集合知を用いた深層学習による自動バグ修正

高橋 裕太[1,a)]　鵜林 尚靖[1,b)]　佐藤 亮介[1,c)]　亀井 靖高[1,d)]

概要：ソフトウェア工学の分野において自動でバグを修正する研究が盛んに行われている．deepfix では C 言語の構文的エラーに対して，深層学習による自動バグ修正の手法を提案している．本論文では deepfix を用いて，バグの対象を Android SDK API に関するものに変更し，自動バグ修正が行えるかどうかを調査する．我々は Q&A サイトや OSS リポジトリから Android API に関するバグ修正の情報を収集し，自動バグ修正モデルの訓練データセットを作成，deepfix で用いられた sequence-to-sequence ニューラルネットによる学習を行った．実験の結果，修正すべき API の特定に一部成功した．現時点では完全な自動修正は難しいが，少なくとも修正すべき API 候補の推薦等には利用が期待できることが分かった．

キーワード：自動バグ修正，深層学習，集合知，Q&A サイト，OSS リポジトリ，Android SDK API

1. はじめに

ソフトウェア開発において，バグ修正は時間を要し開発者にとって大きな負担である．デバッグ作業がソフトウェアの開発コスト全体の 50 %を占めるとも言われている [3]．そのためバグ修正のコストを削減することを目的に，自動バグ修正の研究が盛んに行われている．Luca らによる自動バグ修正に関するサーベイ論文 [7] によると年々自動バグ修正に関する論文の投稿が増えており研究者の関心が集まっていることが分かる．

Rahul らは自動バグ修正に深層学習を用いる手法 deepfix を提案した [8]．Deepfix では，初歩的な C 言語プログラムの構文的エラーによるバグを対象にしている．彼らはより困難なエラーに対しても deepfix の手法が適用できる可能性について言及しているが，実際に適用対象を変更した追実験は実施されていない．そのため，deepfix に代表される深層学習による自動バグ修正が，構文エラー以外の一般的なバグに対してどの程度の修正能力を持つかは明らかになっていない．本論文では，最初の一歩として，API プログラミングにおけるバグ修正に deepfix を適用する．現在のソフトウェア開発において API は極めて重要な役割を果たしており，一般的なバグの事例として API プログラミングを取り上げることは研究的にも興味深いと思われる．

本論文では，Android SDK の API メソッドを用いたプログラムを対象に deepfix による自動バグ修正を試みる．また，deepfix の適用対象を「C 言語における初歩的構文エラー」から「API の使用方法という論理的エラー」に広げた時にどの程度の修正能力があるのかについても実証的に評価する．Android SDK のドキュメントは一般的に不足しているという指摘が存在し [10]，また Q&A サイトである Stack Overflow[*1] に寄せられた投稿のうち，プログラミング言語を除けば「android」というタグが付けられた投稿件数が最も多く，このことからも Android SDK に関する情報の需要は高いことが分かる．Deepfix の適用対象としては妥当な選択と言える．

我々は自動バグ修正モデルを深層学習により作成するに当たって，データセットとなる Android SDK API に関するバグ修正情報を集合知から収集する．集合知とは Stack Overflow に代表される Q&A サイトや，OSS リポジトリに開発者によって集められた知識のことである．本実験では Stack Overflow に寄せられた android タグをもつ投稿と，F-Droid[6] と呼ばれるオープンソースのアプリケーションストアから訓練のためのバグ修正情報を抽出し，データセットを作成する．先に述べたように，Stack Overflow には多くの android に関する投稿が集まっており，その中には Android SDK API メソッドに関するバグ修正の情報も多く存在していると予想できる．また OSS リポジトリは実際のアプリケーションを開発した際のバグの修正情報が存在している．そのため OSS リポジトリのソースコード

[1] 九州大学
Kyushu University
[a)] takahashi@posl.ait.kyushu-u.ac.jp
[b)] ubayashi@ait.kyushu-u.ac.jp
[c)] sato@ait.kyushu-u.ac.jp
[d)] kamei@ait.kyushu-u.ac.jp

[*1] https://stackoverflow.com/

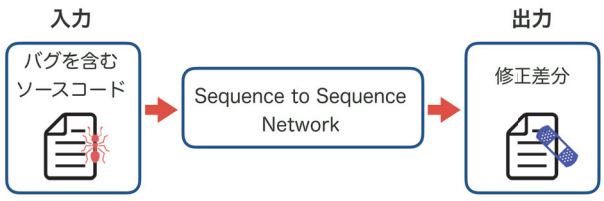

図 1　作成する学習モデル

を元に自動バグ修正モデルを作成することで，実際の開発者に受け入れられる修正の出力が期待できる．実際に OSS リポジトリを用いた自動バグ修正の研究は行われており，既存の手法から性能の向上に成功している [11][15]．以上が Stack Overflow と OSS リポジトリから Android SDK に関する修正情報を取得する理由である．今回の実験では，入力としてバグを含むソースコードを受け取り，修正のための差分情報を出力するモデル（図 1）を作成する．

Deepfix の学習手法を API のバグに適用させるに当たって，いくつかの疑問が考えられる．一つは，deepfix で用いられた C 言語の構文エラーに関する修正情報から，集合知から集めたバグ修正情報に訓練データを変更することで，Android SDK API メソッドの使用方法に関するバグを修正するような自動バグ修正モデルは作成が可能であるのかという点である．そして可能であるならば，どのようなデータセットによりモデルの訓練を行うことで性能が向上するのかという考察が必要と考える．こうした動機から，以下の二つの Research Question を設定した．

RQ1. 集合知から作成したデータセットにより，自動バグ修正モデルが作成できるか

Deepfix で用いられた C 言語のプログラムと，今回対象とする Android SDK に関するソースコードは，プログラムの長さそのものや，バグの種類，修正内容に違いがあると考える．そのため C 言語プログラムから訓練データセットを変更し，モデルの学習を行う上での課題の考察や，バグ修正の能力を評価する必要がある．

