修士学位論文

題目

深層学習を用いたコードクローン検出器の汎化性能分析

指導教員
井上 克郎 教授

報告者
福家 範浩

令和 4 年 2 月 2 日

大阪大学 大学院情報科学研究科
コンピュータサイエンス専攻 ソフトウェア工学講座

令和 3 年度 修士学位論文

深層学習を用いたコードクローン検出器の汎化性能分析

福家 範浩

内容梗概

ソフトウェア保守の問題の要因の 1 つにコードクローンがある．コードクローンとはソースコード中に存在する同一，または類似した部分を持つコード片のことである．ソースコード中のコードクローンの箇所をソフトウェア開発者が把握しやすくするために，コードクローン検出器が提案されている．近年，字句だけが異なるコードクローンだけでなく，構文的に類似しているコードクローンや意味的に類似しているコードクローンも検出するコードクローン検出器が多く提案されている．そこで近年自然言語処理の分野などで発展している深層学習を取り入れたコードクローン検出器の提案が多く行われている．現在まで提案されてきた，深層学習を用いたコードクローン検出器では，1 つのベンチマークデータセットを学習用と精度計測用に分割して評価されている．しかし，深層学習の分野において学習データとは独立したデータを用いて精度計測を行った場合に性能が悪くなることが報告されており，汎化性能の問題が指摘されている．汎化性能の問題は深層学習を用いたコードクローン検出器に対しても起きる可能性があるが，既存の深層学習を用いたコードクローン検出器では学習と異なるベンチマークデータセットを用いて評価を行う，コードクローン検出器の汎化性能に関する調査が行われていない．

そこで，本研究では深層学習を用いた代表的なコードクローン検出器 CCLearner や ASTNN, CodeBERT に対し，BigCloneBench と Google Code Jam データセット，CodeNet の 3 つのコードクローンベンチマークデータセットを用いて汎化性能を分析した．本分析を行うために，次の 3 つの Research Question（RQ）を設定した．RQ1 では汎化性能の問題があるか調べるために，学習と評価に異なるデータセットを用いた場合に，同じデータセットで学習と評価した場合に比べて検出精度が維持できるか，を設定した．RQ2 ではコードクローンのタイプと汎化性能に関係がある場合にコードクローン検出器の学習法の改善のための情報になると考え，学習と評価に異なるデータセットを用いた場合に，コードクローンのタイプと検出精度は関係しているか，を設定した．RQ3 では類似した性質のデータに対して検出精度が維持できる場合に適用ソースコード群に類似する学習データを作成する動機になると考え，競技プログラミングを基にしたデータセット同士で学習と評価を行うことで，検出精度は維持できるか，を設定した．

その結果，学習と異なるベンチマークデータセットに対していずれのコードクローン検出器も精度が下がることや，CCLearner や ASTNN では類似度の高いコードクローンは精度が高く，類似度が低いコードクローンのタイプの精度は大幅に低い反面，CodeBERT はコードクローンの類似度に精度が関係していないを確認した．また，ASTNN や CodeBERT では性質が類似するベンチマークデータセットを用いることで精度の低下を抑えられることを確認した．

これらの結果から，コードクローン検出器に対しても汎化性能の問題があり，コードクローンのタイプごとに汎化性能が異なることがあることからコードクローンのタイプごとの分析を行うべきだと言える．また，学習データと類似した性質のデータで評価を行うことで精度が低下を抑えられるコードクローン検出器もあることから，類似するデータセット間での汎化性能の分析も行うべきだと言える．

主な用語

コードクローン検出器
深層学習
汎化性能

目 次

1　まえがき

コードクローンとは，ソースコード中に存在する同一，または類似した部分を持つコード片を示し，主に開発者が効率よく開発を行うために既存のコード片をコピーアンドペーストすることで生成される．一般的にコードクローンは，ソフトウェアの保守を困難にする要因の 1 つとして挙げられている．例えば，あるコード片にバグが存在するとした時，そのコード片に一致または類似するコード片にも同様のバグが含まれている可能性が高い．そのため，ソフトウェア開発者はバグを見つけた場合，見つけたコード片に対してだけではなく一致または類似するコード片を探し，それらに対してもバグの修正を検討をする必要がある．つまり，ソフトウェアの保守を行う際にコードクローンの位置を把握し，管理することで，ソフトウェアの保守性を高める可能性がある．しかし，大規模なソフトウェアの開発となると非常に多くのコードクローンが存在するため，ソフトウェア開発者が全てのコードクローンについて手作業で探すことは現実的ではない．そこで開発者はコードクローンを管理するためにコードクローン検出器を利用する [5,6,11,12]．

代表的な従来のコードクローン検出器には，CCFinder [14] や CCFinder の後継機である CCFinderX[1] などが挙げられる．これらの検出器は検出の前処理として字句解析を行うことで，字句単位でのコードクローンの検出を行う．しかし，ソースコード中で文の挿入や削除など大きな構造の違いがあるコードクローンや，異なる実装で処理が同じであるコードクローンを検出することは従来の検出器では困難である．そこで最近では，コード片の特徴量同士から深層学習を用いてコードクローンであるかを分類する CCLearner [15]，ASTNN [31]，CodeBERT [10] 等の検出器が提案されている．これら深層学習を用いた検出器では，事前にコードクローンのデータセットを用いて深層学習モデルの学習を行い，学習結果に基づいてコードクローンを検出する．しかし，実際にソフトウェア開発者が深層学習を用いたコードクローン検出器を利用するために，事前に学習したモデルを用いるが，コードクローン検出対象のソースコード群に対して学習させたソースコード群が類似しないため，検出精度が低い可能性がある．そのため，深層学習モデルを用いたコードクローン検出器は未知のデータに対する精度が高い，つまり，コードクローン検出器には汎化性能が高いことが求められている．しかし，これらの検出器の精度計測では 1 つのベンチマークデータセットの中から学習と計測のデータを取得しており，現実的な利用を想定した汎化性能を調べることが出来ていない．

そこで本稿では，特徴の異なる有名な深層学習を用いたコードクローン検出器 CCLearner，ASTNN，CodeBERT に対して，学習と検証に別のデータセットを用いて検出器の汎化性能の調査を行った．

[1] http://www.ccfinder.net/

以降，2 章では，本調査に関わるコードクローン検出器の関連研究や精度調査のためのコードクローンデータセットに関して説明する．3 章では，汎化性能を調べるための調査方法について説明を行う．4 章では，本分析で行った実験の結果について説明する．5 章では本分析の考察，6 章ではまとめを述べる．

2　背景

2.1　コードクローン

2.1.1　コードクローンとタイプ分類

ソースコード中の一致または類似するコード片対をコードクローンと呼ぶ．コードクローンは違いによって次のように大きく 4 つのタイプに分類されている． [20]

Type-1 コードクローン（以降 T1） Type1 のコードクローンは，空白やタブ・改行・コメント以外は一致しているコード片対である．Listing 1, 2 は，ソースコードの改行位置だけが括弧の前と後で異なるため，Type1 のコードクローンに分類される．

Type-2 コードクローン（以降 T2） Type2 のコードクローンは，Type1 のコードクローンの違いに加えて，変数名や関数名等のユーザー定義，変数の型の違いがあるコード片対である．Listing 1, 3 は，変数の名前と型が異なるため Type2 のコードクローンに分類される．

Type-3 コードクローン（以降 T3） Type3 のコードクローンは，Type2 のコードクローンの違いに加えて，文の挿入や削除変更等の違いを含むコード片対である．Listing 1, 4 は，文が変更されているため Type3 のコードクローンに分類される．

Type-4 コードクローン（以降 T4） Type4 のコードクローンは，Type3 のコードクローンの違いに加えて，同一の処理を行うが構文上の実装の違いを含むコード片対である．Listing 1, 4 は，同じ処理を行っており for 文で実装されていた所が while 文で実装されており Type4 のコードクローンに分類される．

Listing 1: コード片 1

```
1 int sum(int[] a){
2     int sum = 0;
3     for(int i=0; i<a.length; i++){
4         sum = sum + a[i];
5     }
6     return sum;
7 }
```

Listing 2: コード片 1 との Type1 コードクローン

```
1 int sum(int[] a)
2 {
3     int sum = 0;
```

```
    for(int i=0; i<a.length; i++)
    {
        sum = sum + a[i];
    }
    return sum;
}
```

Listing 3: コード片 1 との Type2 コードクローン

```
float sum(float[] b){
    float sum = 0;
    for(int i=0; i<b.length; i++){
        sum = sum + b[i];
    }
    return sum;
}
```

Listing 4: コード片 1 との Type3 コードクローン

```
int sum(int[] a){
    int sum = 0;
    for(int i=0; i<a.length; i++){
        sum += a[i];
    }
    return sum;
}
```

Listing 5: コード片 1 との Type4 コードクローン

```
int sum(int[] a){
    int sum = 0;
    int i = 0;
    while(i<a.length){
        sum = sum + a[i];
        i++;
    }
    return sum;
}
```

T3 は構文的な類似度によって更に細かく分類されている．構文的な類似度が 90%以上のコードクローンは Very Strong Type 3（以降 VST3），70%から 90%のコードクローンは Strongly Type 3（以降 ST3），50%から 70%のコードクローンは Moderately Type 3（以降 MT3），50%未満のコードクローンは Weakly Type 3（以降 WT3）のようにさらに細分化できる．また，VST3 は ST3 とひとまとめで ST3 として，WT3 は T4 とまとめて WT3/T4 として分類される．

