THE IEICE TRANSACTIONS ON INFORMATION AND SYSTEMS (JAPANESE EDITION)

電子情報通信学会 論文誌 D 情報・システム

VOL. J107-D NO. 2
FEBRUARY 2024

情報・システムソサイエティ

一般社団法人 電子情報通信学会
THE INFORMATION AND SYSTEMS SOCIETY
THE INSTITUTE OF ELECTRONICS, INFORMATION AND COMMUNICATION ENGINEERS

論　文

SemanticCloneBench を用いた深層学習に基づくコードクローン検出手法の評価

鶴　智秋[†a)]　松下　誠[†b)]　肥後　芳樹[†c)]

An Evaluation of Deep-Learning-Based Code Clone Detection Methods Using SemanticCloneBench

Tomoaki TSURU[†a)], Makoto MATSUSHITA[†b)], and Yoshiki HIGO[†c)]

あらまし　構文上異なるにもかかわらず同様の振る舞いが記述されているソースコード片を意味的クローンと呼ぶ．近年では，意味的クローンの検出を目的として，深層学習を用いたクローン検出手法が多数提案されている．しかし，クローン検出の分野において有名かつ巨大なベンチマークである BigCloneBench には，意味的クローン検出性能評価において多くの問題が存在する．本研究では，SemanticCloneBench と呼ばれる，意味的クローンの性能評価に特化したベンチマークを用いて，深層学習に基づくクローン検出手法である ASTNN，CodeBERT，InferCode の 3 手法を対象とした検出性能の評価を行う．適用実験の結果より，ASTNN における F 値は 0.921 であり，評価対象となる検出手法の中で最も高い検出性能を示した．また，F 値が高くなるようなしきい値の範囲を調査した結果，CodeBERT と InferCode で高い F 値を示したしきい値の範囲は，ASTNN において F 値が高くなるようなしきい値の範囲よりも狭いと判明した．

キーワード　コードクローン，意味的コードクローン，深層学習，SemanticCloneBench

1. まえがき

コードクローン（以後，クローン）とは，他と一致または類似する箇所があるソースコード片である．一般的に，クローンは，構文的な類似度に基づいて，構文的クローンと意味的クローンの 2 種類に分類される．構文的クローンは，トークン列や命令文など，ソースコードの構造が類似するクローンを指す．一方，意味的クローンは，同じ振る舞いであるにもかかわらず，構文的に異なるクローンを指す．意味的クローンの検出は，ソースコードをクエリとしたソースコード検索 [1] や，API の使用例に基づく再利用のためのソースコード推薦 [2] などへの応用が期待されている．

これまでに，構文的クローンに着目したクローン検出手法が多数提案されている [3], [4]．しかし，構文的クローンに着目した古典的なクローン検出手法では，意味的クローンの検出が非常に困難である，なぜならば，意味的クローンは，振る舞いだけが一致しているに過ぎず，トークン列や命令文などの構文的構造が異なるため，その検出難度は構文的クローンの検出難度よりも非常に高くなるためである．したがって，意味的クローン検出のためには，構文的クローン検出とは異なるアプローチが必須となる．

近年では，意味的クローンを検出するために，深層学習的なアプローチからクローンを検出する手法が提案され始めている [5]〜[8]．多数の深層学習を用いたクローン検出手法が提案され続ける一方で，それらの検出手法の評価に課題が発生する．多くの深層学習を用いたクローン検出手法では，意味的クローン検出性能評価のために，BigCloneBench [9], [10] と呼ばれる，オープンソースソフトウェアに含まれるソースコードから構成された巨大なベンチマークが用いられる．BigCloneBench は，構文的クローンだけでなく，多くの意味的クローンを正解クローンとして含むため，正解クローンを大量に学習した場合の意味的クローン

† 大阪大学，吹田市
Osaka University, Suita-shi, 565–0871 Japan
a) E-mail: t-tsuru@ist.osaka-u.ac.jp
b) E-mail: matusita@ist.osaka-u.ac.jp
c) E-mail: higo@ist.osaka-u.ac.jp
DOI:10.14923/transinfj.2023JDP7022

検出性能の評価が可能である．しかし，Krinke らの報告によると，異なる振る舞いを行うソースコードのペアが正解クローンとして定義されているなど，意味的クローンの検出性能評価において多くの問題が存在する [11]．BigCloneBench を意味的クローンの検出性能評価に用いる場合，誤った正解クローンの影響で妥当性が脅かされると考えられるため，意味的クローンの検出を目的とする，深層学習を用いたクローン検出手法における評価に対して，結果の信頼性が低下する [11].

