情報処理学会論文誌 Vol.67 No.4 789–797 (Apr. 2026)

機能等価メソッドデータセットを利用したコードクローン検出におけるLLMの最適化

井上 龍太郎[1,a)] 肥後 芳樹[1,b)]

受付日 2025年7月25日, 採録日 2026年1月13日

概要：コードクローンとはソースコード中の一致または類似した部分を持つコード片である．コードクローンはバグの拡散の原因となるため，効率的な検出とリファクタリングが必要である．これまで，静的解析を用いた検出手法や機械学習を用いた検出手法が多数提案されている．近年は，LLM を用いたコーディング支援に注目が集まっており，コードクローンの検出も例外ではない．我々の先行研究では 3 つの LLM に対して FEMPDataset を用いたファインチューニングを行い，コードクローンの検出精度の向上を確認した．本研究では，先行研究を拡張し，より多くのモデルを対象に，RAG とファインチューニングをそれぞれ適用し，コードクローンの検出精度を評価する．実験の結果，ファインチューニングを用いて学習を行った場合は，すべてのモデルで F1 スコアが 7〜72%の範囲で向上した．一方で，RAG を使用した場合は，Chat に特化したモデルでは F1 スコアの向上は認められなかったが，Code に特化したモデルでは 14〜42%の範囲で向上した．また，すべてのモデルで RAG を用いた場合に比べファインチューニングを用いた場合の方が F1 スコアが高かった．

キーワード：コードクローン，LLM，ファインチューニング，RAG

Optimizing LLMs for Code Clone Detection Using Functional Equivalent Methods

Ryutaro Inoue[1,a)] Yoshiki Higo[1,b)]

Received: July 25, 2025, Accepted: January 13, 2026

Abstract: A Code clone is a code snippet identical or similar to another in the source code. The presence of code clones causes the spread of bugs, which means that efficient code clone detection and appropriate refactoring are necessary. Various methods for detecting code clones have been proposed, including static analysis and machine learning techniques. Coding support using LLMs has recently gained significant attention, and code clone detection is no exception. In our previous work, we fine-tuned three LLMs using the FEMPDataset and confirmed improvements in accuracy. In this study, we extend our previous research by applying RAG and fine-tuning to more models to evaluate their F1-score in detecting code clones. The results showed that fine-tuning improved F1-score by 7–72% across all models. Using RAG yielded no improvement in F1-score for chat-specialized models, whereas code-specialized models improved by 14–42%. Overall, fine-tuning performed better than RAG in all models.

Keywords: code clone, LLM, Fine-tuning, RAG

1. まえがき

コードクローンとは，「ソースコード中に存在する互いに一致または類似した部分を持つコード片」のことである [1]．コードクローンに対するコーディングは一貫した変更が必要となることがある．この場合，一貫性のない変

[1] 大阪大学大学院情報科学研究科
Graduate School of Information Science and Technology, The University of Osaka, Suita, Osaka 565–0871, Japan
a) ry-inoue@ist.osaka-u.ac.jp
b) higo@ist.osaka-u.ac.jp

© 2026 Information Processing Society of Japan

更は欠陥の原因となる [2]．このように，コードクローンの生成はソースコードの修正に大きな障害となってしまい，システムの保守性を大きく損ねてしまう．このため，コードクローンを効率的に検出し，必要に応じてリファクタリングする必要がある．

これまで，コードクローンの検出を行うツールが数多く開発されてきた．既存のコードクローン検出ツールとしてCCFinder [3]，NiCad [4]，NIL [5] などのツールがあげられる．これらのツールは字句解析やメトリクスなどを用いて検出を行っている．しかし，既存のツールでは構文的な類似度の高いコードクローンに対しては高い検出精度を示すが，構文的な類似度の低いコードクローンに対しては精度が低いことが課題としてあげられる．

また，機械学習を使用したコードクローン検出手法も提案されている．機械学習を用いたコードクローン検出ツールの例として，ASTNN [6]，GMN [7]，CodeBERT [8] などがあげられる．これらのツールは，構文的な類似度の低いType-3 や Type-4 のコードクローンに対して高い検出精度を示している [9]．

LLM（大規模言語モデル）を用いたコードクローン検出においても，構文的な類似度の低いコードクローンに対して高い検出精度を実現している．LLM は主に自然言語処理の分野で高い成果を上げており，現在，多くの研究分野で注目されている．OpenAI が提供している gpt-4 [10] はテキスト生成のベンチマークで高い性能を示している．また，Meta が開発した商用利用が可能なオープンソースである Llama3.1 [11] や，Llama2 [12] をベースモデルとしてプログラムに関する知識を学習させた code-llama [13] などのモデルが開発されている．その他にも，phi-3 [14]，gemma-2 [15]，codegemma [16] など，多くのモデルが開発されている [17]．LLM には Chat に特化したモデルや Code に特化したプログラミング知識を学習したモデル，その両方に対応したモデルが存在する．

Dou らの先行研究では，複数の LLM を用いてコードクローンを検出し，その性能を LLM 以外の既存ツールと比較している [18]．NiCad といった静的解析を用いたコードクローン検出ツールでは構文的な類似度の低いコードクローンは，検出が難しく精度が悪い．一方で LLM を用いたコードクローン検出では構文的な類似度の低いコードクローンに対して，LLM を用いない既存ツールよりも高い検出精度を実現している．

以上の研究をもとに，我々の先行研究では 3 つの LLM に対して FEMPDataset を用いたファインチューニングを行い，精度の向上を認めた [19]．しかし，先行研究では 3 つの LLM に対象が限定されており，精度向上の手法もファインチューニングに限定されている．