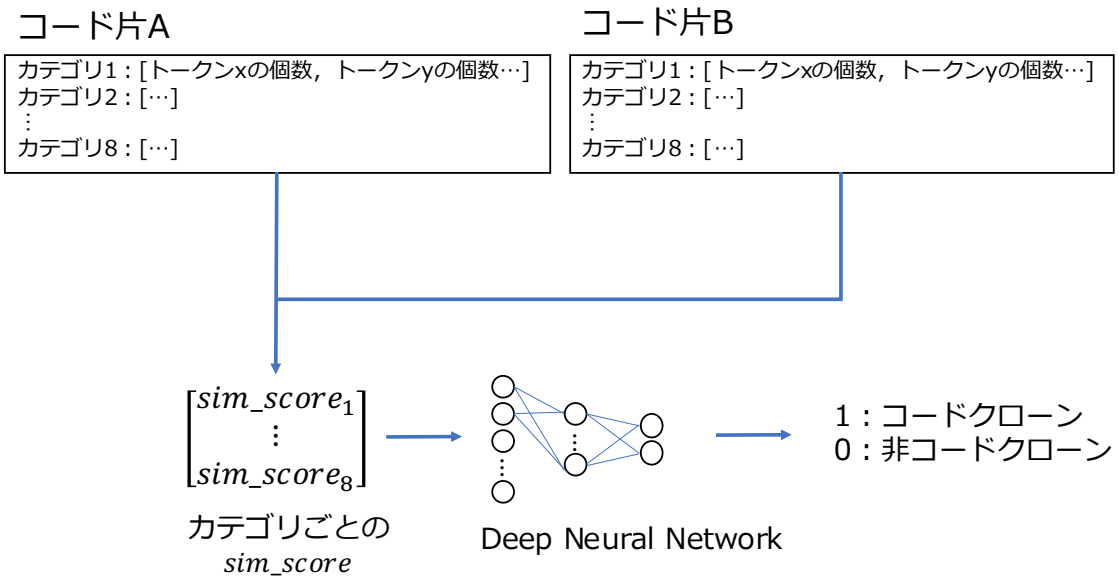


図 1: CCLearner の処理

2.1.2 コードクローン検出器

これまで，コードクローンを機械的に検出するために CCFinder 等のコードクローン検出器が提案されてきた．従来の検出器では，T1，T2 のコードクローンが高い精度で検出できる．しかし，T3 や T4 のコードクローンに対しては，構文的類似度が低く検出精度が低くなってしまうことが知られている．近年では，T3 や T4 のコードクローンに対してもコードクローンの検出精度を高めるために深層学習を用いたコードクローン検出器が多く提案されていて，代表的なコードクローン検出器として以下の CCLearner [15]，ASTNN [31]，CodeBERT [10] 等が挙げられる．

CCLearner 図 1 に示すように CCLearner はトークンベースの検出器である．まずは，CCLearner ではまずコード片をトークン分割し，分割されたトークンを予め決められたカテゴリに分類する．カテゴリは，予約語，演算子，マーカー，リテラル，タイプ識別子，メソッド識別子，修飾名，変数識別子の 8 つである．これらのカテゴリごとで，どのトークンが何個ずつあるかの頻度リストを取得する．コード片 A のあるカテゴリのトークン頻度リストを L_A とする．同様にコード片 B の頻度リストを L_B とする．

次にこのトークン頻度を用いてカテゴリごとの類似度 sim_score を計算する．類似度の計算式を以下の式 1 に示す．l_Ax をトークンリスト L_A から取得したトークン x の個数とする．$|l_Ax - l_Bx|$ は，あるトークン x についてコード片 A とコード片 B の個数差の絶対値である．$l_Ax + l_Bx$ は，個数の和である．これらをトークンごとに足し合わせて，差を和で割

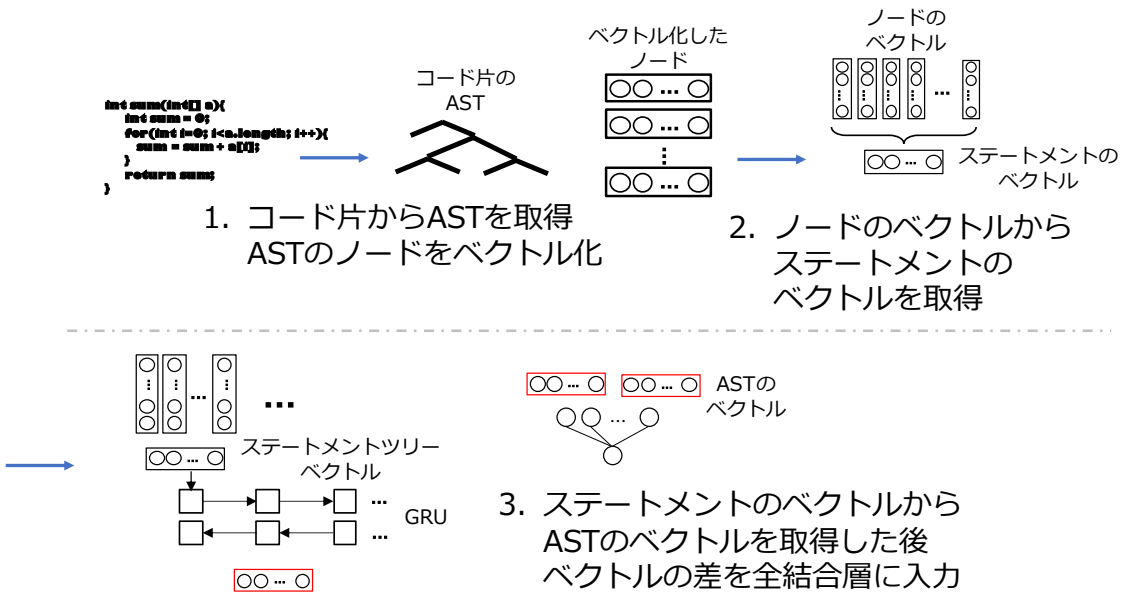


図 2: ASTNN の処理

る．これにより，完全に一致する場合は分数が 0 となり sim_score が 1 となる．仮にそのカテゴリのトークンが存在しない場合は，sim_score を 0.5 にする．これは CCLearner の論文によると，sim_score を 0 や 1 にしたときよりも精度が良かったからである．

$$sim\_score_i = 1 - \frac{\sum_x |l_A x - l_B x|}{\sum_x l_A x + l_B x} \tag{1}$$

最後に各々のトークンカテゴリごとの sim_score を Deep Neural Network に入力し，コードクローンであれば 1 を非コードクローンであれば 0 を出力するように学習する．

AST-based Neural Network（ASTNN） 図 2 に示すように ASTNN は抽象構文木（AST）の構造をベースとした検出器である．木構造を基にした深層学習モデルでは，Recursive Neural Nerwork [21] や Tree-based CNN [18]，Tree-based Long-Short-Term Memory [24] などが提案されていて，それらをもとに提案されたのが ASTNN である．ASTNN では，まずコード片から AST を生成し，より細かい木構造であるステートメントツリーを取得する．次にステートメントツリーのノード n に対する語彙ベクトル v_n を取得する．次にノードの隠れ状態である h を木構造に従って以下の式 2 のように計算する．

$$h = \sigma(W_n^\top v_n + \sum_{i \in [1,C]} h_i + b_n) \tag{2}$$

$W_n^\top \in \mathbb{R}^{d\times K}$ は単語の埋め込み次元 d，符号化次元 k の重み行列，b_n はバイアス項である．C はノード n の子のノードの数であり，$\sum_{i\in[1,C]} h_i$ は子ノード i の隠れ状態を足している．σ は，$tanh$ 等の活性化関数である．これらの隠れ状態 h を用いて，以下の式 3 から最終的なステートメントツリーの表現を得る．N はステートメントツリー t のノード数である．

$$e_t \quad = \quad [max(h_{i1}), \cdots, max(h_{ik})], i = 1, \cdots, N \tag{3}$$

その後，AST に含まれる T 個のステートメントツリーの表現を bidirectional GRU [25] に入力し，コード片のベクトル表現を得る．

最後に，2 つのコード片のベクトル r_1，r_2 間の距離 $r = r_1 - r_2$ を全結合層に入力し，活性化関数として sigmoid 関数を利用してコードクローンであれば 1 を非コードクローンであれば 0 を出力するように学習する．

CodeBERT　近年，自然言語処理の分野で，GLUE ベンチマーク [27] にて高いスコアを取った BERT [8] を用いた研究が盛んである．ソフトウェア工学の分野においても BERT から派生したモデルをソースコードに適用した CodeBERT などが提案されている．ここでは，BERT 及びソフトウェア工学の分野で用いられている CodeBERT について述べる．

BERT　BERT や BERT の基になった Transformer [26] 等の深層学習モデルが出る前は，自然言語処理を行う多くの深層学習モデルでは Recurrent Neural Netowrk（RNN）や Convolutional Neural Network（CNN）を基にしていた．Transformer では RNN や CNN のような再帰性や畳み込みの構造を取り入れず，Attention 機構のみで構成されたシンプルなネットワークアーキテクチャとして提案された．

BERT のモデルはこの Transformer を基にしており，Deep Bidirectional Transformers（Transformer の Encoder の部分）を用いている．また，語彙には WordPiece embeddings [29] が用いられている．BERT の大きな特徴としてモデルの学習方法があり，事前学習と fine tuning の 2 種類の学習の段階に分かれていることが挙げられる．事前学習には 2 種類の教師なしタスクがあり，1 つは Masked Language Modeling（Masked LM），もう 1 つは Next Sentence Prediction（NSP）である．

図 3 は，1 つ目のタスクである Masked LM のイメージ図である．入力される文章のうちいくつかのトークンを [MASK] トークンでマスク処理を行い，マスクされたトークンの ID を文脈に基づいて予測する．この事前学習で用いた [MASK] は，fine tuning 時には使わないため誤りが生じる可能性があるというデメリットがある．それを解消するために BERT では，トークン位置のうち 15%をランダムに選び，選ばれた i 番目のトークンの処理を以下のように行う．

図 3: BERT の Masked LM のイメージ図

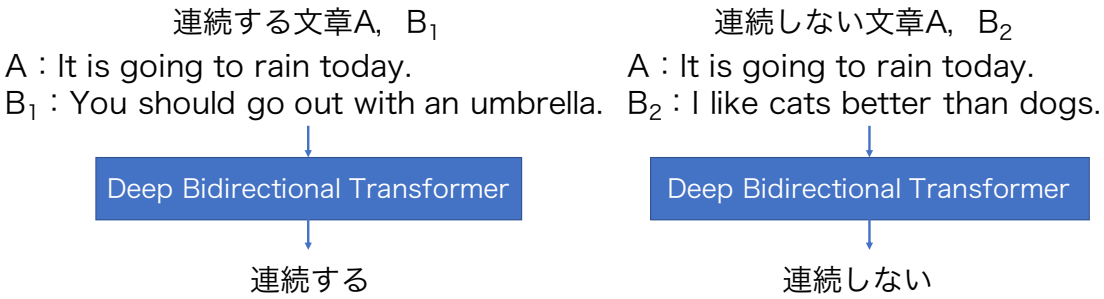


図 4: BERT の NSP のイメージ図

- 80%の確率で [MASK] トークンに置き換え
- 10%の確率でランダムなトークンに置き換え
- 10%の確率で置き換えない

図 4 は，もう 1 つのタスクである NSP のイメージ図である．入力された 2 文が連続するものかどうかを判定するタスクである．自然言語処理には質問応答（QA）や自然言語推論（NLI）等の 2 文の関係を理解することで行える下流タスクがあり，それらのために文の関係を理解するためのタスクとして NSP が取り入れられた．例文として入力される文章 B は，50%の確率で A から連続した文章である B_1 であり，残り 50%の確率でランダムな文章である B_2 である．

