修　士　論　文

題　　目　　API呼び出し情報に基づいたメソッド名の推薦手法の提案

主任指導教員　　水野　修　教授

指導教員　　崔 恩瀞　助教

京都工芸繊維大学大学院 工芸科学研究科

情報工学専攻

学生番号　　19622049

氏　　名　　若林　奎人

令和3年2月10日提出

様式 2 号

学位論文内容の要旨（和文）

令和 3 年 2 月 10 日

京都工芸繊維大学大学院
工芸科学研究科長　殿

工芸科学研究科　　情報工学専攻
平成 31 年入学
学生番号　　19622049
氏　　名　　若林　奎人　㊞

（主任指導教員　　水野　修　㊞　）

本学学位規則第 4 条に基づき、下記のとおり学位論文内容の要旨を提出いたします。

1.　論文題目

API 呼び出し情報に基づいたメソッド名の 推薦手法の提案

2.　論文内容の要旨（400 字程度）

ソフトウェア開発において，識別子名の質はプログラムの可読性に大きな影響を及ぼすため重要であるが，適切な命名を行うには開発に関する知識や経験が必要とされる．そこで、識別子の命名の支援を行うため，プログラムの内容から識別子名を推薦する様々な手法が提案されている．中でもメソッド名の推薦に関する先行研究では，メソッド定義のソースコードの構文木に基づきメソッド名を推薦する手法や、API の呼び出し情報を利用し名前を推薦する手法が存在する．また，ソースコードの構文木と API の呼び出し情報を利用した手法が存在するが，API を呼び出さないメソッドに対しては手法を適用できず命名精度は向上していない．

そこで本研究では，ナレッジグラフを利用し API 呼び出しの有無に関わらずメソッド名を推薦可能な手法を提案する．

提案手法の有効性を評価するため，既存のメソッドに対して名前の推薦を行い，命名精度を検証した．その結果，ナレッジグラフは API 呼び出しの無いメソッドにおいて推薦精度を約 1.5%向上可能であることがわかった．

A method name recommendation approach based on API calls information

2021 19622049 *WAKABAYASHI Keito*

Abstract

Names of identifier are important because they affect the readability of program under the environment of developing software. However, naming identifiers requires various knowledge and experience corresponding the target software. Therefore, there are several methods for supporting developers in naming identifiers. Among them, there is a method to suggest names based on the source code syntax tree and the API call graph, but it cannot be applied to source code without API calls.

Therefore, we propose a method based on knowledge graph which can be applied to source code with/without API calls. We confirm effectiveness of the proposed method by applying the method to the naming task in the existing software. The experiments show that our method can improve naming accuracy by 1.5% against the methods without API calls.

目　次

1. 緒言

ソフトウェア開発において，プログラムを理解することは重要であるが [1]，コストがかかる作業である．開発工程のうち，ソフトウェアの保守にかかるコストは全体の約 7 割を占めており [2]，保守作業にはプログラムを理解する必要がある．また，この作業は保守作業の大部分を占めるといわれる [3].

プログラムの理解には，ソースコード中に存在するクラス名，メソッド名，変数名といった識別子名を手がかりとする [4]．これは，識別子の名前がその識別子の動作やプログラム内での役割を示唆するよう命名されているためである．よって，開発者がプログラム中の識別子に対し，その要素の動作や役割を表すよう適切に命名することにより，保守作業者がプログラムを理解するために必要なコストを削減できる．

しかし，識別子に対して適切な命名を行うことは難易度の高い作業である．なぜなら，この作業を行うためには，命名対象の識別子がプログラム中でどのような役割を担っているかや、他の識別子がどのように命名されているかといった、プログラム全体に関する理解が必要となるためである．また、プログラミング言語に特有の命名規則に関する知識も必要となる．

これらの背景から，開発者が識別子に命名する作業を支援するための様々な手法が提案されている．その中には，プログラム内で重要な役割を持つ動詞と目的語の関係をソースコードから抽出し辞書を作成する手法や [5]，メソッドの処理内容に対してどのような単語がメソッド名に使われるかという規則を抽出した手法 [6]，畳み込みニューラルネットワークを利用した手法 [7]，ソースコードを AST として表現し構造的な情報を利用する手法 [8][9]，コールグラフにより API の呼び出し情報を考慮した手法 [10]，また，ソースコードの構造的な情報と API の呼び出し情報の両方を利用する手法 [11] が存在する．

特に，米内らの手法 [11] では，code2vec[8] や [9] のようなソースコードの構造的な情報を利用し高い推薦精度を達成している手法と，コールグラフ [12] から得た API の呼び出し情報を組み合わせることで，更に精度を向上させることを達成している．しかし，API を呼び出していないメソッドに対しては手法を適用できず，精度は向上していないという問題点がある．

そこで本研究では，ナレッジグラフによるAPIの呼び出しの有無に関わらず適用できる手法を提案し，メソッド名の推薦精度の向上を目指す．

本研究ではデータセットとしてGitHubに公開されているスター数の多いJavaを利用しているプロジェクトのソースコードから，学習用に3,004,536個，検証用に410,699個，評価用に411,751個のメソッドを得た．提案手法の有効性を示すため，以下の観点から研究設問について調査した．

