

木編集距離に着目した類似解答ソースコード検索器における 深層学習モデルの性能評価

沖野 健太郎^{1,a)} 松尾 春紀^{1,b)} 山本 大貴^{1,c)} 近藤 将成^{1,d)} 亀井 靖高^{1,e)} 鵜林 尚靖^{1,f)}

受付日 2021年8月2日, 採録日 2022年1月11日

概要: 近年の IT 社会の発展によって IT 人材の不足が深刻になり, プログラム自動生成を含むソフトウェア開発の自動化が求められている. 多くの研究が行われているなかで, プログラム自動生成をより実用的なものとするために, 自動生成の過程でソースコード検索器を使用している研究がある. その研究では, 求めるソースコードに木構造が近いと推測される類似解答ソースコードを検索し, 自動生成の雛形としている. この手法を用いることで, プログラミングコンテスト AtCoder の解答ソースコードの自動生成において, 検索を行わない場合と比較して自動生成できた件数が増加したと報告されている. 本研究では, 木編集距離を学習に用いたソースコード検索器に着目した. ソースコード検索器の性能に影響を与える要因を実証的に調査することで, プログラム自動生成の精度向上への知見を得ることを目指す. 調査では, 検索精度に影響を与える要因として, 深層学習モデルの構造, ソースコードの入力形式, 問題の複雑度の3つを対象とし, AtCoder の問題を使用して検索精度の比較を行った. 調査の結果, 類似解答ソースコード検索において Transformer のエンコーダ部分の使用は有効であることが期待できること, AtCoder の問題に対して抽象構文木のベクトル表現の使用は有効であるとはいえないこと, 問題の複雑度は検索精度に影響を与えることを示した.

キーワード: コード検索, 深層学習, 木編集距離, 自動プログラミング

Performance Evaluation of Deep Learning Models for Similar Source Code Search By Focusing on Tree Edit Distance

KENTARO OKINO^{1,a)} HARUKI MATSUO^{1,b)} HIROKI YAMAMOTO^{1,c)} MASANARI KONDO^{1,d)}
YASUTAKA KAMEI^{1,e)} NAOYASU UBAYASHI^{1,f)}

Received: August 2, 2021, Accepted: January 11, 2022

Abstract: Automatic program generation is an active research topic in software engineering. To make automatic program generation more practical, a prior study applies source code search to the method of automatic program generation. In that study, source codes whose tree structures may be close to the desired source code developers require are searched and used as a template for the method. They reported that the method with source code search increases the number of generated source codes compared to the method without source code search. In this study, we use source code search using the tree edit distance. By empirically investigating the factors that affect the performance of source code search, we aim to improve the accuracy of automatic program generation. We focused on three factors that affect the search accuracy: the structure of the deep learning model, the input format of the source code, and the complexity of the problem. We compared the search accuracy on the AtCoder problems. We found that the encoder part of Transformer is promising for source code search, the use of vectorized representation of abstract syntax trees is less effective for the AtCoder problems, and the complexity of the problem affects the search accuracy.

Keywords: code search, deep learning, tree edit distance, automatic programming

1. はじめに

近年の IT 社会の発展により、今後 IT 人材の不足が深刻化すると予想されている。経済産業省によると、2015 年時点では日本国内における IT 人材が約 17 万人不足していたとされ、2030 年には不足が約 79 万人に拡大すると予想されている [1]。この問題を解決するため、少人数で効率的な開発を行うことを目的として、ソフトウェア開発の自動化が注目されている。ソフトウェア開発の自動化を行うことは、今後の IT 社会の発展には重要であると考えられる。

ソフトウェア開発の自動化の 1 つに、プログラム自動生成がある。プログラム自動生成に関連した実用化されている研究の例としては、表計算ソフトである Excel で自動的にデータを入力する FlashFill [2] や、統合開発環境である Visual Studio でコード補完を行う IntelliCode^{*1}などがあげられるが、自動化の対象が限定的であったり実装支援にとどまっていたりとプログラム自動生成の実用化の難しさがうかがえる。

倉林ら [3] の研究では、ソフトウェア開発におけるプログラム自動生成をより実用的なものとするため、ソースコード検索器を利用したプログラム自動生成手法を提案している。この研究の特徴として、木編集距離を学習に用いたソースコード検索器を利用し、自動生成したいプログラムの雛形を準備する点があげられる。具体的には、自動生成したい AtCoder の問題の問題文をソースコード検索器の検索クエリ（以降クエリ）とし、クエリに類似した問題の解答ソースコードを検索している。この手法により、ソースコード検索器を用いない場合と比較して自動生成できる件数が増加したと報告されている。この手法を用いたソースコード検索では、プログラム自動生成の部品となる類似解答ソースコードの検索性能が自動生成の精度に影響すると考えられる。

本研究では、倉林らの手法における木編集距離を学習に用いた類似解答ソースコード検索器に着目した。ソースコード検索器の性能に影響を与える要因を実証的に調査することで、プログラム自動生成の精度向上への知見を得ることを目指す。調査では、検索精度に影響を与える要因として、深層学習モデルの構造、ソースコードの入力形式、ソースコードの複雑度の 3 つの要因についてプログラミングコンテストサイト AtCoder の問題を使用して検索精度の比較を行い、それぞれの要因が検索精度に与える影響を

調査した。

本稿では、2 章で研究の背景と目的について述べる。3 章では調査における実験の設計について述べる。4 章では提案した調査課題について実験を行い、その結果について考察する。5 章では妥当性に対する脅威について述べ、6 章でまとめを行う。

2. 背景と目的

2.1 ソースコード検索器を用いたプログラム自動生成

倉林ら [3] の研究では、ソフトウェア開発におけるプログラム自動生成をより実用的なものとするため、ソースコード検索器を利用したプログラム自動生成手法を提案している。プログラム自動生成自体は遺伝的アルゴリズムを用いて行われ、問題の入出力例を満たすようなプログラムを合成している。この遺伝的アルゴリズムのベースとなる初期生成個体の選定において、ソースコード検索器が利用されている。

ソースコード検索器は自動生成を行いたい問題の問題文をクエリとし、過去に開催されたコンテストの問題で構成されているデータセットから最も類似していると予測される問題の解答ソースコードを検索する。類似問題の解答ソースコードを遺伝的アルゴリズムの初期生成個体とすることで正解となるソースコードまでの合成に必要な手順が少なくなるため、プログラム自動生成を行ううえで有利になると考えられる。論文 [3] では、ソースコード検索器を用いることでプログラム自動生成に成功した件数が研究対象とした AtCoder の 92 件の問題のうち 35 件から 40 件に増加したと報告されている。

2.2 情報検索を用いたソースコード検索

Sourcerer [4] や CodeHow [5] をはじめとする従来のソースコード検索手法は情報検索（Information Retrieval）に基づいているものが多い。情報検索はデータベースに蓄積されている検索対象となるデータに対するメタデータを検索アルゴリズムによって選択する検索手法である。また、ソースコードとドキュメント間に関連づけるトレーサビリティを回復するための手法としても、情報検索は古くから用いられている [6]。

Gu ら [7] によると、情報検索に基づいたソースコード検索がかかえる問題として、自然言語で記述されたクエリに含まれる高レベルの意図とソースコードに含まれる低レベルの意図との不一致が存在すると述べられている。その例の 1 つに、情報検索に基づいた検索では類義語を検索することが困難であるという点があげられている。たとえば、情報検索に基づいたソースコード検索では、“read”と同様に読み込むという意味を持った“load”や、“object”と同