そこで，本研究では先行研究を拡張し，多くのモデルに対して RAG とファインチューニングを用いた場合の精度を評価し，LLM を用いたコードクローンの検出精度向上に向けた指針を示す．対象とする LLM は，gpt-4o，codegemma-7b-it，CodeLlama-7b-Instruct，gemma-2-9b-it，Phi-3-small-128k-instruct，Llama3.1-8B-Instruct の 6 つである．また，ファインチューニングを用いた場合と RAG を用いた場合での精度の比較を行う．ファインチューニングは gpt-4o に関しては OpenAI の API を使用し，その他の LLM に関しては Huggingface からモデルを取得しローカルの環境で実行した．ローカルの環境でモデルをファインチューニングするにあたっては LoRA や ZeRO などの技術を使用した．RAG では OpenAI の埋め込みモデルを使用してベクトル化したデータベースから関連する情報を取得し，プロンプトにその情報を追加することで精度向上を目指した．RAG やファインチューニング，およびその性能評価には FEMPDataset を使用した．性能評価は，再現率，適合率，F1 スコアの 3 つの指標を用いた．実験の結果から，RAG よりもファインチューニングを用いた場合に精度が向上することが示された．また，RAG を使用した場合，Chat に特化したモデルでは精度が変化しなかったが，Code に特化したモデルでは精度が向上した．

本研究の貢献は以下のとおりである．

- 複数の LLM に対する RAG とファインチューニングの適用とその評価．
- RAG を用いた場合とファインチューニングを用いた場合の精度を比較．
- Chat に特化したモデルと Code に特化したモデルでの精度の比較．

2. 準備

この章では，コードクローンの定義や，コードクローン検出に関する先行研究，LLM に関する技術について述べる．

2.1 コードクローン

コードクローンとは，「ソースコード中に存在する互いに一致または類似した部分を持つコード片」のことである [1]．コードクローンは，主にコピーアンドペーストなどの既存コードの再利用や，類似した機能を大規模システム内に再び実装することでプログラム内に作られる [20]．また，コードクローンの関係にあるコード片の組をクローンペアと呼ぶ．

2.1.1 コードクローンの分類

Roy らはコードクローンを類似度をもとに 4 つに分類し，定義した [21]．

Type-1（T1） 改行・スペース・コメントなどのレイアウトの違いを除いて一致するコードクローン．

Type-2（T2） Type-1 に加えて，識別子，リテラル，型の違いを除いて一致するコードクローン．

Type-3（T3） Type-2 に加えて，文の変更・挿入・削除

© 2026 Information Processing Society of Japan

などの違いを除いて一致するコードクローン．

Type-4（T4） 構文的な違いを持つが，同一の機能を提供するコードクローン．

2.2 FEMPDataset

FEMPDataset [19] は異構造で機能等価なメソッドを集めたデータセットである．テストケースの相互実行により機能等価なメソッドペアの候補を抽出し，人が目視で真に機能等価なメソッドペアを選別することで，正確なデータセットを構築している．FEMPDataset に含まれる機能等価なメソッドペアと機能等価でないメソッドペアは，コードクローン検出ツールにおいて正しく分類することが求められる．

2.3 LLM（大規模言語モデル）

LLM（大規模言語モデル）は，大規模なコーパスを用いて学習した言語モデルである．2017 年に Transformer [22] と呼ばれるモデルが発表され，以後モデルの大規模化が進み，学習するデータの量が増えている．今後もモデルの大規模化にともない，その性能は向上することが予想される．

2.4 ファインチューニングにおけるメモリ削減の技術

ファインチューニングでは多くの GPU メモリが必要になる．ファインチューニングにおいて GPU メモリを削減するいくつかの技術について述べる．

2.4.1 LoRA（Low-Rank Adaptation）

LoRA（Low-Rank Adaptation）[23] は，ファインチューニング時に変更するパラメータ数を削減し，少ないリソースでファインチューニングを実現する技術である．

LoRA では，ファインチューニング前後のパラメータの差分を低ランク行列で近似することで，パラメータ数を削減し効率的に学習する．また，ハイパーパラメータ r を指定し低ランク行列の大きさを決定する．r が小さいほどパラメータ数が削減される．

2.4.2 ZeRO（Zero Redundancy Optimizer）

ZeRO（Zero Redundancy Optimizer）[24] は複数の GPU を活用することで，学習時に各 GPU で必要な GPU メモリを削減する技術である．

ファインチューニング時には，学習率の動的変化を管理するための情報，勾配，重みを GPU 上に保持し，計算する必要がある．そこで ZeRO では，各 GPU が LLM の特定の層に対してパラメータ変更を行い，その結果をまとめて LLM 全体のパラメータの変更とする．各 GPU は LLM の特定の層に対するパラメータの変更のみを担当することから，従来の手法よりも各 GPU で必要な GPU メモリを削減することができる．

2.5 RAG

ユーザーが入力したプロンプトに対して，事前に用意されたデータベースを参照し，関連する情報をプロンプトに付与する仕組みを RAG と呼ぶ [25]．RAG は牽引付け，検索，生成の 3 つの処理からなる．牽引付けでは，テキストデータを埋め込みモデルでベクトル化し，データベースに保存する．検索では，ユーザーからの質問をもとにデータベースから関連するデータを検索，取得する．生成では，取得したデータを埋め込んだプロンプトを LLM に入力して回答を生成する．RAG は外部情報による知識の補完だけではなく，コード補完タスクなどの構造的タスクに対しても有効であり，コードクローン検出においても同様の効果が期待される [26]．また，RAG はファインチューニングより少ない計算資源で実現できる利点があり，特定のタスクに対してはファインチューニングよりもベースモデルに対する精度が向上することが報告されている [27]．一方で，検索で無関係なテキストが取得されることがあり，回答の精度が低くなる場合がある．