本研究では，深層学習を用いたクローン検出手法における評価の見直しを目的として，SemanticCloneBench [12] を題材とした深層学習に基づくクローン検出手法における評価を実施する．比較対象の深層学習を用いたクローン検出手法として，ASTNN [6], CodeBERT [7]，InferCode [8] の 3 手法を採用する．

SemanticCloneBench は，意味的クローンの検出性能評価に特化したベンチマークである．SemanticCloneBench では，プログラミング言語における質問回答 Web サイトである Stack Overflow の同一質問に含まれる回答ソースコードを，意味的クローンとして定義する．SemanticCloneBench では，正解クローンの定義が明確である，かつ，全てのソースコードペアに対する手動検証により，類似度の低いコードのペアが SemanticCloneBench 中に混入される可能性が低くなるため，SemanticCloneBench を用いた意味的クローン検出性能の評価における結果は，BigCloneBench を用いた意味的クローン検出性能の評価における結果よりも信頼できると考えられる．

適用実験では，意味的クローン検出において，クローン検出手法ごとに F 値が高くなるようなしきい値を明らかにするために，SemanticCloneBench に対してしきい値を変化させた場合の F 値曲線の観察を実施する．F 値が高くなるようなしきい値の範囲が広いクローン検出手法は，最適なしきい値を事前に選択できない場合でも高い性能をもたらしうると考えられる．したがって，しきい値を変化させた場合の F 値曲線の観察により，最適なしきい値を事前に選択できない場合でも高い性能をもたらしうる手法を特定できる．適用実験の結果より，SemanticCloneBench に対して，RNN を用いた手法である ASTNN において F 値が高くなるようなしきい値の範囲は，CodeBERT や InferCode といった事前学習モデルを用いた手法で高い F 値を示したしきい値の範囲よりも広いと判明した．また，事前学習モデルを用いた手法で高い F 値を示したしきい値の範囲は，ASTNN において F 値が高くなるようなしきい値の範囲と比べて，しきい値の高い位置に分布している．

本研究の貢献は，以下の 3 点である．

- SemanticCloneBench を用いた評価実験の実施
- 教師あり学習を用いたクローン検出手法と教師なし学習を用いたクローン検出手法とを初めて比較
- 意味的クローン検出において F 値が高くなるようなしきい値の範囲の観察

2. 準　　備

2.1 コードクローン

コードクローン（クローン）とは，他と一致または類似する箇所があるソースコード片である．また，互いにクローン関係にあるソースコード片のペアを，クローンペアと呼ぶ．

一般的に，クローンは，構文的な類似度の違いに基づき，Type-1 から Type-4 までの 4 種類に分類される [13].

- Type-1: 空白や改行，及びコメントの差異を除いて完全一致するクローン．
- Type-2: 変数名や関数名などの識別子名の差異を除いて一致するクローン．
- Type-3: 命令文単位での挿入や削除，変更が行われたクローン．
- Type-4: 同じ振る舞いであるにもかかわらず，構文的に異なるクローン．

上記クローンのうち，Type-1 から Type-3 までを構文的クローン，Type-4 を意味的クローンと呼ぶ．

これまでに，多数のクローン検出手法が提案されている [3], [4]．クローンの検出により，利用 API の推薦 [14] や，冗長な記述に対するリファクタリングの推薦 [15]，同一変更を要する箇所に対する変更推薦 [16]，ライセンス違反の検出 [17] といった様々な応用が可能となる．また Type-4 クローンの検出により，ソースコードをクエリとしたソースコード検索 [1] や，API の使用例に基づく再利用のためのソースコード推薦 [2] などの応用が期待される．

2.2 BigCloneBench

BigCloneBench [9], [10] は，クローン検出性能評価で用いられる大規模なベンチマークである．BigCloneBench には，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(注1)からマイニングされ，手動または自動で振り分けられた，特定の機能ごとにおけるクローン情報等のラベルが含まれる．Type-3 及び Type-4 の境界が曖昧であるため，BigCloneBench では，命令文単位での構文的類似度に基づいて，[0.0, 1.0) の範囲で Type-3/Type-4 クローンを定義している．

- Strongly Type-3: [0.7, 1.0)
- Moderately Type-3: [0.5, 0.7)
- Weakly Type-3/Type-4: [0.0, 0.5)

BigCloneBench は，多くの古典的なクローン検出手法において，クローン検出性能評価のベンチマークとして幅広く使用されている [18], [19]．また，BigCloneBench に含まれる正解クローンペアの大多数が Weakly Type-3/Type-4 であるため，深層学習を用いた手法における性能評価ベンチマークとしても使用されている [6].

しかし，Krinke らの調査により，BigCloneBench には，異なる振る舞いを行うソースコードのペアが正解クローンとして定義されている，複数機能を有するソースコードの存在を考慮していないなど，Type-4 クローンの検出性能評価において多くの問題が報告されている [11].

