情報検索技術と深層学習を用いたコード片類似性判定法の比較調査

横井　一輝[†a)]　崔　恩瀞[††]　吉田　則裕[†††]　松下　誠[†]
井上　克郎[††††]

Comparing Code Similarity Using Information Retrieval Techniques and Deep Learning

Kazuki YOKOI[†a)], Eunjong CHOI[††], Norihiro YOSHIDA[†††], Makoto MATSUSHITA[†], and Katsuro INOUE[††††]

あらまし　コード片の類似性判定法はソフトウェア工学における重要な基礎技術であり，コードクローン検出やコード片検索などで使用される．コード片の類似性判定法では，構文的な類似性だけでなく，処理内容の意味的な類似性も判定することが重要である．我々の先行研究では，情報検索技術に基づくベクトル表現とコサイン類似度を組み合わせた類似性判定法を用いていた．この手法は処理内容の意味的な類似性を高速に判定できる一方で，検出漏れが多い課題がある．また近年，深層学習を用いた類似性判定法が提案されている．この手法は意味的な類似性を高い精度で判定できる反面，実行速度が遅い課題がある．そこで本研究では，判定精度と実行速度の二つの観点で情報検索技術に基づくベクトル表現と深層学習の効果的な組み合わせを調査する．調査の結果，情報検索技術の一種である LSI (Latent Semantic Indexing) と深層学習モデルの組み合わせが判定精度の面で最も高い値となった．またこの組み合わせは実行速度が最も速いことも確認した．

キーワード　ソフトウェア保守，コードクローン，情報検索技術，深層学習

1. まえがき

コード片の類似性判定法はソフトウェア工学における重要な基礎技術である．類似したコード片を見つける作業はソフトウェア開発や保守において重要であり，コードクローン検出 [1] やコード片検索 [2], [3] などではコード片の類似性判定法が使用される．コードクローンとはソースコード中に含まれる互いに一致または類似した部分をもつコード片である．一般的に，コードクローンの存在はソフトウェアの保守を困難にするといわれている．膨大な量のコードクローンを目視で見つけることは困難であり，コードクローンを自動的に検出する手法が提案されている [4], [5]．またコード片検索は，再利用可能なコード片を特定するために使用する [6]．既存のコード片を再利用することで，ソフトウェア開発の生産性及び信頼性の向上が期待できる．更にコード片検索を用いて，再利用元のライセンス記述を調べたり，より優れた脆弱性対策がされているコード片を探したりでき，再利用の安全性を高めることができる．

コード片の類似性判定法では，構文的な類似性に基づく判定法と，意味的な類似性に基づく判定法の 2 種類がある [7]．既存研究の多くはコード片の構文的な類似性に基づいて判定する [8]〜[10]．一方で，意味的な類似性に基づいた判定法は少なく，様々な課題が残されている．コード片の意味的な類似性判定法として Zhao らは DeepSim という手法を提案した [7]．DeepSim は制御フローグラフとデータフローグラフの情報をもとに，深層学習モデルを用いてコード片を判定する．DeepSim は類似したコード片を高い精度で判定できる一方で，実行速度が遅いという課題もある．

また我々は既存研究で，情報検索技術に基づくコードクローン検出法を提案した [11]．この手法は情報検索技術の一種である TF-IDF (Term Frequency-Inverse Document Frequency) [12] を用いてコード片をベクトル化し，ベクトル空間上で距離が近いコード片をコー

† 大阪大学，吹田市
Osaka University, Suita-shi, 565–0871 Japan
†† 京都工芸繊維大学，京都市
Kyoto Institute of Technology, Kyoto-shi, 606–8585 Japan
††† 立命館大学，草津市
Ritsumeikan University, Kusatsu-shi, 525–8577 Japan
†††† 南山大学，名古屋市
Nanzan University, Nagoya-shi, 466–8673 Japan
a) E-mail: k-yokoi@ist.osaka-u.ac.jp
DOI:10.14923/transinfj.2022PDP0009

ドクローンとして検出する．ベクトル空間上の距離尺度はコサイン類似度を用いる．コードクローン検出にベクトル表現を用いることで，構文や字句単位で類似していなくてもベクトル空間で距離が近ければコードクローンとして検出できる．この手法は高速に検出できる一方で，構文的な類似性が低いコードクローンの検出漏れが多い課題がある [13], [14].

構文的な類似性が低くても意味的に類似しているコード片を，高い精度で高速に判定できるコード片の類似性判定が望ましい．コード片の類似性判定法では，情報検出技術や深層学習が広く使われている．藤原らは深層学習を用いたソースコード検索において情報検索技術の一種である BoW (Bag of Words) と Doc2Vec [15] の精度の比較を行っているが，他の情報検索技術について評価していない [16]．また，再帰型ニューラルネットワークを用いてコード片をベクトル化してコード片の類似性を判定する手法はあるが [17], [18]，本研究の目的である情報検索技術を用いたコード片のベクトル化は網羅的に調査されていない．これらの研究では本研究と目的が異なるため，判定精度が高く実行速度が速い，情報検出技術と深層学習の組み合わせは明らかになっていない．そこで本研究では，意味的なコード片の類似性判定において，判定精度と実行速度の観点から有効な情報検索技術と深層学習の組み合わせを調査する．本調査では，情報検索技術に基づきコード片をベクトル表現に変換し，変換したベクトル表現を深層学習モデルに入力することで，コード片が類似しているか否かを判定する．これにより，情報検索技術と深層学習を組み合わせたコード片の類似性判定法を実現する．また本調査で使用する情報検索技術と深層学習の組み合わせを，深層学習を用いた既存手法と比較する．

本調査では，データセットとして Google Code Jam（以降 GCJ）(注1) と BigCloneBench（以降 BCB）[19] を用いて調査する．GCJ は Google が開催している競技プログラミングコンテストであり，同じ問題に正解したソースコードは類似コードとみなす．BCB はコードクローンの大規模ベンチマークであり，600 万以上のクローンペア（処理内容が類似しているコードクローンの対）と 26 万以上の非クローンペアが登録されている．これらのデータセットについて調査した結果，情報検索技術の一種である LSI (Latent Semantic Indexing) [12] と深層学習モデルを組み合わせた手法が，適合率，再現率，F 値においてより高い値となった．また，この組み合わせは実行速度が最も速いことも確認した．

以降，**2.** では，本研究の背景について述べる．**3.** では，本研究の調査手法について述べる．**4.** では，本研究の調査結果について述べる．**5.** では，調査結果から得られた考察について述べる．**6.** では，関連研究について述べる．最後に，**7.** でまとめと今後の課題について述べる．

2. 背　　景

2.1 情報検索技術に基づくコードクローン検出法

コード片の類似性判定法が用いられる場面の一つに，コードクローン検出がある [4]．コードクローンとは，ソースコード中に含まれる互いに一致または類似した部分をもつコード片である．Roy らはコードクローンの定義として，コードクローン間の編集の度合いに応じて以下の四つのタイプに分類した [1].