RQ2. 学習データセットの違いが，修正結果にどのような差を生むか

今回用いる Stack Overflow と OSS リポジトリのコード差分間で学習結果にどのような差異が生まれるかを比較する．また集合知として集められたソースコードには，よく用いられる API とそうでない API が存在している．自動バグ修正モデルを作成する際，訓練に用いるデータセットによって結果が変わることが想定できる．Android API メソッドの数は 180,000 を超えており，すべてを対象にバグ修正を試みるよりも，対象とする API を絞った方が性能がよくなると考えられる．そのため頻繁に用いられる API を対象にデータセットを作成した場合とそれ以外のデータセットを用いて学習を行った結果，性能に影響が出るかどうかを調査する．

以降，2 節では関連研究について紹介する．3 節ではデータセットの入手について述べる．4 節では実験について述べる．5 節では実験の結果と考察を述べる．6 節では妥当性への脅威について述べ，7 節ではまとめを述べる．

2. 関連研究

本節では，自動バグ修正に関する関連研究について述べる．そして本論文で用いる deepfix についてと，そこで用いられた基礎技術となる sequence-to-sequence ニューラルネットワークについて説明する．また本論文で対象とする集合知をデバッグ支援へ利用した事例を紹介する．

2.1 自動バグ修正

ソフトウェア開発においてバグ修正は非常にコストの高い工程である．そうした背景からバグを自動で修正する研究は盛んに行われている．自動バグ修正は欠陥箇所を特定し，修正の生成・適用を繰り返しテストケースをパスさせる，という過程から成り立つ．修正の生成の方法は原始的な変更による修正とテンプレートを用いた修正に分けられる [7]．原始的な変更による修正はプログラム中の演算子を変換することで修正を試みる手法で，代表的なものに遺伝的アルゴリズムを用いる GenProg[12] や GenProg を派生させた RSRepair[17] などが上げられる．テンプレートを用いた修正はあらかじめ定義してある修正テンプレートを用いることでバグ修正を試みる手法で，人が作成したパッチから抽出したパターンを用いて修正を行う PAR[9] といった手法がある．

本論文で用いる sequence-to-sequence によるバグ修正は，モデルに修正パターンの情報を蓄積することになるためテンプレートを用いた修正に分類することができる．

2.2 Sequence-to-sequence モデル

sequence-to-sequence ニューラルネットワークモデル [19] は，エンコーダとデコーダと呼ばれる二つの Recurrent Neural Network(RNN) から成るモデルである．エンコーダはシーケンスを入力に取り単一のベクトルを出力し，デコーダがエンコーダが出力したベクトルを読み，出力シーケンスを生成する．sequence-to-sequence モデルは長さ n のシーケンスを入力に受け取ると，長さ m のシーケンスを出力することができる．そのため文章の翻訳などに応用されている．本論文で目的とする自動バグ修正というコンテキストにおいての入出力は，入力がバグを含むソースコード，出力が修正後のソースコード (もしくは修正のための操作) に対応させることができる．

また sequence-to-sequence モデルはシーケンスが長くなると，始めの方に入力されたデータがエンコーダが作成する特徴ベクトルに反映されにくくなるという特徴がある．この課題を解決するために attention モデル [1] と呼ばれる

メカニズムが提案されている．attention はデコーダでのシーケンス予測時に，入力シーケンスのどの部分に注目をするかを考慮する手法である．これにより文章翻訳における指示語などの前に出現した単語を示す言葉などを対応させることができる．ソースコードの学習においては，} といったスコープの終わりを示すトークンを，前に出現した { に対応させることができる，といった効果が期待できる．

2.3 Deepfix

Rahul らは deepfix と呼ばれる sequence-to-sequence ニューラルネットと attention を用いた C 言語の自動バグ修正手法を提案している [8]．彼らは 93 個の C 言語のプログラミングタスクに対して，プログラミング入門コースの学生が作成したプログラムにこの手法を適用している．6,971 件のエラープログラムのうち，1,881(27%) 件の修正に成功し，1,338(19%) 件を部分的に修正した．deepfix ではプログラム中に複数のエラーが含まれている場合でも，繰り返し修正を行うことで全てのエラーの修復を可能にする．

また彼らは正しいプログラムからエラーを含むプログラムを 9,230 件生成し，これらに対しても修正を適用している．結果として 5,185(56%) 件の修正に成功している．

彼らの提案手法はプログラミング言語に依存するものではないため，本論文で対象とする Java プログラムにも適用可能である．また彼らは C 言語の構文的エラーのみを対象としていたが，API の使用方法といった論理的エラーに対しても学習のフェイズはそのまま適用できる．そこで我々は自動バグ修正のモデルの学習に deepfix の学習フェイズの手法を用いる．

2.4 集合知を用いたデバッグ支援

Stack Overflow は 2019 年現在 17,000,000 件を超える投稿があり，バグ修正に役立つ記事が多く存在する．そうした背景から Stack Overflow をバグ修正に役立てる研究も行われている．Chen らの研究では Stack Overflow の質問投稿と回答投稿に含まれるコードと，OSS に蓄積されたソースコードとの間のコードクローンを検出することによりバグを検出する手法を提案している [4]．Liu らの研究では Stack Overflow から修正テンプレートを作成する手法を提案している [13]．このように Stack Overflow に着目した研究があり，バグ修正を手助けする情報が多く含まれていることが期待できる．

