ディープコードコメント生成*。

Xing Hu1 , Ge Li1, Xin Xia2, David Lo3, Zhi Jin1
[1]北京大学高信頼ソフトウェア技術重点実験室、MoE、北京、中国
[2]オーストラリア、モナシュ大学情報技術学部
[3]シンガポール経営大学情報システム学部（シンガポール
[1]{huxing0101,lige,zhijin}@pku.edu.cn, [2]xin.xia@monash.edu, [3] davidlo@smu.edu.sg

ABSTRACT

ソフトウェアのメンテナンスにおいて、コードコメントは開発者がプログラムを理解するのに役立ち、ソースコードを読んだりナビゲートしたりするのに費やす時間を短縮します。しかし、残念なことに、これらのコメントは、ソフトウェアプロジェクトにおいて、しばしば不一致、欠落、または古くなっています。このため、開発者はソースコードから機能を推測しなければならない。この論文では、Javaメソッドのコードコメントを自動生成するDeepComという新しいアプローチを提案する。生成されたコメントは、開発者がJavaメソッドの機能を理解するのを助けることを目的としている。DeepComは自然言語処理（NLP）技術を適用し、大規模なコードコーパスから学習し、学習した特徴からコメントを生成する。ディープニューラルネットワークを用い、Javaメソッドの構造情報を解析することで、より適切なコメントを生成する。GitHubの9,714のオープンソースプロジェクトから構築した大規模なJavaコーパスで実験を行う。また、実験結果を機械翻訳メトリクスの上で評価する。実験の結果、我々の手法DeepComは最先端技術をかなりのマージンで上回ることが実証された。

CCS コンセプツ

· ソフトウェアとそのエンジニアリング→ドキュメンテーション→計算方法論→ニューラルネット。

キー ワード

プログラム理解、コメント生成、ディープラーニング

ACMリファレンスフォーマット。
Xing Hu1 , Ge Li1, Xin Xia2, David Lo3, Zhi Jin1.2018.Deep Code Comment Generation（ディープコードコメント生成）。*Proceedings of IEEE/ACM International Conference on Program Comprehension, Gothenburg, Sweden, May 27 - May 28, 2018 (ICPC'18)* に収録されています。ACM, New York, NY, USA, 11 pages. https://doi.org/10.475/123_4

*本研究は、中国国家基礎研究プログラム（973プログラム）（助成番号2015CB352201）および中国国家自然科学基金（助成番号61232015および61620106007）による支援を受けている。Zhi Jin と Ge Li が対応する著者である。

ICPC'18、*2018年5月27日～5月28日*、*スウェーデン・ヨーテボリ*

ISB 123-4567-24-567/08/06…$15.00
https://doi.org/10.475/123_4

1 イントロダクション

ソフトウェアの開発・保守において、開発者はプログラム理解活動に約59%の時間を費やしている[45]。開発者は自然言語によるコメントからコードの意味を理解することができるため、優れたコメントはプログラム理解にとって重要であることが、これまでの研究で示されています[35]。しかし，プロジェクトのスケジュールやその他の理由により，多くのプロジェクトでコードコメントが不一致であったり，欠落していたり，古かったりすることがよくあります．コード・コメントの自動生成は，開発者がコメントを書く時間を短縮するだけでなく，ソースコードの理解にも役立つ．

プログラミング言語として最も普及しているJavaのメソッド[24, 35]やクラス[25]に対して，過去10年間に多くのコメント生成の手法が提案されている[1]．その手法は、手作業で作成された[25]ものを用いるものから、情報検索
(IR) [14, 15]がある．Morenoら[25]は、Javaクラスのコメントを合成するために、ヒューリスティックとステレオタイプを定義した。これらのヒューリスティックとステレオタイプは、コメントに含まれる情報を選択するために使用される。Haiducら[14, 15]は、クラスとメソッドの要約を生成するために、IRのアプローチを適用した。Vector Space Model (VSM) や Latent Semantic Indexing (LSI) などのIRアプローチは、通常、類似のコード・スニペットからコメントを検索します。しかし、これらの手法には主に2つの限界がある。1つ目は、識別子やメソッドの命名が不十分な場合、類似コードスニペットを特定するための正確なキーワードを抽出することができない点です。第二に、類似のコード・スニペットが検索可能かどうか、またそのスニペットがどの程度類似しているかに依存する。

近年，大規模なソースコードに対する確率モデルの構築に関心が集まっている．Hindle ら [17] はソフトウェアの自然さに着目し、コードが確率的モデルでモデル化できることを示した。その後、いくつかの研究により、異なるソフトウェアタスクのための様々な確率的モデルが開発された[12, 23, 40, 41]。コード要約に適用する場合、IRベースのアプローチとは異なり、既存の確率モデルベースのアプローチは通常、キーワードから合成するのではなく、コードから直接コメントを生成する。Iyerら[19]はCODE-NNと呼ばれる注目ベースのRecurrent Neural Network（RNN）モデルを提案している。これは、自然言語コメントのための言語モデルを構築し、コメント中の単語を注意コンポーネントによって直接個々のコードトークンと整合させる。CODE-NNはStack Overflowから抽出されたソースコードスニペットを用いて、コードコメントを推奨する。実験結果は、コード要約における確率モデルの有効性を実証している。これらの研究は、自然言語記述とソースコードの両方における曖昧さを確率的にモデル化し、解決するための原理的な方法を提供するものである。

[1] https://www.tiobe.com/tiobe-index/

本論文では、深層学習技術の利点を活用するために、Java言語の機能単位であるJavaメソッドの説明文を生成する新しいアプローチDeepComを提案する。DeepComは、ニューラル機械翻訳（NMT）の進歩に基づいている。NMTは、ある言語（例えば、中国語）から別の言語（例えば、英語）への自動翻訳を目的とし、これまでは、自然言語コーパスで大きな成功を収めていることが示されている[6, 37]。

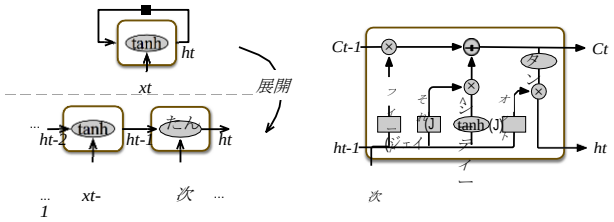
直感的には、コメント生成は、プログラム言語で書かれたソースコードを自然言語で書かれたテキストに翻訳する必要があるNMT問題の一種と考えることができる。

コメントに対してのみ言語モデルを構築するCODE-NNと比較して、NMTモデルはソースコードとコメントの両方に対して言語モデルを構築する。コメント中の単語は、コードトークンのセマンティクスに関わるRNNの隠れ状態と一致する。DeepComは、大規模Javaコーパスから抽出した特徴量（識別子名、書式、意味、構文特徴など）から自動的に学習し、コメントを生成する。従来の機械翻訳とは異なり、我々のタスクは以下の点で挑戦的である。

(1) **ソースコードは構造化されている。**自然言語のテキストが弱く構造化されているのとは対照的に、プログラミング言語は形式言語であり、その中で書かれたソースコードは一義的で構造化されている[3]。NMTで使われる多くの確率モデルはシーケンスベースのモデルであり、構造化されたコード解析に適応させる必要がある。ソースコードの豊富で曖昧さのない構造情報を利用して、既存のNMT技術の有効性をいかに高めるかが主な課題であり、チャンスでもある。

(2) **語彙のこと。**自然言語(NL)コーパスでは通常NMTに使用されるコーパスでは、通常、語彙は最も一般的な単語、例えば3万語に限定され、語彙外の単語は未知の単語として扱われ、しばしばUNKとマークされる。このようなNLコーパスでは、支配的な語彙以外の単語は非常にまれであるため、有効である。コードコーパスの場合、語彙はキーワード、演算子、識別子から構成される。開発者は様々な新しい識別子を定義するのが普通であり、そのため識別子が増殖する傾向がある。今回のデータセットでは、数字や文字列を一般的なトークンであるNUMやSTRに置き換えた結果、234,146個のユニークトークンが得られています。確率モデル構築のためのコードベースでは、語彙のない識別子が多く存在する可能性がある。表1に示すように、このデータセットには234,055個の識別子が含まれている。最も一般的な30,000トークンをコードボキャブラリーとすると、約85%の識別子がUNKとみなされることになる。さらに、ソースコード中の約30%のトークンがUNKである。Hellendoorn and De- vanbu [16]は、ソースコードがこのような語彙を使用するのは不合理であることを実証している。