¹ 九州大学
Kyushu University, Fukuoka 819-0395, Japan
a) okino@posl.ait.kyushu-u.ac.jp
b) matsuo@posl.ait.kyushu-u.ac.jp
c) h.yamamoto@posl.ait.kyushu-u.ac.jp
d) kondo@ait.kyushu-u.ac.jp
e) kamei@ait.kyushu-u.ac.jp
f) ubayashi@ait.kyushu-u.ac.jp

^{*1} <https://visualstudio.microsoft.com/ja/services/intellicode/>

プログラム 1 “reverseArray” と予測されるメソッドの例

```

1 String[] f(final String[] array) {
2     final String[] newArray = new String[array
      .length];
3     for (int index = 0; index < array.length;
      index++) {
4         newArray[array.length - index - 1] =
      array[index];
5     }
6     return newArray;
7 }

```

様に実体を指す “instance” のような類義語で構成されたソースコードは検索できない可能性がある。また、クエリ内の無関係なキーワードやノイズを効果的に処理できないという問題点が存在するとも述べられている。自然言語とソースコードの間には意図の乖離があり、情報検索を用いたソースコード検索には限界があると考えられる。

2.3 深層学習を用いたソースコード検索

深層学習を用いたソースコード検索では、自然言語とソースコードの分散表現の獲得を行うことにより、従来の情報検索に基づいたソースコード検索で問題となっていたそれぞれの意図の乖離の解消を期待できる。分散表現は単語や文章を高次元の実数ベクトルで表現する方法のことであり、近い意味を持つ表現は近い実数ベクトルで表現されるという特徴がある。たとえば、ファイルの読み込みなどで用いられる “read” や “load” という単語は近い意味として処理され、クエリと検索対象のソースコードが乖離している場合でも検索結果に出現することを期待できる。また、自然言語であるクエリの文字列とソースコードをそれぞれ実数ベクトルという共通した要素に変換することで、数学的にそれぞれの類似度を比較することが可能となる。この手法は Joint Embedding と呼ばれ、異なる種類のデータどうしの関連度を学習する際に用いられる [8]。

DeepCS [7] は深層学習を用いたソースコード検索の代表的な研究の 1 つである。GitHub 上の Java プロジェクトに含まれる関数とドキュメンテーション文字列を入力として学習を行い、プログラム技術関連のコミュニティである Stack Overflow^{*2} に質問として投稿された “convert an input stream to a string” のような具体的なクエリに対して深層学習を用いたソースコード検索器の検証を行っている。従来研究である CodeHow [5] などの情報検索に基づくソースコード検索器による検索結果と比較して評価値が高いことを示し、深層学習を用いたソースコード検索の有用性を評価している。また、クエリに対するベストマッチなソースコードに対する検索精度だけでなく、関連したソースコードの検索精度についての有用性も評価している。

^{*2} <https://stackoverflow.com>

NCS [9] は教師なし学習を用いたソースコード検索の手法である。この手法では、自然言語テキストでの単語の分布がソースコードテキストにもあてはまるという仮説に基づき、ソースコード内から関数名やコメントなどの特徴的な文字列を抽出してソースコードの埋め込み表現を獲得し、クエリ文字列を埋め込んだものとの距離を用いて検索を行っている。また、論文 [9] では、NCS を用いた Stack Overflow からの質問に回答した事例研究が紹介されている。Stack Overflow データセットの質問に対して GitHub からダウンロードされたコードコーパスを用いた検索を行い、3 分の 2 の質問に答えることができたことが報告されている。

2.4 抽象構文木を用いた分散表現の獲得

code2vec [10] では効果的なソースコードの分散表現を獲得するための手法の提案を行っている。この研究では、ソースコードの抽象構文木 (Abstract Syntax Tree, AST) の情報を深層学習モデルへ入力することで、ソースコードの構造的な情報を効果的に獲得できると述べられている。

具体的な例として、配列を逆順に並べ替える Java メソッド (プログラム 1) からメソッド名を予測するタスクが紹介されている。このソースコードには “reverse” という単語や類義語は含まれていないにもかかわらず、“reverseArray” というメソッド名が予測される。AST を用いることにより、プログラム内で逆順の配列を作成する構造の特徴を学習できていると考えられる。

論文 [10] では、メソッド名予測タスクを含むいくつかの検証結果を AST を利用していない従来の深層学習モデルで行った場合の結果と比較し、AST を用いたソースコードの分散表現の獲得の有用性を評価している。また、AST を用いた分散表現の獲得手法は、ソースコード検索器への応用も期待できると述べられている。

code2seq [11] は code2vec を発展させたもので、code2vec と同様に効果的なソースコードの分散表現を獲得する手法を提案している。code2vec が既存の各メソッド名に対する確率を出力していたのに対し、code2seq はメソッド名自体を出力するようになっている。そのため、code2vec は既存のメソッド名しか予測できないという問題点があったのに対し、code2seq では単語を組み合わせて新たなメソッド名を作り出すことができるようになっている。論文 [11] では、いくつかのタスクに対して code2vec を含む従来手法との検証結果の比較を行い、code2seq を用いたソースコードの分散表現の獲得の有用性を評価している。

PathPair2Vec [12] は code2vec に触発されて提案された AST パスを用いたソースコードの表現手法である。code2vec や code2seq が AST の末端から末端までの長いパスを取り出すのに対し、PathPair2Vec は末端から内部ノードまでの短いパスを取り出して、対応したパスのベ

アを組み合わせているという手法をとっている。また、パスのトップノードを削除することで、冗長な情報が破棄されて残りのソースコードの表現力が向上すると述べられている。論文 [12] では、ソースコードの欠陥予測タスクに対して code2vec や code2seq との検出精度の比較を行い、PathPair2Vec を用いたソースコードの分散表現の獲得の有効性を評価している。

2.5 本研究の目的

本研究では、問題文に対応する解答ソースコードに構造的に類似している別のソースコードのことを問題文に対応する類似解答ソースコードとし、問題文をクエリとしてその類似解答ソースコードを検索することを類似解答ソース検索と定義する。

類似解答ソースコード検索器を用いたプログラム自動生成手法では、類似解答ソースコード検索器自体の性能がプログラム自動生成の精度に影響すると考えられる。深層学習を用いて自然言語からソースコードを検索する研究は、DeepCS や NCS をはじめとして複数の提案が行われている [13]。しかし、本研究ではプログラム自動生成に向けた類似解答ソースコード検索器を調査しており、そのために、ソースコードどうしの類似度を木編集距離という指標を用いて学習する点が、既存の類似解答ソースコード検索手法のみを提案した研究と異なる。木編集距離の活用により、ソースコード検索からプログラム自動生成まで一貫した基準を用いることができる [3]。

本研究では、倉林らの研究 [3] で用いられていた木編集距離を用いたソースコード検索器をもとに、その性能に影響を与える要素の実証的な調査を行う。そのために、倉林らの手法に対して、AST の分散表現の基礎的な手法である code2seq を参考にした AST による入力形式の変換、および、Transformer のエンコーダ部分のモデル構造の適用を行い、その影響調査を行った。さらに検索を行う問題の難易度の影響についても調査を行った。プログラム自動生成のための、木編集距離を学習に用いた類似解答ソースコード検索器の性能に影響を与える要因を調査することで、プログラム自動生成の精度向上への知見を得ることを目指す。

