Software Engineering Laboratory

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

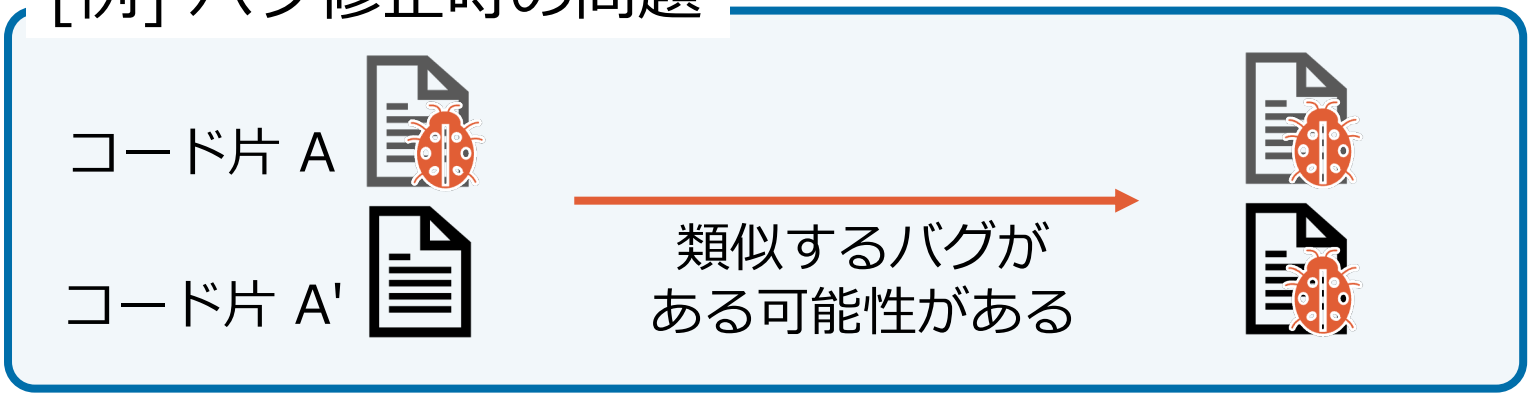
◯福家範浩[1] 藤原裕士[1] 吉田則裕[2] 崔恩瀞[3] 井上克郎[1]

[1]大阪大学 [2]名古屋大学 [3]京都工芸繊維大学

コードクローン（以降CC）

- 同一，または類似したコード片を持つコード片
- ソフトウェア保守を困難にする要因の1つ

[例] バグ修正時の問題



CCとタイプ分類[6]（1/2）

Type-1（以降T1）

空白やタブ，改行，コメント以外は一致するコード片対

Type-2（以降T2）

T1に加えて
変数名や関数名，変数の型が異なるコード片対

元のコード

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

T1：括弧の位置

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

T2：型・変数名

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

CCとタイプ分類[6]（2/2）

Type-3

T2に加えて
文の挿入や削除，変更などの違いがあるコード片対

Type-4（以降T4）

Type-3に加えて
同一の処理を行うが，構文上の違いがあるコード片対

ST3（Strongly Type-3）：**70**%以上，**MT3**（Moderately Type-3）：**50**～**70**%
WT3（Weakly Type-3）：**50**%未満

元のコード

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

ST3：文の変更

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

T4：構文的な異なり

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

ソフトウェア開発者が，数多くのコード片対から，CCを探すための検出器が提案されている

深層学習を用いたCC検出器

- 従来の字句解析などだけでは，MT3やWT3/T4のCCの検出精度の向上は困難である

- 近年，WT3/T4のCCを検出するため
  CC検出器に深層学習が取り入れられている

  [例] CCLearner [2]
  トークンカテゴリごとの類似度を特徴として入力

深層学習モデルにおける汎化性能の問題

学習データと適用対象データ

学習データの性質と適用対象のデータの性質が
異なることで，想定していた精度より悪くなる問題

汎化性能の必要性

1. センサAで取得したデータを
   学習データとして収集

2. 同じデータを取得することができる
   センサ B/C も用いられる可能性

例：メーカーやセンサの
バージョンなどの違い

取得している
データは同じ

しかし，センサ B/C
によるデータは未学習

精度を維持できるか
→ 汎化性能の問題

深層学習を用いたCC検出器における汎化性能の問題

学習データと適用対象データ

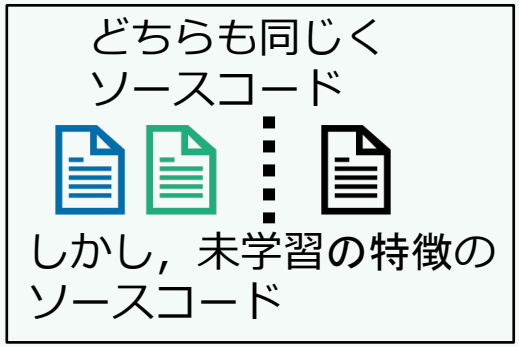
学習データの性質と適用対象のデータの性質が
異なることで，想定していた精度より悪くなる問題