BigCloneBench を Type-4 クローンの検出性能評価に用いる場合，上記の問題により，誤った正解クローンの影響で妥当性が脅かされるため，Type-4 クローンの検出を目標とする，深層学習を用いたクローン検出手法の評価結果に対する信頼性が低下する．

3. 関 連 研 究

3.1 深層学習を用いたクローン検出手法

深層学習を用いたクローン検出では，教師あり学習を用いた手法が多数提案されている [5]〜[7]．一方，教師なし学習を用いた手法も数件提案されている [8]．教師あり学習を用いた手法では，学習のために検出対象データセットに対して正解クローンが必須である．したがって，あらかじめ検出対象データセット内部に含まれるソースコードのペアに対して，人力で正解クローンをアノテーションする必要があり，正解クローン集合に依存してクローンの検出能力が制限される．教師あり学習を用いたクローン検出手法における正解クローンの必要性を軽減するために，教師なし学習を用いた手法が提案され始めている [8].

(注1)：https://github.com/clonebench/BigCloneBench

著者らが知る限り，教師あり学習を用いた手法と教師なし学習を用いた手法を直接比較した研究は存在しない．教師あり学習を用いた手法と教師なし学習を用いた手法との未比較により，深層学習を用いたクローン検出手法の文脈において，教師あり学習と教師なし学習との検出性能の違いが不明瞭となる．したがって，本研究では，教師あり学習を用いた手法と教師なし学習を用いた手法との比較を実施する．具体的には，適合率 (Precision)，再現率 (Recall)，F 値 (F1-Score) の 3 種類を用いて，教師あり学習を用いた手法と教師なし学習を用いた手法との検出性能の違いを明瞭にする．

3.2 Type-4 クローン検出の評価ベンチマーク

Type-4 クローン検出手法の評価に用いられるベンチマークとして，BigCloneBench [9], [10] のほかに，OJClone [20], Google Code Jam [5], SemanticCloneBench [12] が提案されている．

OJClone [20] や Google Code Jam [5] は，オンラインプログラミングコンテスト上に提出されたソースコードから Type-4 クローンを収集したベンチマークである．オンラインプログラミングコンテスト上に提出されたソースコードの数は膨大であり，かつ，同一問題に対する解答ソースコードの集合をクローンとして定義できるため，OJClone や Google Code Jam をベンチマークとして使用可能である．しかし，オンラインプログラミングコンテスト上に提出されたソースコードには，実際のプロジェクトには見られないプログラミングコンテスト特有のテクニックが存在する [21].

一方，SemanticCloneBench [12] は，プログラミング言語における質問回答 Web サイトである Stack Overflow から Type-4 クローンを収集したベンチマークである．Stack Overflow に含まれる回答ソースコードは，実際のプロジェクトに含まれていない可能性が考えられる．しかし，SemanticCloneBench に含まれるクローンは，実際の開発者による Stack Overflow に投稿された質問に対する解決例の集合であるため，実際のプロジェクト開発で発生した問題に対する解決ソースコードが Stack Overflow に存在する可能性は，オンラインプログラミングコンテスト上におけるソースコードの集合に存在する可能性よりも高いと考えられる．また，Al-Omari らは，SemanticCloneBench を構成するにあたり，Java 言語で記述された全 1,000 個のコードペアに対する正解クローンの手動検証を実施するために，2 名の審査員を雇った．2 名の審査員によって実施される全てのコードペアを対象とした正解クロー

ンの手動検証により，類似度の低いコードのペアが SemanticCloneBench 中に混入される可能性が低くなる．したがって，本研究では，深層学習を用いたクローン検出手法を応用する場合を考慮して，信頼性の高い，かつ実際のプロジェクトで用いられるソースコードに近い評価用ベンチマークとして SemanticCloneBench を採用する．

4. 実　　験

図 1 は，本研究における適用実験の手順である．まず，SemanticCloneBench に含まれるソースコードを，実験対象のクローン検出器で使われる深層学習モデルに入力できるように，トークン列や抽象構文木 (AST) 等のコード表現に変換する．次に，10×10 重交差検証に基づいて訓練用データと評価用データに分割された SemanticCloneBench を用いて，SemanticCloneBench に含まれるソースコードペアごとの類似度を出力する．なお，深層学習モデル内部のハイパーパラメータとして，比較対象となる各クローン検出手法の論文で示されている値を用いる [6], [8], [22]．最後に，クローン検出器のしきい値を変化させて，しきい値ごとに計測した検出性能を示す評価尺度の表を作成する．ここで，しきい値の変化範囲は [0.0, 1.0] である．また，しきい値の変化間隔を 0.001 に設定する．本研究では，上記手順によって作成される，しきい値ごとの検出性能を示す評価尺度が記録された表を用いて，各クローン検出器の性能を評価する．適用実験は，10 コアの 2.2 GHz Intel Xeon E5-2630 v4 CPU と，5.5 TB HDD，256 GB メインメモリ，NVIDIA Tesla V100S GPU 上で実施される．また，適合率 (Precision)，再現率 (Recall)，F 値 (F1-Score) といった 3 種類の評価指標を用いて，各手法における Type-4 クローンの検出性能を評価する．

