特別研究報告

題目

機能等価メソッドデータセットを利用した LLM によるコードクローン検出の精度向上

指導教員
肥後 芳樹 教授

報告者
井上 龍太郎

令和 6 年 2 月 7 日

大阪大学 基礎工学部 情報科学科

令和 5 年度 特別研究報告

機能等価メソッドデータセットを利用した LLM によるコードクローン検出の精度向上

井上 龍太郎

内容梗概

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．コードクローンは，プログラムのコピーアンドペーストを通して生み出されてしまう．コードクローンに対しては，一貫した変更が必要になる場合が多く，保守性を低下させるほか，変更漏れによるバグの拡散の原因となる．このため，コードクローンは効率的に検出し，必要に応じてリファクタリングする必要がある．

また，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のことである．モデルは年々大規模化し，学習するコーパスのサイズも大きくなっており，これらのモデルは高い言語能力を持っている．ChatGPT，Google Bard，Bing AI，などのサービスは LLM を用いて構築されている．

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．既存の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ほど，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精度での検出を実現しているが，検出精度は十分に高いとは言えず，改善の余地がある．

そこで本研究では，ファインチューニングを用いて LLM によるコードクローン検出の精度向上を試みる．対象とする LLM は GPT-3.5-turbo で，ファインチューニングは OpenAI が提供する API を用いて行った．ファインチューニングの訓練用のデータセットには FEMPDataset を，ファインチューニング後の性能評価には BigCloneBench のうち 4,000 個を抽出し用いた．ファインチューニング前後のモデルについて，FEMPDataset のテストデータと BigCloneBench を用いて検出精度を比較し，構文的な類似度の低いコードクローンに対する検出精度の変化を観察した．

評価実験では，クローンペアを含む複数のメソッドペアを LLM に入力しコードクローンであるかどうかを判定した．ファインチューニング後のモデルは，訓練データである FEMPDataset に対して精度の向上が見られたが，BigCloneBench に対しては精度の向上が見られなかった．

主な用語

コードクローン
LLM (大規模言語モデル)
ファインチューニング
FEMPDataset
BigCloneBench

目 次

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 [19]．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，一貫性のない変更は変更もれによって欠陥の原因となる [5]．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．

コードクローンの検出を行うため，多くのツールが開発されてきた．既存のコードクローン検出ツールとして CCFinder[3]，NiCad[9]，NIL[6] などのツールが挙げられる．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の検出では高い精度を示すが，構文的な類似度の低いコードクローンの検出では精度が低いことが課題として挙げられる．

一方で，LLM (大規模言語モデル) 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，既存のツールよりも高い精度での検出を実現している．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．GPT といったモデルを Web サイトやアプリケーションを通して利用することができる ChatGPT は公開から約 2 か月でユーザー数が 1 億人を達するなど，多くのユーザーに利用されている．また，Meta が開発した Llama2[15] は商用利用が可能なオープンソースとして注目されており，それらをファインチューニングした Alpaca[13] や Vicuna[18]，code-llama[11] などのモデルが開発されている．その他にも，Google が開発した Gemini[14]，Anthropic が開発した Claude2[1] など，多くのモデルが日々開発されている [17]．

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 [2]．NiCad や Oreo といった既存の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 GPT-3.5-trubo や GPT-4，Llama2-Chat-7B といった 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精度での検出を実現している．しかしながら，GPT-3.5-turbo や GPT-4 は構文的な類似度の低いコードクローンに対する LLM の検出精度は十分に高いとは言えず，改善の余地がある．

そこで，本研究では，多方面で今後使用が期待されている LLM にファインチューニングを行い，構文的な類似度の低いコードクローンの検出の精度の向上を試みる．対象とする LLM は GPT-3.5-turbo である．GPT-3.5-turbo のファインチューニングは OpenAI が提供する API を用いて行い，必要に応じてエポック数，バッチサイズ，学習率などのハイパーパラメータを変更する．ファインチューニングでは，異構造で機能等価なコードクローンを集

めた FEMPDataset を用いた．FEMPDataset は訓練データ，検証データ，テストデータの 3 つに分割し，学習と FEMPDataset に対する検出精度向上の評価を行う．また，評価用のデータセットとして，大規模なコードクローン検出のベンチマークである BigCloneBench を用いて，構文的な類似度の低いコードクローンに対する検出精度を比較する．

以下，第 2 章では，本研究の準備として，コードクローンの定義や，LLM によるコードクローン検出に関する先行研究，実験で使用するデータセットについて述べる．第 3 章では，本研究の実験方法について述べる．第 4 章では，実験結果について述べる．第 5 章では，実験結果を踏まえた考察を行う．第 6 章では，本研究のまとめと今後の課題について述べる．

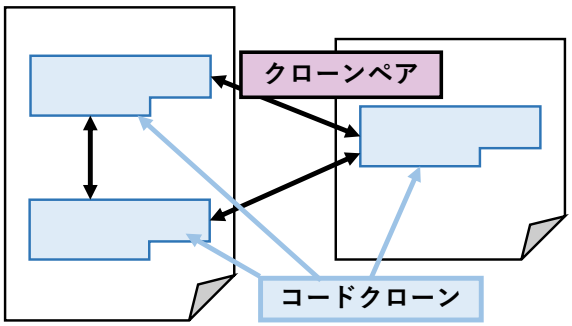


図 1: コードクローンとクローンペア

2　準備

この章では，本研究の背景として，コードクローンの定義や，LLM によるコードクローン検出に関する先行研究，実験に使用するデータセットについて述べる．

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 [19]．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によってプログラム内に作られる [8]．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 (図 1)．

