Vol.2022-SE-210 No.8
2022/3/11

深層学習を用いたコードクローン検出器のベンチマーク間精度調査

福家 範浩[1]　藤原 裕士[1]　吉田 則裕[2]　崔 恩瀞[3]　井上 克郎[1]

概要：ソフトウェア保守において問題となる要因の 1 つとしてコードクローンがある．近年，多様なコードクローンを検出するために教師あり深層学習を取り入れた検出器が提案されている．既存の深層学習を用いたコードクローン検出器では，検出器の学習と評価のために 1 つのデータセットを分割して使用している．しかし，開発者が実開発現場でコードクローン検出器を用いる場合，検出器を適用する対象と学習データが類似しているとは限らない．そこで，本研究では学習と評価に複数のベンチマークを用いて，深層学習を用いたコードクローン検出器である CCLearner，ASTNN，CodeBERT に対して精度の調査を行った．その結果，全ての検出器で学習と異なるベンチマークに対し精度が下がり，CCLearner と ASTNN ではクローンのタイプごとで精度が異なったことを確認した．また，ASTNN と CodeBERT では，学習データに類似するベンチマークで評価を行った時，精度が高かったことを確認した．

Investigating the Performance of Deep Learning-based Code Clone Detectors Using Multiple Benchmarks

Norihiro FUKE[1]　Yuji FUJIWARA[1]　Norihiro YOSHIDA[2]　Eunjong CHOI[3]　Katsuro INOUE[1]

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，開発者は，バグを見つけたコード片に対してだけではなくそのコード片の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する必要がある．そこで，ソースコードの中からコードクローンを自動的に検出するため，多くのコードクローン検出器が提案されている．

代表的な従来のコードクローン検出器には，CCFinder [1] 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トークン解析を行うことで，トークン単位でのコードクローンを検出する．しかし，従来の検出器で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困難である．そこで最近では，深層学習を用いてコードクローンを検出する CCLearner [2]，ASTNN [3]，CodeBERT [4] 等が提案されている．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で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．実開発環境で開発者がコードクローン検出器を利用する時，学習済みのモデルを用いるが，検

[1] 大阪大学
Osaka University
[2] 名古屋大学
Nagoya University
[3] 京都工芸繊維大学
Kyoto Institute of Technology

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

出器を適用する対象と学習データが類似しているとは限らない．我々の先行研究 [5] では，CCLearner と ASTNN に対して，BigCloneBench [6] で学習を行い，Google Code Jam データセット [7] で評価を行ったところ，検出精度が下がったことが分かった．しかし，ベンチマークの特徴によってコードクローン検出器の精度がどのような影響を受けるか知るために，先行研究での実験結果は少ない．

そこで本稿では，深層学習を用いたコードクローン検出器 CCLearner，ASTNN に CodeBERT を加え，ベンチマークを 1 つ追加して 3 つ用意し，学習と評価に異なるベンチマークを用いた精度の調査を行った．

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ベンチマークに関して説明する．3 章では，調査方法とベンチマークの分析について説明を行う．4 章では，本調査で行った実験の結果について説明する．5 章では本調査の考察，6 章ではまとめを述べる．

2. コードクローン検出

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 [8].

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．

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．

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．

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．

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．

2.1 コードクローン検出器

これまで，ソースコード中からコードクローンを自動的に検出するために CCFinder 等，多く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が，T3 や T4 のコードクローンに対しては検出精度が低い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おり，代表的なコードクローン検出器として以下の CCLearner [2]，ASTNN [3]，CodeBERT [4] 等が挙げられる．

2.1.1 CCLearner

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．また，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，$l_Ax + l_Bx$ 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．

$$sim\_score_i = 1 - \frac{\sum_x |l_Ax - l_Bx|}{\sum_x l_Ax + l_Bx} \tag{1}$$

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．

2.1.2 AST-based Neural Network（ASTNN）

まず，コード片から抽象構文木（以降 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．そして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．

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

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 [9] 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．

