

Explaining Software Bugs Leveraging Code Structures in Neural Machine Translation

Parvez Mahbub

*Department of Computer Science
Dalhousie University
Nova Scotia, Canada
parvezmrobin@dal.ca*

Ohiduzzaman Shuvo

*Department of Computer Science
Dalhousie University
Nova Scotia, Canada
oh599627@dal.ca*

Mohammad Masudur Rahman

*Department of Computer Science
Dalhousie University
Nova Scotia, Canada
masud.rahman@dal.ca*

2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE), pp. 640-652

発表者：野口 優輝

背景

ソフトウェアバグ解決

- 開発とメンテナンスの主要なタスクであり，多量の予算がかかる^[1]
- 開発者は，バグの原因を理解し，コードを修正



バグの場所を自動的に特定する研究が多くある^{[2][3]}

- 意味のある説明を提供せず，コードの特定の部分をバグとして特定
- バグを理解することは，デバッグ時間のかなりの部分を占める

開発者がバグを理解するためには，
バグの説明が必要になってくる

[1] R. Glass, "Frequently forgotten fundamental facts about software engineering," *IEEE Software*, vol. 18, no. 3, pp. 112–111, 2001.

[2] W. Zou, D. Lo, Z. Chen, X. Xia, Y. Feng, and B. Xu, "How practitioners perceive automated bug report management techniques," *TSE*, vol. 46, no. 8, pp. 836–862, 2018.

[3] M. M. Rahman, F. Khomh, S. Yeasmin, and C. K. Roy, "The forgotten role of search queries in ir-based bug localization: An empirical study," *EMSE*, vol. 26, no. 6, pp. 1–56, 2021.

バグの説明文生成

静的解析ツールでバグ検出のルール作成

- メッセージテンプレートを使って，バグを説明

課題

- 誤検出が多く，説明に実用的な洞察がされていない
- 説明は文脈を意識していないことが多い

近年の研究

バグのあるコードをソース言語とし，対応する説明をターゲット言語とする翻訳タスクとして，ニューラル機械翻訳が多く使われている

ニューラル機械翻訳(NMT)