これらの事前学習を行ったあと，それぞれのタスクに合わせた固有の入出力で fine tuning を行う．fine tuning は事前学習に比べて比較的短時間で行え，GPU であれば数時間で学習可能である．

BERT と CodeBERT の違い　CodeBERT は BERT や BERT を改良した RoBERTa [16] を基にしており，モデル構造は RoBERTa-base と全く同じである．RoBERTa の事前学習では，BERT に用いられていた NSP の必要性が疑問視されており用いられていない．従って，CodeBERT の事前学習においても NSP は取り入れられていない．CodeBERT の事前

学習には，Masked LM と Replaced Token Detection（RTD）の 2 つのタスクが用いられている．CodeBERT の Masked LM では，自然言語とプログラミング言語のペアにマスク処理を行いマスクされた元のトークンを予測する学習を行う．RTD は自然言語とプログラミング言語のペアだけでなく，プログラミング言語だけのデータも用いて学習を行う．まず，ランダムにマスクされたトークンに対し，自然言語には自然言語の生成器を，プログラミング言語にはプログラミング言語の生成器を用いて，もっともらしく置き換えられるトークンを生成する．置き換えたデータを識別機であるモデルに入力し，どのトークンが取り替えられたトークンかを判定する．この 2 つの事前学習により CodeBERT は，言語モデルを学習する．事前学習に用いられたプログラミング言語は，Python, Java, JavaScript, PHP, Ruby, and Go の 6 つである．

2.1.3 コードクローンデータセット

コードクローン検出器を評価するためには，事前にコードクローンのタイプなどをタグ付けしたデータセットを用いる．有名なコードクローンのデータセットには，BigCloneBench や Google Code Jam，CodeNet などが挙げられる．

BigCloneBench（以降 BCB） 最も用いられるコードクローン検出器評価用のデータセットの 1 つに BigCloneBench [23]（以降 BCB）が挙げられる．BCB は Svajlenko らによって作成された Java ソースコードの大規模データセットで，プロジェクト間リポジトリの大規模データである IJaDataset 2.0 [4] の中から特定の機能のソースコードをマイニングし，それらに手動でクローンと非クローンのラベル付けをすることでコードクローン検出器のベンチマークとした．BCB には，T1・T2・T3・T4 のタイプ分類のタグやコード片同士の類似度もつけられているため，BCB を用いてクローンタイプごとの検出精度も求めることが出来る．

Google Code Jam データセット（以降 GCJ） 競技プログラミングサイト Google Code Jam に提出されたソースコードを利用してコードクローン検出器を評価する研究が行われている [22, 30, 32]．Zhao と Huang は，DeepSim の評価用に Google Code Jam に提出されたデータセットを公開している [32]．このデータセットを以降，GCJ と呼ぶ．このデータセットは，同じ設問に対して提出された解答ソースコードは，実装の上では異なるが行う処理は同じである T4 のコードクローンという仮定のもとに作成されている．設問数は 12 個で設問ごとの解答数には差があり，合計で 1669 個のプロジェクトである．コードクローン対は約 275,000 個あり，非コードクローン対は約 1,116,000 個ある．

CodeNet（以降 CN） Project CodeNet [19] は，ImageNet [7] が Computer Vision の分野にもたらしたことを AI for Code の分野にもたらすことを目的としたプロジェクトである．このデータセットは，競技プログラミングサイト AIZU Online Jadge [1] と AtCoder [2] から収集されたデータセットであり，主に C++，Python，Java，C，Ruby，C#の 6 つのプログラミング言語で構成されている．また，それらのソースコードのメタデータやトークン列や構文木コードグラフ等に変換するツールなどもデータセットとともに提供されている．これらのデータセットを用いてコード分類やコードクローンの検出を行うためのベンチマークデータセットも構成されており，有名なベンチマークデータセットの 1 つである POJ-104 [18] の約 10 倍のサイズの C++のベンチマークデータセットや Python ベンチマークデータセット，Java のベンチマークデータセットがある．Java のベンチマークデータセットには，250 の設問に対する解答ソースコードが 300 個ずつ含まれている．

2.2 既存の深層学習を用いたコードクローン検出器の問題点

近年，深層学習モデルを用いたシステムにおいて，学習データと独立した集団から精度計測のデータを取得した場合精度が悪くなることがあることが示されている．例えば，データを取得するためのセンサの違いで性能が下がる問題 [9] 等が報告されており，何が深層学習を用いたシステムの性能を下げる要因となるかは明確に分かっていない．深層学習を用いたシステムが異なるデータに対して精度が維持できない可能性は，深層学習を用いたシステムの性能の信頼性に影響する．そのため，深層学習を用いたシステムでは，異なるデータに対しても検出精度を維持できることが求められている．この，システムを汎用的に利用しても精度が維持できる性能のことをシステムの汎化性能と呼ぶ．

汎化性能の問題点は，コードクローン検出器に対しても当てはまると考えられる．ソースコードにもコーディングの経験の長さや，コーディング規約，トークンの類似度，保守するソースコードかどうか等，データセットに様々な要因が影響を与える可能性がある．既存の CCLearner や ASTNN，CodeNet など多くの深層学習を用いたコードクローン検出器では，同一のベンチマークデータセットを分割して学習データと精度計測のデータを作成している．しかし，同一のベンチマークデータセットでは，同じ競技プログラミングサイトから集めた GCJ や OSS から似た機能を持ったソースコードを抽出して作成した BCB 等，似た特徴のデータ間で検証を行っており，異なる特徴を持ったデータに対する精度を調べることが出来ていない．現状では，コードクローン検出器の汎化性能は明らかになっていない．仮にコードクローン検出器の汎化性能が低い場合，ソフトウェア開発者が実際に利用する時に検出精度が低く，ソフトウェア保守の手助けにならなくなってしまう．

3　汎化性能分析

本研究では，2.2 章で説明した問題を解決するために，深層学習を取り入れたコードクローン検出器の汎化性能について分析を行い，以下の 3 つのことを確かめたい．1 つ目は，学習と精度計測で異なるデータセットを用いた場合でも，同一データセット内での精度計測と同等の精度が出るのかを確かめ，コードクローン検出器の汎化性能調査が必要かを確かめる．2 つ目は，学習と異なるデータセットのコードクローンに対する検出精度はクローンのタイプごとに異なるのかを調査し，コードクローンのタイプごとの調査が汎化性能の調査に必要か確かめる．3 つ目は，学習データセットの選び方がコードクローン検出器の精度に影響するのかどうかについて確かめる．

そこで，本研究では，既存の深層学習を用いたコードクローン検出器である CCLearner や ASTNN，CodeBERT に対し，異なるベンチマークデータセットに対し，以下の 3 つの Research Question（RQ）を立てて汎化性能について調べた．

RQ1 学習と評価に異なるベンチマークデータセットを用いた場合，検出精度は維持できるか？

RQ2 学習と評価に異なるベンチマークデータセットを用いた場合，コードクローンのタイプと検出精度は関係しているか？

RQ3 競技プログラミングを基にしたデータセット同士で学習と評価を行うことで，検出精度は維持できるか？

RQ1 の調査により，同じデータセット内で評価した時の精度を維持できていないコードクローン検出器がある場合，深層学習を用いたコードクローン検出器には汎化性能の問題があると言える．これは，ソフトウェア開発者が実際に学習済みのコードクローン検出器を用いる時にコードクローンの検出精度が悪く，ソフトウェアの保守の手助けにならない可能性を示している．RQ2 の調査により分かる，コードクローンのタイプとコードクローンの検出精度の関係性は，検出精度の低いタイプのコードクローンの学習データを増やす等，コードクローン検出器の学習法改善のための情報になると考えられる．また，タイプごとの検出精度を調べることで，ソフトウェア開発者がコードクローン検出器の特性を理解し，それぞれにあったコードクローン検出器を選択出来ると考えられる．RQ3 の調査により，競技プログラミングを基にしたデータセット同士で学習と評価を行うことで検出精度が上がる場合，適用したいソースコード群に類似した性質を持つソースコード群で学習を行うことで検出精度が上がると考えられる．これは，ソフトウェア開発者が，適用したいソースコード群に類似した学習データセットを作成するための動機となる．

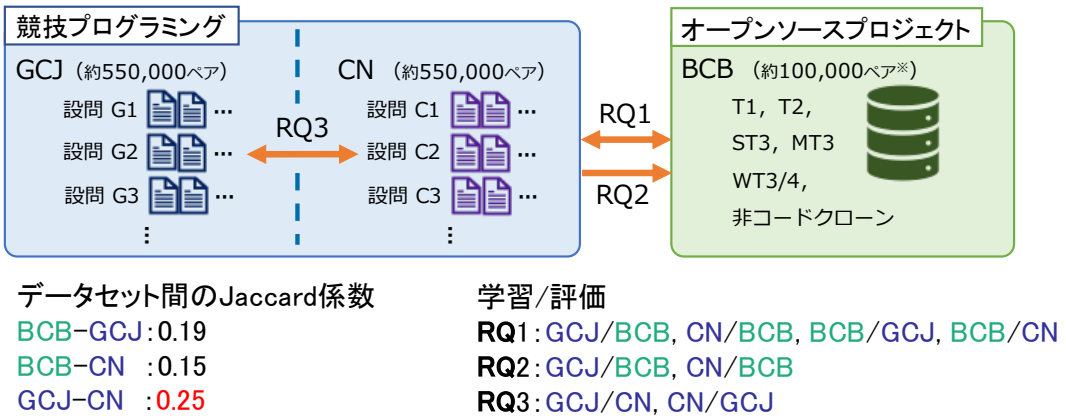


図 5: RQ に対する分析手順．RQ1 では，競技プログラミングサイトの解答ソースコードを基にしたデータセットとオープンソースプロジェクトのソースコード等を基にしたデータセットで双方向に学習と評価を行う．RQ2 では，コードクローンのタイプ分類が行われている BCB で精度計測，学習は BCB 以外のデータセットで行う．RQ3 では，競技プログラミングサイトの解答ソースコード同士で取得元が異なるソースコードを用いて学習と評価を行う．

