REST API仕様に基づく 大規模言語モデルを用いた自動バグ修正手法

山岸 克紀[1]　吉田 則裕[1,a)]　槇原 絵里奈[1]

概要：クラウドサービスの多くは，ウェブ API の一種である REST API を提供しており，クライアントのプログラムからのアクセスが可能となっている．REST API のクライアント開発では，開発中のプログラムが REST API 仕様を満足しないことを，テスト時にレスポンスを確認する時点になって気づくことになりやすい．多くの場合，レスポンスに含まれるエラーコードやエラーメッセージは，API 仕様のどの部分を満足していないかを特定するためには不十分な情報しか含まれていない．そのため，REST API のクライアントを開発する際は，リクエストの送信とレスポンスの受信を繰り返しながら，デバッグをすることになりやすい．そこで本研究では，REST API を呼び出すクライアントのプログラムを対象として，REST API の誤用を検出し，自動修正する手法を提案する．提案手法では，まずプログラムから API 仕様を満足しないコード片を検出する．次に，大規模言語モデルに与えるための検出したコード片および満足しない仕様を含むプロンプトを生成する．最後に，そのプロンプトを大規模言語モデルに与えることで誤用の自動修正を行う．適用実験では，API の誤用事例を収集し，提案手法を適用した．その結果，提案手法はほとんどの事例について，API 仕様を満足しないコード片の検出に成功した．また，提案手法は，大規模言語モデルに誤用事例のプログラム全体を与えた場合と比較して，より多くの誤用事例を修正できることがわかった．

Automated Program Repair Based on REST API Specifications Using Large Language Models

Katsuki Yamagishi[1]　Norihiro Yoshida[1,a)]　Erina Makihara[1]

1. はじめに

近年，IaaS（Infrastructure as a Service）や PaaS（Platform as a Service），SaaS（Software as a Service）などのクラウドサービスが爆発的に普及している．これらクラウドサービスの多くは，REST (REpresentational State Transfer) アーキテクチャスタイル [1] に準拠したウェブ API である REST API を提供しており，クラウドサービス提供者以外が作成したプログラム（クライアント）からのアクセスが可能となっている．

クライアントは，REST API を提供するクラウドサービスが管理するリソースに対してアクセスや操作を行うために，HTTP リクエストを送信する．この HTTP リクエストでは，エンドポイントの指定のためにリソースのパス名を，リソースに対して実行されるアクションとしてメソッド名（例：GET や POST）を記述する．多くの場合，HTTP ヘッダにメタデータ（例：データ形式）を付加したり，HTTP ボディにペイロード（例：テキストデータ）を記述したりする．

REST API のクライアント開発では，プログラムが API 仕様を満足しないことを，テスト時にレスポンスを確認することによって気づくことになりやすい．多くの場合，レスポンスに含まれるエラーコードやエラーメッセージは，API 仕様のどの部分を満足していないかを特定するためには不十分な情報しか含まれていない．そのため，REST API のクライアントを開発する際は，HTTP リクエストの

[1] 立命館大学
Ritsumeikan University, Ibaraki, Osaka 567–8570, Japan
[a)] norihiro@fc.ritsumei.ac.jp

送信とレスポンスの受信を繰り返しながら，デバッグを行うことになりやすい．

これまで，Java の API を対象とした誤用検出 [2], [3], [4] や自動修正 [5], [6], [7] に関する研究が行われている．これら研究は， Java の API を対象として，仕様を満足しない API 呼出しをソースコートから検出している．Java の API は，REST API とはクライアントにおける呼び出し方法が大きく異なる．具体的には，Java の API では，API を呼び出すメソッド呼出しが仕様を満足している必要があるが，REST API では，クライアントにおいて記述されたエンドポイントやリクエストヘッダ，リクエストボディが仕様を満足している必要がある．そのため，これらの研究で行っている誤用検出を REST API に適用することは困難である．

そこで本研究では，REST API を呼び出すクライアントのプログラムを対象として，REST API の誤用を検出し，自動修正する手法（図 4 参照）を提案する．提案手法では，まずプログラムから API 仕様を満足しないコード片（逸脱点）を特定する．次に，大規模言語モデル（LLM）に与えるための逸脱点および満足しない仕様（不満足仕様）を含むプロンプトを生成する．最後に，そのプロンプトを LLM に与えることで誤用の自動修正を行う．

適用実験では，SwitchBotAPI の誤用事例を収集し，提案手法を適用した．まず RQ1 として，逸脱点の検出を行うことができるか調査したところ，ほとんどの事例について逸脱点の検出に成功した．次に RQ2 として，LLM に誤用事例のプログラム全体を与えた場合と比較して，逸脱点および不満足仕様を含むプロンプトを生成する提案手法がより多くの誤用事例を修正できるかどうか調査した．その結果，提案手法の方がより多くの誤用事例を修正できることがわかった．