2.6 検出精度指標

コードクローン検出や LLM の性能指標には再現率，適合率，F1 スコアが用いられる．これらのメトリクスの意味と計算式は以下のとおりである．

再現率：実際にクローンであるメソッドペアのうち，コードクローンであると判定されたメソッドペアの割合．

$$\text{再現率} = \frac{TP}{TP + FN}$$

適合率：コードクローンであると判定されたメソッドペアのうち，実際にコードクローンであるメソッドペアの割合．

$$\text{適合率} = \frac{TP}{TP + FP}$$

F1 スコア：再現率と適合率の調和平均．

$$\text{F1 スコア} = \frac{2 \times \text{再現率} \times \text{適合率}}{\text{再現率} + \text{適合率}}$$

また，TP（True Positive），FP（False Positive），FN（False Negative），TN（True Negative）の意味は以下のとおりである．

TP：実際にコードクローンであるメソッドのうち，コードクローンと判定されたメソッドペア数．

FP：実際にコードクローンでないメソッドのうち，コードクローンと判定されたメソッドペア数．

FN：実際にコードクローンであるメソッドのうち，コードクローンでないと判定されたメソッドペア数．

TN：実際にコードクローンでないメソッドのうち，コードクローンでないと判定されたメソッドペア数．

© 2026 Information Processing Society of Japan

3. 関連研究

この章では，LLM を利用したコードクローン検出に関する関連研究とクローン検出の評価ベンチマークに関する関連研究を述べる．

3.1 LLM を用いたコードクローン検出

LLM を用いたコードクローン検出に関する先行研究として，Khajezade らの研究 [28] を紹介する．Khajezade らは，ChatGPT（GPT-3.5-turbo）を用いて CodeNet データセット上の Type-4 クローン検出を行い，問題の難易度やコード構造の複雑さが検出精度に与える影響を分析した．分析には，Cyclomatic Complexity（制御構造の複雑さを表す指標）と，Acceptance Rate（各問題の正答率を示す難易度指標）を用いて，コードの構造的複雑さと問題難易度の両面から検討を行った．その結果，誤判定されたコードペアでは，クローン対の平均 Cyclomatic Complexity が 2.54，非クローン対では 3.16 と，全体平均 2.98 より高く，また Acceptance Rate も低いことが確認された．このことから，分岐や制御構造が多く複雑な問題ほど判断が難しく，誤判定が増加することが示された．

3.2 ファインチューニング前後のクローン検出精度の評価

我々は以前，LLM を用いたコードクローン検出の精度向上を目指し，FEMPDataset を用いたファインチューニングを行った [29]．対象とした LLM は gpt-3.5-turbo，Llama2-Chat-7B，CodeLlama-7B-Instruct の 3 つである．ファインチューニングには OpenAI の API と LoRA や ZeRO といった技術を使用した．ファインチューニング前後のクローン検出精度を比較した結果，すべてのモデルでファインチューニングを行った方が精度が向上したことが示された．また，Code に特化した CodeLlama-7B-Instruct では精度が大きく向上した．

3.3 Type-4 クローン検出の評価ベンチマーク

クローン検出の評価ベンチマークとして，BigCloneBench [30] が提案されている．BigCloneBench [30] は Svajlenko らによって作成されたコードクローン検出のベンチマークである．ファイルのコピーやバブルソートといった 45 の機能ごとに，その機能を持つメソッドを抽出し，クローンペアを作成している．BigCloneBench は，Type-1 から Type-4 までのコードクローンを含むデータセットであり，しばしばコードクローン検出ツールの性能評価に用いられる．しかしながら，データの偏りやデータのラベリングに対して異議を唱える声もあり，機械学習の学習データとしては適していないとの指摘もある [31]．

SemanticCloneBench [32] は，プログラミング言語における質問回答 Web サイトである StackOverflow から作成された Type-4 クローンのデータセットである．正解クローンを 2 名の審査員によって手動検証し，検証結果を元にデータセットを作成している．

GPTCloneBench [33] は，SemanticCloneBench のデータセットをもとに gpt-3 を用いて新たに作成された Type-4 クローンのデータセットである．生成されたメソッドペアは最終的に審査員 6 名によって検証され，検証結果を元にデータセットを作成している．

これらのデータセットは，人間の目視による判断でラベル付けされており，実際の動作とメソッドペアの等価性を保証するものではない．対して，FEMPDataset はテストを相互に実行し，3 名による目視確認を実施することでメソッドの機能等価性を担保している．以上のデータセットの特性から，本研究では FEMPDataset を使用してファインチューニングおよび RAG を行い，その性能を評価する事とする．

4. 実験

この章では，本研究の実験の詳細について述べる．

本研究では複数の LLM に対して, RAG とファインチューニングをそれぞれ適用し，クローンの検出精度を評価する．

以下に実験の手順を示す．

STEP1：ファインチューニング

FEMPDataset を用いて，LLM のファインチューニングを行う．

STEP2：**LLM の実行**

ベースモデル，ファインチューニング後の LLM，ベースモデルに対して RAG を使用した場合それぞれに対して，FEMPDataset のメソッドペアがクローンペアであるかを質問し，回答を “Yes” または “No” で得る．

STEP3：性能評価

LLM の回答を集計し，性能の比較を行う．

4.1 対象の LLM

実験対象とする LLM は以下のとおりである．研究で多く使用されていることから OpenAI 社の gpt を選択した．また，ローカル環境で動作するモデルの中から，Chat に特化したモデルを 3 つ，Code に特化したモデルを 2 つ選択した．

- gpt-4o
- codegemma-7b-it
- CodeLlama-7b-Instruct
- gemma-2-9b-it
- Phi-3-small-128k-instruct
- Llama-3.1-8B-Instruct

以降，順に gpt-4o，codegemma-7b，codellama-7b，gemma-2-9b，phi-3-small，llama3.1-7b と表記する．

© 2026 Information Processing Society of Japan

表 1 ファインチューニング時の各データのメソッドペア数の内訳
Table 1 Breakdown of method pairs in fine-tuning datasets.