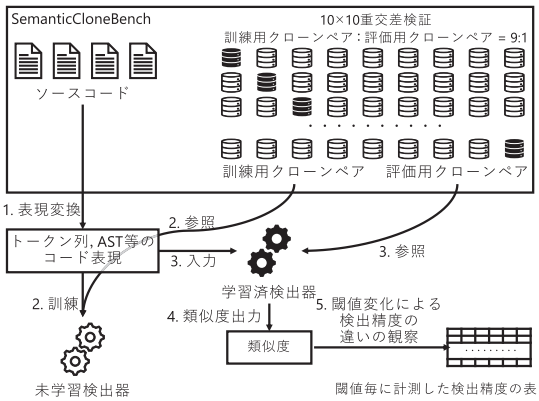

図 1　適用実験の概要

4.1 比 較 対 象

本研究では，教師あり学習を用いたクローン検出手法として ASTNN [6], CodeBERT w/ fine-tuning [7], 教師なし学習を用いたクローン検出手法として CodeBERT w/o fine-tuning，InferCode [8] を比較対象として適用実験を実施する．

4.1.1 ASTNN

ASTNN は，抽象構文木ベースのニューラルネットワークモデルである [6]．抽象構文木と回帰型ニューラルネットワーク (RNN) との併用により，字句情報や命令文単位での構文情報を学習できる．クローン検出では，ソースコードのペア同士における類似度として，L1 ノルムを使用する．

ASTNN を提案した Zhang らは，しきい値を 0.5 に設定し，BigCloneBench [9] と OJClone [20] を題材とした適用実験を実施した．彼らは，OJClone では，適合率が 98.9%，再現率が 92.7%，F 値が 0.955 であったと報告している．また，BigCloneBench に対しては，クローン分類ごとに検出性能を評価し，Weakly Type-3/Type-4 では，適合率が 99.8%，再現率が 88.3%，F 値が 0.938 であったと報告している．

適用実験では，GitHub で公開されている ASTNN リポジトリ(注2) を利用する．クローン検出のために，Skip-gram アルゴリズムを利用した Word2vec [23] のエンベディングシンボルを訓練する．エンベディングシンボルのサイズを 128 に，隠れ層のサイズを 100 に設定する．また，バッチサイズを 32 に，訓練時のエポック数を 5 に設定する．最後に，訓練時のオプティマイザ AdaMax における学習率を 0.002 に設定する．

4.1.2 CodeBERT

CodeBERT は，トークンベースの事前学習モデルである [7]．ソースコードのトークンと自然言語のテキスト，及び BERT [24] の併用により，プログラミング言語及び自然言語のマルチモーダルな情報を学習できる．CodeBERT では，ソースコードのトークンと自然言語のテキストとを事前学習したのち，ソースコードドキュメント生成などのタスクのためにファインチューニングする．

CodeBERT を提案した Feng らは，CodeBERT を用いたクローン検出に対して評価を実施していない．代

(注2)：https://github.com/zhangj111/astnn

わりに，Guo らが，BigCloneBench に対して適用実験を実施した [22]．彼らは，適合率が 94.7%，再現率が 93.4%，F 値が 0.941 であったと報告している．

本研究では，クローン検出をタスクとしてファインチューニングし，CodeBERT におけるクローン検出性能の評価を実施する．また，事前学習のみにおけるクローン検出の性能評価を行うために，ファインチューニングあり CodeBERT (CodeBERT w/ fine-tuning) とファインチューニングなし CodeBERT (CodeBERT w/o fine-tuning) の 2 種類を比較対象とする．

適用実験では，Hugging Face にて公開されている CodeBERT のモデル(注3)，(注4)を利用する．また，ソースコードのペア同士における類似度として，コサイン類似度を使用する．事前学習モデルにおける入力ソースコードトークンの長さを 256 に，バッチサイズを 16 に設定する．また，ファインチューニング時における学習率を 0.00002，エポック数を 1 に設定する．最後に，ファインチューニング時のオプティマイザとして Adam を利用する．

4.1.3 InferCode

InferCode は，抽象構文木ベースの事前学習モデルである [8]．抽象構文木と，木構造に基づいた畳み込みニューラルネットワーク (TBCNN) [20] との併用により，事前学習モデルとして，ソースコードクラスタリングなどの教師なし学習タスクや，ソースコード分類などの教師あり学習タスクに応用できる．Bui らは，InferCode をクローン検出を教師なし学習タスクとして実装している．本研究でも，InferCode は教師なし学習を用いたクローン検出手法として，適用実験の比較対象とする．クローン検出では，ソースコードのペア同士における類似度として，コサイン類似度を使用する．

InferCode を提案した Bui らは，しきい値を 0.8 に設定し，BigCloneBench [9] と OJClone [20] を題材とした適用実験を実施した．彼らは，BigCloneBench では，適合率が 90%，再現率が 56%，F 値が 0.75，OJClone では，適合率が 61%，再現率が 70%，F 値が 0.64 であったと報告している．

適用実験では，PyPI にて公開されている InferCode のモデル(注5)，(注6)を利用する．また，クローン検出時におけるバッチサイズを 5 に設定する．