また github のような OSS リポジトリには，開発者によるコード差分が多く蓄積されている．その中にはプログラム中に含まれるバグに修正を行ったコード差分の情報が含まれている．Long らは GenProg に代表されるような，演算子を変更してテストケースをパスさせる自動バグ修正手法において，実際の開発者が行わないような無意味な修正を行うことがあるという課題を解決するため，OSS リポジトリの開発履歴に含まれるバグ修正の情報を用いた自動バグ修正手法を提案した [15]．彼らはテストケースをクリアするパッチを生成したのち，実際に適応させるパッチの順位付けを確率モデルを用いて行った．この確率モデルの作成に，OSS から抽出した修正パッチを用いている．結果として Long らが過去に提案したパッチ生成システム [14] から，正しいパッチを生成する能力の向上に成功している．Le らの研究においても OSS プロジェクトでのバグ修正パターンを用いて修正候補のランク付けを行い，修正パッチの生成時間の向上に成功している [11]．これらの研究から，OSS リポジトリに蓄積されているバグ修正の情報は，実際の開発者が作成した修正であるという性質により，開発者が受け入れるパッチを生成するシステムへの利用価値が高いと言える．

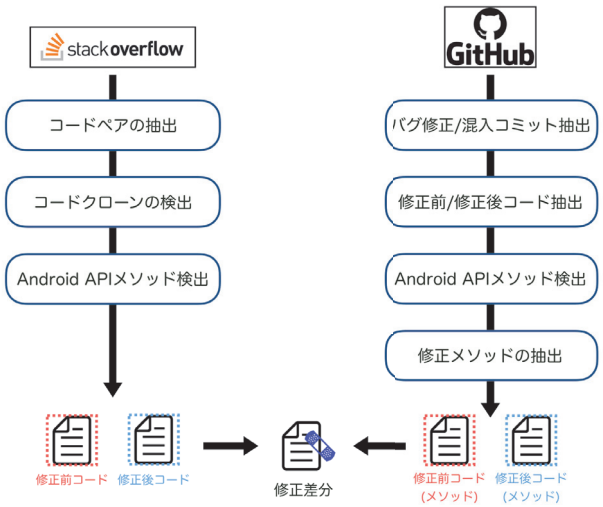

図 2　データセット入手概要

以上より，Stack Overflow と OSS リポジトリのソースコードは共にデバッグ支援の研究に役立てられており，本論文においても集合知の利用によって API メソッドに関する自動バグ修正への期待ができると考える．

3. データセットの準備

集合知を用いて sequence-to-sequence ネットワークモデルを学習させるに当たって，Stack Overflow の投稿中に含まれるバグ修正に関するソースコード，github で公開されている OSS リポジトリからバグ修正に関するコード差分をそれぞれ抽出する必要がある．本節ではデータセット作成までの手法を説明する．データセット作成の概要を図 2 に示す．

次節ではデータセット作成時に必要となる，本実験で対象とする Android SDK の抽出について説明する．

表 1　対象とする Android SDK

プラットフォーム	パッケージ	クラス	メソッド
2（android，androidx）	240	3,972	129,912

3.1　対象とする Android SDK の抽出

今回対象とする Android SDK の API は，Android Developer で公開されているページをクローリングし得られた計 3,777 クラスのメソッド 183,068 個を対象とする (表 1)．ただし，android.util.Log クラスのメソッドは，ログ出力に関するメソッドであり，修正するバグの本質に関係がないと考えられるため除外した．

3.2　Stack Overflow

本節では Stack Overflow 投稿中の文章から，どのようにバグ修正情報を含んだソースコードの抽出を行うかを説明する．なお今回は Stack Exchange が公開している Stack Overflow 投稿のデータ [*2] を用いる．投稿の期間は 2008/08/07〜2017/03/13 である．そこから Android タグに関する投稿 984,533 件を抽出したものを対象とする．

3.2.1　ソースコードペアの抽出

Stack Overflow の投稿は質問投稿と回答投稿から成っており，また本文は自然言語とソースコードで構成されている．ソースコードを抜き出すために，投稿ページの HTML ソース中の<code>タグを利用する．本手法では<code>タグに囲まれている文章をソースコードとみなす．質問投稿とその質問に対する回答投稿に含まれるソースコードを抜き出し，ペアを作成する．なお質問投稿一つに対して複数の回答投稿が存在する場合もあるため，ソースコードを含む回答それぞれにつきペアを作成する．

3.2.2　コードクローンの検出

質問投稿と回答投稿からソースコードをそれぞれ抽出しただけでは，後者が前者に変更を加えたソースコードかどうかは分からない．そこで作成したペア全てに対し，コードクローン (type3) の検出を試みる．type3 は文の挿入，削除，変更が行われたコードクローンであり [2]，差分を含む二つのソースコード間でクローンを検出できる．よって type3 コードクローンが検出されれば，二つのソースコードは類似していてかつ差分により互いに変換に可能となる．見つかったクローンペアの質問投稿側をバグ修正前のソースコード，回答投稿側をバグ修正後のソースコードと仮定する (図 3)．Type3 コードクローンの検出には scorpio[20] を用いる．なお今回は最低 4 行一致しているコードクローンを検出する．一致行数が少なすぎる場合，メソッドの return 部とその次の行の } など，一般的なコードに対してクローンを検出してしまうため，この値を設定した．

なお，type3 コードクローンの検出を行うためには，ソースコードがコンパイル可能である必要がある．しかし Stack Overflow 投稿に含まれるソースコードはメソッドのみが記述されている場合等，コンパイル可能なソースコードであるとは限らない．そこでまず抽出したソースコードペア全てに対して，コンパイルを試みる．その後コンパイル不可能なソースコードに対して，テンプレートコードに挿入することによりコンパイルを試みる．図 4 に今回用いるテンプレートコードを示す．Scorpio ではコンパイル可能なもののみがふるい分けられるため，投稿中の<code>タグ内に今回対象とする Java 言語以外のソースコードが含まれていたとしても問題なくクローンペアの抽出が行える．