	訓練データ	検証データ	テストデータ
クローンペア数	1,081	132	129
非クローンペア数	674	88	90
合計	1,755	220	219
			計 2,194

4.2 データセット

RAG とファインチューニングおよびその性能評価には，FEMPDataset を使用する．FEMPDataset は異構造で機能等価なメソッドペアを集めたデータセットである．2,194 のメソッドペアからなり，このうち機能等価なメソッドペアが 1,342 ペア，機能等価ではないメソッドペアが 852 ペアである．ファインチューニングを適用する場合には，FEMPDataset を訓練データ，検証データ，テストデータに分割する．RAG の場合には，ファインチューニングの際に分割した訓練データと検証データを外部データ，テストデータをテストデータとして使用する．ベースモデル，ファインチューニングとベースモデルに対して RAG を用いた場合の性能評価には同じテストデータを使用する．

ファインチューニングを適用した場合における FEMPDataset の分割とメソッドペア数の内訳は**表 1** のとおりである．

4.3 ファインチューニング

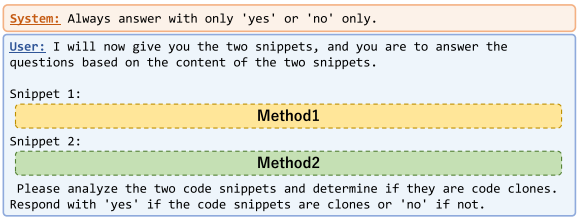
ファインチューニングの学習データには FEMPDataset を用いる．データセットは訓練データ，検証データ，テストデータの 3 つに分割する．ファインチューニングは訓練データと検証データを，性能評価はテストデータを用いて行う．

gpt-4o のファインチューニングは，OpenAI API を用いて行う．その他の LLM はローカル環境でファインチューニングを行う．ローカル環境での実行は GPU メモリ削減のため，LoRA と ZeRO を利用して実行する．また，2.4 節で説明した LoRA のハイパーパラメータ r は 2 とした．過学習を防ぐため Early stopping で学習エポックを決定した．学習には Quadro RTX 6000 を 4 つ用いた．

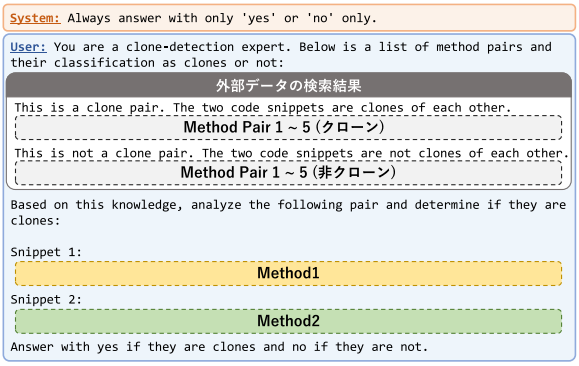
4.4 RAG

RAG ではファインチューニングの際に分割した訓練データと検証データを外部データとして使用する．テストデータはファインチューニングの際に分割したテストデータを使用する．

外部データをベクトル化するのには OpenAI の text-embedding-3 を使用した．クローンペアとクローンではないメソッドペアをベクトル化した外部データを作成した．メソッドペアのベクトル化においては，メソッドの順番によるベクトルの違いを考慮し，各メソッドペアに対して 2 つのベクトルを生成した．メソッド A とメソッド B からなるペアに対して，「Snippet 1: メソッド A Snippet 2: メソッド B」と「Snippet 1: メソッド B Snippet 2: メソッド A」の 2 種類のテキストを作成しベクトル化を行った．この操作によって，メソッドの順番に依存しない形で RAG の検索データを抽出できる．入力されたメソッドペアをベクトル化し，外部データの中からコサイン類似度の高い順に，ペアの重複なくクローンペアとクローンでないペアを 5 個ずつ取得した．取得したデータを埋め込んだプロンプトを LLM に入力し，回答を得た．



(a) 基本のプロンプト



(b) RAG 使用時のプロンプト

図 1 本研究で使用したプロンプトのテンプレート
Fig. 1 Prompt templates used in this study.