タイプ 1　空白やタブの有無，コーディングスタイル，コメントの有無などの違いを除き完全に一致するコードクローン．

タイプ 2　タイプ 1 の違いに加えて，変数名などのユーザ定義名，変数の型などが異なるコードクローン．

タイプ 3　タイプ 2 の違いに加えて，文の挿入や削除，変更などが行われているコードクローン．

タイプ 4　類似した処理を実行するが，構文上の実装が異なるコードクローン．

タイプ 1 から 3 のコードクローンは，構文的な類似性に基づく判定法を用いて検出できる．しかしタイプ 4 のように構文上の実装が異なるコードクローンの検出には，意味的な類似性に基づく判定法が必要である．

コードクローンを自動的に検出する手法は，これまでにも数多く提案されている [4], [5]．我々は先行研究において，情報検索技術に基づくコードクローン検出法 CCVolti(注2) を提案した [11]．CCVolti はまず，情報検索技術の一種の TF-IDF を用いてコード片をベクトル化する．そして，与えられた二つのベクトル表現間のコサイン類似度を求める．これによりコード片の処理内容が意味的に類似しているか判定し，タイプ 4 までのコードクローンを検出する．この手法は高速に検出できる一方で，構文的な類似性の低いコードクロー

（注1）：https://codingcompetitions.withgoogle.com/codejam/

（注2）：https://github.com/k-yokoi/CCVolti

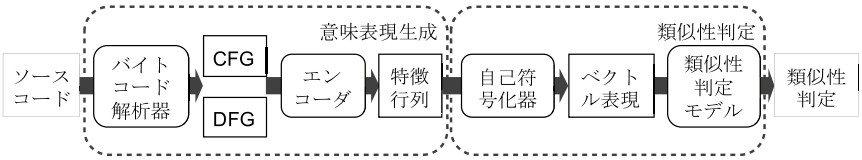

図 1　DeepSim の概要

ンの検出漏れが多い課題が指摘されている [13], [14].

2.2　深層学習を用いたコード片類似性判定法

深層学習を用いたコード片の類似性判定法として，Zhao らは DeepSim [7] を提案した．DeepSim の概要を図 1 に示す．この図が示すように，DeepSim は前半の意味表現生成過程と後半の類似性判定過程に分かれる．意味表現生成過程では，制御フローグラフ（以降 CFG）とデータフローグラフ（以降 DFG）を解析し，特徴行列に変換することでコード片の意味表現を生成する．また，類似性判定過程では，ニューラルネットワークの一種である自己符号化器 [20] を用いて，特徴行列の情報を最もよく表すベクトル表現に変換する．その後，ベクトル表現を類似性判定モデルに入力し，二つのコード片が類似しているか否かの二値分類を行う．DeepSim は高い精度でコード片の類似性を判定する．一方で DeepSim はモデルの学習に時間がかかるという課題がある [7].

3. 調 査 手 法

本研究では，コード片類似性判定法に用いる情報検索技術と深層学習の組み合わせについて調査する．また，情報検索技術と深層学習の組み合わせについて，深層学習を用いた既存手法である DeepSim [7] と FA-AST [21] の二つの結果と比較する．本章ではその調査手法について述べる．

3.1　調査目的とリサーチクエスチョン

2.1 で述べた情報検索技術に基づくコードクローン検出法 [11] は高速な検出が可能だが，構文的な類似性が低いコードクローンの検出漏れが多い課題がある [13], [14]．一方で，**2.2** で述べた深層学習を用いた類似性判定法は高い精度でコード片の類似性を判定するが，モデルの学習に時間がかかる課題がある [7]．コード片の類似性判定はソフトウェア開発や保守において広く使われており，判定精度が高く実行速度が速い手法が望ましい．しかし，判定精度と実行速度の観点でどの情報検出技術と深層学習の組み合わせが，判定精度が高く実行速度が速い手法かが明らかになっていない．そこで，本調査では二つのリサーチクエスチョンを設定した．

RQ1　高い精度で類似性を判定する情報検索技術と深層学習の組み合わせは何か？

RQ2　短時間で類似性を判定する情報検索技術と深層学習の組み合わせは何か？

この二つのリサーチクエスチョンを解くことで，精度と実行速度の二つの観点から有用な情報検索技術と深層学習の組み合わせを明らかにする．

3.2　調査に用いるコード片類似性判定法

本調査で用いるコード片類似性判定法は以下の三つのステップで実行する．手法の概要を図 2 に示す．

STEP 1　ソースコード解析を用いた前処理を行い，コード片を単語列に変換する

STEP 2　情報検索技術に基づき単語列をベクトル表現に変換する

STEP 3　ベクトル表現を類似性判定モデルに入力し，深層学習によりコード片の類似性を判定する

3.2.1　STEP 1：ソースコード解析を用いた前処理

最初に，入力コード片に対して字句解析を行いトークン列に変換する．字句解析には構文解析器生成系 ANTLR(注3) が生成した字句解析器を用いる．

次にトークン列に対して前処理を行い単語列に変換する．本調査では以下の前処理を行う．

- 予約語と識別子以外を除去
- 識別子名をキャメルケースやスネークケースをもとに分割
- 分割後の識別子を全て小文字に正規化

これらの前処理は既存研究と同じ方法を用いる [11]．識別子分割や正規化は，情報検索技術によりソースコードを解析する際に有用な手法として用いられる [22].

3.2.2　STEP 2：情報検索技術に基づくベクトル化

次に STEP1 で生成した単語列に対して，情報検索技術に基づくベクトル化によりベクトル表現に変換する．情報検索技術に基づくベクトル化には Python ラ

（注3）：https://www.antlr.org/

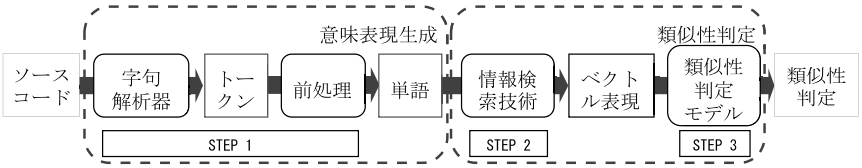

図 2　調査に用いるコード片類似性判定法の概要

イブラリ gensim を用いる [23].

本研究では情報検索技術に基づくベクトル表現として，LSI (Latent Semantic Indexing), LDA (Latent Dirichlet Allocation), Doc2Vec, WV-avg (Word2Vec average) の四つのベクトル表現を調査対象として選択した．WV-avg は単語ベクトル Word2Vec [24], [25] の平均ベクトルを意味する．調査対象のベクトル表現は，先行研究 [13] において採用されたベクトル表現を参考に選択した．また，先行研究で使用されているベクトルの表現のうち，BoW (Bag of Words) と TF-IDF はベクトルの次元数が変動するため，FT-avg は WV-avg より再現率が低く計算速度も遅い [13] ため，本調査では使用しない．ただし Doc2Vec の実装として PV-DBoW (Distributed Bag of Words version of Paragraph Vector) と PV-DM (Distributed Memory version of Paragraph Vector) の二つの異なるアルゴリズムが提案されている [15] ため，本調査ではこの二つのアルゴリズムを別個のベクトル表現として使用する．したがって，LSI, LDA, PV-DBoW, PV-DM, WV-avg の五つのベクトル表現を調査対象とする．なお Word2Vec の実装としても複数のアルゴリズムが提案されているが，本研究では SGNS (skip-gram algorithm with negative sampling) [25] を用いる．

本研究で対象とするベクトル表現と，そのベクトル表現に与える主なハイパーパラメータを付録 **1.** に示す．ハイパーパラメータは Python ライブラリ gensim のデフォルト値を参考に決定した．ハイパーパラメータのチューニングにより判定精度が上昇する可能性がある一方で，実行速度が遅くなる懸念もある．本調査では，ハイパーパラメータチューニングを行わずにライブラリのデフォルト値における判定精度と実行速度を調査する方針とした．