*2　https://archive.org/details/stackexchange

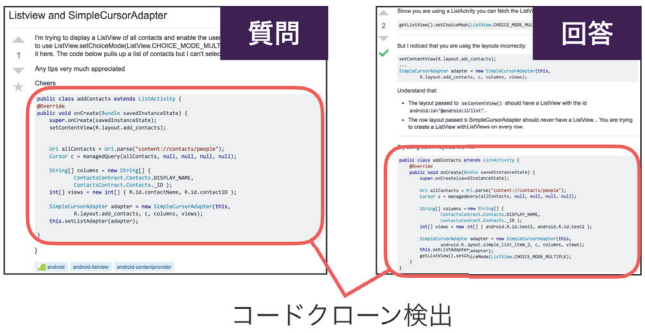

図 3　質問と回答投稿の間のコードクローン検出 [*3]

```
public class Main {
    public static void main(String args[]) {
        /*insert here*/
    }
}
```

図 4　テンプレートコード

表 2　Stack Overflow から抽出したコード差分の件数

クローンが検出されたコードペア	7,583
API が検出されたコード差分	2,210

3.2.3　コード差分の作成と Android API 検出

コードクローンが検出され得られたソースコードのペアの差分情報を作成する．コード差分は Linux の diff コマンドを用いて作成する．コード差分作成の段階で，差分の変更箇所に Android SDK のメソッドが用いられているかどうかを判定し，含まれないものは除外する．またコメントはこの時点で削除する．Android SDK メソッドの検出には正規表現を用い，アプリケーションパスが android で始まる API メソッドが検出されたコード差分を抽出した．Stack Overflow からコードクローンが検出された質問投稿と回答投稿のペアの件数，そこから Android API が検出された得られたコード差分の件数を表 2 に示す．

3.3　OSS リポジトリ

OSS リポジトリの入手のために，F-Droid リポジトリ [6] を用いる．F-Droid はオープンソースの Android アプリ

ケーションパッケージを提供するアプリケーションストアであり，開発ソースコードを公開しているページへのリンクが存在する．F-Droid で公開されている 1,380 件 (2017/12/19) のプロジェクトのうち，ソースコードが Git で管理されていてかつ Java のソースコードを持つ 713 件のプロジェクトを抽出し利用する．対象となる総コミット数は 1,144,866 件である．

3.3.1 ソースコードの抽出

入手した OSS リポジトリのソースコード差分から，以下の条件を満たすコード差分を抽出する．

(1) バグ修正が行われたコード差分

(2) Android SDK メソッドの呼び出しがあるコード差分

バグ修正が行われたコミットを特定するため，コミットメッセージにバグを修正したコミットを特定し，SZZ アルゴリズム [18] を用いてバグが混入したソースコードを特定する．SZZ アルゴリズムは，バージョン管理システムにより記録されたバグ報告メッセージと，ソースコードの差分の履歴から，バグが混入したコミットを特定する方法である．本論文では，「コミットメッセージに”fix bug”もしくは”resolve bug”を含む」もしくは「GitHub で管理されている，”bug”タグの付いた課題番号をコミットメッセージに含んでいる」のどちらかの条件を満たすコミットをバグ修正コミットとする．このバグ修正コミットに SZZ アルゴリズムを適用し，バグが混入したコミットを特定した．このようにしてバグが混入したコミットと，それが修正されたコミットのソースコードをそれぞれ入手した．

3.3.2 コード差分が含まれるメソッドの抽出

バグが混入したコミットとそれが修正されたコミットのそれぞれのソースコードファイルをそのまま用いると，ソースコードの長さが非常に長くなってしまい，学習に不向きである．そのため，コードが変更された行を特定し，その変更行が含まれるメソッドのみを抽出する (図 5)．これにより入力ソースコードの長さを飛躍的に短くすることが可能になる．またメソッドのみを抜き出した場合，メソッドが定義されているクラスの情報が欠落するため，抽出されたメソッドが定義されたクラスの定義行と対応する } で，抽出されたメソッドをラッピングする．変更行が複数のメソッドに存在する場合は，メソッドごとに別のデータセットとして作成する．また変更行がどのメソッドの内部にも含まれていない場合，「新たなメソッドの定義」「既存のメソッドの修正」「既存のメソッドの削除」変更であるとみなし，今回は対象外とする．

こうして得られたメソッドごとの変更前，変更後のソースコードのコード差分情報を，Stack Overflow のデータセットと同様に Linux の diff コマンドで作成した．OSS リポジトリから得られたコード差分件数と，差分が検出されたソースから抽出したメソッドごとのコード差分件数を表 3 に示す．

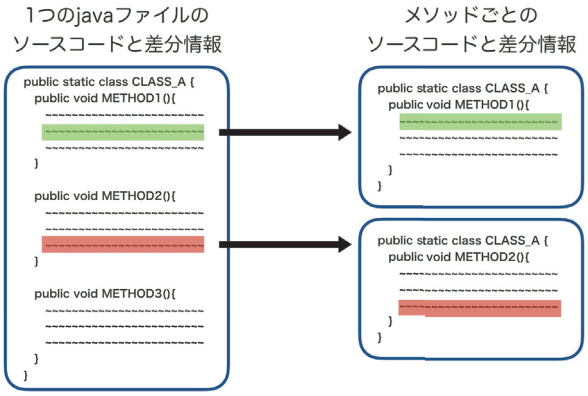

図 5 メソッドごとのソースコードと差分情報の抽出

表 3 OSS から抽出したコード差分の件数

API が検出されたコード差分件数	11,546
メソッドごとのコード差分件数	37,066

3.4 得られたデータセット

コード差分の最終的な件数は，Stack Overflow から 11,546，OSS からは 37,066 である．得られたデータセットのうち，バグを含むソースコードの例を図 6 に，差分情報の例を図 7 に示す．前者を整形して学習モデルの入力データに，後者を整形したものを出力データとする．データの整形は次節で説明する．