DeepLearningに基づく自動翻訳のためのアプローチ

- エンコーダとデコーダで構成される
 - エンコーダ：ソースコードの単語を用いて数値的な中間表現を生成
 - デコーダ：中間表現に基づき，1トークンずつ生成

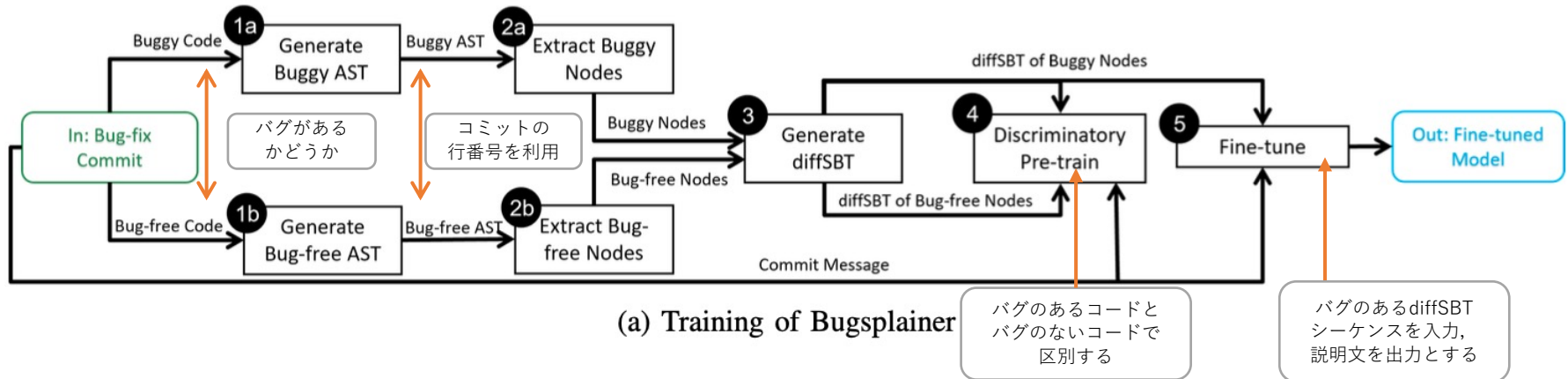
課題

1. ソースコードの構造を理解する
2. バグの多いコードパターンの理解と検出

提案手法

バグ修正コミットの大規模コーパスから学習することで，ソフトウェアバグの自然言語による説明文を生成する **Bugsplainer** を提案

Bugsplainer



- 1: ソースコードから抽象構文木 (AST) を構築する
- 2: ASTからノードを抽出 (変更された行の上下3行を抽出)
- 3: ASTノードをdiffSBTシーケンスに変換
- 4: Bugsplainerを事前学習
- 5: Bugsplainerを訓練し, 説明文を生成

diffSBTシーケンス

diffSBTアルゴリズムを使用して、バグのあるコードとバグを含まないコードから生成される

```
diff --git a/a.java b/a.java
index 51c0..3659 100644
```

```
--- a/a.java
```

```
+++ b/a.java
```

```
@@ -1,4 +1,4 @@ names_str = ""
    for name in names:
        names_str += name + ","
-   sanitize(names_str)
+   sanitize(names_str)
```

(a) Bug-fix commit diff

```
names_str = ""
for name in names:
    names_str += name + ","
    sanitize(names_str)
```

(b) Buggy code

```
names_str = ""
for name in names:
    names_str += name + ","
    sanitize(names_str)
```

(c) Bug-free code

バグ修正コミット

構造的な情報を含まない場合

pythonに従ったスコープに関連する問題であり、バグを理解することは難しい

バグありの構造
バグなしの構造

```
(Assign(Name_names_str)Name(Constant_)Constant)Assign(For(Name_name)
Name(Name_names)Name(AugAssign(Name_names_str)Name(Add)Add(BinOp
(Name_name)Name(Add)Add(Constant_)Constant)BinOp)AugAssign(Expr
(Call(Name_sanitize)Name(Name_names_str)Name(Call)Expr)For
<s>
(Assign(Name_names_str)Name(Constant_)Constant)Assign(For(Name_name)
Name(Name_names)Name(AugAssign(Name_names_str)Name(Add)Add(BinOp
(Name_name)Name(Add)Add(Constant_)Constant)BinOp)AugAssign(For
(Expr(Call(Name_sanitize)Name(Name_names_str)Name(Call)Expr)
```