これらの問題に対処するため、DeepComはシーケンスベースの言語モデルをカスタマイズして、Javaメソッドの構造とセマンティクスをキャプチャする抽象構文木(AST)を解析します。ASTは、DeepComに入力される前にシーケンスに変換されます。一般に、前置トラバーサルや後置トラバーサルなどの古典的な探索手法によって生成されたシーケンスから木を復元することはできないとされている。ASTの構造をより良く表現し、配列を曖昧にしないために、我々はASTを走査するための新しい構造ベース走査法（SBT）を提案する。SBTを用いると、



(a) 標準的なRNNモデルとその時間ステップを経たアンフォールドアーキテクチャ (b) LSTMユニット

与えられたノードの下のサブツリーが一対の括弧に含まれる。

図1：基本的なRNNとLSTMの説明図

括弧はASTの構造を表しており、SBTを用いて生成したシーケンスから曖昧さなく木を復元することができる。

さらに、語彙の課題に対処するため、未知のトークンを表現する新しい方法を提案する。ASTシーケンス中のトークンには、我々の仕事では、終端ノード、非終端ノード、および括弧が含まれる。未知トークンはASTの終端トークンに由来する。我々は未知トークンを普遍的な特殊UNKトークンの代わりに、その型に置き換える。

DeepComは、ASTの文節から単語単位でコメントを生成する。我々は、GitHubの9,714のJavaプロジェクトで構成されるJavaデータセットでDeepComを訓練し、評価する。実験結果は、DeepComが情報量の多いコメントを生成できることを示している。さらに、Iyerら[19]による最先端のアプローチを含む多くのベースラインと比較した場合、DeepComが最高のパフォーマンスを達成することが示されている。

本論文の主な貢献は以下の通りである。

- 我々は、コードコメント生成タスクを機械翻訳タスクとして定式化する。
- ソースコードから抽出された構造情報を処理し、Javaメソッドに対するコメントを生成するために、シーケンスベースモデルをカスタマイズする。特に、新しいAST探索手法（構造ベース探索）と、語彙のないトークンをうまく処理するためのドメイン特有な手法を提案する。

論文の構成本論文の残りの部分は以下のように構成されている。セクション II では言語モデルと NMT に関する背景資料を示す。セクション III では，DeepCom の詳細について述べる．セクション IV とセクション V では、実験のセットアップと結果を示します。セクション VI では、DeepCom の長所と、有効性に対する脅威を説明する。セクション VII では、関連する作業を調査する。最後に、セクション VIII で本論文の結論を述べ、今後の方向性について言及する。

2 背景

2.1 言語モデル

私たちの研究は、自然言語処理分野における機械翻訳問題から着想を得ている。我々は大規模なソースコードコーパスから学習した言語モデルを利用する。言語モデルは学習された特徴量からコードコメントを生成する。言語モデルは単語の並びに対する確率的な分布を学習する。この言語モデルは様々な問題（例えば、機械翻訳[6]、音声認識[9]、質問応答[46]）で非常によく機能している。

x = *x1*、*x2*、...、*xn* という並び（例：文）に対して、Lan-of-Line は、そのような並びを表す。

ージモデルは、その確率を推定することを目的としている。

その確率は

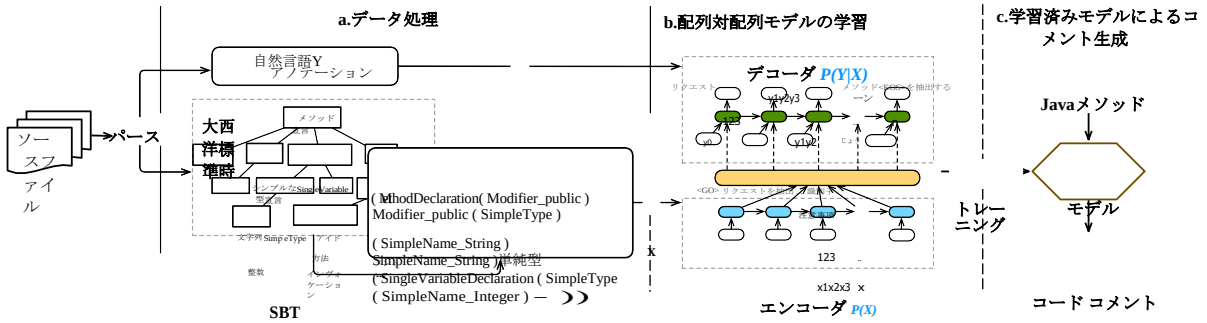


図2：DeepComの全体的なフレームワーク。

はその各トークンを介して計算される。つまり

$$P(x) = P(x1)P(x2|x1) \cdots P(xn|x1 \ldots xn\text{-}1) \quad (1)$$

本論文では、Long Short-Term Memory (LSTM) と呼ばれる深層ニューラルネットワークに基づく言語モデルを採用する[18]。LSTMは、最先端のRNNの一つである。LSTMは長期的な依存関係を学習することができるため、一般的なRNNよりも性能が優れている。これは、長い依存関係を持つソースコード（例えば、クラスがそのimport文から遠く離れた場所で使用されるなど）に使用するのに適したモデルである。RNNとLSTMの詳細を図1に示す。

*2.1.1 リカレントニューラルネットワーク。*RNNは、その鎖のような性質から、シーケンスやリストと密接に関係している。原理的には、過去の入力の全履歴から各出力にマッピングすることができる。各時間ステップtにおいて、RNNのユニットは、現在のステップの入力だけでなく、その前の時間ステップ$t1$によって 出力された隠れ状態も受け取る。図1（a）に示すように、時間ステップtの隠れ状態は、入力ベクトルxtとその前の隠れ状態ht_1に応じて、$ht = tanh\ W\,xt + Uht_{1} + b$ここで、$W$、$U$、bは学習中に更新される学習パラメータ、tanhは活性化関数：$tanh\ z = \frac{ez}{e\text{-}zez + e^{(-z)}}$ ...と更新される。

標準的なRNNモデルの顕著な欠点は、バックプロパゲーション中にグラフが爆発したり消えたりすることである。これらの現象は、シーケンスに長い依存関係が存在する場合によく現れる。この問題に対処するため、いくつかの研究者は、長期的な依存関係を保持するいくつかのバリエーションを提案した。本論文では、LSTMとGRU(Gated Recurrent Unit)を採用する。本論文では、多くの自然言語処理タスクで成功を収めているLSTMを採用する[6, 37]。

*2.1.2 長短期記憶（Long Short-Term Memory）。*LSTMは、通常のRNNがデータの長期依存性を学習しにくいという問題を解決するために、*メモリセル*と呼ばれる構造を導入している。LSTMは、隠れ状態から選択的に情報を「忘れる」ように訓練されているため、より重要な情報を取り込む余地がある[18]。LSTMでは、メモリセルから前の情報を読み出し、新しい情報を書き込むタイミングと方法を制御するゲーティング機構を導入している。リカレントユニットのメモリセルベクトルは長期依存性を保持する。このように、LSTMはバニラRNNよりも効果的に長期依存性を処理することができる。LSTMは、意味論的な課題を解決するために広く使われており、納得のいく性能を達成している。このような利点から、ソースコードやコメントのモデルを構築するためにLSTMを利用することが動機づけられます。図

1(b)は典型的なLSTMユニットを示しており、LSTMの詳細については、[10, 18]を参照してください。

2.2 ニューラル機械翻訳

NMT[44]は自動トランスレーションのためのエンドツーエンドの学習アプローチである。NMTは深層学習ベースのアプローチであり、近年急速に発展している。NMTはフレーズベースのシステムを凌駕する素晴らしい結果を示し、手作業による特徴の必要性などの欠点に対処している。NMTのアーキテクチャは通常2つのRNNで構成されており、1つは入力テキスト列を消費し、もう1つは翻訳された出力列を生成する。また、ターゲットとソーストークンを一致させるアテンション機構を伴うことが多い[6]。

NMTは異なる自然言語間のギャップを埋めるものである。ソースコードからコメントを生成することは、ソースコードと自然言語の間の機械翻訳問題の一種である。我々はNMTのアプローチがコメント生成に適用可能かどうかを検討する。本論文では、長いソースコードに効果的に対処するために、一般的なSequence-to-Sequence (Seq2Seq) [37]学習フレームワークに注目した[6]学習を行う。

3 プロダクツアプローチ

ソースコードとコメントの間の移行プロセスは、異なる自然言語間の翻訳プロセスに似ています。既存の研究では、あるソース言語（Javaなど）から別のソース言語（C#など）にコードを変換する機械翻訳手法を適用しています[13]。また、ソースコードから自然言語記述を生成するために、機械翻訳を採用した研究もいくつかあります。Oda et al. [30]は、ステートメントレベルでソースコードの自然言語疑似コードを生成する機械翻訳アプローチを提示しています。本論文では、DeepComはソースコードをメソッドレベルの高レベルな記述に翻訳する。