- **RQ1) ナレッジグラフはメソッド名の推薦精度の向上に有効か**
  正解メソッド名と推薦メソッド名が完全に等しい場合に推薦に成功したとして，評価用データにおける推薦に成功したメソッド名の割合 (成功率) を測定した．ソースコードの構文情報のみを利用したAlonらの手法 [8] の成功率を約1.0%上回る結果となったが，構文情報とAPIの呼び出し情報を組み合わせた米内らの手法 [11] とほぼ同じ成功率となった．メソッド名の構成単語を基準とした評価においてもAlonらの手法 [8] の評価を約1.0%上回ったが，再現率で米内らの手法 [11] を約0.7%上回り，適合率では約0.3%下回った．
- **RQ2) メソッド内のAPI呼び出しの有無に関係なく推薦精度の向上が可能か**
  メソッド内のAPI呼び出しの有無を基準にデータセットを分割し，推薦成功率を測定した．Alonらの手法 [8] にはメソッド内のAPI呼び出しの有無に関わらず成功率が約1.0%上回ったが，米内らの手法 [11] にはAPI呼び出しの無い場合のみ約1.1%上回り，API呼び出しを含む場合は約0.3%下回った．

RQ1の結果から，ナレッジグラフに基づいたAPIの呼び出し情報は，メソッド名の推薦精度の向上に貢献可能であるが，コールグラフから得られるAPI呼び出し情報との間に差は無いと考えられる．

また，RQ2の結果から，提案手法はメソッド内のAPI呼び出しの有無に関わらず推薦精度を向上させることは不可能であり，API呼び出しの無い場合のみ推薦精度の向上が可能であることがわかった．

これらの結果の考察として，ナレッジグラフに追加するノードに制限を設けることで推薦精度の向上の可能性が考えられた．また，ナレッジグラフはあらゆる言語において作成可能であるため，提案手法の一般化の可能性も考えられる．

本稿の構成は以下の通りである．まず2章で本研究における事前知識と先行研究について言及し，3章で提案手法について説明する．4章で研究設問と実験に使用するデータセットについて説明し，5章で各研究設問に対する実験結果と考察を示す．6章で各研究設問対する回答と提案手法の課題，妥当性の脅威について言及し，9章で結言とする．

2. 関連研究

メソッド名の推薦手法には，分散表現を利用した手法が存在する [7][8][9]．分散表現とは，単語を実数値のベクトルで表す手法 [13] であり，自然言語処理の分野で活発に利用されている [14] が，本研究や先行研究のようにプログラミング言語にも利用されている．

本研究は API の呼び出し関係を表すナレッジグラフの分散表現を生成する手法と，ソースコードの AST(Abstract Syntax Tree) から分散表現を生成する手法を組み合わせた手法である．よって本章では，ナレッジグラフの分散表現の生成に利用した DeepWalk[15] と，AST の分散表現の生成に利用した code2vec[8]，code2vec にコールグラフに基づいた API 呼び出し情報を組み合わせた米内らの手法 [11] について説明する．

2.1 DeepWalk

DeepWalk[15] とは，ナレッジグラフから分散表現を生成する手法の 1 つである．ナレッジグラフとは，人や場所などあらゆる事実をグラフの頂点として表現し，関係性のある頂点を辺で繋いだものである．Freebase[16]，WordNet[17]，GeneOntology[18] などの知識ベースや，Google(注 1)[19] などの検索エンジンに利用されている．図 2.1 にナレッジグラフの例を示す．グラフの各頂点を分散表現で表すことが可能であり，グラフ上での各頂点の位置関係が保持されることが特徴である．

DeepWalk における処理を以下に述べる．

2.1.1 ランダムウォーク

DeepWalk では，まずグラフの各頂点と繋がっている他の頂点をランダムウォーク [20] により一定数選択し，頂点の列を作る．図 2.1 の例で 3 つノードを選択する場合，

●, ▲, ◆

■, ◆, ●

(注 1): https://www.google.com/

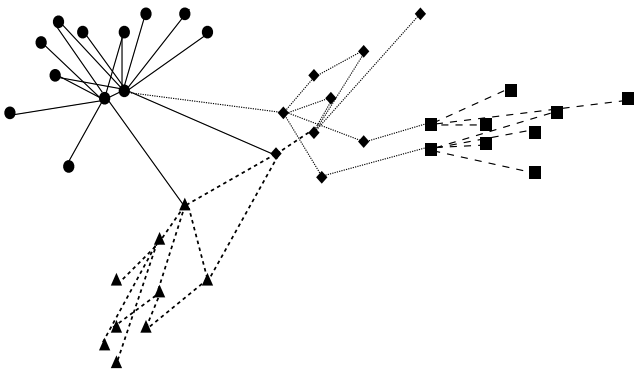

図 **2.1**　ナレッジグラフの例

などが挙げられる．この列を文章として扱うことで，自然言語処理の分野で利用されている従来の分散表現の手法を適用することができる．

2.1.2 SkipGram

SkipGram[21] は単語の分散表現手法の 1 つである．文章中の 1 単語を中心とし，その周辺に現れる単語を予測するように学習することで単語の分散表現を得る手法である．

DeepWalk では，1 つ目の処理で作成したの頂点の列を文章とし、SkipGram を適用する．これにより，グラフの各頂点の分散表現を生成する．

2.2 code2vec

code2vec[8] とは，ソースコードの AST の分散表現に基づいたメソッド名推薦手法である．AST とは，ソースコードの構文情報を表現した木構造である．図 2.2 は Java で書かれたメソッドのソースコードから AST を得る例である．この図で非終端ノードはソースコードの構文を表し，終端ノードはソースコード上に出現する識別子を表す．code2vec は入力としてメソッド定義のソースコードを受け取り，出力としてそのメソッドの名前を推薦する．この手法は以下の 3 つの段階で構成される．

