたコードの 1.1%において、開発者は ChatGPT が提案したコードから、そのアイデアを取り入れて新しいコードを作成している。コード 4 にその例を示す。このコード 4 も、コード 2 と同様に、ChatGPT が提案したコードと、開発者が実際にコミットしたコード<sup>7</sup> の差分を示したものである。開発者は ChatGPT の提案コードから、「marked」モジュールはデフォルトのエクスポートがないことを理解し、より正しい import 文に修正している。

RQ1 に対する回答を以下にまとめる。

diff-cd467d

<sup>6</sup><https://github.com/PyBites-Open-Source/italki-class-assistant/commit/8dc9e3#diff-b10564>

<sup>7</sup><https://github.com/tisztamo/Junior/commit/91ff33#diff-79f4c7>

```

1 import PySimpleGUI as sg
2 from translate import translate_text, LANGUAGES
3
4
5 def create_window():
  @@ -15,4 +15,4 @@
15     sg.InputText(key="-TEXT-", size=(50,
  ↳ 5))
16     ],
17 -     [sg.Button("Translate"),
  ↳ sg.Button("Exit")],
17 +     [sg.Button("Translate",
  ↳ bind_return_key=True), sg.Button("Exit")],
18     [sg.Text("Translation output:", size=(40,
  ↳ 1))],
  @@ -35,7 +35,7 @@
35     if event == sg.WINDOW_CLOSED or event ==
  ↳ "Exit":
36         break
37
38     # Check if "Translate" button is clicked
  ↳ or Enter key is pressed
39 -     if event == "Translate" or event == '\r':
39 +     if event == "Translate":
40         target_language_key = {v: k for k, v
  ↳ in LANGUAGES.items()}[values["-LANG-"]]
41         translated_text =
  ↳ translate_text(values["-TEXT-"],
  ↳ target=target_language_key)

```

コード 3: 開発者による ChatGPT 提案コードの再利用時の変更例 (再利用タイプ C3)

```

4 - import * as marked from 'marked';
4 + import { marked } from 'marked';

```

コード 4: 開発者による ChatGPT 提案コードの再利用時の変更例 (再利用タイプ C4)

ChatGPT が提案したコードは、修正せずに再利用されることが最も多く、次に、コードの機能的な修正をして再利用されることが多い。

## 5. RQ2: 開発者は ChatGPT が提案したコードを再利用するまでにどの程度の回数 ChatGPT と会話しているか?

### 5.1. 動機

ソフトウェア開発における ChatGPT 利用動向の指標として、開発者が ChatGPT の提案を再利用するまでのプロセスに着目する。開発者は、ChatGPT が提案したコードを再利用するまでにどの程度の回数 ChatGPT と会話しているかを調査する。会話回数が多いことは、開

発者は ChatGPT の提案したコードが必要とするコードであるかの判断を複数回しなければならないことを示している。そのため、ソースコードの再利用にかかる時間が増え、開発者の負担が大きくなることになる。ChatGPT の提案したコードが再利用されるまでにかかる会話の回数を調査することで、ソフトウェア開発における ChatGPT の利用動向を理解できる。

## 5.2. 方法

RQ2 を答えるために、開発者が再利用した ChatGPT の提案コードについて、そのコードが ChatGPT に提案されるまでにどの程度の回数会話されているかを調査した。調査方法は、以下の手順で進めた。この手順を図 3 に示す。

手順 i データセットから、コミットとそれに対応する開発者と ChatGPT の会話ログを取得する。

手順 ii 開発者と ChatGPT の会話ログから、ChatGPT 提案コードを Python, Java, JavaScript で記述されたコードに絞る。

手順 iii NCDSearch を使用して ChatGPT 提案コードに対して、対応するコミットから類似ソースコード片を検索する。

手順 iv 検索されたソースコード片から、再利用したソースコード片を手作業で識別する。

手順 v 再利用されたコードが、ChatGPT によって提案されるまでの会話回数を求める。

手順 i~手順 iv は、RQ1 の手順 1~手順 4 と同様であるため、本章では省略する。

手順 v では、開発者が ChatGPT の提案したコードを再利用するまでの開発者と ChatGPT の会話回数を求める。会話回数は、開発者の与えたプロンプトに対して ChatGPT が応答するのを 1 回の会話として数えることにする。手順 i で取得した会話ログから、開発者によって再利用されたコードが、ChatGPT によって開発者に提案されるまでに行われた会話の回数を求める。この時、1 つの会話ログの中で、複数のソースコードの再利用がある場合、2 回目以降のソースコードの再利用については、その 1 つ前の再利用以降の会話を対象にプロンプトの回数を求めた。