3.2.3　STEP 3：深層学習を用いた類似性判定モデル

最後に，STEP2 で生成したベクトル表現に対して，深層学習モデルを用いてコード片の類似性を判定する．本研究では順伝播型ニューラルネットワークをもとにコード片類似性判定モデルを設計した．コード片類似性判定モデルのアーキテクチャを図 3 に示す．図 3 が示すように，二つのコード片から得られたベクトル表現 V_a と V_b を第 1 層に入力する．第 2 層では，入力した二つのベクトルを $[V_a, V_b]$，$[V_b, V_a]$ と異なる順序で連結する．このとき二つの連結ベクトルは重み w_1 を共有して計算される．第 3 層は平均プーリング層である．二つの層からなる第 2 層の出力値の平均値を第 3 層への入力として計算する．二つの異なる順序でベクトルを連結し平均プーリングを行うことは，二つのベクトルの対称性を成立させる役割を果たしている．最後は，出力層としてロジスティック関数を用いてコード片の類似性を出力する．ロジスティック関数は入力を u として次の関数であらわされる ($f(u) = \frac{1}{1+e^{-u}}$)．出力値は 0 から 1 の間であり，出力値が 0.5 以上であれば類似している，0.5 未満であれば類似していないと判定する．活性化関数は ReLU，損失関数は CrossEntropy，最適化アルゴリズムは Adam を用いる．このニューラルネットワークアーキテクチャを実現するために，本研究では TensorFlow(注4) と Keras(注5) を用いて実装した．

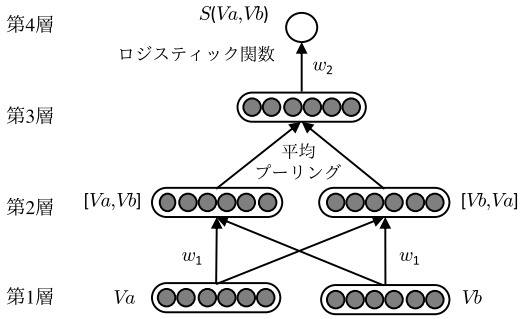

図 3　コード片類似性判定モデルのアーキテクチャ

3.3　比較調査対象

本調査では，情報検索技術と深層学習の組み合わせの 5 種類と **2.2** で説明した DeepSim の，合計 6 種類を比較調査対象とする．

(注4)：https://www.tensorflow.org/
(注5)：https://keras.io/

3.3.1 情報検索技術と深層学習の組み合わせ

3.1 で説明した二つの RQ に答えるため，**3.2.2** で説明した 5 種類のベクトル表現と **3.2.3** で説明した深層学習モデルの組み合わせを比較調査対象とする．本論文では，深層学習モデルに入力した 5 種のベクトル表現を区別するために以下の表記を用いる．NN はニューラルネット (Neural Network) の略である．

- LSI+NN (Latent Semantic Indexing + Neural Network)
- LDA+NN (Latent Dirichlet Allocation + Neural Network)
- PV-DBoW+NN (Distributed Bag of Words version of Paragraph Vector + Neural Network)
- PV-DM+NN (Distributed Memory of Words version of Paragraph Vector + Neural Network)
- WV-avg+NN (Word2Vec average vector + Neural Network)

深層学習モデルで用いたハイパーパラメータは付録 **2.** に示す．ハイパーパラメータは Python ライブラリ gensim のデフォルト値を参考に決定した．

3.3.2 深層学習を用いた既存手法

情報検索技術と深層学習の組み合わせを，深層学習を用いた既存手法とも比較した．本調査では Zhao らの DeepSim [7] と Wang らの FA-AST+GMN [21] の二つの既存手法を比較調査対象とする．DeepSim は CFG と DFG を解析して得た意味表現から自己符号化器を用いてベクトル表現を生成し，深層学習モデルを用いてコード片の類似性を判定する．また FA-AST+GMN は抽象構文木に CFG と DFG の情報を加えて拡張したグラフ表現からグラフニューラルネットワークを用いてベクトル表現を生成し，コサイン類似度を用いてコード片の類似性を判定する．

DeepSim と FA-AST+GMN は **3.4** で説明する Google Code Jam と BigCloneBench [19] をデータセットして評価を行っている．そのため，深層学習を用いたコード片類似性判定法の中からこの二つの既存手法を採用した．なお，Wang らは論文中 FA-AST+GGNN と FA-AST+GMN の二つの手法を評価しているが，FA-AST+GMN の方が F 値が高かった [21] ため，本調査では FA-AST+GMN のみを比較調査対象とする．DeepSim と FA-AST+GMN で用いたハイパーパラメータは付録 **2.** に示す．

3.4 対象データセット

本調査では，オンラインジャッジサイトと広く使われているベンチマークの 2 種類のデータセットを用いて調査する．オンラインジャッジサイトは意味的な類似性判定のベンチマークとして信頼性がある [7]．また広く使われているベンチマークと比較することは，他の手法と比較しやすいことから重要である．これら 2 種類のデータセットを用いることで，より網羅的な調査が可能となる．更に本調査においてはデータセットが広く公開されているという観点から，オンラインジャッジサイトとして Google Code Jam（以降 GCJ）を，ベンチマークとして BigCloneBench（以降 BCB）[19] を用いて調査した．

GCJ は Google(注6) が実施している競技プログラミングコンテストであり，提出されたソースコードは Google によって挙動を検証されている．これにより，異なるプログラマが提出したソースコードも同一の挙動をすることを保証されており，これらは意味的に類似したソースコードとみなせる．本研究では 12 種の問題から集めた 1,669 個の提出コードを用いて実験した [7]．同一問題に対して提出されたソースコードは類似コード，異なる問題に対して提出されたソースコードは非類似コードとみなすことができる．なお，実際に提出されたソースコードによって関数分割の粒度に差がある．例えば一つの関数にまとめて処理を記述したソースコードもあれば，複数の関数に分割して記述したソースコードもある．そのため本研究では提出されたソースコードを main 関数にインライン展開し，展開後の main 関数を実験の対象とした．

BCB は大規模なコードクローンベンチマークである [19]．BCB では約 6 万のコード片に対して，クローンペア（処理内容が類似しているコードクローンの対）または非クローンペアとしてタグ付けされおり，600 万以上のクローンペアと約 26 万の非クローンペアが登録されている．ただし BCB は更新されており，この研究で使用した最新データセットのクローンペア総数は，BCB の原著で報告された数値と若干異なる．本研究では DeepSim の評価実験と条件を揃えるため，クローンペアまたは非クローンペアとタグ付けされていないコード片と，5 行未満のコード片をデータセットから除去した．除去した理由は，行数の小さいコード片はコードクローンとして保守対象になりにくく，コードクローンの評価実験において対象外にされることが多いからである [26]．この除去により，クローン

（注6）：https://about.google/

表 1　コードクローンの各タイプの個数と割合 (BCB)

タイプ	T1	T2	ST3	MT3	WT3/T4
個数（個）	17,398	3,734	12,032	53,616	5,860,619
割合（%）	0.3	0.1	0.2	0.9	98.5

ペアまたは非クローンペアにタグ付けされたコード片は，約 5 万に減少した．除去後のデータセットには，約 594 万のクローンペアと約 18 万の非クローンペアが含まれている．BCB ではコードクローンを五つのタイプに分類する [19]．BCB で用いる分類では，Roy らが提案したコードクローンの四つタイプ分類 [1] のうち，タイプ 3 とタイプ 4 のコードクローンを定量的に分類する．BCB に登録されているクローンペアのほとんどは構文的類似性が低い WT3/T4 である．コードクローン各タイプの個数と割合を表 1 に示す．

3.5　リサーチクエスチョンの調査方法

3.5.1　調査 1：精度の調査

RQ1 に回答するため，本調査では GCJ と BCB の両方のデータセットに対しての精度を調査した．本研究では再現率，適合率，F 値を精度評価指標とする．再現率とは正解集合に対して実際に正しく正解と推定された割合を指し，網羅性を示す指標である．また適合率とは正解として推定された集合に対して真に正しい割合を指し，正確性を示す指標である．一般的に再現率と適合率はトレードオフの関係にあり，一方の値が高くなるともう一方の値が低くなる．そこで，再現率と適合率の総合的な評価指標として F 値が用いられる．F 値は再現率と適合率の調和平均により求められる．