4. 実験

4.1 学習モデル

今回の実験で作成する学習モデルについて説明する．入力としてバグを含むソースコードを受け取り，修正のための差分情報を出力するモデルを作成する．入力のソースコードは各行の先頭に行番号を含ませたものとする．出力は差分情報についてコードの追加を<add>，コードの削除を<delete>，コードによる行の変更を<change>と表現したものとする．学習モデルの作成に用いる入出力の具体例を図 8，図 9 に示す．

4.2 データセットの分類

Rahul らの実験では，C 言語のある課題について，複数のバグを含むプログラムのデータを用いて学習を行なっていた．対して今回集めた Stack Overflow と OSS リポジトリのコード差分は，ある一つのバグに対して多くの修正データを集められていないと考えられる．そのため，作成する学習モデルが修正できる問題空間を狭めることで修正の精度向上を試みる．我々は Stack Overflow と OSS リポジトリから得られた Android SDK メソッドを含むコード差分について，頻繁に使用される API メソッドが含まれるものとそうでないものに分類する．今回は Stack Overflow と OSS リポジトリから得られたコード差分それぞれにつ

```
public static class ListViewFragment extends Fragment {
  @SuppressLint("InflateParams") @Override public View onCreateView(  LayoutInflater inflater,
  ViewGroup container,  Bundle savedInstanceState){
    View root=inflater.inflate(R.layout.fragment_listview,container,false);
    ListView list=(ListView)root.findViewById(android.R.id.list);
    ListViewAdapter listAdapter=new
    ListViewAdapter(getActivity(),getResources().getStringArray(R.array.countries));
    list.setAdapter(listAdapter);
    FloatingActionButton fab=(FloatingActionButton)root.findViewById(R.id.fab);
    fab.attachToListView(list);
    FloatingActionButton fab2=(FloatingActionButton)root.findViewById(R.id.fab2);
    fab2.attachToListView(list);
    return root;
  }
}
```

(https://stackoverflow.com/questions/31948678/multiple-buttons-in-floatingactionbutton-library より)

図 **6**　入手したバグを含むソースの例

```
1c1,2
< public static class ListViewFragment extends Fragment {
---
> public static class ListViewFragment extends Fragment implements com.melnykov.fab.ScrollDirectionListener {
>   private FloatingActionButton fab, fab2;
7,10c8,11
<     FloatingActionButton fab=(FloatingActionButton)root.findViewById(R.id.fab);
<     fab.attachToListView(list);
<     FloatingActionButton fab2=(FloatingActionButton)root.findViewById(R.id.fab2);
<     fab2.attachToListView(list);
---
>     fab=(FloatingActionButton)root.findViewById(R.id.fab);
>     fab.attachToListView(list,this);
>     fab2=(FloatingActionButton)root.findViewById(R.id.fab2);
>     fab2.attachToListView(list,this);
11a13,24
>   }
>   @Override public void onScrollDown(){
>     if (fab.isVisible() || fab2.isVisible()) {
>       fab.hide();
>       fab2.hide();
>     }
>   }
>   @Override public void onScrollUp(){
>     if (!fab.isVisible() || !fab2.isVisible()) {
>       fab.show();
>       fab2.show();
>     }
```

(https://stackoverflow.com/questions/31948678/multiple-buttons-in-floatingactionbutton-library より)

図 **7**　抽出した差分情報の例

```
0 ~  public static class ListViewFragment extends Fragment {
1 ~  @ SuppressLint ( "StringLiteral" ) @ Override public View onCreateView ( LayoutInflater
Var_LayoutInflater_Var , ViewGroup Var_ViewGroup_Var , Bundle Var_Bundle_Var ) {
2 ~  View Var_View_Var = Var_LayoutInflater_Var . inflate ( R . layout . fragment_listview ,
Var_ViewGroup_Var , false ) ;
3 ~  ListView Var_ListView_Var = ( ListView ) Var_View_Var . findViewById ( android . R . id . list ) ;
4 ~  ListViewAdapter Var_ListViewAdapter_Var = new ListViewAdapter ( getActivity ( ) , getResources ( )
. getStringArray ( R . array . countries ) ) ;
5 ~  Var_ListView_Var . setAdapter ( Var_ListViewAdapter_Var ) ;
6 ~  FloatingActionButton Var_FloatingActionButton_Var = ( FloatingActionButton ) Var_View_Var .
findViewById ( R . id . fab ) ;
7 ~  Var_FloatingActionButton_Var . attachToListView ( Var_ListView_Var ) ;
8 ~  FloatingActionButton Var_FloatingActionButton_Var = ( FloatingActionButton ) Var_View_Var .
findViewById ( R . id . fab2 ) ;
9 ~  Var_FloatingActionButton_Var . attachToListView ( Var_ListView_Var ) ;
10 ~  return root ;
11 ~  }
12 ~  }
```

(https://stackoverflow.com/questions/31948678/multiple-buttons-in-floatingactionbutton-library より)

図 **8**　入力データの例

```
<change> 1 public static class ListViewFragment extends Fragment implements com . melnykov . fab . ScrollDirectionListener { private FloatingActionButton
NoType_Var , NoType_Var ; <end>

<change> 7,10 NoType_Var = ( FloatingActionButton ) View_Var . findViewById ( R . id . fab ) ; NoType_Var . attachToListView ( ListView_Var , this ) ; NoType_Var =
( FloatingActionButton ) View_Var . findViewById ( R . id . fab2 ) ; NoType_Var . attachToListView ( ListView_Var , this ) ; <end>

<add> 11 } @ Override public void onScrollDown ( ) { if ( NoType_Var . isVisible ( ) || NoType_Var . isVisible ( ) ) { NoType_Var . hide ( ) ; NoType_Var . hide ( ) ; } } @
Override public void onScrollUp ( ) { if ( ! NoType_Var . isVisible ( ) || ! NoType_Var . isVisible ( ) ) { NoType_Var . show ( ) ; NoType_Var . show ( ) ; } <end>
```