3.1 手順

3.1.1 ベンチマークデータセットの準備

BigCloneBench BCB は CCLearner と ASTNN，CodeBERT のいずれのコードクローン検出器でも精度の計測に用いられていたベンチマークデータセットである．BCB にはバージョンがあり，バージョン 2 の BCB のほうがバージョン 1 の時の BCB の方がデータ数が多くなっている．また，それぞれのコードクローン検出器で，学習に用いるデータの抽出方法が異なっている．本調査では，それぞれの検出器で報告された BCB に対する精度と比較を行うために，BCB を用いた深層学習モデルの学習にはそれぞれの提案論文で提供されている BCB のデータセットを用いることにした．CCLearner では，BCB のデータベースから学習するソースコードのフォルダと精度計測するソースコードのフォルダを選択し，その中の T1，T2，VST3，ST3 のコードクローンと非コードクローンの約 45,000 個を用いてコードクローン検出器の学習を行った．ASTNN では，BCB のデータセットから T1，T2，ST3（VST3），MT3，WT3/T4 のコードクローンと非コードクローンをそれぞれのタイプで最大 20,000 個となるように抽出して学習と検証用のデータセットとしていた．CodeBERT については，提案論文ではコードクローンクローン検出は行われていない．しかし，後のコード生成と理解のためのベンチマークデータセットである CodeXGLUE [17] にて BCB

を用いて精度計測を行っていたため，そのデータを用いることにした．CodeXGLUE では，Kontogiannis らの論文 [28] に従って抽出した BCB のデータセットを用いており，BCB のコード片のうちコードクローンまたは非コードクローンのタグがつけられたコード片のみを残したデータセットである．トレーニングデータセットはコードクローンと非コードクローンが約 450,000 ずつの 901,028 個のデータである．CodeXGLUE で用いられた BCB のデータセットは，Hugging Face [3] にアップロードされており，コード片対の文字列とコードクローンか否かのラベルデータが取得出来る．そのコードクローンがどのタイプのコードクローンかは示されていない．

ベンチマークデータセット間での精度計測には，ASTNN で用いられていた BCB のデータセットを用いた．ASTNN と同様にコードクローンのタイプごとに精度を求めて，データセット全体におけるタイプごとの割合で重みをつけた精度を BCB 全体の精度結果とする方法を取った．

Google Code Jam データセット　本研究ではメソッド単位のコード片対のコードクローン検出を行う．GCJ は，競技プログラミングサイトに提出された解答ソースコードのファイルを基にして構成されたベンチマークデータセットである．GCJ に含まれているコードクローンのデータは解答ソースコードのファイルごとになっているため，ファイルからメソッドを抽出する必要がある．そこで本調査では，ソースコードファイルから main メソッドを抽出する．競技プログラミングサイトのソースコードを基にしたベンチマークデータセットは，同じ設問に対する解答ソースコードは意味的に類似している T4 のコードクローンであると仮定して作成されているため，main メソッドの外で定義されている処理もコードクローンかどうかの判定に用いられるべきと考えられる．そのため，main メソッドの外で定義されて用いられている関数は，main メソッドの中にインライン展開することにした．

また，競技プログラミング特有の処理として，問題を解くための入出力処理の文があることが挙げられる．入出力の形式は，競技プログラミングの設問によって異なることが多く，その入出力の形式を見るだけで同じ設問に対する解答ソースコードか否かを判断する可能性がある．T4 のコードクローンは同一の処理を行うが構文上の実装の違いを含むコード片対であり，コードクローン検出器はその処理からコードクローンか否かを判断することが求められている．そのため，本調査ではソースコードの入出力に関する部分を取り除いた．

最後に，異なる設問に対する解答ソースコードは非コードクローンとしているため，コードクローンよりも非コードクローンの方が非常に多くなるため，GCJ では，コードクローンと同じ数の非コードクローンをランダムに抽出してデータセットとした．

ベンチマークデータセット GCJ の設問数は 12 問，解答ソースコード数は 1,669 個，コードクローン数は約 275,000，非コードクローン数は約 275,000 である．

CodeNet　CN は GCJ と同様に競技プログラミングサイトのソースコードを基に作成されたベンチマークデータセットであるため，GCJ と同様の処理を行った．また，設問数とコードクローン数を GCJ と同程度にするために，CN に付与されていた問題番号の若い順に問題を取得した．その時に，処理が 1 行で書かれていることが多い問題，再帰関数などインライン展開のために大きな変更が必要なコード等をデータセットから外した．CodeNet の元のデータセットには AIZU OnlineJudge と AtCoder の問題の両方の問題があるが，本実験のデータセットには AIZU Online Judge の問題を 12 問，それぞれの解答ソースコード数 215 個，合わせてコードクローン数約 276,000，非コードクローン数も約 276,000 のデータセットとした．

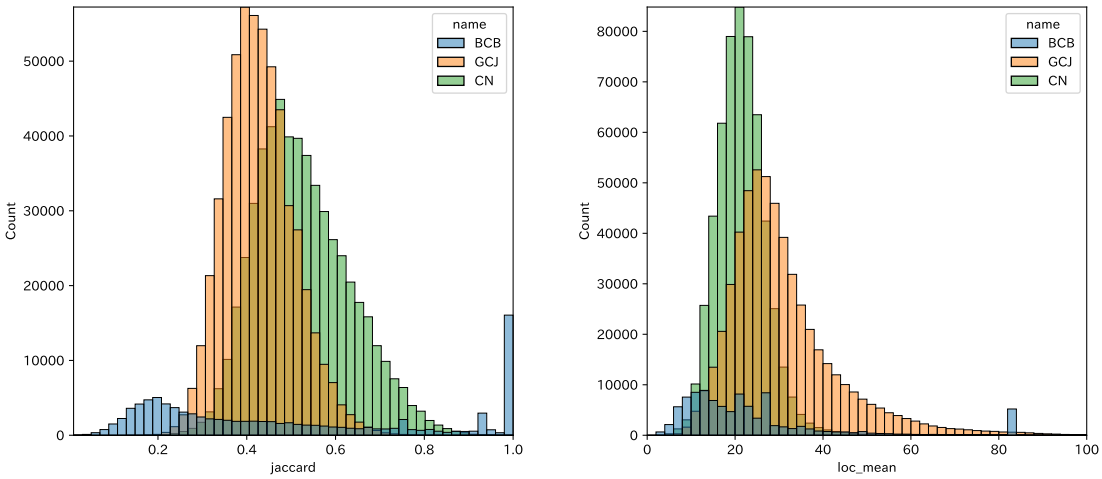
3.1.2　ベンチマークデータセットの分析

ここではベンチマークデータセットの分析を行う．図 6 は，BCB，GCJ，CodeNet の 3 つのベンチマークデータセットのコード片対の Jaccard 係数とコード行数（LOC）の平均のヒストグラムである．図 7，8 は，ASTNN で使われている BCB と CodeXGLUE で使われている BCB の Jaccard 係数と LOC のヒストグラムである．図 7 の ASTNN の BCB はラベル 0 から順に非コードクローン，T1，T2，ST3，MT3，WT3/T4 となっていて，図 8 の CodeXGLUE の BCB はラベル True がコードクローンで False が非コードクローンである．BCB，GCJ，CN の大半の LOC は 100 以下のため，図を見やすくするために，図 6，7，8 では LOC の最大値を 100 とした．表 1，2 は，GCJ，CodeNet のベンチマークデータセットのそれぞれの設問のコード片対の Jaccard 係数と LOC の平均の中央値である．

BigCloneBench　図 6 が示す通り，LOC は低く，Jaccard 係数は 0.2 と 1 のところが山となっている．BCB のラベルごとである図 7 から分かるように，Jaccard 係数が 1 が多いのは T1 のコードクローンであり，LOC が大きいものは T1 のコードクローンが中心となっていることが分かる．

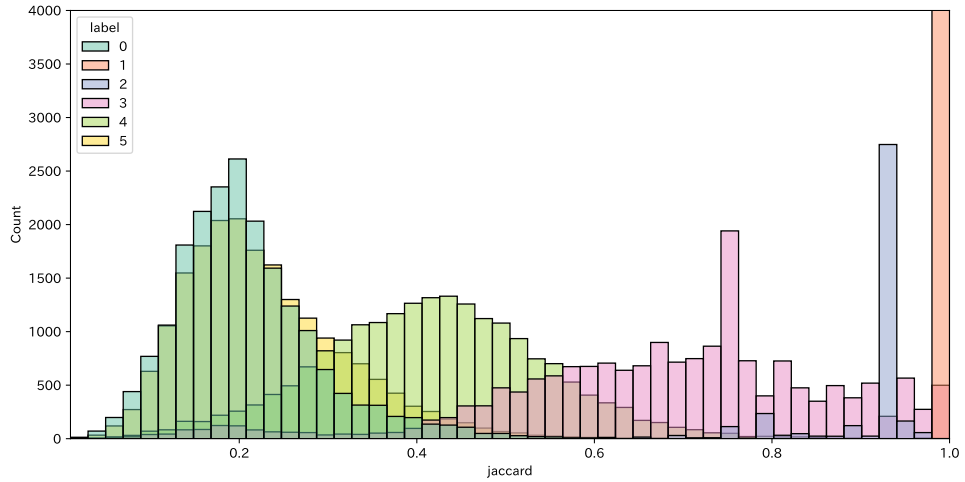
Google Code Jam データセット　図 6 が示す通り，LOC は BCB よりも大きく，Jaccard 係数は 0.4 程度のところが高くなっている．GCJ の設問ごとの表 1 から分かるように，設問ごとの Jaccard 係数はあまり差は無く，LOC は設問によっては長い．

CodeNet　図 6 が示す通り，LOC は BCB よりも大きく GCJ よりも小さい．Jaccard 係数は，0.5 程度のところが高くなっている．CN の設問ごとの表 2 から分かるように，Jaccard 係数は低い設問もあるが多くは 0.5 程度のところが高くなっている．LOC はどの設問も大きな違いは見られない．