更に情報検索技術と深層学習の組み合わせで意味的な類似性の判定精度が向上することを確認するため，BCB のデータセットに対してタイプ別の調査を行った．精度は F 値を用いて比較する．なお情報検索技術のみを用いた手法では，**3.2.2** で説明した 5 種類のベクトル表現に対して，コサイン類似度 0.9 のしきい値以上になるコード片を類似していると判定した．しきい値を 0.9 とした理由は既存研究においてコードクローンの検出精度の観点から優れた値だったからである [27]．

本研究では 10 分割交差検証を用いて調査対象の類似性判定モデルの精度を推定する．10 分割交差検証ではデータセットを 10 分割し，一つをテスト用データ，残りの九つを学習用データとして検証する．この手順をテスト用データと学習用データの組み合わせを変えて 10 回繰り返す．こうして得られた結果を平均し，類似性判定モデルの精度を推定する．ただし FA-AST+GMN は 10 分割交差検証を行っておらず，学習データ:検証データ:テストデータの比を 8:1:1 の割合でデータセットを分割して評価している．またクローンペアと非クローンペアの数が 1:1 になるよう，クローンペアを減らす調整も行っている [21]．FA-AST+GMN の評価方法が本調査と異なるため，参考記録として表に掲載する．

表 2　意味表現生成の時間評価に用いた対象プロジェクト

プロジェクト	バージョン	ファイル数	LOC
ANTLR	4.7.2	233	96,043
Apache Ant	1.10.5	787	92,219
Apache Commons Lang	3.8.1	153	27,646
Apache Log4j	1.2.17	213	21,050
JUnit	4.12	195	9,317
Guava	27.0.1	573	86,542

3.5.2　調査 2：実行時間の調査

RQ2 に回答するため，本調査では GCJ のデータセットを用いて実行時間を調査した(注7)．実行時間として，類似性判定モデルの学習にかかる時間と，類似性判定モデルによる類似性の推定にかかる時間の二つを調査する．本調査では GCJ の 9 割を学習し，1 割を推定するために所要した時間を測定した(注8)．

更に意味表現生成過程の実行時間を測定した(注9)．図 1 が示すように，DeepSim は意味表現生成過程においてバイトコード解析を用いてソースコードから CFG と DFG を生成し，その後エンコーダを用いて特徴行列を生成する．一方で図 2 のように調査で用いるコード片類似性判定法の意味表現生成過程では，字句解析によるトークン分割後に前処理によって単語列を生成する．生成方法の違いによる実行時間の差を明らかにするため，表 2 に示す六つのオープンソースソフトウェアプロジェクトを用いて意味表現生成に要する時間を測定した．六つのプロジェクトは MvnRepository(注10) に含まれており，ソースコードとバイトコードの両方が公開されている．なお調査で用いるコード片類似性判定法はソースコードを解析対象とし，DeepSim はコンパイル後のバイトコードを解析対象とした．意味表現生成過程の実行時間の測定は LSI+NN の 5 回平均より求めた．

(注7)：BCB は関数単位のコードクローンのベンチマークとして作られており，ソースコードのコンパイルができない．DeepSim のバイトコード解析器ではソースコードのコンパイルが必要なため，BCB を用いた実行時間調査は比較していない．
(注8)：実験環境は Intel Xeon E5 2.7 GHz 4 コア CPU，NVIDIA Quadro 5000 GPU，32 GB メモリのワークステーション．
(注9)：実験環境は Intel Xeon E5 2.7 GHz 4 コア CPU，32 GB メモリのワークステーション．
(注10)：https://mvnrepository.com/

4. 調査結果

4.1 精度の調査結果

GCJ を用いた精度評価の結果を表 3 の左側に示す(注11)．表 3 が示すように，LSI+NN の再現率 93%，適合率 96%，F 値 0.94 と情報検索技術と深層学習の組み合わせの中で最も高い精度を示した．LDA+NN は再現率 54%，適合率 61%，F 値 0.55 と最も低い精度となった．また DeepSim の再現率 81%，適合率 71%，F 値 0.76 と比較して，LSI+NN，PV-DBoW+NN，WV-avg+NN の三つの組み合わせが再現率，適合率，F 値ともに DeepSim の結果を上回った．PV-DM+NN の再現率は DeepSim を上回ったが，適合率と F 値は DeepSim より低い値となった．LDA+NN は再現率，適合率，F 値ともに DeepSim より低い値となった．一方で FA-AST+GMN は再現率 97%，適合率 99%，F 値 0.98 と，LSI+NN よりも高い精度となった．

BCB を用いた精度評価の結果を表 3 の右側に示す(注11)．なお DeepSim の値は有効桁数 2 桁までしか掲載されていないため，本調査で調べた組み合わせとは表中の有効桁数が異なる．再現率に関しては五つの組み合わせの全てが 99.9% と高い値となった．適合率に関しては五つの組み合わせの中では PV-DBoW が 99.9% と最も高く，他の四つの手法も 99.5% 以上の値が得られた．また DeepSim は再現率 97%，適合率 98%，F 値 0.98，FA-AST+GMN は再現率 94%，適合率 96%，F 値 0.95 となった．これらと比較して，五つの組み合わせの全てが再現率，適合率，F 値いずれも DeepSim と FA-AST+GMN の結果を上回った．

更に BCB におけるクローンタイプごとの F 値の調査結果を表 4 に示す．この表より，特に WT3/T4 において，情報検索技術と深層学習の組み合わせた手法の F 値が向上していることが確認できた．これにより，情報検索技術と深層学習の組み合わせることで，構文的類似性が低いクローンも高い精度で判定できることが分かった．

RQ1 への回答

情報検索技術と深層学習を用いたコード片類似性判定法の中で LSI+NN の精度が最も高い

表 3　GCJ と BCB を用いた精度評価

	GCJ			BCB		
手法	再現率	適合率	F 値	再現率	適合率	F 値
LSI+NN	**0.93**	**0.96**	**0.94**	**0.999**	0.995	0.997
LDA+NN	0.54	0.61	0.55	**0.999**	0.995	0.997
PV-DBoW+NN	0.88	0.86	0.86	**0.999**	**0.999**	**0.999**
PV-DM+NN	0.89	0.68	0.75	**0.999**	0.998	0.998
WV-avg+NN	0.91	0.90	0.88	**0.999**	0.998	0.998
DeepSim	0.82	0.71	0.76	0.98	0.97	0.98
FA-AST+GMN	0.97	0.99	0.98	0.94	0.96	0.95

（注）：FA-AST+GMN の論文における精度評価の条件設定 [21] が本調査と異なるため，参考記録として掲載する．

表 4　BCB におけるクローンタイプごとの F 値

クローンタイプ	T1	T2	ST3	MT3	WT3/T4
LSI+NN	1.00	1.00	0.999	0.997	0.996
LDA+NN	1.00	1.00	0.999	0.997	0.997
PV-DBoW+NN	1.00	1.00	0.999	0.997	0.996
PV-DM+NN	0.999	0.999	0.997	0.996	0.992
WV-avg+NN	1.00	1.00	0.999	0.997	0.998
LSI	0.999	0.988	0.833	0.216	0.003
LDA	0.999	0.986	0.889	0.624	0.108
PV-DBoW	0.999	0.987	0.702	0.056	0.000
PV-DM	0.685	0.832	0.310	0.021	0.000
WV-avg	0.999	0.994	0.938	0.762	0.144
DeepSim	0.81	0.71	0.76	0.97	0.98
FA-AST+GMN	1.00	1.00	0.998	0.982	0.946