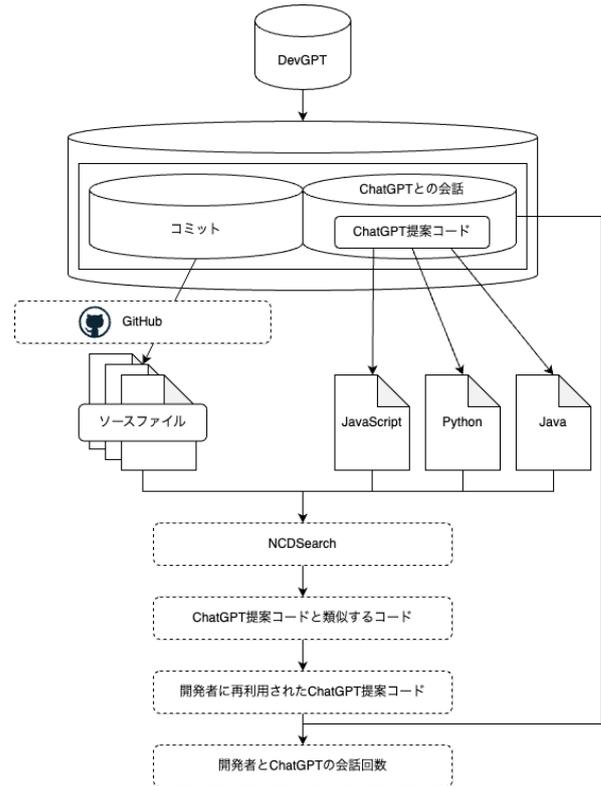


図 3. RQ2 における調査方法の概要図

## 5.3. 結果

開発者が再利用した ChatGPT の提案コードについて、そのコードを ChatGPT が提案するまでの会話回数を求めた結果を表 3 に示す。再利用の分類がタイプ C2 と C4 のコードは、それぞれ 1 つずつであったため、表には示していない。2 つのタイプのコードが開発者によって再利用されるまでに、タイプ C2 に分類されたコードは 1 回、タイプ C4 に分類されたコードは 4 回会話されていた。

表 3 の情報に基づいて作成した箱ひげ図を図 4, 図 5 に示す。図 4 に示すように、開発者が ChatGPT の提案コードを再利用するまでに、平均で 3.65 回会話されており、全体の半分を占めるコードが 1 回から 4 回の範囲で会話されている。これは、開発者が ChatGPT と複数回会話してから、コードを再利用していることが多いとわかる。