追加実験では，REST API を呼び出す Python モジュールの誤用に対する提案手法の有効性を確認した．適用対象として，REST API の 1 つである OpenAI API を呼び出す Python モジュールを選んだ．追加の実験の結果，LLM のみを使用した場合と比べて，OpenAI API の Python モジュールに関する誤用をより多く修正することができた．

以降，2 章で本研究に関する用語や背景について述べ，3 章で提案手法を述べる．4 章で，実験に用いるデータセットや実験結果を述べ，5 章で OpenAI の Python モジュールを対象とした追加実験を述べる．6 章で関連研究を述べた後に 7 章で本研究のまとめと課題点を述べる．

2. 背景

2.1 REST API

クラウドサービスの多くは，REST (REpresentational State Transfer) アーキテクチャスタイル [1] に準拠したウェブ API である REST API を提供しており，クラウドサービス提供者以外が作成したプログラム（クライアント）からのアクセスが可能となっている．

```
1  switchbot: 1.1
2  :
3  paths:
4    v1.1/devices:
5      post:
6  :
7        requestBody:
8          properties:
9              command:
10                 dsecription: the name of the command
11                 type: string
12             parameter:
13                 description: some commands require
14                 parameters,such as SetChannel
15                 type: string
16             commandType:
17                 dexcription: for customized buttons,
18                 this needs to be set to customzie
19                 type: string
20         required:
21             - command
22 :
```

図 1: SwitchBotAPI 仕様の一部抜粋

REST API を利用するクライアントは，必要なパラメータをリクエストに含める必要がある．必要なパラメータとは主に以下の 3 つである．

- エンドポイント：多くの場合，URL が指定される．
- リクエストヘッダ：POST や GET などのリクエスト．
- リクエストボディ：サーバ側に送りたい内容を含む．

SwitchBotAPI[8] とは, SwitchBot 社が提供している API であり，SwitchBot を用いて IoT 機器の操作を行うことができる．主にロボット掃除機やホームオートメーション，スマートロックなどの IoT 機器に対応しており，API を用いて様々な操作をすることができる．SwitchBotAPI は 2022 年にバージョン 1.1 に変更され，それに伴い新製品は旧バージョンの対応を停止している．SwitchBotAPI の利用者は，主に公式が提供している仕様を参考に開発を行っていく．この仕様書は API リファレンスとしてウェブで公開されており，ユーザはこれを参照して API を記述していく．また，API リファレンスはオープンソースとして GitHub に公開されている．**図 1** は，yaml で記述された API 仕様の一部を抜粋したものである．図 1 の 7 行目から 21 行目にかけてリクエストボディに関するパラメータが記述されている．リクエストボディには必須であるパラメータとそうでないものがあり，必須であるパラメータは 20 行目の required に記述されている．

```
1             :
2  def get_device_list() -> json:
3      url = f'{API_URL}/devices'
4      response = requests.get(url, headers=HEADERS)
5      return response.json()
6      HEADERS = {
7      'Authorization': OPEN_TOKEN,
8      'Content-Type': 'application/json; charset=utf-8'
9      #リクエストヘッダ内に`sign`や`t`, `nonce`の値が記述
10     #されていない
11     }
12            :
```

図 2: SwitchBotAPI の誤用事例

```
1             :
2  def get_device_list() -> json:
3      url = f'{API_URL}/devices'
4  +   string_to_sign = bytes(f"{self._token}{t}{nonce}",
5     "utf-8")
6  +   sign = base64.b64encode(
7                  hmac.new(
8                  bytes(self._secret, "utf-8"),
9                  msg=string_to_sign,
10                 digestmod=hashlib.sha256,
11             ).digest()
12     response = requests.get(url, headers=HEADERS)
13     return response.json()
14     HEADERS = {
15     'Authorization': OPEN_TOKEN,
16     'Content-Type': 'application/json; charset=utf-8',
17 +   't' = int(round(time.time() * 1000)),
18 +   'sign' = sign,
19 +   'nonce' = uuid.uuid4()
20     }
21            :
```

図 3: SwitchBotAPI の誤用修正例

2.2 REST API の誤用事例

図 2 は，実際に誤用を含んだコード例である．

図 2 の場合，誤用は 6 行目から 8 行目のリクエストヘッダ部分である．リクエストヘッダには Authorization のほかに，sign や t，nonce の記述が必要である．**図 3** に修正例を示す．

2.3 大規模言語モデルに基づく自動修正

Xia らは，事前トレーニングを行った大規模言語モデル（LLM）を用いた自動バグ修正方法を提案している [9]．Xia らが行った研究は，LLM を自動バグ修正に直接適用することを目的としており，9 つの最新 LLM とデータセットを用いて自動バグ修正の評価を行っている．LLM の評価を行うにあたり，様々な修復設計を用いて LLM の自動バグ修正に対する性能を調べている．以下に修復設計を示す．

- 完全な関数生成：バグを含んだ関数を入力とし，関数全体のパッチ生成を行う．
- 正しいコードの挿入：バグの位置が判明している場合，関数の接頭辞と接尾辞を指定して，パッチを生成する．
- 単一行の修正：バグを単一行で修正し，パッチ生成を行う．