汎化性能の必要性

1. 既存のある機能を持った
   コードクローンを学習用に収集

2. 異なる特徴を持ったコード片に
   対して検出器を使う可能性

例：プログラマの熟練度や
コードの用途などの違い



精度を維持できるか
→ 汎化性能の問題

我々の先行研究[5]

深層学習を用いたCC検出器に対し，学習と異なるベンチマークで評価

- 2つの深層学習を用いたCC検出器で調査
- 2つのCCベンチマークを準備し、片方で学習し他方で評価

学習と同じベンチマークで評価時より精度が低いことを確認

先行研究の問題

しかし，ベンチマークの特徴によって，CC検出器の精度がどのような影響を受けるか判断するには，実験結果が少ない

本研究の目的と概要

- 深層学習を用いたCC検出器において
  汎化性能の問題があるかさらなる知見を得る
- 深層学習を用いたCC検出器の検出精度に
  データの性質が影響するか知見を得る

概要

1. 先行研究で使用した2つのベンチマークに1つ
   似た収集基のベンチマークを加えて調査
2. 3つの深層学習を用いたCC検出器について調査
3. 3つのResearch Questionを設定

Research Question（RQ）（1/2）

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?

RQの答えが分かることによるメリット

深層学習を用いたCC検出器を使う時，汎化性能の問題が生じるのかどうか知見を得ることが出来る

RQ2：学習と評価に異なるベンチマークを用いた場合，CCのタイプと検出精度は関係しているか?

RQの答えが分かることによるメリット

CCのタイプと汎化性能に関係がある場合，CC検出器の学習法改善のための情報になると考えられる

Research Question（RQ）（2/2）

RQ3：競プロ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?

RQの答えが分かることによるメリット

類似するデータに対して精度が維持出来る場合，類似する学習データを作成する動機になる

※競プロ：競技プログラミング

概要:汎化性能の分析

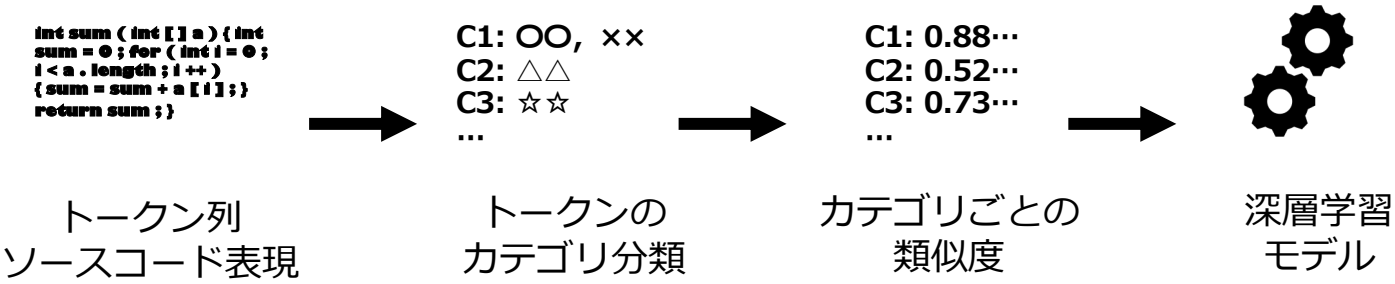
深層学習を用いた複数の検出器について，複数のベンチマークを用いて性能を評価

- 深層学習を用いたCC検出器
  - CCLearner[2]　　トークンカテゴリごとの類似度
  - ASTNN[3]　　AST-based Neural Network
  - CodeBERT[4]　　BERT系
- ベンチマーク
  - BigCloneBench[6]（BCB）　OSS等
  - Google Code Jam[7]（GCJ）競技プログラミング
  - CodeNet[17]（CN）　　競技プログラミング