diffSBTシーケンスを用いた場合

構造的な情報を含む場合

コード構造の違いを特定することができ、バグを説明するのに役立つ可能性がある

研究目的

Bugsplainerが生成したバグの説明文が，有用であるかどうかを明らかにする

Research Question

RQ1：Bugsplainerはバグの説明に対しどのような役割があるか

RQ2：構造情報と事前訓練はBugsplainerにどのような影響を与えるか

RQ3：Bugsplainerは自動評価の観点でベースラインを上回れるか

RQ4：生成されたバグの説明はどの程度正確か？

データセット

リポジトリの選択

- pythonのスター数が300以上の合計10154件リポジトリを収集

バグ修正コミットの収集

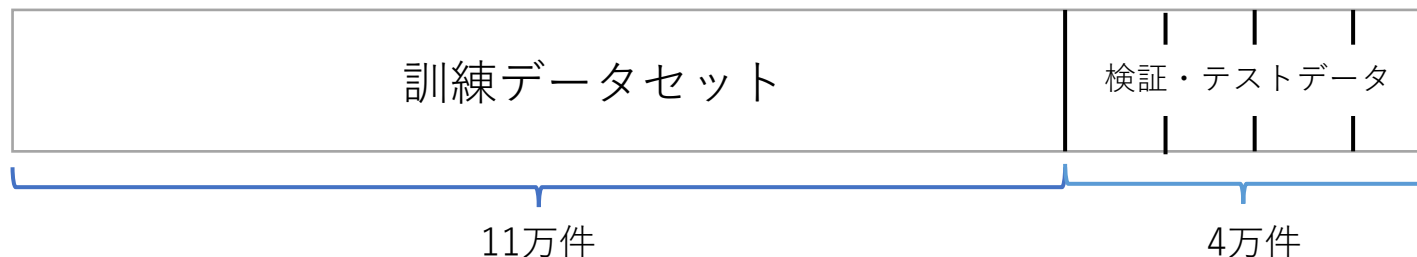
- リポジトリから合計1180万件のコミットを収集

フィルタリング

- コミットメッセージに「fix」, 「solve」を含むコミット
- 構文的に正しくないプログラムを破棄

15万件のdiffSBT
シーケンスを取得

訓練データとテストデータの構築



RQ1 | Bugsplainerパフォーマンス（分析）

モデル

- Bugsplainerとパラメータ数を増やしたBugsplainer 220Mを用いる

データセット

- random split：訓練・検証・テストをランダムに分割
- No CodeSearchNet Repository：テストインスタンスを破棄
- Cross-project：様々なプロジェクトのコミットを含む

評価指標

- BLUEスコア，意味的類似度，完全一致度で評価

RQ1 | Bugsplainerパフォーマンス（結果）

Bugsplainerは理解しやすく，優れたバグ説明を生成することができ，パラメータの数が多いほど，複数のプロジェクトにわたって一般化できる

Model	Dataset	BLEU	Semantic Similarity	Exact Match
Bugsplainer	Random split	33.15	55.76	22.37
	No CodeSearchNet Repository	34.53	56.67	19.55
	Cross-project	17.16	44.98	7.15
Bugsplainer 220M	Random split	33.87	56.35	23.50
	No CodeSearchNet Repository	35.59	57.29	20.74
	Cross-project	23.83	49.00	15.47

RQ3 | 既存技術を上回れるか（分析）

バグを自然言語で説明する技術は存在しない

→コミットメッセージ生成技術をベースラインとして使用

—CommitGen^[22], NNGen^[25], Fine-tuned CodeT5^[26], Pyflake^[9]と比較

既存手法はバグを含むコードとバグを含まないコードを区別して学習されていない

⇒ バグのあるコードに対して、
意味のある説明を生成する可能性が高い

評価指標

—BLUEスコア, 意味的類似度, 完全一致度で評価

[22] S. Jiang, A. Armaly, and C. McMillan, "Automatically generating commit messages from diffs using neural machine translation," in *2017 32nd ASE*, IEEE, 2017, pp. 135–146.

[25] Z. Liu, X. Xia, A. E. Hassan, D. Lo, Z. Xing, and X. Wang, "Neural-machine-translation-based commit message generation: How far are we?" In *Proc.33rd ASE*, 2018, pp. 373–384.

[26] Y. Wang, W. Wang, S. Joty, and S. C. Hoi, "Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation," in *Proc.2021 EMNLP*, 2021, pp. 8696–8708.

[9] A. Sottile, *pyflakes: A simple program which checks Python source files for errors*, 2022. [Online]. Available: <https://github.com/PyCQA/pyflakes> (visited on 01/18/2022).

RQ3 | 既存技術を上回れるか（結果）

Bugsplainerは全ての指標において、ベースラインよりも優れていることが明らかになった

Technique	BLEU	Semantic Similarity	Exact Match
pyflakes	0.49	5.68	0.00
CommitGen	9.94	35.39	1.04
NNGen	24.16	47.33	14.17
Fine-tuned CodeT5	26.19	54.52	8.85
Bugsplainer*	32.90	55.22	18.14

統計的に有意か？

マン・ホイットニーのU検定で、有意差を検定
→Fine-tuned CodeT5とBugsplainerは有意差があり、
Bugsplainerは高いパフォーマンスを示すことがわかった