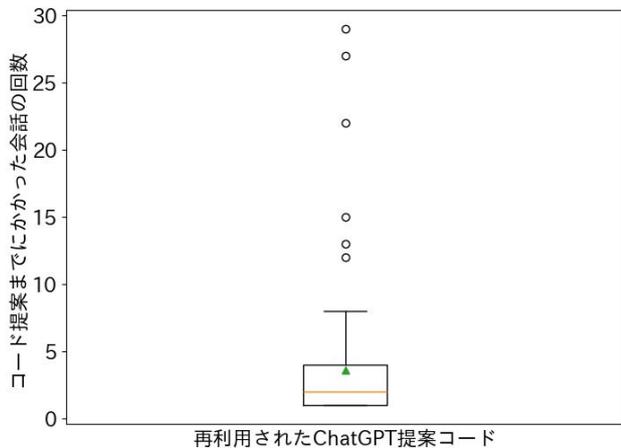


図 4. 開発者が再利用したコード（全体）を ChatGPT が提案するまでの会話回数の分布

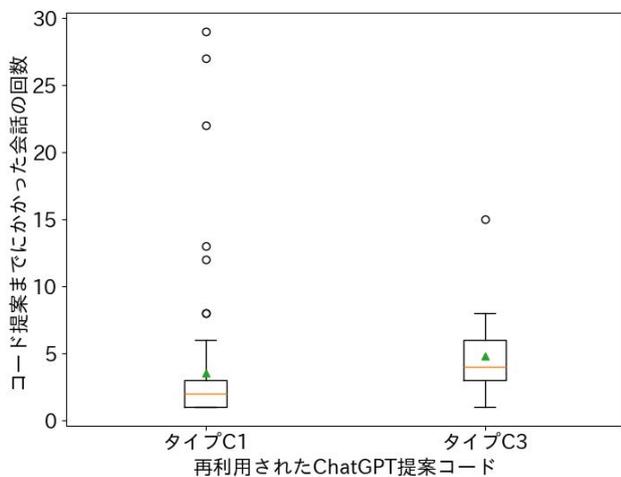


図 5. 開発者が再利用したコード（タイプ C1 と C3）を ChatGPT が提案するまでの会話回数の分布

表 3. 開発者と ChatGPT の会話回数の統計量

	C1	C2	C3	C4	全体
平均	3.49	-	4.77	-	3.65
第 1 四分位数	1.00	-	2.00	-	1.00
中央値	2.00	-	4.00	-	2.00
第 3 四分位数	3.00	-	6.50	-	4.00

また、図 5 に示すように、再利用されたコードのうち、再利用の分類がタイプ C1 のコードは、平均で 3.49 回会話されており、全体の半分を占めるコードが 1 回から 3 回の範囲で会話されている。一方で、図 5 に示すように、再利用の分類がタイプ C3 のコードは、平均で 4.77 回会話されており、全体の半分が 2 回から 6.5 回の範囲で会話されている。これは、開発者が ChatGPT の提案したコードを再利用するまでの会話回数が、修正せず再利用する場合、少なくなる傾向にあり、コードの機能的な修正をして再利用する場合、多くなる傾向にあることを示唆している。

RQ2 に対する回答を以下にまとめる。

開発者は ChatGPT が提案したコードを再利用するまでに、複数回 ChatGPT と会話していることが多く、また、その会話回数は、修正せず再利用する場合、少なくなる傾向にあり、機能的な修正をして再利用する場合、多くなる傾向にある。

## 6. 議論

### 6.1. RQ1 ChatGPT が提案したコードは開発者によってどのように再利用されているか？

ChatGPT が提案したコードは開発者によってどのように再利用されているかを調査した。その結果、第 4 章に示したように、表 2 より、ChatGPT が提案したコードを再利用する際、修正することが少ないとわかった。これは、開発者が必要とするコードを ChatGPT が提案できていることが多いと考えられ、ChatGPT を開発者のプロンプトに関連するコードを返すことができるコード検索ツールとして活用できる可能性があると考えられる。しかし、表 2 より、ChatGPT が提案したコードを再利用する際、コードの機能的な修正を必要とする場

合があることも分かった。すなわち、ChatGPT は開発者の必要とするコードを完全に提案できていないこともあることを示唆している。そのため、ChatGPT をコード検索ツールとして利用する上で、提案されたコードを開発者のシステムに統合する際、自動化機能があると、ChatGPT を開発者の障壁をより取り除いたコード検索ツールとして活用できると考えられる。

本論文では、ChatGPT を活用したコミットについて、ChatGPT の提案したコードが開発者によってどのように再利用されたかを調査したため、その再利用した提案コードがバグなど問題を引き起こすことがなかったかどうかはわかっていない。そのため、今後は再利用された ChatGPT の提案コードの動向を調査し、その影響について明らかにする必要があると考えられる。また、本論文では、ChatGPT との会話をリンクしたコミットについて調査したため、ChatGPT から何かしら有用な提案を得られ、それを利用したケースに、調査対象が限られていると考えられる。ChatGPT を利用しようとしたが、ChatGPT から開発者にとって有用な提案を得られなかったケースも存在すると考えられ、そのようなケースについても調査する必要があると考えられる。さらに、本論文では、ChatGPT が提案しているコードの種類については調査しなかったため、開発者が採用したコードがボイラプレートのようなものが多い傾向にある可能性が考えられる。ChatGPT がどの程度汎用的に使用できるかを理解するためには、ChatGPT が提案するコードの種類についても調査する必要があると考えられる。