CCLearner

ソースコード中のトークンのカテゴリごとの類似度を特徴量とした深層学習モデル

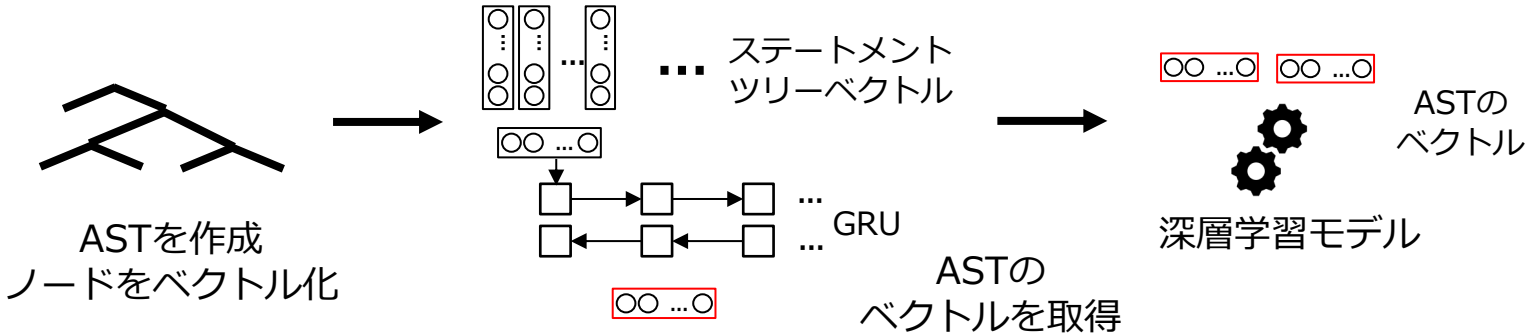
- 特徴量を出す方法
  1. ソースコードをトークンごとに分割
  2. トークンを8つのカテゴリに分類
  3. カテゴリごとの類似度を特徴量とする
- 従来の検出器より検出精度が向上し，広い範囲のCCを検出可能



ASTNN

抽象構文木（以降AST）をベクトル化して
特徴量とした深層学習モデル

- ASTのベクトル化の手順
  1. トークンをベクトル化
  2. 1でのベクトルを用いて ステートメントをベクトル化
  3. 2でのベクトルを用いて ASTをベクトル化
- 従来の検出器より検出精度が向上し，さらに広い範囲のCCを検出可能



CodeBERT

近年，自然言語処理分野で多く使われている
BERTをソースコードに適用した深層学習モデル

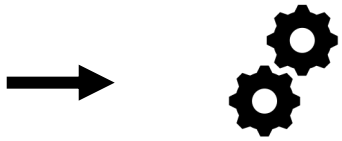
- CodeBERTは，教師なし事前学習による汎用言語モデルとして公開されている
- コード片対を2文とみなして連続で入力し，CCか非CCかを出力するタスクでfine-tuning（再学習）
- 従来の検出器より検出精度が向上し，さらに広い範囲のCCを検出可能

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

コード片1

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

コード片2



汎用言語モデルに対して
CC検出タスクでさらに学習

本調査に用いたベンチマーク

- オープンソースソフトウェアのソースコード
  - BigCloneBench
- 競技プログラミングの解答ソースコード
  - Google Code Jam，CodeNet の2種
    競技プログラミングの解答ソースコード同士は
    類似していると考えられる

ベンチマーク（BCB）

オープンソースソフトウェアのソースコード

- BCB
  - Svajlenko らによって作成された Java ソースコードの大規模ベンチマーク [6]
  - 大規模データである IJaDataset 2.0の中から特定の機能のソースコードをマイニング，人手でタイプ等のラベル付け
  - BCBには複数のバージョンがあり，CC検出器によってもデータの抽出数に違い