4.2 SemanticCloneBench

適用実験では，Type-4 クローン検出の性能評価に用いるベンチマークとして，SemanticCloneBench [12] を用いる．SemanticCloneBench は，Type-4 クローンの検出性能評価に特化したベンチマークである．SemanticCloneBench に含まれるクローンは，プログラミング言語における質問回答 Web サイトである Stack Overflow(注7)から抽出されている．SemanticCloneBench では，同一質問に含まれる回答ソースコードを，Type-4 クローンとして定義している．SemanticCloneBench では，C，C#，Java，Python で記述されたソースコードが収集されており，言語ごとにそれぞれ 1,000 個の Type-4 クローンのペアが含まれている．

適用実験では，SemanticCloneBench に含まれる Java 言語で記述された 1,000 個のクローンのうち，Python の javalang(注8)パッケージで構文解析可能な 997 個のクローンを対象とする．また，正解クローンと非正解クローンの比が 1 : 1 となるように，非正解クローンを作成する．具体的には，SemanticCloneBench に含まれるメソッドをランダムに 2 個抽出し，抽出したメソッドのペアが SemanticCloneBench でクローンとして定義されていないならば，このメソッドのペアを非正解クローンとして定義する．更に，上記の作業により作成された非正解クローンを含めた SemanticCloneBench を 10×10 重交差検証に基づいて学習及び評価を実施する．10×10 重交差検証は，10 重交差検証を 10 回実行する評価手法であり，10×10 重交差検証によって得られる結果の平均値を計算し，クローン検出器の検出性能を計測する．

4.3 Research Questions

本研究では，以下に示す 2 個の Research Question（RQ）を立てる．

- RQ1: しきい値に基づく評価指標より，どのクローン検出器の検出性能が高いか．
- RQ2: Type-4 クローン検出では，各クローン検出器において F 値が高くなるようなしきい値の範囲がどのように分布するのか．

RQ1 の調査により，SemanticCloneBench を対象とした Type-4 クローン検出において，F 値が一番高いときのしきい値を基に，検出性能が一番高いクローン検

（注3）：https://github.com/microsoft/CodeBERT
（注4）：https://huggingface.co/microsoft/codebert-base
（注5）：https://github.com/bdqnghi/infercode
（注6）：https://pypi.org/project/infercode
（注7）：https://stackoverflow.com/
（注8）：https://github.com/c2nes/javalang

出器を明示する．そして，各検出器において検出性能の違いが生じた理由を考察する．

RQ2 の調査により，Type-4 クローン検出において，クローン検出手法ごとに F 値が高くなるようなしきい値の範囲を明らかにし，最適なしきい値を事前に選択できない場合でも高い性能をもたらしうるクローン検出手法を特定する．なお，SemanticCloneBench 以外のデータセットにおいて F 値が高くなるようなしきい値の範囲は不明であるため，本研究では，SemanticCloneBench に対して F 値が最大となるしきい値付近を，F 値が高くなるようなしきい値の範囲と定義する．ここで，F 値が最大となるしきい値付近とは，F 値が [(F 値の最大値の 0.95 倍), (F 値の最大値)] となるときのしきい値とする．

F 値が高くなるようなしきい値の範囲が広いクローン検出手法は，最適なしきい値を事前に選択できない場合でも高い性能をもたらしうると考えられる．一方で，F 値が高くなるようなしきい値の範囲が狭いクローン検出手法は，最適なしきい値を事前に選択できない場合において，その手法が有するはずの検出性能が発揮できないと予想される．

5. 実験結果

5.1 RQ1: しきい値に基づく検出性能の評価

表 1 は，SemanticCloneBench 上で，各クローン検出器の F 値が最大となるしきい値における，適合率，再現率，F 値である．

ASTNN は，しきい値が 0.393 のときに F 値が最大（0.921）となり，適合率と再現率はそれぞれ 87.5%，97.3% を得る．他のクローン検出器と比べると，再現率の値が約 12 ポイント–14 ポイント，また F 値が 0.042–0.162 高く，本実験において最も高い検出性能を示す．ASTNN では，検出対象データセットごとに，そのデータセットに含まれるトークンや抽象構文木のノードに対するエンベディングを構成するため，再現率及び F 値の値が高いと考えられる．

CodeBERT w/ fine-tuning は，しきい値が 0.918 のときに F 値が最大（0.879）となり，適合率と再現率はそれぞれ 90.5%，85.4% を得る．他のクローン検出器と比べると，適合率の値が約 3 ポイント–約 21 ポイント高く，CodeBERT w/ fine-tuning は，本実験において最も誤検出を起こしにくいクローン検出器であると言える．CodeBERT w/ fine-tuning では，正解クローンのファインチューニングにより，事前学習で得たコード表現だけでなく，意味的クローンの分類能力が強化される．これにより，CodeBERT w/ fine-tuning の適合率が他のクローン検出手法よりも高くなると考えられる．

