関数単位の修正箇所特定によるリポジトリレベルのバグ修正

近藤瑞希 [1]　河原大輔 [1]　倉林利行 [2]
[1] 早稲田大学　[2] 日本電信電話株式会社
{kondmiznotfound@toki., dkw@}waseda.jp　toshiyuk.kurabayashi@ntt.com

概要

近年、大規模言語モデル (LLM) のコード生成能力は飛躍的に上昇し、ソフトウェアエンジニアリングタスクを LLM に適用する研究が多く行われている。本研究では、LLM の並列処理能力に着目し、LLM エージェントを用いないリポジトリレベルでのバグ修正手法を提案する。リポジトリ内から修正が必要なファイルと関数を特定し、各関数を LLM で独立に修正し、統合する。これらを複数のモデルを用いてアンサンブルする。SWE-bench Lite を用いた評価の結果、ファイルの検索と関数の特定は高精度だったが、バグ修正の精度は他のシステムと比べて大きな改善は見られなかった。

1　はじめに

日本において、IT 人材の不足は深刻な問題であり、2030 年には最大で 79 万人が不足するとされている[1)]。この課題を解決する方法の 1 つとして、LLM を用いたソフトウェア開発支援が考えられる。

ソフトウェア開発は主にリポジトリレベルで行われており、ソースコードの管理や共同開発の効率化のために GitHub などのサービスが広く利用されている。開発中に発生する問題の管理・共有には GitHub Issue（以下、Issue）が提供されており、開発者はバグの報告などを行うことができる。

実在するリポジトリの Issue を集めた、Issue 解決ベンチマーク SWE-bench [1] を基に、LLM を活用した多くの手法が提案されている [2, 3, 4, 5, 6, 7, 8, 9]。これらの手法は、LLM をエージェント化して Issue 解決を指示する手法と、LLM エージェントを用いずにシステムを構築して Issue 解決を試みる手法の二種類に分類される。本研究ではエージェントを用いない手法に注目する。

エージェントを用いないシステムでは、コードベースから必要なファイルを検索し、修正箇所を特定し、修正する 3 ステップで処理されることが多い。従来の手法では、ファイルの検索や修正箇所の特定にリポジトリのグラフ情報 [2] やモンテカルロ法 [3] などを活用したものが提案されている。また、修正箇所の特定は行単位、クラス単位、関数単位など、多様な粒度で行われてきた。

本研究では、LLM の並列性と SWE-bench Lite における正解データのパッチ (バグ修正に適用されるコード変更であり、修正前後の差分）の修正箇所が少ないという特性に着目し、LLM エージェントを用いない新たな手法を提案する。我々の検索手法では、グラフ情報などを用いずに、複数の LLM の結果をアンサンブルするシンプルな手法によって、高精度なファイル検索および関数単位の修正箇所の特定を実現する。また、特定した関数を、独立に修正する手法を提案する。

SWE-bench Lite において、本手法は 25.3% の Issue 解決率を達成した。ファイルの検索と関数の特定は高精度であったが、バグ修正の精度は他の手法と比較して、大きな改善は見られなかった。

2　関連研究

2.1　SWE-bench

SWE-bench は、GitHub にある 12 個の著名な Python リポジトリから構成され、GitHub の Issue とそれに対応する修正を集めたデータセットである。全部で 2,294 問があり、そのサブセットとして 300 問の SWE-bench Lite や 500 問の SWE-bench Verified が存在する。SWE-bench ではリポジトリ内のコードベースとその中でどのような問題が起きているかの文章 (Issue 文) が与えられる。それら以外の情報は与えられず、インターネットでの検索等は行えない。Issue を解決するためのパッチを提出し精度を評価する。その正誤判定はすべてのテストケースに通過することで正解とみなし、1 つでも間違っていたら不正解