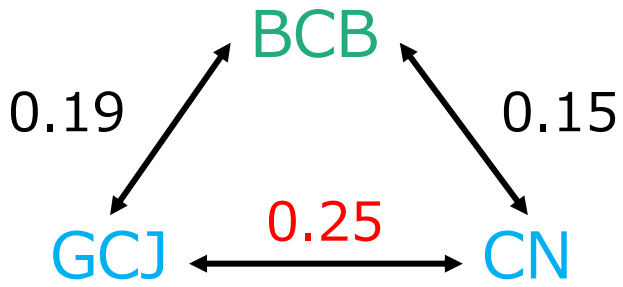
ベンチマーク（GCJ・CN）

競技プログラミングの解答ソースコード
同じ設問に対する解答は，同一の処理を行うが
構文的に異なるT4のCCとして構成

- GCJ
  - 既存研究で Google Code Jamから収集されたもの[7]
  - CC：約275,000，非CC：約275,000
- CN
  - AtCoderやAIZU Online Judge（以降 AOJ）から収集[17]
  - 本調査ではAOJのコードをGCJに近い設問数，CC数取得
  - CC：約276,000，非CC：約276,000

ベンチマーク間での類似度調査

- ベンチマークそれぞれの頻出1,000語同士でJaccard係数を調査
  - コード片ごとに出てくる単語をリストアップ
  - ベンチマーク内の何個のコード片にあるかカウント
  - 上位1,000語を取得し，Jaccard係数を調査
- 結果



競技プログラミングを基にした
ベンチマーク間が一番類似している

評価指標

- f1

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

f1: 1が最高，0が最悪

- マシューズ相関係数（MCC）

$$\mathrm{MCC} = \frac{tp \cdot tn - fp \cdot fn}{(tp + fp)(tp + fn)(tn + fp)(tn + fn)}$$

tp: true-positive，*tn*: true-negative，　MCC: 1が最高，-1が真逆，0が最悪
fp: false-positive，*fn*: false-negative

正負の数	f1	MCC
不均衡	精度が悪くても 数値が高い可能性	精度が悪いと 数値は0に近い値
均衡	大部分を 正と予測：約0.6 負と予測：約0	大部分を 正と予測：約0 負と予測：約0

3つのRQ

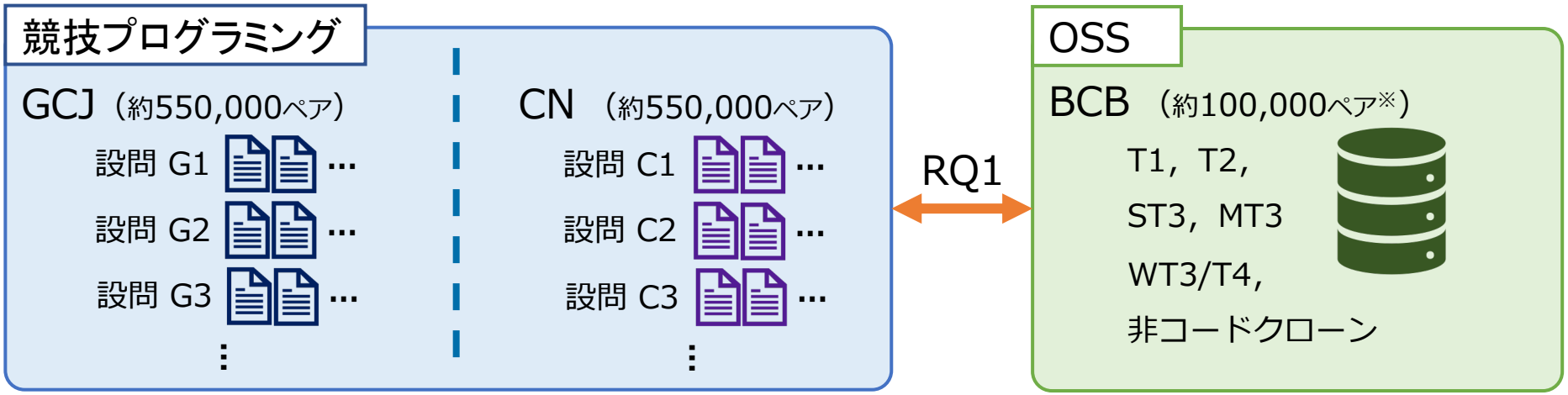
- RQ1

  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，
  CC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：異なるベンチマークに対して