教師なし学習を用いたクローン検出器である CodeBERT w/o fine-tuning は，しきい値が 0.985 のときに F 値が最大（0.791）となり，適合率と再現率はそれぞれ 74.9%，83.7% を得る．また，InferCode は，しきい値が 0.884 のときに F 値が最大（0.759）となり，適合率と再現率はそれぞれ 69.4%，83.7% を得る．教師あり学習を用いた検出器と比べると，両検出器の適合率は約 13 ポイント–21 ポイント，また F 値は 0.088–0.162 低い．Type-4 クローンにおける検出難度の高さ，及びコード表現のみの事前学習を用いたクローン検出により，教師なし学習を用いたクローン検出器における適合率及び F 値は，教師あり学習を用いた検出器と比べて低い値になると考えられる．一方，CodeBERT w/o fine-tuning と InferCode における再現率はどちらも 83.7% であり，古典的手法の一つである NiCad [25] の再現率（4.0%）[12] よりも非常に高い．また，教師あり学習を用いた手法と比べて，再現率の減少が高々約 14 ポイントに抑えられている．

表 2 は，BigCloneBench を用いたときの実験結果と，SemanticCloneBench を用いたときの実験結果を示した表である．SemanticCloneBench に対する ASTNN を用いたクローン検出では，BigCloneBench と比べて適合率が約 12 ポイント減少した一方で，再現率は約 9 ポイント増加した．その結果，SemanticCloneBench における ASTNN の F 値は，BigCloneBench と比較して 0.017 減少した．一方，SemanticCloneBench に対する CodeBERT w/ fine-tuning を用いたクローン検出では，BigCloneBench と比べて適合率・再現率ともに減少し，それぞれ約 4 ポイント，約 8 ポイントであった．その結果，SemanticCloneBench における CodeBERT w/ fine-tuning の F 値は，BigCloneBench と比較して 0.062 減少した．SemanticCloneBench に対する InferCode を用いたクローン検出では，BigCloneBench と比べて適合率が約 20 ポイント減少した一方で，再現率は約 28

表 1　SemanticCloneBench に対する各クローン検出器の適合率，再現率，F 値

	しきい値	適合率	再現率	F 値
ASTNN	0.393	87.5%	97.3%	0.921
CodeBERT w/ fine-tuning	0.918	90.5%	85.4%	0.879
CodeBERT w/o fine-tuning	0.985	74.9%	83.7%	0.791
InferCode	0.884	69.4%	83.7%	0.759

表 2　BigCloneBench 及び SemanticCloneBench を対象としたクローン検出手法の実験結果

	BigCloneBench [6], [8], [22]			SemanticCloneBench		
	適合率	再現率	F 値	適合率	再現率	F 値
ASTNN	99.8%	88.4%	0.938	87.5%	97.3%	0.921
CodeBERT w/ fine-tuning	94.7%	93.4%	0.941	90.5%	85.4%	0.879
InferCode	90%	56%	0.75	69.4%	83.7%	0.759

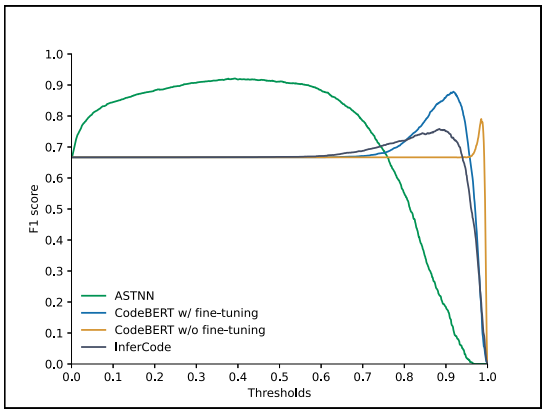

図 2　SemanticCloneBench に対してしきい値を変化させた場合の F 値曲線

表 3　SemanticCloneBench に対する各クローン検出器において F 値が高くなるようなしきい値の範囲

	F 値の範囲	しきい値の範囲
ASTNN	[0.875, 0.921]	[0.175, 0.617]
CodeBERT w/ fine-tuning	[0.835, 0.879]	[0.876, 0.935]
CodeBERT w/o fine-tuning	[0.751, 0.791]	[0.980, 0.989]
InferCode	[0.721, 0.759]	[0.803, 0.921]

ポイント増加した．結果として，SemanticCloneBench における InferCode の F 値は，BigCloneBench における F 値とほぼ同じ値の 0.79 であった．

5.2　RQ2: Type-4 クローン検出において F 値が高くなるようなしきい値の範囲

図 2 は，SemanticCloneBench に対してしきい値を変化させた場合の F 値曲線である．また，表 3 は，適用実験の比較対象である各クローン検出手法において F 値が高くなるようなしきい値の範囲の一覧である．

本実験より，SemanticCloneBench に対する，CodeBERT や InferCode などの事前学習モデルを用いた手法で高い F 値を示したしきい値の範囲は，RNN を用いた手法である ASTNN において F 値が高くなるようなしきい値の範囲よりも狭いと判明した．また，事前学習モデルを用いた手法で高い F 値を示したしきい値の範囲は，ASTNN において F 値が高くなるようなしきい値の範囲と比べて，しきい値の高い位置に分布している．