4.5 プロンプト

LLM に入力するプロンプトは “system” と “user”，2 つのロールで構成される．“system” では，“Yes” または “No” のどちらかで回答するように回答方法を指定する．“user” では，判定対象の 2 つのメソッドを提示し，それらがクローンペアであるかを判定するように指示する．プロンプトのテンプレートを**図 1** (a) に示す．

RAG を使用する場合は，“user” のプロンプトに外部データから取得されたクローンペア 5 個とクローンでないペア 5 個を提示する．これにより，判定対象のメソッドペアについて，提示された例と比較してクローンペアかどうかを判定するよう指示する．なお，意味ベクトルによる類似度検索により選択される外部データは，使用するモデルにかかわらず同一である．プロンプトのテンプレートを図 1 (b) に示す．

© 2026 Information Processing Society of Japan

4.6 性能評価

LLM から得た回答を集計し，再現率，適合率，F1 スコアの 3 値を計算し性能を評価する．評価に使用するデータセットは，FEMPDataset のテストデータ計 219 ペアである．

4.7 評価項目

本研究では以下に示す 3 個の評価項目を設定する．

(1) ファインチューニングと RAG を使用した場合に LLM を用いたクローン検出精度が向上するか．
(2) ファインチューニングを使用した場合と RAG を使用した場合で LLM を用いたクローン検出精度の向上に差があるか．
(3) ファインチューニングと RAG を使用した場合に Code に特化した LLM と Chat に特化した LLM の間にクローン検出精度の差があるか．

5. 実験結果

この章では，4.7 節で示した評価項目に基づいて，実験結果を述べる．各モデルに関して，ファインチューニングと RAG を用いた場合のクローン検出精度を評価する．その後，各モデルの特徴とクローン検出精度との関連性について調査する．実験結果を**表 2** に示す．

表 2 RAG および FT[*1] を用いたクローン検出の実験結果
Table 2 Code clone detection results using RAG and fine-tuning.

モデル	手法	再現率	適合率	F1 スコア
gpt-4o（大規模汎用）	ベース[*2]	**0.93**	0.76	0.84
	FT	0.89	**0.90**	**0.89**
	RAG	0.85	0.83	0.84
codegemma-7b（Code 特化）	ベース	0.06	**0.89**	0.12
	FT	**0.89**	0.80	**0.84**
	RAG	0.41	0.79	0.54
codellama-7b（Code 特化）	ベース	0.51	0.71	0.59
	FT	**0.95**	**0.76**	**0.85**
	RAG	0.85	0.64	0.73
gemma-2-9b（Chat 特化）	ベース	**0.98**	0.65	0.78
	FT	0.90	**0.82**	**0.86**
	RAG	0.74	0.77	0.75
phi-3-small（Chat 特化）	ベース	0.82	0.74	0.78
	FT	0.86	**0.85**	**0.85**
	RAG	**0.97**	0.67	0.79
llama3.1-7b（Chat 特化）	ベース	0.89	0.66	0.76
	FT	0.84	**0.81**	**0.83**
	RAG	**0.99**	0.62	0.76

[*1] FT はファインチューニングの略
[*2] ベースはベースモデルの略

5.1 各モデルにおけるクローン検出精度

対象の 6 つの LLM について，FEMPDataset のテストデータを用いてファインチューニングと RAG を使用した場合の評価を行った．LLM によってファインチューニングと RAG を使用した場合の性能評価が異なることが分かった．

5.1.1 gpt-4o

ファインチューニング後は，再現率が低下し，適合率が 0.76 から 0.90 に大きく向上した．F1 スコアは 0.84 から 0.89 に向上した．一方で RAG を使用した場合は，再現率が 0.93 から 0.85 に低下し，適合率は 0.83 から 0.84 に向上した．F1 スコアは 0.84 から変化が見られなかった．gpt-4o では RAG による性能の向上は見られず，ファインチューニングによる性能向上が見られた．また，ファインチューニングでは適合率に大きな向上が見られた．

5.1.2 codegemma-7b

ベースモデルの再現率は 0.06 と本研究で計測した LLM のなかで一番低く，ほぼすべてのメソッドペアをクローンペアではないと判定した．ファインチューニング後は再現率が 0.89 に大きく向上し，適合率は 0.89 から 0.80 に低下した．F1 スコアは 0.12 から 0.84 に大きく向上した．RAG を使用した場合は，再現率が 0.06 から 0.41 に向上し，適合率は 0.89 から 0.79 に低下した．F1 スコアは 0.12 から 0.54 に向上した．ファインチューニングで大きな性能向上が見られた．また，RAG を使用した場合も性能向上が見られたが，ファインチューニングに比べると性能向上は小さかった．

5.1.3 codellama-7b

ファインチューニング後は，再現率が 0.51 から 0.95 に大きく向上し，適合率も向上した．F1 スコアは 0.59 から 0.85 に大きく向上した．RAG を使用した場合は，再現率が 0.52 から 0.85 に向上し，適合率は 0.71 から 0.64 に低下した．F1 スコアは 0.59 から 0.73 に向上した．ファインチューニングで大きな性能向上が見られた．また，RAG を使用した場合も性能向上が見られたが，ファインチューニングに比べると性能向上は小さかった．

5.1.4 gemma-2-9b

ファインチューニング後は，再現率が 0.98 から 0.90 に低下し，適合率は 0.65 から 0.82 に大きく向上した．F1 スコアは 0.78 から 0.86 に向上した．RAG を使用した場合は，再現率が 0.98 から 0.74 に低下し，適合率は 0.65 から 0.77 に向上した．F1 スコアは 0.78 から 0.75 に低下した．ファインチューニングで性能向上が見られた一方で，RAG を使用した場合は性能が低下した．

5.1.5 phi-3-small

ファインチューニング後は，再現率が向上し，適合率も 0.74 から 0.85 に大きく向上した．F1 スコアは 0.78 から 0.85 に向上した．RAG を使用した場合は，再現率が 0.82

© 2026 Information Processing Society of Japan

から 0.98 に向上し，適合率は 0.74 から 0.65 に低下した．F1 スコアは 0.78 から 0.79 とほぼ変化しなかった．ファインチューニングで性能向上が見られた一方で，RAG を使用した場合は性能に大きな変化は見られなかった．