学習/評価

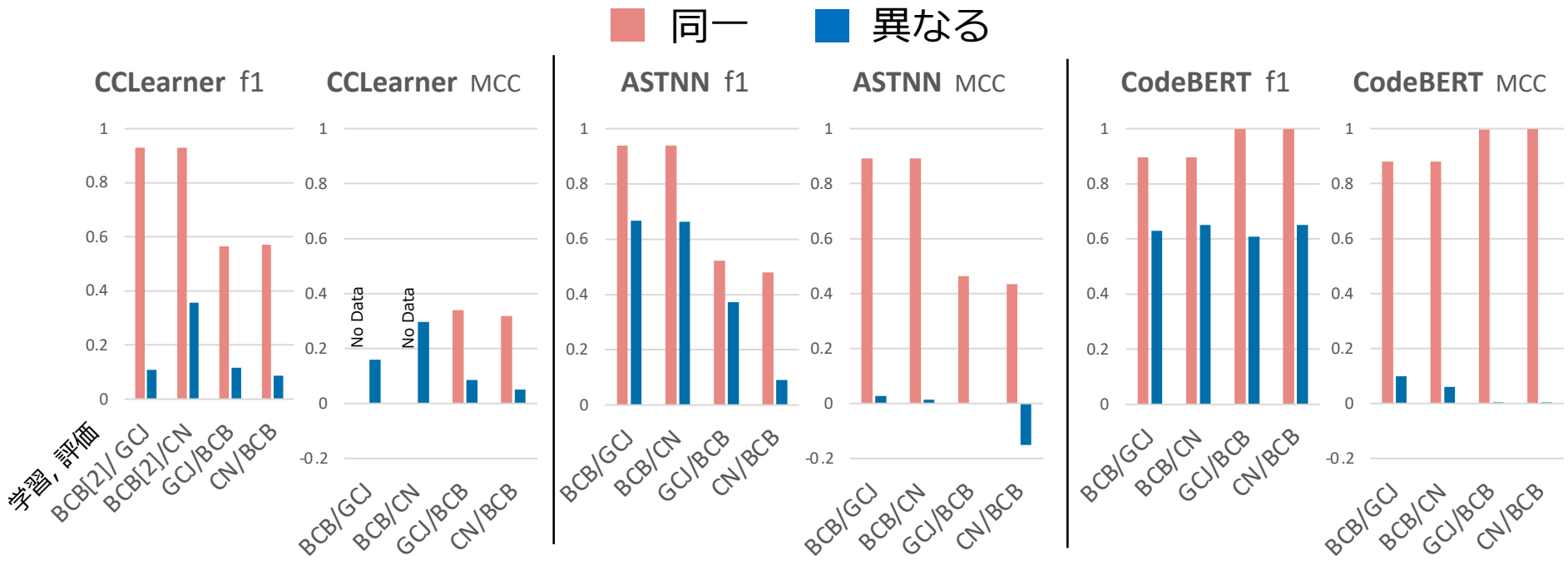
RQ1：GCJ/BCB, CN/BCB, BCB/GCJ, BCB/CN

1. OSS等を基にしたベンチマークと，競技プログラミング（以降，競プロ）を基にしたベンチマークで学習と評価
2. 同じベンチマークを用いた時の精度と別のベンチマークを用いた時の精度で比較

※ CCLearnerの学習：約45,000対，CodeBERTの学習：約900,000対

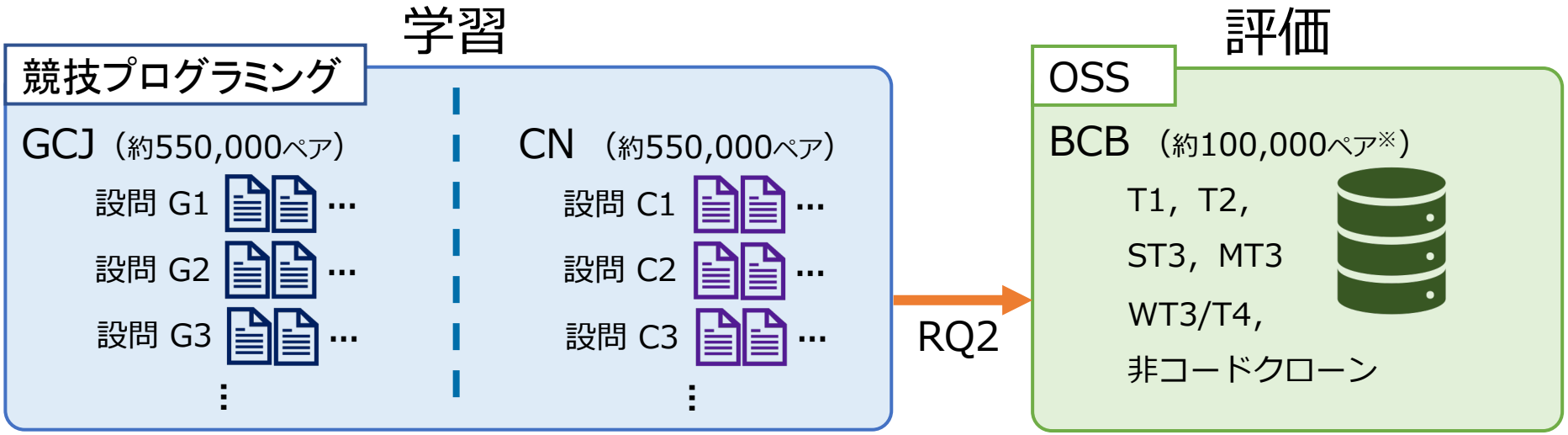
RQ1：異なるベンチマークに対して

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.