RQ4 | 説明文はどの程度正確か（分析）

Pythonプログラミング経験を持ち，英語を話せるソフトウェアエンジニアを対象に，生成されたバグの説明文を評価

- ソフトウェア開発経験：1~10年
 - バグ修正経験：1~7年
- } 20名の回答を得た

分析

- 15件のバグ修正コミットに取り組む
- 生成された説明文のAccurate, Precise, Concise, Usefulを5段階のリッカード尺度で評価

既存技術と比較して，Bugsplainerはどの程度正確かを調査

RQ4 | 説明文はどの程度正確か（結果）

リッカード尺度

参照文と同じ事実
を説明しているか

コードの問題を
突き止めているか

短く、文を伝えているか

説明文はバグを修正する
可能性があるか

Quality	Model	平均値	中央値	最頻値	2 nd Mode
		Mean	Median	Mode	
Accurate	pyflakes	1.841	1	1	3
	CommitGen	2.176	2	1	2
	NNGen	2.653	3	1	4
	Fine-tuned CodeT5	2.842	3	4	3
	Bugsplainer	4.074	4	5	4
Precise	pyflakes	1.768	1	1	3
	CommitGen	1.884	2	1	2
	NNGen	2.529	2	1	2
	Fine-tuned CodeT5	2.772	3	4	2
	Bugsplainer	3.891	4	4	5
Concise	pyflakes	2.182	2	1	4
	CommitGen	2.350	2	1	2
	NNGen	2.881	3	4	1
	Fine-tuned CodeT5	3.044	3	4	3
	Bugsplainer	3.974	4	4	5
Useful	pyflakes	1.724	1	1	3
	CommitGen	1.960	2	1	2
	NNGen	2.576	2	1	4
	Fine-tuned CodeT5	2.728	3	4	1
	Bugsplainer	3.923	4	4	5

まとめ

バグの説明を，ニューラル機械翻訳の技術を利用した Bugsplainer を提案

- BLUEスコア，意味的類似度，完全一致度で評価

結果

- Bugsplainerは全てのベースラインを上回る性能だった
- 生成される説明文に関しても，既存技術のモデルより Accurate, Precise, Concise, Usefulの観点で上回っていた

今後の課題

- バグのあるコードとないコードの構造的違いをうまく活用して調査

所感

diffSBTシーケンスについてはよくわからなかった.

RQ4での評価対象は、開発経験が1年の人も対象に含めても大丈夫なのか.

—もう少しベテランの開発者じゃなくていいの？

補足 (diffSBTアルゴリズム)

Algorithm 1 Generate diffSBT sequence from commit diff

```
1: function DIFFSBT(c) ▷ Generate diffSBT sequence for commit
2:   buggyAST ← BUILDAST(c.buggyCode)
3:   bugfreeAST ← BUILDAST(c.bugFreeCode)
4:   buggyNodes ← INTERSECTIONS(buggyAST, c.removed)
5:   bugfreeNodes ← INTERSECTIONS(bugfreeAST, c.added)
6:   return SBT(buggyNodes) +  $\langle /s \rangle$  + SBT(bugfreeNodes)
7: end function

8: function INTERSECTIONS(r, ln)
9:   nodes ←  $\phi$  ▷ Initialize nodes with an empty list
10:  for all n in r do ▷ Get intersections for all nodes in r
11:    if ISINSIDE(n, ln) or ISEXPRESSION(n) then
12:      APPEND(nodes, n)
13:    else if STARTSINSIDE(r, ln) then ▷ Keep node r but
    prune the children outside ln
14:      r.children ← INTERSECTIONS(r.children)
15:      APPEND(nodes, n)
16:    else if ENDSINSIDE(r, ln) then ▷ Node r starts before
    the ln. Return only the children of r that intersect with ln.
17:      children ← INTERSECTIONS(r.children)
18:      APPEND(nodes, children)
19:    end if
20:  end for
21:  return nodes
22: end function
```