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 [10]．

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，同じ処理を行うコードクローン．

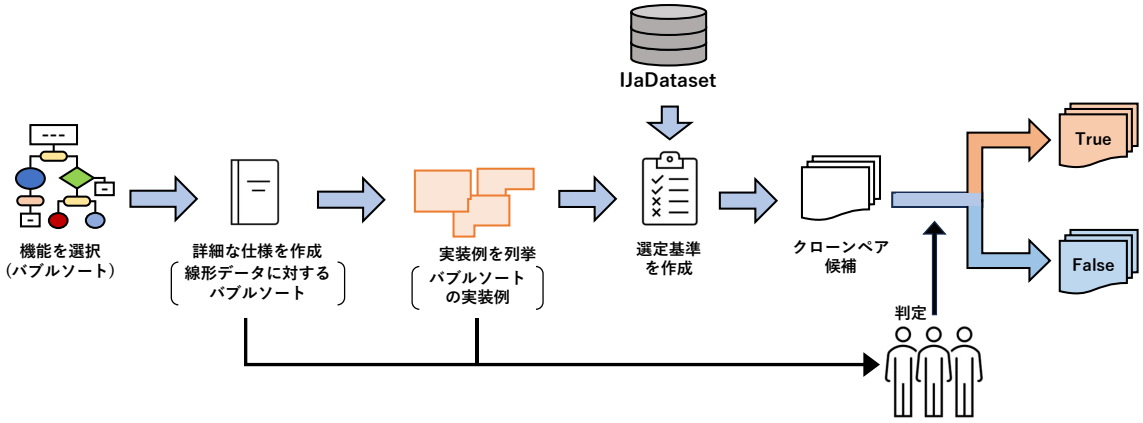


図 2: BigCloneBench のデータセット作成手順

2.2 BigCloneBench

BigCloneBench[12] 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．BigCloneBench では，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 の構築は以下の手順で行われている．また，その様子を図 2 に示す．

STEP-1：対象とする機能の選択

バブルソートやファイルのコピーや削除など，抽出するメソッドの機能を選択する．

STEP-2：抽出するメソッドの機能の詳細な仕様書の作成

インターネットのディスカッションや API ドキュメントなどを参考に，対象の機能が満たすべき仕様を作成する．

STEP-3：実装例を列挙

対象の機能を持つと思われる実装例について考えられるパターンを列挙する．

STEP-4：選定基準を作成

STEP1 で作成した仕様，STEP2 で作成した実装例をもとに，キーワードやソースコードパターンを用いた選定基準を作成する．

STEP-5：対象の機能を持つと思われるメソッドを機械的に抽出

選定基準をもとに，対象の機能を持つと思われるメソッドを機械的に抽出する．

STEP-6：最終的な判断を目視で判定

機械的に抽出したメソッドに対して，人が目視で対象の機能を持つか判定する．

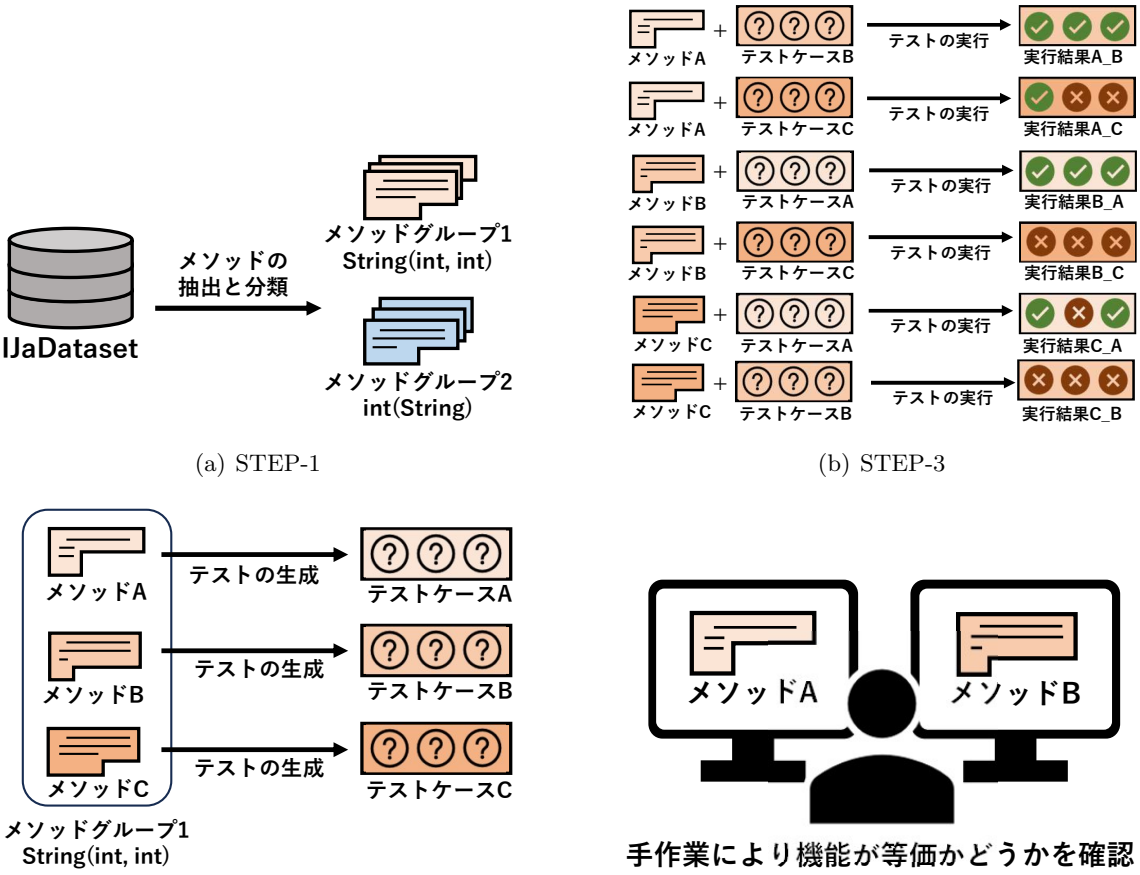
また，BigCloneBench では Type-1，Type-2 に所属しない，Type-3，Type-4 のコードクローンに対して構文的な類似度を用いて詳しい分類を行っている (表 1)．具体的な類似度の数値は，BigCloneBench に作成においてトークンと行の 2 つの視点から GNU Diffutils と呼ばれるプログラムを用いて計算されたもので，メソッドペア間のファイルの差異をもとに計算されている．データセットで定義されるタイプとクローンペア数は表 2 の通りである．

タイプ	**類似度**
Very Strongly T3 (VST3)	[0.9, 1.0)
Strongly T3 (ST3)	[0.7, 0.9)
Moderately T3 (MT3)	[0.5, 0.7)
Weakly T3/Type-4 (WT3/T4)	[0.0, 0.5)

表 1: BigCloneBench のより詳しい分類とコードクローンの定義

タイプ	**T1**	**T2**	**VST3**	**ST3**	**MT3**	**WT3/T4**
クローンペア数	35,787	4,573	4,156	14,997	79,756	7,729,291
						計 7,868,560

表 2: BigCloneBench のタイプごとメソッドペア数



(a) STEP-1

(b) STEP-3

(c) STEP-2

(d) STEP-4

図 3: 機能等価な Java メソッドのペアを得るための手順

2.3 FEMPDataset

FEMPDataset[20] 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．FEMP-Dataset の構築は以下の手順で行われている．また，その様子を図 3 に示す．

STEP-1：対象プロジェクトに含まれるメソッドを返り値の型と引数の型を基に分類.