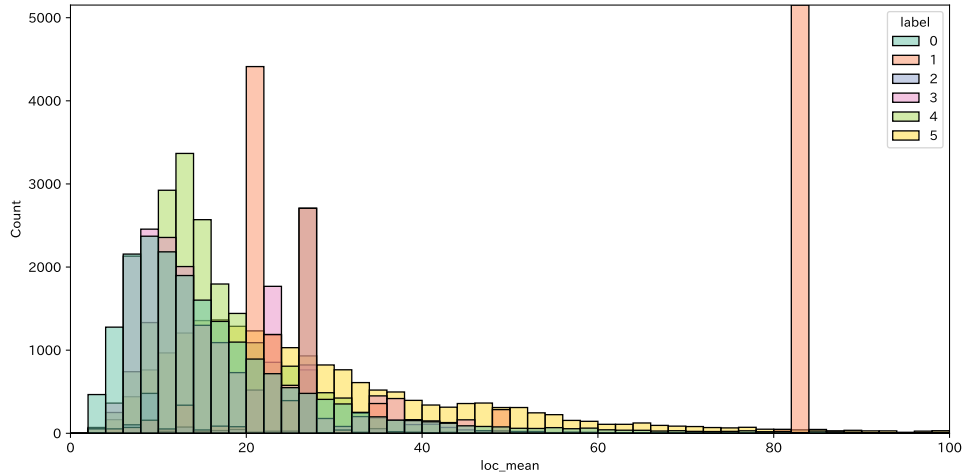


(a) Jaccard 係数　　(b) LOC の平均

図 6: ベンチマークデータセット BCB，GCJ，CN 全体の分析．青が BCB，橙が GCJ，緑が CDN．LOC の平均は 100 以上となるコード片対は少ないため，図の評価しやすさの都合上横軸の上限を 100 とした．Jaccard 係数は BCB が一番低く，GCJ，CN の順になっている．ただし，BCB のタイプ 1 コードクローンは Jaccard 係数 1 となるため BCB は 1 にも山がある．LOC の平均は，BCB の低い方の山が一番低く，CN，GCJ，BCB の高い方の山の順になっている．

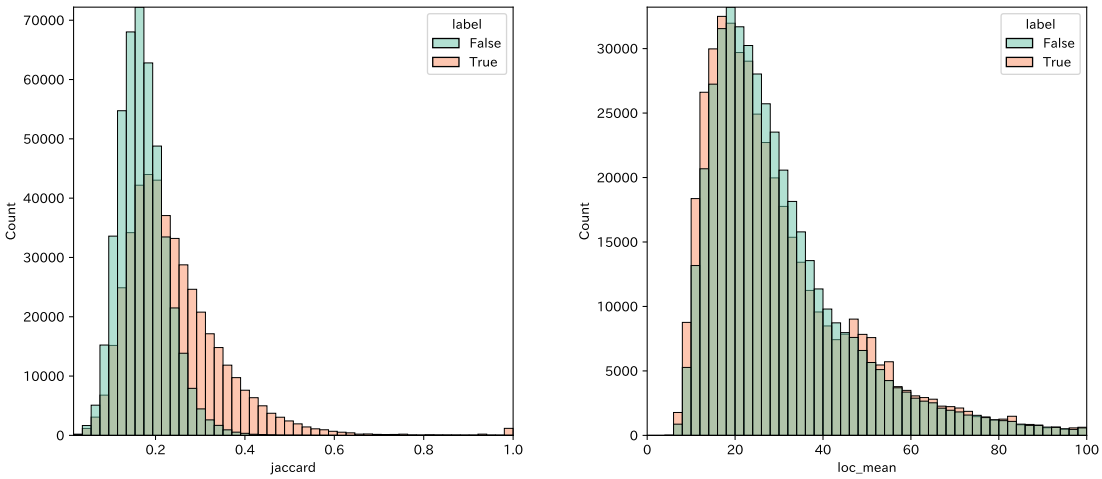


(a) Jaccard 係数．Jaccard 係数 1 は約 20,000 個あるが，図の評価しやすさの都合上縦軸の上限を 4,000 とした．コードクローンのタイプごとの中央値は，T1 が 1.00，T2 が 0．93，ST3 が 0.70，MT3 が 0.42，WT3/T4 が 0.21，非コードクローンが 0.19.



(b) LOC の平均．100 以上となるコード片対は少ないため，図の評価しやすさの都合上横軸の上限を 100 とした．

図 7: ASTNN の BCB のクローン Type ごとのベンチマークデータセットのヒストグラム．ラベル 0 から順に，非コードクローン，T1，T2，ST3，MT3，WT3/T4 である．



(a) Jaccard 係数．中央値はコードクローンが 0.22，非コードクローンは 0.17.

(b) LOC の平均

図 8: CodeXGLUE の BCB のベンチマークデータセットのヒストグラム．ラベル False が非コードクローン，True がコードクローンである．LOC の平均は 100 以上となるコード片対は少ないため，図の評価しやすさの都合上横軸の上限を 100 とした．

表 1: GCJ の設問ごとのコードクローンの分析

設問	個数	Jaccard 係数の中央値	LOC の平均
非コードクローン	274048	0.388	30
1	113526	0.478	27
2	3828	0.417	28
3	29161	0.471	15
4	703	0.462	66.5
5	1	0.455	68
6	93961	0.457	25
7	351	0.365	58
8	29890	0.423	49.5
9	2278	0.452	36.5
10	153	0.413	58.5
11	190	0.489	35
12	6	0.424	38

表 2: CN の設問ごとのコードクローンの分析

設問	個数	Jaccard 係数の中央値	LOC の平均
非コードクローン	276060	0.463	21
1	23005	0.627	23
2	23005	0.579	22.5
3	23005	0.587	23.5
4	23005	0.596	25.5
5	23005	0.556	27.5
6	23005	0.604	20.5
7	23005	0.480	16.5
8	23005	0.571	16
9	23005	0.580	15
10	23005	0.582	20
11	23005	0.596	22.5
12	23005	0.607	18.5

表 3: ベンチマーク頻出 1,000 語間の Jaccard 係数

ベンチマーク 1	ベンチマーク 2	Jaccard 係数
BCB	GCJ	0.1947
BCB	CN	0.1541
GCJ	CN	0.2516

3.1.3 ベンチマークデータセット間の類似度計算

Jaccard 係数と LOC の観点から GCJ と CN は，GCJ と BCB または CN と BCB よりも特徴が似ていることが図 6 等から分かった．ここでは，更にベンチマークの頻出語同士での Jaccard 係数を計測する．まず，コード片ごとに使われているトークンのリストを作る．ベンチマークの中であるトークンが使われている頻度を求め，その頻度の上位 1,000 トークンを選ぶ．その後，それら 1,000 トークンの Jaccard 係数を求める．表 3 は，その求めた Jaccard 係数の表である．BCB と GCJ や BCB と CN は Jaccard 係数が 0.1947，0.1541 と，GCJ と CN の Jaccard 係数 0.2516 よりも低く，GCJ と CN は BCB とよりも類似している．

3.1.4 コードクローン検出器の学習と精度計測

ここでは，3つのRQ全てに関わることを先に述べ，その後RQごとの説明を行う．CCLearnerとASTNNの提案論文では，コードクローン検出器から出力される値が1に近ければコードクローン，0に近ければ非コードクローンと識別する．CCLearnerとASTNNではBCBでコードクローン検出器を学習した場合，それぞれ閾値が0.98と0.5と設定している．しかし，BCB以外のデータセットで学習した検出器で閾値を変えずに使った場合，コードクローンの検出精度が非常に悪くなった．そこで，それぞれのデータセットで正答率が高くなる値を探索して閾値とすることとした．また，CodeBERTでは，BERTで文のペアの分類として使われている方法と同様に，コード片とコード片の間に[SEP]トークンを挟んで入力し，コードクローンを判定出来るようにfine tuningした．

RQ1　RQ1では，それぞれのコードクローン検出器に対し，BCBで学習を行いGCJまたはCNで精度計測を，また逆にGCJまたはCNで学習を行いBCBで精度計測を行った．CCLearnerではBCBの精度計測を著者がサンプリングして行っているため，BCBで学習したCCLearnerの精度比較には提案論文での結果を用いた．

RQ2　RQ2では，それぞれのコードクローン検出器に対し，コードクローンのType分類が行われているBCBで精度計測を行ったデータを用いて，コードクローンのTypeのごとの精度を調べた．また，コードクローンタイプごとのJaccard係数の中央値と，3.2章で示すMCCとの相関を調べたBCBのコードクローンのタイプごとのJaccard係数は，T1が1.00，T2が0.929，ST3が0.705，MT3が0.418，WT3/T4が0.213となっている．

RQ3　RQ3では，それぞれのコードクローン検出器に対し，GCJで学習を行いCNで精度計測を，また逆のCNで学習を行いGCJで精度計測を行った．

3.2 評価指標

本調査では，評価指標にprecision，recall，f1，Matthews Correlation Coefficient（MCC）を用いた．評価指標を説明する上でのtrue positiveやtrue negative，false positive，false negativeについて以下で説明する．

true positive（tp）　コードクローンのうち，検出されたコード片

true negative（tn）　非コードクローンのうち，検出されなかったコード片

false positive（tp）　非コードクローンのうち，誤って検出されたコード片

false negative（fn） コードクローンのうち，検出されなかったコード片

precision precision は日本語で適合率と呼ばれ，コードクローンと予測したコード片対のうち実際にコードクローンだったコード片対の割合である．precision の計算式は以下のようになる．

$$precision = \frac{tp}{tp + fp} \tag{4}$$

recall recall は日本語で再現率と呼ばれ，実際にコードクローンであるコード片対のうちどれだけのコードクローンを予測することが出来たかの割合である．recall の計算式は以下のようになる．

$$recall = \frac{tp}{tp + fn} \tag{5}$$

f1 f1 は，precision と recall の調和平均である．precision または recall のどちらかだけが極端に数値が高い場合，ほとんどのコード片対に対してコードクローンまたは非コードクローンと判断している可能性がある．それらを解消するための指標の 1 つに f 値があり，以下の式 (6) で表される precision と recall の調和平均は f1 と呼ばれる指標である．調和平均では，どちらかの値だけが極端に大きい場合にも低い結果となる．検出器の性能の比較を行うには，主にこの指標が用いられることが多い．

$$f1 = 2\frac{precision \cdot recall}{precision + recall} \tag{6}$$

Matthews Correlation Coefficient（MCC） 本調査では GLUE のタスクの 1 つである CoLA などにも用いられている MCC も評価指標に用いた．MCC は混同行列を 1 つの値にまとめることができ，クラスごとに不均衡なデータに対する性能評価としても用いられている． [13] f1 の場合，precision と recall に true negative の値が含まれておらず，positive と negative の数に違いがある場合に精度は悪いが評価が悪くならない可能性がある．式 7 が MCC の計算式である．

$$MCC = \frac{(tp \times tn) - (fp \times fn)}{\sqrt{(tp + tn) \times (tp + fn) \times (fp + tn) \times (fp + fn)}} \tag{7}$$

MCC の値が 1 の場合正確に予測できており，-1 の場合全く逆の予測をしている．0 の場合が最悪で，正確な予測でも全て反対の予測でも無い予測である．

4　分析結果

本章では，本研究で行った分析の結果を述べる．

4.1　RQ1：異なるベンチマークデータセットに対する検出精度

学習と評価に異なるベンチマークデータセットを用いた場合，検出精度は維持できるか？
表 4，図 9 は，RQ1 の実験結果である．まず，コードクローン検出器ごとに精度の計測結果を示し，RQ1 全体について述べる．