（注）：FA-AST+GMN の論文における精度評価の条件設定 [21] が本調査と異なるため，参考記録として掲載する．

4.2 実行時間の調査結果

GCJ を用いた実行時間の評価結果を表 5 に示す．また情報検索技術に基づくベクトル化に要する時間を明らかにするため，学習時間の内訳も表 5 に示す．表 5 より，学習時間に関して LSI+NN が 210 秒と最も高速だった．一方で PV-DM が 533 秒と最も遅く，LSI+NN と比較して約 2.6 倍の時間がかかっている．学習時間の内訳では，ベクトル化に所要する時間は LSI+NN が 0.6 秒と最も高速であり，類似性判定に所要する時間は LDA+NN が 201 秒と最も高速であった．一方でベクトル化に所要する時間は PV-DM が 318 秒と最も遅く，類似性判定に所要する時間は PV-DBoW+NN と WV-avg+NN が 216 秒と最も遅かった．推定時間に関しては LSI+NN と LDA+NN が 21 秒と最も高速だった．一方で PV-DBoW+NN, PV-DM+NN, WV-avg+NN の三つの組み合わせが 25 秒と最も遅く，LSI+NN や LDA+NN と比較して約 1.2 倍の時間がかかっている．

また DeepSim の学習時間に関しては 70,503 秒と，約 20 時間かかった(注12)．最も高速な LSI+NN と比較

（注11）：DeepSim と FA-AST+GMN の再現率，適合率，F 値は各論文より引用した．本調査と DeepSim は同じ条件設定 [7] で調査しているが，FA-AST+GMN は条件設定が異なる [21] ため参考記録として掲載する．

（注12）：DeepSim の論文に掲載されている学習時間の 13525 秒 [7] と比較し，本調査では約 5.2 倍の学習時間を要している．実験環境の性能差により学習時間に差が生じた．

表 5　GCJ を用いた実行時間評価（秒）

手法	学習時間	推定時間	学習時間内訳	
			ベクトル化	類似性判定
LSI+NN	**210 秒**	**21 秒**	**0.6 秒**	209 秒
LDA+NN	228 秒	**21 秒**	27 秒	**201 秒**
PV-DBoW+NN	224 秒	25 秒	8 秒	216 秒
PV-DM+NN	533 秒	25 秒	318 秒	215 秒
WV-avg+NN	256 秒	25 秒	40 秒	216 秒
DeepSim	70503 秒	27 秒	-	-

表 6　意味表現生成過程の実行時間（秒）

プロジェクト	DeepSim	LSI+NN
ANTLR	メモリ不足	13 秒
Apache Ant	42 秒	12 秒
Apache Commons Lang	20 秒	6 秒
Apache Log4j	15 秒	4 秒
JUnit	9 秒	4 秒
Guava	29 秒	12 秒

して約 336 倍の時間がかかっている．なお DeepSim はベクトル化を行わないため，学習時間の内訳の記載はない．また DeepSim の推定時間 27 秒と比較し，五つの組み合わせの全てが短時間で推定が完了した．

図 1，図 2 の左枠で囲われた意味表現生成過程の実行時間の調査結果を表 6 に示す．この表より，LSI+NN は DeepSim の 27〜44% の実行時間で意味表現を生成できた．更に DeepSim による ANTLR の解析時には，Java 仮想マシンのヒープ領域不足により意味表現の生成を完了できなかった．なお，Java 仮想マシンの最大ヒープサイズを初期値の 8 GB から 16 GB まで増やして再度実行したが結果は変わらなかった．

RQ2 への回答

情報検索技術と深層学習を用いたコード片類似性判定法の中で LSI+NN の実行時間が最も短い

5. 考　　察

5.1　調査結果 1:精度の比較

情報検索技術に基づくベクトル表現を用いた 5 種類の類似性判定法の中では，**4.1** の表 3 より LSI+NN が再現率，適合率，F 値ともに最も高く，次いで WV-avg+NN が高かった．その反面，LDA+NN は再現率 54%，適合率 61% と両者とも最も低い値となった．LDA はベイズ統計に基づく確率的トピック生成モデルによってコード片のベクトル表現を求めるが，プログラミング言語は自然言語と比べてトピック数が少ないため，コード片のベクトル表現を求める上で LDA は効果的でない可能性がある．また，PV-DBoW+NN，PV-DM+NN，WV-avg+NN が LSI+NN より再現率と適合率ともに低い値となった．これらの三つの類似性判定法は，機械学習を用いてベクトル表現を生成する．しかし，**3.2.2** で述べたとおり，本調査では Python ライブラリのデフォルト値を参考にしてベクトル表現を生成する際のハイパーパラメータを設定した．本調査ではハイパーパラメータを調整しなかったことが，上記の三つの類似性判定法が LSI+NN より低い精度となった原因の可能性がある．

BCB を用いた精度評価では，5 種類全ての組み合わせにおいて再現率，適合率，F 値ともに 0.99 以上と大きな差はなかった．更に BCB におけるクローンタイプごとの判定精度の比較も行った．情報検索技術と深層学習を組み合わせた手法は，構文的類似性が低いクローンも高い精度で判定できることを示した．情報検索技術を用いた手法は検出漏れが多い課題があったが，この調査結果より情報検索技術と深層学習を組み合わせることの有用性が明らかになった．

GCJ と BCB の精度結果を比較すると，BCB の方が精度が高い傾向にある．これはデータセットを構成するクローンの特徴が，精度の差につながっている．BCB に含まれる WT3/T4 クローンは，一致する行数が 50% 以下であるものの，同様の構造をしているクローンが多く含まれている [7]．一方で，GCJ はさまざまなプログラマが開発したソースコードが提出されており，さまざまな構造のコード片の類似性を求める必要がある．また，提出の速度が求められる競技プログラミングにおいて，ソースコードの理解のしやすさは二の次にされることが多く，同一問題に提出されたソースコードもプログラマによって異なる識別子名をつけられる可能性がある．したがって GCJ の方が類似性を判定することが難しく，BCB の方が GCJ に比べて F 値が高くなる傾向にある．

5.2　調査結果 2：実行時間の比較

情報検索技術に基づくベクトル表現を用いた 5 種類の類似判定法の中では，**4.2** の表 5 より学習時間にばらつきがあるのに対し，推定時間の差は少なかった．更に学習時間の内訳をみると，ベクトル化に所要する時間が学習時間の差に寄与していることが分かった．

ベクトル化に所要する時間に関しては，LSI+NN が 0.6 秒と非常に短時間であった．2 番目の PV-DBoW+NN も 8 秒であり，残りの手法は 20 秒以上であることから，LSI+NN がベクトル化の観点から非常に高速であるこ

とが分かる．これらにより，LSI+NN が最も有用性の高い手法であることが確認できた．また，付録 **2.** が示すように PV-DBoW+NN，PV-DM+NN，WV-avg+NN のベクトル生成のためのエポック数は全て 20 に固定し，ベクトル化にかかる時間を公平に比較した．しかしベクトル化のアルゴリズムによって，ベクトル化に所要する時間は大きく異なる結果となった．

LSI+NN が高速である理由として，LSI は主成分分析を用いてベクトルの次元圧縮を行うことが挙げられる．特に，主成分分析は行列計算に置き換えることで高速に計算可能なアルゴリズムが提案されており [28]，高速にベクトル化を行える．