本研究では、3つの調査課題について調査を行う。調査内容は以下のとおりである。

調査課題 1：Transformer のエンコーダ部分の利用は類似解答ソースコード検索において有効か。

多くの分野において Transformer [14] の有効性の研究が進められている。一方で、プログラム自動生成のための、木編集距離に着目した類似解答ソースコード検索器の研究においては、その有用性はまだ調査されておらず、LSTM を活用した研究のみである [3]。本調査では、Transformer のエンコーダ部分を木編集距離に着目した類似解答ソースコード検索器のエンコード手法として活用することで、

その精度を向上させることができると仮説を立てた。本調査課題では、従来の LSTM (Long Short Term Memory) を用いて作成した検索器と Transformer のエンコーダ部分を参考に作成した検索器の検索精度を比較することで、Transformer のエンコーダ部分の利用が有効であるかを調査する。

調査課題 2：AST のベクトル表現は類似解答ソースコード検索において有効か。

ソースコード要約などのタスクにおける AST の有効性の研究が行われている [10] が、類似解答ソースコード検索に対してもソースコードの AST のベクトル表現を用いることが有効であるかを確かめる。本調査課題では、ソースコードをテキストベースで入力しベクトル化を行う検索器と AST ベースで入力しベクトル化を行う検索器の検索精度を比較することで、AST のベクトル表現を用いることの有効性を調査する。

調査課題 3：問題の複雑度は類似解答ソースコード検索に影響があるか。

先行研究 [3] では検索対象の問題の複雑度については触れられておらず、複雑度が類似解答ソースコード検索に与える影響は分かっていない。本調査課題では、検索対象の問題の複雑度に応じて2つのデータセットを作成し、それぞれにおける類似解答ソースコード検索の精度を確かめることで、問題の複雑度が検索精度に与える影響について調査する。

3. 実験の設計

3.1 データセット

本研究では、競技プログラミングコンテストサイトである AtCoder^{*3}のデータを用いて調査を行う。過去に開催された初級者向けコンテストである AtCoder Beginner Contest (ABC) の第 42 回以降の A 問題および B 問題各 157 問を調査対象とした。第 42 回以降のみを対象とした理由は、類似解答ソースコード検索器の入力に用いる英語で記述された問題文が、第 41 回以前は公式で用意されていないためである。AtCoder では過去の問題および各ユーザの提出したソースコードが Web サイト上ですべてのユーザに対して公開されており、本研究では各問題に対して問題文と解答ソースコードを取得してデータセットを作成した。各問題に対する解答ソースコードは、Python で記述されており AC (正答) となっているソースコードの中で実行時間が最も短いものを使用した。この選択の理由は、ABC の問題においては実行時間制限が定められており、実行時間の短い効率的なアルゴリズムを用いて解答されたソースコードほど優れたソースコードであると考えたためである。以降 ABC の A 問題で構成されたデータセットの

^{*3} <https://atcoder.jp>

表 1 ABC データに対する複雑度を表す各指標の中央値
Table 1 The median values of the complexity metrics for the ABC data.

データセット	Words	LOC	CC
ABC-A	48	5	2
ABC-B	63	8	3

プログラム 2 ABC130 の A 問題の解答ソースコード

```
1 x, a = map(int, input().split())
2 if x < a:
3     print('0')
4 else:
5     print('10')
```

プログラム 3 ABC152 の A 問題の解答ソースコード

```
1 N, M = map(int, input().split())
2 if N == M:
3     print('Yes')
4 else:
5     print('No')
```

ことを ABC-A データ, B 問題で構成されたデータセットのことを ABC-B データ, データセット全体を ABC データと呼ぶ。また, それぞれの問題文と解答ソースコードの組を単に問題と呼ぶ。

ABC データに含まれる問題の複雑度の目安として, 問題文の語数 (Words), ソースコードの論理行数 (LOC) および循環的複雑度 (CC) の中央値を表 1 に示す。問題文, ソースコードの各指標ともに ABC-A データに対して ABC-B データの値が大きいことから, ABC-A データよりも ABC-B データのほうが全体として複雑度が高いといえる。

ABC データの特徴として, 問題文は類似していない場合でも解答ソースコードの構造が類似している組合せが存在することがあげられる。例として, ABC130 の A 問題 (プログラム 2) と ABC152 の A 問題 (プログラム 3) を示す。ABC130 は入力された整数の大小を判別する問題, ABC152 はテストケースの数 N と通ったテストケースの数 M から AC (正答) になるかを判定する問題となっており, 問題文の観点からは両者は類似していないと考えられる。一方で解答ソースコードは両者とも入力した 2 つの数値の比較結果により出力を行うというソースコードであり, 構造的な観点からは類似しているといえる。

3.2 問題の類似度の定義

本調査では, ソースコード間の木編集距離 (Tree Edit Distance, TED) [15] に着目して問題の類似度の評価を行う。TED は 2 つの木構造で表されるデータどうしの編集距離を表す指標であり, この値が小さいほど木構造が類似

していることを示す。ソースコードを AST に変換することで, TED を用いた問題の類似度を客観的に評価することが可能である。

問題の類似度の評価に TED を用いる理由としては, プログラム自動生成の過程で遺伝的アルゴリズムを用いる際に, ソースコードの木構造の近さが実行時間に影響を与えると考えられるためである [3]。また, 類似解答ソースコード検索からプログラム自動生成まで, 一貫した評価基準を使うことができる。

本調査では, 2 つの問題 m, n 間の類似度 S を TED を用いた以下の式によって定める。

$$S(m, n) = \text{Max}(1 - \text{TED}(p_m, p_n) / T_{const}, 0) \quad (1)$$

式中の p_m, p_n は問題データ m, n に対する解答ソースコードを示す。 T_{const} は閾値であり, TED がこの値を超えた場合は類似度が 0 になる。この式により, ソースコードの木構造が類似している組合せに対しては 1 に, 類似していない組合せは 0 に近くなるように類似度を定めることができる。この類似度を教師ラベルとして学習を行うことで, 類似解答ソースコード検索器がクエリと検索対象の問題の類似度を予測できるようになる。

3.3 AST の入力手法

本調査では, ソースコードの AST を深層学習モデルに入力する際に AST パス化を行う。AST パス化とは code2seq [11] で用いられている手法であり, ソースコードを AST に変換し, ある末端ノードから別の末端ノードまでのパスの集合を取り出す手法である (1 つのソースコードから複数の AST パスが生成される)。AST パスを用いることで, テキストベースの学習と比較して深層学習モデルがソースコードの構造的な意味を学習することが期待できる。

図 1 に AST パス化の例を示す。まず, “print(1+2)” というプログラムを AST に変換する。この AST に対して末端ノードを 2 つ選んでそのパスを抽出する処理をすべての末端ノードの組合せに対して行う。この際, 末端ノードの部分は “print” と “NameLoad” のように名称部分と動作部分に分けて抽出される (名称部分はサブトークン分割が行われる)。この例の AST の末端ノードは 3 個であるため, この AST から AST パスを ${}_3C_2 = 3$ 個抽出できる。この AST パスを末端の名称部分と経路部分に分割し, 深層学習モデルへの入力とする。

また, AST パスは AST のノード数に対して指数関数的に増加するため, 本調査では生成された AST パスをランダムに 64 個取得して利用した。AST パスの上限を 64 個にした理由としては, 32 個の場合に ABC-B データに対する検索精度の低下が見られることと, 128 個の場合に検索精度の向上が見られなかったことがあげられる。

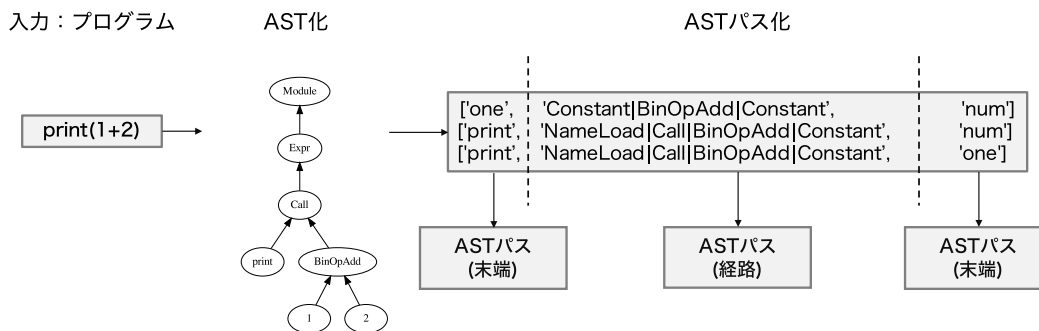


図 1 AST パス化の例
Fig. 1 An example of AST paths.

