木編集距離を用いた類似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)]

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中で，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，類似ソースコード検索において Transformer 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使用は有効性が低いこと，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

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] などが挙げられるが，自動化の対象が限定的であったり実装支援に留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窺える．

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挙げられる．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

[1] 九州大学
Kyushu University
[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/

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 節で研究の背景と目的について述べる．3 節では調査における実験の設計について述べる．4 節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 節では妥当性に対する脅威について述べ，6 節でまとめを行う．

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上で有利となると考えられる．論文では，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] を始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．

Gu ら [6] によると，情報検索に基づいたソースコード検索が抱える問題として，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．その例の一つに，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挙げられている．例えば，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．

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 }
```

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．例えば，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同士の関連度を学習する際に用いられる [7].

DeepCS[6] 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一つである．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

2.4 抽象構文木を用いた分散表現の獲得

code2vec[8] 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[9] 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．

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する類似ソースコードとし，その類似ソースコードを検索することを類似ソースコード検索と定義する．

類似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．本研究では，プログラム自動生成に適した木編集距離を学習に用いた類似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．調査内容は以下の通りである．

調査課題 1：Transformer の利用は類似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[10] の有効性の研究が進められているが，木編集距離に着目した類似ソースコード検索器でも有効であるかを確かめる．本調査課題では，従来の LSTM(Long Short Term Memory) 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することで，Transformer の利用が有効であるかを調査する．

調査課題 2：AST の利用は類似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 [8] が，類似ソースコード検索に対してもソースコードの AST 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する検索器と AST ベースで入力する検索器の検索精度を比較することで，AST の有効性を調査する．

調査課題 3：問題の複雑度は類似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，複雑度が類似ソースコード検索に与える影響はわかっていない．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．

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')
```

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 問題及び 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ーの提

[*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

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

出したソースコードが Web サイト上で全てのユーザー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．以降 ABC の A 問題で構成されたデータセットのことを 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言える．

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み合わせが存在することが挙げられる．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言える．

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）[11] 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同士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]．

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．

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

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み合わせに対しては 1 に，類似していない組み合わせは 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[9] 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全ての末端ノードの組み合わせに対して行う．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．この例の AST の末端ノードは 3 個であるため，この AST からは 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挙げられる．

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，それぞれの組み合わ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み合わせを変化させて調査を行い，各組み合わせごとに類似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 通りを組み合わせて作成した類似ソースコード検索器について調査を行う．