CCLearner　BCB で学習し BCB で精度計測したデータは，CCLearner の提案論文 [15] での値である．提案論文では precision と recall と f1 のみ検出精度が計測されている．BCB で学習した時の結果について述べる．f1 の値は，同じデータセットを使用して評価すると 0.93 だったのに対し，GCJ で評価すると 0.109，CN で評価すると 0.357 といずれも検出精度が下がっていた．recall の値は，GCJ で評価した時の値は 0.0577，CN で評価した時の値は 0.223 であり，同じデータセットで評価した時の WT3/T4 の値より高く，MT3 の値よりは低くなっている．GCJ または CN で学習した時の結果について述べる．f1 の値は，GCJ または CN のいずれのデータセットで学習した時も同じデータセットで評価すると約 0.55 だったが，異なるデータセットである BCB で評価すると約 0．1 と大幅に下がった．MCC の値は，GCJ または CN のデータセットで学習した時，同じデータセットで評価すると約 0.3 から 0.35 だったが，異なるデータセットである BCB で評価すると約 0.05 から 0.1 と大幅に下がった．これらのことから CCLearner は，学習とは異なるデータセットで評価をした時に検出精度が，同じデータセットで評価した時に比べて下がることが分かった．

ASTNN　BCB で学習した時の結果について述べる．f1 の値は，同じデータセットを使用して評価すると 0.939 だったのに対し，GCJ または CN で評価するといずれも約 0.66 と大幅に下がった．MCC の値は，同じデータセットを使用して評価すると 0.891 だったのに対し，GCJ または CN で評価するといずれも 0.03 以下と大幅に下がった．特に precision が約 0.5 で recall が約 1 なことから，ほとんどのコード片対に対してコードクローンと判断していることが分かった．GCJ または CN で学習した時の結果について述べる．f1 の値は，GCJ または CN のいずれのデータセットで学習した時も同じデータセットで評価すると約 0.5 だったが，異なるデータセットである BCB で評価すると GCJ で学習した時は 0.372，CN で学習した時は 0.0900 といずれも下がっていた．MCC の値は，GCJ または CN のいずれのデータセットで学習した時も同じデータセットで評価すると約 0.45 だったが，異なるデータセットである BCB で評価すると GCJ で学習した時は約 0.001，CN で学習した時は約 − 0.15 と

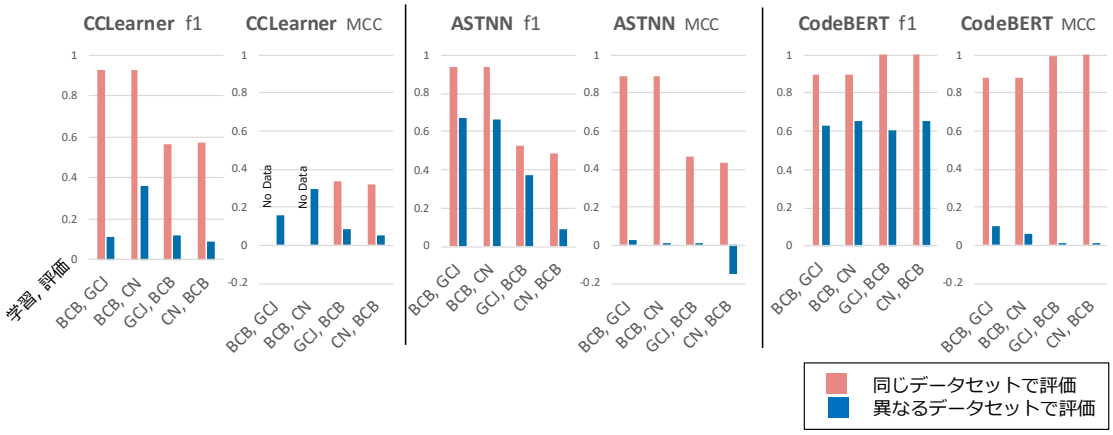


図 9: RQ1：学習と同じデータセットで評価した時と異なるデータセットで評価した時の f1 の値と MCC の値．CCLearner に対し BCB で学習し BCB で評価した時の値は CCLearner [15] での値で，MCC は計測されていない．

大幅に下がった．これらのことから ASTNN は，学習とは異なるデータセットで評価をした時に検出精度が，同じデータセットで評価した時に比べて下がることが分かった．

CodeBERT　BCB で学習した時の結果について述べる．f1 の値は，同じデータセットを用いて評価すると 0.896 だったのに対し，GCJ または CN で評価するといずれも約 0.65 と大幅に下がった．MCC の値は，同じデータセットを用いて評価すると 0.881 だったのに対し，GCJ または CN で評価するといずれも 0.1 以下と大幅に下がった．GCJ または CN で学習した時の結果について述べる．f1 の値は，GCJ または CN のいずれのデータセットで学習した時も同じデータセットで評価すると約 1 だったのに対し，異なるデータセットである BCB で評価すると GCJ で学習した時は約 0.6，CN で学習した時は約 0.65 と大幅に下がった．MCC の値は，CJ または CN のいずれのデータセットで学習した時も同じデータセットで評価すると約 1 だったのに対し，異なるデータセットである BCB で評価するといずれも約 0.004 と大幅に下がった．これらのことから CodeBERT は，学習とは異なるデータセットで評価をした時に検出精度が，同じデータセットで評価した時に比べて下がることが分かった．

RQ1 の答え

CCLearner，ASTNN，CodeBERT のいずれのコードクローン検出器も，学習と評価に異なるデータセットを用いると検出精度が同じデータセットで評価した時に比べて下がることを確認した．これらのことから，深層学習を用いたコードクローン検出器に対しても，汎化性能の問題があるということが確認出来た．

表 4: RQ1 類似度の低いベンチマークデータセット間でのコードクローンの検出精度

検出器	データセット 1	データセット 2	precision	recall	f1	MCC
CCLearner	BCB	BCB[a]	0.93	0.89[b]	0.93[c]	—[d]
	BCB	GCJ	0.946	0.0577	**0.109**	0.159
	BCB	CN	0.891	0.223	**0.357**	0.296
	GCJ	GCJ	0.761	0.451	0.566	0.340
	GCJ	BCB	0.681	0.0668	**0.116**	**0.0847**
	CN	CN	0.732	0.469	0.572	0.318
	CN	BCB	0.611	0.0504	**0.0874**	**0.0500**
ASTNN	BCB	BCB	0.999	0.886	0.939	0.891
	BCB	GCJ	0.501	0.997	**0.667**	**0.0275**
	BCB	CN	0.501	0.982	**0.663**	**0.0129**
	GCJ	GCJ	1.00	0.353	0.522	0.463
	GCJ	BCB	0.498	0.299	**0.372**	**0.00101**
	CN	CN	1.00	0.316	0.481	0.434
	CN	BCB	0.263	0.0569	**0.0900**	**-0.150**
CodeBERT	BCB	BCB[e]	0.882	0.911	0.896	0.881
	BCB	GCJ	0.530	0.778	**0.630**	**0.0988**
	BCB	CN	0.512	0.895	**0.651**	**0.0612**
	GCJ	GCJ	0.999	0.998	0.998	0.997
	GCJ	BCB	0.500	0.774	**0.608**	**0.00429**
	CN	CN	0.998	1.00	0.999	0.998
	CN	BCB	0.500	0.933	**0.651**	**0.00432**

[a] BCB で学習し，BCB で精度計測したデータは，CCLearner [15] で報告された値.

[b] ST3 の *recall* の値．詳細は，T1/1.00，T2/0.98，ST3/0.89，MT3/0.28，WT3/T4/0.01.

[c] recall は，T1-ST3 を使用（$recall_{T1-ST3}$ は，0.93）.

[d] CCLearner の提案論文では，MCC は計測されていなかった.

[e] CodeBERT の精度計測をした BCB は，コードクローン約 50,000，非コードクローン約 360,000 と非コードクローンが多い．他の検出器と同様にコードクローンと非コードクローンの数を合わせて計測すると，*precision* が 0.980，*recall* が 0.911，$f1$ が 0.944，*mcc* が 0.895 となった.

4.2 RQ2：コードクローンタイプごとの検出精度

学習と評価に異なるベンチマークデータセットを用いた場合，コードクローンのタイプと検出精度は関係しているか？

RQ2 では，それぞれのコードクローン検出器について，GCJ または CN で学習を行い BCB でのタイプごとの精度の計測結果を調査した．図 10 は，それぞれの検出器の MCC の値をグラフにしたものである．

CCLearner 表 5 は，CCLearner を GCJ または CN で学習し BCB で精度計測した時のコードクローンのタイプごとの結果である．MCC の値は，GCJ と CN のいずれのデータセットで学習した時も，異なるデータセットである BCB で評価するとコードクローンのタイプが T1 から T4 へ類似度が低くなっていくに従って下がった．コードクローンのタイプのそれぞれの Jaccard 係数の中央値と MCC の値の相関係数は，GCJ で学習した場合が 0.982，CN で学習した場合が 0.997 となった．これらのことから，CCLearner は学習とは異なるデータセットでの評価において，コードクローンのタイプと検出精度に強い相関関係があることが分かった．

RQ1 において BCB で学習し異なるデータセットである GCJ または CN で評価した時の recall の値が MT3 に対する値よりも低く WT3/T4 に対する値よりも高かったことについて言及する．RQ2 でコードクローンのタイプごとに検出精度を調べたことにより，MT3 または WT3/T4 ではわずかに recall の値が上がることもあるが，ST3 では recall の値が大幅に下がっていた．ST3 のコードクローンに対してもともと recall は約 0.9 と検出精度がもとは高く，MT3 より類似度が低いコードクローンに対しては recall が 0.3 以下と大きく低かったことを考えると，CCLearner の汎化性能は低いと言える．

ASTNN 表 6 は，ASTNN を GCJ または CN で学習し BCB で精度計測した時のコードクローンのタイプごとの結果である．MCC の値は，GCJ，CN のいずれのデータセットで学習した時も，異なるデータセットである BCB で評価するとコードクローンのタイプが T1 から T4 へ類似度が低くなっていくに従って下がる傾向にあった．コードクローンのタイプのそれぞれの Jaccard 係数の中央値と MCC の相関係数は，GCJ で学習した場合が 0.970，CN で学習した場合が 0.995 となった．ASTNN は，学習とは異なるコードクローンに対するコードクローンの検出において，コードクローンのタイプと検出精度に強い相関関係があることが分かった．