STEP-2：各メソッドからテストケースの生成.

STEP-3：テストを相互実行することにより機能等価メソッドペアの候補を取得.

STEP-4：機能等価メソッドペアの各候補を閲覧し，真に機能等価であるかを判定.

この手順を踏むことで，真に機能等価であるメソッドペアを取得している．FEMPDataset 内には，機能等価であるメソッドとそうでないメソッドが含まれており，コードクローン検

出ツールにおいてこれらは正しく分類，検出される必要がある．

先述の BigCloneBench とは異なる点は以下の 3 つが挙げられる．1 つ目は機能等価の確認をテストケースを用いた動的な観点から行っている点である．2 つ目は，BigCloneBench のデータセットは構文的な類似度が高いものから低いものまで，全てのコードクローンを対象としているのに対し，FEMPDataset は異構造で機能等価なメソッド，すなわち構文的な類似度が近いコードクローンを対象としている点である．3 つ目は，FEMPDataset はメソッド全体の機能等価を見ているのに対し，BigCloneBench はメソッド全体が機能等価なのではなくその一部に機能が同等と思われるコードを含む点である．

2.4 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 と呼ばれるモデルが発表され [16]，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．

近年では，LLM を利用した ChatGPT，Google Bard，Bing AI，などのサービスが注目を集めている．また，自然言語処理の分野で高い成果を上げている．その中でも OpenAI が開発した GPT-4 は，専門分野や学術分野に関するテストにおいて，人間と同等レベルの成績を収めている [7]．また，Meta が開発した Llama2[15] は商用利用が可能なオープンソースとして注目されている．Alpaca[13] や Vicuna[18] は，Llama2 の前身のモデルである Llama をファインチューニングを用いて精度向上を行ったモデルであり，Llama2 よりも高い性能を示している．さらに，Meta は Llama2 をファインチューニングし，プログラミングタスクに特化した code-llama[11] も発表している．

2.5 検出精度指標

コードクローン検出や LLM の性能指標には Recall，Precision，Accuracy 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通りである．

Recall

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．

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

Precision

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．

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

Accuracy

正しい判定をしたメソッドペアの割合．

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

また，TP (True Positive)，FP (False Positive)，FN (False Negative)，TN (True Negative) は以下を表す．

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ないメソッドのうち，コードクローンでない判定されたメソッドペア数．

TN：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．

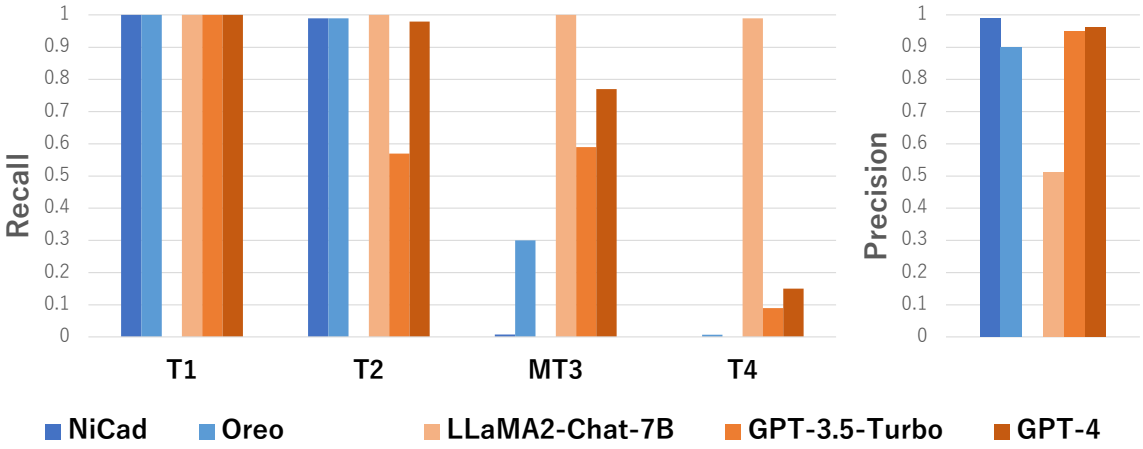


図 4: 既存のツールと LLM を用いた場合のコードクローン検出性能の可視化

2.6 BigCloneBench と LLM に関する先行研究

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，Shihan Dou らの研究 [2] を紹介する．Dou らの研究では，単一のプロンプトを入力に用いた LLM によるコードクローン検出を行い，BigCloneBench を用いて性能評価を行っている．また，LLM を用いない既存のツールと LLM を用いた場合の性能を BigCloneBench を用いて比較している．結果は表 3 の通りである．なお，”-”で表されている部分はそのツールが有効な結果を返すことができなかったことを表している．これらのモデルからいくつかのモデルを選択し，検出精度の比較を可視化した．結果は図 4 の通りである．

結果を見ると，構文的な類似度の高いコードクローンは既存のツールの方が高い Recall を示しているが，構文的な類似度の低いコードクローンは LLM の方が高い Recall を示している．

