LLMを活用した仕様書とソースコード間のトレーサビリティリンク回復

永田出海[1]　メンドンサドスサントスイスラエル[2]
宇都宮魁斗[3]　髙木裕仁[4]　佐々木啓輔[5]
有次正義[6]

トレーサビリティ情報は，効果的なソフトウェアの保守と運用に不可欠である．しかし，手作業でトレーサビリティリンクを生成するのはコストがかかり，ミスが発生しやすい．多くの研究では，情報検索理論を活用し，ソースコードと仕様書の語彙的類似性を利用して，トレーサビリティリンクを自動的に回復している．しかし，これらのアプローチは，仕様書とソースコードの記述品質に大きく依存するため，トレーサビリティリンク回復が必ずしも有意な成果をもたらすとは限らない．本研究では，語彙的類似性だけでなく，仕様書とソースコード間の意味的類似性も考慮することで，トレーサビリティリンク回復の品質を向上させることを目的とする．これを実現するために，大規模言語モデル（LLM）の埋め込み機能を利用して意味情報を抽出する．実験により，意味的類似性を組み込むことで，トレーサビリティリンク回復のF1スコアが最大33%向上することを示した．

1　はじめに

ソフトウェアのトレーサビリティとは，ソフトウェアシステムの開発過程で作成された成果物を，それぞれの異なる視点や抽象レベルから相互に関連付ける能力として定義される [27]．多くの研究により，トレーサビリティがソフトウェアの保守および運用において重要な役割を果たすことが示されている [14] [15] [29]．例えば，トレーサビリティの活用により，ソフトウェアの品質特性の向上 [4]や保守作業の効率性・正確性の改善 [20]が実現されることが報告されている．

このようなトレーサビリティの重要性が認識される一方，トレーサビリティリンクの作成や維持における本質的な課題も指摘されている．既存のシステムにおいて事前に定義されていないトレーサビリティリンクを構築するプロセス（トレーサビリティリンク回復） [6]では，成果物間の要素のすべての組合せを確認する必要があるため，大規模なソフトウェアプロジェクトでは作業コストが著しく高くなる [7]．さらに，手作業でのトレーサビリティリンクの作成はエラーが発生しやすい [11]．

これらの課題に対し，トレーサビリティリンク回復プロセスの自動化および半自動化に関する研究が進められている．特に，情報検索（IR）や機械学習（ML）を活用したアプローチが注目を集めている．MLベースの手法はIRベースと比較して高い精度を示しているものの，手作業で注釈を付けた学習セットに依存するため，開発者の作業負荷が増大するという課題がある [31]．IRベースの手法では，成果物からIRモデルにより算出された成果物間の語彙的類似性に基づき，候補トレースリンクを降順でランク付けする．このランク付け結果を基に，ソフトウェア開発者はトレーサビリティの検証を行うことが可能である．しかしながら，IRベースの手法は，抽象度の差異などに起因する成果物間の意味的ギャップが大きい場合，有効な候補トレースリンクの生成が困難となる．この課題に対し，トピックモデル [3]や単語埋め込み [5]などの技術的アプローチが提案されてきたものの，意味的ギャップの十分な解消には至っていない [13]．

意味的ギャップの解消には成果物の意味理解が不可欠であり，LLMの高度な言語理解能力はこの課題に対する有効な解決策となり得る．LLMは，質問応答 [28]や要約 [33]など，さまざまな自然言語処理タスクにおいて大きな成果を示している．プログラミングの分野においては，LLMによるコード生成の実用性が示されている．Yetistirenら [32]は，GitHub Copilotが生成したコードの91.5%が構文的に正しく実行可能であったことを報告している．また，GitHub Copilotユーザは提案されたコードを平均30%近く受け入れており，LLMの活用が開発の生産性を向上させていることが確認されている [9]．このようなLLMによるコード生成の普及を背景に，LLMによってコードが生成される過程で仕様書とコード間のトレーサビリティリンクを追跡し維持することを目的とした研究も進められている [25]．

本研究では，LLMの高度な言語理解能力を活用した意味的類似度とIRベースの手法による語彙的類似度を統合し，仕様書とソースコード間の意味的・語彙的な関係性を考慮したトレーサビリティリンク回復のアプローチを提案する．提案手法は以下の3つの主要な要素から構成される：

1. 意味的類似性の計算
   - LLMによるコード生成を介した成果物間の表現形式の統一
   - LLMの言語理解能力を活用した成果物の意味的表現の獲得
   - 獲得した表現に基づく成果物間の意味的類似性の定量化
2. 語彙的類似性の計算
   - LLMによる文脈やトピックを考慮したキーワード抽出
   - 抽出キーワードに基づく成果物間の語彙的類似性の定量化
3. 類似性の統合
   - 意味的類似性と語彙的類似性の重み付き統合

統合された類似度に基づきトレーサビリティ候補リンクを生成することで，意味的および語彙的な観点から成果物間の関係性を正

[1] 学生会員　熊本大学大学院自然科学教育部
nagata@st.cs.kumamoto-u.ac.jp
[2] 正会員　熊本大学大学院先端科学研究部
israel@cs.kumamoto-u.ac.jp
[3] 非会員　株式会社アキバ共和国
kaito.utsunomiya@akiba-republic.com
[4] 非会員　株式会社アキバ共和国
yuto.takaki@akiba-republic.com
[5] 非会員　株式会社日経統合システム
keisuke.sasaki@nex.nikkei.com
[6] 正会員　熊本大学大学院先端科学研究部
aritsugi@cs.kumamoto-u.ac.jp