5.3 情報検索技術と深層学習の組み合わせの比較

GCJ を用いた精度評価では LSI+NN が再現率，適合率，F 値ともに最も精度が高かった．また BCB を用いた精度評価では大きな差はないため，精度の観点から LSI+NN がコード片の類似性判定法に適しているといえる．更に学習時間と推定時間のいずれにおいても LSI+NN が最も高速であり，実行時間の観点からも LSI+NN がコード片の類似性判定法に適している．

以上の精度と実行速度の二つの観点から，LSI+NN が最もコード片の類似性判定法に適していることが判明した．したがって情報検索技術の一種である LSI と深層学習の組み合わせが，コード片類似性判定法として最も効果的な組み合わせである．

5.4 情報検索技術と深層学習を用いたコード片類似性判定法と既存手法の比較

DeepSim と比較したところ，適合率，再現率，F 値の精度の観点と，学習や推定に所要する実行速度の観点から，LSI+NN の有用性が明らかになった．また FA-AST+GMN と比較したところ，GCJ における精度評価に関して LSI+NN より高い精度となった．これは，Wang らの論文と本調査の評価方法が異なることが理由として挙げられる．Wang らは 10 分割交差検証を行っておらず，更にクローンペアの数を減らして非クローンペアと数を揃えるなどの調整を行っており，本調査と条件設定が異なるため参考記録として掲載している．

調査で用いたコード片類似性判定法が高い精度を得られた理由として，人間はコードを書く際にソースコードに処理の意味をもたせることが挙げられる．特にソフトウェアの保守性を高めるために，他人が読んでも理解しやすいソースコードを記述することが求められ，その処理内容を理解できるようにソースコードを記述する．本研究の実験により，人間がソースコードにもたせた処理内容の意味を情報検索技術に基づくベクトル表現が充分に表現できることが分かった．

DeepSim は CFG と DFG を解析した特徴行列から自己符号化器を用いてコード片のベクトル表現を求める．一方で調査で用いたコード片類似性判定法はコード片中の出現単語から，情報検索技術を用いてベクトル表現を求める．つまりベクトル表現を求めるまでの過程の違いにより，調査で用いたコード片類似性判定法は高速な学習が可能になった．特に情報検索技術の一種である LSI にて用いられる主成分分析は，行列計算に置き換えて高速に計算するアルゴリズムが提案されており，ネットワークの学習のために反復計算が必要な自己符号化器と比較して高速に学習ができる．

5.5 コード片類似性判定の実例

ここでは，BCB 上で正しく類似性判定できたクローン及び正しく判定できなかったクローンの具体例を示し，その差異を定性的に述べる．

最初に，本調査で用いた五つの類似性判定法で正しくクローンと判定した例を示す．図 4 と図 5 のコード片は，ファイルをコピーする処理を行うコードクロー

```
public static void copyFile3(File srcFile, File destFile)
    throws IOException {

  InputStream in = new FileInputStream(srcFile);
  OutputStream out = new FileOutputStream(destFile);

  byte[] buf = new byte[1024];
  int len;
  while ((len = in .read(buf)) > 0) {
    out.write(buf, 0, len);
  } in .close();
  out.close();
}
```

図 4　コード片 1:ファイルをコピーする処理 (1)

```
static void copy(String src, String dest)
    throws IOException {

  File ifp = new File(src);
  File ofp = new File(dest);
  if (ifp.exists() == false) {
    throw new IOException(
      "file '" + src + "' does not exist");
  }
  FileInputStream fis = new FileInputStream(ifp);
  FileOutputStream fos = new FileOutputStream(ofp);
  byte[] b = new byte[1024];
  int readBytes;
  while ((readBytes = fis.read(b)) > 0)
    fos.write(b, 0, readBytes);
  fis.close();
  fos.close();
}
```

図 5　コード片 2：ファイルをコピーする処理 (2)

```
public void prepare() throws IOException {

  this.extractDirectory = TempFileFactory.get()
    .createTempDirectory("pdash-compressed-wd", ".tmp");
  File srcZip = getTargetZipFile();
  InputStream in = new FileInputStream(srcZip);
  if (isPdbk(srcZip))
    in = new XorInputStream( in , PDBK_XOR_BITS);
  boolean sawEntry = false;
  try {
    ZipInputStream zipIn =
      new ZipInputStream(new BufferedInputStream( in ));
    ZipEntry e;
    while ((e = zipIn.getNextEntry()) != null) {
      sawEntry = true;
      if (!e.isDirectory()) {
        String filename = e.getName();
        File f = new File(extractDirectory, filename);
        if (filename.indexOf('/') != -1)
          f.getParentFile().mkdirs();
        FileUtils.copyFile(zipIn, f);
        f.setLastModified(e.getTime());
      }
    }
  } finally {
    FileUtils.safelyClose( in );
  }
  if (!sawEntry) throw new ZipException("Not a valid ZIP
file: " + srcZip);
}
```

図 6　コード片 3：圧縮ファイルを展開する処理

```
public static String downloadWebpage3(String address)
    throws ClientProtocolException, IOException {

  HttpClient client = HttpClientBuilder.create().build();
  HttpGet request = new HttpGet(address);
  HttpResponse response = client.execute(request);
  BufferedReader br = new BufferedReader(
    new InputStreamReader(
      response.getEntity().getContent()));
  String line;
  String page = "";
  while ((line = br.readLine()) != null) {
    page += line + "¥n";
  }
  br.close();
  return page;
}
```

図 7　コード片 4：WEB ページを取得する処理 (1)

```
public boolean open() {

  try {
    URL url = new URL(resource);
    conn = url.openConnection();
    in = new BufferedReader(
      new InputStreamReader(conn.getInputStream()));
  } catch (MalformedURLException e) {
    System.out.println(
      "Uable to connect URL:" + resource);
    return false;
  } catch (IOException e) {
    System.out.println(
      "IOExeption when connecting to URL" + resource);
    return false;
  }
  return true;
}
```

図 8　コード片 5：WEB ページを取得する処理 (2)

ンである．“File"，“Input"，“Output"，“Stream"，“src"，“dest"など共通して現れる文字が多く，情報検索技術を用いた手法でクローンと判定しやすい．

次に偽陽性の例を示す．図 4 は単にファイルをコピーする処理に対し，図 6 は圧縮ファイルを展開する処理を行う．これらは BCB 上で非クローンペアとして登録されているが，本調査で用いた五つの類似性判定法はクローンと判定した．“File"，“Input"，“Output"，“Stream"，“src"などが共通して出現するため，類似していると誤って判定された可能性がある．

最後に偽陰性の例を示す．図 7 と図 8 は Web ページを取得する処理を行うクローンとして BCB に登録されている．しかし，本調査で用いた五つの類似性判定法はクローンでないと判定した．これらには共通して現れる文字が少ないため，情報検索技術を用いた手法では類似性を判定できない可能性がある．

以上のように，本調査で用いた類似性判定法は情報検索技術を用いているため，構文的に類似性が低いコードクローンの類似性を判定できる．一方で，共通した文字列に影響を受けて誤ってコードクローンと判定したり，共通した文字列が少ないことにより誤ってコードクローンでないと判定したりする欠点がある．

5.6　妥当性の脅威

本研究では交差検証によりモデルが過学習していないことを確認しているが，学習とテストを同じデータセット内にて行っている．学習したデータセット以外でも高い精度が得られる，より汎化性能が高い類似性判定モデルの作成は課題である．本研究では，学習とテストで異なるデータセットを使用した精度調査も行った．BCB で学習し GCJ でテストした結果は，再現率 0.95，適合率 0.21，F 値 0.34 となった．また GCJ で学習し BCB でテストした結果は，再現率 0.63，適合率 0.97，F 値 0.76 となった．これらは表 3 の結果より F 値が低い．つまり学習したデータセット以外では精度が低くなることが分かる．これは，データセット内のコード片には偏りが存在することが理由として挙げられる．実際に BCB は 10 種類の機能のコード片しか含まれておらず [19]，また GCJ のデータセットには 12 種類の問題しか含まれていない．しかし全ての機能をもつコード片のデータセットの作成は困難である．学習したデータセット以外でも高い精度が得られる，より汎化性能が高いモデルの作成は課題である．