これらの修復設計とデータセットを用いて，LLM によるバグ修正個数や，パッチ生成速度，コンパイル速度などを評価している．Xia らは，LLM による自動バグ修正にはスケーリング効果があり，モデルが大きいほど性能が向上することを示した．さらに，修復のテンプレートを用いることや，サンプルサイズの増加を行うことによってさらに性能を向上させることができると示している．

3. 提案手法

本章では，REST API 仕様から逸脱したプログラムを修正するために，LLM を用いた自動修正手法を提案する．本論文では，プログラム言語の構文上は正しいが，REST API 仕様に満足しないコード片を「逸脱点」と呼ぶ．また，REST API 仕様上において，満足に記述されていない部分を「不満足仕様」とする．**図 4** に，提案手法の概要を示すまた，提案手法は以下の手順で構成される．

手順 1．仕様の逸脱点を検出： プログラムから API 仕様に満足しない逸脱点を検出するために，API 利用に関するプログラム要素（エンドポイントである URL やリクエストヘッダの変数など）を抽出する．API 仕様のパラメータ（**2.1** 節参照）は GitHub から仕様書を取得し，API 記述に必要な仕様パラメータを取得する．その後，プログラム要素と仕様パラメータを照合することで不満足仕様を検出し，それに対応したプログラム要素が逸脱点として検出される．

手順 2．LLM を用いた自動バグ修正： **手順 1** で得た逸脱点のプログラムにおける位置情報や属性値を含んだプロンプトを作成する．具体的には予め用意したテンプレートに基づき，逸脱点を修正するプロンプトを生成する．そして，そのプロンプトを入力とし，逸脱点の自動修正を行う．

3.1 仕様の逸脱点を抽出

自動修正を実行する前にプログラム解析を行う目的は，逸脱点を検出し，不満足仕様および逸脱点を含むプロンプトを LLM に与えることで自動修正を行うことである．これにより，単純に API の誤用が混入したコード片の位置情報を含むプロンプトを LLM に与える方式と比べて，高精度な自動修正を期待できる．**手順 1** は以下の 3 つに分割できる．