CCLearner，**ASTNN**，**CodeBERT**は同じベンチマークを用いて評価した場合に比べて全て精度が下がっている
→ 汎化性能が低い

RQ2：CCのタイプごとの検出精度

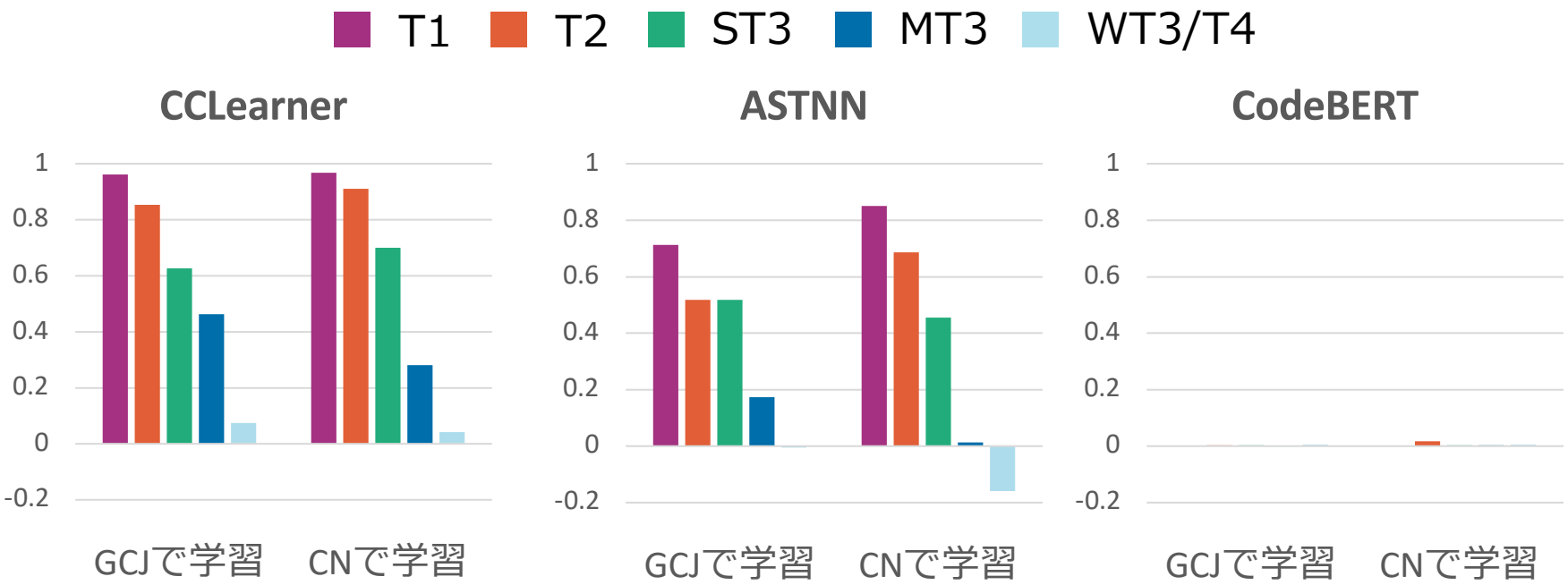


学習/評価
RQ2：GCJ/BCB, CN/BCB

1. 競プロを基にしたデータセットで学習を行い，CCのタイプがラベル付けされたOSSを基にしたベンチマークで評価
2. CCのタイプごとに精度を調査