2.1.3 CodeBERT

CodeBERT は BERT [10] や BERT を改良した RoBERTa [11] を基にしており，モデル構造は RoBERTa-base と全く同じである．CodeBERT の事前学習には，Masked Language Modeling（Maseked LM) と，BERT で取り入れられていた連続する文かを判定する Next Sentence Prediction（NSP）の代わりに 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．

事前学習した後，行いたいタスクのデータで fine tuning を行う．例えば 2 文の関係性を予測するには，BERT の事前学習の NSP のように 2 文をつなげて入力して判定するという方法がある．本実験では，事前学習済みの CodeBERT のモデルに，この方法と同様に 2 つのコード片をつなげて入力し，コードクローンであれば 1，非コードクローンであれば 0 が出力されるように fine tuning した．

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，一般的な深層学習を用いたシステムにおいて，汎化性能の問題が指摘されている．汎化性能の問題は，学習データとシステムの適用対象が異なることから生じ，データを取るセンサが異なることで精度が下がる問題 [12] 等が報告されている．何が深層学習を用いたシステムの精度を低下させるかは明確に分かっておらず，従って，深層学習を用いたシステムは学習データと異なるデータに対しても精度が維持できるか確かめておく必要がある．

汎化性能の問題点は，コードクローン検出器に対しても生じる可能性がある．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，トークンの類似度や保守するソースコードかどうか等，ソースコードのデータセットの様々な要因が検出精度に影響を与える可能性がある．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を分割して学習データと評価のデータを作成している．しかし，既存の検出器では，同一のベンチマークを用いることで，同じ競技プログラミングサイトから集めた Google Code Jam データセット [7] やオープンソースソフトウェア（以降 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 BigCloneBench [6] 等，似た特徴のデータ間で検証を行っており，学習データと異なる特徴を持ったデータに対する検出精度が明らかになっていない．しかし，開発者が深層学習を用いたコードクローン検出器を学習データと異なるデータに適用した時に検出精度が低いとソフトウェア保守を行うことが困難であるため，コードクローン検出器に汎化性能を調査する必要がある．

3. ベンチマーク間の精度調査

本研究では，2.2 節で説明した問題を解決するために，本研究では，2.1 節で説明した CCLearner や ASTNN，CodeBERT に対し，異なるベンチマークを用いて以下の 3 つの Research Question（RQ）を立て，**図 1** に示すように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ベンチマーク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，学習と評価で異なるベンチマークを用いた場合でも，同一ベンチマーク内での評価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に汎化性能の問題があるかを確かめることが出来る．同じベンチマーク内で評価した時の精度を維持できない検出器がある場合，深層学習を用いたコードクローン検出器にも汎化性能の問題があると言える．これは，実開発環境で開発者が学習済みの検出器を用いたとしても検出精度が悪いため，ソフトウェア保守作業の手助けにならない可能性を示す．RQ2 の調査により，学習と異なるベンチマークに対する検出精度とクローンのタイプの関係を調査し，検出器ごとに異なる関係となるか確かめることが出来る．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が検出器によって異なる場合，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，検出器ごとに学習法改善案を考えるための動機になる．また，クローンの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検出器の特性を理解し，それぞれにあった検出器を選択出来ると考えられる．RQ3 の調査により，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確かめることが出来る．競技プログラミングを基にしたベンチマーク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．

3.1 ベンチマークの準備と分析

本実験では，代表的なコードクローンベンチマークである BigCloneBench や Google Code Jam，CodeNet を用いた．

3.1.1 BigCloneBench

BigCloneBench（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ある [6]．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 [13] 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タイプ等のラベル付けがされたベンチマークである．BCB には複数のバージョンがあり，検出器ごとでもデータセットとして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，CCLearner と ASTNN では，深層学習モデルの学習に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．CodeBERT については，提案論文ではコードクローン検出が行われていなかったため，CodeXGLUE [14] で使われていた BCB を用いた．ベンチマーク間での評価には，ASTNN で用いられていた BCB のデータセットを用いた．ASTNN の論文と同様にコードクローンのタイプごとに精度を求めて，ベンチマーク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．