2.2.1 AST の作成

図 2.2 のように，ソースコードの構文や識別子に基づいて AST を作成する．その後，任意の 2 つの終端ノードの間の非終端ノードを連結させたものを AST パスとして作成する．図 2.2 の例の場合，AST パスは

$$(Type \hat{} MetodDacl\_Name)$$

$$(Name \hat{} MetodDacl\_Block\_Return)$$

などが挙げられる．作成された AST パスと，その AST パスが繋ぐ 2 つの終端ノード x_s, x_t を<x_s, $path$, x_t> として並べたものをパスコンテキストと表現する．図 2.2 の例では，

```
public String getID(){
  return ID;
}
```

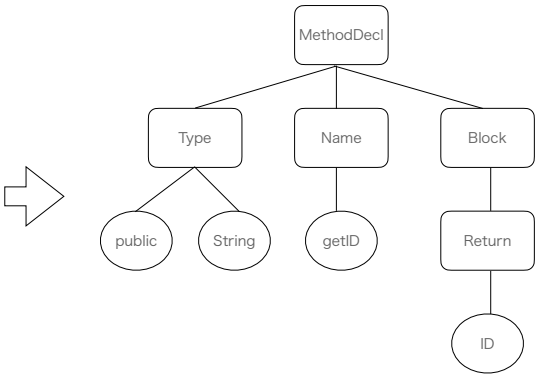

図 2.2　AST の例

<String, (Type^MetodDacl_Name), getID>

<getID, (Name^MetodDacl_Block_Return), ID>

などが挙げられる．

2.2.2 メソッドの分散表現の作成

1 つのメソッド定義のソースコードから複数のパスコンテキストを作成し，それらの分散表現を 1 つに集約したものをメソッド定義の分散表現として生成する．また、各パスコンテキスト<x_s, *path*, x_t>の分散表現は，終端ノード x_s,x_t の分散表現と、AST パス *path* の分散表現を集約することで生成する．

メソッド名の候補は作成された分散表現と類似度の高い学習済みの分散表現を持つメソッドが推薦される．

2.3 米内らの手法 [11]

米内らの手法 [11] では，code2vec のようなソースコードの構文情報に基づいた手法に，コールグラフに基づいた API の呼び出し情報を組み合わせることで推薦精度の向上を達成している．しかし，ソースコードに API 呼び出しを含まない場合においては手法を適用できず，精度は向上していない．

そこで本研究では，ソースコード中の API 呼び出しの有無に関わらず適用可能な手法を提案する．具体的には，API の呼び出し関係に加えて，ソースコードの類似度に基づいて作成したナレッジグラフの分散表現を利用する．これにより，メソッドが API 呼び出しを含まない場合でも，ソースコードが類似したメソッドがグラフ内に存在すれば新たなノードとしてグラフに結合し，分散表現を生成することができる．このナレッジグラフの分散表現を，API の呼び出し情報を考慮した分散表現として code2vec と組み合わせることにより，推薦精度の向上を目指す．

3. 提案手法

本章では提案手法である，ソースコードの構文情報と API 呼び出し情報を組み合わせたメソッド名推薦手法について述べる．図 3.1 に提案手法の概要を示す．提案手法は code2vec のモデルをベースとし，code2vec が生成する終端ノード x_s, x_t の分散表現を DeepWalk が生成した分散表現に置き換えるという拡張を行っている．また，提案手法はナレッジグラフの分散表現の生成とメソッドの分散表現の生成の 2 つのフェーズで構成されている．

以下では提案手法の各フェーズについて述べる．

3.1 ナレッジグラフの分散表現の生成

最初のフェーズでは，学習用データのソースコードから API の呼び出し関係を表すナレッジグラフを作成し，このグラフに DeepWalk を適用することで，API 呼び出し関係の情報を含んだ分散表現を作成する．

ソースコードからナレッジグラフを作成する例を図 3.2 に，DeepWalk による処理の概要を図 3.3 に示す．分散表現の作成までの手順を以下に述べる．

1. ソースコードから AST を作成する．
2. AST の情報を基に，API 呼び出しを行っているメソッドと呼び出されている API を辺で繋ぎナレッジグラフに追加する．
3. API 呼び出しを行っていないメソッドの場合，定義ソースコードの類似度が高いメソッドがナレッジグラフに存在すれば追加する．
4. 作成されたナレッジグラフに DeepWalk を適用し，各 API の分散表現を作成する．

手順 3 におけるソースコードの類似度は tf-idf を基に計算され，閾値を超えたメソッドのみナレッジグラフに追加する．

tf-idf は情報検索において，自然言語で書かれた文書の要約や類似性の判定などに利用されている．tf-idf の値は tf 値 (関数中の単語の出現頻度) と idf 値 (ソースコード全体における単語の希少さ) の積で与えられる．各指標の計算式を以下に示す．

$$\mathrm{tf}(\mathrm{x},\mathrm{f}) = \frac{\text{関数 f 内での単語 x の出現回数}}{\text{関数 f 内の全ての単語の出現回数の和}}$$

$$\mathrm{idf}(\mathrm{x}) = \log \frac{\text{関数の合計数}}{\text{単語 x が出現する関数の数}}$$

$$\mathrm{tfidf}(\mathrm{x},\mathrm{f}) = \mathrm{tf}(\mathrm{x},\mathrm{f}) * \mathrm{idf}(\mathrm{x})$$

これにより，API 呼び出しを含まないメソッドの場合でも分散表現を生成し，提案手法が適用可能となる．

図 3.2 で，*person()* は *getAge()* を呼び出しているので辺でつながっている．また，*person()* と *member()* は呼び出し関係にないが，ソースコードが類似しているため結合される．比較手法 [11] で利用されているコールグラフでは呼び出し関係にあるメソッドのみ組み込まれているが，本研究におけるナレッジグラフでは呼び出し関係に無いメソッドも組み込まれる．