確に捕捉する手法を実現する．

本論文の構成は以下の通りである．2章では関連研究について述べ，3章で提案手法の詳細を説明する．4章では評価実験の設定と結果について述べ，5章でまとめと今後の課題を示す．

2 関連研究

トレーサビリティリンク回復に関する研究は長年にわたり進展し，多様なアプローチが提案されている．これらの中で，IRベースの手法とMLベースの手法は特に高い有効性が示されており，現在の主要なアプローチとして位置づけられている．本章では，IRベースの手法について2.1節で，MLベースの手法について2.2節で詳述する．

2.1 IRベースのトレーサビリティリンク回復

トレーサビリティリンク回復の研究分野において，IRベースの手法が広く採用されている．この手法では，異なるソフトウェア成果物間の語彙的類似性を定量化し，その類似性スコアに基づいて成果物のペアのランク付けを行う．典型的なIRベースの手法は主に3つの主要なフェーズから構成される [10]：(1)コーパスの構築：ストップワード除去やキャメルケース分割などの前処理を行った後，成果物から単語を抽出してコーパスを構築する．(2)term-by-document matrixの作成：tf-idfやBag-of-Wordsを用いて$m \times n$行列を作成する．ここで，mは成果物から抽出された固有の単語の数，nは成果物の要素数を表す．(3)類似性の計算：Vector Space Model [2]やLatent Semantic Indexing [22]などのIRモデルを用いて成果物間の類似性を定量化する．

しかしながら，大規模なソフトウェアシステムにおいては，語彙の増加に伴うterm-by-document matrixのスパース性が課題となる．また，典型的なIRベースの手法では単語間の関係性が考慮されず，成果物間での表現の差異への対応が困難である．これらの課題に対し，近年では単語埋め込みを使用した手法 [5]が注目されている．単語埋め込みは，単語間の意味的関係を考慮した表現学習を可能とし，類似した意味を持つ単語がベクトル空間上で近くに配置される．この手法により，トレーサビリティリンク回復において単語間の関係を効果的に考慮することが可能となったが，多くのアプローチでは単語埋め込みの平均ベクトルの計算などにより成果物全体の表現を生成するため，文脈や構造的な情報が不明確となり，成果物間の類似性を適切に捉えられない可能性があることが指摘されている [13].

IRベースの手法の性能向上を目的として，実行トレース [8]やコード依存関係との組合せ [17]など，多様なアプローチが提案されている．これらのアプローチはIRベースのトレーサビリティリンク回復の性能向上に寄与し得るものの，その効果は基盤となるIRベースの類似度計算の結果に大きく依存する．また，IRベースの手法の半自動的な性質に着目して，ユーザのフィードバックを取り入れたアプローチも提案されている [26]．しかしながら，これらの手法ではユーザによる候補リンクの妥当性判断が必要となる．IRベースの手法により生成された候補リストにおいて関連トレースが下位にランクされることが多く，妥当性判断に多大な人的コストを必要とすることが指摘されている [16].

IRベースの手法は，成果物から抽出した単語を基盤としており，成果物の記述品質により性能が大きく変動する．本研究では，従来のIRベースの手法である成果物間の語彙的類似度計算に加えて，成果物間の意味的類似度を計算することによって，成果物の記述品質による影響を緩和することが可能である．

2.2 MLベースのトレーサビリティリンク回復

トレーサビリティリンク回復のために機械学習を使用する研究が近年増加している． Millsら [24]は、トレーサビリティリンク回復を二値分類問題として捉えた分類モデルの学習手法を提案した． Guoら [12]は，単語埋め込みとリカレントニューラルネットワーク（RNN）を組み合わせて，成果物の意味情報とドメイン知識をトレーサビリティリンク生成に組み込んだアプローチを提案した． Linら [18]は，事前学習済み言語モデルと転移学習を活用したTrace BERT（T-BERT）を提案し，訓練データが不十分の場合に性能が低下するという深層学習モデルの問題に対処している．また，能動学習を導入し，学習データ量を大幅に削減しながら同等の性能を維持することに成功している研究も報告されている [23].

これらのMLベースのアプローチは，一部のトレーサビリティリンクが事前に特定されている場合，IRベースの手法と比較して高い性能を示す．しかしながら，正解データが存在しない未知のプロジェクトへの適用可能性については十分な検証がなされていない．本研究では，事前学習済みLLMを活用することで学習を必要とせず，トレーサビリティリンクが未特定のシステムに対しても適用可能なアプローチを実現する．これにより，新規プロジェクトにおけるトレーサビリティリンク回復の即時実行が可能となる．

3 提案手法

本研究の提案手法は，図1に示すように，意味的類似度の算出，語彙的類似度の算出，および両類似度の統合という3段階のプロセスから構成される．まず，仕様書からコードを生成し，生成コードとソースコード間の意味的類似度を定量化する．次に，仕様書からキーワードを抽出し，仕様書とソースコード間の語彙的類似度を定量化する．最後に，これら2つの類似度を統合することで，仕様書とソースコード間の総合的な類似性を評価する．

3.1 意味的類似度の算出

意味的類似度の算出は，仕様書からのコード生成，生成コードとソースコードの埋め込み生成，埋め込み間の類似度計算の3つのステップから構成される．

3.1.1 コード生成

仕様書はシステムの要件や動作を概念レベルで説明する一方，ソースコードは具体的な実装手順を提供することを目的としている．さらに，仕様書は自然言語で記述される一方，ソースコードはプログラミング言語で実装されており，両者間には顕著な意味的ギャップが存在する [21]．本研究では，コード生成を通じて成果物間の表現形式を統一することで，この意味的ギャップの解消を図る．

コード生成においては，生成コードの実行可能性ではなく，仕様書の意図の反映および適切な識別子の命名を重視する．仕様書の意図の反映は，仕様書からコードへの変換における意味的一貫