3.1.2 Google Code Jam

競技プログラミングサイト Google Code Jam [*2]に提出されたソースコードを利用して，同じ設問に提出された解答は T4 のコードクローンとして，コードクローン検出器を評価する研究が行われている [7, 15, 16]．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おり [7]，このデータセットを以降，GCJ と呼ぶ．GCJ の解答ソースコードは 12 問に対し 1669 個で，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．

本研究ではメソッド単位のコードクローン検出を行う．そのため，GCJ の解答データのソースコードの中の main メソッドを抽出し，main メソッド外で定義されている処理を main メソッド内にインライン展開して学習データとした．また，問題を解くための入出力処理でコードクローンの判定をさせないため，入出力処理を削除した．非コードクローンは，コードクローンより非常に多いため，非コードクローンをランダムに抽出して同じ数にした．ベンチマーク 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 個，非コードクローン数は約 275,000 個である．

3.1.3 CodeNet

CodeNet（以降 CN）は，競技プログラミングサイト AIZU Online Jadge[*3]と AtCoder[*4]から収集されたベンチマークである [17]．CN も競技プログラミングを基にしたベンチマークのため，3.1.2 節で説明した 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，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て 12 問取得した．CN 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．

3.1.4 ベンチマークの分析

各ベンチマークのコードクローンの Jaccard 係数と LOC の平均について調べて，文献 [18] の**図 6**，**図 7** にまとめた．ASTNN の BCB のタイプごとの Jaccard 係数の中央値は，T1 が 1.00，T2 が 0.929，ST3 が 0.705，MT3 が 0.418，WT3/T4 が 0.213，非コードクローンが 0.195 であった．LOC の平均の中央値は，コードクローンが 20 で非コードクローンが 13.5 であった．GCJ は，コードクローンの中央値は Jaccard 係数が 0.463，LOC の平均が 26.5，非コードクローンの中央値は Jaccard 係数が 0.388，LOC の平均が 30 であった．CN は，コードクローンの中央値は Jaccard 係数が 0.583，LOC の平均が 21，非コードクローンの中央値は Jaccard 係数が 0.463，LOC の平均が 21 であった．ここから，BCB ではコード片が短く T4 や非コードクローンの Jaccard 係数が低く，GCJ と CN ではコード片が長く Jaccard 係数が高い傾向にあることが分かった．

さらにベンチマークの頻出 1,000 語同士での Jaccard 係数を計測した．BCB と GCJ，BCB と CN は Jaccard 係数が 0.19，0.15 と，GCJ と CN の Jaccard 係数 0.25 よりも低く，GCJ と CN は BCB とよりも類似していて，競技プログラミングを基にしたベンチマーク同士が相対的に類似