また，Kartsaklis らの研究 [22] より，閾値は 0.5 から 0.75 の間が適切とされている．本研究では検証用データにおける精度が最も高くなった 0.5 に設定した．

3.2 メソッドの分散表現の生成

2 つ目のフェーズでは，1 つ目のフェーズで作成した分散表現を code2vec の処理に組み込み，メソッド名の推薦を行う．code2vec による処理の概要を図 3.4 に示す．推薦までの手順を以下に示す．

1. ソースコードの AST を作成し，パスコンテキストを抽出する．
2. パスコンテキストの各要素 x_s, *path*, x_t の分散表現を作成する．
3. パスコンテキスト<x_s, *path*, x_t>が API 呼び出しを表す場合，呼び出されている API x_s または x_t の分散表現を DeepWalk が作成した分散表現に置き換える．
4. パスコンテキストの各要素 x_s, *path*, x_t の分散表現を集約し，パスコンテキストの分散表現を作成する．
5. パスコンテキストの分散表現を集約し，メソッドの分散表現を作成する．

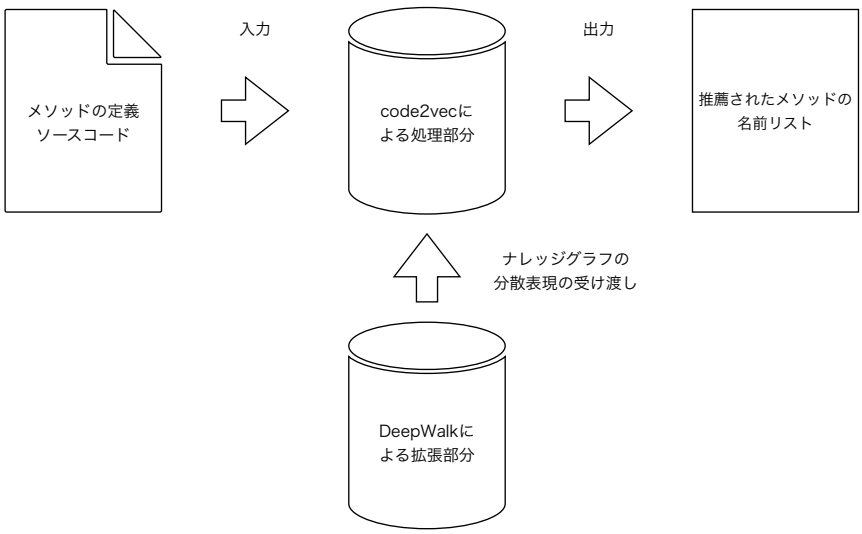

図 3.1　提案手法の概要

```
public String person(){
  age = getAge();
}
```

```
public String user(){
  age = getAge();
  id = getID();
}
```

```
public String member(String age){
  this.age = age;
}
```