また，Type-4 に属する構文的な類似度の低いコードクローンに対しては，GPT-3.5-turbo と GPT-4 の 2 つの LLM は既存のツールよりも高い Recall を示し，同時に高い Precision を示している．これは LLM が既存のツールよりも構文的な類似度に依存しないコードクローン検出を行える可能性を示唆している．しかし，GPT-3.5-turbo と GPT-4 の Type-4 に対する Recall は Type-1，Type-2 に比べると低く十分な検出精度であるとは言い難い．また，Llama2-chat-7B は Recall が高い一方で，Precision がかなり低くなっており，クローンペアとそうでないペアを区別できず，ほぼ全てのメソッドペアをクローンペアと認識してしまっていることがわかる．そこで，本研究ではファインチューニングを用いて LLM によるコードクローン検出の精度の向上を目指す．

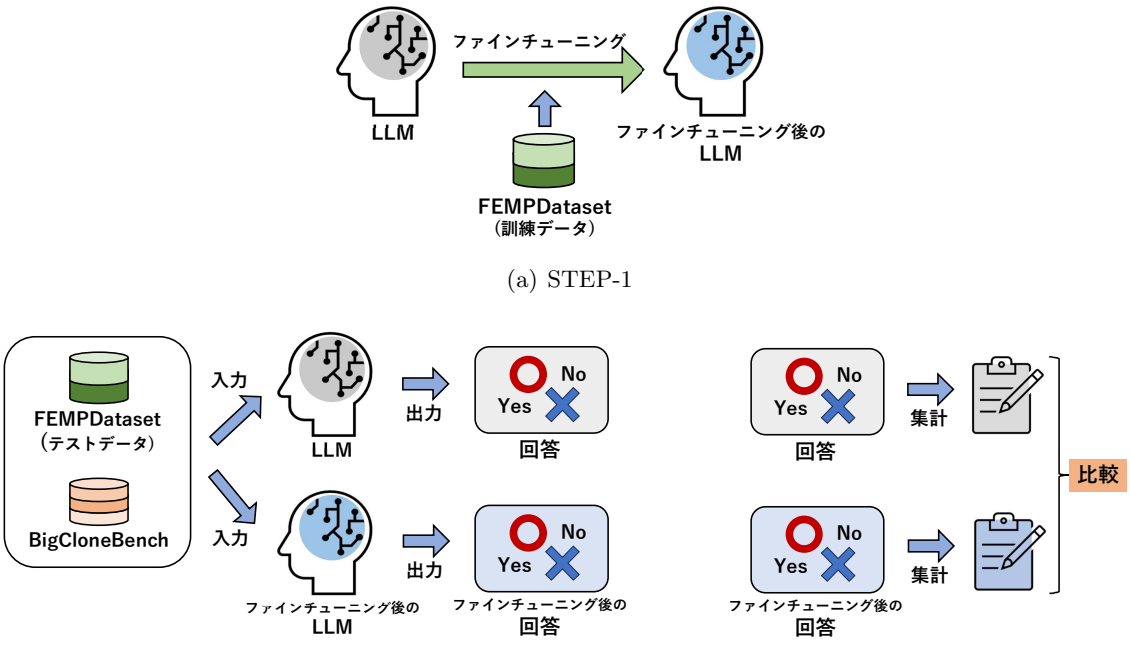
表 3: 既存のツールと LLM を用いたツールのコードクローン検出性能

(a) LLM を用いないツールの検出精度

Methods	**T1**	**T2**	**VST3**	**ST3**	**MT3**	**T4**	**Precision**
SourcererCC	1.00	0.97	0.93	0.60	0.05	0.00	0.98
CCFinder	1.00	0.93	0.62	0.15	0.01	0.00	0.72
NiCad	1.00	0.99	0.98	0.93	0.01	0.00	0.99
Deckard	0.60	0.58	0.62	0.31	0.12	0.01	0.35
CCAligner	1.00	0.99	0.97	0.70	0.10	-	0.80
Oreo	1.00	0.99	1.00	0.89	0.30	0.01	0.90
LVMapper	0.99	0.99	0.98	0.81	0.19	-	0.58
NIL	0.99	0.96	0.93	0.67	0.10	-	0.94

(b) LLM を用いたツールの検出精度

Methods	**T1**	**T2**	**VST3**	**ST3**	**MT3**	**T4**	**Precision**
Llama-7B	-	-	-	-	-	-	-
Llama2-7B	-	-	-	-	-	-	-
Aplaca-7B	0.76	0.93	0.65	0.87	0.89	0.71	0.55
Vicuna-7B	0.42	0.30	0.72	0.74	0.90	0.60	0.45
Llama2-Chat-7B	1.00	1.00	1.00	1.00	0.99	-	0.51
Falcon-Instruct-7B	1.00	1.00	1.00	1.00	0.99	-	0.48
MPT-Instruct-7B	0.47	0.08	0.23	0.33	0.28	0.15	0.74
StarChat-β-16B	0.93	0.49	0.42	0.43	0.26	0.37	0.62
GPT-3.5-Turbo	1.00	0.57	0.85	0.78	0.59	0.09	0.95
GPT-4	1.00	0.98	0.99	0.94	0.77	0.15	0.96



(a) STEP-1

(b) STEP-2

(c) STEP-3

図 5: 実験の手順

3　実験方法

この章では，本研究の具体的な実験方法について述べる．

実験手順

実験では，ファインチューニングを用いて構文的な類似度の低いコードクローンに対する検出精度の向上を試みる．実験は以下の手順で行う．また，実験の手順を図 3 に示す．

1. **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．

2. **LLM の実行**
   ファインチューニング前と後の LLM に対して，FEMPDataset のテストデータ，BigCloneBench のメソッドペアを与え，回答を”Yes”または”No”で得る．

3. **性能評価**
   ファインチューニング前後の LLM の回答を集計し，性能の比較を行う．

対象

また，対象とする LLM は以下の通りである。

- GPT-3.5-turbo

3.1 ファインチューニング

LLM に対してファインチューニングを行う．

3.1.1 ファインチューニング用のデータセット

ファインチューニングには FEMPDataset を用いる．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．ファインチューニングのため，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．分割後の各データの役割は以下の通りである．

訓練データ

ファインチューニングの訓練時に LLM のパラメータを変更するために用いる．

検証データ

ファインチューニングの訓練時に LLM の学習が適切に行われているか，過学習が起きていないかを確認するために用いる．

テストデータ

ファインチューニング後の LLM の性能評価を行うために用いる．

各データのメソッドペア数の内訳は表 4 の通りである．

3.1.2 GPT-3.5-turbo のファインチューニング

GPT-3.5-turbo のファインチューニングは，OpenAI API を用いて行う．ファインチューニングでは，エポック数，バッチサイズ，学習率の 3 つのハイパーパラメータを必要に応じて変更する．