DeepCom の全体的なフレームワークを図 2 に示す。DeepCom は主に、データ処理、モデル学習、オンラインテストの 3 段階で構成されている。GitHubから取得したソースコードは、Javaメソッドとそれに対応するコメントの並列コーパスに解析・前処理される。構造情報を学習するために、モデルに入力する前に、Javaメソッドを特殊なトラバーサルアプローチによってASTシーケンスに変換する。ASTシーケンスとコメントの並列コーパスを用いて、NMTの考え方に基づく生成ニューラルモデルを構築し、学習させる。学習過程では2つの課題がある。

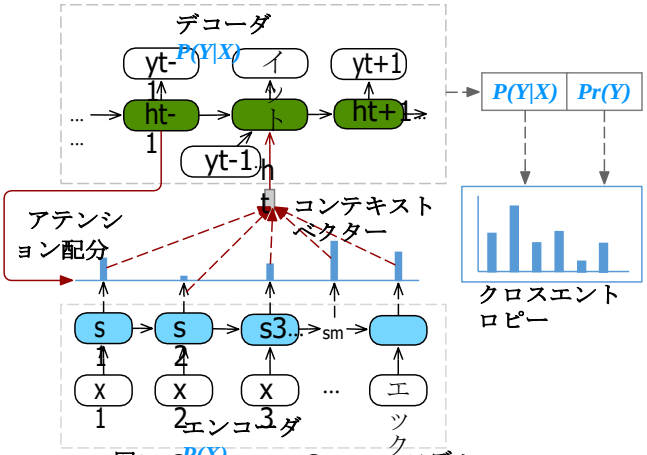


図3：Sequence-to-Sequenceモデル。

- 構造情報を格納し、探索中に表現を曖昧にしないためのASTの表現方法
  ASTは？
- ソースコードに含まれるout-of-vocabularyトークンをどのように扱うか？

以下では、これらの詳細と、上記の課題を解決するために提案するアプローチについて紹介します。

3.1 配列間モデル

本稿では、ソースコードの学習とコメント生成にSequence-to-Sequence（Seq2Seq）モデルを適用する。Seq2Seqモデルは機械翻訳[37]、テキスト要約[34]、対話システム[39]などで広く用いられている。このモデルはエンコーダ、デコーダ、アテンションコンポーネントの3つから構成されており、エンコーダとデコーダは共にLSTMである。図3はSeq2Seqの詳細なモデルを示している。

*3.1.1 エンコーダである。*エンコーダはセクション 2 で説明した LSTM であり、ソースコードの学習を担当する。各時間ステップ t で，
は，トークン列の1つのトークンxt を読み，現在の隠れ状態$_{st}$ ，を更新して記録する。

$$_{st} = f(\text{xt}, _{st\text{-}1}) \quad (2)$$

ここで、fはLSTMユニットで、ソース言語の単語xtを隠れ状態$_{st}$にマッピングする。エンコーダはソースコードから潜在的な特徴を学習し、その特徴をコンテキストベクトルcに符号化する。本論文では、DeepComはコンテキストベクトルcを計算するためにアテンション機構を採用している。

*3.1.2 アテンションアテンション*機構は、各ターゲットワードに対して、入力シーケンスから重要な部分を選択する最近のモデルである。例えば、コメント中の "whether "というトークンは通常ソースコード中の "if "ステートメントと一致する。各単語の生成は、Bahdanauらによって提案された古典的なアテンション手法によって導かれる[6]。

各ターゲット単語 yi を予測するための個人 ci を、エンコーダのすべての隠れ状態 s_1 , ..., sm の加重和として定義し、計算されます。

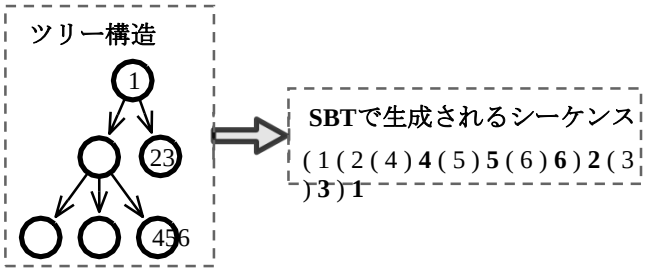


図4：SBTによるASTから配列への順序付けの例。(数値の場合、括弧の後の太字の数値はノード自身を示し、括弧内の数値はそれをルートノードとした木構造を示す)。

as

$$\text{シーアイ} = \sum_{j=1}^{m} \alpha ijsj \quad (3)$$

各隠れ状態 sj の重み αij は次のように計算される。

$$\alpha ij = \frac{exp(eij)}{\sum_{k=1}^{m} exp(eik)} \quad (4)$$

と

$$eij = a(_{hi\text{-}1}, sj) \quad (5)$$

は、位置j付近の入力と位置iの出力がどの程度一致しているかをスコア化するアライメントモデルである。

*3.1.3 復号器。*デコーダは、文脈ベクトル ci とその前に生成された単語 yi の確率を順次予測することで、目標文 y を生成することを目的とする。
語$y1$, ..., $yi\text{-}1$, すなわち。

$$p(yi\,|y1, ..., yi\text{-}1, x) = д(yi\text{-}1, _{hi}, ci) \quad (6)$$

where ∂ is used to estimate the probability of the word y_i. The goal of the model is to minimize the cross-entropy, i.e., minimize the following objective function:

$$H(y) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{n} log p(y_{ji}^{()}) \quad (7)$$

ここで、N は学習インスタンスの総数、n は各ターゲット配列の長さ、 $y_{ji}^{()}$ は第 i インスタンスの*第 j* 単語を意味する。を用いて目的関数を最適化することにより
勾配降下法などのmizationアルゴリズムにより、パラメータを推定することができます。

3.2 SBTトラバーサルによる抽象構文木

ソースコードとNL間の翻訳は、ソースコードの構造上、困難である。ソースコードをモデル化する簡単な方法の1つは、ソースコードを単にプレーンテキストとして見ることである。しかし、この方法では、構造情報が省略され、生成されるコメントに不正確さが生じる。そこで、意味情報と構文情報を同時に学習するために、ASTを走査して特殊な形式のシーケンスに変換する。古典的な探索手法（例えば前置走査）によって得られた配列は、元のASTから曖昧さなく再構築することができないため、損失が大きい。この曖昧さにより、異なるJavaメソッド（それぞれ異なるコメントを持つ）が同じシーケンスにマッピングされることがある

を表現する。特定の入力に対して複数のラベル（我々の設定ではコメント）が与えられると、ニューラルネットワークは混乱する。この問題を解決するために、我々はASTをトラバースするStructure-based Traver- Sal (SBT) メソッドを提案する。その詳細はアルゴリズム1に示されている。図4はSBTによる木の探索の簡単な例であり、詳細な手順は以下の通りである。

- ルートノードから、まず一組の括弧を使って木構造を表し、ルートノードそのものを右括弧の後ろに置く、つまり図4に示すように(1)1とするのである。
- 次に、ルートノードの部分木を走査し、部分木のすべてのルートノードを括弧に入れる、すなわち、
- (1(2)2(3)3)1 とする。再帰的に、すべてのノードが走査されるまで各サブツリーを走査し、最終的に (1(2(4)4(5)5(6)6)2(3)3)1 という順序になる。を取得しました。

アルゴリズム1 構造に基づくトラバーサル

```
procedure SBT(r )▷                ルートrからツリーをトラバースする
    seq ← ∅                       seqはトラバース後のツリーのシーケンス
    if ! r .hasChild then
        seq ← (r)r                終端ノードの括弧を追加する。
    else
        seq ← (r                  非終端ノードに左括弧を追加する。
        for c in childs do
            seq ← seq + SBT(c)です。
        seq ← seq + )r            すべての子ノードをトラバースした後、非終端ノードの右ブラケットを追加する。
    return seq
```

DeepCom は、SBT アルゴリズムに従って、各 AST をシーケンスに処理します。例えば、プロジェクト Eclipse Che2 から抽出された以下の Java メソッドの AST シーケンスは、図 5 に示されるとおりです。

```java
public String extractFor(Integer id){
  LOG.debug("Extracting method with ID:{}", id);
  return requests.remove(id);
}
```

図 5 の左側が本手法の AST である。非終端ノード（ボックスのないもの）は、ソースコードの構造情報を示す。これらは固定集合である「型」を持っている（例：IfStatement、Block、ReturnStatement）。末端ノード（ボックス内のノード）は、「型」だけでなく、「値」（括弧内のトークン）も持っている。値」はソースコードに出現する具体的なトークンであり、「型」はそのトークンの種類を示す。図の右側は、ASTをトラバースして構築されたシーケンスである。末端ノードは「type」と「value」（「_」で接続）で表され、例えば「log」は「SimpleName_Log」と表される。非終端ノードは「型」で表現される。サブツリーは括弧の中に含まれ、与えられたシーケンスから簡単にASTを復元することができる。このように、構造情報を保持したまま、ロスレス表現が可能であり、シーケンスから元のASTを曖昧さなく復元することができる。