手順 1-A．API 仕様パラメータを取得する： API 仕様

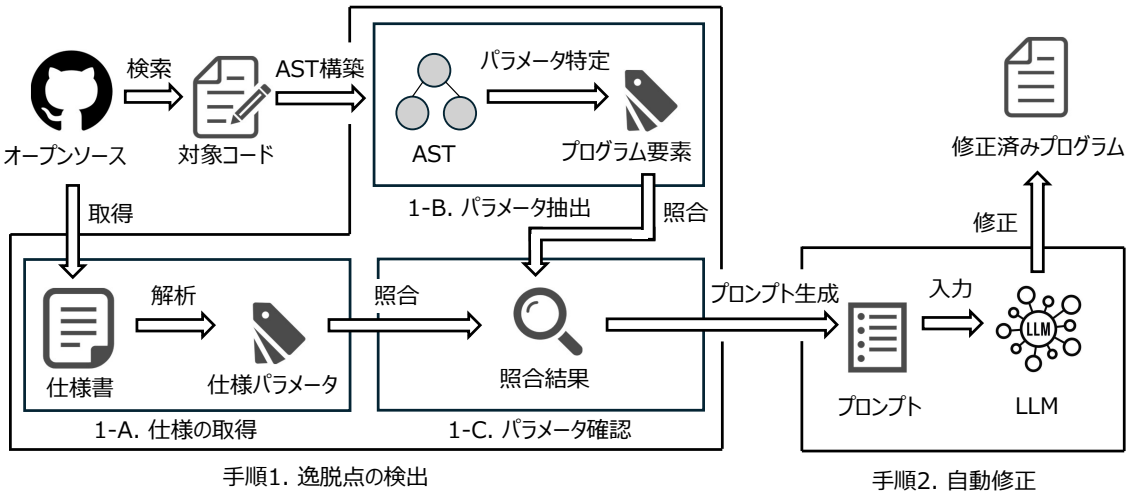

図 4: 提案手法

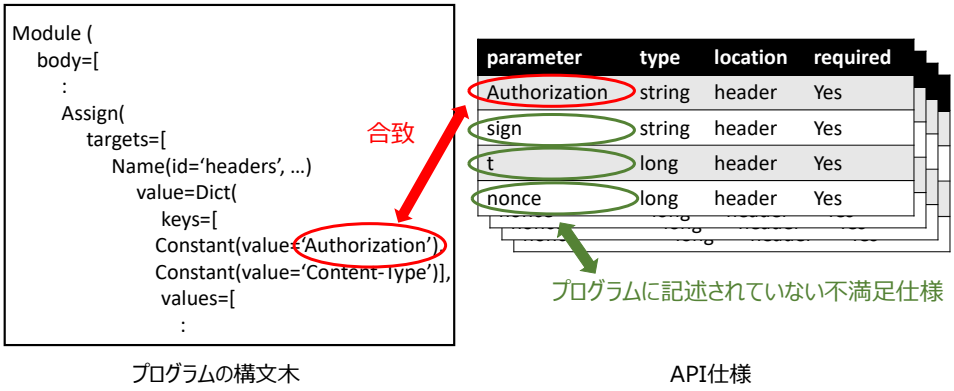

図 5: 解析対象プログラムと API 仕様の比較図

書から，API 呼出しに必須である仕様パラメータを取得する．

手順 1-B. パラメータ抽出： API 仕様に関する要素をプログラム要素として取得する．

手順 1-C. パラメータ確認： 取得したプログラム要素と API 仕様を比較し，不満足仕様を検出する．その後，不満足仕様に対応するプログラム要素が逸脱点として検出される．

手順 1-A では，API を使用する上で必須である仕様パラメータを取得する．次に，**手順 1-B** における，API 仕様に必要な要素の取得方法について説明を行う．この手順では，プログラムの構文解析を行う．抽出すべきプログラム要素は，**2.1** 節末尾で示した 3 つのパラメータに関するものである．ここでは SwitchBotAPI のそれぞれについて取得を行う．最後に，**手順 1-C** において，API 仕様を取得し，**手順 1-B** で抽出したプログラム要素と比較する（**図 5**）．比較基準はパラメータの種類によって異なるため，以下のように基準を定める．

1. エンドポイント： API を呼び出す URL が API 仕様中の URL と一致しているか．

2. リクエストヘッダ： API 仕様で必須とされている API リクエストヘッダ中の各属性が，定義されているか．ただし，各属性の値は考慮しない．

3. リクエストボディ： API 仕様書において必須とされている属性が定義されているか．ただし，各属性の値が API 仕様で指定されている場合は，比較を行い，そうでない場合は，比較を行わない．

図 5 は，図 2 のプログラムを**手順 1** に基づいて実行した．図 2 のプログラムは，リクエストヘッダの部分に不満足仕様があるとして検出される．図 5 のように，プログラムに API 仕様の 1 つである Authorization が記述されているが，sign や t，nonce といった他の値は記述されていない．このように不満足仕様に対応するプログラム要素が逸脱点として検出される．

3.2 LLM に基づいた修正

逸脱点の位置情報や属性値を含むプロンプトを LLM への入力として修正を試みる．このプロンプトは，**手順 1** で

```
"""
以下のコードには API の誤用が含まれている．
最新の仕様に基づくと，${逸脱点}が修正すべき場所である．
正しくは，${修正提案}であり，それに伴う修正が必要です．
その部分を特定し最新の仕様に基づいたプログラムを
出力してください．
"""
${逸脱点を含むプログラム}
```

図 6: プロンプトのテンプレート例

```
"""
以下のコードには API の誤用が含まれている．
最新の仕様に基づくと，リクエストヘッダが
修正すべき場所である．
正しくは，リクエストに sign，t，nonce が必要であり，
それに伴う修正が必要です．
その部分を特定し最新の仕様に基づいたプログラムを
出力してください．
"""
            :
def get_device_list() -> json:
    url = f'{API_URL}/devices'
    response = requests.get(url, headers=HEADERS)
    return response.json()
    HEADERS = {
    'Authorization': OPEN_TOKEN,
    'Content-Type': 'application/json; charset=utf-8'
    }
            :
```

図 7: 図 6 のテンプレートに基づくプロンプト生成例（SwitchBotAPI）

検出した逸脱点の位置情報や属性値を含み，**図 6** のようなテンプレートに基づき生成する．プロンプトには，逸脱点である変数（仕様上の属性に対応）と修正提案を記述し，LLM に与える．テンプレートは，なるべく逸脱点や修正提案に焦点を向け，その他の部分で出力結果に影響を出させないよう簡潔にした．

8 行目以降には逸脱点を含むプログラムを記述する．

図 7 は，図 2 のコードから逸脱点を検出した後，実際にテンプレートを用いたプロンプト生成例である．逸脱点は変数 READERS で定義されているリクエストヘッダの sign や t，nonce が不満足仕様である．この内容を図 7 の 5 行目と 6 行目で記述している．また，API 仕様を満足するために不満足仕様をプロンプトに記述する．

4. 適用実験

本章では，実験に用いた誤用事例の準備方法と実験結果を述べる．LLM を用いた実験について，まず逸脱点を示すコード片を含むプロンプトと含まないプロンプトの両者を作成した．逸脱点を含まないプロンプトは，図 6 の 3 行目と 4 行目を省略し，逸脱点の情報や修正提案を含まない．そのプロンプトを用いて，LLM への修正をそれぞれ 5 回ずつ実行した．そのうち 1 回以上成功したプロンプトを修正成功とする．本研究の適用実験では，2 つの RQ を設定する．