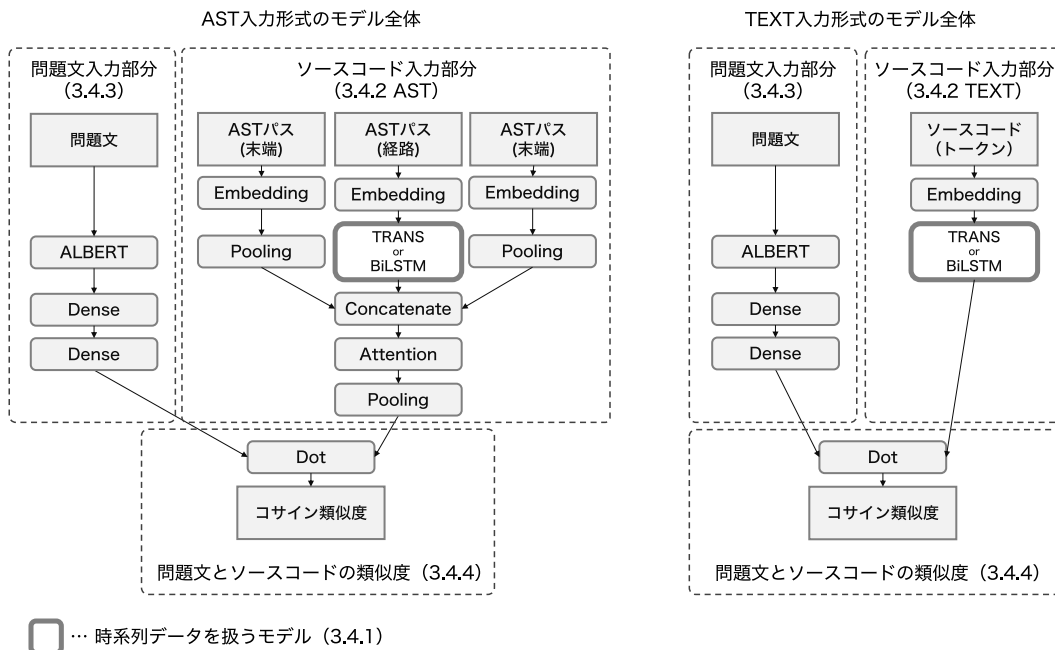


図 2 各入力形式のモデル構造の概略図
Fig. 2 Overviews of the model structure for each input type.

3.4 調査する深層学習モデル

本節では、調査で用いる深層学習モデル全体の概要について述べた後、それぞれの組合せ部分について述べる。本調査ではソースコードの入力形式とモデル構造の組合せを変化させて調査を行い、各組合せごとに類似解答ソースコード検索の精度の評価を行う。時系列データを扱うモデルについて2通り、ソースコード入力部分の入力形式について2通り、問題文入力部分の深層学習モデルについて1通りを組み合わせで作成した類似解答ソースコード検索器について調査を行う。本調査で用いた最適化アルゴリズムや損失関数については付録 A.1 に、用いた計算資源については付録 A.2 に示した。

図 2 に各入力形式ごとの深層学習モデルの概略図を示す。図中の各部分の番号は、本文中の見出し番号に対応している。問題文入力部分では、学習済みモデルである ALBERT [16] を用いて問題文の分散表現 (実数ベクトル) を出力する。ソースコード入力部分 (AST 入力形式) で

は、前半部分でソースコードの各 AST パスの分散表現を得た後、後半部分で AST パス全体 (ソースコード全体) の分散表現を出力する。ソースコード入力部分 (TEXT 入力形式) では、トークン全体 (ソースコード全体) の分散表現を出力する。その後、問題文入力部分で得られた問題文の分散表現と、ソースコード入力部分で得られたソースコードの分散表現のコサイン類似度を計算し、深層学習モデル全体の出力としている。この構造により、深層学習モデルが問題文とソースコードの類似度を学習できるようになっている。

3.4.1 時系列データを扱うモデル

TRANS. TRANS は、Transformer のエンコーダ部分を採用した深層学習モデルであり、図 2 のソースコード入力部分のそれぞれの入力形式において Transformer のエンコーダ部分の構造を持つ (AST-TRANS のように入力形式と組み合わせることで1つの深層学習モデルとなる)。TRANS は、Positional Encoding 層、Multi-Head Attention 層、Dense

層（全結合層）、Layer Normalization 層および Pooling 層から構成されており、時系列データ（AST パスやトークン列）を入力とし実数ベクトルを出力する。Transformer は自然言語処理などの分野において従来の RNN（Recurrent Neural Network）ベースのモデルと比較して学習が高速で高精度であることが報告されている [14]。本研究では、類似解答ソースコード検索において Transformer のエンコーダ部分の利用が有効であるかを確かめるために用いる。なお、データセットの規模に合わせて、TRANS は一般的な Transformer より小規模な構成となっている。具体的なパラメータの値については付録 A.1 に示した。

BiLSTM. BiLSTM は、RNN の一種である LSTM を採用した深層学習モデルであり、図 2 のソースコード入力部分のそれぞれの入力形式において双方向 LSTM 層 (BiLSTM) を選択した構造を持つ。BiLSTM は、TRANS と同様に時系列データを入力とし実数ベクトルを出力する。本研究では、時系列データの学習に使用されている従来の方式の 1 つとして、TRANS を用いた場合との検索精度の比較に用いる。

3.4.2 ソースコード入力部分の入力形式

AST. AST 入力形式は、深層学習モデルにソースコードを入力するための前処理として AST パス化を行った入力形式である。図 2 のソースコード入力部分 (AST 入力形式) は、各 AST パスの分散表現を出力する前半部分と、AST パス全体の分散表現を出力する後半部分に分けられる。前半部分では、各 AST パスの経路部分と末端部分それぞれに対して Embedding 層、BiLSTM 層もしくは TRANS 層、Pooling 層を用いて分散表現を獲得した後、Concatenate 層を用いて結合を行い各 AST パスの分散表現を出力する。後半部分では、前半部分で出力された各 AST パスの分散表現に対して Attention 層を用いて重み付けを行った後、Pooling 層を用いて圧縮を行い AST パス全体の分散表現を出力する。本研究では、類似解答ソースコード検索において AST のベクトル表現の利用が有効であるかを確かめるために用いる。