※ CCLearnerの学習：約45,000対，CodeBERTの学習：約900,000対

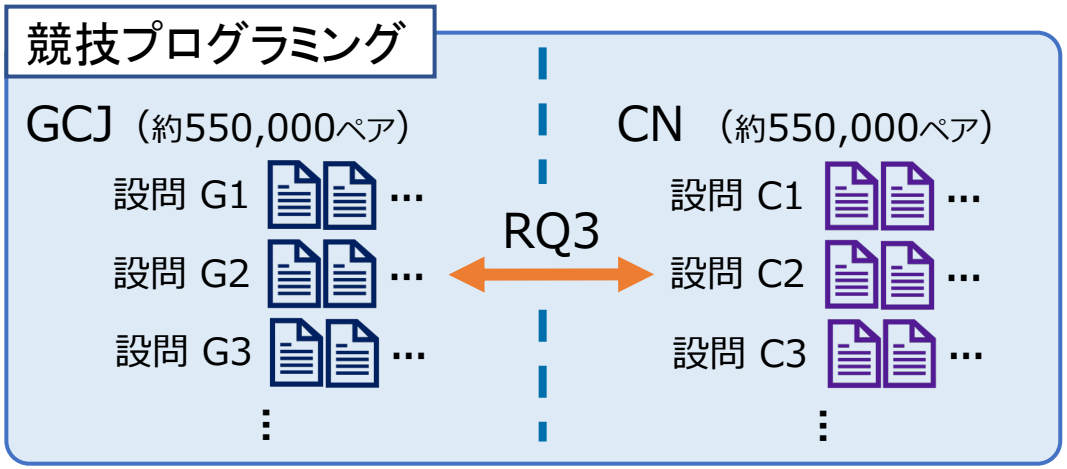
RQ2：CCのタイプごとの検出精度（MCC）

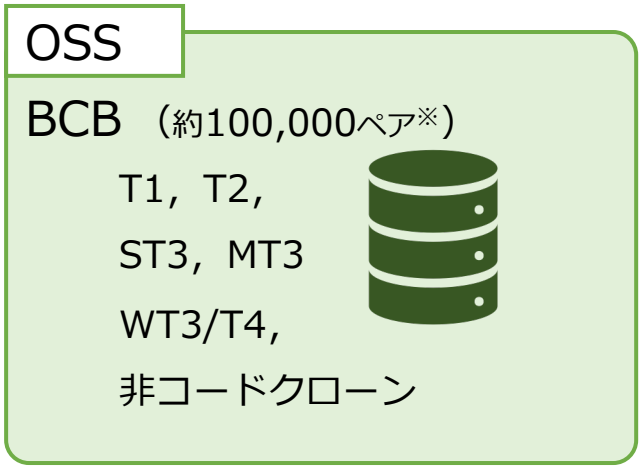


全て0.1未満

CCLearner，**ASTNN**は，類似度が低いCCのタイプで精度が低下 → CCのタイプと精度の関係が強い
CodeBERTは，全てのタイプで精度はほとんど等しく低い
→ CCのタイプと精度の関係が弱い