(https://stackoverflow.com/questions/31948678/multiple-buttons-in-floatingactionbutton-library より)

図 **9**　出力データの例

表 4　Stack Overflow から得られたコード差分に含まれる Android SDK メソッド上位 15 件

クラス. メソッド	コード差分の数
Activity.findViewById	636
View.findViewById	379
TextView.setText	308
LayoutInflater.inflate	220
Button.setOnClickListener	92
Intent.putExtra	76
Context.getSystemService	65
Paint.setColor	58
ListView.setAdapter	53
ImageView.setImageResource	41
View.setTag	40
Bundle.get	34
MediaPlayer.start	34
SharedPreferences.edit	34
View.getTag	34

表 5　OSS リポジトリから得られたコード差分に含まれる Android SDK メソッドの上位 15 件

クラス. メソッド	コード差分の数
Toast.makeText	594
Intent.putExtra	561
TextView.setText	203
SharedPreferences.getBoolean	187
PreferenceManager.getDefaultSharedPreferences	180
ContentValues.put	139
Uri.parse	107
View.findViewById	98
PendingIntent.getActivity	98
LayoutInflater.from	91
SharedPreferences.getString	89
Window.setStatusBarColor	88
DialogInterface.dismiss	86
SharedPreferences.edit	82
LayoutInflater.inflate	75

いて，API メソッドの出現頻度分布を作成し，出現頻度が上位 20%の API メソッドを含む Stack Overflow/OSS リポジトリコード差分，それ以外の Stack Overflow/OSS リポジトリコード差分の 4 つのデータセットに分類した．以降，それぞれのデータセットを「SO high rank」「SO low rank」「OSS high rank」「OSS low rank」と呼ぶ．Stack Overflow コード差分，OSS リポジトリコード差分それぞれの API メソッドの分布上位 15 件を表 4，表 5 に示す．

4.3　データ整形

4.3.1　変数名の一般化

データセットのソースコードに対して，変数名を型情報を保持して一般化する．例えば **ListView foobar;** といった変数宣言があった場合，**ListView ListView_Var;** というように変数名を一般化する．

4.3.2　入出力データの作成

変数名の一般化を行ったバグを含むソースコードに対して，各行の先頭に行番号を追加する．その後単語単位でソースコードを区切りトークン化する．この時スペースや改行は削除する．得られたトークン列の最後に列の終端を意味する<end>タグを挿入する．コード差分も同様にトークン化を行い，出力データとする．

4.4　学習

4.4.1　トークンの長さによるふるい分け

作成した入出力データを，トークン列の長さの制約を設けてふるい分ける．トークン列が長すぎるものは学習に不向きであると考えたことに加え，後述する学習環境でのメモリの制約もあるためである．今回は，入力トークンすなわちバグを含むソースコードのトークンの長さを〜450 まで，出力すなわちコード差分のトークンの長さを〜323 とした．前者は Rahul らが学習に用いた時のパラメータであり，同様のものを設定した．後者は Rahul らが用いたパラメータでは短すぎるため，SO high rank データセットのコード差分のトークンの長さの分布のうち，75%のデータセットが含まれるような値を設定した．この段階で 4 つのデータセット数は以下のようになった: SO high rank) 747 件，SO low rank) 512 件，OSS high rank) 3,761 件 (訓練にはうち 3000 件を用いる)，OSS low rank) 31,177 件 (訓練にはうち 3000 件を用いる)．

4.4.2　sequence-to-sequence モデルの学習

4 つのデータセットのうち，それぞれについて学習モデルを作成する．データセットのうち，10%をテスト用，残りを訓練用のデータとする．またテストデータが 100 件を越える場合，データセットのうち 100 件をテストデータとした．OSS high/low rank についてはデータセット件数が多く，学習結果の比較の際に影響が出ると考えたため，それぞれ 3000 件を訓練データに用いることとする．訓練データを用いて sequence-to-sequence ネットワークモデルを訓練させる．入力データを sequence-to-sequence のエンコーダ部に，出力データをデコーダ部に入力して学習を行う．

得られた学習モデル 4 つそれぞれに対して，4 つのデータセットから得られたテストデータを用いてテストを行う．テストデータのバグ修正を含むコードを入力とし，得られた出力を調査することで，バグ修正モデルとしての性能を考察する．

5.　結果と考察

4 つのデータセットそれぞれを用いて学習したモデルで，テストデータに対しどのような出力を行ったかを目視調査した．調査の観点として出力結果を以下の段階で分類した: レベル 0) 全く関係のない修正候補を出力，レベル 1) 修正対象の API を用いた修正候補を出力，レベル 2) バグを部

表 6　SO high rank 学習モデルによるテスト結果

テストデータ	SO high rank	SO low rank	OSS high rank	OSS low rank
テストデータ総数	71	52	100	100
レベル 1	31	13	2	1
レベル 2	1	0	0	0

表 7　SO low rank 学習モデルによるテスト結果

テストデータ	SO high rank	SO low rank	OSS high rank	OSS low rank
テストデータ総数	71	52	100	100
レベル 1	16	19	1	0
レベル 2	0	1	0	0

表 8　OSS high rank 学習モデルによるテスト結果

テストデータ	SO high rank	SO low rank	OSS high rank	OSS low rank
テストデータ総数	71	52	100	100
レベル 1	11	2	36	0
レベル 2	0	0	0	0

表 9　OSS low rank 学習モデルによるテスト結果

テストデータ	SO high rank	SO low rank	OSS high rank	OSS low rank
テストデータ総数	71	52	100	100
レベル 1	1	0	6	3
レベル 2	0	0	0	0