5.1.6 llama3.1-7b

ファインチューニング後は，再現率が 0.89 から 0.84 に低下し，適合率は 0.66 から 0.81 に大きく向上した．F1 スコアは 0.76 から 0.83 に向上した．RAG を使用した場合は，再現率が 0.89 から 0.99 に向上し，適合率は 0.66 から 0.62 に低下した．F1 スコアは 0.76 から変化しなかった．ファインチューニングで性能向上が見られた一方で，RAG を使用した場合は性能が変化しなかった．

5.2 ファインチューニングと RAG の比較

対象とした 6 つの LLM について，ファインチューニングを使用した場合はすべてのモデルで性能向上が見られた．ベースモデルで性能が低いモデルほどファインチューニングによる性能向上が大きかった．

一方で，RAG を使用した場合は，どのモデルも F1 スコアにおいてファインチューニングにおける性能向上を上回ることはなかった．codegemma-7b と codellama-7b では，ベースモデルから性能が向上した．一方で，gpt-4o，gemma-2-9b，phi-3-small，llama3.1-7b では，性能に大きな変化は見られなかった．

これらの結果から，ファインチューニングは RAG に比べて Type-4 のコードクローンの検出において性能の向上に効果的であることが分かる．また，RAG を使用した場合は，ファインチューニングに比べて性能向上が見られず，性能が低下するモデルもあった．特に，Chat に特化したモデル（gemma-2-9b，phi-3-small，llama3.1-7b）では性能に大きな変化は見られなかった一方で，Code に特化したモデル（codegemma-7b，codellama-7b）では性能が向上した．5.3 節で詳述するように，Chat に特化したモデルが RAG においてメソッド名や変数名などの識別子に過度に反応し，本来クローンでないペアをクローンと誤判定する傾向が観察された．

本研究の結果から，Type-4 のクローンに対する検出性能向上において，RAG よりもファインチューニングを使用するべきと考える．しかしながら，ファインチューニング時にはモデルのパラメータを VRAM 上に保持する必要があるため，大量の計算資源が必要である．特にパラメータ数が多いモデルにおいては膨大計算資源が必要となりファインチューニングが困難な場合がある．Code に特化した LLM においては RAG による性能向上が一定程度認められたことから，モデルのパラメータ数が膨大で計算資源が限られるなどを理由にファインチューニングが困難である場合には RAG の使用を検討するべきであると考える．

5.3 Code に特化した LLM と Chat に特化した LLM の比較

本研究で使用したモデルのうち，codegemma-7b，codellama-7b は Code に特化したモデルであり，gemma-2-9b，phi-3-small，llama3.1-7b は Chat に特化したモデルである．gpt-4o は Code と Chat のどちらに対しても学習された大規模汎用モデルである．Code に特化したモデルである codegemma-7b と codellama-7b，Chat に特化したモデルである gemma-2-9b，phi-3-small と llama3.1-7b を比較し，ファインチューニングと RAG を使用した場合のクローン検出精度に差があるかを調査した．

ベースモデルの F1 スコアをモデルごとに比較すると，Code に特化したモデル（codegemma-7b: 0.12，codellama-7b: 0.59）に対して，Chat に特化したモデル（gemma-2-9b:0.78，phi-3-small: 0.78，llama3.1-7b: 0.76）の方が高い性能を示している．このことから，ベースモデルにおいて Chat に特化したモデルは Code に特化したモデルに比べて性能が高かった．ファインチューニングを行った場合，Code に特化したモデルと Chat に特化したモデルのどちらも F1 スコアにおいて性能の向上が見られた．Code に特化したモデルでは F1 スコアが codegemma-7b で 0.12 から 0.84（+0.72），codellama-7b で 0.59 から 0.85（+0.26）と大きく向上した．一方，Chat に特化したモデルでは gemma-2-9b で 0.78 から 0.86（+0.08），phi-3-small で 0.78 から 0.85（+0.07），llama3.1-7b で 0.76 から 0.83（+0.07）の向上にとどまった．このように，Code に特化したモデルではファインチューニングによる性能向上が大きかった．

一方で，RAG を使用した場合は，Code に特化したモデルでは性能が向上したが，Chat に特化したモデルでは性能に大きな変化は見られなかった．Code に特化したモデルでは F1 スコアが codegemma-7b で 0.12 から 0.54（+0.42），codellama-7b で 0.59 から 0.73（+0.14）に向上し，Chat に特化したモデルでは F1 スコアが gemma2-9b で 0.78 から 0.75（-0.03），phi-3-small で 0.78 から 0.79（+0.01），llama3.1-7b で 0.76 から 0.76（+0.00）と性能に大きな変化は見られなかった．特に Chat に特化したモデルのうち，phi-3-small と llama3.1-7b は，ファインチューニング後の LLM では正しく判定できたクローンでないペアの多くをクローンペアと誤って判定した．テストデータに含まれるクローンでないペア 90 個のうちクローンペアと誤って判定されたデータ数は，codegemma-7b では 0 個，codellama-7b では 10 個であったのに対し，phi-3-small では 22 個，llama3.1-7b では 40 個であった．Chat に特化した phi-3-small と llama3.1-7b で誤ってクローンと判定したペアの和集合を取ると 17 ペアであった．この 17 ペアに対して，RAG によって生成された外部データを含んだプロンプトを目視で確認したところ，17 ペアの内 8 ペアで外

© 2026 Information Processing Society of Japan

部データのうちクローンであるペアに判定対象のペアに含まれるメソッド名や変数名が多く含まれている事を確認した．これが phi-3-small と llama3.1-7b において，誤判定が増加した原因の 1 つであると考える．