事前学習モデルを用いたクローン検出手法では，クローン検出の前にコード表現を事前学習するため，RNN などの正解クローンを直接学習する手法よりも，類似度の値が高く出力される．したがって，事前学習モデルに基づくクローン検出手法を Type-4 クローン検出に用いる場合，SemanticCloneBench に対するクローン検出のときと同様に，しきい値を高めに設定する必要があると考えられる．

CodeBERT を含む事前学習モデルに基づくクローン検出手法は，F 値が高くなるようなしきい値の範囲が狭い．したがって，事前学習モデルを用いたクローン検出器において，しきい値を適切に設定できない場合の F 値は，最大値と比べて非常に小さくなるため，事前学習モデルを用いたクローン検出器が有するはずの高い検出性能を発揮できないと予想される．

一方，ASTNN では，F 値が高くなるようなしきい値の範囲が広い．したがって，ASTNN において，仮に F 値が最大となるようなしきい値を設定できなかったとしても，ASTNN の F 値が最大となるような状態に近い検出性能を発揮できる可能性が高い．

6.　妥当性への脅威

適用実験で使用した SemanticCloneBench に含まれる正解クローンペア数は 997 個であり，BigCloneBench に含まれる Weakly Type-3/Type-4 の正解クローンペア数（約 8,400,000 個）と比べて遥かに小さい．そのため，RQ1，RQ2 の結果に影響を及ぼすと考えられる．今後，実際のプロジェクトで頻繁に出現すると期待される意味的クローンを抽出して，巨大な，かつ，多くの研究者によって審議された意味的クローンの検出性能を評価できるベンチマークの作成が必要である．

また，本研究では，SemanticCloneBench に限定して，Type-4 クローン検出において F 値が高くなるようなしきい値を予測しており，これは，RQ2 に対する妥当性への脅威である．今後，SemanticCloneBench 以外の Type-4 クローン検出評価ベンチマークに対する評価も実施し，一般的な Type-4 クローンにおいても F 値が

高くなるしきい値の有効性を検証したい．

更に，適用実験では，各クローン検出手法において，比較対象クローン検出手法の論文で示されているハイパーパラメータの値を使用している．比較対象クローン検出手法では，ハイパーパラメータの具体的な調節プロセスが明記されていないため，これは RQ1 や RQ2 に対する妥当性への脅威である．今後，各クローン検出手法のハイパーパラメータが，本研究の実験結果に対してどのように影響を及ぼすかの検証が必須である．

7. むすび

本研究では，教師あり学習を用いたクローン検出手法である ASTNN と CodeBERT w/ fine-tuning，教師なし学習を用いたクローン検出手法である CodeBERT w/o fine-tuning と InferCode に対して，クローン検出性能の評価を実施した．SemanticCloneBench を題材とした適用実験により，教師あり学習を用いた手法と教師なし学習を用いた手法の両手法において，Type-4 クローンを検出できた．教師あり学習を用いた手法と教師なし学習を用いた手法の F 値は，それぞれ 0.85，0.75 を超えた．また，F 値が高くなるようなしきい値の範囲を調査した結果，CodeBERT と InferCode で高い F 値を示したしきい値の範囲は，ASTNN において F 値が高くなるようなしきい値の範囲よりも狭いと判明した．

今後の課題として，実際の開発において発生するクローンで構成された，巨大な Type-4 クローンの検出性能評価ベンチマークの作成が挙げられる．

謝辞　本研究の一部は，JSPS 科研費（18H04094，JP20H04166，JP21K18302，JP21K11820，P21H04877，JP22H03567，JP22K11985）を得て行われた．

文　献

[1] Y. Fujiwara, N. Yoshida, E. Choi, and K. Inoue, “Code-to-code search based on deep neural network and code mutation,” Proc. International Workshop on Software Clones, pp.1–7, 2019.

[2] S. Abid, S. Shamail, H.A. Basit, and S. Nadi, “FACER: An API usage-based code-example recommender for opportunistic reuse,” Empirical Software Engineering, vol.26, no.6, pp.1–58, June 2021.

[3] A. Sheneamer and J. Kalita, “A survey of software clone detection techniques,” International Journal of Computer Applications, vol.137, no.10, pp.1–21, Oct. 2016.

[4] H. Min and Z. Li Ping, “Survey on software clone detection research,” Proc. Int. Conf. Management Engineering, Software Engineering and Service Sciences, pp.9–16, 2019.

[5] G. Zhao and J. Huang, “DeepSim: Deep learning code functional similarity,” Proc. Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp.141–151, 2018.

[6] J. Zhang, X. Wang, H. Zhang, H. Sun, K. Wang, and X. Liu, “A novel neural source code representation based on abstract syntax tree,” Proc. Int. Conf. Software Engineering, pp.783–794, 2019.