実際の修正後のコード

```
public class highscores extends Activity {
  private ListView scorebox;
  @Override protected void onCreate(  Bundle
savedInstanceState){
    super.onCreate(savedInstanceState);
    setContentView(R.layout.highscores);
    DBHandler db=new DBHandler(this);
  scorebox=(ListView)findViewById(R.id.scorebox);
    Log.d("Insert: ","Inserting ..");
    db.addScore(9000);
    Log.d("Reading: ","Reading all contacts..");
    ArrayList<String> ar=new ArrayList<>();
    ar=db.getAllScores();
    ArrayAdapter<String> ar=new
ArrayAdapter<String>(this,android.R.layout.simple
_list_item_1,s);
    scorebox.setAdapter(ar);
  }
}
```

出力した修正を適用したコード

```
public class highscores extends Activity {
  private ListView scorebox;
  @Override protected void onCreate(  Bundle
savedInstanceState){
    super.onCreate(savedInstanceState);
    setContentView(R.layout.highscores);
    DBHandler DBHandler_Var = new DBHandler ( this ) ;
    NoType_Var=(ListView) findViewById (R.id.scorebox) ;
    Log.d("Insert: ","Inserting ..");
    db.addScore(9000);
    Log.d("Reading: ","Reading all contacts..");
    String[] NoType_Var = DBHandler_Var.getAllScores( );
    ArrayAdapter <String> NoType_Var = new
ArrayAdapter<String>(this,android.R.layout.simple_list_it
em_1, s ) ;
    NoType_Var.setAdapter ( NoType_Var ) ;
  }
}
```

図 10　部分的な修正に成功したテストの例

分的に (または完全に) 修正する修正候補を出力．4 つのデータセットそれぞれについてのテストの結果を表 6，表 7，表 8，表 9，に示す

5.1　RQ1．集合知から作成したデータセットにより自動バグ修正モデルが作成できるか

SO high rank，SO low rank，OSS high rank による学習モデルは，それぞれ同データセットにおけるテストではレベル 1 以上の出力結果の割合が 45% (=(31+1)/71)，38% (=(19+1)/52)，36% (=36/100) という結果になった．これはすなわち，修正すべき API(の一部) を特定した割合となる．また SO high rank，SO low rank においては 1 件ずつ部分的に望まれる修正を出力した．その例を図 10 に示す．SO low rank においてのテストの結果の一つであり [*5]，図左のソースコードが実際に Stack Overflow の回答投稿で修正されたコードで，右が学習モデルが出力した修正差分を適応したコードである．赤字に示すソースコードが修正された行である．変数の型は正しく出力できていない部分があるものの，ほとんどの修正が一致している．しかしな

[*5] https://stackoverflow.com/questions/42468018/why-is-this-sqlite-database-not-working/より作成されたテスト

がら，4 つの学習モデルすべてのテスト結果において，修正すべき行数がほとんど一致していないという結果も得られた．これは作成した学習モデルがバグの箇所を特定する能力が低いことを示している．そのためソースコードの部分を削除すべきか判断する能力よりも，どのようなソースコードを加えるべきか判断する方が得意なモデルを構築したと言える．

このことから，バグ修正モデルとして実用的なレベルでなくとも，修正すべき API の推薦への実用価値があると考えられる．

> 3 つのデータセットにおいて，修正すべき API を特定する能力のあるモデルが作成できた．また一部のテストにおいては，望まれる修正を部分的に行なっており，自動バグ修正への可能性が示された．バグの箇所の特定能力は低い結果になったが，修正すべき API の推薦への利用については期待できる．

5.2 RQ2. 学習データセットの違いが修正結果にどのような差を生むか

SO データセット，OSS データセット双方ともに，high rank データセットの方がレベル 1 以上の割合が高くなるという結果になった．よって頻出する API に対しては学習モデルの性能が良い言える．またこのことは訓練データが行いたい修正に対して適切に選択されていることを示している．OSS low rank についてはテストにおいてほとんど無関係の修正を出力している．OSS high rank に関しては自身のデータセットに対するテストにおいては，レベル 1 の修正を 36 件行なっていることを考えると，バグ修正の出現頻度を考慮した学習を行う必要があると言える．

> Stack Overflow，OSS で行われたバグ修正に頻出する API に対してはモデルの性能が向上した．

6. 妥当性への脅威

6.1 内的妥当性

本実験では SO と OSS データセットの違いによって性能の比較を行ったが，二つのデータセットの違いが抽出した母集団以外の違いが含まれていることへの懸念が考えられる．今回 OSS データセットは，ソースコードの長さの制約によりメソッド単位で修正差分を分割したが，SO から抽出したソースコードに対してはその操作を行っていない．よって母集団の違い以外の因子がデータセット間に存在している可能性がある．

6.2 外的妥当性

今回の実験において OSS high/low rank 訓練データセットは，得られたデータセットの内その一部をランダムサンプリングして用いている．サンプルするデータを変えることで結果が変わることが予想できる．また今回の実験において，SO high/low rank の訓練データセット件数はどちらも 1000 件にも満たず，深層学習のデータセットとしては数が少ないということが考えられる．また Stack Overflow には似たような投稿は削除される仕組み [*6] があり，特定のバグについての修正ノウハウを複数集めにくい．機械学習の分野ではしばしばミュータントを生成することによりデータセットを増やしている．本手法でもミュータントの生成によりデータセット数を増やす余地がある．