TEXT. TEXT 入力形式では、深層学習モデルにソースコードを入力するための前処理としてトークンごとに分解を行った入力形式である。たとえば、“print(1+2)” というプログラムは print,(1,+2,) のように分割され、深層学習モデルに入力される。図 2 のソースコード入力部分 (TEXT 入力形式) は、トークン全体に対して Embedding 層、BiLSTM 層もしくは TRANS 層を用いてトークン全体 (ソースコード全体) の分散表現を出力する。TEXT 入力形式は、ソースコードを深層学習モデルに入力するための基本的な方法の 1 つとして、AST を入力に用いた場合との検索精度の比較に用いる。

3.4.3 問題文入力部分

ALBERT. 本研究では、深層学習モデルの問題文入力部

分に ALBERT [16] の学習済みモデルを使用している*4。ALBERT は BERT [17] から派生した自然言語処理に用いられる Transformer 構造を持つ言語モデルで、BERT より高速な学習と高性能を実現したとされている。

3.4.4 問題文とソースコードの類似度

各深層学習モデルの上層部分では、問題文の分散表現とソースコードの分散表現を出力することができる。最終層では Dot 層（内積）の出力を正規化することでこの 2 つの分散表現のコサイン類似度を計算し、深層学習モデルの出力としている。コサイン類似度とは 2 つのベクトル \mathbf{A} , \mathbf{B} がどの程度同じ方向を向いているかを表す指標で、以下の式 (2) により算出される。

$$\text{cosine_similarity}(\mathbf{A}, \mathbf{B}) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} \quad (2)$$

このコサイン類似度をシグモイド関数を用いて [0, 1] の範囲に正規化した値が、教師ラベルとなる式 (1) の値に近づくように（損失関数の値が小さくなるように）学習を行うことで、類似解答ソースコード検索器が問題の類似度を予測できるようになる。

3.5 実験の全体像

実験の全体像を図 3 に示す。深層学習を用いた各ソースコード検索器に対する 1 回の実験は以下の流れで実施される。なお、図中の問題文 A-コード B のように英字部分が異なる組合せは、元々の問題とは異なる組合せであることを示している。

- (1) 分割. ABC-A データ、もしくは ABC-B データ 157 件からシード値を用いてランダムに 10 件の問題を取り出し、その問題の問題文をクエリ用データとする。残った 147 件のデータのソースコードを検索対象データとする。
- (2) 生成. 検索対象の 147 件のデータに対し、問題文とソースコードが異なる組合せのデータを生成する。元々の組合せのデータと生成された異なる組合せのデータを合わせて 21609 (= 147²) 件のデータを各深層学習モデルの学習データとする。このとき、各問題データの組合せに対して式 (1) を用いて教師ラベルとなる類似度を計算する。
- (3) 学習. 学習データを用いて各深層学習モデルの学習を行う。深層学習モデルには問題文とソースコードを入力し、式 (2) で計算した類似度が出力される。この類似度が式 (1) の値に近くなるように学習を行う。ただし、前処理の方法や入力形式は各ソースコード検索器によって異なる。
- (4) 結合. それぞれのクエリ用データの問題文に対し、検索対象データのソースコード 147 件との組合せデータ

*4 https://tfhub.dev/tensorflow/albert_en_base/2

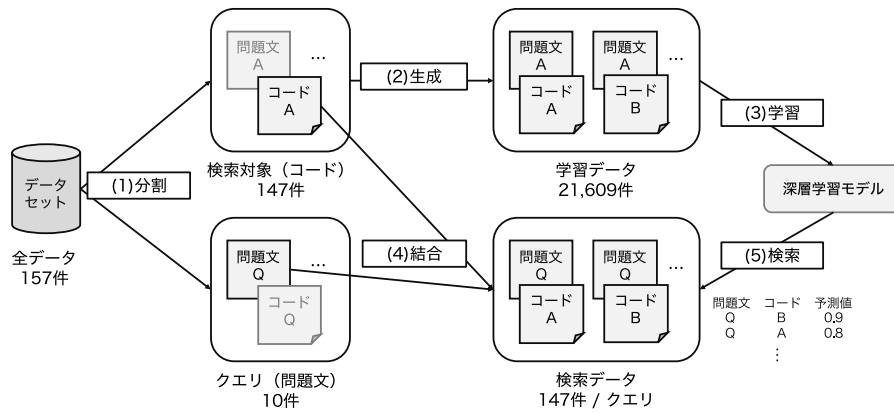


図 3 実験の全体像

Fig. 3 Overview of the experiment.

表 2 評価指標の計算に用いる各 TED の定義

Table 2 Definitions of each TED that are utilized to calculate the evaluation measures.

値	定義
TED_{node}	解答ソースコードと何も書かれていないソースコードとの TED. 解答ソースコードの AST のノード数と等しい.
TED_{pred}	検索器によって予測されたソースコードと解答ソースコードの TED.
TED_{best}	最適な (TED が最小の) ソースコードと解答ソースコードの TED.
TED_{rand}	全検索対象ソースコードから無作為に選んだソースコードと解答ソースコードとの TED の期待値. 全検索対象ソースコードと解答ソースコードの TED の平均値と等しい.

を作成する。これを検索データとする。

- (5) **検索.** 検索データの問題文とソースコードを深層学習モデルに入力し、問題文とソースコードの類似度を推測する。推測された類似度を降順に並べ替えたものを類似解答ソースコード検索器の検索結果とする。

3.6 評価指標

各ソースコード検索器に対する検索精度を客観的に評価するため、以下の評価指標を用いて検索精度の評価を行う。一般的な検索器の評価指標としては検索順位に注目した MRR (Mean Reciprocal Rank) や SuccessRate などが用いられることが多いが、本調査ではソースコードの TED に注目した評価指標を定義して用いる。その理由は、遺伝的アルゴリズムを用いたプログラム自動生成を行う場合、初期個体のソースコードと解答ソースコードの木構造が近いほど短い実行時間で合成できることが期待でき、そのためには TED ができるべく小さい類似解答ソースコードを検索する性能が重要であるためである。

各評価指標の値の算出には表 2 に示した TED に関連した値を用いる。クエリ 1 件に対して TED_{node} , TED_{best} , TED_{rand} は類似解答ソースコード検索器によらず一意に決まり、 TED_{pred} は類似解答ソースコード検索器ごとに決まる。

SearchValidity. SearchValidity (SV) は、プログラム自動生成の観点で検索した類似解答ソースコードを用いる

ことが妥当であるかを測定する指標である。この指標の値が大きいくほど解答ソースコードとの類似度が高く、検索した類似解答ソースコードがプログラム自動生成において有効であるといえる。SearchValidity は以下の式で計算される。 $TED_{pred} = 0$ の場合に最大値 1 をとり、 $TED_{node} \leq TED_{pred}$ の場合に最小値 0 をとる。

$$SearchValidity = \text{Max} \left(1 - \frac{TED_{pred}}{TED_{node}}, 0 \right) \quad (3)$$

SearchAccuracy. SearchAccuracy (SA) は検索結果がどれほど正確であるかを測定する指標である。この指標の値が大きいくほど理想的な検索結果に近く、0 に近づくほどランダムに検索を行った場合の結果に近いといえる。SearchAccuracy は以下の式で計算される。 $TED_{pred} = TED_{best}$ の場合に最大値 1 をとり、 $TED_{pred} = TED_{rand}$ の場合に 0 になる。

$$SearchAccuracy = \frac{TED_{rand} - TED_{pred}}{TED_{rand} - TED_{best}} \quad (4)$$