1) https://www.meti.go.jp/policy/it_policy/jinzai/houkokusyo.pdf

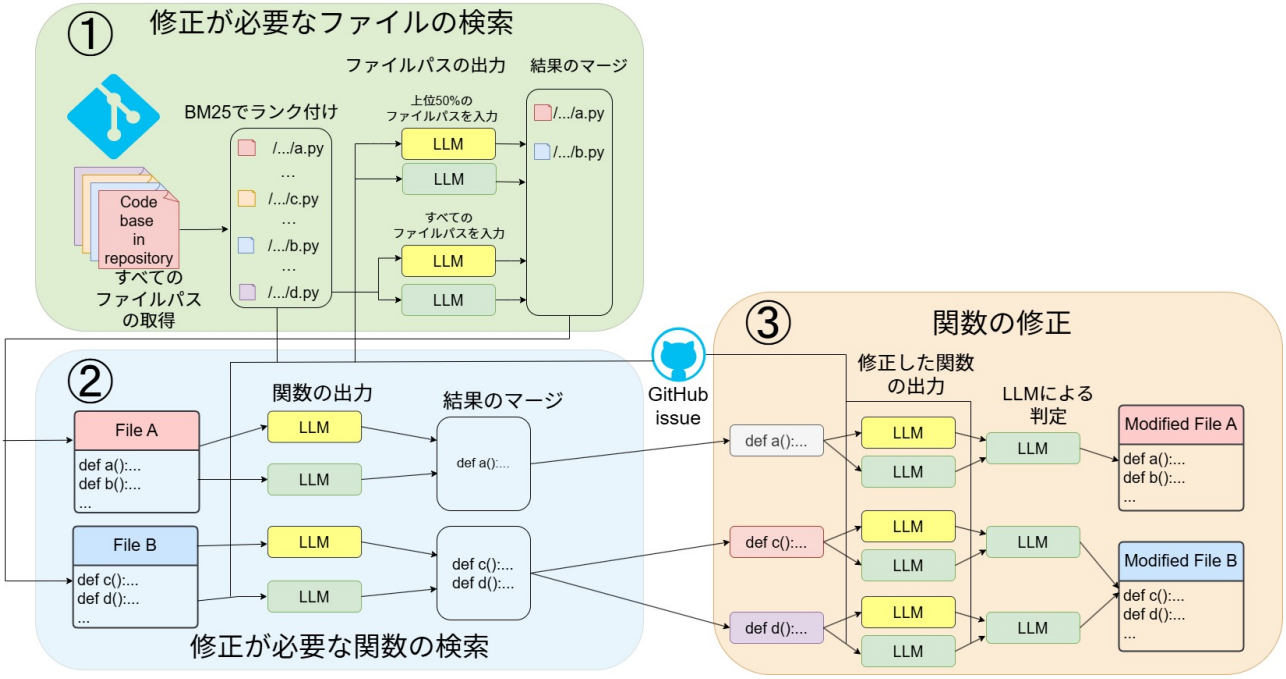

図 1: 提案手法

とみなす。BM25 [10] を用いた RAG [11] によるベースラインは約 3%程度の精度で難易度が高いタスクである。2024 年以降、多くの研究成果が発表されている。

2.2 SWE-bench を解くシステム

SWE-bench が発表されて以降、SWE-bench を LLM で解くさまざまな手法が提案されている。それらは主にエージェントを使うシステムと、エージェントを使わないシステムの 2 種類に分類できる。LLM エージェントとは、指示に基づくタスクを自律的に遂行し、外部ツールやデータと連携して問題を解決するシステムである。与えられた指示に対してどのような行動がよいかを思考し、それをもとに行動を決定する ReACT [12] などが挙げられる。

SWE-bench を解く LLM エージェントとしては、ACI というインターフェースで決められたコマンドを実行できる SWE-Agent [4] や、役割に応じて複数のエージェントを使用する CodeR [5] などがある。一方、エージェントを用いないシステムとしては、検索と特定の後に多数の回答サンプルを作成し、それらを LLM で選択する Agentless [7] などがある。また、既存の手法で解ける問題群が異なっていることを利用した DEI [6] といった手法も存在する。2024 年 1 月 10 日時点での SWE-bench Lite における最高精度は 49.00%である。

3 正解パッチの分析

正解のパッチがどのような傾向を持つかについて、SWE-bench Lite を用いた調査を行った。調査内容は 1 パッチあたりのファイルの変更数、関数の変更数、新しい関数の生成数、そして関数以外の箇所の変更数である。その結果を表 1 に示す。