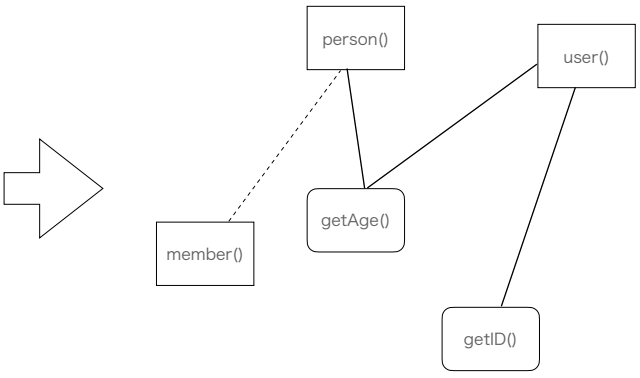

図 3.2　ソースコードからナレッジグラフを作成する例

6. 作成された分散表現と候補メソッドの分散表現の類似度の高いものを推薦する．

手順 3 が提案手法による拡張部分であり，これにより API の呼び出し関係を考慮した推薦が可能となる．

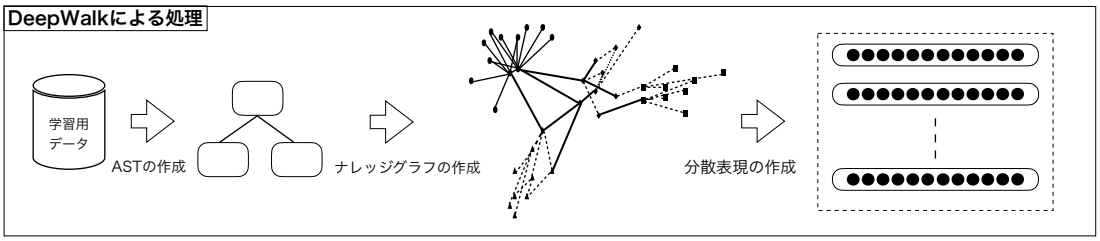

図 3.3　DeepWalk による処理の概要

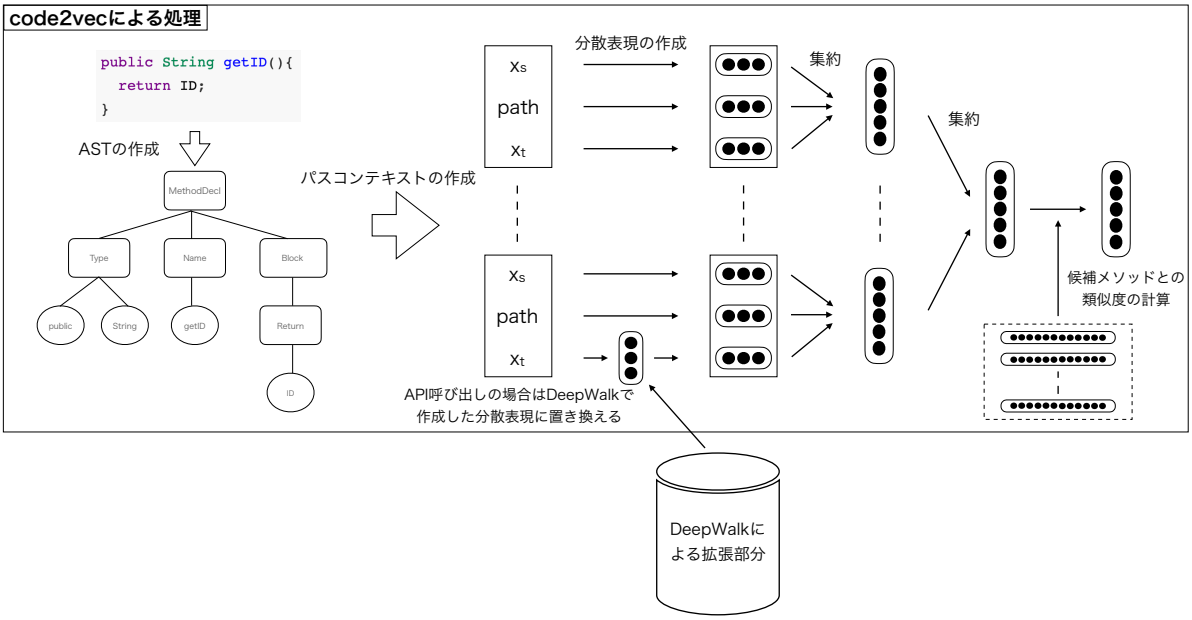

図 3.4　code2vec による処理の概要

4. 実験準備

本研究では，提案手法の有効性を検証するために 2 つの研究設問について調査する．

- **RQ1) ナレッジグラフはメソッド名の推薦精度の向上に有効か**
  メソッド名推薦におけるナレッジグラフの有効性を検証するため，推薦精度を先行研究 [8][11] と比較する．
- **RQ2) メソッド内の API 呼び出しの有無に関係なく推薦精度の向上が可能か**
  メソッド内で呼び出している API の有無に基づいて評価用データを分割し，各評価用データにおける推薦精度を先行研究と比較する．

4.1 評価指標

メソッド名の推薦精度を測定するため，以下の評価指標を用いる．

4.1.1 完全一致による評価

推薦したメソッド名と正解のメソッド名が完全に等しい場合のみ推薦が成功したものとし，評価用データの内推薦に成功したデータの割合を測定する．

4.1.2 部分一致による評価

メソッド名を構成する単語に基づいて評価する．評価指標には Allamanis らの研究 [23] で使用された適合率 (Precision)，再現率 (Recall)，F1 値を用いる．この評価手法では，推薦したメソッド名と正解のメソッド名が完全に等しくない場合でも，メソッド名を構成する単語で一致しているものがあれば一定の評価を得ることができる．完全に等しいメソッド名を推薦できない場合でも，メソッド名の構成単語を適切な命名のヒントとして利用できるためである．

適合率 (Precision)，再現率 (Recall)，F1 値は，true positive，false positive，false negative を用いて

$$Precision = \frac{true\ positive}{true\ positive + false\ positive}$$

$$Recall = \frac{true\ positive}{true\ positive + false\ negative}$$

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall}$$

と計算される．

推薦されたメソッド名が *getLinse* であり，正解のメソッド名が *countBlankLines* であった場合，*lines* が true positive，*get* が false positive，*count, blank* が false negative となる．よって，適合率は 0.5，再現率は 0.33，F1 値は 0.4 となる．

4.2 データセット

提案手法の有効性を検証するために，GitHub(注 1)[24] に公開されている Java を使用しているプロジェクトのソースコードを利用する．プロジェクトはスター数が上位 1000 件のものを選択する．スター数はそのプロジェクトに興味を持つ開発者の数を表しているため，スター数が多いプロジェクトは多くの開発者により保守されており，適切なメソッド名が与えられているものと考える．また，このデータセットは比較手法や code2seq で使われているものであり，[25](注 2)で公開されている．

取得した Java プロジェクトのうち 800 件を訓練用，100 件を検証用，100 件を評価用として分割する．その後，プロジェクトのソースコードからメソッド名とその定義ソースコードを抽出し，各サブセット用のデータとする．各プロジェクトからメソッドを抽出した結果，学習用に 3,004,536 個，検証用に 410,699 個，評価用に 411,751 個のメソッドが得られた．

本研究では上記の学習データにより提案手法のモデルを学習し，検証用データにおける精度が最大になるようパラメータチューニングを行った．

(注 1): https://github.com
(注 2): https://code2seq.org

5. 実験

5.1 RQ1) ナレッジグラフはメソッド名の推薦精度の向上に有効か

5.1.1 実験手法

4.2 節で説明した GitHub から取得したデータセットを用いてメソッド名の完全一致による評価と部分一致による評価を行う．完全一致による評価は推薦するメソッドの数を 1〜10 個に場合分けして行う．このとき，推薦したメソッドの中に 1 つでも正解メソッド名が含まれていれば推薦に成功したものとし，評価用データにおける成功率を測定する．部分一致による評価にはメソッド名を 1 つだけ推薦し，4.2.1 で説明した Precision，Recall，F1 値を測定する．