以上の結果から，LLM を用いた Type-4 のクローン検出において，RAG を使用した際に全モデルに同一の外部データが提供されるにもかかわらず，Chat に特化した LLM は外部データに含まれるメソッド名や変数名などの識別子の一致に過度に依存し，コードの構文的・機能的類似性を適切に評価できずに誤判定を行う可能性があると考える．一方，Code に特化した LLM は識別子名の表面的な類似性に惑わされることなく，より本質的なコード構造に基づいて判定を行っていると推察される．

本節で述べた，ファインチューニングと RAG を行った場合の検出精度と Code に特化した LLM と Chat に特化した LLM の性能差は，LLM を用いた Type-4 のクローン検出においてモデルの選択と最適化において重要な情報であると考える．

6. 妥当性への脅威

本研究で使用した FEMPDataset に含まれるメソッドペアは，計 2,194 ペアであり，ファインチューニングで使用する訓練データは 1,755 ペア，評価で使用するテストデータは 219 ペアである．訓練データとして使用したメソッドペア数が少ないため，ファインチューニングの性能が十分に評価できていない可能性がある．RAG を使用した場合も同様に，外部データとして使用したメソッドペア数が少ないため，性能が十分に評価できていない可能性がある．

また，本研究では，FEMPDataset に限定してファインチューニングと RAG の効果を評価した．SemanticCloneBench や BigCloneBench などの分布の異なる Type-4 クローンのデータセットに対してファインチューニングと RAG を評価した場合，結果が異なる可能性がある．また，本研究では，ファインチューニングと RAG を使用した場合の性能を比較したが，他の手法との比較を行っていないため，ファインチューニングと RAG が他の手法よりも優れているかどうかは不明である．

7. むすび

本研究では，5 つの LLM に対してファインチューニングと RAG を用いたコードクローン検出性能の向上を試みた．ファインチューニングを使用した場合にはすべてのモデルで F1 スコアが 7〜72%の範囲で向上した．一方で RAG を使用した場合には，Code に特化した LLM で F1 スコアが 14〜42%の範囲で向上したが，Chat に特化した LLM では精度の向上は認められなかった．これらの結果から，ファインチューニングは RAG に比べて性能向上に効果的であると考えられる．また，RAG を使用した場合の結果からクローン検出において Chat に特化した LLM は識別子名などの自然言語に対して敏感である可能性があることが分かった．

今後の課題として，パラメータの多いモデルに対してファインチューニングを行い，性能向上が見られるかを調査することがあげられる．その際は，FEMPDataset の規模が小さいという制約を踏まえ，より大きなデータセットを使用する必要があると考える．また，本研究では LoRA で一部パラメータのみを更新したのに対して，計算資源とのトレードオフを考慮しつつパラメータの更新範囲の最適化も検討する．

謝辞 本研究の一部は，JSPS 科研費（24H00692，21K18302，21H04877，23K24823，22K11985）を得て行われた．

参考文献

[1] Baxter, I., Yahin, A., Moura, L., Sant'Anna, M. and Bier, L.: Clone detection using abstract syntax trees, *Proc. ICSM*, pp.368–377 (online), DOI: 10.1109/ICSM.1998.738528 (1998).

[2] Mondal, M., Roy, C. and Schneider, K.: A Fine-Grained Analysis on the Inconsistent Changes in Code Clones, *IEEE ICSME*, pp.220–231 (online), DOI: 10.1109/ICSME46990.2020.00030 (2020).

[3] Kamiya, T., Kusumoto, S. and Inoue, K.: CCFinder: A multilinguistic token-based code clone detection system for large scale source code, *IEEE Trans. Software Engineering*, Vol.28, No.7, pp.654–670 (online), DOI: 10.1109/TSE.2002.1019480 (2002).

[4] Roy, C. and Cordy, J.: NICAD: Accurate Detection of Near-Miss Intentional Clones Using Flexible Pretty-Printing and Code Normalization, *IEEE 16th ICPC*, pp.172–181 (online), DOI: 10.1109/ICPC.2008.41 (2008).

[5] Nakagawa, T., Higo, Y. and Kusumoto, S.: NIL: Large-scale detection of large-variance clones, *Proc. 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, Association for Computing Machinery, pp.830–841 (online), DOI: 10.1145/3468264.3468564 (2021).

[6] Zhang, J. et al.: A Novel Neural Source Code Representation Based on Abstract Syntax Tree, *IEEE/ACM 41st ICSE*, pp.783–794 (online), DOI: 10.1109/ICSE.2019.00086 (2019).

[7] Wang, W. et al.: Detecting Code Clones with Graph Neural Network and Flow-Augmented Abstract Syntax Tree, *IEEE 27th SANER*, pp.261–271 (online), DOI: 10.1109/SANER48275.2020.9054857 (2020).

[8] Feng, Z. et al.: CodeBERT: A Pre-Trained Model for Programming and Natural Languages, *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp.1536–1547 (online), DOI: 10.18653/v1/2020.findings-emnlp.139 (2020).

[9] Pinku, S.N., Mondal, D. and Roy, C.K.: On the Use of Deep Learning Models for Semantic Clone Detection, *IEEE ICSME*, pp.512–524 (online), DOI: 10.1109/ICSME58944.2024.00053 (2024).

[10] OpenAI: GPT-4 Technical Report (2023).