	訓練データ	検証データ	テストデータ
メソッドペア数	1,755	220	219
			計 2,194

表 4: 分割後の各データのメソッドペア数の内訳

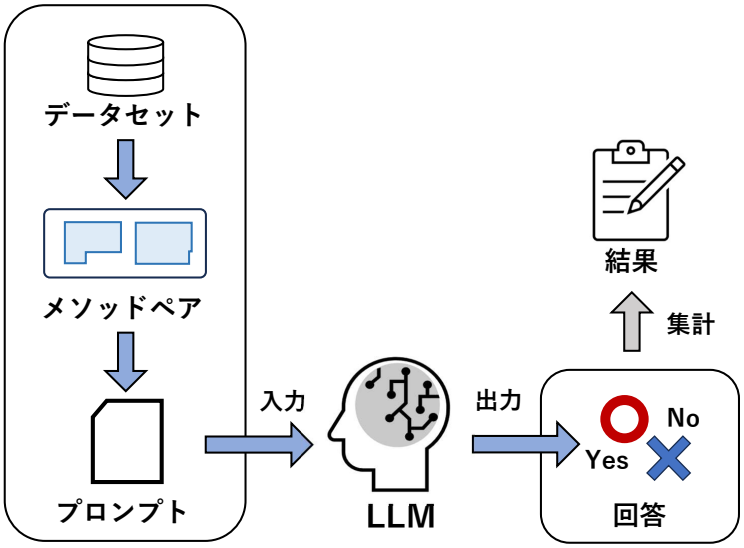


図 6: LLM による判定の手順

3.2 LLM の実行

ファインチューニング前とファインチューニング後の LLM の性能評価は以下の手順で行う．なお，判定の手順を図 6 に示す．

1. **BigCloneBench のメソッドペアを抽出**
   BigCloneBench から T4 に分類されるクローンペアと，クローンではないペアを抽出する．この 2 種類のメソッドペアを正しく分類することができるかを評価する．

2. **プロンプトの作成**
   抽出したメソッドペアはクローンペアであるかどうかを質問するプロンプトに変換し，LLM に入力する形式にする．

3. **プロンプトの入力，回答の取得**
   ファインチューニング前の LLM とファインチューニング後の LLM に対して，作成したプロンプトを入力し，”Yes” または”No” の回答を得る．

4. **回答の集計，性能評価**
   回答を集計し，ファインチューニング前後の LLM の性能を数値として比較する．

3.2.1 評価用データセット

性能評価は BigCloneBench を用いて行う．BigCloneBench のデータセットは 2.2 節で述べたように，構文的な類似度によって 6 つに分類されている．このうち，構文的な類似度の低い T4 に分類されるクローンペアと，クローンではないペアを用いて実験を行う．この 2 種類のメソッドペアを正しく分類することができるかを評価する．

データセットに含まれる T4 のメソッドペア数は 7,729,291 と膨大である．このため，データセットからランダムに 2,000 個のメソッドペアを抽出し，実験に用いることとした．また，クローンではないペアも同様に 2,000 個抽出し，実験に用いる．したがって，実験に用いるデータセットのメソッドペア数の内訳は表 5 の通りである．

ただし今回，メソッドペアの 2 つを Llama のトークナイザーでトークナイズした際のトークン数の合計が 10,000 トークンを超えるメソッドペアは，メソッドペア抽出の際に対象から除外した．これは，Llama を実行する際に 10,000 トークンを超えるプロンプトを入力すると，GPU サーバーがメモリ不足で動かないためである．メソッドの抽出はランダム性を確保するために BigCloneBench のデータセットに格納されているデータからランダムにメソッドを順に取り出し，10,000 トークン以内のメソッドペアであれば，評価用のメソッドペアの一つとして採用した．これを 2,000 ペア取り出すまで繰り返した．クローンペアは 2,068 ペアを抽出し 2,000 ペアを，クローンではないペアは 2,037 ペアを抽出し 2,000 ペアを最終的に取得した．

	クローンペア	クローンではないペア
メソッドペア数	2,000	2,000
		計 4,000

表 5: 評価用データセットのメソッドペア数の内訳

```
System: Always answer with only 'yes' or 'no' only.

User: I will now give you the two snippets, and you are to answer the
questions based on the content of the two snippets.

Snippet 1:
double kroneckerDelta(double i,double j){
  if (Double.isNaN(i) || Double.isNaN(j))   return Double.NaN;
  if (i == j)   return 1;
 else   return 0;
}

Snippet 2:
double eq(double a,double b){
  double result=0;
  if (a == b)   result=1;
  if ((Double.isNaN(a)) || (Double.isNaN(b)))   result=Double.NaN;
  return result;
}

 Please analyze the two code snippets and determine if they are code
clones. Respond with 'yes' if the code snippets are clones or 'no' if not.
```

図 7: LLM への入力プロンプト

3.2.2 プロンプト

LLM に入力するプロンプトは図 7 の通りである．黄色の部分，緑色の部分にそれぞれメソッドペアを構成する 2 つのメソッドのプログラムをそれぞれ入力すする．

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，提示した 2 つのメソッドが，クローンペアであるかを判定をするように指示を記述する．

3.3 性能評価

性能評価は，FEMPDataset のテストデータ計 219 ペアと BigCloneBench のメソッドペア計 4,000 ペアに対して，Recall，Precision，Accuracy の 3 値を計算し性能を評価する．