5.1.2 実験結果

メソッド名の完全一致による評価結果を表 5.1 に示す．1 列目に推薦するメソッド数，2 列目以降には各手法における推薦の成功率を示している．この表より，提案手法の成功率は code2vec の値を約 1.0%上回ったが，比較手法とはほぼ同じ値となったことがわかる．

次に，メソッド名の部分一致による評価の結果を表 5.2 に示す．1 列目に評価指標の種類，2 列目以降に各手法の評価結果を示している．この表から，提案手法はどの評価指標においても code2vec の値を約 1.0%上回っているが，比較手法 [11] については再現率は約 0.7%，F1 値は約 0.2%上回り，適合率は比較手法の値を約 0.3%下回ったことがわかる．

5.1.3 考察

表 5.1 から，ナレッジグラフを基にした API の呼び出し情報はメソッド名の推薦精度の向上に貢献可能であることがわかる．しかし，比較手法の値と大きな差は無かったため，正解率においては，ナレッジグラフから生成できる API 呼び出し情報と，コールグラフから生成できる情報に差は無いと言える．

また，表 5.2 から，提案手法は比較手法より適合率は低く，再現率は高いことがわかる．これにより，提案手法は比較手法より多くの種類のメソッド名構成単語を推

表 5.1　メソッド名の推薦数毎の成功率

推薦メソッド数	code2vec	比較手法 [11]	提案手法
1	0.23264	0.24204	0.24239
2	0.25968	0.27166	0.27165
3	0.27247	0.28160	0.28492
4	0.28016	0.29259	0.29310
5	0.28599	0.29842	0.29989
6	0.29078	0.30282	0.30246
7	0.29449	0.30679	0.30845
8	0.29785	0.30996	0.31177
9	0.30056	0.31260	0.31253
10	0.30301	0.31483	0.31688

表 5.2　メソッド名の部分一致による評価

	code2vec	比較手法 [11]	提案手法
Precision	0.37015	0.38413	0.38112
Recall	0.26697	0.27459	0.28123
F1	0.31021	0.32025	0.32217

薦していると考えらえる．これはナレッジグラフを作成する際，呼び出し関係にないメソッドをグラフに組み込んでいるため，比較手法に比べてより多種類のメソッド名が推薦候補に含まれていることが原因と考えられる．

5.2 RQ2) メソッド内のAPI呼び出しの有無に関わらず推薦精度の向上が可能か

5.2.1 実験手法

メソッドの定義ソースコード内でのAPI呼び出しの有無に基づいて評価用データセットを分割し，それぞれのデータセットにおけるメソッド名推薦の成功率を測定する．

また，推薦するメソッド名の数は10個とし，10個の中に正解メソッドが含まれていれば推薦に成功したものとする．

5.2.2 実験結果

メソッド中のAPI呼び出しの有無によって分割したデータセットにおけるメソッド名の推薦成功率を表5.3に示す．この表から，提案手法はいずれのデータセットにおいてもcode2vecの値を約1.0%上回っていることがわかる．また，比較手法については，API呼び出しの無いメソッドへの推薦成功率は約1.5%上回ったが，API呼び出しを含むメソッドの成功率は約0.3%下回ったことがわかる．

5.2.3 考察

表5.3から，API呼び出しの無いメソッドにおいては提案手法の成功率が最も高いことがわかる．これはナレッジグラフを作成する際，ソースコードの類似度に基づいてAPI呼び出しのないメソッドの場合でもグラフに組み込んだことが原因と考えられる．

しかし，API呼び出しを含むメソッドの推薦成功率は比較手法より低くなっている．提案手法ではメソッドの定義ソースコードから生成される分散表現の一部を，ナレッジグラフから生成した分散表現に置き換える処理を行っているが，提案手法では比較手法より多くの種類のグラフ分散表現が生成され，その結果より多くの

表 5.3　API 呼び出しの有無による成功率

API 呼び出し	code2vec	比較手法 [11]	提案手法
無	0.44259	0.44016	0.45561
有	0.25041	0.26761	0.26460

ソースコードから生成した分散表現が置き換えられる．これにより，ソースコードの情報がメソッド名の推薦に貢献できず，成功率が低くなった可能性がある．

6. 考察

6.1 RQ への回答

5.1 節の完全一致による評価では，提案手法は code2vec の推薦成功率を上回ったが，コールグラフを利用した比較手法と成功率はほぼ同じであった．また，部分一致による評価においては，適合率は比較手法の方が高く，再現率は提案手法の方が高い結果となったが，これらはトレードオフの関係であり，F1 値に大きな差は生まれていない．

以上の結果から，RQ1 に対して，ナレッジグラフから得られる API の呼び出し情報はメソッド名の推薦精度の向上に有効であるが，コールグラフから得られる API の呼び出し情報との間に差は無いという回答を得る．

次に，5.2 節の API 呼び出しの有無による評価では，API 呼び出しの無い場合は提案手法の成功率が高く，API 呼び出しを含む場合は比較手法の方が成功率が高い結果となった．これにより，RQ2 に対し，提案手法はメソッド中の API 呼び出しの有無に関わらず推薦精度を向上させることは不可能であり，API 呼び出しの無い場合のみコールグラフを利用した比較手法より推薦精度の向上が可能であるという回答を得る．