ValidRatio. ValidRatio (VR) は理想的な検索が妥当であるクエリのうち、実際の検索が妥当であるクエリの割合を測定する指標である。ここでの理想的な検索が妥当であるクエリとは、 $TED_{pred} = TED_{best}$ のときに SearchValidity が 0 より大きいクエリのことを指す。この指標の値が大きいくほど妥当な検索が可能であったクエリの割合が高いといえる。ValidRatio は以下の式で計算される。 δ は条件式が真の場合は 1、偽の場合は 0 を返す関数である。

$$ValidRatio = \frac{\sum \delta(TED_{pred} < TED_{node})}{\sum \delta(TED_{best} < TED_{node})} \quad (5)$$

4. 結果と考察

本章では、実験のアプローチと3つの調査課題についてそれぞれの調査内容と調査結果について述べ、考察を行う。

各ソースコード検索器のABC-A データに対する評価指標の値を表 3, ABC-B データに対する評価指標の値を表 4 に示す。各データセット, 入力形式, モデル構造の組合せにおいてクエリ 10 件に対する検索に対する評価指標の中央値をそのクエリ群に対する評価指標の値とし, クエリデータを変えて 5 回調査を行った評価指標の中央値をその組合せの評価指標の値として表に示している。

表 3, 表 4 より, どの組合せにおいても SearchAccuracy の値は 0 より大きくなっている。SearchAccuracy は 0 であればランダムに検索をしたときと同じ結果であることから, いずれの組合せでも学習に成功していることが分かる。各調査課題では, 表に示したそれぞれの比較対象ごとの評価指標の比を算出して結果を考察する。

4.1 調査課題 1: Transformer のエンコーダ部分の利用は類似解答ソースコード検索において有効か。

調査内容. 従来の LSTM を用いた検索器と Transformer のエンコーダ部分を用いた検索器の検索精度を比較し, Transformer のエンコーダ部分の利用が類似解答ソースコード検索において有効であるかを確かめる。

結果と考察. 調査課題 1 の結果は表 5 のとおりである。表の値は BiLSTM の評価指標の値に対する TRANS の評価指標の値の比を表しており, BiLSTM の評価指標よりも TRANS の評価指標の方が高いときに 1 より大きくなる。SearchValidity (SV) に注目すると, ABC-A データに対しては AST 形式が 1.241, TEXT 形式が 1.767, ABC-B データ

表 3 ABC-A データに対する各検索器の評価指標の値

Table 3 The values of the evaluation measures for each method for the ABC-A data.

入力形式	モデル	SV	SA	VR
AST	TRANS	0.196	0.479	0.760
AST	BiLSTM	0.158	0.440	0.680
TEXT	TRANS	0.288	0.449	0.760
TEXT	BiLSTM	0.163	0.360	0.680

表 4 ABC-B データに対する各検索器の評価指標の値

Table 4 The values of the evaluation measures for each method for the ABC-B data.

入力形式	モデル	SV	SA	VR
AST	TRANS	0.128	0.707	0.776
AST	BiLSTM	0.090	0.636	0.694
TEXT	TRANS	0.128	0.764	0.755
TEXT	BiLSTM	0.055	0.598	0.633

タに対しては AST 形式が 1.422, TEXT 形式が 2.327 と評価指標の値が高くなっている。同様に, SearchAccuracy (SA), ValidRatio (VR) においてもすべての組合せにおいて TRANS の評価指標の値が BiLSTM の値を上回っている。このことから, 類似解答ソースコード検索において Transformer のエンコーダ部分の利用は有効であることが期待できると考えられる。

BiLSTM に対する TRANS の評価指標の値はどのデータセット・入力形式の組合せにおいてもすべて上回っており, 類似解答ソースコード検索において Transformer のエンコーダ部分の利用は有効であることが期待できる。

4.2 調査課題 2: AST のベクトル表現は類似解答ソースコード検索において有効か。

調査内容. ソースコードをテキストベースで入力しベクトル化を行う検索器と AST ベースで入力しベクトル化を行う検索器の検索精度を比較し, ソースコードの AST のベクトル表現を用いることが類似解答ソースコード検索において有効であるかを確かめる。

結果と考察. 調査課題 2 の結果は表 6 のとおりである。表の値は TEXT の評価指標の値に対する AST の評価指標の値の比を表しており, TEXT の評価指標よりも AST の評価指標の方が高いときに 1 より大きくなる。SearchValidity (SV) に注目すると, ABC-A データに対しては TRANS が 0.681, BiLSTM が 0.969, ABC-B データに対しては TRANS が 1.000, BiLSTM が 1.636 となっている。SearchAccuracy (SA) に注目すると, ABC-A データに対しては TRANS が 1.067, BiLSTM が 1.222, ABC-B データに対しては TRANS が 0.925, BiLSTM が 1.064 と評価

表 5 BiLSTM に対する TRANS の各評価指標の比
Table 5 The ratios of the evaluation measures of TRANS to those of BiLSTM.

データセット	入力形式	SV	SA	VR
ABC-A	AST	1.241	1.089	1.118
ABC-A	TEXT	1.767	1.247	1.118
ABC-B	AST	1.422	1.112	1.118
ABC-B	TEXT	2.327	1.278	1.193

表 6 TEXT に対する AST の各評価指標の比

Table 6 The ratios of the evaluation measures of AST to those of TEXT.

データセット	モデル	SV	SA	VR
ABC-A	TRANS	0.681	1.067	1.000
ABC-A	BiLSTM	0.969	1.222	1.000
ABC-B	TRANS	1.000	0.925	1.028
ABC-B	BiLSTM	1.636	1.064	1.096

表 7 ABC-A データに対する ABC-B データの各評価指標の比

Table 7 The ratios of the evaluation measures of ABC-B to those of ABC-A.

入力形式	モデル	SV	SA	VR
AST	TRANS	0.653	1.476	1.021
AST	BiLSTM	0.570	1.445	1.021
TEXT	TRANS	0.444	1.702	0.993
TEXT	BiLSTM	0.337	1.661	0.931

指標の値にばらつきがある。ValidRatio (VR) についても TEXT 形式と AST 形式での大きな差はなかった。

この結果から、今回の調査では AST のベクトル表現を類似解答ソースコード検索に用いることの有効性を確認することはできなかった。AST が有効にはたらかなかった理由としては、ABC データに含まれるソースコードのサイズ (LOC) が小さいため特徴的な AST パスが現れにくくなり、相対的にテキスト情報の重要性が上がったためだと考えられる。また、AST パスを用いた学習はテキストベースの学習と比較して類似した特徴量 (AST パス) を何度も学習してしまう可能性があるため、過学習が発生してしまったことも考えられる。

AtCoder の解答ソースコードの AST のベクトル表現を用いることは、類似解答ソースコード検索において有効であるとはいえない。

4.3 調査課題 3：問題の複雑度は類似解答ソースコード検索に影響があるか。

調査内容. 問題の複雑度が異なる ABC-A データと ABC-B データそれぞれに対する検索精度を比較することで、問題の複雑度が類似解答ソースコード検索の精度に影響を与えるか確かめる。

結果と考察. 調査課題 3 の結果は表 7 のとおりである。表の値は ABC-A データの評価指標の値に対する ABC-B データの評価指標の値の比を表しており、ABC-A データの評価指標よりも ABC-B データの評価指標の方が高いときに 1 より大きくなる。SearchValidity (SV) に注目すると、AST-TRANS は 0.653、AST-BiLSTM は 0.570、TEXT-TRANS は 0.444、TEXT-BiLSTM は 0.337 とすべての組合せにおいて ABC-A データと比較して ABC-B データでの評価指標の値が低くなっている。一方で、SearchAccuracy (SA) に注目すると、AST-TRANS は 1.476、AST-BiLSTM は 1.445、TEXT-TRANS は 1.702、TEXT-BiLSTM は 1.661 とすべての組合せにおいて ABC-A データと比較して ABC-B データでの評価指標の値が高くなっている。