3.3 語彙のないトークン

語彙は、ソース・コードのモデル化におけるもう1つの課題です[16]。NL では、通常、研究は語彙を最も一般的な単語に制限しています (例,

[2] https://github.com/eclipse/che

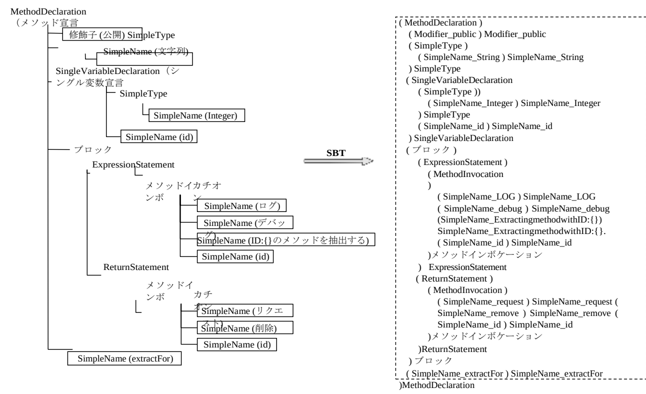


図5：*extractFor*というJavaメソッドのAST。

上位30,000位まで)のデータ処理に使用されます。語彙外トークンは特別な未知トークンであるUNK などに置き換えられる。語彙外の単語は非常に稀であるため、自然言語処理に有効である。しかし、この方法は、ソースコードに関しては不適切であると言わざるを得ない。固定された演算子やキーワードに加えて、コード・トークンの大部分を占めるユーザー定義の識別子が存在します[7]。これらの識別子はラングッジモデルの語彙に大きな影響を与えます。ソースコードの語彙数を一定に保つと、多くの未知のトークンが存在することになります。UNKトークンの出現をできるだけ少なくしようとすると、語彙数が非常に多くなる。語彙サイズが大きいと、学習データ、時間、メモリが多く必要になるため、深層学習モデルの学習が困難になる。最適かつ安定した結果を得るためには、モデルはより多くの反復を実行し、語彙内の各単語のパラメータを調整する必要がある。

そこで、我々はソースコードの語彙外トークンを表現する新しい方法を提案する。ASTでは、非終端ノードは「型」の特徴を持ち、終端ノードは「型」の特徴だけでなく「値」の特徴も持っている。DeepComはASTシーケンスを入力とし、語彙は括弧、ノードのすべての「型」（非終端ノードTnonと終端ノード*Tterm*を含む）、および終端トークンの部分的*型-値*ペアから構成される。AST配列の語彙として、最も頻度の高い3万個のトークンに出現するトークンを残すことにした。語彙の外にある*型-値*ペアについては、Deep-ComはUNKトークンの代わりにその「型」*Tterm*を使用して置き換えます。たとえば、上で紹介したコードのターミナル・ノード "extractFor" と "id" では、図 5 に示すように、それらのタイプは両方とも "SimpleName" です。モデルに入力されるトークンは、それぞれ「Sim-pleName_extractFor」と「SimpleName_id」であるべきである。しかし、トークン "SimpleName_extractFor "はボキャブラリー外なので、代わりにその型 "SimpleName "で表現する。このように、語彙外のトークンは無意味な単語ではなく、関連する型情報によって表現される。

4 実験装置

次に，EclipseのJDTコンパイラ3 を用いて，JavaメソッドをASTにパースし，対応するJavadocコメント（Javaメソッドの標準コメント）を抽出する．Javadoc のないメソッドは本稿では省略した．コメントを持つ各メソッドについて，その Javadoc の記述に現れる最初の文を、以下のように用いる。

[3] http://www.eclipse.org/jdt/

表1：データセットに含まれるコードスニペットの統計値

#メソッド	#全てトークン	#♪すべて識別情報	#ユニークトークン	#ユニーク識別情報
69,708	8,713,079	2,711,496	234,146	234,055

表2：コード長とコメント長の統計値

コードの長さ				
AvgMode	Median	<100	<150	<200
99.94166568		.63%	82.06%	89.00%
コメント長さ				
AvgMode	Median	<20	<30	<50
8.8681375		.50%	86.79%	95.45%

このコメントは，Javadoc ガイダンス4 に従って Java メソッドの機能を記述するのが一般的であるため，このようなコメントは除外している．空白や一語だけの記述は、Javaメソッドの機能を表現する能力がないため、この作業ではフィルタリングしている。また，セッター，ゲッター，コンストラクター，テストメソッドは，モデルによるコメント生成が容易であるため，除外した．

最後に、69,708のJava⟨⟩メソッドとコメントのペア5を得ることができました。類似のJiangら[20]の研究と同様に，80%のペアを訓練用に，10%のペアを検証用に，残りの10%をテスト用にランダムに選択します．表1、表2にコーパスの統計情報を示す。また、メソッドの長さとコメントの長さの詳細も示している。また、このコーパスのJavaメソッドとコメントの平均長は99.94と8.86トークンである。95%以上のコードコメントが50ワード以下であり、約90%のJavaメソッドが200トークン以下であることがわかります。

学習中、数字と文字列はそれぞれ一般的なトークン⟩と⟩に置き換えられる。最大長のASTシーケンスは400に設定される。短いシーケンス⟨⟩には特別な記号PADを使用し、長いシーケンスは400個のトークンでシーケンス⟨⟩に切り分けられることになる。学習時にデコーダシーケンスに特殊なトークンであるSTART⟨⟩とEOSを追加する。STARTはデコードシーケンスの開始を意味し、EOSはその終了を意味する。コメントの最大長は30に設定されている。ASTシーケンスとコメントの語彙サイズは、本パ-フェルでは共に3万である。ASTシーケンスには⟨UNK⟩はないが、コメントには⟨UNK⟩に置き換えられる語彙外トークンが数個存在する。

4.1トレーニングの　内容

モデルは検証セット上で2,000ミニバッチごとに、NMTによく使われる自動評価指標であるBLEU[31]によって検証される。学習は約50エポック実行され、検証セットで最も良い結果を得たモデルを最終的なモデルとして選択する。その後、テストセットにおいて、平均BLEUスコアを計算することでモデルを評価し、その結果についてはセクション5で述べる。すべてのモデルは Tensorflow フレームワーク6 を用いて実装され、それに基づをTensorflowのチュートリアル7にあるSeq2Seqモデルで確認しました。パラメータは以下のように示されている。

- SGD（ミニバッチサイズは学習インスタンスからランダムに100個選択）は、パラメータの学習に使用されます。
- DeepComは2層のLSTMを用い、隠れ状態は512次元、単語埋め込みは512次元である。学習率は0.5とし、勾配ノルム
- を5で切り取る。学習率は0.99を用いて減衰させる。
- オーバーフィッティングを防ぐため、ドロップアウトを0.5とする。

4.2評価　　指標：BLEU-4

DeepComは、生成されたコメントの品質を測定するために、機械翻訳評価指標BLEU-4スコア[31]を使用しています。BLEUスコアは、NMT[22]の精度指標として広く使用されており、ソフトウェアタスクの評価[12, 20]にも使用されています。これは、生成された配列と参照配列（通常は人間が書いた配列）との間の類似度を計算するものである。BLEUスコアは1～100のパーセンテージ値である。BLEUが高いほど、候補が参照に近いことを意味する。候補が参照と完全に等しい場合、BLEUは100％となる。Jiangら[20]は、コミットメッセージの生成された要約を評価するためにこれを利用する。Gu et al. [12]は自然言語のクエリから生成されたAPIシーケンスの精度を評価するためにBLEUを使用している。彼らの実験は、BLEUスコアが生成されたシーケンスの精度を測定するために合理的であることを示している。

候補配列の参照配列に対するn-gramの精度を計算する。スコアは以下のように計算される。

$$BLEU = BP \cdot \exp\left(\sum_{n=1}^{N} wn \log pn\right) \quad (8)$$

ここで、pn は候補に含まれる長さ n の部分配列が参照にも含まれる割合である。本論文では、Nを最大グラム数である4とした。BPはbrevity penaltyである。

$$BP = \begin{cases} 1 & if\ c > r \\ e^{(1-r/c)} & if\ c \le r \end{cases} \quad (9)$$

いて拡張されている。

[4] http://www.oracle.com/technetwork/articles/java/index- 137868.html

ここで、*c*は翻訳候補の長さ、rは有効な参照配列の長さである。

本稿では、生成されたコメントを候補とし、プログラマが書いたコメント（Javadocから抽出したもの）を参照と見なす。

5 RESULTS

このセクションでは、Javaメソッドのコメント生成の精度を測定することで、様々なアプローチを評価します。具体的には、主に以下のリサーチクエスチョンに注目する。

- RQ1：DeepComは、最先端のベースラインと比較して、どの程度の効果があるのか？
- RQ2: DeepCom は、ソース・コードや様々な長さのコメントに対してどの程度効果的ですか?