またハイパーパラメータに関する懸念もある．**4.1**

にて PV-DBoW+NN と PV-DM+NN の適合率は 86%，68% と，PV-DM の方が適合率が低い．ハイパーパラメータのエポック数を PV-DM は PV-DBoW より増やす場合が多いが，本調査では 20 に揃えた．そのためコード片の意味を充分に学習できず，PV-DM の適合率が低下した可能性がある．上記以外の手法においても，ハイパーパラメータを調整することにより精度が向上する可能性がある．

更には，本研究で調査したベクトル表現と深層学習モデル以外に，より有用性の高い組み合わせがある可能性がある．本研究では，gensim ライブラリに採用されているベクトル表現から調査対象を選択した．本研究の調査対象外のベクトル表現も調査対象に加えることで，更に有用性の高い組み合わせを見つけられる可能性がある．

本調査では FA-AST+GMN の精度評価結果を論文より引用したが，調査実験の条件設定が異なる [21] ため参考記録として掲載した．FA-AST+GMN らは 10 分割交差検証を行っておらず，更にクローンペアの数を減らして非クローンペアと数を揃えるなどの調整を行っており，本調査と条件設定が異なる．今後，本調査と条件設定を揃えて FA-AST+GMN の精度を評価することで，また異なる結果が得られる可能性がある．

6. 関 連 研 究

深層学習を用いたコード片類似性判定法として White らの手法と Wei らの手法がある．White らは RtvNN というコード片類似性判定法を提案した [17]．RtvNN はトークン列と抽象構文木の情報から再帰型ニューラルネットワーク [29] を用いてベクトル表現を求め，ユークリッド距離を用いてベクトル表現の類似性を判定する．また，Wei らは CDLH というコード片類似性判定法を提案した [30]．CDLH は字句と構文の情報から抽象構文木ベースの LSTM (Long Short-Term Memory) を用いてハッシュコードを求め，高速に類似性を学習する．コード片の類似性判定にはハッシュコードのハミング距離を用いる．RtvNN と CDLH は，深層学習を用いてベクトル表現を求めるが，類似性判定には深層学習モデルを使用しない．更に BCB を対象に上記の二つの手法と DeepSim の精度評価を比較した結果，DeepSim より再現率と適合率ともに低かった [7]．

我々は情報検索技術に基づく様々なベクトル表現をソースコードに適用し，コードクローン検出における各ベクトル表現の特徴を調査した [13], [14]．これらのベクトル表現を用いて BCB [19] に対してコードクローン検出を行い，その再現率を比較することで，よりコード片の意味を表すベクトル表現を調査した．また，コード片の類似性を判定する距離尺度として，コサイン類似度と Word Mover's Distance [31] の二つの距離尺度を比較した．一方で本研究では，コード片の類似性を判定するために深層学習モデルを用いている．本研究は，情報検索技術と深層学習の効果的な組み合わせを調査した点で先行研究と異なる．

我々の研究グループでは深層学習を用いたソースコード分類手法を比較調査した [32]．入力された一つのソースコードを機能に応じて分類するタスクを設定し，そのタスクを高い精度で解く深層学習モデルを明らかにした．一方，本研究では入力された二つのソースコードが類似しているか否かの判定を深層学習モデルのタスクとして設定した．深層学習モデルが解くタスクの観点で，先行研究と本研究は異なる．

また同じく我々の研究グループでは，深層学習モデルとして回帰モデルを用いたコードクローン検出法を提案した [33]．深層学習を用いた既存のコードクローン検出法に対し，深層学習モデルを回帰モデルに変更し汎化性能が高くなることを明らかにした．深層学習モデルへの入力は，既存のコードクローン検出法に則りソースコードメトリクスを用いている．一方，本研究では情報検索技術に基づくベクトル表現を深層学習モデルの入力として用いる．深層学習モデルへの入力の観点で，先行研究と本研究は異なる．

7. む す び

本研究では，情報検索技術に基づくベクトル表現と深層学習の効果的な組み合わせについて，判定精度と実行速度の観点から調査した．調査実験では，5 種類のベクトル表現間での比較と，深層学習を用いた既存手法との比較を行った．調査の結果，情報検索技術に基づくベクトル表現として LSI を用いた手法が，最も高精度かつ高速であることが分かった．また既存手法の DeepSim と比較して，LSI を用いた手法が再現率，適合率，F 値ともに高く，学習時間も約 350 倍高速であることが確認できた．既存手法の FA-AST+GMN と比較して，GCJ を用いた精度評価において LSI を用いた手法より FA-AST+GMN の方が再現率，適合率，F 値ともに高かった．ただし調査実験の条件設定が異なるため FA-AST+GMN の精度は参考記録として扱う．

今後の課題として，以下の 3 点が挙げられる．

- 学習したデータセット以外でも高い精度が得られる，より汎化性能が高いモデルの作成
- 本研究で用いたベクトル表現以外の調査
- FA-AST+GMN の精度を，本研究の調査と条件設定を合わせて評価

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

文　献

[1] C.K. Roy, J.R. Cordy, and R. Koschke, “Comparison and evaluation of code clone detection techniques and tools: A qualitative approach,” Sci. Comput. Program., vol.74, no.7, pp.470–495, 2009.

[2] R. Holmes and G.C. Murphy, “Using structural context to recommend source code examples,” Proc. ICSME, pp.117–125, 2005.

[3] I. Keivanloo, J. Rilling, and Y. Zou, “Spotting working code examples,” Proc. ICSE, pp.664–675, 2014.

[4] 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，” 信学論（D），vol.J91-D, no.6, pp.1465–1481, June 2008.

[5] D. Rattan, R. Bhatia, and M. Singh, “Software clone detection: A systematic review,” Inf. Softw. Technol., vol.55, no.7, pp.1165–1199, 2013.

[6] K. Inoue, Y. Sasaki, P. Xia, and Y. Manabe, “Where does this code come from and where does it go? — integrated code history tracker for open source systems,” Proc. ICSE, pp.331–341, 2012.

[7] G. Zhao and J. Huang, “Deepsim: Deep learning code functional similarity,” Proc. ESEC/FSE, pp.141–151, 2018.

[8] T. Kamiya, S. Kusumoto, and K. Inoue, “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.

[9] Z. Li, S. Lu, S. Myagmar, and Y. Zhou, “Cp-miner: finding copy-paste and related bugs in large-scale software code,” IEEE Trans. Softw. Eng., vol.32, no.3, pp.176–192, 2006.

[10] H. Sajnani, V. Saini, J. Svajlenko, C.K. Roy, and C.V. Lopes, “Sourcerercc: Scaling code clone detection to big-code,” Proc. ICSE, pp.1157–1168, 2016.

[11] 横井一輝，崔　恩瀞，吉田則裕，井上克郎，“情報検索技術に基づく細粒度ブロッククローン検出，” コンピュータ ソフトウェア，vol.35, no.4, pp.16–36, 2018.

[12] R. Baeza-Yates and B. Ribeiro-Neto, Modern information retrieval: The concepts and technology behind search, Addison-Wesley, 2011.