RQ3：類似するベンチマーク間での検出精度





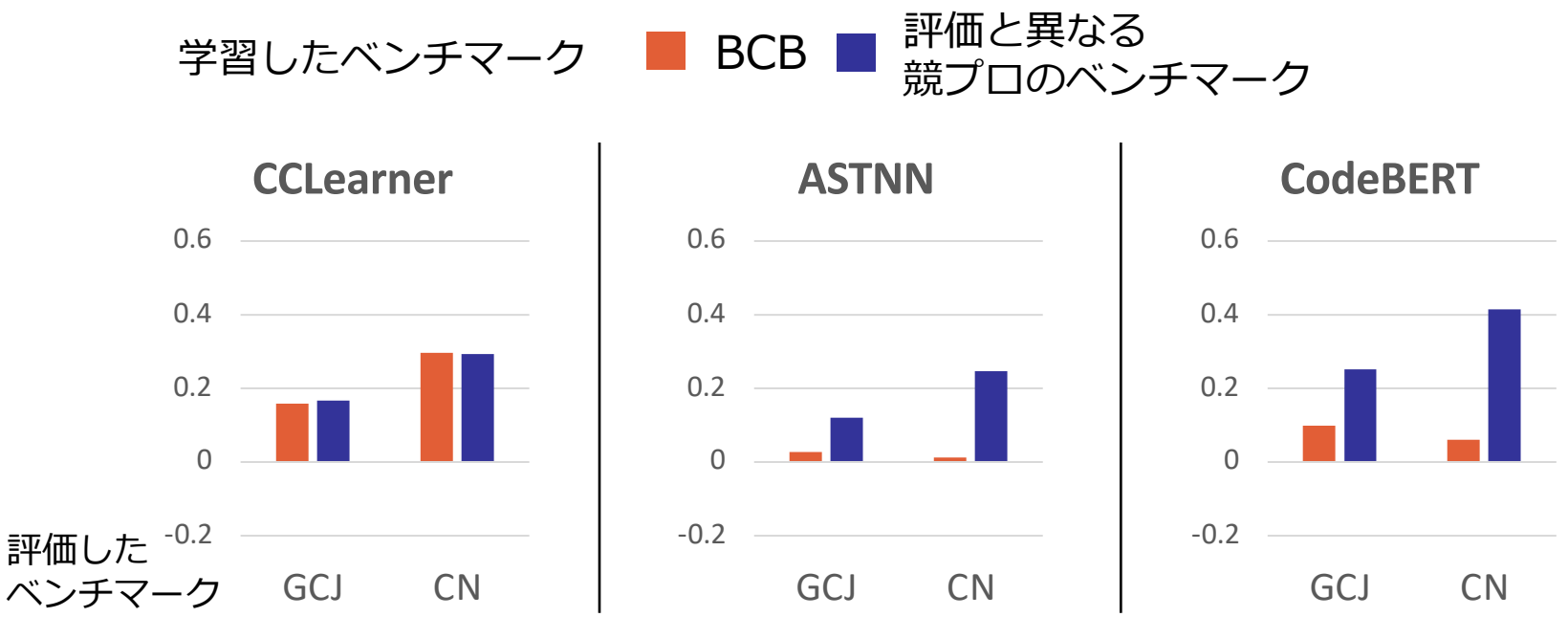
学習/評価
RQ3：GCJ/CN, CN/GCJ

1. 競プロを基にしたベンチマーク間で学習と評価
2. RQ1で調べた精度と比較

ベンチマーク間の
Jaccard係数
BCB-GCJ：0.19
BCB-CN ：0.15
GCJ-CN ：0.25

※ CCLearnerの学習：約45,000対，CodeBERTの学習：約900,000対

RQ3：類似するベンチマーク間での検出精度（MCC）



いずれの検出器も同じベンチマークでの評価に比べると
精度が下がっている
ASTNNと**CodeBERT**は，RQ1に比べると検出精度が上がった
CCLearnerは，RQ1と検出精度が変わらなかった

RQまとめ（1/2）

- 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?
  - 学習と異なるベンチマークで評価を行うと，CCLearner，ASTNN，CodeBERTいずれも精度が低い

- RQ2：学習と評価に異なるベンチマークを用いた場合，CCのタイプと検出精度は関係しているか?
  - CCLearner と ASTNNはCCのタイプによって精度が変わり，CodeBERT では変わらない

RQまとめ（2/2）

- 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?
  - 学習と類似するベンチマークに対して，ASTNN と CodeBERT は精度が上がり，特にCodeBERTは精度が高かった．CCLearner は精度が変わらなかった

考察（1/2）

3つのRQから考えられること

1. 学習データと適用対象が異なる場合
   - CCLearner，ASTNN，CodeBERT
     検出されたCCが正確か確認する必要

2. コードクローンのタイプに関して
   - CCLearner，ASTNN
     T3やT4のCCで学習しても，T1やT2は正確に検出出来るが，
     T3やT4の検出精度が低い
     → 検出されたCCが正確か確認する必要

考察（2/2）

3つのRQから考えられること

3．学習と適用対象の類似に関して

- CodeBERT，ASTNN
  適用対象に類似するデータを作成し，学習することで
  検出精度を高められる
  → 適用対象に類似した学習データを
  　作成する方が良い
- CCLearner
  適用対象に類似するデータを作成しても検出精度を高める
  ことはできない

まとめ

深層学習を用いたCC検出器の汎化性能を分析

1. 深層学習を用いたCC検出器にも，汎化性能の問題があること
2. 学習と異なるベンチマークに対する精度は，CCLearner，ASTNNではCCのタイプと関係が強く，CodeBERTではCCのタイプと関係が弱い
3. ASTNN，CodeBERTは，検出対象に類似するデータで学習することで検出精度が上がる

今後の課題

OSSを基にした新たなベンチマークを作成し，
OSSを基にしたベンチマーク同士で調査