[*2] https://codingcompetitions.withgoogle.com/codejam

[*3] https://judge.u-aizu.ac.jp/onlinejudge/

[*4] https://atcoder.jp

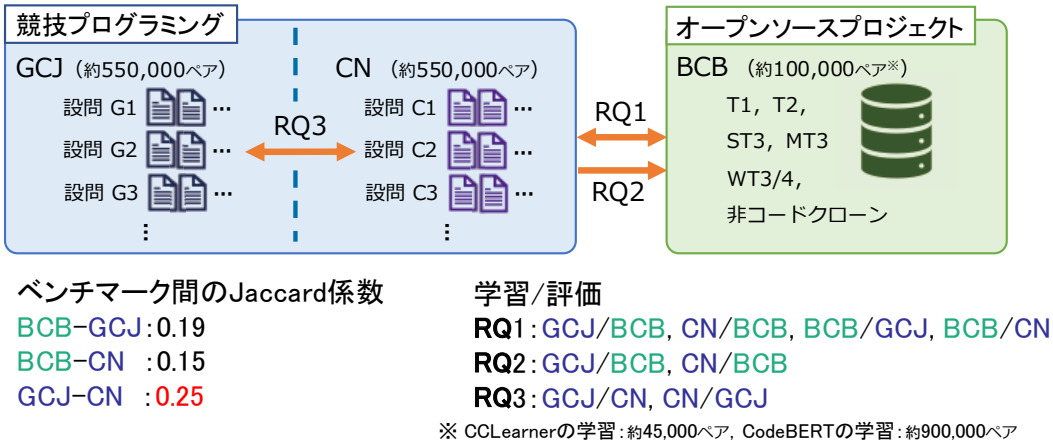

図 1 RQ に対する分析手順

していることが確認できた．

3.2 コードクローン検出器の学習と評価

CCLearner と ASTNN では，出力が閾値を超えていればコードクローンと判定しており，BCB で学習した時の閾値はそれぞれ 0.98 と 0.5 と設定した．しかし，同じ閾値を用いて BCB 以外のベンチマークで学習した時には精度が非常に悪かった．そこで，それぞれのベンチマークで正答率が高くなる値を探索して閾値を設定した．その結果，CCLearner では GCJ 学習時 0.17，CN 学習時 0.01，ASTNN では GCJ 学習時 1.1×10^{-4}，CN 学習時 2.8×10^{-4} とした．

RQ1 では，それぞれの検出器に対し，BCB で学習を行い GCJ または CN で評価を，また逆に GCJ または CN で学習を行い BCB で評価を行った．

RQ2 では，それぞれの検出器に対し，コードクローンのタイプ分類が行われている BCB で評価を行ったデータを用いて，タイプのごとの精度を調べた．また，タイプごとの Jaccard 係数の中央値と，MCC 値（3.3 節参照）との相関を調べた．

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 で評価を，また逆の CN で学習を行い GCJ で評価を行った．

3.3 評価指標

本調査では，コードクローン検出器の精度評価に f1，Matthews Correlation Coefficient（MCC）を用いた．**f1** は，precision と recall の調和平均である．**MCC** は，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おり [19]，コードクローンと非コードクローンが不均衡なデータセットに対しても評価するために用いた．以下の式 (4) が MCC の計算式である．

$$MCC = \frac{(tp \times tn) - (fp \times fn)}{\sqrt{(tp+tn)(tp+fn)(fp+tn)(fp+fn)}} \quad (4)$$

tp は true positive, tn は true negative, fp は false positive, fn は false negative である．MCC 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. 調査結果

4.1 RQ1 の結果

表 1 は，検出器ごとの f1 値と MCC 値の結果である．以降，それぞれの検出器毎に検出精度を説明する．

まず，**CCLearner** の結果について述べる．BCB で学習し BCB での評価には，CCLearner の提案論文 [2] での f1 値を使用した．BCB で学習した時，f1 値は，学習と評価に同じベンチマーク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た．GCJ または CN で学習した時，学習と同じベンチマークで評価すると，f1 値が約 0.55，MCC 値が約 0.3 から 0.35 だったが，BCB で評価すると f1 値が約 0.1，MCC 値が約 0.05 から 0.1 と大幅に低下した．

ASTNN の結果について述べる．BCB で学習した時，学習と同じベンチマークで評価すると f1 値が 0.939，MCC 値が 0.891 だったのに対し，GCJ または CN で評価すると f1 値が約 0.66，MCC 値が約 0.01 から 0.03 と大幅に下がった．GCJ または CN で学習した時，学習と同じベンチマークで評価すると f1 値が約 0.5，MCC 値が約 0.45 であったが，BCB で評価すると GCJ で学習した検出器の f1 値が 0.372，MCC 値が 0.001，CN で学習した検出器の f1 値が 0.09，MCC 値が −0.15 と大幅に低下した．

最後に，**CodeBERT** の結果について述べる．BCB で学習した時，学習と同じベンチマークで評価すると f1 値が 0.896，MCC 値が 0.881 だったのに対し，GCJ または CN で評価すると f1 値が約 0.65，MCC 値が約 0.1 と大幅に下がった．GCJ または CN で学習した時，学習と同じベンチ