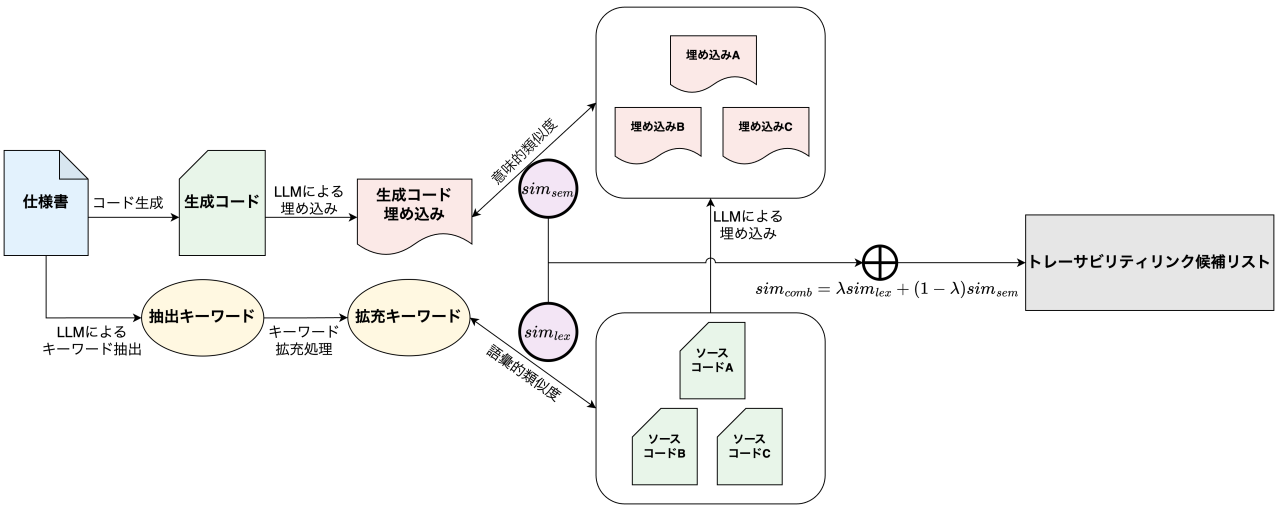


図1: 提案手法のフレームワーク

性を保持する上で重要な要素となる．また，適切な識別子の使用はコード理解やコード検索タスクの性能向上に寄与し，コードの埋め込み表現生成において重要な役割を果たす [19].

これらの要件に基づき，成果物間の意味的ギャップの解消を目的として，仕様書の意図の保持と適切な識別子の命名に重点を置いたプロンプトを設計した．図2に設計したプロンプトを示す．

```
# タスク内容
あなたは {実装言語} を用いて Web アプリケーションを開発しています.
以下に示す仕様書の内容をもとに，対応するプログラムを作成してください.

# 仕様書
{仕様書のテキスト}

# 注意事項
- 仕様書に記載された内容のみをコード化してください.
不要な機能や推測での追加を避けてください.
- 変数名，関数名，クラス名などは仕様書に基づいて適切に命名してください.
```

図2: コード生成のプロンプト

3.1.2 埋め込み生成

LLMはTransformer [30]アーキテクチャを基盤としており，テキスト埋め込みは主に以下のプロセスにより実現される：

1. トークナイズ
   入力テキストをLLMが処理可能な形式に変換するため，トークン単位に分割する．
2. エンコーディング
   トークン単位に分割されたテキストは，LLMの埋め込み層で高次元のベクトル表現に変換される．
3. 文脈情報の取得
   self-attentionメカニズムにより，トークン間の依存関係を考慮した表現を生成する．これにより，各トークンの表現は文脈を反映したものとなり，単語単位では捉えられない複雑な関係性が表現される．
4. テキスト埋め込みの取得
   文脈を考慮したトークンレベルの表現から，特殊トークンの利用や出力の集約などにより，テキスト全体の意味を捉えた高次元な埋め込みベクトルを生成する．

このプロセスにより生成されるテキスト埋め込みは文脈情報を保持しており，埋め込みベクトル間の類似度計算により高精度なテキスト間の関連性評価が可能となる．

3.1.3 類似度計算

生成コードとソースコード間の意味的類似度の定量化にはコサイン類似度を採用する．コサイン類似度は2つのベクトル間の角度の余弦を計測する指標であり，同一方向のベクトル間では1，直交するベクトル間では0となる特性を持つ．生成コードの埋め込み表現$\boldsymbol{v}_a$とソースコードの埋め込み表現$\boldsymbol{v}_b$間のコサイン類似度は以下のように表される．

$$sim_{sem} = \frac{\boldsymbol{v}_a^T \boldsymbol{v}_b}{\|\boldsymbol{v}_a\| \, \|\boldsymbol{v}_b\|} \tag{1}$$

3.2 語彙的類似度の算出

データベース名や出力メッセージなどのシステム固有表現は，トレーサビリティリンクの特定において重要な手がかりとなる．前述の意味的類似度の算出では，成果物間の語彙の共有は考慮されるものの，各語彙のトレーサビリティリンク特定における重要

度を適切に評価することが困難である．したがって，トレーサビリティリンク特定において有効である，重要語の共起情報を十分に活用するため，本研究では重要語の共有度を定量化する語彙的類似性を導入する．

3.2.1 キーワード抽出および拡充

本研究では，語彙的類似度を算出するためのキーワード抽出にLLMを採用する．LLMによるキーワード抽出は，プロンプトエンジニアリングを通じて抽出対象を柔軟に制御できるという利点を持つ．図3は，異なるプロンプトによるキーワード抽出結果の違いを示している．このように，プロンプトの設計により異なる観点でのキーワード抽出が可能となる．