表 1a を見ると SWE-bench Lite ではすべてのタスクにおいて 1 つのファイルの変更しか行われていなかった。また、表 1b によると関数の変更数は 7 割以上が 1 つの変更であり最大でも 4 つの変更にとどまる。表 1c から、新しい関数の生成や関数以外の箇所の変更が少ないことがわかる。同様の傾向は SWE-bench 全体でも確認された。

4 提案手法

3 節の分析から、関数の変更のみでも十分な精度が出るのではないかと仮説を立てる。この仮説に基づき、修正対象のファイルと関数を特定する検索モジュールと、特定した後に修正する修正モジュールからなる手法を提案する。提案手法を図 1 に示す。

4.1 検索モジュール

検索モジュールは修正対象のファイルの検索と、ファイルから修正する関数を特定する 2 ステップからなる。

表 1: SWE-bench Lite 正解パッチの統計 (300 件)

(a) 正解パッチの合計変更数

項目	合計変更数
ファイル変更	300
関数変更	315
新規関数生成	43

(b) 関数変更数の分布

関数変更数	個数
0	27
1	238
2	29
3	5
4	1

(c) 正解パッチの変更内容

項目	300 件の中の変更数
関数変更	273
新規関数生成	31
関数以外の変更	59

4.1.1 修正が必要なファイルの検索

最初にリポジトリ内のすべてのコードファイル (.py ファイル) を取得する。それらを BM25 でランキング付けする。具体的には、ファイルの内容と Issue 文の BM25 スコアをもとにファイルをランキング付けする。

次に、BM25 の結果を用いて LLM でファイルの検索を行う。LLM に Issue 文と、BM25 のランキング上位 50 %のファイルのパス、または、すべてのファイルを BM25 でランキング付けした順番で入力し、修正が必要なファイルをすべて出力させる。この処理を複数回、複数のモデルで実行し、結果の和集合を取り、ファイルの検索を行う。

4.1.2 修正が必要な関数の特定

4.1.1 節で得られたファイルから LLM を用いて修正が必要な関数を特定する。LLM にファイルの内容と Issue 文を入れ、それぞれの関数について、関数が必要かどうかを「Yes」か「No」で出力させる。この処理を複数回、複数のモデルで実行し、結果の和集合を取り、関数の特定する。

4.2 修正モジュール

4.1.2 節で得られた各関数を LLM で独立に修正する。LLM には、Issue 文と検索したファイル全体、及び特定した関数と関数名を入力し、修正した関数のみを出力させる。また、必要に応じて他のモジュールのインポート文の記述も出力させる。これを複数モデルで実行する。その後、Issue 文と 2 つのモデルの結果を LLM に入力し、どちらの修正が正しいかを判断させる。最後に LLM が正しいと判定した一つの修正を適用する。

5 実験

4 節で提案した手法を SWE-bench Lite で検証し、評価する。検索と修正において LLM はすべて GPT-4o (gpt-4o-2024-08-06) [2] と Claude (claude-3-5-sonnet-20241022) [3] を使用する。

5.1 実験設定

5.1.1 検索モジュール

ファイル検索は、BM25 スコアの上位 50%の入力と全ファイルパスを用いる入力の 2 種類と、LLM として GPT-4o と Claude の 2 種類で実験する。これらをそれぞれ 3 回生成し、生成した回数が 1, 2, 3 回のそれぞれで評価する (1 回の生成で 4 つの結果が得られる)。関数特定では、GPT-4o と Claude で、それぞれ 3 回ずつ生成する。ファイル検索と関数特定結果でそれぞれ和集合を取り評価する。

共通の設定条件として、結果の和集合をとる際に、生成回数に基づく閾値を設定し、その閾値以上生成されたもののみを採用する場合についても評価を行う。生成の多様性を確保するため、温度は 1 とする。評価指標はファイル検索では正しいファイル検索の再現率を、関数特定は修正すべきファイルが既知の時の再現率とする。複数関数修正する場合、それらすべてが出力に含まれている必要があるものとする。

5.1.2 修正モジュール

関数の修正を GPT-4o と Claude で 1 回ずつ行う。その後、得られた 2 つの修正の正しさを GPT-4o または Claude で判定する。また、修正や正しさの判定は多様性よりも正確性を重視させるため、温度を 0 にして生成する。正しさを判定する対象として、修正された関数ごとと、修正を統合したパッチごとの 2 種類で行う。