CodeBERT 表 7 は，CodeBERT を GCJ または CN で学習し BCB で精度計測した時のコードクローンのタイプごとの結果である．MCC の値は，GCJ，CN のいずれのデータセッ

表 5: RQ2 BCB 以外を用いて CCLearner の学習を行い，BCB を用いて精度計測した結果

(a) CCLearner を GCJ で学習し，BCB で精度計測した結果

Type	precision	recall	f1	**MCC**
T1	0.963	0.992	0.977	0.961
T2	0.846	0.903	0.874	0.853
ST3	0.948	0.595	0.731	0.626
MT3	0.931	0.400	0.560	0.462
WT3/T4	0.676	0.0567	0.105	0.0745

(b) CCLearner を CN で学習し，BCB で精度計測した結果

Type	precision	recall	f1	**MCC**
T1	0.963	0.999	0.981	0.967
T2	0.859	0.994	0.922	0.911
ST3	0.955	0.685	0.798	0.699
MT3	0.873	0.203	0.330	0.281
WT3/T4	0.606	0.0417	0.0781	0.0408

表 6: RQ2 BCB 以外を用いて ASTNN の学習を行い，BCB を用いて精度計測した結果

(a) RQ2 ASTNN を GCJ で学習し，BCB で精度計測した結果

Type	precision	recall	f1	**MCC**
T1	0.724	1.00	0.840	0.713
T2	0.381	0.999	0.552	0.517
ST3	0.715	0.814	0.761	0.518
MT3	0.611	0.466	0.529	0.174
WT3/T4	0.495	0.292	0.367	-0.00596

(b) RQ2 ASTNN を CN で学習し，BCB で精度計測した結果

Type	precision	recall	f1	**MCC**
T1	0.845	1.00	0.916	0.851
T2	0.559	0.990	0.715	0.687
ST3	0.788	0.579	0.667	0.456
MT3	0.516	0.152	0.235	0.0129
WT3/T4	0.256	0.0492	0.0825	-0.159

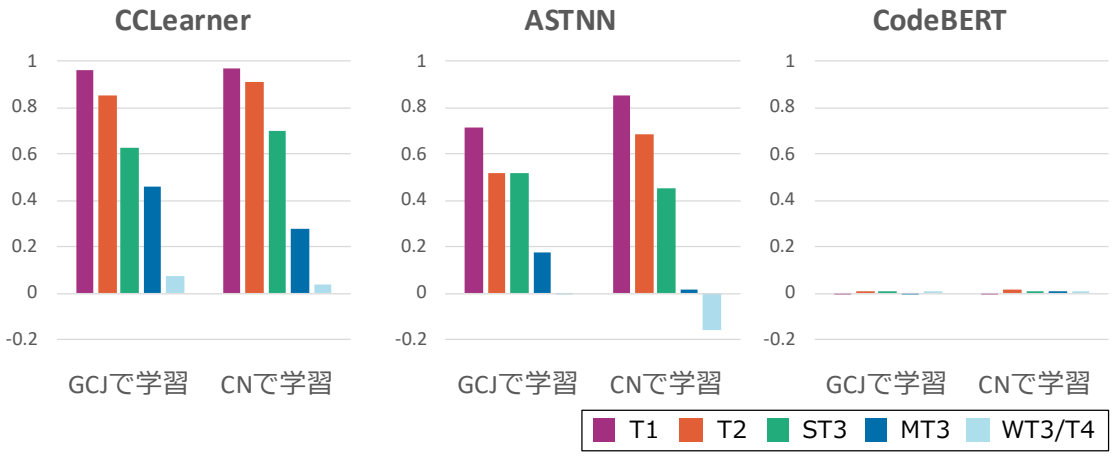


図 10: RQ2：学習と異なるデータセットで評価した時のコードクローンのタイプごとの MCC の値

トで学習した時も，異なるデータセットである BCB で評価するとコードクローンのタイプごとに精度はほとんど差がなく約 0 であった．コードクローンのタイプそれぞれの Jaccard 係数の中央値と MCC の相関係数は，GCJ で学習した場合が −0.308，CN で学習した場合が 0.217 となった．CodeBERT は，学習とは異なるコードクローンに対するコードクローンの検出において，コードクローンのタイプと検出精度にほとんど相関関係が無いことが分かった．

RQ2 の答え

CCLearner や ASTNN ではコードクローンのタイプと検出精度には関係があり，CodeBERT ではコードクローンのタイプと検出精度にほとんど関係がないことが分かった．このことから，汎化性能にコードクローンのタイプが関係しているコードクローン検出器と関係していないコードクローン検出器があることが分かった．

4.3 RQ3：類似するベンチマークデータセット間での検出精度

競技プログラミングを基にしたデータセット同士で学習と評価を行うことで，検出精度は維持できるか？

RQ3 では，各検出器の検出結果を述べたあと RQ3 全体を通した結果を述べる．図 11 は，それぞれの検出器の MCC の値をグラフにしたものである．

CCLearner MCC の値は，GCJ または CN で学習を行い同じデータセットで評価した時にいずれも 0.3 は超えていたが，競技プログラミングを基にした異なるデータセットで評

表 7: RQ2 BCB 以外を用いて CodeBERT の学習を行い，BCB を用いて精度計測した結果

(a) RQ2 CodeBERT を GCJ で学習し，BCB で精度計測した結果

Type	precision	recall	f1	**MCC**
T1	0.437	0.770	0.558	-0.000266
T2	0.155	0.772	0.258	0.00156
ST3	0.479	0.774	0.592	0.00390
MT3	0.500	0.769	0.606	-0.00144
WT3/T4	0.501	0.774	0.608	0.00438

(b) RQ2 CodeBERT を CN で学習し，BCB で精度計測した結果

Type	precision	recall	f1	**MCC**
T1	0.437	0.930	0.595	-0.00114
T2	0.156	0.942	0.268	0.0166
ST3	0.478	0.931	0.632	0.0000778
MT3	0.500	0.931	0.651	0.000612
WT3/T4	0.501	0.933	0.652	0.00438

価した場合，0.3 を下回っており検出精度は下がった．OSS を基にしたデータセットである BCB で学習した場合との比較を行う．GCJ で評価を行った時の MCC の値は，BCB で学習した検出器で 0.159，CN で学習した検出器で 0.166 と同等で，CN で評価を行った時の MCC の値は，BCB で学習した検出器で 0.296，GCJ で学習した検出器で 0.293 と同等だった．このことから，学習したデータセットに類似したデータセットを用いても，CCLearner の検出精度はほとんど変化しないことを確認した．

ASTNN　MCC の値は，GCJ または CN で学習を行い同じデータセットで評価した時にいずれも約 0.45 であったが，競技プログラミングを基にした異なるデータセットで評価した場合，CN 学習 GCJ 評価で約 0.12，GCJ 学習 CN 評価で約 0.25 と検出精度は下がっている．OSS を基にしたデータセットである BCB で学習した場合との比較を行う．GCJ で評価を行った時の MCC の値は，BCB で学習した検出器で約 0.03，CN で学習した検出器で約 0.12，CN で評価を行った時の MCC の値は，BCB で学習した検出器で約 0.01，GCJ で学習した検出器で約 0.25 と，競技プログラミングを基にしたデータセット同士で学習と評価を行った方が検出精度が上がっている．これらのことから，ASTNN では学習と異なるデータセットを用いることで検出精度は下がるが，学習と類似した性質のデータセットで評価を行うことで検出精度が上がることを確認した．

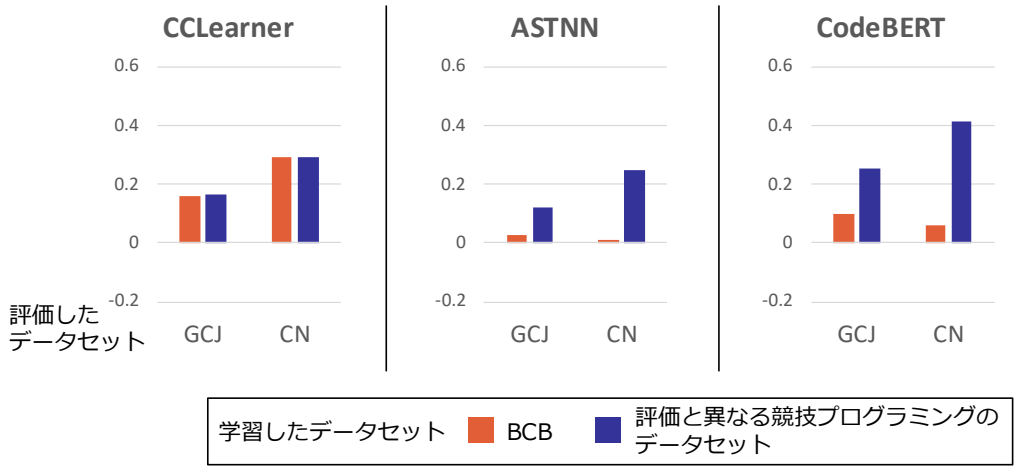


図 11: RQ3：OSS 等を基にした BCB で学習し競技プログラミングを基にしたデータセットで評価した時と，競技プログラミングを基にしたデータセット同士で学習と評価した時の MCC の値

CodeBERT　MCC の値は，GCJ または CN で学習を行い同じデータセットで評価した時にいずれも約 1 であったが，競技プログラミングを基にした異なるデータセットで評価した場合，CN 学習 GCJ 評価で約 0.25，GCJ 学習 CN 評価で約 0.42 と検出精度は下がっている．OSS を基にしたデータセットである BCB で学習した場合との比較を行う．GCJ で評価を行った時の MCC の値は，BCB で学習した検出器で約 0.10，CN で学習した検出器で約 0.25，CN で評価を行った時の MCC の値は，BCB で学習した検出器で約 0.06，GCJ で学習した検出器で約 0.42 と，競技プログラミングを基にしたデータセット同士で学習と評価を行った方が検出精度が上がっている．これらのことから，CodeBERT では学習と異なるデータセットを用いることで検出精度は下がるが，学習と類似した性質のデータセットで評価を行うことで検出精度が上がることを確認した．また，ASTNN に比べて精度が高いことも確認した．

RQ3 の答え

いずれのコードクローン検出器も同じデータセット内で学習と評価をした時の精度を維持できてはいなかった．しかし，ASTNN や CodeBERT では類似するデータセットに対してはコードクローン検出の精度が上がる傾向にあることが分かった．CCLearner に関しては，類似するデータセットに対してはコードクローンの検出精度がほとんど変わらないことが分かった．このことから，ASTNN と CodeBERT では，適用したいソースコード群と類似した性質のデータセットを用いることで，同じデータセット内で精度計測した場合に比べると精度は下がるが，検出精度が上がる可能性があることが分かった．

表 8: RQ3 RQ1 よりも類似しているベンチマークでの精度計測結果

検出器	データセット 1	データセット 2	precision	recall	f1	MCC
CCLearner	BCB	GCJ	0.946	0.0577	0.109	0.159
	CN	GCJ	0.664	0.266	0.380	0.166
	BCB	CN	0.891	0.223	0.357	0.296
	GCJ	CN	0.686	0.526	0.595	0.293
ASTNN	BCB	GCJ	0.501	0.997	0.667	0.0275
	CN	GCJ	0.920	0.0375	0.0721	**0.121**
	BCB	CN	0.501	0.982	0.663	0.0129
	GCJ	CN	0.867	0.177	0.294	**0.247**
CodeBERT	BCB	GCJ	0.530	0.778	0.630	0.0988
	CN	GCJ	0.590	0.782	0.672	**0.252**
	BCB	CN	0.512	0.895	0.651	0.0612
	GCJ	CN	0.668	0.806	0.731	**0.415**

5 考察

5.1 分析結果の考察

本実験の 3 つの RQ から以下のことが分かった.