仕様書
ユーザが「ユーザ名」と「パスワード」を入力する入力フォームを作成する．
パスワードの有効期限が切れている場合，「パスワードの有効期限が切れています」というメッセージを表示する．

プロンプト1：入力の仕様書から画面表示テキストやシステム固有の表現を抽出してください．
抽出されるキーワード
ユーザ名，パスワード，パスワードの有効期限が切れています

プロンプト2：入力の仕様書に基づいて実装されたコードを検索する際に有用な検索ワードを仕様書から抽出してください．
抽出されるキーワード
ユーザ名，パスワード，入力フォーム，有効期限

図3: プロンプトによるキーワード抽出の違い

従来の形態素解析に基づくアプローチでは，複数語から構成される重要な表現が分断されるという課題が存在した．例えば，出力メッセージ「パスワードの有効期限が切れています」は形態素解析により，「パスワード」「有効」「期限」「切れ」などの個別要素に分割される．これに対し，LLMを用いたキーワード抽出では，このようなメッセージを一つの表現として抽出することが可能であり，抽出された表現がソースコードに出現する場合，そのソースコードはトレーサビリティリンクの有力な候補となる．

しかし，LLMによって抽出されたキーワードのみを用いて語彙的類似度を算出する場合，その表現が完全一致でソースコード中に出現しない際に類似度の評価に寄与しないという課題が生じる．この課題に対処するため，抽出されたキーワードの拡充処理を導入する．

出力メッセージなどの複数語から構成される表現は，ソースコード中で完全な文字列として記述される場合と，複数の変数に分割され結合して生成される場合が存在する．後者のケースに対応するため，GiNZA [34]を用いたキーワード分割により，キーワードの拡充を実施する．具体的には，図4に示すように，ginza.bunsetu_spansによる文節分割と，ginza.bunsetu_phrase_spansによる文節主要語の抽出を行い，それらをキーワードとして追加する．文節分割された表現がソースコード中の表現と一致する場合，文節主要語の一致と比較して該当するソースコードが絞り込まれやすくなり，トレーサビリティリンク特定においてより有効なキーワードとなることが期待される．

キーワード
パスワードの有効期限が切れています

文節分割
パスワードの，有効期限が，切れています

文節主要語
パスワード，有効期限，切れ

拡充されたキーワード
パスワードの有効期限が切れています，パスワードの，有効期限が，切れています，パスワード，有効期限，切れ

図4: キーワードの拡充

3.2.2 類似度計算

仕様書から抽出・拡充したキーワード集合に基づき，各キーワードの重要度を考慮したベクトル表現を生成する．トレーサビリティリンク特定においては，キーワードによるソースコード検索結果の限定性がそのキーワードの重要性を示唆する．この観点から，本研究では各キーワードを逆文書頻度（IDF）により重み付けしたベクトル表現を採用する．キーワードtのIDFは以下のように計算される．

$$IDF(t) = \log \frac{N}{N(t)+1} \tag{2}$$

ここで，Nは総文書数，$N(t)$はキーワードtが出現する文書数を表す．すべてのキーワードに対してIDF計算を適用してベクトル表現を生成し，仕様書とソースコード間の語彙的類似度をコサイン類似度により算出する．

$$sim_{lex} = \frac{\boldsymbol{v}_c^T \boldsymbol{v}_d}{\|\boldsymbol{v}_c\| \, \|\boldsymbol{v}_d\|} \tag{3}$$

ここで，$\boldsymbol{v}_c$と$\boldsymbol{v}_d$はそれぞれ，仕様書のベクトル表現とソースコードのベクトル表現を表し，仕様書から抽出・拡充された全キーワードを要素とする同一のベクトル空間上で定義される．

3.3 意味的類似性と語彙的類似性の統合

意味的類似度と語彙的類似度を統合し，仕様書とソースコード

間の総合的な類似度を定量化する．統合は両類似度の線形結合として，以下の式で定義する．

$$sim_{comb} = \lambda sim_{lex} + (1 - \lambda) sim_{sem} \tag{4}$$

ここで，λは0以上1以下の重み係数である．λが1に近づくほど語彙的類似度sim_{lex}の影響が強くなり，0に近づくほど意味的類似度sim_{sem}の影響が強くなる．この統合類似度sim_{comb}に基づき降順でランク付けを行い，トレーサビリティリンク候補のランキングリストを生成する．

4 実験

トレーサビリティリンク回復タスクにおける提案手法の有効性を評価するため，実験的検証を行う．実験では，コード生成およびキーワード抽出にGPT-4 [1]を，埋め込み生成にtext-embedding-3-large[*1]を使用した．

4.1 データセット

評価用データセットとして，ユーザインターフェースとデータベース操作を含む非公開のソフトウェアシステムのデータを使用する．データセットの作成にあたり，仕様書をページ単位で分割し169の仕様書要素を，ソースコードを関数単位で分割し1,854のソースコード要素を抽出し，これらの要素間のトレーサビリティリンクを人手で作成した．なお，ソースコードにはコメントは含まれていない．表1にデータセットの概要を示す．

表1: データセットの概要

項目	値
自然言語	日本語
実装言語	HTML/Classic ASP
仕様書要素数	169
ソースコード要素数	1,854
トレーサビリティリンク総数	265
仕様書要素あたりの最大リンク数	5

4.2 評価指標

性能評価には，Precision，Recall，およびそれらの調和平均であるF1-scoreを採用する．評価指標の算出に用いる真陽性（TP），偽陽性（FP），偽陰性（FN）を以下のように定義する．