© 2026 Information Processing Society of Japan

[11] Sam, K. and Vavekanand, R.: Llama 3.1: An In-Depth Analysis of the Next Generation Large Language Model (2024).
[12] Touvron, H. et al.: Llama 2: Open Foundation and Fine-Tuned Chat Models (2023).
[13] Rozière, B. et al.: Code Llama: Open Foundation Models for Code (2023).
[14] Abdin, M. et al.: Phi-3 Technical Report: A Highly Capable Language Model Locally on Your Phone (2024).
[15] Gemma Team: Gemma (2024).
[16] CodeGemma Team: CodeGemma: Open Code Models Based on Gemma (2024).
[17] Zhao, W. et al.: A Survey of Large Language Models (2023).
[18] Dou, S. et al.: Towards Understanding the Capability of Large Language Models on Code Clone Detection: A Survey (2023).
[19] Higo, Y.: Dataset of Functionally Equivalent Java Methods and Its Application to Evaluating Clone Detection Tools, *IEICE Trans. Inf. and Syst.*, Vol.E107-D, No.6, pp.751–760 (online), DOI: 10.1587/transinf.2023EDP7268 (2024).
[20] Roy, C. and Cordy, J.: A survey on software clone detection research, *Queen's School of Computing TR*, Vol.541, No.115, pp.64–68 (2007).
[21] Roy, C., Cordy, J. and Koschke, R.: Comparison and evaluation of code clone detection techniques and tools: A qualitative approach, *Science of Computer Programming*, Vol.74, No.7, pp.470–495 (online), DOI: 10.1016/j.scico.2009.02.007 (2009).
[22] Vaswani, A. et al.: Attention is all you need, *Proc. 31st International Conference on Neural Information Processing Systems*, pp.6000–6010 (2017).
[23] Hu, E.J. et al.: LoRA: Low-Rank Adaptation of Large Language Models, *Proc. International Conference on Learning Representations* (2022).
[24] Rajbhandari, S., Rasley, J., Ruwase, O. and He, Y.: ZeRO: Memory optimizations Toward Training Trillion Parameter Models, *Proc. SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp.1–16 (online), DOI: 10.1109/SC41405.2020.00024 (2020).
[25] Lewis, P. et al.: Retrieval-augmented generation for knowledge-intensive NLP tasks, *Proc. 34th International Conference on Neural Information Processing Systems*, pp.793–808 (2020).
[26] Lu, S., Duan, N., Han, H., Guo, D., Hwang, S.-W. and Svyatkovskiy, A.: ReACC: A Retrieval-Augmented Code Completion Framework, *Proc. 60th Annual Meeting of the Association for Computational Linguistics* (*Volume 1: Long Papers*), Muresan, S., Nakov, P. and Villavicencio, A., (Eds.), pp.6227–6240 (online), DOI: 10.18653/v1/2022.acl-long.431 (2022).
[27] Ovadia, O., Brief, M., Mishaeli, M. and Elisha, O.: Fine-Tuning or Retrieval? Comparing Knowledge Injection in LLMs, *Proc. 2024 Conference on Empirical Methods in Natural Language Processing*, pp.237–250 (online), DOI: 10.18653/v1/2024.emnlp-main.15 (2024).
[28] Khajezade, M., Wu, J.J., Fard, F.H., Rodríguez-Pérez, G. and Shehata, M.S.: Investigating the Efficacy of Large Language Models for Code Clone Detection, *IEEE/ACM 32nd ICPC*, pp.161–165 (online), DOI: 10.1145/3643916.3645030 (2024).
[29] Inoue, R. and Higo, Y.: Improving Accuracy of LLM-based Code Clone Detection Using Functionally Equivalent Methods, *IEEE/ACIS 22nd SERA*, pp.24–27 (online), DOI: 10.1109/SERA61261.2024.10685589 (2024).
[30] Svajlenko, J., Islam, J., Keivanloo, I., Roy, C. and Mia, M.: Towards a Big Data Curated Benchmark of Inter-project Code Clones, *Proc. ICSME*, pp.476–480 (online), DOI: 10.1109/ICSME.2014.77 (2014).
[31] Krinke, J. and Ragkhitwetsagul, C.: BigCloneBench Considered Harmful for Machine Learning, *IEEE 16th IWSC*, pp.1–7 (online), DOI: 10.1109/IWSC55060.2022.00008 (2022).
[32] Al-Omari, F., Roy, C.K. and Chen, T.: SemanticCloneBench: A Semantic Code Clone Benchmark using Crowd-Source Knowledge, *IEEE 14th IWSC*, pp.57–63 (online), DOI: 10.1109/IWSC50091.2020.9047643 (2020).
[33] Alam, A.I., Roy, P.R., Al-Omari, F., Roy, C.K., Roy, B. and Schneider, K.A.: GPTCloneBench: A comprehensive benchmark of semantic clones and cross-language clones using GPT-3 model and SemanticCloneBench, *IEEE ICSME*, pp.1–13 (online), DOI: 10.1109/ICSME58846.2023.00013 (2023).

井上 龍太郎

2024 年大阪大学基礎工学部情報科学科卒業．現在，同大学大学院情報科学研究科博士前期課程に在学中．コードクローン分析手法に関する研究に従事．

肥後 芳樹 （正会員）

2002 年大阪大学基礎工学部情報科学科中退．2006 年同大学大学院博士後期課程修了．2007 年同大学院情報科学研究科コンピュータサイエンス専攻助教．2015 年同准教授．2022 年同教授．博士（情報科学）．ソースコード分析，特にコードクローン分析，リファクタリング支援，ソフトウェアリポジトリマイニングおよび自動プログラム修正に関する研究に従事．日本ソフトウェア科学会．本会シニア会員．

© 2026 Information Processing Society of Japan