図 2 に各入力形式ごとの深層学習モデルの概略図を示

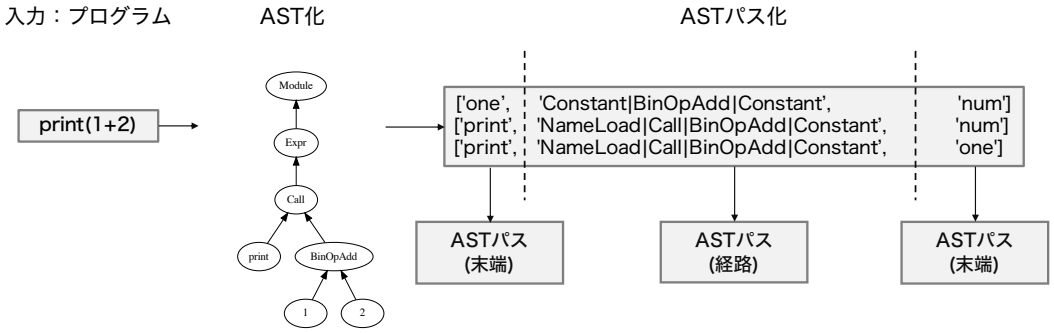

図 1 AST パス化の例

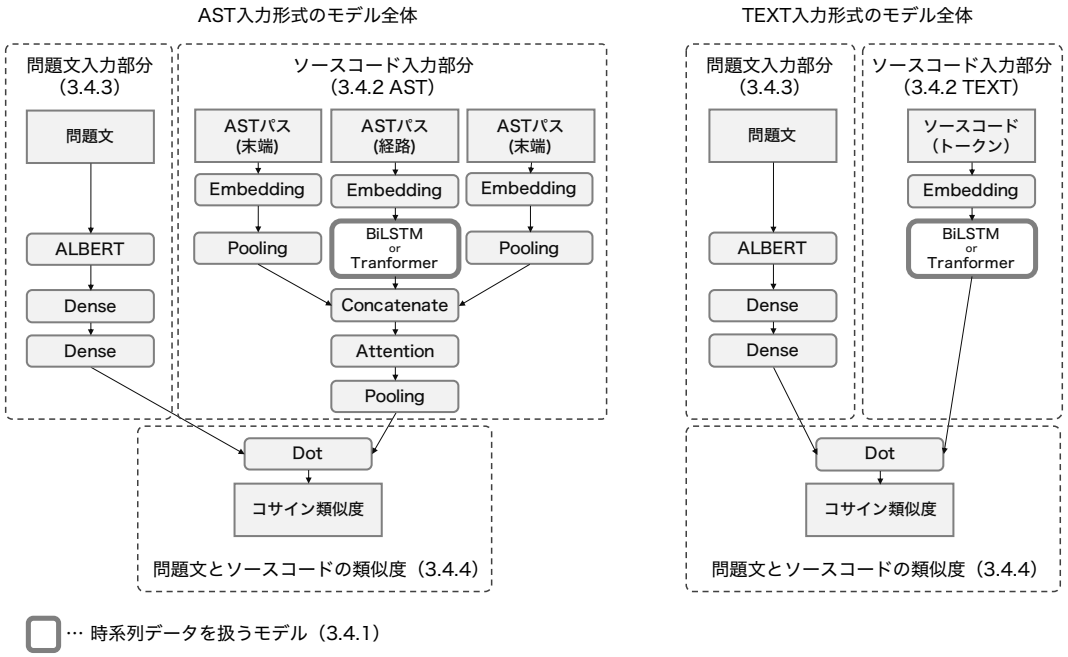

図 2 各入力形式のモデル構造の概略図

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[12] 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 層を選択した構造を持つ（AST-TRANS のように入力形式と組み合わせることで一つの深層学習モデルとなる）．Transformer 層は，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 [10]．本研究では，類似ソースコード検索において Transformer の利用が有効であるかを確かめるために用いる．

LSTM. LSTM 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 BiLSTM 層（双方向 LSTM 層）を選択した構造を持つ．BiLSTM 層は，Transformer ブロック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一つとして，Transformer 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 Transformer 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，類似ソースコード検索において AST 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．例えば，“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 Transformer 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一つとして，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[12] の学習済みモデルを使用している[*4]．ALBERT は BERT[13] 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．

$$cosine\_similarity(\mathbf{A}, \mathbf{B}) = \frac{\mathbf{A} \cdot \mathbf{B}}{|\mathbf{A}||\mathbf{B}|} \qquad (2)$$

このコサイン類似度を $[0, 1]$ の範囲に正規化した値が式 1 の値に近づくように学習を行うことで，類似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み合わせは，元々の問題とは異なる組み合わ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み合わせのデータを生成する．元々の組み合わせのデータと生成された異なる組み合わせのデータを合わせて 21609（$= 147^2$）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，各問題データの組み合わせに対して式 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 件との組み合わせデータを作成する．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

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

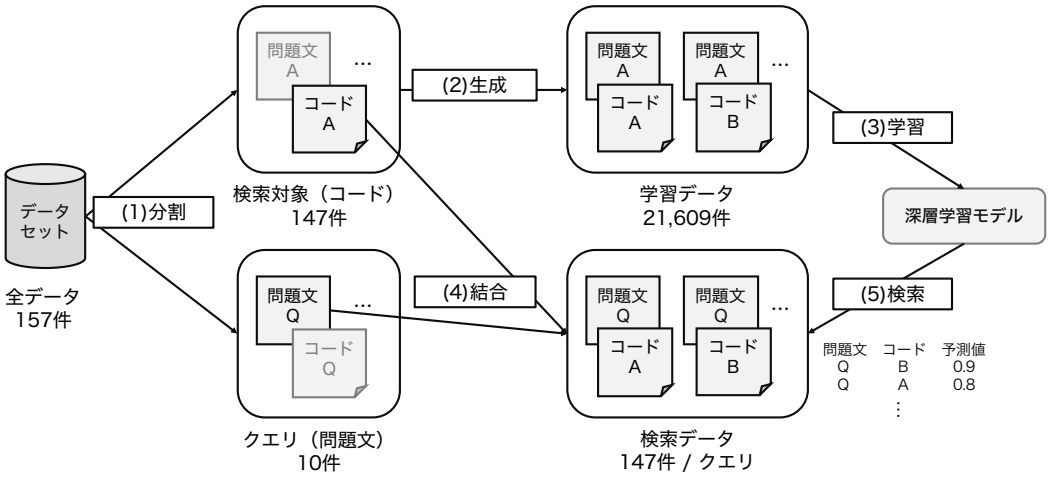

図 3 実験の全体像

表 2 評価指標の計算に用いる各 TED の定義

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.