SearchValidity (SV) が低くなった理由としては、ABC-B データは ABC-A データと比較してソースコードの複雑度が高い分ソースコード間の類似度が低くなる傾向がある

ため、類似解答ソースコード検索を行うこと自体の妥当性が低くなっていると考えられる。一方で SearchAccuracy (SA) が高くなった理由としては、ソースコードの複雑度が高い分ソースコードに含まれる情報量が増え、より類似したソースコードを検索する性能が向上したと考えられる。ValidRatio (VR) はどの組合せにおいても他の指標と比較して ABC-A データと ABC-B データ間での差が小さいため、問題の複雑度による影響は小さいと考えられる。

問題の複雑度が上がると検索の正確度は上昇する一方、類似解答ソースコード検索を行うことの妥当性が下がる傾向がある。また、検索が妥当である割合については影響が小さい。

5. 妥当性に対する脅威

5.1 内的妥当性

本研究では学習データとして約 20,000 件の組合せデータを使用した。組み合わせる前のデータ数は 157 件と十分であるとはいえず、学習に偏りがある可能性がある。

また、各深層学習モデルの学習を行う際は損失関数の最小値が 10 エポックの間更新されなくなるまで学習を行ったため学習不足ではないと考えられるが、過学習についての検証は行っておらず、学習に偏りがある可能性がある。

他にも、本調査において最適化アルゴリズムや学習率などのパラメータについては最適化が行われておらず、より優れた組合せが存在する可能性がある。

5.2 外的妥当性

本研究では AtCoder Beginner Contest の A 問題および B 問題に対してのみ検証を行っており、C 問題以降やより難易度の高いコンテストの問題に対する一般性は保証されていない。同様に、AtCoder 以外のプログラミング問題に対する一般性も保証されていない。

また、AtCoder の各問題に対する解答ソースコードは公開されているものの中から実行時間が最も短いものを使用した。解答ソースコードといえるものは 1 通りではなく複数存在するため、これも一般性は保証されない。

6. おわりに

本研究では、AtCoder の問題を使用して木編集距離に注目した類似解答ソースコード検索器について実証的に調査を行った。今回の結果から、本研究の実験設定において (1) 類似解答ソースコード検索において Transformer のエンコーダ部分の利用は有効であることが期待できること、(2) AtCoder の解答ソースコードの AST のベクトル表現を使用することは類似解答ソースコード検索において有効であるとはいえないこと、(3) 問題の複雑度は類似解答

ソースコード検索に影響を与えることを示せた。よって、Transformer のエンコーダ部分を木編集距離に着目した類似解答ソースコード検索器のエンコード手法として活用することで、その精度を向上させることができるという我々の仮説は、本研究の実験設定において真であった。

本研究により、類似解答ソースコード検索に適した深層学習モデルの構成や、類似解答ソースコード検索の結果に影響を与える要因に関する知見を示すことができた。本調査における妥当性への脅威として用いたデータセットの小ささと、その妥当性への脅威ともなうモデル規模の小ささ（詳細は、付録 A.1 を参照されたい）があげられる。しかし、本実験の結果は、規模の小さなプログラムを検索する場合の 1 つの結果である。今後、より大規模で複雑なデータセットに対する研究を行っていく予定である。また、本調査では最適化アルゴリズムや学習率については最適化を行っていないため、今後それらを最適化していくことも必要であると考えられる。

今後の展望としては、1 つの問題に対する解答ソースコードを複数用いることや、GitHub などのより大規模で複雑なデータセットに対して調査を行うこと、理論的裏付けを持って、より適切な深層学習モデルや構成を選択して比較実験を行っていくことが考えられる。また、さらに発展した調査として、類似解答ソースコード検索器の精度が実際にプログラム自動生成の精度にどれほど影響を与えるかの検証も必要であると考えられる。

謝辞 本研究の一部は JSPS 科研費 JP18H04097・JP18H03222・JP21H04877, および, JSPS・国際共同研究事業 (JPJSJRP20191502) の助成を受けた。

参考文献

- [1] デジタルトランスフォーメーションに向けた研究会：DX レポート—IT システム「2025 年の崖」の克服と DX の本格的な展開 (2018).
- [2] Gulwani, S.: Automating String Processing in Spreadsheets Using Input-Output Examples, *Proc. 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '11*, pp.317–330, Association for Computing Machinery (2011).
- [3] 倉林利行, 吉村 優, 切貫弘之, 丹野治門, 富田裕也, 松本淳之介, まつ本真佑, 肥後芳樹, 楠本真二: 深層学習と遺伝的アルゴリズムを用いたプログラム自動生成, ソフトウェアエンジニアリングシンポジウム 2020 論文集, pp.143–152 (2020).
- [4] Linstead, E., Bajracharya, S., Ngo, T., Rigor, P., Lopes, C. and Baldi, P.: Sourcerer: Mining and Searching Internet-Scale Software Repositories, *Data Min. Knowl. Discov.*, Vol.18, No.2, pp.300–336 (2009).
- [5] Lv, F., Zhang, H., Lou, J., Wang, S., Zhang, D. and Zhao, J.: CodeHow: Effective Code Search Based on API Understanding and Extended Boolean Model (E), *Proc. 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp.260–270 (2015).
- [6] Antoniol, G., Canfora, G., Casazza, G., Lucia, A. and Merlo, E.: Recovering Traceability Links between Code and Documentation, *IEEE Trans. Softw. Eng.*, Vol.28, pp.970–983 (2002).
- [7] Gu, X., Zhang, H. and Kim, S.: Deep Code Search, *Proc. 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pp.933–944 (2018).
- [8] Xu, R., Xiong, C., Chen, W. and Corso, J.J.: Jointly Modeling Deep Video and Compositional Text to Bridge Vision and Language in a Unified Framework, *Proc. 29th AAAI Conference on Artificial Intelligence*, pp.2346–2352 (2015).
- [9] Sachdev, S., Li, H., Luan, S., Kim, S., Sen, K. and Chandra, S.: Retrieval on Source Code: A Neural Code Search, *Proc. 2nd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages, MAPL 2018*, pp.31–41, Association for Computing Machinery (2018).
- [10] Alon, U., Zilberstein, M., Levy, O. and Yahav, E.: code2vec: Learning Distributed Representations of Code, *Proc. ACM Program. Lang.*, Vol.3, No.POPL (2019).
- [11] Alon, U., Brody, S., Levy, O. and Yahav, E.: code2seq: Generating Sequences from Structured Representations of Code, *Proc. International Conference on Learning Representations* (2019).
- [12] Shi, K., Lu, Y., Chang, J. and Wei, Z.: PathPair2Vec: An AST path pair-based code representation method for defect prediction, *Journal of Computer Languages*, Vol.59, p.100979 (2020).
- [13] Cambronero, J., Li, H., Kim, S., Sen, K. and Chandra, S.: When Deep Learning Met Code Search, *Proc. 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2019*, pp.964–974, Association for Computing Machinery (2019).
- [14] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L. and Polosukhin, I.: Attention is All you Need, *Proc. Advances in Neural Information Processing Systems*, Vol.30, pp.5998–6008 (2017).
- [15] Bille, P.: A survey on tree edit distance and related problems, *Theoretical Computer Science*, Vol.337, No.1, pp.217–239 (2005).
- [16] Zhenzhong, L., Mingda, C., Sebastian, G., Kevin, G., Piyush, S. and Radu, S.: ALBERT: A Lite BERT for Self-supervised Learning of Language Representations, *Proc. International Conference on Learning Representations* (2020).
- [17] Devlin, J., Chang, M.-W., Lee, K. and Toutanova, K.: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, *Proc. 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Vol.1, pp.4171–4186 (2019).