5.1 RQ1：DeepComとベースラインの比較

5.1.1 *ベースライン*DeepComをCODE-NN [19]と比較します。CODE-NNは最先端のコード要約アプローチであり、また深層学習に基づく手法でもあります。CODE-NNは、エンドツーエンドの生成

[5] データは https://github.com/huxingfree/DeepCom から入手可能です。
[6] https://www.tensorflow.org/ [7] https://github.com/tensorflow/nmt

表3：Javaメソッドに関する評価結果

ApproachesBLEU-4 スコア(%)	
CODE-NN	25.30
Seq2Seq	34.87
アテンションベース Seq2Seq	35.50
ディープコム (予約)	36.01
ディープコム（SBT）	**38.17**

コードスニペットの要約を生成するシステム。これは、ソースコードのための言語モデルを構築する代わりに、ソースコードのトークン埋め込みを統合して要約を生成するために、注目のRNNを利用するものである。CODE-NNの結果はIRベースのアプローチを凌駕しているため、IRアプローチをベースラインとして使用することはしない。

また、DeepComとその変種、すなわち、基本的なSeq2Seqモデル、注意に基づくSeq2Seqモデル、および古典的な探索手法（すなわち前置探索）を用いたDeepComとの比較も行う。Seq2Seqモデルおよび注意に基づくSeq2Seqモデルは、ソースコードを入力とする。これらのモデルは、コメント生成のためのNMTアプローチの有効性を評価することを目的としている。SBTの有効性を評価するために、最も一般的な探索手法の一つである前方一致探索と比較する。また、CODE-NNが使用しているデータセットにおいて、DeepComとCODE-NNの比較も行った。

*5.1.2 結果*自動生成されたコメントと人間が書いたコメントとの差を測定した。この差は、機械翻訳の指標であるBLEU-4スコアによって評価される。表3は、Javaメソッドのコメントを生成するためのさまざまなアプローチの平均BLEU-4スコアを示している。機械翻訳モデルSeq2Seqの精度は、CODE-NNを大幅に上回っている。CODE-NNは、ソースコードのトークン埋め込みからコメントを生成する際に、ソースコードの意味を学習することができない。Seq2SeqモデルはRNNを利用してソースコードの言語モデルを構築し、Javaメソッドのセマンティクスを効果的に学習する。構造情報を統合しながら、BLEU-4スコアがさらに上昇する。前置トラバーサルを用いたDeepComと比較して、SBTベースのモデルは、Javaメソッド内の意味的および構文的情報を学習する能力がはるかに高くなっています。一言で言えば、我々の提案するDeepCom（SBT）のCODE-NNに対する改善度は大きい。DeepComの平均BLEU-4スコアはCODE-NNと比較して約13%向上しています。この結果は、自然言語翻訳における最新のNMTモデルのBLEUスコアが約40%であることと比較可能である[21, 44]。

さらに、CODE- NNが用いたのと同じ、Stack Overflowから収集したC#とSQLのスニペットを含むデータセットで実験を行っている。その結果を表4に示す。彼らの提供するデータセットのコードスニペットの多くは不完全で、ASTに解析するのが難しいので、Seq2SeqモデルとCODE- NNを比較します。その結果、Seq2Seqは異なる言語において最先端の手法であるCODE-NNを凌駕していることが明らかになりました。Seq2Seqの平均BLEUスコアはCODE-NNと比較して、様々なプログラム言語で10%以上向上している。

評価を通じて、コメント生成タスクが機械翻訳と非常によく似ていることが確認されました。

表4：C#とSQLプログラミング言語を含むCODE-NNデータセットでの評価結果。

言語アプローチ		BLEU-4スコア (%)
C#	CODE-NN	20.4
	Seq2Seq	**30.00**
エスキュ	CODE-NN	17.0
	Seq2Seq	**30.94**

ソースコードの構造情報を考慮する必要があります。DeepComは、最先端の手法よりも情報量の多いコメントを生成することができる。ASTを用いないモデルと比較して、DeepComのBLEUスコアは38.17%に増加し、約38%のインスタンスのBLEU-4スコアは50%を超えている。2つの探索手法SBTとpre-order traversalを評価する。SBTを用いたDeepComは、従来のpre-order traversalよりも良い性能を発揮します。これは、SBTの方がASTの構造をよりよく保存しているためである。実験結果は、構造化言語のテキストを非構造化言語に翻訳する際に、構造情報が重要であることを示している。

5.2 RQ2：ソースコードとコメントの長さが異なる場合のBLEU-4スコア

さらに、Javaメソッドと異なる長さのコメントに対する予測精度を分析する。図 6 は、長さの異なるソースコードとグランドトゥルースコメントに対する DeepCom と CODE-NN の平均 BLEU-4 スコアを示しています。図6（a）が示すように、ソースコードの長さを長くすると、平均BLEU-4スコアは低くなる傾向があります。ほとんどのコード長では、DeepComの平均BLEU-4スコアは、CODE-NNのスコアを約10%向上させることができます。DeepComの場合、ソースコード長が長くなるとAST長が急激に伸び、その結果、学習時に長いAST列を固定長の列に切り出す際に、いくつかの特徴が失われてしまうのである。

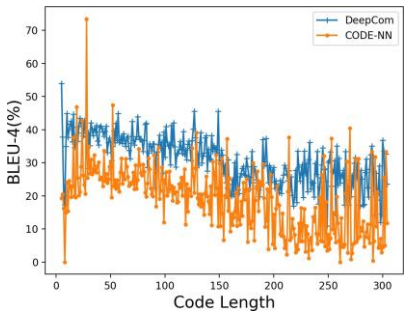
異なる長さのコメントに対して、DeepComは、図6（b）に示すように、同様の精度を維持します。しかし、CODE-NNの精度は、コード・コメントの長さが増加するにつれて急激に低下します。コード・コメントの長さが25トークンを超えると、CODE-NNの精度は10%未満に低下します。25～28語からなるコメントを生成する必要がある場合、DeepComは依然として優れたパフォーマンスを発揮する。

6 ディスカッション

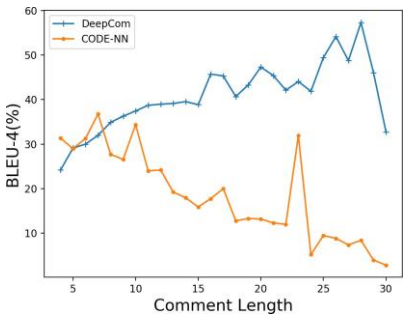
6.1 定性的分析

ここでは、人間が書いたコメントと、我々のアプローチによって自動的に生成されたコメントについて、定性的な分析を行う。表 5 に，Java メソッドの例，DeepCom が生成したコメント，および人間が書いたコ メントを示す．生成された結果の事例を分析した結果，以下のような状況に分類されることがわかった．

*6.1.1 正確なコメント。*DeepCom は、異なる長さのソース・コードから正確に正しいコメントを生成できます (ケース 1 とケース 2)。これは、Java メソッドのエンコードおよびコメントのデコードに対するこのアプローチの能力を検証するものです。一般に、DeepCom



(a) コード長を変えた場合のBLEU-4スコア



(b) コメントの長さを変えた場合のBLEU-4スコア

図6：Java言語におけるコードとコメントの長さの違いによるBLEU-4スコアの平均値。(DeepComとSBT、CODE-NNの2つの方式を比較。)

は、これらのJavaメソッドのビジネスロジックが明確で、コード規約が普遍的である場合に、うまく機能します。

*6.1.2 アルゴリズム実装。*ビジネス・ロジックよりもアルゴリズムに重点を置いた Java メソッドに対して、Deep-Com は正確なコメントを生成できま す。アルゴリズムに関係する Java メソッドは、通常、同じアルゴリズム関数を実装するために類似した構造を使用します。ケース 5 に示すように、"sort" メソッドはバイナリ・ソートを使用して配列を並べ替えることを目的としており、 DeepCom は正しい機能を捕捉し、正しいコメントを生成します。

*6.1.3 生成されたコメントが人間が書いたコメントよりも優れているケース。*生成されたコメントとソース・コードを分析した結果、Java メソッドが真偽の判断を目的としている場合、DeepCom は人間が書いたコメントよりも優れたパフォーマンスを示すことがわかりました。開発者は、コメントとして疑問文を書くことがあります(ケース6およびケース7に示す)。このようなコメントは、Javaメソッドの機能を表現できるにもかかわらず、非標準的なものです。DeepCom は、正確なコメントを生成するだけでなく、より標準的なコメントを生成することもできます。