- TP：検出されたトレーサビリティリンクが正しい場合
- FP：検出されたトレーサビリティリンクが誤っている場合
- FN：実際に存在するトレーサビリティリンクが検出されなかった場合

Precisionは検出されたトレーサビリティリンクの正確性を表す指標であり，以下の式で定義される．

$$Precision = \frac{TP}{TP + FP} \tag{5}$$

Recallは実際に存在するトレーサビリティリンクの検出率を表す指標であり，以下の式で定義される．

$$Recall = \frac{TP}{TP + FN} \tag{6}$$

F1-scoreはPrecisionとRecallのトレードオフを考慮した総合的な性能指標であり，両者の調和平均として以下の式で定義される．

$$F1\text{-}score = \frac{2Precision \times Recall}{Precision + Recall} \tag{7}$$

4.3 実験設定

評価実験では，以下のアプローチを比較対象として設定する：

1. 語彙的類似度のみ：
   - sim_{lex}（キーワード抽出）
   - sim_{lex}（形態素解析）
2. 意味的類似度のみ：sim_{sem}
3. 統合アプローチ：
   - sim_{lex}（キーワード抽出）とsim_{sem}の統合
   - sim_{lex}（形態素解析）とsim_{sem}の統合

IRベースの手法が語彙的類似度に基づくアプローチであることを考慮し，sim_{lex}（キーワード抽出）をベースラインとして採用する．ここでsim_{lex}（キーワード抽出）は，3.2.1節で説明したキーワード抽出および拡充処理を適用したアプローチを指す．sim_{lex}（形態素解析）は，キーワード抽出および拡充処理の有効性を評価するための比較手法として導入し，形態素解析にはGiNZAを使用する．統合アプローチにおける重み係数λは，0から1までを0.1刻みで変化させ，F1-scoreに基づいて最適値を決定する．

評価用データセットがユーザインターフェースやデータベース操作を主とするシステムであることを考慮し，キーワード抽出のためのプロンプトを設計した．

> 入力の仕様書に基づいて実装されたコードに出現する可能性が高いキーワードを抽出してください．これには，出力メッセージ，データベース名，テーブル名などが含まれます．

図5: キーワード抽出のプロンプト

図5に示すように，出力メッセージやデータベース名などの抽出対象を明示的に指定することで，トレーサビリティリンク特定で重要となる表現を選択的に抽出することが可能となる．

4.4 実験結果

表2に，提案手法である$0.2sim_{lex} + 0.8sim_{sem}$（キーワード抽出）と比較対象のアプローチの比較結果を示す．トレーサビリティリンクの最終的な予測手法は現在検討中であるため，本実験では候補リストの上位1位，3位，5位，10位，20位までの予測で比較を行った．実験結果から，提案手法がすべての評価指標（Precision，Recall，F1-score）において比較対象アプローチを顕著に上回ることが確認された．データセットにおける仕様書要素あたりのトレーサビリティリンクの最大数が5であることを考

[*1] https://openai.com/index/new-embedding-models-and-api-updates

表2: トレーサビリティリンク回復の比較結果

アプローチ	@1 Pre	@1 Rec	@1 F1	@3 Pre	@3 Rec	@3 F1	@5 Pre	@5 Rec	@5 F1	@10 Pre	@10 Rec	@10 F1	@20 Pre	@20 Rec	@20 F1
sim_{lex}（形態素解析）	.075	.061	.066	.045	.109	.063	.039	.154	.061	.024	.188	.043	.017	.251	.032
sim_{lex}（キーワード抽出）	.231	.184	.200	.184	.377	.236	.152	.504	.225	.100	.635	.168	.058	.718	.105
sim_{sem}	.313	.214	.241	.209	.386	.256	.159	.492	.230	.101	.609	.167	.059	.703	.106
$0.1sim_{lex}+0.9sim_{sem}$（形態素解析）	.381	.260	.292	.245	.477	.307	.170	.542	.247	.110	.670	.182	.066	.785	.119
$0.2sim_{lex}+0.8sim_{sem}$（キーワード抽出）	**.490**	**.348**	**.390**	**.283**	**.564**	**.361**	**.205**	**.658**	**.300**	**.120**	**.741**	**.201**	**.069**	**.826**	**.125**

慮し，候補リストの上位5位までの結果（@5）に着目して数値的な比較を行う．ベースラインのsim_{lex}（キーワード抽出）と比較して，提案手法は33.3%のF1-scoreの向上を達成した．また，sim_{lex}（形態素解析）のアプローチと比較して，398.8%の大幅な性能向上が確認された．

sim_{lex}（形態素解析）の性能が著しく低い要因として，ソースコードにコメントが含まれていないことが挙げられる．これにより，仕様書とソースコード間の語彙の一致が限定的となり，形態素解析に基づく語彙的類似度では仕様書とソースコード間の関係を適切に捉えることができなかったと考えられる．一方，sim_{lex}（キーワード抽出）では，データベース名や出力メッセージなどのソースコード特定に重要な語彙が抽出可能となり，形態素解析と比較してF1-scoreが268.9%改善された．

sim_{sem}のアプローチについては，ベースラインと比較してF1-scoreの向上が2.2%に留まった．これは，成果物間の意味的類似性は捕捉できているものの，トレーサビリティリンク特定において重要となる，対象ソフトウェアにおける重要語の共有が考慮されていないためだと考えられる．その結果，意味的に類似しているが正解ではないソースコードを適切に除去できなかったと推察される．

sim_{lex}（形態素解析）およびsim_{lex}（キーワード抽出）とsim_{sem}の統合アプローチは，いずれもsim_{lex}，sim_{sem}と比較してF1-scoreの改善が確認された．この結果は，語彙的類似度単体の性能が低い場合でも，意味的類似度との統合によりトレーサビリティリンク回復の性能向上が達成可能であることを示唆している．