6.2 課題

メソッド名の推薦は開発者の命名支援を目的としており，正解のメソッド名の推薦が不可能でも，ヒントとなる単語を提示することが重要となる [26]．しかし，提案手法では適合率が低いことから，命名のヒントとならない単語を多く提示していることがわかる．本研究ではナレッジグラフを作成する際，呼び出し関係にあるメソッドや API を全て辺で繋いだが，1 つのノードに接続できるノード数に制限を設けることにより，ランダムウォークで選択できるノードが減り，余分な情報を省くことが可能と考えられる．

また，本研究で使用した評価用データのうち約 70%が API 呼び出しを含むメソッドであった．データセットは GitHub 上で多くの開発者が興味を持つプロジェクトのソースコードを利用しているため，実際に開発者がメソッドに命名する場合も API

呼び出しを含んでいる場合が多いと考えられる．よって，code2vec に組み込むグラフ分散表現の数を制限することで，代わりに考慮できるソースコードの構文情報を増やし、API 呼び出しを含んだメソッドに対する推薦精度を向上させる必要がある．

6.3 妥当性への脅威

6.3.1 構成概念妥当性

本研究の評価指標にはデータセットが不均衡であった場合に備えて，メソッド名の完全一致による評価 (正解率) だけでなく部分一致による評価 (適合率，再現率，F1 値) を利用した．また，これらの評価指標は先行研究 [8][9][11] においても利用されており，妥当であると考えられる．

しかし，RQ2 における評価には完全一致による評価のみ行っているため，API 呼び出しの有無で分割したそれぞれのデータセットに不均衡が存在した場合には妥当性への脅威となり得る．

6.3.2 外的妥当性

本研究のデータセットには GitHub 上のスター数が多い Java プロジェクトを選択している．スター数の多いプロジェクトは多くの開発者により保守され適切なメソッド名が付けられており，妥当であると考える．

また，提案手法の一部である DeepWlak はあらゆる自然言語やプログラミング言語等あらゆるものに対し利用可能であり，code2vec は Java，C 言語，C#，Python に対し利用できるため，本研究結果の一般化も考えられる．

6.3.3 内的妥当性

提案手法におけるナレッジグラフを作成する際，ソースコードの類似度を基準にしているが，ソースコード中に適切な命名のされていない識別子が存在した場合，関係性の低いはずのノードがグラフに繋がれる可能性がある．これに対処するため，パラメータの調整により接続するノードの数やランダムウォークで選択するノード数を制限できるが，本研究では時間的な制約から調整できなかった．本研究では利

用していないが，コード要約ツール [27] によりソースコード中のキーワードを抽出することで，より正確なナレッジグラフを作成できると考えられる．

また，類似度は tf-idf により測定されるが，CCFinder[28] のようなコードクローン [29] 検出ツールを利用することにより，識別子でなくソースコードの構文的な情報に基づいた類似度を測定できる可能性がある．識別子の情報を考慮する場合においても，キャメルケースで書かれた識別子をスネークケースに置き換えることでより正確に類似度を測定できる可能性がある．更に，ユーザ定義の識別子である場合は同じ単語に置き換え，正規化することで結果に影響すると考えられる．

また，提案手法における code2vec と DeepWalk による処理部分は実装が公開されているが，DeepWalk が作成した分散表現を code2vec に組み込む部分は著者が実装したため，不具合が残っている可能性がある．

7. 結言

本研究では，メソッド名の命名支援という目的に対し，ナレッジグラフを利用したメソッド名の推薦手法を提案した．また，評価実験を通して，先行研究の課題であったAPI呼び出しを行っていないメソッドへの名前推薦の精度を測定した．

RQ1での評価結果から，ナレッジグラフに基づいたAPI呼び出し情報を利用することで，推薦精度の向上が可能であることが明らかとなった．しかし，推薦の成功率において，先行研究で利用されている，コールグラフから得られるAPI呼び出し情報との間に差は無いことがわかった．更に，推薦の部分一致による評価から，ナレッジグラフはより多くの種類のメソッド構成単語を推薦できると考えられる．これは，本来呼び出し関係に無いメソッドをグラフに組み込んでいることが原因であると考えられる．

次に，RQ2での評価結果から，提案手法はAPI呼び出しの無いメソッドに対してのみ推薦精度が向上することがわかった．これは呼び出し関係に無いメソッドをナレッジグラフに組み込んだことが原因と考えられる．一方で，API呼び出しを行っているメソッドに対する評価は先行研究を下回る結果となった．これは，提案手法ではソースコードの構文情報をAPIの呼び出し情報に置き換える処理を行っているため，APIの呼び出し数が増えると構文情報を利用できなくなることが原因と考えられる．

これらの結果から，提案手法はAPI呼び出しの無いメソッドへの名前推薦精度は向上可能であるが，API呼び出しを含むメソッドに対しては，利用するAPI呼び出し情報にの数に制限を設けるなどにより，構文情報とのバランスを取る必要があると考えられる．

謝辞

本研究を行うにあたり，研究課題の設定や研究に対する姿勢，本報告書の作成に至るまで，全ての面で丁寧なご指導を頂きました，本学情報工学・人間科学系水野修教授並びに崔恩瀞助教に厚く御礼申し上げます．また，本報告書執筆にあたり貴重な助言を多数頂きました本学設計工学専攻 西浦生成先輩，近藤将成先輩，洪浚通

先輩をはじめとするソフトウェア工学研究室の皆さん，学生生活を通じて著者の支えとなった家族や友人に深く感謝致します.

参考文献

[1] A.V. Mayrhauser and A.M. Vans, “Program comprehension during software maintenance and evolution,” Computer, vol.28, no.8, pp.44–55, 1995.

[2] B.P. Lientz, E.B. Swanson, and G.E. Tompkin, “Characteristics of application software maintenance,” Commun, vol.21, no.6, pp.466–471, 1978.