表 1: SwitchBotAPI に関する逸脱点データ

誤用の種類	
エンドポイント	7
API リクエスト	4
リクエストボディ	1
計	12

表 2: SwitchBotAPI の逸脱点に関する検出成功数

誤用の種類	
エンドポイント	5/7
API リクエスト	2/4
リクエストボディ	1/1
計	8/12

表 3: SwitchBotAPI の自動修正結果

	LLM のみ	提案手法
エンドポイント	0/5	4/5
API リクエスト	0/2	1/2
リクエストボディ	1/1	1/1
計	1/8	6/8

RQ1: 提案手法により仕様の逸脱点を検出できたか？
収集した API の誤用事例の中で，検出に成功した数を調査する．

RQ2: 提案手法により LLM の性能を向上させることができたか？
提案手法により，LLM の場合と比べて LLM の修正性能が向上したかを実験する．

4.1 データセットの作成

データセットに関して，実験では表 1 に記載したデータセットを用いた．これは我々が，GitHub から収集したデータセットである．対象としたリポジトリは https://github.com/OpenWonderLabs/SwitchBotAPI で，すでにクローズされたイシューからデータを収集した．また，どの基準で採用するか判断する必要があるため，我々は API 仕様に反しているプログラムを基準としてデータを収集した．しかし SwitchBotAPI では，イシューによる修正例を見つけることができなかったため，コミットによって API 記述や API 仕様パラメータが修正されたものを収集した．その中で仕様に反したコード片が含まれたプログラムを目視で抽出し，データセットとした．また，このデータセットを準備するための作業は全て第一著者が行った．

4.2 実験結果

本項目では，実験において設定した RQ について結果を

述べる．RQ1 について，表 2 のとおり SwitchBotAPI では，エンドポイントに関して 7 件中 5 件，API リクエストに関して 4 件中 2 件，リクエストボディに関して 1 件中 1 件の逸脱点の検出に成功した．

RQ2 について，表 3 は，提案手法による逸脱点の検出が可能になったデータを LLM に入力として与えた結果を示す．次に逸脱点を検出できたデータの修正を試みた．表 3 は，修正に成功したかどうかの内訳を示している．

提案手法により修正できた誤用事例に関して，SwitchBotAPI の場合は，提案手法を用いた場合 8 件中 6 件，提案手法を用いない場合，8 件中 1 件の修正が可能になった．いずれの API でも逸脱点に関する情報を与えたほうが修正数が向上した．

> RQ1 への回答：SwitchBotAPI では，12 件中 8 件の逸脱点の検出が可能であった．RQ2 への回答：提案手法により，修正の成功件数が増加した．

4.3 考察

RQ1 について，提案手法が行うプログラム解析により，API 仕様に満足しない逸脱点を発見することができた．SwitchBotAPI では，すべてのデータから逸脱点を検出することが難しかった．理由の一つとして，エンドポイントを表す URL を記述する位置がプログラムによって大きく異なることや文字列結合などによる複雑な計算により特定が難しかったことが例として挙げられる．表 2 の API リクエストが 4 件中 2 件検出できなかった理由として，プログラムがリクエストヘッダの要素をわけて記述していたことが挙げられる．

RQ2 について，提案手法におけるプログラム解析を行うことによって修正できた個数が向上したことが示された．提案手法の有無による比較実験を行ったことで，逸脱点や不満足仕様の抽出が修正に有効であることが示された．提案手法を用いずに LLM を実行した場合，ほとんど修正に失敗している．API 仕様からの逸脱点以外を変更しているデータが多かったため，LLM が逸脱点の部分を特定できていないことが理由として考えられる．

具体例として，LLM が修正したプログラムを**図 8** と**図 9** に示す．図 8 はプロンプト，図 9 は図 8 の出力結果であるが，差分をわかりやすくするために LLM による変更点を diff 形式で示している．実際の LLM では-で示された箇所は出力されていない．図 9 の 12 行目と 13 行目のように，プロンプトで示された修正提案に沿って逸脱点が修正されており成功している．一方 20 行目から 22 行目で示されている nonce，t，sign のパラメータ追加は修正が成功したように見える．しかし値が任意の文字列となっており，適切な値を入力することができていない．よって 20 行目から 22 行目の API リクエストに関しては修正が不可能という結果になった．考えられる原因として，LLM がなんらかの理由で nonce，t，sign のパラメータを SwitchBotAPI の仕様パラメータであることを認識することができず，値を任意の文字列として補完したと考えられる．しかし，API リクエスト内のパラメータを逸脱点とする他のデータでは nonce，t，sign に適切な値設定ができていたパターンがあったため，LLM が SwitchBotAPI の仕様を保持していないことは考えにくい．LLM は内部構造がブラックボックスであるため，仕様パラメータを認識できなかった詳細な理由は明確にできなかった．

5. 追加実験

4 章の適用実験で用いた SwitchBotAPI ではエンドポイントを示す URL を記述する必要があったが，OpenAI API では Python モジュールを利用することによってエンドポイントを示す URL の記述を直接行う必要がない．追加実験では，OpenAI API を呼び出すモジュールの誤用に提案手法を適用することで，REST API を呼び出すモジュールの誤用を対象とした有効性を確認した．