4　実験結果

この章では，実験の結果について述べる．実験結果は以下の 3 つの項目に分けて示す．

1. FEMPDataset によるファインチューニングの過程
2. FEMPDataset に対するファインチューニングの評価
3. BigCloneBench に対するファインチューニングの評価

4.1　FEMPDataset によるファインチューニングの過程

FEMPDataset によるファインチューニングの過程を示す．

4.1.1　GPT-3.5-turbo のファインチューニングの過程

GPT-3.5-turbo に対してファインチューニングを行った．各エポックの損失の振れ幅が安定するようにバッチサイズは 32 とした．8 エポックまで学習した際の損失関数の値の推移を図 8 に示す．図 8 の横軸はエポック，縦軸は損失関数の値を示している．1 エポックは全ての訓練データを使用して学習を行ったことを指す．また，バッチサイズとは 1 回の学習で使用するデータの数を指す．エポック数が 3 を超えたあたりで，検証データの損失が上昇していることから，過学習が発生していることがわかる．よって，エポック数が 3 付近で損失関数の値が収束していることがわかる．

よって，最終的なエポック数は 3 とし，学習を行った．学習した際の損失関数の値の推移を図 9 に示す．

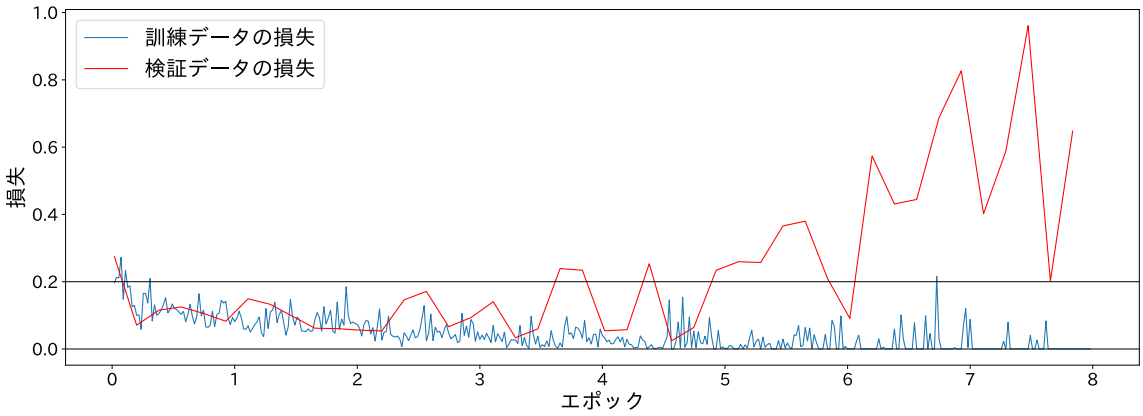


図 8: GPT-3.5-turbo のファインチューニングの損失関数の推移 (8 エポック・32 バッチ)

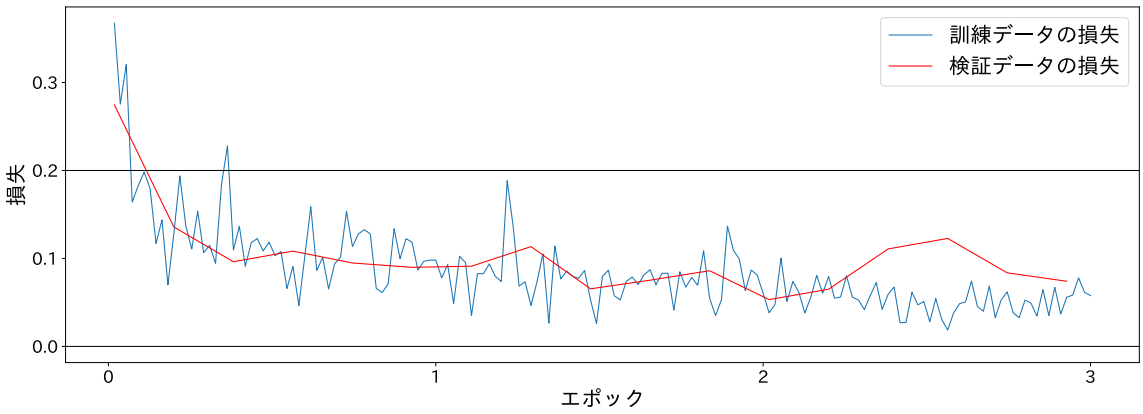


図 9: GPT-3.5-turbo のファインチューニングの損失関数の推移 (3 エポック・32 バッチ)

4.2 FEMPDataset によるファインチューニングの評価

FEMPDataset によるファインチューニングの評価を行った.

4.2.1 GPT-3.5-turbo の FEMPDataset による評価

GPT-3.5-turbo のファインチューニング前後の評価比較を FEMPDataset のテストデータを用いて行った．テストデータは計 219 のメソッドペアから構成される.

ファインチューニング前の GPT-3.5-turbo，ファインチューニング後の GPT-3.5-turbo, GPT-4-turbo の評価結果を表 6 に示す．この結果をもとに，Recall，Precision，Accuracy の変化を計算した．結果は表 7 の通りである．また，以上の結果をグラフにまとめたものを図 10 に示す.

ファインチューニング後の GPT-3.5-turbo とファインチューニング以前の GPT-3.5-turbo を比較すると，Precision が大きく向上していることがわかる．また，Recall はほぼ横ばいとなっている．このことから，ファインチューニングによってクローンでないペアを正しく判定するようになったことがわかる.

モデル名	正解	LLM による回答 Yes	No	回答なし
GPT-3.5-turbo	Yes	108	21	0
	No	48	42	0
ファインチューニング後の GPT-3.5-turbo	Yes	107	22	0
	No	20	70	0
GPT-4-turbo	Yes	118	11	0
	No	41	49	0

表 6: GPT-3.5-turbo の FEMPDataset による評価

モデル名	Recall	Precision	Accuracy
GPT-3.5-turbo	0.84	0.69	0.69
ファインチューニング後の GPT-3.5-turbo	0.83	0.84	0.81
GPT-4-turbo	0.92	0.69	0.76