*6.1.4 API 呼び出しの集中的な Java メソッド。*開発者は通常、特定の機能を実装するためにAPIを呼び出す。これらの API には、プラットフォーム標準 API と、サードパー ティまたは開発者自身によって定義されたカスタマイズされた API があります。DeepComは、API呼び出しのほとんどがプラットフォーム標準APIである場合（ケース1）、正確なコメントを生成できることが分かっています。しかし、Java メソッド内の大部分の API 呼び出しがカスタマイズされた API である場合、DeepCom は人間が書いたコメントほどには機能しません (ケース 1 の例)。4 とケース 9)。API呼び出しの影響から、Deep-Comは大規模データセットからプラットフォーム標準APIの使用パターンを学習することができることがわかる。しかし、カスタマイズされたAPIは、同じ名前のAPIでもプログラムによって使用パターンが異なるため、うまく学習することができない。

*6.1.5 BLEUスコアが低いケース*BLEUスコアが低い結果は、主に「意味のない文章」と「意味が明確な文章」の2種類に分けられる。前者には主に空文や繰り返しの多い単語が含まれる。これは、元のコメントの語彙が少ないか、Javaメソッドと元のデータセットのコメントとの不一致が原因であると考えられる。

後者では、そのほとんどが意味論において原文と無関係である。また、関連するセマンティクスを持ちながらBLEUスコアが低い興味深い結果もある（Case 4に示す）。また、自動生成されたコメントと手動で生成されたコメントは、類似の機能を記述しているが、単語や順序が異なる場合がある。

*6.1.6 生成されたコメントに未知の単語がある。生成さ*れたコメントには、未知の単語が含まれることがあります。ケース3では、開発者が定義したメソッド名である"FactoryConfigu-rationError "というトークンを予測することができません。DeepCom は、コメント内で発生するメソッド名や識別子名を学習することが苦手です。開発者はプログラミング中に様々な名前を定義しており、これらのトークンの多くはコメント中に一度でも出現する。学習過程では、AST列の未知識別子トークンをすべて型に置き換えたが、コメント中の未知識別子については置き換えることはしていない。未知トークン⟨UNK⟩に置き換えられたコメント中のこれらのユーザー定義トークンをDeepComが学習することは困難である。

6.2 DeepComの強み

コードからコメントを生成するための大きな課題は、コードと自然言語の記述の間のセマンティックなギャップである。既存のアプローチは、手作業で作成したテンプレートや情報検索に基づいており、ソースコードと自然言語の間の意味的な関係を捉えるモデルが不足している。機械翻訳モデルであるDeepComは、プログラミング言語と自然言語という2つの言語の間のギャップを埋める能力を有している。

*6.2.1 コードとコメントのセマンティクスを結びつける確率的モデル。*DeepComの利点は、キーワードからコメントを合成したり、類似のコードスニペットのコメントを検索したりするのではなく、ソースコードを学習して直接コメントを生成することである。

キーワードからコメントを合成するには、通常、手動で作成したテンプレートを使用する。しかし、テンプレートの定義には時間がかかり、また、キーワードの質はJavaメソッドの質に依存する。また、識別子やメソッドの名前付けが不十分な場合、正確なキーワードを抽出することができない。IRベースのアプローチは、通常、類似のコードスニペットを検索し、そのコメントを最終結果とする。これらのIRベースのアプローチは、類似のコードスニペットが検索可能かどうか、また、スニペットがどの程度類似しているかに依存する。

DeepComは、コードと自然言語記述のための言語モデルを構築する。言語モデルは、コードとテキストの対応関係の不確実性を処理することができます。DeepCom

表 5: DeepCom によって生成されたコメントの例。これらのサンプルは、スペースの制限により、必然的に短いメソッドに限定されています。AST はソース・コードよりもはるかに長いため、AST 構造は表には示されていません。

ケースID	ジャワメソッド	コメント

1	public static byte[] bitmapToByte(Bitmap b){
	ByteArrayOutputStream o = new
	ByteArrayOutputStream().ByteArrayOutputStream().ByteA
	rrayOutputStream().ByteArrayOutputStream();
	b.compress(Bitmap.CompressFormat.PNG,100,o)を実行します。
	return o.toByteArray();
	}
		自動生成：Bitmap をバイト配列に変換 人間が書く
		：Bitmap をバイト配列に変換

2	private static void addDefaultProfile(SpringApplication
	app, SimpleCommandLinePropertySource source){
	(SpringApplication app,
	SimpleCommandLinePropertySource source) 。
	if(!source.containsProperty("spring.profiles.active"))。
	&&!System.getenv().containsKey("SPRING_PROFILES_ACTIVE")){...
	app.setAdditionalProfiles(Constants.SPRING_PROFILE_DEVELOPMENT);
	}
		自動的に生成されます。プロファイルが設定されていない場合、デフォルトで"dev "プロファイルが設定されます。
		人間が書いたものです。プロファイルが設定されていない場合、デフォルトで "dev "プロファイルが設定されます。

3	}
	public FactoryConfigurationError(Exception
	e){...
	super(e.toString());
	this.exception=e;
	}
		自動的に生成される。与えられたExceptionのベースとなるエラーの原因を持つ新しいUNKを作成する。
		人間が書いたものです。与えられたExceptionベースのエラーの原因を持つ新しいFactoryConfigurationErrorを作成します。

4	protected void
	createItemsLayout(){ if
	(mItemsLayout == null){ ...
	mItemsLayout=new LinearLayout(getContext());
	mItemsLayout.setOrientation(LinearLayout.VERTICAL);
	}
	}
		自動生成。パラメータがあれば、項目レイアウトを作成します。必要に応じて項目のレイアウトを作成

5	public static void sort(Comparable[]
	a){ int n=a.length;
	for (int i=1; i < n; i++){
	Comparable v=a[i];
	int lo=0, hi=i;
	while (lo < hi) { ...
	...
	}
	isSorted(a)をアサートする。
	}
		自動的に生成されます。自然数で昇順に並び替えます。
		人間が書いたものです。自然な順序で昇順に並べ替えます。

6	public boolean isEmpty(){。
	return root == null;
	}
		自動的に生成される。シンボルが空の場合、true を返します。人間が書いたものこのシンボルテーブルが空かどうか？

7	public boolean contains(int key){。
	return rank(key) != -1;
	}
		自動的に生成されます。指定されたオブジェクトが指定された集合に含まれるかどうかをチェックします。
		人間が書いたものです。キーはこの整数の集合の中にあるか？

8	public void tag(String inputFileName,String
	outputFileName, OutputFormat outputFormat){
	(Public void tag(String
	inputFileName,String
	outputFileName,OutputFormat outputFormat))
	.
	List<String> sentences=jsc.textFile(inputFileName).collect();
	tag(sentences,outputFileName,outputFormat)を指定します。
	}
		自動的に生成されます。メッセージを指定されたタグに置き換える人力による書き込み。テキストファイルにタグを付け、一行ごとに文章を書き出し、任意の出力形式で出力ファイルに書き出す。

9	public void unlisten(String pattern){。
	UtilListener listener=listeners.get(pattern);
	if(listener!=null){。
	リスナー.destroy();
	listeners.remove(pattern)。
	}else{
	client.onError(Topic.RECORD,Event.NOT_LISTENING,pattern)。
	}
	}
		自動的に生成される。商品のみ、または更新が終了したときに呼び出すことができる。
		人間が書いたものです。listenFor- サブスクリプションに登録されていたリスナーを削除します。

DeepComは、大規模なソースコードから共通パターンを学習し、エンコーダー自体が言語モデルとして、異なるJavaメソッドの尤度を記憶する。DeepComのデコーダは、ソースコードの文脈を学習し、自然言語とコードの間のギャップを埋める。さらに、アテンション機構は、コードトークンと自然言語の単語を整合させるのに役立つ。

*6.2.2 構造情報による生成の支援*プログラム言語は、より構造化された形式言語である。

テキストよりも密度が高く、正式な構文と意味論を持っている。コード列が与えられただけで、意味と構文の情報を同時に学習することはモデルにとって困難である。既存のアプローチは通常、ソースコードを直接解析し、その構文表現を省略する。

従来のNMTモデルとは対照的に、DeepComは、豊富で曖昧さのないコード構造を利用している。このように、DeepComはソースコード内の構造情報の助けを借りて、コードと自然言語の間のギャップを埋めることができる。より

評価結果では、構造情報がコメントの品質を向上させることがわかった。標準的なアルゴリズムを実装したメソッドでは、より顕著に改善される。同じアルゴリズムを実現するJavaのメソッドは、ASTが似ていても、異なる変数を定義している場合があります。