6.3 構成概念的妥当性

集合知から抽出データについては，実際に開発者が受け入れるバグ修正のデータが集められていることは保証できない．Stack Overflow にはユーザや投稿に対して評価を行うことができ，投稿の質として考慮することができる．本実験では質問や回答，または投稿者の質を考慮していない．そのため質の低い投稿を使用している危険性も存在している．Stack Overflow では質問者が寄せられた回答のうち，どの回答を受け入れるか選択することができる (Accepted Answer)．Nasehi らはどのような回答が Accepted Answer に選ばれやすいのかを調査しており，簡潔なコードであること，外部情報ソースへのリンクなどの指標を挙げている [16]．投稿の質を考慮する上で参考にできると考えられる．また OSS においても，プロジェクトの質を考慮していない．OSS のデータを利用した研究においては，リポジトリがフォークされた数やスター数といったメトリクスによって研究対象とする OSS の品質を評価している．また成熟している OSS プロジェクトの評価モデルを作成した研究もあり [5]，OSS プロジェクトの選出の基準の一つにできると考えられる．

本実験で対象とした Android SDK を用いた Java のソースコードは，多くの種類のクラスやメソッドを含んでおり，学習の際の語彙数が多くなる．Rahul らが行った sequence-to-sequence モデルを用いた自動バグ修正では，問題空間は C 言語の初歩的なプログラムに絞っており，出現する語彙も C 言語を構成する演算子に少量の標準関数を加えたものとなっている．Rahul らの実験での語彙数は 102 であるのに対し，本実験での語彙数は 10,000 を超えている．語彙数を減らす方法として，対象とする Android SDK 以外のクラス，メソッドを抽象化することが考えられる．特に OSS のデータセットにおいてはユーザ定義関数が頻繁に出現し，学習の際に不必要な語彙を学習していると言える．本実験の RQ2 の結果から修正したい API に範囲を絞って訓練データを作成することで，性能が向上していることが

[*6] https://stackoverflow.com/help/duplicates

わかる．そのため不必要なユーザ定義関数の抽象化によって学習結果の向上に期待が持てる．

7. まとめ

本論文では Stack Overflow と OSS リポジトリに集められたバグ修正情報を抽出し，Android SDK API メソッドによるバグを修正する自動バグ修正モデルの作成を行った．また訓練に用いたデータセットごとにテストを行い，結果を比較した．結果として，頻繁に用いられる API に関するバグ修正情報によって訓練したモデルは，テストにおいて修正すべき API を一定数特定することに成功した．このことから，修正すべき API を推薦への利用が期待できる．またごく一部ではあるものの，実際のバグを部分的に修正しており，本手法による自動バグ修正の可能性が示された．

謝辞

本研究は JSPS 科研費 JP18H04097 の助成を受けた．

参考文献

[1] Bahdanau, D., Cho, K. and Bengio, Y.: Neural machine translation by jointly learning to align and translate, *arXiv preprint arXiv:1409.0473* (2014).
[2] Bellon, S., Koschke, R., Antoniol, G., Krinke, J. and Merlo, E.: Comparison and evaluation of clone detection tools, *IEEE Transactions on software engineering*, Vol. 33, No. 9 (2007).
[3] Britton, T., Jeng, L., Carver, G., Cheak, P. and Katzenellenbogen, T.: Reversible debugging software, *Judge Bus. School, Univ. Cambridge, Cambridge, UK, Tech. Rep* (2013).
[4] Chen, F. and Kim, S.: Crowd debugging, *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ACM, pp. 320–332 (2015).
[5] Duijnhouwer, F.-W. and Widdows, C.: Open Source Maturity Model, *Capgemini Expert Letter* (2003).
[6] F-Droid: Free and open source android app repository, [Online; accessed 2015-12-19] (2017).
[7] Gazzola, L., Micucci, D. and Mariani, L.: Automatic software repair: A survey, *IEEE Transactions on Software Engineering* (2017).
[8] Gupta, R., Pal, S., Kanade, A. and Shevade, S.: DeepFix: Fixing Common C Language Errors by Deep Learning, *AAAI*, pp. 1345–1351 (2017).
[9] Kim, D., Nam, J., Song, J. and Kim, S.: Automatic patch generation learned from human-written patches, *Proceedings of the 2013 International Conference on Software Engineering*, IEEE Press, pp. 802–811 (2013).
[10] Kim, J., Lee, S., Hwang, S.-w. and Kim, S.: Towards an Intelligent Code Search Engine., *AAAI* (2010).
[11] Le, X.-B. D., Lo, D. and Le Goues, C.: History driven automated program repair (2016).
[12] Le Goues, C., Nguyen, T., Forrest, S. and Weimer, W.: Genprog: A generic method for automatic software repair, *Ieee transactions on software engineering*, Vol. 38, No. 1, p. 54 (2012).
[13] Liu, X. and Zhong, H.: Mining stackoverflow for program repair, *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, IEEE, pp. 118–129 (2018).
[14] Long, F. and Rinard, M.: Staged program repair with condition synthesis, *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ACM, pp. 166–178 (2015).
[15] Long, F. and Rinard, M.: Automatic patch generation by learning correct code, *ACM SIGPLAN Notices*, Vol. 51, No. 1, pp. 298–312 (2016).
[16] Nasehi, S. M., Sillito, J., Maurer, F. and Burns, C.: What makes a good code example?: A study of programming Q&A in StackOverflow, *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, IEEE, pp. 25–34 (2012).
[17] Qi, Y., Mao, X., Lei, Y., Dai, Z. and Wang, C.: The strength of random search on automated program repair, *Proceedings of the 36th International Conference on Software Engineering*, ACM, pp. 254–265 (2014).
[18] Śliwerski, J., Zimmermann, T. and Zeller, A.: When do changes induce fixes?, *ACM sigsoft software engineering notes*, Vol. 30, No. 4, pp. 1–5 (2005).
[19] Sutskever, I., Vinyals, O. and Le, Q. V.: Sequence to sequence learning with neural networks, *Advances in neural information processing systems*, pp. 3104–3112 (2014).
[20] 肥後芳樹，楠本真二ほか：プログラム依存グラフを用いたコードクローン検出法の改善と評価，情報処理学会論文誌，Vol. 51, No. 12, pp. 2149–2168 (2010).