5.1 OpenAI API の Python モジュール誤用事例

OpenAI API[10] とは，OpenAI が提供している API であり，言語処理や文章生成，画像生成などを行うことができる．代表的な生成 AI として ChatGPT が挙げられるが，そのほかにも様々な生成 AI が使用できるようになっている．ChatGPT の API は 2023 年にリリースされ，現在も活発なアップデートが行われており，API 仕様が高い頻度で変更されている．OpenAI API は公式が実装した Python モジュールを使用することによりエンドポイントを示す URL を直接記述しなくとも API を利用することができる．**図 10** は OpenAI API モジュールの誤用例が示されている．図 10 の 2 行目の Python package である openai は OpenAI の仕様書から自動生成されたものであり，Python モジュールパスが API のエンドポイントパスと対応づくように自動生成されている．よってユーザは，図 10 の 5 行目にある openai.ChatCompletion のような Python モジュールパスを指定することで，どの API リクエストを実行するか決めることができる．OpenAI API の Python モジュールの最新バージョンではモジュールパスの記述方法の変更やクライアントのインスタンス化が必要である．モジュールパスに関して，図 10 の 5 行目のようなモジュールパスでは API リクエストは動作せず，**図 11** の 8 行目のようにモジュールパスを client.chat.completions と記述する必要がある．

5.2 実験手法

実験手法に関して，大まかな流れは 4 章と同じであるた

```
"""
以下のコードには API の誤用が含まれている．
最新の仕様に基づくと，エンドポイント URL と API リクエストが
修正すべき場所である．
正しくは，URL は v1.1 に，API リクエストは nonce,t,sign の
値を設定することであり，それに伴う修正が必要です．
その部分を特定し最新の仕様に基づいたプログラムを
出力してください．
"""
import requests
import json
from dotenv import load_dotenv

load_dotenv()

OPEN_TOKEN = os.getenv('OPEN_TOKEN')

API_HOST = 'https://api.switch-bot.com'

DEBIVELIST_URL = f"{API_HOST}/v1.0/devices"

HEADERS = {
    'Authorization': OPEN_TOKEN,
    'Content-Type': 'application/json; charset=utf8'
}
            :
```

図 8: LLM による修正例（プロンプト）

```
                :
  import requests
  import json
  import os
  from dotenv import load_dotenv
  load_dotenv()

  OPEN_TOKEN = os.getenv('OPEN_TOKEN')

  API_HOST = 'https://api.switch-bot.com'

- DEBIVELIST_URL = f"{API_HOST}/v1.0/devices"
+ DEBIVELIST_URL = f"{API_HOST}/v1.1/devices"
# <-- Update to v1.1

# リクエスト情報
  HEADERS = {
    'Authorization': OPEN_TOKEN,
    'Content-Type': 'application/json; charset=utf8'
+   'nonce': 'nonceValue',  # <-- add nonce value
+   't': 'tValue',  # <-- add t value
+   'sign': 'signValue'  # <-- add sign value
}
                :
```

図 9: LLM による修正プログラム差分（出力結果）

```
import os
import openai

openai.api_key = os.getenv("key")
completed = openai.ChatCompletion.create(
#クライアントのインスタンス化，及びモジュールパスの変更が必要
model="gpt-3.5-turbo",
messages=[
{"role": "user", "content": "Hello!"}
]
)
print(completion.choices[0].message)
```

図 10: OpenAI API の Python モジュール誤用の事例

```
  import os
- import openai
+ from openai import OpenAI

+ client = OpenAI(openai.api_key = os.getenv("key"))
- openai.api_key = os.getenv("key")
- completed = openai.ChatCompletion.create(
+ completed = client.chat.completions.create(
  model="gpt-3.5-turbo",
  messages=[
  {"role": "user", "content": "Hello!"}
  ]
  )
  print(completion.choices[0].message)
```

図 11: OpenAI API の Python モジュール誤用に対する修正例

め，ここでは 4 章と異なる部分のみを述べる．一つ目はデータセットの作成についてである．データセットの作成は https://github.com/openai/openai-python のリポジトリからイシューを確認し，逸脱点を含むプログラムを目視で抽出した．4 章では SwitchBotAPI の場合，GitHub でイシューの修正例を見つけることができなかったためコミットから収集したことを述べたが，OpenAI API の Python モジュールの場合はイシューによる修正例を見つけることができたため，こちらを採用した．LLM に入力するプロンプトに関して，例を**図 12** に示す．二つ目は LLM への入力とするテンプレートの生成方法である．プロンプトのテンプレートは 4 章同様図 6 を使用しているが，提案手法のプログラム解析によって修正提案を提示することができない場合，図 6 の 4 行目を省略する．これは極端な例であるが，逸脱点であるモジュールパスが aaaa.bbbb と記述されていた場合，開発者がどのようなパスを使おうとしたのか不明であるため修正提案を提示することができない．このような場合は省略する．

```
"""
以下のコードには API の誤用が含まれている．
最新の仕様に基づくと，completed を変数とするモジュールパスが
修正すべき場所である．
その部分を特定し最新の仕様に基づいたプログラムを
出力してください．
"""
import os
import openai

openai.api_key = os.getenv("key")
completed = openai.ChatCompletion.create(
            :
```