6.3 妥当性への脅威

妥当性を脅かすものとして、以下のものを挙げている。

自動評価メトリクス生成されたコメントと人間が書いたコメントとの差を、生成論的なソフトウェア問題で徐々に使われるようになった機械翻訳指標BLEUで評価する[12, 20]。この設定の理由は、手動評価による主観の影響を軽減したいからである。

収集したコメントの質Javaメソッドに対するコメントは、他の研究と同様、Javadocの最初の文から収集した[12]。また，ヒューリスティックルールを定義し，コメントのノイズを低減させたが，データセット中には不一致のコメントも存在する．今後、より良い並列コーパスを構築するための手法を検討する。

Javaデータセットでの比較。有効性に対するもう一つの脅威は、我々のアプローチがJavaデータセットで実験されていることです。ASTへのパースが困難なCODE-NN'データセットでDeepComを直接評価することはできませんでしたが、Javaでの結果はDeepComの有効性を証明するものでした。将来的には、他のプログラミング言語（Pythonなど）にも本アプローチを拡張していく予定です。

7 関連作品

7.1 コード要約

ソフトウェアエンジニアリングの重要なタスクであるコード要約は、ソースコードの簡潔な自然言語記述を生成することを目的としています。自動コード要約のアプローチは、手動で作成したテンプレート[24, 35, 36]、IR [14, 15, 43]、学習ベースのアプリケーション[15] など様々です。
プローチ[4, 19, 28]がある。

コードコメントを生成するために手動で作成したテンプレートを作成することは、最も一般的なコード要約アプローチの1つです。Srid- hara ら [35] は Software Word Usage Model (SWUM) を使用して、Java メソッドの自然言語記述を生成するルールベースのモデルを作成しています。Morenoら[25]は、情報を選択するためのヒューリスティックなルールをあらかじめ定義し、その情報を組み合わせることによってJavaクラスのコメントを生成している。これらのルールベースのアプローチは、テストケース[48]やコード変更[8]などの特殊なタイプのコード成果物をカバーするために拡張されました。人間のテンプレートは通常、与えられたソースコードからキーワードを抽出することによってコメントを合成します。

IRアプローチは、サマリー生成や類似のコードスニペットからのコメント検索に広く利用されています。Haiducら[15]はVector Space Model (VSM) と Latent Semantic Indexing (LSI) を適用して、クラスとメソッドのための用語ベースのコメントを生成しています。彼らの研究は、階層的なトピックモデルを利用するEddyら[11]によって複製され、拡張されています。Wongら[42]はコードクローン検出技術を適用し、類似のコードスニペットを見つけ、そのコメントを使用する。この研究は、Stack Overflowからコメントを自動生成するために、人間が書いたデスクリプションをマイニングする、彼らの以前の仕事AutoComment [43]と似ている。
近年、深層学習による自然言語要約の付与を試みる研究がある。Iyerら[19]は、RNNネットワーク

C#コードのスニペットやSQLクエリを記述したサマリーを作成することに重点を置いています。これは、ソースコードをプレーンテキストとして受け取り、要約の条件分布をモデル化する。Allamanisら[4]は、ソースコードスニペットを短い、名前のような要約に極端に要約する問題に、ニューラル畳み込み注意力モデルを適用します。これらの学習ベースのアプローチは、主にソースコードからセマンティクス、フォーマットなどの潜在的な特徴を学習します。これらの学習された特徴に基づき、コメントが生成される。これらの実験結果は、コード要約における深層学習手法の有効性を証明している。本論文では、DeepComがコメント生成に重要であることが確認された構造情報を統合している。

7.2 ソースコードの言語モデル

最近，Hindleら[17]の洞察により，ソースコードの言語モデルを構築することに関心が高まっている．これらの言語モデルは、N-gramモデル[1, 29]、バイモーダルモデル[5]、RNN[12, 19]など様々である。Hindleら[17]は、ソースコードのモデルとしてN-gramを最初に提案し、ほとんどのソフトウェアも自然であることを示し、自然なコードに規則性を見いだしました。また，プログラミング言語と自然言語記述の間のギャップを埋めるモデルを構築する研究もある．Allamanisら[1]は、コードベースのコード規則を学習するフレームワークを開発し、このフレームワークはJava識別子の命名にN-gramモデルを利用する。Alla-manisら[2]とRaychevら[33]は、変数、メソッド、クラスの名前を提案している。Mou ら[26]は、ソースコードをモデル化し、プログラムを分類するために、ツリーベースのコンボリューションニューラルネットワークを提案している。Guら[12]は、JavaのAPI配列と自然言語の間のギャップを埋めるために、古典的なエンコーダ・デコーダのモデルを提示している。Yin and Neubig [47]は自然言語からコードを生成するためのデータ駆動型シンタックスベースの ニューラルネットワークモデルを構築した。

ソースコードからの学習は、様々なソフトウェアエンジニアリングのタスク、例えば、障害検出[32]、コード補完[27、29]、コードクローン[38]、コード要約[19]に適用されています。本論文では、コードコメントを生成するために、深層学習手法とソースコードの特徴を組み合わせることを検討する。これまでの作品と比較して、DeepComは機械翻訳の観点からコード要約の手順を説明する。また、実験結果はDeepComの能力を証明する。

8 結論

本論文では、コード要約タスクを、プログラミング言語で書かれたソースコードを自然言語のコメントに変換する機械翻訳問題として定式化する。我々は、Javaメソッドのコメントを生成するために、注意に基づくSeq2SeqモデルであるDeep-Comを提案する。DeepComはASTシーケンスを入力として受け取る。これらのASTは、新しい構造ベーストラバーサル（SBT）法を用いて、特別にフォーマットされたシーケンスに変換される。SBTは構造情報を表現し、同時に可逆的な表現を維持することができる。DeepComは最先端のアプローチを凌駕し、機械翻訳メトリクスでより良い結果を得ることができた。今後の課題としては、提案手法の有効性を高めるために、よりドメインに特化したカスタマイズを行う予定である。また、機械翻訳問題にマッピング可能な他のソフトウェアエンジニアリングのタスク（例：コードの移行など）にも提案アプローチを適用する予定である。

参考文献

[1] ミルティアディス・アラマニス、アール・T・バー、クリスチャン・バード、チャールズ・サットン。2014.Learn- ing natural coding conventions.In *Proceedings of the 22nd ACM SIGSOFT Interna- tional Symposium on Foundations of Software Engineering*.ACM, 281-293.

[2] ミルティアディス・アラマニス、アール・T・バー、クリスチャン・バード、チャールズ・サットン。2015.Sug- gesting accurate method and class names.In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*.ACM, 38-49.

[3] ミルティアディス・アラマニス、アール・T・バー、プレムクマール・デバンブー、チャールズ・サットン。2017.A Survey of Machine Learning for Big Code and Naturalness. *arXiv preprint arXiv:1709.06182* (2017).

[4] ミルティアディス・アラマニス、ハオ・ペン、チャールズ・サットン。2016.A convolutional at-tention network for extreme summarization of source code.In *International Conference on Machine Learning*.2091-2100.

[5] Miltos Allamanis, Daniel Tarlow, Andrew Gordon, and Yi Wei.2015.ソースコードと自然言語のバイモーダルモデリング。In *International Conference on Machine Learning*.2123-2132.

[6] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio.2014.整列と翻訳の共同学習によるニューラル機械翻訳.*コンピュータサイエンス*(2014).

[7] Manfred Broy, Florian Deißenböck, and Markus Pizka.2005.A holistic approach to software quality at work.In *Proc. 3rd World Congress for Software Quality (3WCSQ)*.

[8] レイモンド PL ビューズ、ウェストレイ R ワイマー。2010.プログラム変更の自動的な文書化.自動*化ソフトウェアエンジニアリングに関するIEEE/ACM国際会議論文集(*In *Proceedings of the IEEE/ACM international conference on Automated Software Engineering*).ACM, 33-42.

[9] Ciprian Chelba, Dan Bikel, Maria Shugrina, Patrick Nguyen, and Shankar Kumar.2012.*自動*音声認識における大規模言語モデリング.

[10] チョン・ジュンヨン、カグラル・グルセール、チョ・キョンヒョン、ベンジオ・ヨシュア.2014.シーケンスモデリングに関するゲート型リカレントニューラルネットワークの経験的評価.*arXiv preprint arXiv:1412.3555* (2014).

[11] Brian P Eddy, Jeffrey A Robinson, Nicholas A Kraft, and Jeffrey C Carver.2013.ソースコードの要約技術を評価する。レプリケーションと拡張。*プログラム理解 (ICPC), 2013 IEEE 21st International Conference on*.IEEE, 13-22.

[12] グ・シャオドン、チャン・ホンユー、チャン・ドンメイ、キム・ソンフン。2016.ディープ API 学習。In *Proceedings of the 2016 24th ACM SIGSOFT International Sympo-sium on Foundations of Software Engineering*.ACM, 631-642.