付 録

A.1 使用モデルのパラメータなど

本章では、図 2 についての補足を行う。

A.1.1 使用ライブラリ

本調査では、深層学習ライブラリである TensorFlow (tf),

表 A.1 図中の名称と使用クラスの対応

Table A.1 The associations between the names in the figures and the classes that we used in our scripts.

名称	クラス名
Dense	tf.keras.layers.Dense
Embedding	tf.keras.layers.Embedding
Pooling	tf.keras.layers.GlobalAveragePooling1D
Concatenate	tf.keras.layers.Concatenate
Attention	tf.keras.layers.Attention
Dot	tf.keras.layers.Dot
BiLSTM	tf.keras.layers.LSTM
	tf.keras.layers.Bidirectional

表 A.2 本節での名称と使用クラスの対応

Table A.2 The associations between the names in the paper and the classes that we used in our scripts.

名称	クラス名
TimeDistributed	tf.keras.layers.TimeDistributed
MultiHeadAttention	tfa.layers.MultiHeadAttention

表 A.3 学習関連の使用クラス

Table A.3 The classes that we used to train the models.

項目	クラス名
モデル構築	tf.keras.Model
活性化関数	tf.keras.layers.ReLU
損失関数	tf.keras.losses.BinaryCrossentropy
最適化アルゴリズム	tf.keras.optimizers.Adam

TensorFlow Addons (tfa), および, Keras を用いて学習を行った. 図 2 内の名称と使用クラスの対応は表 A.1 に, 本節において追加で用いる名称と使用クラスの対応を表 A.2 に示した. また, 学習の際に使用したクラスについては表 A.3 に示した. 特に記載のないものはデフォルトのパラメータを使用している. 具体的な値については TensorFlow 公式リファレンスに示されている*5.

A.1.2 パラメータの設定

ABC データの作成の際に設定したパラメータを表 A.4 に示した. 前後のパラメータとの学習時間や評価指標の比較を行い, 総合的に判断を行い設定を行った.

A.1.3 層の次元数の設定

層の次元数について, 問題文入力部分は表 A.5 に, ソースコード入力部分 (AST) は表 A.6 に, ソースコード入力部分 (TEXT) は表 A.7 に示した (Pooling 層などは省略してある). Attention 層は Self-Attention を用いており, key と query はともに上層の出力を入力としている. 各 LSTM 層に対しては Bidirectional を用いることで双方向 LSTM 層を作成し, ソースコード入力部分 (AST) の

*5 https://www.tensorflow.org/api_docs

表 A.4 入力における設定項目

Table A.4 Input settings.

設定項目	上限
問題文の単語数	128
AST パスの数	64
AST パスの長さ	16
AST パスの深さの差	4
サブトークンの数	4

表 A.5 問題文入力部分の層の次元数

Table A.5 The number of dimensions of the text input layers of our models.

層	次元数
(ALBERT)	768
Dense (上部)	64
Dense (下部)	32

表 A.6 ソースコード入力部分 (AST) の層の次元数

Table A.6 The number of dimensions of the source code input layers (AST) of our models.

層	次元数
Embedding (末端)	8
Embedding (経路)	16
BiLSTM	
- LSTM	8
TRANS	
- MultiHeadAttention	16
- Dense	16

表 A.7 ソースコード入力部分 (TEXT) の層の次元数

Table A.7 The number of dimensions of the source code input layers (TEXT) of our models.

層	次元数
Embedding	32
BiLSTM	
- LSTM	16
TRANS	
- MultiHeadAttention	32
- Dense	32

表 A.8 調査で用いた計算資源の種類と数

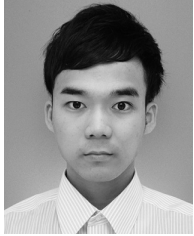
Table A.8 The type and the number of computing resources.

資源	種類と数
CPU	AMD Ryzen Threadripper 3990X 64-Core Processor 1 基
GPU	NVIDIA GeForce RTX 3090 1 基

Concatenate 層以前では TimeDistributed を用いて各 AST パスごとに処理を行っている. また, TRANS については, MultiHeadAttention のヘッド数は 4 であり, エンコーダの繰り返し部分は 2 層で構成されている. また, 最終層では, 1 次元の Dense 層を用いてシグモイド関数への入力を増幅している.

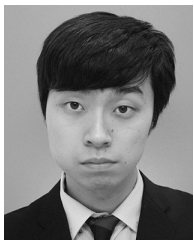
A.2 使用した計算資源

本調査で用いた計算資源について、表 A-8 に示した。



井野 健太郎

2021年九州大学工学部電気情報工学科卒業。現在、同大学大学院システム情報科学府情報理工学専攻修士課程在学中。プログラム自動生成分野やプログラミング教育分野に関する研究に従事。



松尾 春紀

2020年九州大学工学部電気情報工学科卒業。現在、同大学大学院システム情報科学府修士課程在学中。プログラム自動修正分野とプログラミング教育分野に関する研究に従事。



山本 大貴

2021年九州大学工学部電気情報工学科卒業。現在、同大学大学院システム情報科学府情報理工学専攻修士課程在学中。プログラミング教育分野に関する研究に従事。



近藤 将成 (正会員)

京都工芸繊維大学大学院工芸科学研究科設計工学専攻修了。博士(工学)。カナダ Queen's 大学で1年間訪問研究員として研究に従事。2021年度より九州大学大学院システム情報科学研究院助教。ソフトウェア工学領域で有

名な国際会議(ICSE, ICSME)や学術雑誌(EMSE)で数多くの研究成果を発表。過去に情報処理学会山下記念研究賞、ソフトウェアテスト技術振興協会善吾賞等を受賞。主にソフトウェアのバグ検出手法、ソフトウェアリポジトリマイニング、機械学習モデルの品質保証に関する研究に従事。



亀井 靖高 (正会員)

2005年関西大学総合情報学部卒業。2009年奈良先端科学技術大学院大学情報科学研究科博士後期課程修了。同年日本学術振興会特別研究員(PD)。2010年カナダ Queen's 大学博士研究員。2011年九州大学大学院システム情報科学研究院助教。2015年同大学同研究院准教授。博士(工学)。マイニングソフトウェアリポジトリ、ソフトウェアメトリクスの研究に従事。ESEM 2007 Best Paper Award, MSR 2014 Distinguished Paper Award, 2015年度情報処理学会論文賞等各賞受賞。ACM, ソフトウェア科学会, 電子情報通信学会各会員。IEEE Senior Member。



鵜林 尚靖 (正会員)

1982年広島大学理学部数学科卒業。1999年東京大学大学院総合文化研究科広域科学専攻広域システム科学系博士課程修了。博士(学術)。1982~2003年(株)東芝に勤務。2003年九州工業大学情報工学部助教授。2010年九州大学大学院システム情報科学研究院教授。現在に至る。2003年度情報処理学会山下記念研究賞受賞。ソフトウェア工学、プログラミング言語モデルの研究に従事。日本ソフトウェア科学会, 電子情報通信学会, ACM, IEEE-CS 各会員。本会フェロー。