5.2 結果

5.2.1 検索結果

ファイル検索の結果を表 2 に示す。表 2 を見ると、生成した回数が増えるほど精度とファイル数が増加していることがわかる。ファイル数と精度の

2) https://platform.openai.com/docs/models#gpt-4o
3) https://www.anthropic.com/news/claude-3-5-sonnet

表 2: ファイル検索の結果。top50 は BM25 で得られた上位 50%のファイルパスを入力した時の結果で、all はすべてのファイルパスを入力した時の結果。Claude の結果の重みは 2 倍にしている。

生成回数	入力	再現率	平均ファイル数
1	top50	83.0%	1.78
	all	83.0%	1.80
	top50+all	85.3%	2.09
	top50+all (閾値=2)	84.0%	1.76
	top50+all (閾値=3)	79.0%	1.36
2	top50	85.3%	2.09
	all	85.3%	2.12
	top50+all	88.0%	2.51
	top50+all (閾値=2)	**87.7%**	**2.07**
	top50+all (閾値=3)	84.0%	1.72
3	top50	86.0%	2.32
	all	87.0%	2.32
	top50+all	88.6%	2.80
	top50+all (閾値=2)	88.3%	2.28
	top50+all (閾値=3)	86.3%	1.93

表 3: 修正すべきファイルが既知のときの関数の特定精度

閾値	再現率	関数の数
1	249/273 (91.2%)	1121
2	236/273 (86.4%)	782
3	**229/273 (83.8%)**	**624**
4	204/273 (74.7%)	428
5	183/273 (67.0%)	336
6	158/273 (57.8%)	255

兼ね合いを考慮し、生成した回数が 2 回の場合で、top50 と all の結果を閾値 2 でフィルタリングした結果を以降の処理で用いた。

また、SWE-bench の設定ではソフトウェアテストを実行するための test コードファイルの修正は行わないため、検索した test コードファイルは最終的な検索結果から除外した。最終的に再現率はそのままで平均ファイル数は 1.95 となった。

関数の特定結果を表 3 に示す。閾値を上げるごとに再現率と、必要と判定した関数の数が減少していることがわかる。関数数と精度の兼ね合いを考慮し、閾値 3 でフィルタリングした結果を用いて関数の修正を行った。

ファイル検索に関して和集合を取れば取るほど再現率は向上するが、各モデルで 3 回ずつ実行すると再現率の向上がほとんど見られなくなる。また、和集合を取るほどファイル数に関しても増えていくが、閾値を設定することで、再現率をある程度維持しながらファイル数を減らすことができる。

表 4: 修正結果

(a) モデル単体での結果

モデル	精度
Claude	75/300
GPT-4o	59/300

(b) 2 つの結果を LLM で判定した時の精度

判定モデル	判定手法	精度
	理想判定	88/300
GPT-4o	関数単位の判定	70/300
	パッチ単位の判定	**76/300**
Claude	関数単位の判定	72/300
	パッチ単位の判定	75/300

5.2.2 Issue 修正結果

5.2.1 節で検索したファイルから特定した関数をそれぞれ修正した結果を表 4 に示す。同表の中において、各モデルで修正を行った場合の精度を表 4a に示す。

表 4a を見ると、GPT-4o と比較して Claude の方が精度が高いことが分かる。この傾向は他の研究結果でも確認されている。

さらに、表 4a で得られた結果を LLM によって判定した際の結果を表 4b に示す。表 4b を見ると、2 つの正解の和集合を取る理想的な和集合を取った場合には精度が大きく向上するが、LLM によってどちらが良いかを判定する方法では精度の向上は見られなかった。その原因として、入力に Issue 文と対応する関数またはパッチのみを使用しており、コンテキストが不足していた可能性が考えられる。

6 おわりに

本研究では、LLM の並列処理能力とアンサンブル学習の利点に着目し、LLM エージェントを用いない新たなリポジトリレベルでのバグ修正手法を提案した。SWE-bench を用いた評価の結果、修正が必要なファイルと関数の特定において高い精度を達成したものの、バグ修正の精度においては他のシステムと比べて大きな改善は見られなかった。

今後の課題として、LLM による修正の精度を向上させるためのプロンプト設計の最適化や、関数間の依存関係を考慮した修正手法の導入が挙げられる。また、より大規模なデータセットや多様な Issue 文を対象として評価することによって、本手法の汎用性と有効性のさらなる検証を進める予定である。

謝辞

本研究は日本電信電話株式会社と早稲田大学の共同研究により実施した。

参考文献

[1] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. Swe-bench: Can language models resolve real-world github issues? In **The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024**. OpenReview.net, 2024.