仕様書とソースコード間の意味的ギャップの解消におけるコード生成の有効性を検証するため，仕様書の自然言語表現から直接埋め込みを生成するsim_{sem}（仕様書）と，仕様書からコードを生成し，その生成コードから埋め込みを生成するsim_{sem}（生成コード），これらをLLMによるキーワード抽出に基づく語彙的類似度sim_{lex}と統合したアプローチで，トレーサビリティリンク回復の比較を行った．表3に比較結果を示す．sim_{sem}（生成コード）は，sim_{sem}（仕様書）と比較して，F1-scoreの顕著な向上を示した．これは，sim_{sem}（仕様書）では成果物間の記述言語の違いや記述の品質による意味的ギャップを解消できなかったのに対し，sim_{sem}（生成コード）では，コード生成を介して成果物間の表現形式を統一し，統一された形式で埋め込みを生成することで，このギャップを効果的に解消できたためと考えられる．また，sim_{lex}との統合のアプローチにおいて，最適な重み係数が異なることが確認された．sim_{sem}（生成コード）の最適重みが0.2であるのに対し，sim_{sem}（仕様書）では0.4となった．この差異は，sim_{sem}（仕様書）における意味的類似性の捕捉精度の低さをsim_{lex}により補完していると考えられる．その一方で，重み係数が0.4に留まっていることから，トレーサビリティリンク特定において意味的類似性が依然として重要な役割を果たしていることが示唆される．

表3: 仕様書の埋め込みと生成コードの埋め込みの性能比較

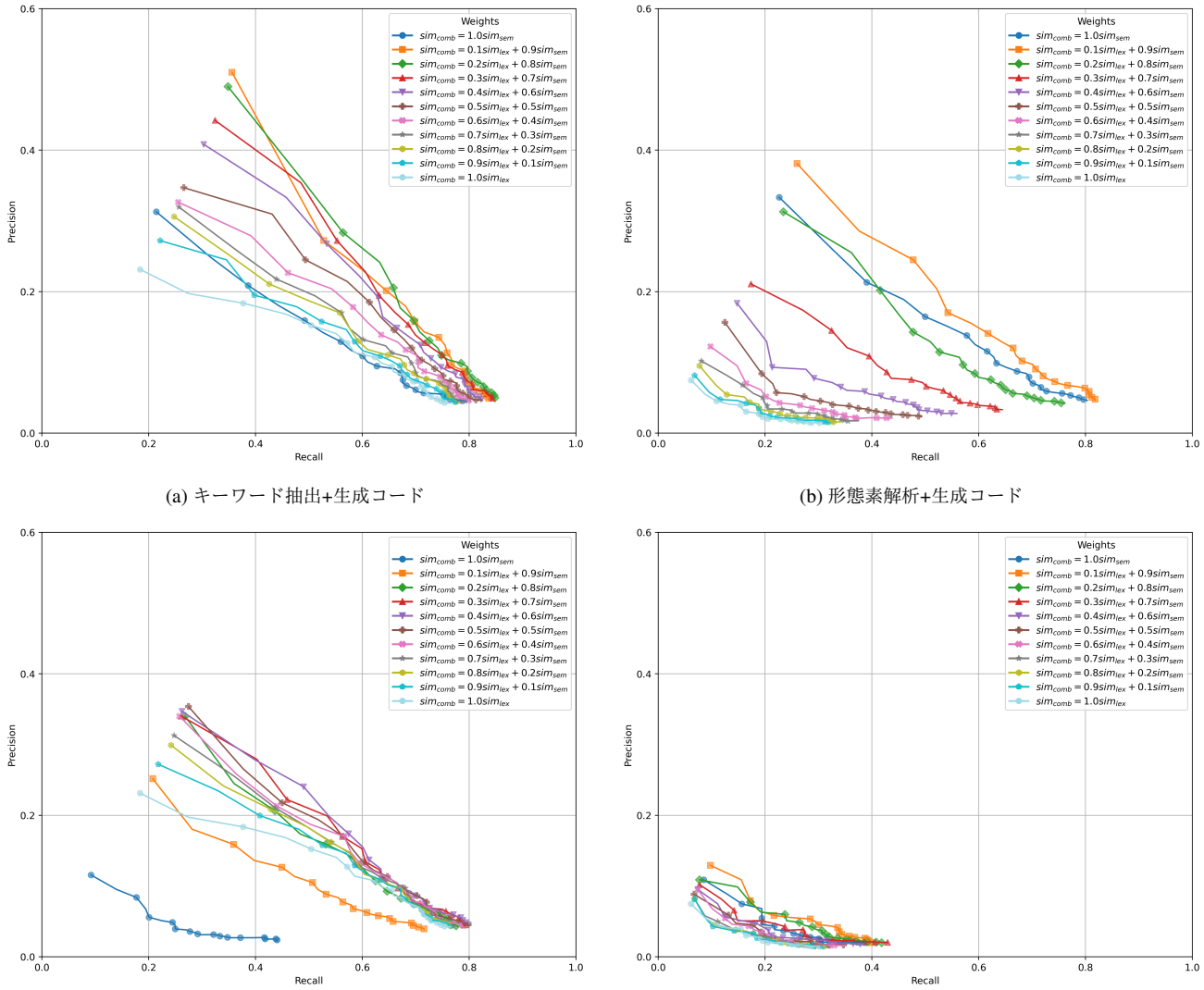
アプローチ	@1 F1	@3 F1	@5 F1	@10 F1
sim_{sem}（仕様書）	.100	.111	.086	.066
sim_{sem}（生成コード）	.241	.256	.230	.167
$0.4sim_{lex}+0.6sim_{sem}$（仕様書）	.288	.309	.258	.173
$0.2sim_{lex}+0.8sim_{sem}$（生成コード）	**.390**	**.361**	**.300**	**.201**