さらに、Stack Overflow からのソースコード再利用について調査した研究 [4] では、調査されたファイルのうち 20.5% が、Stack Overflow からソースコードを修正せずそのまま再利用しており、31.5% が、何らかの修正をして再利用していることが報告されている。これを本調査の結果と比較すると、開発者が ChatGPT の提案したコードを再利用する際、修正することは少なく、より完全なコピーが多く、ChatGPT の提案コードの再利用は、Stack Overflow からのソースコード再利用よりも、修正が少ない傾向にあるとわかる。すなわち、ChatGPT を使用したソースコードの再利用は、Stack Overflow を使用した場合と比べて、ソースコードを開発者のシステムに統合する障壁が低いと考えられる。

## 6.2. RQ2 開発者は ChatGPT が提案したコードを再利用するまでにどの程度の回数 ChatGPT と会話しているか？

開発者は ChatGPT が提案したコードを再利用するまでにどの程度の回数 ChatGPT と会話しているかを調査した。その結果、第 5 章に示すように、開発者は ChatGPT が提案したコードを再利用するまでに、ChatGPT と 1 回から 4 回程度の会話をしていることが多いとわかった。開発者が ChatGPT の提案コードを再利用するまでどれくらい会話をしているかは、開発者が ChatGPT の提案をどの程度検討してから受け入れられているかを示す指標となる。平均 3.65 回という会話回数は、開発者が ChatGPT に複数回提案を提示させてから、受け入れる提案を決定していることを示唆している。すなわち、ChatGPT の提案を受け入れるまでに開発者が ChatGPT との会話を重ね、自身の必要とするコードを得ようとしていると考えられる。

また、表 3 と図 5 から、タイプ C1 のコードは、平均して 3.49 回の会話を経て ChatGPT の提案が受け入れられていることが明らかになった。タイプ C1 全体の半数のコードが、1 回から 3 回目の会話で受け入れられていることから、ChatGPT が提案したコードを修正なしで再利用する場合、会話回数が少なくなる傾向にあることを示唆している。一方、タイプ C3 のコードは、平均して 4.77 回の会話を経て受け入れられた。タイプ C3 のコードの全体の半数が、2 回から 6.5 回の会話で受け入れられていることから、開発者が ChatGPT の提案コードを機能的な修正をして再利用する場合、ChatGPT が提案したコードを修正なしで再利用する場合により会話回数が増える傾向があることが示唆している。

本調査から、ChatGPT が提案するコードをそのまま再利用する場合に比べ、機能的な変更をして再利用する場合は、ChatGPT との会話回数が増える傾向にあると明らかになった。しかし、それがプロンプトによるものか ChatGPT の性能によるものかは本調査だけではまだ明らかになっていない。そのため、今後さらにデータセットを増やし、深い分析及び調査を進めて明らかにする必要があると考えられる。

## 7. 結言

本研究では、ソフトウェア開発者は ChatGPT が提案したコードをどのように再利用しているか調査した。

調査の結果、ソフトウェア開発者は ChatGPT が提案したコードを修正せず再利用することが最も多く、次に、コードの機能的な修正をして再利用することが多いとわかった。また、開発者は ChatGPT が提案したコードを再利用するまでに、複数回会話をして再利用することが多く、その会話回数は、修正せずに再利用するとき、少なくなる傾向にあり、機能的な修正をして再利用するとき、多くなる傾向にあるとわかった。