表 1 RQ1 の結果：学習と同じ，または異なるベンチマークでの評価

		CCLearner		ASTNN		CodeBERT	
	学習 \ 評価	同じ	異なる	同じ	異なる	同じ	異なる
f1	BCB	0.93	GCJ: 0.109 CN: 0.357	0.939	GCJ: 0.667 CN: 0.663	0.896	GCJ: 0.630 CN: 0.651
	GCJ	0.566	BCB: 0.116	0.522	BCB: 0.372	0.998	BCB: 0.608
	CN	0.572	BCB: 8.74×10^{-2}	0.481	BCB: 9.00×10^{-2}	0.999	BCB: 0.651
MCC	BCB	–	GCJ: 0.159 CN: 0.296	0.891	GCJ: 2.75×10^{-2} CN: 1.29×10^{-2}	0.881	GCJ: 9.88×10^{-2} CN: 6.12×10^{-2}
	GCJ	0.340	BCB: 8.47×10^{-2}	0.463	BCB: 1.01×10^{-3}	0.997	BCB: 4.29×10^{-3}
	CN	0.318	BCB: 5.00×10^{-2}	0.434	BCB: -0.150	0.998	BCB: 4.32×10^{-3}

マークで評価すると f1 値と MCC 値がともに約 1 だったのに対し，BCB で評価すると f1 値が GCJ 値で学習した検出器で 0.6，CN で学習した検出器で 0.65，MCC 値がいずれも約 0.004 と大幅に下がった．

RQ1 の答え

いずれのコードクローン検出器も，学習と評価に異なるベンチマークを用いると検出精度が同じベンチマークで評価した時に比べて下がる．

4.2 RQ2 の結果

表 2 は，クローンタイプごとの MCC の結果である．以降，それぞれの検出器毎の MCC 値を説明する．

まず，**CCLearner** では，MCC 値が GCJ と CN のいずれのデータセットで学習した時も，異なるベンチマークである BCB で評価するとコードクローンのタイプが T1 から T4 へ類似度が低くなるに従って下がった．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 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．

ASTNN では，MCC 値が GCJ，CN のいずれのデータセットで学習した時も，異なるベンチマークである 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 値の相関係数は，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．

最後に，**CodeBERT** では，MCC 値が GCJ，CN のいずれのデータセットで学習した時も，異なるベンチマークである 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 と精度が低かった．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 値の相関係数は，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．

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では関係が強く，CodeBERT ではコードクローンのタイプと検出精度では関係が弱い．

4.3 RQ3 の結果

表 3 は，検出器ごとの MCC 値の結果である．以降，それぞれの検出器毎の MCC 値を説明する．

まず，**CCLearner** は，GCJ または CN で学習を行い，学習と同じベンチマークで評価した場合に比べて，競技プログラミングを基にした異なるベンチマークでの評価では MCC 値が下がっていた．OSS を基にした BCB で学習した場合と比べると，BCB，CN のいずれのデータセットで学習した場合も GCJ で評価すると MCC 値が約 0.16 で，BCB，GCJ のいずれのデータセットで学習した場合も CN で評価すると MCC 値が約 0.29 と同等であった．

ASTNN は，GCJ または CN で学習を行い，学習と同じベンチマークで評価した場合に比べて，競技プログラミングを基にした異なるベンチマークでの評価では MCC 値が下がっていた．しかし，OSS を基にした BCB で学習した場合と比べると，CN で学習し GCJ での評価で MCC 値が約 0.12 と BCB で学習時の約 0.03 から上がっており，GCJ で学習し CN での評価で MCC 値が約 0.25 と BCB で学習時の約 0.01 から上がっている．

最後に，**CodeBERT** は，GCJ または CN で学習を行い，学習と同じベンチマークで評価した場合に比べて，競技プログラミングを基にした異なるベンチマークでの評価では MCC 値が下がっていた．しかし，OSS を基にした BCB で学習した場合と比べると，CN で学習し GCJ での評価で MCC 値が約 0.25 と BCB で学習時の約 0.10 から上がっており，GCJ で学習し CN での評価で MCC 値が約 0.42 と BCB で学習時の約 0.06 から上がっている．