図 12: 図 6 のテンプレートに基づくプロンプト生成例（OpenAI API）

5.3 実験結果

RQ.1 について，逸脱点の検出成功数の結果を表 5 に示す．Python のモジュールパスに関する逸脱点を含むコード片の検出は 20 件，リクエストボディは 1 件となり，すべての逸脱点を検出することに成功した．RQ.2 について，LLM における自動修正の結果を表 6 に示す．提案手法により修正できた誤用事例は 21 件中 9 件であった．一方で，提案手法を用いず LLM のみで修正できた誤用事例は 21 件中 1 件であった．この 1 件は，モジュールパスに不備があったもので，実行した 5 回の内 1 回のみ修正に成功した．他のエラーに関しては，提案手法を用いなかった場合では修正することができなかった．結果，逸脱点に関する情報を LLM に与えたほうが修正数が向上した．

5.4 考察

実験結果から，Python のモジュールパスが満足しない場合も，提案手法を用いることでより多くの逸脱点を抽出することができた．OpenAI API ではモジュールを介して API リクエストが行われるため，仕様に関する部分の特定が比較的容易であった．しかし，今回収集したデータセットには複数の関数にまたがってモジュールが記述される等，複雑にラッピングされたデータが無い．今後そのようなデータが含まれた場合現時点では逸脱点の特定が困難になると予想する．LLM による修正の結果に関して，OpenAI API の Python モジュールは SwitchBotAPI よりも修正割合が低いという結果になった．原因として，LLM が保持している OpenAI API の Python モジュールに関する情報が最新のバージョンに対応していないことが考えられる．今回収集したデータはほとんどがモジュールパスによるものであり，プロンプトにモジュールパスの修正提案を行っていても修正がされないパターンが多かった．具体的に新バージョンの仕様では chat.completions と記述しなければいけないが旧バージョンの仕様である ChatCompletion と

表 4: OpenAI API の Python モジュールに関する逸脱点データ

誤用の種類	
モジュールパス	20
API リクエスト	0
リクエストボディ	1
計	21

表 5: OpenAI API の Python モジュールに関する逸脱点の検出成功数

誤用の種類	
モジュールパス	20/20
API リクエスト	0
リクエストボディ	1/1
計	21/21

表 6: OpenAI API の Python モジュール誤用に対する自動修正結果

	LLM のみ	提案手法
モジュールパス	1/20	8/20
API リクエスト	0	0
リクエストボディ	0/1	1/1
計	1/21	9/21

出力されていた場合が多かった．これは LLM が OpenAI API の Python モジュールに関する古いバージョン情報を保持しており，それを優先して回答を出力しているからだと予想している．今回の実験では，LLM が修正を行った際に LLM がもともと保持してる情報にもとづいて修正されているか判別することはできなかった．

6. 関連研究

Java の API を対象とした誤用検出 [2], [3], [4] や自動修正 [5], [6] に関する研究が行われている．Sven らは， Java API の使用法のグラフ表現である API Usage Graph を提案し，そのグラフ表現に基づいて誤用事例検出および順位付けを行う MUDetect を提案している [4]．Ren らは Java API の使用制約に関する知識をグラフ化し，そのグラフに基づき誤用事例を検出する手法を提案している [3]．Li らは，クライアントやライブラリ，Java API 仕様書から API 使用制約をそれぞれ抽出し，API 使用制約グラフを構築することで，API の誤用事例検出を行っている [2]．荒木らは，誤用事例に関連する Java API 使用パターンを抽出し，誤用事例に不足しているメソッド呼び出しを追加することで，自動修正を行う手法を提案している [6]．Kechagia らは，既存の自動修正ツールを用いて，Java の API 誤用事例を修正できるか評価を行っている [5]．Java の API は，REST API とはクライアントにおける呼び出し方法が大きく異なる．具体的には，Java の API では，API を呼び出すメソッド呼出しが仕様を満足している必要があるが，REST API では，クライアントにおいて記述されたエン

ドポイントやリクエストヘッダ，リクエストボディが仕様を満足している必要がある．そのため，これらの研究で行っている誤用検出を REST API に適用することは困難である．

REST API に関する研究として，クラウドサービスが提供する REST API を検証する研究が挙げられる [11], [12]．Atlidakis は， REST API 使用の依存関係解析を行うことで，REST API の呼び出し列を自動生成するファジングツール RESTler を開発し，GitLab や Microsoft Azure，Office365 が提供する REST API の検証に使用している [11]．また Huang は，プログラム解析を行うことで，クラウドサービスが提供する REST API が仕様どおりに実装されているか検証する手法を提案している [12]．本研究では，REST API を呼び出すクライアントを対象として自動修正を行う手法を提案した．