- RQ1 から本実験で調査したコードクローン検出器は汎化性能が悪いこと
- RQ2 からコードクローンのタイプによってコードクローンの検出精度が変わる検出器と変わらない検出器があること
- RQ3 から学習データと評価データの性質が類似していることで，コードクローンの検出精度が上がる検出器があること

このことから，深層学習を用いたコードクローン検出器を実際の開発環境に取り入れる場合，事前にコードクローン検出器の汎化性能を調べておく必要があると言える．また，仮にコードクローンの検出精度が維持できていたとしても，コードクローンのタイプによって汎化性能が維持できているかどうかが異なっている可能性があるため，コードクローンのタイプごとの検出精度も確かめ，苦手とするコードクローンのタイプの把握と改善を行うべきである．また，適用したいソースコード群に類似した性質のデータセットを用いることで検出精度が向上する可能性があるため，複数のデータセットを用いて検出精度を調査し，適用させたいデータに対して学習データがどうあるべきか等の分析を行うべきであると言える．

5.2 妥当性への脅威

本実験では，提案論文で使用された学習データが最適であると考え提案時の BCB を使用し，その他の学習データには一般的に用いられている GCJ と新たなベンチマークとして公開された CN を使用した．また，BCB 以外のベンチマークデータセットで学習し BCB で精度計測する時には，どの検出器も同じデータで計測出来るように ASTNN で用いられていた BCB を用いた．これらのデータが精度に影響していることは考えられるが，本実験で主張するコードクローンの汎化性能の低さに大きな影響を及ぼすことはないと考えられる．また，今回選択した深層学習を用いたコードクローン検出器は，互いに異なる特徴を持った検出器を選んでおり，パラメータも固定して実験を行ったが，これも本実験で主張するコードクローンの汎化性能の低さに大きな影響を及ぼすことはないと考えられる．実験に使用した評価指標も同様に，precision，recall，f1 はコードクローンの研究において一般的に用いられており，MCC については機械学習の分野などで一般的に用いられている評価指標であり，本実験の結果に影響を及ぼすことはないと考えられる．

6　まとめ

本分析によって，深層学習を用いたコードクローン検出器に対しても汎化性能の問題があることが分かった．また，コードクローンのタイプによって，異なるデータセットに対する検出精度が変わる検出器と，変わらない検出器があることが分かった．また，検出器の学習に適用したいソースコード群と類似した性質のデータを用いることで，コードクローンの検出精度が上がる傾向にあることが分かった．

今後，さらなるデータセット，コードクローン検出器，評価指標を用いてコードクローン検出器の汎化性能について調査，分析を行うことで，深層学習を用いたコードクローン検出器の汎化性能についての研究が発展していくであろう．

謝辞

本研究を行うにあたり，御多忙の中懇切なる御指導，御助言を賜りました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 ソフトウェア工学講座 井上 克郎 教授に，心から感謝の意を評します.

本研究の全過程を通して，研究の方向性の検討，実験，分析における問題点の提示，論文の執筆など，手厚く御指導，御助言を賜りました名古屋大学大学院情報学研究科附属組込みシステム研究センター 吉田 則裕 准教授，ならびに京都工芸繊維大学情報工学・人間科学系 崔 恩瀞 助教に心から感謝の意を表します.

本研究の全過程を通して，研究の方向性の検討，分析における問題点の提示など，懇篤な御指導，御助言を賜りました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 ソフトウェア工学講座 松下 誠 准教授，春名 修介 特任教授ならびに神田 哲也 助教 に心から感謝の意を表します.

本研究の全過程を通して，実験，分析における問題点の提示，論文の執筆，発表について様々な御意見，御鞭撻を賜りました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 ソフトウェア工学講座 藤原 裕士 氏に心から感謝の意を表します.

最後に，日々御助言や御協力頂き，激励を賜りました井上研究室の皆様ならびに事務職員 軽部 瑞穂氏に心から感謝致します.

参考文献

[1] Aizu online judge: Programming challenge. `https://judge.u-aizu.ac.jp/onlinejudge/`. (Accessed on 01/19/2022).

[2] Atcoder. `https://atcoder.jp`. (Accessed on 01/19/2022).

[3] Hugging face – the ai community building the future. `https://huggingface.co/`. (Accessed on 01/22/2022).

[4] Ambient Software Evoluton Group. BigCloneBench. https://github.com/clonebench/BigCloneBench.

[5] Brenda S. Baker. Finding clones with dup: Analysis of an experiment. *IEEE Transactions on Software Engineering*, Vol. 33, No. 9, pp. 608–621, 2007.

[6] I.D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier. Clone detection using abstract syntax trees. In *Proceedings. International Conference on Software Maintenance (Cat. No. 98CB36272)*, pp. 368–377, 1998.

[7] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009.

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[9] Vahid Farrahi, Maisa Niemelä, Petra Tjurin, Maarit Kangas, Raija Korpelainen, and Timo Jämsä. Evaluating and enhancing the generalization performance of machine learning models for physical activity intensity prediction from raw acceleration data. *IEEE Journal of Biomedical and Health Informatics*, Vol. 24, No. 1, pp. 27–38, 2020.

[10] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. CodeBERT: A pre-trained model for programming and natural languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 1536–1547, Online, November 2020. Association for Computational Linguistics.

[11] 肥後芳樹, 神谷年洋, 楠本真二, 井上克郎. コードクローンを対象としたリファクタリング支援環境. 電子情報通信学会論文誌. D-I, Vol. 88, No. 2, pp. 186–195, feb 2005.

[12] 肥後芳樹, 楠本真二, 井上克郎. コードクローン検出とその関連技術. 電子情報通信学会論文誌. D, Vol. 91, No. 6, pp. 1465–1481, jun 2008.

[13] Giuseppe Jurman, Samantha Riccadonna, and Cesare Furlanello. A comparison of mcc and cen error measures in multi-class prediction. 2012.

[14] T. Kamiya, S. Kusumoto, and K. Inoue. CCFinder: a multilinguistic token-based code clone detection system for large scale source code. *IEEE Transactions on Software Engineering*, Vol. 28, No. 7, pp. 654–670, 2002.

[15] Liuqing Li, He Feng, Wenjie Zhuang, Na Meng, and Barbara Ryder. CCLearner: A deep learning-based clone detection approach. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 249–260, 2017.

[16] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[17] Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin B. Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, Ming Gong, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie Liu. CodeXGLUE: A machine learning benchmark dataset for code understanding and generation. *CoRR*, Vol. abs/2102.04664, , 2021.

[18] Lili Mou, Ge Li, Lu Zhang, Tao Wang, and Zhi Jin. Convolutional neural networks over tree structures for programming language processing. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, p. 1287–1293. AAAI Press, 2016.

[19] Ruchir Puri, David S Kung, Geert Janssen, Wei Zhang, Giacomo Domeniconi, Vladimir Zolotov, Julian Dolby, Jie Chen, Mihir Choudhury, Lindsey Decker, et al. CodeNet: A large-scale ai for code dataset for learning a diversity of coding tasks. *arXiv preprint arXiv:2105.12655*, 2021.

[20] Chanchal K. Roy, James R. Cordy, and Rainer Koschke. Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. *Science of Computer Programming*, Vol. 74, No. 7, pp. 470–495, 2009.

[21] Richard Socher, Cliff Chiung-Yu Lin, Andrew Y. Ng, and Christopher D. Manning. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, p. 129–136, Madison, WI, USA, 2011. Omnipress.

[22] Fang-Hsiang Su, Jonathan Bell, Gail Kaiser, and Simha Sethumadhavan. Identifying functionally similar code in complex codebases. In *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*, pp. 1–10, 2016.

[23] Jeffrey Svajlenko, Judith F. Islam, Iman Keivanloo, Chanchal K. Roy, and Mohammad Mamun Mia. Towards a big data curated benchmark of inter-project code clones. In *2014 IEEE International Conference on Software Maintenance and Evolution*, pp. 476–480, 2014.

[24] Kai Sheng Tai, Richard Socher, and Christopher D. Manning. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 1556–1566, Beijing, China, July 2015. Association for Computational Linguistics.

[25] Duyu Tang, Bing Qin, and Ting Liu. Document modeling with gated recurrent neural network for sentiment classification. In *Proceedings of the 2015 conference on empirical methods in natural language processing*, pp. 1422–1432, 2015.

[26] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

[27] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pp. 353–355, Brussels, Belgium, November 2018. Association for Computational Linguistics.

[28] Wenhan Wang, Ge Li, Bo Ma, Xin Xia, and Zhi Jin. Detecting code clones with graph neural network and flow-augmented abstract syntax tree. In *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 261–271, 2020.

[29] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[30] Yueming Wu, Deqing Zou, Shihan Dou, Siru Yang, Wei Yang, Feng Cheng, Hong Liang, and Hai Jin. SCDetector: Software functional clone detection based on semantic tokens analysis. In *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 821–833, 2020.

[31] Jian Zhang, Xu Wang, Hongyu Zhang, Hailong Sun, Kaixuan Wang, and Xudong Liu. A novel neural source code representation based on abstract syntax tree. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pp. 783–794, 2019.

[32] Gang Zhao and Jeff Huang. DeepSim: Deep learning code functional similarity. New York, NY, USA, 2018. Association for Computing Machinery.