図6に，意味的類似度と語彙的類似度の統合における4種類のバリエーションについて，重み係数を変化させた際のPrecision-Recall曲線を示す．各曲線は候補リストの上位30位までの結果をプロットしている．(a)は，キーワード抽出と生成コードの埋め込みの組合せで，すべての重み係数において，語彙的類似度単体や意味的類似度単体と比較して性能向上が確認された．(b)は，形態素解析と生成コードの埋め込みの組合せで，意味的類似度の重み増加に伴う性能向上が観察され，特に重み係数0.1において，意味的類似度単体を上回る性能を示した．(c)は，キーワード抽出と仕様書の埋め込みの組合せで，重み係数が0.3から0.5の範囲で最適な性能を示した．(b)と(c)では，どちらも意味的類似度単体と語彙的類似度単体の性能に顕著な差があるものの，両者の統合により性能改善を達成している．(d)は，形態素解析と仕様書の埋め込みの組合せで，意味的類似度，語彙的類似度の性能がともに低いものの，それらの統合により性能改善を達成した．

これらの結果は，適切な重み係数の選択による意味的類似度と語彙的類似度の統合が，トレーサビリティリンク回復の性能向上に寄与することを示している．

5 おわりに

本研究では，仕様書とソースコード間のトレーサビリティリンク回復において，LLMを活用した新たなアプローチを提案した．本手法は，LLMによるキーワード抽出に基づく語彙的類似度と，生成コードを介した意味的類似度を統合することで，成果物間の意味的および語彙的な関係性を捕捉する．評価実験の結



(a) キーワード抽出+生成コード

(b) 形態素解析+生成コード

(c) キーワード抽出+仕様書

(d) 形態素解析+仕様書

図6: Precision-Recall曲線

果，LLMによるキーワード抽出が形態素解析と比較して性能向上を示し，コード生成を介した意味的類似度の計算が成果物間の意味的ギャップを効果的に解消することを確認した．さらに，語彙的類似度と意味的類似度の統合により，単独アプローチと比較して顕著な性能向上を実現した．

今後の研究課題として，以下の3点が挙げられる．第一に，提案手法の汎用性を検証するため，より多様なデータセットを用いた評価実験が必要である．特に，本実験ではソースコードの記述品質が低い事例を対象としたが，仕様書の記述品質が低い場合におけるコード生成アプローチの有効性について検証する必要がある．第二に，生成コードの品質評価手法の確立が求められる．具体的には，生成コードが仕様書の意図を適切に反映しているかを定量的に評価する手法の開発が必要である．第三に，類似度統合における重み係数の自動決定手法の確立，および候補リストから最終的なトレーサビリティリンク予測手法の開発が必要となる．

参考文献

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2024.

[2] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo. Recovering traceability links between code and documentation. *IEEE Transactions on Software Engineering*, 28(10):970–983, 2002.

[3] Hazeline U. Asuncion, Arthur U. Asuncion, and Richard N. Taylor. Software traceability with topic modeling. In *2010 ACM/IEEE 32nd International Conference on Software Engineering*, volume 1, pages 95–104, 2010.

[4] Sofia Charalampidou, Apostolos Ampatzoglou, Evangelos Karountzos, and Paris Avgeriou. Empirical studies on software traceability: A mapping study. *Journal of Software: Evolution and Process*, 33(2):e2294, 2021.

[5] Lei Chen, Dandan Wang, Junjie Wang, and Qing Wang. Enhancing unsupervised requirements traceability with sequential semantics. In *2019 26th Asia-Pacific Software Engineering Conference (APSEC)*, pages 23–30, 2019.

[6] Jane Cleland-Huang, Orlena C. Z. Gotel, Jane Huffman Hayes, Patrick

Mäder, and Andrea Zisman. Software traceability: trends and future directions. In *Future of Software Engineering Proceedings*, FOSE 2014, page 55–69, New York, NY, USA, 2014. Association for Computing Machinery.

[7] David Cuddeback, Alex Dekhtyar, and Jane Hayes. Automated requirements traceability: The study of human analysts. In *2010 18th IEEE International Requirements Engineering Conference*, pages 231–240, 2010.

[8] Bogdan Dit, Meghan Revelle, and Denys Poshyvanyk. Integrating information retrieval, execution and link analysis algorithms to improve feature location in software. *Empirical Softw. Eng.*, 18(2):277–309, April 2013.

[9] Thomas Dohmke, Marco Iansiti, and Greg Richards. Sea change in software development: Economic and productivity analysis of the ai-powered developer lifecycle. *arXiv preprint arXiv:2306.15033*, 2023.

[10] Malcom Gethers, Rocco Oliveto, Denys Poshyvanyk, and Andrea De Lucia. On integrating orthogonal information retrieval methods to improve traceability recovery. In *Proceedings of the 2011 27th IEEE International Conference on Software Maintenance*, ICSM '11, pages 133–142, USA, 2011. IEEE Computer Society.

[11] O.C.Z. Gotel and C.W. Finkelstein. An analysis of the requirements traceability problem. In *Proceedings of IEEE International Conference on Requirements Engineering*, pages 94–101, 1994.

[12] Jin Guo, Jinghui Cheng, and Jane Cleland-Huang. Semantically enhanced software traceability using deep learning techniques. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, pages 3–14, 2017.