Xia らは，既存の自動バグ修正ツールよりも LLM の方が自動バグ修正を行う能力が高いことを示した [9]．また，Jin らは，静的解析ツールと LLM を組み合わせることで，LLM のみを使用する場合と比べて，より効率的な自動バグ修正を行うことができることを示した [13]．プログラム解析と LLM を組み合わせる点については，Jin らの研究と共通しているが，本研究で検出する不満足仕様や逸脱点は，一般的な静的解析ツールでは検出できないと考えられる．

7. まとめと今後の課題

本研究では，API 仕様に満足しないプログラムを，LLM を用いて自動修正を適用することを目的とした．提案手法では，API 仕様書からパラメータと，クライアントのプログラムから API 仕様に関するプログラム要素を抽出する．その後，両者を照合することでプログラム要素の逸脱点を検出する．この取得した逸脱点の位置情報や値を含めてプロンプトを生成した．最後に作成したプロンプトとプログラムを LLM の入力とし，自動バグ修正を試みた．適用実験では，SwitchBotAPI を対象にして，自動バグ修正を行った．結果，逸脱点の情報を含んだプロンプトの方が含んでいないプロンプトと比較して，より多くの逸脱点を修正することができた．追加実験では，OpenAI API の Python モジュールの誤用を対象として，自動バグ修正を行った．モジュールを使用した API であった場合でも，SwitchBotAPI と同様に提案手法を用いたほうがより多くの逸脱点を修正することができた．

今後の課題として，LLM が保持している古いバージョンの情報が出力のすべてに影響を受けているか調査する必要がある．また，逸脱点が複雑にラッピングされた場合の特定や，修正提案を詳細に記述できるようにプログラム解析を改善する必要がある．さらに，LLM の修正性能をより向上させるため，本実験の提案手法で取得した API 仕様を学習させることを考えている．

謝辞 本論文の校正に際し，ご協力くださった立命館大学 小澤隆氏および西本優作氏に感謝します．本研究は，JST さきがけ JPMJPR21PA ならびに JSPS 科研費 JP24K02923 の支援を受けたものです．

参考文献

[1] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, 2000.

[2] Can Li, Jingxuan Zhang, Yixuan Tang, Zhuhang Li, and Tianyue Sun. Boosting API misuse detection via integrating api constraints from multiple sources. In *Proc. of MSR 2024*, pp. 14–26, 2024.

[3] Xiaoxue Ren, Xinyuan Ye, Zhenchang Xing, Xin Xia, Xiwei Xu, Liming Zhu, and Jianling Sun. API-misuse detection driven by fine-grained API-constraint knowledge graph. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, ASE '20, p. 461–472, New York, NY, USA, 2021. Association for Computing Machinery.

[4] Amann Sven, Hoan Anh Nguyen, Sarah Nadi, Tien N. Nguyen, and Mira Mezini. Investigating next steps in static API-misuse detection. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pp. 265–275, 2019.

[5] Maria Kechagia, Sergey Mechtaev, Federica Sarro, and Mark Harman. Evaluating automatic program repair capabilities to repair API misuses. *IEEE Transactions on Software Engineering*, Vol. 48, No. 7, pp. 2658–2679, 2022.

[6] 荒木良仁, 桑原寛明, 國枝義敏. API 利用パターンを用いた自動プログラム修正手法. 情報処理学会研究報告, Vol. 2021-SE-207, No. 3, pp. 1–9, 2021.

[7] Sebastian Nielebock. Towards API-specific automatic program repair. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 1010–1013, 2017.

[8] strunker. SwitchBot API v1.1. `https://github.com/OpenWonderLabs/SwitchBotAPI`(2024 年 1 月 24 日アクセス).

[9] Chunqiu Steven Xia, Yuxiang Wei, and Lingming Zhang. Automated program repair in the era of large pre-trained language models. In *Proceedings of the IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pp. 1482–1494, 2023.

[10] OpenAI. OpenAI documentation. `https://platform.openai.com/docs/overview`(2024 年 5 月 29 日アクセス).

[11] Vaggelis Atlidakis, Patrice Godefroid, and Marina Polishchuk. Restler: Stateful rest api fuzzing. In *Proceedings of the IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pp. 748–758, 2019.

[12] Ruikai Huang, Manish Motwani, Idel Martinez, and Alessandro Orso. Generating rest api specifications through static analysis. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ICSE '24, New York, NY, USA, 2024. Association for Computing Machinery.

[13] Matthew Jin, Syed Shahriar, Michele Tufano, Xin Shi, Shuai Lu, Neel Sundaresan, and Alexey Svyatkovskiy. Inferfix: End-to-end program repair with llms. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foun-*

dations of Software Engineering, ESEC/FSE 2023, p. 1646–1656, New York, NY, USA, 2023. Association for Computing Machinery.