表 7: ファインチューニング前後の GPT-3.5-turbo と GPT-4-turbo の評価

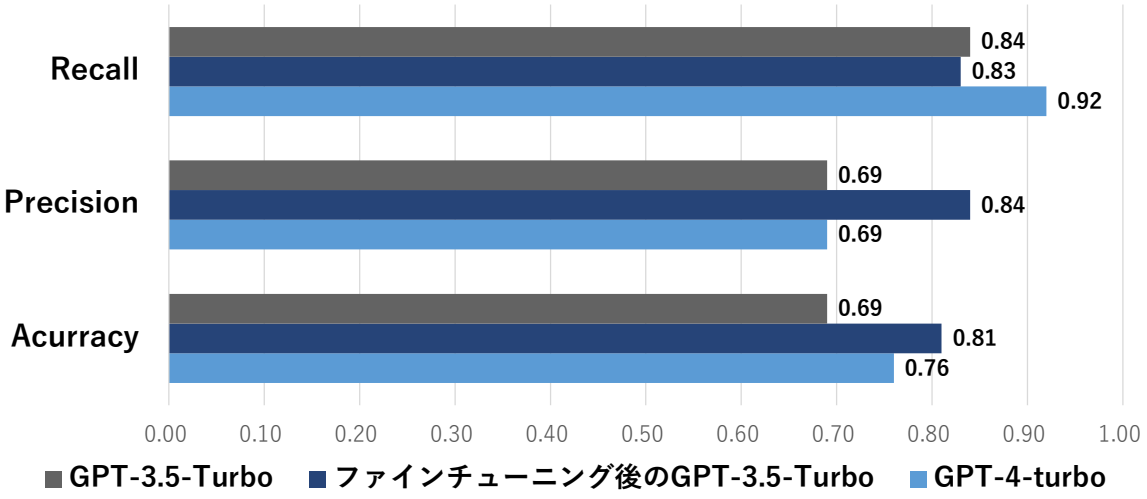


図 10: FEMPDataset によるファインチューニングの評価比較

4.3 BigCloneBench によるファインチューニングの評価

BigCloneBench によるファインチューニングの評価を行った.

4.3.1 GPT-3.5-turbo の BigCloneBench による評価

GPT-3.5-turbo のファインチューニング前後の評価比較を BigCloneBench を用いて行った．対象のデータは BigCloneBench の T4 に分類されるクローンペアと，クローンでないペア各 2,000 ペア，計 4,000 ペアである．

ファインチューニング前の GPT-3.5-turbo，ファインチューニング後の GPT-3.5-turbo，GPT-4-turbo の評価結果を表 8 に示す．この結果をもとに，Recall，Precision，Accuracy の変化を計算した．結果は表 9 の通りである．また，結果をグラフにまとめたものを図 11 に示す.

ファインチューニング後の GPT-3.5-turbo とファインチューニング以前の GPT-3.5-turbo を比較すると，Precision が少し向上しているが，あまり大きな変化は見られなかった．また，GPT-4-turbo と比較すると，Recall，Precision，Accuracy の全てで劣っている．

		LLM による回答		
モデル	正解	Yes	No	回答なし
GPT-3.5-turbo	Yes	133	1,865	2
	No	48	1,946	6
ファインチューニング後の GPT-3.5-turbo	Yes	55	1,945	2
	No	14	1,986	0
GPT-4-turbo	Yes	161	1,839	0
	No	35	1,965	0

表 8: GPT-3.5-turbo の BigCloneBench による評価

モデル	Recall	Precision	Accuracy
GPT-3.5-turbo	0.07	0.74	0.52
ファインチューニング後の GPT-3.5-turbo	0.03	0.80	0.51
GPT-4-turbo	0.08	0.82	0.53

表 9: ファインチューニング前後の GPT-3.5-turbo と GPT-4-turbo の評価比較

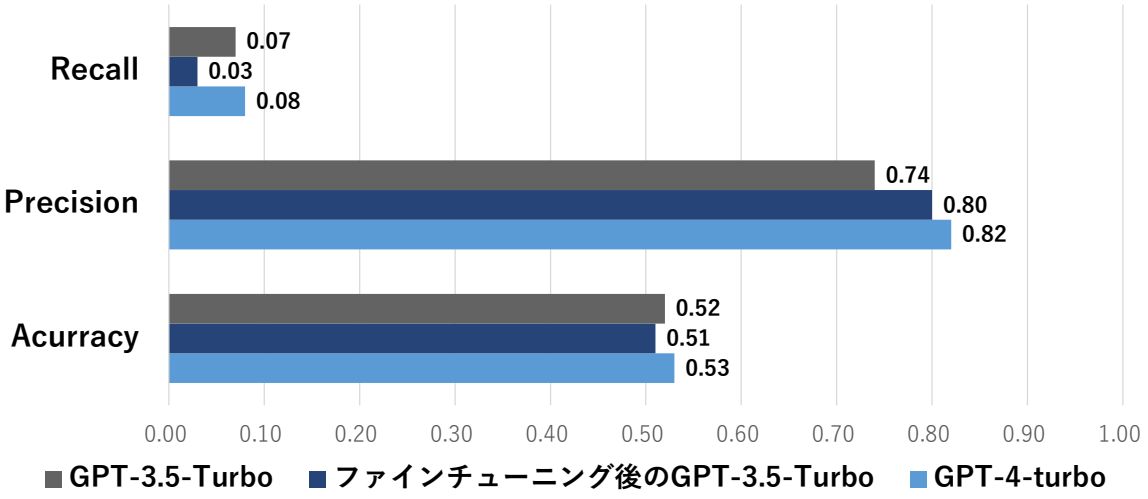


図 11: BigCloneBench によるファインチューニングの評価比較

5　考察

この章では，実験結果について順に考察する．

5.1　GPT-3.5-turbo で BigCloneBench に対して精度向上が見られなかった理由

GPT-3.5-turbo を FEMPDataset を使用してファインチューニングを行った．しかし，BigCloneBench に対して精度向上が見られなかった．

これは，FEMPDataset と BigCloneBench のデータセットが異なるためであると考える．

BigCloneBench では特定の機能を持ったメソッドペアを抽出している．ここで，Krinke らの研究 [4] を紹介する．この研究では BigCloneBench に対して，詳しい検証を行い，BigCloneBench を機械学習の学習データとして使用することに対する問題点を提起している．

この研究の中で，BigCloneBench で定義されているメソッドペアの内，複数の機能を持つメソッドペアがクローンペアとしてデータセットに存在することが具体例を用いて示されている．具体的に，Sunippet10151252 は FTP サーバーにファイルをアップロードするため，”Copy File”の機能を持ち，かつ”Connect to FTP Server”の機能を持つメソッドであるとされ，この 2 つの機能それぞれに対して機能的に等価であるクローンペアを構成している．すなわち，BigCloneBench 内のデータセットのクローンペアは，類似したメソッドペアを抽出しているが，メソッド全体として完全に機能的に等価であるとは限らない．