[13] 横井一輝，崔　恩瀞，吉田則裕，井上克郎，“コード片のベクトル表現に基づく大規模コードクローン集合の特徴調査，” ソフトウェアエンジニアリングシンポジウム 2018 論文集，pp.192–199, 2018.

[14] K. Yokoi, E. Choi, N. Yoshida, and K. Inoue, “Investigating vector-based detection of code clones using bigclonebench,” Proc. APSEC, pp.699–700, 2018.

[15] Q. Le and T. Mikolov, “Distributed representations of sentences and documents,” Proc. ICML, pp.1188–1196, 2014.

[16] 藤原裕士，崔　恩瀞，吉田則裕，井上克郎，“順伝播型ニューラルネットワークを用いた類似コードブロック検索の試み，” ソフトウェアエンジニアリングシンポジウム 2018 論文集，pp.24–33, 2018.

[17] M. White, M. Tufano, C. Vendome, and D. Poshyvanyk, “Deep learning code fragments for code clone detection,” Proc. ASE, pp.87–98, 2016.

[18] W. Hua, Y. Sui, Y. Wan, G. Liu, and G. Xu, “Fcca: Hybrid code representation for functional clone detection using attention networks,” IEEE Trans. Rel., vol.70, no.1, pp.304–318, 2021.

[19] 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. ICSME, pp.476–480, 2014.

[20] G.E. Hinton and R.R. Salakhutdinov, “Reducing the dimensionality of data with neural networks,” Science, vol.313, no.5786, pp.504–507, 2006.

[21] W. Wang, G. Li, B. Ma, X. Xia, and Z. Jin, “Detecting code clones with graph neural network and flow-augmented abstract syntax tree,” Proc. SANER, pp.261–271, 2020.

[22] E. Enslen, E. Hill, L. Pollock, and K. Vijay-Shanker, “Mining source code to automatically split identifiers for software analysis,” Proc. MSR, pp.71–80, 2009.

[23] R. Řehůřek and P. Sojka, “Software framework for topic modelling with large corpora,” Proc. LREC, pp.45–50, 2010.

[24] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient estimation of word representations in vector space,” arXiv preprint, arXiv:1301.3781, 2013.

[25] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, “Distributed representations of words and phrases and their compositionality,” Proc. NIPS, pp.3111–3119, 2013.

[26] E. Choi, N. Yoshida, T. Ishio, K. Inoue, and T. Sano, “Extracting code clones for refactoring using combinations of clone metrics,” Proc. IWSC, pp.7–13, 2011.

[27] 山中裕樹，崔　恩瀞，吉田則裕，井上克郎，“情報検索技術に基づく高速な関数クローン検出，” 情処学論，vol.55, no.10, pp.2245–2255, 2014.

[28] M. Brand, “Fast low-rank modifications of the thin singular value decomposition,” Linear Algebra Appl., vol.415, no.1, pp.20–30, 2006.

[29] R. Socher, C.C.-Y. Lin, A.Y. Ng, and C.D. Manning, “Parsing natural scenes and natural language with recursive neural networks,” Proc. ICML, pp.129–136, 2011.

[30] H. Wei and M. Li, “Supervised deep features for software functional clone detection by exploiting lexical and syntactical information in source code,” Proc. IJCAI, pp.3034–3040, 2017.

[31] M.J. Kusner, Y. Sun, N.I. Kolkin, and K.Q. Weinberger, “From word embeddings to document distances,” Proc. ICML, vol.37, pp.957–966, 2015.

[32] 藤原裕士，崔　恩瀞，吉田則裕，井上克郎，“深層学習を用いたソースコード分類手法の比較調査，” 信学論（D），vol.J104-D, no.8, pp.622–635, Aug. 2021.

[33] 藤原裕士，森　彰，井上克郎，“回帰モデルを用いたコードクローン検出手法の提案と汎化性能の評価，” 信学論（D），vol.J104-D, no.9, pp.678–689, Sept. 2021.

付　　録

1. ベクトル表現とハイパーパラメータ

ベクトル表現	ハイパーパラメータ
LSI	トピック数:200
LDA	トピック数:100
PV-DBoW	ベクトルサイズ:300，ウィンドウサイズ:15，エポック数:20
PV-DM	ベクトルサイズ:300，ウィンドウサイズ:5，エポック数:20
WV-avg	ベクトルサイズ:300，ウィンドウサイズ:5，エポック数:20

2. 手法ごとのハイパーパラメータ設定

手法	ハイパーパラメータ
LSI+NN	トピック数:200, レイヤーサイズ:200-100, エポック数:4, 初期学習率:0.001, L2 正則化の λ :0.00003
LDA+NN	トピック数:100, レイヤーサイズ:100-100, エポック数:4, 初期学習率:0.001, L2 正則化の λ :0.00003
PV-DBoW+NN	ベクトルサイズ:300，ウィンドウサイズ:15，エポック数（PV-DBoW）:20, レイヤーサイズ:300-100, エポック数:4, 初期学習率:0.001, L2 正則化の λ :0.00003
PV-DM+NN	ベクトルサイズ:300, ウィンドウサイズ:5, エポック数（PV-DM）:20, レイヤーサイズ:300-100, エポック数:4, 初期学習率:0.001, L2 正則化の λ :0.00003
WV-avg+NN	ベクトルサイズ:300, ウィンドウサイズ:5, エポック数（Word2Vec）:20, レイヤーサイズ:300-100, エポック数:4, 初期学習率:0.001, L2 正則化の λ :0.00003
DeepSim	レイヤーサイズ:88-6, (128x6-256-64)-128-32, エポック数:4, 初期学習率:0.001, L2 正則化の λ :0.00003, ドロップアウト:0.75
FA-AST+GMN	レイヤーサイズ:100, ベクトル次元数:100, エポック数:4, 初期学習率:0.001, バッチサイズ:32

（2022 年 5 月 27 日受付，10 月 5 日再受付，12 月 1 日早期公開）

横井　一輝

2017 大阪大学基礎工学部情報科学科卒．2019 大阪大学大学院情報科学研究科博士前期課程了．同年株式会社 NTT データに入社．現在，大阪大学大学院情報科学研究科博士後期課程に社会人学生として在学．コードクローン分析手法に関する研究に従事．

崔　　恩瀞　（正員）

2015 大阪大学大学院情報科学研究科博士後期課程了．同年同大学大学院国際公共政策研究科助教．2016 奈良先端科学技術大学院大学情報科学研究科助教．2018 より同大学先端科学技術研究科助教（改組による）．2018 より京都工芸繊維大学情報工学・人間科学系助教．博士（情報科学）．コードクローン管理やリファクタリング支援手法に関する研究に従事．

吉田　則裕　（正員）

2009 大阪大学大学院情報科学研究科博士後期課程了．同年日本学術振興会特別研究員 (PD)．2010 奈良先端科学技術大学院大学情報科学研究科助教．2014 名古屋大学大学院情報科学研究科附属組込みシステム研究センター准教授．2022 より立命館大学情報理工学部教授．博士（情報科学）．コードクローン分析手法やリファクタリング支援手法，ソフトウェアテストに関する研究に従事．

松下　　誠

1998 大阪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 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助手．2005 同専攻助教授．2007 同専攻准教授．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．

井上　克郎　（正員：フェロー）

1984 大阪大学大学院基礎工学研究科博士後期課程了（工博）．同年大阪大学基礎工学部情報工学科助手．1984～1986 ハワイ大学マノア校コンピュータサイエンス学科助教授．1991 大阪大学基礎工学部助教授．1995 同学部教授．2002 同学情報科学研究科教授．2022 より南山大学理工学部ソフトウェア工学科教授．ソフトウェア工学，特にコードクローンやコード検索などのプログラム分析や再利用技術の研究に従事．