[3] G.C. Murphy, M. Kersten, M.P. Robillard, and D. Cubranic, “The emergent structure of development tasks,” In Proceedings of the 19th European conference on Object-Oriented Programming, vol.5, pp.33–48, 2005.

[4] A.D. Lucia, M.D. Penta, R. Oliveto, A. Panichella, and S. Panichella, “Using ir methods for labeling source code artifacts: Is it worthwhile?,” In Proceedings of the 20th IEEE International Conference on Program Comprehension (ICPC), pp.193–202, 2012.

[5] 鹿島悠，早瀬康裕，真鍋雄貴，松下誠，井上克郎，メソッドに用いられる動詞-目的語関係を収録した辞書構築手法の提案，情報処理学会，2010.

[6] 柏原由紀，鬼塚勇弥，石尾隆，早瀬康裕，山本哲男，井上克郎，“相関ルールマイニングを用いたメソッドの命名方法の分析,” ソフトウェア工学の基礎 (FOSE2013)，pp.25–35，Nov. 2013.

[7] A. Miltiadis, P. Hao, and S. Charles, “A convolutional attention network for extreme summarization of source code,” In Proceedings of the 33rd International Conference on Machine Learning (ICML), pp.2091–2100, 2016.

[8] U. Alon, M. Zilberstein, O. Levy, and E. Yahav, “code2vec: learning distributed representations of code,” In Proceedings of the 46th ACM on Programming Languages, pp.1–29, 2019.

[9] U. Alon, M. Zilberstein, O. Levy, and E. Yahav, code2seq: Generating sequences from structured representations of code, In Proceedings of the 7th International Conference on Learning Representations, 2019.

[10] H. Yonai, Y. Hayase, and H. Kitagawa, “Mercem: Method name recommendation

based on call graph embedding," In Proceedings of the 26th Asia-Pacific Software Engineering Conference (APSEC), pp.134–141, 2019.

[11] 米内裕史，早瀬康裕，北川博之，"ソースコード構文木とコールグラフの統合的な埋め込みに基づくメソッド名の推定," Technical Report 10，筑波大学, 筑波大学, 筑波大学，feb 2020.

[12] W. Zhang and B. Ryder, "Constructing accurate application call graphs for java to model library callbacks," In Proceedings of the 6th IEEE International Workshop on Source Code Analysis and Manipulation, pp.63–74, 2006.

[13] YoshuaBengio, R. ejeanDucharme, PascalVincent, and C. Jauvin, "A neural probabilistic language model," machine learning research, pp.1137–1155, Feb. 2003.

[14] G.G. Chowdhury, Natural language processing, asis&t, Jan. 2005.

[15] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," In Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining, pp.701–710, 2014.

[16] B.N. Son and P.H. Hoang, "New approach for extracting information on freebase based on user's preferences," In Proceedings of the 16th International Conference on Information Integration and Web-based Applications & Services, pp.54–58, 2014.

[17] G. Miller, WordNet: An Electronic Lexical Database, Communications of the ACM, 1998.

[18] G. Zhou, J. Wang, X. Zhang, and G. Yu, "Deepgoa: Predicting gene ontology annotations of proteins via graph convolutional network," In Proceedings oh the 13th IEEE International Conference on Bioinformatics and Biomedicine (BIBM), pp.1836–1841, 2019.

[19] Google, Google, （オンライン）, 入手先〈https://www.google.com/〉（参照 2021-2-7）.

[20] L. Grady, "Random walks for image segmentation," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol.28, no.11, pp.1768–1783, 2006.

[21] T. Mikolov, I. SutskeverChen, G.S. Corrado, and J. Dean, “Distributed representations of words and phrases and their compositionality,” In Proceedings of 26th the NIPS, vol.2, pp.3111–3119, 2013.

[22] D. Kartsaklis, M.T. Pilehvar, and N. Collier, “Mapping text to knowledge graph entities using multi-sense lstms,” In Proceedings of the 2nd EMNLP, pp.1959–1970, 2018.

[23] M. Allamanis, D. Tarlow, A.D. Gordon, , and Y. Wei, “Bimodal modelling of source code and natural language,” In Proceedings of the 32nd ICML, pp.2123–2132, 2015.

[24] GitHub, GitHub, GitHub（オンライン), 入手先〈https://github.com〉（参照 2021-2-7）.

[25] U. Alon, S. Brody, O. Levy, and E. Yahav, tech-srl/code2seq: Code for the model, GitHub（オンライン), 入手先〈https: //github.com/tech-srl/code2seq〉（参照 2021-2-7）.

[26] K. Liu, D. Kim, T.F. Bissyandé, T. Kim, K. Kim, A. Koyuncu, S. Kim, and Y. Le Traon, “Learning to spot and refactor inconsistent method names,” In Proceedings of the 46th IEEE/ACM International Conference on Software Engineering (ICSE), pp.1–12, 2019.

[27] X. Hu, G. Li, X. Xia, D. Lo, S. Lu, and Z. Jin, “Summarizing source code with transferred api knowledge,” In Proceedings of the 27th International Joint Conference on Artificial Intelligence, IJCAI-18, pp.2269–2275, International Joint Conferences on Artificial Intelligence Organization, July 2018.

[28] 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.

[29] R.K. Saha, M. Asaduzzaman, M.F. Zibran, C.K. Roy, and K.A. Schneider, “Evaluating code clone genealogies at release level: An empirical study,” In Proceedings of the 10th IEEE Working Conference on Source Code Analysis and Manipulation, pp.87–96, 2010.