[13] グ・シャオドン、チャン・ホンユー、チャン・ドンメイ、キム・ソンフン。2017.DeepAM:Migrate APIs with Multi-modal Sequence to Sequence Learning. *arXiv preprint arXiv:1704.07734* (2017).

[14] Sonia Haiduc, Jairo Aponte, and Andrian Marcus.2010.ソースコード要約によるプログラム理解の支援.このような場合、「*ソフトウェア・エンジニアリングのための第32回ACM/IEEE国際会議-第2巻*」において、「ソフトウェア・エンジニアリングのための第32回*ACM/IEEE国際会議-第2巻」を開催します*。ACM, 223- 226.

[15] Sonia Haiduc, Jairo Aponte, Laura Moreno, and Andrian Marcus.2010.ソースコードの要約のための自動テキスト要約技術の使用について。In *Reverse Engineering (WCRE), 2010 17th Working Conference on*.IEEE, 35-44.

[16] ヴィンセント・J・ヘレンドーン、プレムクマル・デバンブー。2017.ディープニューラルネットワークはソースコードのモデリングに最適なのか？In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*.ACM, 763-773.

[17] Abram Hindle, Earl T Barr, Zhendong Su, Mark Gabel, and Premkumar Devanbu.2012.を参照。このような場合、「ソフトウェア・*エンジニアリング（ICSE）」、2012 34th International Conference on*.IEEE, 837-847.

[18] Sepp Hochreiter and Jürgen Schmidhuber.1997.長期短期記憶。*Neural computation* 9, 8 (1997), 1735-1780.

[19] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer.2016.Summarizing Source Code using a Neural Attention Model（ニューラル・アテンション・モデルを用いたソースコードの要約）*.ACL (1)* にて。

[20] Siyuan Jiang, Ameer Armaly, and Collin McMillan.2017.ニューラル機械翻訳を使用した差分からのコミットメッセージの自動生成。In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*.IEEE Press, 135-146.

[21] Melvin Johnson, Mike Schuster, Quoc V Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda Viégas, Martin Wattenberg, Greg Corrado, et al.2016.を参照。Google's multilingual neural machine translation system: enabling zero-shot translation.*arXiv preprint arXiv:1611.04558* (2016).

[22] ギョーム・クライン、ユン・キム、ユンティエン・デン、ジャン・セネラート、アレクサンダー・M・ラッシュ。2017.Opennmt: Open-source toolkit for neural machine translation. *arXiv preprint arXiv:1701.02810* (2017).

[23] パブロ・ロヨラ、エジソン・マレーズ＝テイラー、松尾豊。2017.A Neural Ar- chitecture for Generating Natural Language Descriptions from Source Code Changes. *arXiv preprint arXiv:1704.04856* (2017).

[24] ポール・W・マクバーニー、コリン・マクミラン。2014.メソッドコンテキストのソースコード要約による自動ドキュメンテーション生成.In *Proceedings of the 22nd International Conference on Program Comprehension*.ACM, 279-290.

[25] Laura Moreno, Jairo Aponte, Giriprasad Sridhara, Andrian Marcus, Lori Pollock, and K Vijay-Shanker.2013.このような場合、「曖昧さ」を解消することが重要です。*プログラム理解（ICPC）, 2013 IEEE 21st International*

の会議です。IEEE, 23-32.
[26] Lili Mou, Ge Li, Lu Zhang, Tao Wang, and Zhi Jin.2016.Programming Language Processing のための木構造上の畳み込みニューラルネットワーク*AAAI*、Vol.2.4 にて。
[27] Lili Mou, Rui Men, Ge Li, Lu Zhang, and Zhi Jin.2015.On end-to-end program generation from user intention by deep neural networks. *arXiv preprint arXiv:1510.07211* (2015).
[28] ダナ・モブショビッツ-アティアス、ウィリアム・W・コーエン。2013. プログラミングコメントを予測するための自然言語モデル.(2013).
[29] Tung Thanh Nguyen, Anh Tuan Nguyen, Hoan Anh Nguyen, and Tien Nguyen.2013.ソースコードのための統計的意味言語モデル.*2013年第9回ソフトウェア工学の基礎に関する合同会議論文*集.ACM, 532-542.
[30] 小田裕介、札場裕之、グラハム・ノイビッグ、畑英明、サクリアニ・サクティ、戸田智樹、中村聡。2015.統計的機械翻訳を用いたソースコードからの擬似コード生成の学習(t).In *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on*.ieee, 574- 584.
[31] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu.2002.BLEU: a method for automatic evaluation of machine translation.BLEU: method for automatic evaluation for machine translation, In *Proceedings of the 40th annual meeting on association for computational linguistics*.において、機械翻訳の自動評価を行う。
[32] Baishakhi Ray, Vincent Hellendoorn, Saheel Godhane, Zhaopeng Tu, Alberto Bacchelli, and Premkumar Devanbu.2016.On the naturalness of buggy code.In *Proceedings of the 38th International Conference on Software Engineering*.ACM, 428-439.
[33] Veselin Raychev, Martin Vechev, and Andreas Krause.2015.Predicting program properties from big code.*ACM SIGPLAN Notices*, Vol.50 にて。ACM, 111-124.
[34] Alexander M Rush, Sumit Chopra, and Jason Weston.2015.A neural attention model for abstractive sentence summarization. *arXiv preprint arXiv:1509.00685* (2015).
[35] Giriprasad Sridhara, Emily Hill, Divya Muppaneni, Lori Pollock, and K Vijay- Shanker.2010.Towards automatically generating summary comments for java methods.In *Proceedings of the IEEE/ACM international conference on Automated Software Engineering*.ACM, 43-52.
[36] Giriprasad Sridhara, Lori Pollock, and K Vijay-Shanker.2011.このような場合、「Security」(セキュリティ)が重要な役割を果たします。このような場合、「ソフトウェア・エンジニアリング」（*Software Engineering）の第33回国際会議の議事*録に記載されています。ACM, 101-110.
[37] Ilya Sutskever, Oriol Vinyals, and Quoc V Le.2014.ニューラルネットによる配列間学習。In *Advances in neural information processing systems*.3104- 3112.
[38] ジェフリー・スヴァイレンコ、チャンチャルKロイ。2016.一般化された正確な精度のためのクローン検出ツールの評価のための機械学習ベースのアプローチ。*International Journal of Software Engineering and Knowledge Engineering* 26, 09n10 (2016), 1399-1429.
[39] オリオル・ビニャルズ、クオック・レ。2015.A neural conversational model. *arXiv preprint arXiv:1506.05869* (2015).
[40] Song Wang, Taiyue Liu, and Lin Tan.2016.欠陥予測のためのセマンティック特徴の自動学習. In *Proceedings of the 38th International Conference on Software Engineering*.ACM, 297-308.
[41] マーティン・ホワイト、ミケーレ・トゥファーノ、クリストファー・ヴァンドーム、デニス・ポシヴァニク。2016.Deep learning code fragments for code clone detection.In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*.ACM, 87-98.
[42] エドモンド・ウォン、タイユエ・リウ、リン・タン。2015.Clocom:コメント自動生成のための既存ソースコードのマイニング。In *Software Analysis, Evolution and Reengineering (SANER), 2015 IEEE 22nd International Conference on*.ieee, 380- 389.
[43] Edmund Wong, Jinqiu Yang, and Lin Tan.2013.Autocomment:Mining question and answer sites for automatic comment generation.In *Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering*.IEEE Press, 562-567.
[44] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al.2016年。Googleのニューラル機械翻訳システム。Bridging the gap between human and machine translation.*arXiv preprint arXiv:1609.08144* (2016).
[45] Xin Xia, Lingfeng Bao, David Lo, Zhenchang Xing, Ahmed E Hassan, and Shan- ping Li.2017.プログラム理解度の測定。プロフェッショナルを対象とした大規模なフィールドスタディ。*IEEE Transactions on Software Engineering* (2017).
[46] Jun Yin, Xin Jiang, Zhengdong Lu, Lifeng Shang, Hang Li, and Xiaoming Li.2015.Neural generative question answering. *arXiv preprint arXiv:1512.01337* (2015).
[47] Pengcheng Yin and Graham Neubig.2017.A Syntactic Neural Model for General- Purpose Code Generation. *arXiv preprint arXiv:1704.01696* (2017).
[48] Sai Zhang, Cheng Zhang, and Michael D Ernst.2011.自動化されたドキュメントの推論で失敗したテストを説明する。このような場合、「自動化された*ソフトウェア工学の第26回IEEE/ACM国際会議」において、「自動化されたソフトウェア工学の第26回IEEE/ACM国際会議」の議事録が作成されます*。IEEE Computer Society, 63-72.