## 謝辞

本研究は JSPS 科研費 23K11046 の助成を受けたものです。

## 参考文献

- [1] A. Borji. A Categorical Archive of ChatGPT Failures, Apr. 2023. arXiv:2302.03494 [cs].
- [2] R. Cottrell, R. J. Walker, and J. Denzinger. Semi-automating small-scale source code reuse via structural correspondence. In *Proceedings of the 16th International Symposium on Foundations of software engineering*, pages 214–225, 2008.
- [3] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., Boston, 1995.
- [4] B. Guo, X. Zhang, Z. Wang, M. Jiang, J. Nie, Y. Ding, J. Yue, and Y. Wu. How Close is ChatGPT to Human Experts? Comparison Corpus, Evaluation, and Detection, Jan. 2023. arXiv:2301.07597 [cs].
- [5] T. Ishio, N. Maeda, K. Shibuya, and K. Inoue. Cloned Buggy Code Detection in Practice Using Normalized Compression Distance. In *Proceeding of the 34th International Conference on Software Maintenance and Evolution*, pages 591–594, 2018.
- [6] T. Ishio, N. Maeda, K. Shibuya, K. Iwamoto, and K. Inoue. NCDSearch: Sliding Window-Based Code Clone Search Using Lempel-Ziv Jaccard Distance. *IEICE Transactions on Information and Systems*, E105-D(5):973–981, May 2022.
- [7] S. Kabir, D. N. Udo-Imeh, B. Kou, and T. Zhang. Who Answers It Better? An In-Depth Analysis of ChatGPT and Stack Overflow Answers to Software Engineering Questions, Aug. 2023. arXiv:2308.02312 [cs].
- [8] J. Kocoń, I. Cichecki, O. Kaszyca, M. Kochanek, D. Szydło, J. Baran, J. Bielaniec, M. Gruza, A. Janz, K. Kanclerz, A. Kocoń, B. Kopytyra, W. Mieleleszczenko-Kowszewicz, P. Miłkowski, M. Oleksy, M. Piasecki, c. Radliński, K. Wojtasik, S. Woźniak, and P. Kazienko. ChatGPT: Jack of all trades, master of none. *Information Fusion*, 99:1–37, Nov. 2023.
- [9] S. Mitrović, D. Andreoletti, and O. Ayoub. ChatGPT or Human? Detect and Explain. Explaining Decisions of Machine Learning Model for Detecting Short ChatGPT-generated Text, Jan. 2023. arXiv:2301.13852 [cs].
- [10] OpenAI. Introducing chatgpt.
- [11] StackOverflow. Temporary policy: Generative ai (e.g., chatgpt) is banned.
- [12] H. Tian, W. Lu, T. O. Li, X. Tang, S.-C. Cheung, J. Klein, and T. F. Bissyandé. Is ChatGPT the Ultimate Programming Assistant – How far is it?, Aug. 2023. arXiv:2304.11938 [cs].
- [13] L. Tung. Chatgpt can write code. now researchers say it’s good at fixing bugs, too.
- [14] S. Wang, D. Lo, and L. Jiang. Active code search: incorporating user feedback to improve code search relevance. In *Proceedings of the 29th International Conference on Automated Software Engineering*, pages 677–682, 2014.
- [15] J. White, Q. Fu, S. Hays, M. Sandborn, C. Olea, H. Gilbert, A. Elnashar, J. Spencer-Smith, and D. C. Schmidt. A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT, Feb. 2023. arXiv:2302.11382 [cs].
- [16] Y. Wu, S. Wang, C.-P. Bezemer, and K. Inoue. How do developers utilize source code from stack overflow? *Empirical Software Engineering*, 24(2):637–673, Apr. 2019.
- [17] T. Xiao, C. Treude, H. Hata, and K. Matsumoto. Devgpt: Studying developer-chatgpt conversations. In *Proceedings of the 21st International Conference on Mining Software Repositories*, 2024.
- [18] B. Yetiştiren, I. Özsoy, M. Ayerdem, and E. Tüzün. Evaluating the Code Quality of AI-Assisted Code Generation Tools: An Empirical Study on GitHub Copilot, Amazon CodeWhisperer, and ChatGPT, Oct. 2023. arXiv:2304.10778 [cs].