[13] Tobias Hey, Fei Chen, Sebastian Weigelt, and Walter F. Tichy. Improving traceability link recovery using fine-grained requirements-to-code relations. In *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 12–22, 2021.

[14] Khaled Jaber, Bonita Sharif, and Chang Liu. A study on the effect of traceability links in software maintenance. *IEEE Access*, 1:726–741, 2013.

[15] Muhammad Atif Javed and Uwe Zdun. The supportive effect of traceability links in architecture-level software understanding: Two controlled experiments. In *2014 IEEE/IFIP Conference on Software Architecture*, pages 215–224, 2014.

[16] Hongyu Kuang, Hui Gao, Hao Hu, Xiaoxing Ma, Jian Lü, Patrick Mäder, and Alexander Egyed. Using frugal user feedback with closeness analysis on code to improve ir-based traceability recovery. In *Proceedings of the 27th International Conference on Program Comprehension*, ICPC '19, pages 369–379. IEEE Press, 2019.

[17] Hongyu Kuang, Jia Nie, Hao Hu, Patrick Rempel, Jian Lü, Alexander Egyed, and Patrick Mäder. Analyzing closeness of code dependencies for improving ir-based traceability recovery. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 68–78, 2017.

[18] Jinfeng Lin, Yalin Liu, Qingkai Zeng, Meng Jiang, and Jane Cleland-Huang. Traceability transformed: Generating more accurate links with pre-trained bert models. In *Proceedings of the 43rd International Conference on Software Engineering*, ICSE '21, pages 324–335. IEEE Press, 2021.

[19] Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, MING GONG, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie LIU. Codexglue: A machine learning benchmark dataset for code understanding and generation. In J. Vanschoren and S. Yeung, editors, *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021.

[20] Patrick Mäder and Alexander Egyed. Do developers benefit from requirements traceability when evolving and maintaining a software system? *Empirical Softw. Eng.*, 20(2):413–441, 2015.

[21] Junayed Mahmud, Fahim Faisal, Raihan Islam Arnob, Antonios Anastasopoulos, and Kevin Moran. Code to comment translation: A comparative study on model effectiveness & errors. In Royi Lachmy, Ziyu Yao, Greg Durrett, Milos Gligoric, Junyi Jessy Li, Ray Mooney, Graham Neubig, Yu Su, Huan Sun, and Reut Tsarfaty, editors, *Proceedings of the 1st Workshop on Natural Language Processing for Programming (NLP4Prog 2021)*, pages 1–16, Online, August 2021. Association for Computational Linguistics.

[22] A. Marcus and J.I. Maletic. Recovering documentation-to-source-code traceability links using latent semantic indexing. In *25th International Conference on Software Engineering, 2003. Proceedings.*, pages 125–135, 2003.

[23] Chris Mills, Javier Escobar-Avila, Aditya Bhattacharya, Grigoriy Kondyukov, Shayok Chakraborty, and Sonia Haiduc. Tracing with less data: Active learning for classification-based traceability link recovery. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 103–113, 2019.

[24] Chris Mills, Javier Escobar-Avila, and Sonia Haiduc. Automatic traceability maintenance via machine learning classification. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 369–380, 2018.

[25] Marc North, Amir Atapour-Abarghouei, and Nelly Bencomo. Code gradients: Towards automated traceability of LLM-generated code. In *2024 IEEE 32nd International Requirements Engineering Conference (RE)*, pages 321–329, 2024.

[26] Annibale Panichella, Andrea De Lucia, and Andy Zaidman. Adaptive user feedback for IR-based traceability recovery. In *2015 IEEE/ACM 8th International Symposium on Software and Systems Traceability*, pages 15–21, 2015.

[27] George Spanoudakis and Andrea Zisman. Software traceability: A roadmap. *Handbook of Software Engineering and Knowledge Engineering*, 3, 08 2005.

[28] Yiming Tan, Dehai Min, Yu Li, Wenbo Li, Nan Hu, Yongrui Chen, and Guilin Qi. Can ChatGPT replace traditional KBQA models? An in-depth analysis of the question answering performance of the GPT LLM family. In Terry R. Payne, Valentina Presutti, Guilin Qi, María Poveda-Villalón, Giorgos Stoilos, Laura Hollink, Zoi Kaoudi, Gong Cheng, and Juanzi Li, editors, *The Semantic Web – ISWC 2023*, pages 348–367, Cham, 2023. Springer Nature Switzerland.

[29] Fangchao Tian, Tianlu Wang, Peng Liang, Chong Wang, Arif Ali Khan, and Muhammad Ali Babar. The impact of traceability on software maintenance and evolution: A mapping study. *Journal of Software: Evolution and Process*, 33(10):e2374, 2021.

[30] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[31] Bangchao Wang, Zhiyuan Zou, Hongyan Wan, Yuanbang Li, Yang Deng, and Xingfu Li. An empirical study on the state-of-the-art methods for requirement-to-code traceability link recovery. *Journal of King Saud University - Computer and Information Sciences*, 36(6):102118, 2024.

[32] Burak Yetistiren, Isik Ozsoy, and Eray Tuzun. Assessing the quality of github copilot' s code generation. In *Proceedings of the 18th International Conference on Predictive Models and Data Analytics in Software Engineering*, PROMISE 2022, pages 62–71, New York, NY, USA, 2022. Association for Computing Machinery.

[33] Haopeng Zhang, Xiao Liu, and Jiawei Zhang. Extractive summarization via ChatGPT for faithful summary generation. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 3270–3278, Singapore, December 2023. Association for Computational Linguistics.

[34] 松田寛. Ginza - universal dependencies による実用的日本語解析. 自然言語処理, 27(3):695–701, 2020.