表 2 RQ2 の結果：学習と異なるベンチマークで評価した MCC 値（クローンタイプごと）

	CCLearner		ASTNN		CodeBERT	
タイプ \ 学習データ	GCJ	CN	GCJ	CN	GCJ	CN
T1	0.961	0.967	0.713	0.851	-2.66×10^{-4}	-1.14×10^{-3}
T2	0.853	0.911	0.517	0.687	1.56×10^{-3}	1.66×10^{-2}
ST3	0.626	0.699	0.518	0.456	3.90×10^{-3}	7.78×10^{-5}
MT3	0.462	0.281	0.174	1.29×10^{-2}	-1.44×10^{-3}	6.12×10^{-4}
WT3/T4	7.45×10^{-2}	4.08×10^{-2}	-5.96×10^{-3}	-0.159	4.38×10^{-3}	4.38×10^{-3}

表 3 RQ3 の結果：競技プログラミングを基にしたベンチマーク間で学習，評価した MCC

	CCLearner		ASTNN		CodeBERT	
学習 \ 評価	GCJ	CN	GCJ	CN	GCJ	CN
BCB	0.159	0.296	2.75×10^{-2}	1.29×10^{-2}	9.88×10^{-2}	6.12×10^{-2}
異なる競プロ	0.166	0.293	0.121	0.247	0.252	0.415

RQ3 の答え

いずれのコードクローン検出器も同じベンチマーク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 では類似するベンチマーク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．CCLearner に関しては，類似するベンチマーク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．

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検出精度が CCLearner と ASTNN は変わり，CodeBERT は変わらないこと
- 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，ASTNN と CodeBERT は検出精度が上がり，CCLearner は検出精度が変わらないこと

これらの結果から CCLearner，ASTNN，CodeBERT を開発者が実開発現場で使用する場合の有効な知見が次のように得られた．CCLearner，ASTNN，CodeBERT は汎化性能が低いため，学習データと適応対象が異なる場合，検出されたコードクローンが正く検出されたか確認する必要がある．CCLearner を学習データと類似しているデータに適用する場合，コードクローンの検出精度が低いため，検出されたコードクローンが正く検出されたか確認する必要がある．また，T3 及び T4 など類似度が低いコードクローンを含むデータセットで学習する場合でも T1 及び T2 は正確に検出できる反面，T3 及び T4 の検出精度が低いため，コードクローンが正確に検出されたかを確認する必要がある．ASTNN を T3 及び T4 など類似度が低いコードクローンを含むデータセットで学習する場合，T1 及び T2 は正確に検出できる反面，類似度が低い T3 及び T4 コードクローンに対しては正確に検出されたか確認する必要があるが．しかし，学習データと類似するデータセットに対して検出精度が比較的に高いため，適用対象と類似するデータで学習することで正確にコードクローンが検出することが期待できる．最後に，CodeBERT は学習データと類似するデータセットに適用する場合，検出精度が比較的に高いため，適用対象と類似するデータで学習することで正確にコードクローンが検出することが期待できる．また，コードクローンのタイプに影響受けにくい．

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 以外のデータセットで学習し BCB で評価する時には，どの検出器も同じデータで評価出来るように 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，f1 値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 値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

本調査では，深層学習を用いたコードクローン検出器に対して学習と異なるベンチマークを用いて評価を行った，

その結果，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と異なるベンチマークに対する検出精度の関係は CCLearner と ASTNN では強く，CodeBERT では弱いことが分かった．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，ASTNN と CodeBERT は検出精度が上がることが分かった．

謝辞 本研究は JSPS 科研費 18H04094, JP19K20240, JP20K11745，21H03416，ならびに JST さきがけ JPMJPR21PA の助成を受けた．