めには TED がなるべく小さい類似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} は類似ソースコード検索器によらず一意に決まり, TED_{pred} は類似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ほど解答ソースコードとの類似度が高く, 検索した類似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言える. SearchValidity は以下の式で計算される. $TED_{pred} = 0$ の場合に最大値 1 を取り, $TED_{node} \leq TED_{pred}$ の場合に最小値 0 を取る.

$$SearchValidity = Max(1 - \frac{TED_{pred}}{TED_{node}}, 0) \qquad (3)$$

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言える. SearchAccuracy は以下の式で計算される. $TED_{pred} = TED_{best}$ の場合に最大値 1 を取り, $TED_{pred} = TED_{rand}$ の場合に 0 になる.

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

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ほど妥当な検索が可能であったクエリの割合が高いと言える. 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.

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

4. 結果と考察

本節では, 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, モデル構造の組み合わせにおいてクエリ 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み合わせの評価指標の値として表に示している. 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.

表 3 ABC-A データに対する各検索器の評価指標の値

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

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

表 5 LSTM に対する TRANS の各評価指標の比

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

4.1 調査課題 1： Transformer の利用は類似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 を用いた検索器の検索精度を比較し，Transformer の利用が類似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** の通りである．表の値は LSTM 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，LSTM 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）においても全ての組み合わせにおいて TRANS の評価指標の値が LSTM の値を上回っている．このことから，類似ソースコード検索において Transformer は有効であることが期待できると考えられる.

> LSTM に対する TRANS の評価指標の値はどのデータセット・入力形式の組み合わせにおいても全て上回っており，類似ソースコード検索において Transformer 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 は類似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する検索器と AST ベースで入力する検索器の検索精度を比較し，ソースコードの AST を用いることが類似ソースコード検索において有効であるか確かめる.

表 6 TEXT に対する AST の各評価指標の比

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

結果と考察. 調査課題 2 の結果は**表 6** の通りである．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，LSTM が 0.969，ABC-B データに対しては TRANS が 1.000，LSTM 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，LSTM が 1.222，ABC-B データに対しては TRANS が 0.925，LSTM が 1.064 と評価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 を類似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 データの規模では特徴的な AST パスの割合が小さく，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 を学習することは，類似ソースコード検索において有効であるとは言えない.

4.3 調査課題 3： 問題の複雑度は類似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しての検索精度を比較することで，問題の複雑度が類似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** の通りである．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-LSTM は 0.570，TEXT-TRANS は 0.444，TEXT-LSTM は 0.337 と全ての組み合わせにおいて 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

表 7 ABC-A データに対する ABC-B データの各評価指標の比

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

（SA）に注目すると，AST-TRANS は 1.476，AST-LSTM は 1.445，TEXT-TRANS は 1.702，TEXT-LSTM は 1.661 と全ての組み合わせにおいて 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）はどの組み合わ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 件の組み合わせデータを使用したが，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．

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が，解答ソースコードと言えるものは 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ソースコード検索器について調査を行った．今回の結果から，(1) 類似ソースコード検索において Transformer 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 を使用することは類似ソースコード検索において有効であるとは言えないこと，(3) 問題の複雑度は類似ソースコード検索に影響を与えることを示せた．

本研究により，類似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規模の小ささが挙げられる．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一つの結果である．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，さらに発展した調査として，類似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, *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '11, New York, NY, USA, Association for Computing Machinery, pp. 317–330 (online), DOI: 10.1145/1926385.1926423 (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), *In Proc. of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 260–270 (2015).
[6] Gu, X., Zhang, H. and Kim, S.: Deep Code Search, *In Proc. of the 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pp. 933–944 (2018).

[7] 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, *In Proc. of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, pp. 2346–2352 (2015).

[8] Alon, U., Zilberstein, M., Levy, O. and Yahav, E.: code2vec: Learning Distributed Representations of Code, *In Proc. of the ACM Program. Lang.*, Vol. 3, No. POPL (2019).

[9] Alon, U., Brody, S., Levy, O. and Yahav, E.: code2seq: Generating Sequences from Structured Representations of Code, *In Proc. of the International Conference on Learning Representations* (2019).

[10] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u. and Polosukhin, I.: Attention is All you Need, *In Proc. of the Advances in Neural Information Processing Systems*, Vol. 30, pp. 5998–6008 (2017).

[11] Bille, P.: A survey on tree edit distance and related problems, *Theoretical Computer Science*, Vol. 337, No. 1, pp. 217 – 239 (2005).

[12] Zhenzhong, L., Mingda, C., Sebastian, G., Kevin, G., Piyush, S. and Radu, S.: ALBERT: A LITE BERT FOR SELF-SUPERVISED LEARNING OF LANGUAGE REPRESENTATIONS., *In Proc. of the International Conference on Learning Representations* (2020).

[13] Devlin, J., Chang, M.-W., Lee, K. and Toutanova, K.: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, *In Proc. of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Vol. 1, pp. 4171–4186 (2019).