[2] Siru Ouyang, Wenhao Yu, Kaixin Ma, Zilin Xiao, Zhihan Zhang, Mengzhao Jia, Jiawei Han, Hongming Zhang, and Dong Yu. Repograph: Enhancing ai software engineering with repository-level code graph, 2024.

[3] Antonis Antoniades, Albert Örwall, Kexun Zhang, Yuxi Xie, Anirudh Goyal, and William Wang. Swe-search: Enhancing software agents with monte carlo tree search and iterative refinement, 2024.

[4] John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. **CoRR**, Vol. abs/2405.15793, , 2024.

[5] Dong Chen, Shaoxin Lin, Muhan Zeng, Daoguang Zan, Jian-Gang Wang, Anton Cheshkov, Jun Sun, Hao Yu, Guoliang Dong, Artem Aliev, Jie Wang, Xiao Cheng, Guangtai Liang, Yuchi Ma, Pan Bian, Tao Xie, and Qianxiang Wang. Coder: Issue resolving with multi-agent and task graphs. **CoRR**, Vol. abs/2406.01304, , 2024.

[6] Kexun Zhang, Weiran Yao, Zuxin Liu, Yihao Feng, Zhiwei Liu, Rithesh Murthy, Tian Lan, Lei Li, Renze Lou, Jiacheng Xu, Bo Pang, Yingbo Zhou, Shelby Heinecke, Silvio Savarese, Huan Wang, and Caiming Xiong. Diversity empowers intelligence: Integrating expertise of software engineering agents. **CoRR**, Vol. abs/2408.07060, , 2024.

[7] Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Agentless: Demystifying llm-based software engineering agents. **CoRR**, Vol. abs/2407.01489, , 2024.

[8] Wei Tao, Yucheng Zhou, Yanlin Wang, Wenqiang Zhang, Hongyu Zhang, and Yu Cheng. MAGIS: LLM-based multi-agent framework for github issue resolution. In **The Thirty-eighth Annual Conference on Neural Information Processing Systems**, 2024.

[9] Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. Autocoderover: Autonomous program improvement. In **Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis**, ISSTA 2024, p. 1592–1604, New York, NY, USA, 2024. Association for Computing Machinery.

[10] Stephen Robertson and Sparck Jones. Relevance weighting of search terms. **Journal of the American Society for Information science**, Vol. 27, pp. 129–146, 05 1976.

[11] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. **Advances in Neural Information Processing Systems**, Vol. 33, pp. 9459–9474, 2020.

[12] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In **The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023**. OpenReview.net, 2023.

A　使用したプロンプト

ファイル検索とパッチ単位での正しさの判定に用いたプロンプトをそれぞれ示す。{} で囲っている部分はそれぞれタスク毎に変わる変数である。{issue} は Issue 文を表し、{file_path_list} は入力した全てのファイルパス、{patch_a},{patch_b} はそれぞれのモデルで修正をパッチ化した文字列である。

A.1　ファイル検索に用いたプロンプト

```
#Task Description
You are a programmer. You are given a GitHub issue and a list of file names from the
    corresponding codebase. Your task is to determine which files need to be modified to
     resolve the GitHub issue.

Only include files that require modifications in the output. Do not include files that
    do not require any changes.

##Output Format
The output should be in JSON format as shown below:

{{
 "need_fix": [
   "file1",
   "file2",
   ...
 ]
 }}
##Provided Information
GitHub Isasue
{issue}

List of File Names
{file_path_list}
```

A.2　パッチ単位での正しさの判定に用いたプロンプト

```
# Task Description
 You are given a GitHub issue, and two patches that may fix the issue.  Your task is to
    determine which of the two patches is the correct solution to the issue and provide
    your output.

 Below is the GitHub issue:
 {issue}

 Here is the first patch (Patch A):
 {patch_a}

 Here is the second patch (Patch B):
 {patch_b}

 ### Output Format:
 Output only A or B at the beginning of the response. NEVER include other information in
    the output.
```