一方で，ファインチューニングに使用した 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り，メソッド全体で同じ結果を出力する完全に機能等価なメソッドペアを集めたデータセットである．

この 2 つのデータセットの特徴の違いが，ファインチューニングの効果に影響を与えたと考えられる．

6　まとめと今後の課題

本研究では，FEMPDataset を用いて GPT-3.5-turbo にファインチューニングを行い，BigCloneBench に対するコードクローン検出性能の向上を試みた．GPT-3.5-turbo に対してファインチューニングを行った結果，訓練に使用した FEMPDataset に対してモデルが適合したことが確認できた．しかし，BigCloneBench に対しては精度向上が見られなかった．

今後の課題として，以下の 2 点が挙げられる．

調査対象の追加

本研究では GPT-3.5-turbo を対象としたが，他のモデルに対しても同様の実験を行うことで，モデル間の性能比較を行うことができると考える．

BigCloneBench 以外のベンチマークに対しての性能評価

本研究では，ファインチューニングの性能評価に BigCloneBench を使用した．今後，BigCloneBench 以外のベンチマークに対しても同様の実験を行うことで，ファインチューニングの性能評価をより詳細に行うことができると考える．

謝辞

大阪大学大学院情報科学研究科コンピュータサイエンス専攻肥後芳樹教授には，研究の着想から研究活動の直接の御指導，論文の執筆に至るまで，あらゆる場面で多くの御指導を賜りました. 肥後芳樹教授の適切な御指導により，本論文を完成させることができました．心より深く感謝申し上げます.

大阪大学大学院情報科学研究科コンピュータサイエンス専攻松下誠准教授には，研究活動に対して多くの貴重な御助言や御指導を賜りました. 心より深く感謝申し上げます.

大阪大学大学院情報科学研究科コンピュータサイエンス専攻神田哲也助教には，大変貴重な御意見・御助言を賜りました．多くの御助言を頂いた神田助教に心より深く感謝いたします.

大阪大学大学院情報科学研究科コンピュータサイエンス専攻松本真佑助教には，九州大学における研究会において，引率から研究方法や内容に関して多くの御指導を賜りました．心より深く感謝申し上げます.

東京工業大学情報理工学院林晋平准教授には，九州大学における研究会を通して，研究方法や内容に関して多くの御指導を賜りました．心より深く感謝申し上げます.

九州大学システム情報科学研究院情報知能工学部門亀井靖高教授には，九州大学における研究会を通して，研究方法や内容に関して多くの御指導を賜りました．心より深く感謝申し上げます.

九州大学システム情報科学研究院情報知能工学部門近藤将成助教には，九州大学における研究会を通して，研究方法や内容に関して多くの御指導を賜りました．心より深く感謝申し上げます.

最後に，様々な御指導・御助言を頂いた大阪大学大学院情報科学研究科コンピュータサイエンス専攻肥後研究室の皆様に，心より深く感謝いたします.

参考文献

[1] Anthropic. Model Card and Evaluations for Claude Models, 2023.

[2] S. Dou, J. Shan, H. Jia, W. Deng, Z. Xi, W. He, ⋯ X.Huang. Towards Understanding the Capability of Large Language Models on Code Clone Detection: A Survey. 2023.

[3] 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.

[4] J. Krinke and C. Ragkhitwetsagul. BigCloneBench Considered Harmful for Machine Learning. In *2022 IEEE 16th International Workshop on Software Clones (IWSC)*, pp. 1–7, 2022.

[5] M. Mondal, C. Roy, and K. Schneider. A Fine-Grained Analysis on the Inconsistent Changes in Code Clones. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 220–231, 2020.

[6] T. Nakagawa, Y. Higo, and S. Kusumoto. NIL: large-scale detection of large-variance clones. ESEC/FSE 2021, p. 830–841, New York, NY, USA, 2021. Association for Computing Machinery.

[7] OpenAI. GPT-4 Technical Report, 2023.

[8] C. Roy and J. Cordy. A Survey on Software Clone Detection Research. *School of Computing TR 2007-541*, pp. 3–7, 01 2007.

[9] C. Roy and J. Cordy. NICAD: Accurate Detection of Near-Miss Intentional Clones Using Flexible Pretty-Printing and Code Normalization. In *2008 16th IEEE International Conference on Program Comprehension*, pp. 172–181, 2008.

[10] C. Roy, J. Cordy, and R. Koschke. Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. *Science of Computer Programming*, Vol. 74, No. 7, pp. 470–495, 2009.

[11] B. Rozière, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, ⋯ G.Synnaeve. Code Llama: Open Foundation Models for Code, 2023.

[12] J. Svajlenko, J. Islam, I. Keivanloo, C. Roy, and M. Mia. Towards a Big Data Curated Benchmark of Inter-project Code Clones. pp. 476–480, 09 2014.

[13] R. Taori, I. Gulrajani, T. Zhang, Y. Dubois, X. Li, C. Guestrin, T.⋯ Hashimoto. Alpaca: A strong, replicable instruction-following model. Stanford Center for Research on Foundation Models, 2023.

[14] Gemini Team. Gemini: A Family of Highly Capable Multimodal Models, 2023.

[15] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, ⋯ T.Scialom. Llama 2: Open Foundation and Fine-Tuned Chat Models, 2023.

[16] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, ⋯ I.Polosukhin. Attention Is All You Need, 2023.

[17] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, ⋯ J. R.Wen. A Survey of Large Language Models, 2023.

[18] L. Zheng, W. L. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, ⋯ I.Stoica. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena, 2023.

[19] 井上克郎, 神谷年洋, 楠本真二. コードクローン検出法. コンピュータソフトウェア, Vol. 18, No. 5, pp. 529–536, 2001.

[20] 肥後芳樹. 自動テスト生成技術を利用した機能等価メソッドデータセットの構築. ソフトウェアエンジニアリングシンポジウム 2023 論文集, Vol. 2023, pp. 30–38, 08 2023.