参考文献

[1] Kamiya, T., Kusumoto, S. and Inoue, K.: CCFinder: a multilinguistic token-based code clone detection system for large scale source code, *IEEE Trans. Softw. Eng.*, Vol. 28, No. 7, pp. 654–670 (2002).
[2] Li, L., Feng, H., Zhuang, W., Meng, N. and Ryder, B.: CCLearner: A Deep Learning-Based Clone Detection Approach, *Proc. of ICSME*, pp. 249–260 (2017).
[3] Zhang, J., Wang, X., Zhang, H., Sun, H., Wang, K. and Liu, X.: A Novel Neural Source Code Representation Based on Abstract Syntax Tree, *Proc. of ICSE*, pp. 783–794 (2019).
[4] Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., Jiang, D. and Zhou, M.: CodeBERT: A Pre-Trained Model for Programming and Natural Languages, *Proc. of EMNLP* (2020).
[5] 福家範浩，藤原裕士，吉田則裕，崔 恩瀞，井上克郎：深層学習を用いたコードクローン検出器の汎化性能に関する調査，情報処理学会研究報告，Vol. 2021-SE-207, No. 12, pp. 1–7 (2021).
[6] Svajlenko, J., Islam, J. F., Keivanloo, I., Roy, C. K. and Mia, M. M.: Towards a Big Data Curated Benchmark of Inter-project Code Clones, *Proc. of ICSME*, pp. 476–480 (2014).
[7] Zhao, G. and Huang, J.: DeepSim: Deep Learning Code Functional Similarity, *Proc. of ESEC/FSE*, pp. 141–151 (2018).
[8] Roy, C. K., Cordy, J. R. and Koschke, R.: 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).
[9] Tang, D., Qin, B. and Liu, T.: Document modeling with gated recurrent neural network for sentiment classification, *Proc. of EMNLP*, pp. 1422–1432 (2015).
[10] Devlin, J., Chang, M.-W., Lee, K. and Toutanova, K.: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, *Proc. of NAACL-HLT 2019*, Minneapolis, Minnesota, Association for Computational Linguistics (2019).
[11] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L. and Stoyanov, V.: RoBERTa: A robustly optimized bert pretraining approach, *arXiv preprint arXiv:1907.11692* (2019).
[12] Farrahi, V., Niemelä, M., Tjurin, P., Kangas, M., Korpelainen, R. and Jämsä, T.: Evaluating and Enhancing the Generalization Performance of Machine Learning Models for Physical Activity Intensity Prediction From Raw Acceleration Data, *IEEE J Biomed Health Inform*, Vol. 24, No. 1, pp. 27–38 (2020).
[13] Ambient Software Evoluton Group: BigCloneBench, https://github.com/clonebench/BigCloneBench.
[14] Lu, S., Guo, D., Ren, S., Huang, J., Svyatkovskiy, A., Blanco, A., Clement, C. B., Drain, D., Jiang, D., Tang, D., Li, G., Zhou, L., Shou, L., Zhou, L., Tufano, M., Gong, M., Zhou, M., Duan, N., Sundaresan, N., Deng, S. K., Fu, S. and Liu, S.: CodeXGLUE: A Machine Learning Benchmark Dataset for Code Understanding and Generation, *CoRR*, Vol. abs/2102.04664 (2021).
[15] Su, F.-H., Bell, J., Kaiser, G. and Sethumadhavan, S.: Identifying functionally similar code in complex codebases, *Proc. of ICPC*, pp. 1–10 (2016).
[16] Wu, Y., Zou, D., Dou, S., Yang, S., Yang, W., Cheng, F., Liang, H. and Jin, H.: SCDetector: Software Functional Clone Detection Based on Semantic Tokens Analysis, *Proc. of ASE*, pp. 821–833 (2020).
[17] Puri, R., Kung, D. S., Janssen, G., Zhang, W., Domeniconi, G., Zolotov, V., Dolby, J., Chen, J., Choudhury, M., Decker, L. et al.: CodeNet: A Large-Scale AI for Code Dataset for Learning a Diversity of Coding Tasks, *arXiv preprint arXiv:2105.12655* (2021).
[18] 福家範浩：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，大阪大学大学院情報科学研究科修士学位論文 `https://sel.ist.osaka-u.ac.jp/lab-db/Mthesis/archive/157/157.pdf` (2022).
[19] Jurman, G., Riccadonna, S. and Furlanello, C.: A comparison of MCC and CEN error measures in multi-class prediction, *PLoS ONE* (2012).