[7] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, and M. Zhou, “CodeBERT: A pre-trained model for programming and natural languages,” Proc. Conf. Empirical Methods in Natural Language Processing, pp.1536–1547, 2020.

[8] N.D.Q. Bui, Y. Yu, and L. Jiang, “InferCode: Self-supervised learning of code representations by predicting subtrees,” Proc. Int. Conf. Software Engineering, pp.1186–1197, 2021.

[9] J. Svajlenko, J.F. Islam, I. Keivanloo, C.K. Roy, and M.M. Mia, “Towards a big data curated benchmark of inter-project code clones,” Proc. Int. Conf. Software Maintenance and Evolution, pp.476–480, 2014.

[10] J. Svajlenko and C.K. Roy, “BigCloneBench,” Code Clone Analysis: Research, Tools, and Practices, pp.93–105, Springer, 2021.

[11] J. Krinke and C. Ragkhitwetsagul, “BigCloneBench considered harmful for machine learning,” Proc. International Workshop on Software Clones, pp.1–7, 2022.

[12] F. Al-Omari, C.K. Roy, and T. Chen, “SemanticCloneBench: A semantic code clone benchmark using crowd-source knowledge,” Proc. International Workshop on Software Clones, pp.57–63, 2020.

[13] K. Inoue, “Introduction to code clone analysis,” Code Clone Analysis: Research, Tools, and Practices, pp.3–27, Springer, 2021.

[14] I. Keivanloo, J. Rilling, and Y. Zou, “Spotting working code examples,” Proc. Int. Conf. Software Engineering, pp.664–675, 2014.

[15] N. Yoshida, S. Numata, E. Choi, and K. Inoue, “Proactive clone recommendation system for extract method refactoring,” Proc. International Workshop on Refactoring, pp.67–70, 2019.

[16] M. Mondal, C.K. Roy, and K.A. Schneider, “An exploratory study on change suggestions for methods using clone detection,” Proc. Int. Conf. Computer Science and Software Engineering, pp.85–95, 2016.

[17] A. Monden, S. Okahara, Y. Manabe, and K. Matsumoto, “Guilty or not guilty: Using clone metrics to determine open source licensing violations,” IEEE Software, vol.28, no.2, pp.42–47, 2010.

[18] M. Wu, P. Wang, K. Yin, H. Cheng, Y. Xu, and C.K. Roy, “LVMapper: A large-variance clone detector using sequencing alignment approach,” IEEE Access, vol.8, pp.27986–27997, 2020.

[19] T. Nakagawa, Y. Higo, and S. Kusumoto, “NIL: Large-scale detection of large-variance clones,” Proc. Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp.830–841, 2021.

[20] L. Mou, G. Li, L. Zhang, T. Wang, and Z. Jin, “Convolutional neural networks over tree structures for programming language processing,” Proc. AAAI Conf. Artificial Intelligence, pp.1287–1293, 2016.

[21] A. Laaksonen, Guide to competitive programming: Learning and improving algorithms through contests, second edition, Springer, 2020.

[22] D. Guo, S. Ren, S. Lu, Z. Feng, D. Tang, S. Liu, L. Zhou, N. Duan, A. Svyatkovskiy, S. Fu, M. Tufano, S.K. Deng, C. Clement, D. Drain, N. Sundaresan, J. Yin, D. Jiang, and M. Zhou, "GraphCodeBERT: Pre-training code representations with data flow," https://arxiv.org/abs/2009.08366, 2020.
[23] T. Mikolov, I. Sutskever, K. Chen, G.S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," Proc. Int. Conf. Neural Information Processing Systems, pp.3111–3119, 2013.
[24] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," https://arxiv.org/abs/1810.04805, 2018.
[25] C.K. Roy and J.R. Cordy, "NICAD: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization," Proc. Int. Conf. Program Comprehension, pp.172–181, 2008.

（2023 年 3 月 31 日受付，8 月 6 日再受付，
10 月 18 日早期公開）

鶴　　智秋

2021 大阪大学基礎工学部情報科学科卒．2023 同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博士前期課程了．在学中，コードクローン分析に関する研究に従事．

松下　　誠

1993 大阪大学基礎工学部情報工学科卒．1998 同大学大学院基礎工学研究科博士後期課程了．同年同大学基礎工学部情報工学科助手．2002 同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助手．2005 同助教授．2007 同准教授．博士（工学）．リポジトリマイニング，プログラム解析の研究に従事．情報処理学会，日本ソフトウェア科学会，ACM 各会員．

肥後　芳樹　（正員）

2002 大阪大学基礎工学部情報科学科中退．2006 同大学大学院博士後期課程了．2007 同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助教．2015 同准教授．2022 同教授．博士（情報科学）．ソースコード分析，特にコードクローン分析，リファクタリング支援，ソフトウェアリポジトリマイニング及び自動プログラム修正に関する研究に従事．情報処理学会，日本ソフトウェア科学会，IEEE 各会員．