GPT-4o を用いた単体テストの自動生成手法の初期評価

橋浦研究室 121I213 王 淦

1. はじめに

近年，生成 AI は様々な分野で利用され，普及が進んでいる．ソフトウェアテストでテストケースを作成する作業は非常に労力が必要な作業である．

テストケース作成に生成 AI を導入することができればソフトウェアテストの労力の問題を解決できる．テストケースの生成に生成 AI がある程度の効果があることは関連研究等で指摘されている通りである．その一方で，生成 AI がテストケースを生成する問題点として，プロジェクト固有のキーワードや関数名に影響を受けることなどが指摘されている．

ここで，リスト 1 のようなコードを考える．このコードは関数名が increment と定義されている．しかしながら，処理の内容は偶数と奇数の判定となっている．このように関数名と処理の内容が異なる場合や，プロジェクト独自の用語を利用する場合には，処理の内容ではなく仕様書に基づいてテストケースを生成しないと正しいテストを生成することができない．そこで，本研究では仕様書を入力として与え，仕様書の記述から現状のコードで実行可能な，カバレッジが高いテストを生成できるかを確かめる．

```
std::string increment(int a){
  if (a % 2 == 0) {
     return "偶数";
   } else {
     return "奇数";
   }
}
```

リスト 1. 関数名と内容処理が異なる例

2. 関連研究

生成 AI を利用してテストの生成と評価を行った研究として Schäfer[1]らの研究がある．この研究では LLM を用いて JavaScript に対しての自動で単体テスト生成プロセスを提案している．この研究では従来のテスト生成技術と比較し，カバレッジが平均 18.9%向上している．

また，Hourani[2]らの研究では，機械学習や自然言語処理などの人工知能がソフトウェアテストに与える影響について調査を行っている．機械学習などの人工知能がテストの精度と効率の向上に効果があると述べている．

3. テストケースの要素

テストケースを作成するためには以下の作業が必要になる[3].

1. テスト要件の分析
2. テスト項目の設計
3. テストデータの作成
4. テストコードの作成
5. テストケースの検証

生成 AI でテストケースを生成する場合は，コードと仕様書を生成 AI の入力とすることができるので，1 から 4 の作業は自動化することができる．5 の要素についてもツールに組み込むことである程度の自動化が可能である．

4. 研究目的

本研究では以下二点のリサーチクエスチョン(以下 RQ)を設定する．

RQ1. 生成されたテストは付属と比較して実用的と言えるか

RQ2. 生成 AI は仕様書を基準に正しくテストケースを生成できるか

RQ を検証するために本研究では OSS で公開されている C++のライブラリを対象として，生成 AI とライブラリに付属している作成したテストの比較する．

5. 実装

本研究ではできるだけ高性能なモデルを利用するために OpenAI のモデルを利用する．具体的には GPT-4o を利用してテストケースの生成を行うこととした．

テストを自動で生成し，検証まで行う自動化ツールの流れを図 1 に示す．テスト対象のコードと仕様書をツールの入力とする．OpenAI の API を利用して生成 AI でテストを生成する．実行が失敗した場合にはエラーログと生成されたテストを含んだデータで再生成プロセスを行う．正しい生成が不能な場合を考慮し，テストの実行失敗が指定回数を超えたら処理を停止する．対象のプログラムは C++であるため，カバレッジの計測には gcov[4]と lcov を使用する．

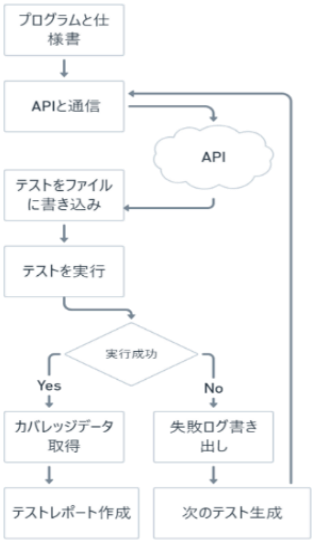

図 1. ツール実行の流れ

6. 評価

評価は一般公開されている C++のライブラリである{fmt}[5]に含まれる 13 のモジュールを対象に実施した．使用するカバレッジは命令網羅(C0)を用いる．評価にあたっては比較対象として，対象ライブラリに付属するテストスイートを利用することとし，各モジュールに対して生成 AI で最大で 10 回までテストケースの生成を行う．生成 AI によってテストは生成できたものの，実行不可能なテストであった場合，そのモジュールのカバレッジは 0%とし，テストケース自体の生成ができない場合には，そのモジュールを評価対象から除くこととした．さらに，生成されたテストの項目とライブラリの仕様書を比較し，テストケースに正しく仕様書の内容が反映されているかを確認する．

評価を実施した結果，13 個のモジュールの内に OpenAI の API 側の制限によって入力できる文字数に上限により，テストケースの生成が不能なモジュールは 5 つであった．よって，今回のカバレッジの算出対象のモジュールは 8 つである．カバレッジの算出結果を表 1 に示す．

表 1．カバレッジの比較

#	生成カバレッジ	付属カバレッジ	差
1	44%	63%	19%
2	14%	58%	44%
3	25%	41%	16%
4	8%	19%	11%
5	15%	23%	8%
6	36%	70%	34%
7	9%	36%	27%
8	12%	27%	15%
全体	17%	47%	30%

生成されたテストは比較的低いカバレッジを示した．例えば，{fmt}[5]の chrono モジュールの仕様書では次のメソッドと説明が記載されている．

1. std::chrono::duration
2. std::chrono::time_point
3. std::tm
4. fmt::localtime
5. fmt::gmtime
6. Chrono Format Specifications
7. スレッドセーフな動作

この 7 つの項目に対して生成 AI が対応出来なかった項目は 2 と 7 の 2 項目である．他のモジュールについても同様にメソッドや特別な記術を項目に仕様書と生成されたテストの項目の比較を行った．その結果，生成 AI が仕様書の中で網羅出来なかった項目としてメモリ確保やスレッド関連項目，ユーザによるカスタムが必要な項目であることが明らかになった．

よって，生成 AI によって生成されたテストケースは仕様書に対しての網羅率は高いが，C0 カバレッジは低いことが分かる．これは生成 AI によるテスト生成作業は入力となる仕様書への依頼が高いとも言える．このため，より詳細な仕様書を入力することで，生成するテストの実用性改善が見込める．

7. まとめ

本研究では OSS として一般公開されている C++のライブラリに対して生成 AI を用いてテストを生成した．生成されたテストに対してライブラリ付属のテストとカバレッジ比較することで評価を行った．また，仕様書に含まれる項目にと生成されたテストケースに反映されているかどうか調査した．

評価結果として，生成されたテストのカバレッジはライブラリ付属のテストケースと比較して低い値に留まった．その一方で，仕様書に含まれる記述内容の網羅度を考慮した場合，すべてのモジュールに対して 70%以上の高い網羅率を示した．

生成 AI によって生成されるテストの実用性を向上させるには仕様書をより具体的に記述にする必要性が明らかになった．今後は自動化のプロセスに仕様書の拡張，補完を組み込むことでより実用性の高いテストを生成する事が可能と思われる．さらに，自動化プロセスに生成されたテストを入力にフィードバックすることでテストケースを改良することでさらなるカバレッジの向上を目指すことができる．

謝辞

本研究は JSPS 科研費 24K15214 の助成を受けた．

参考文献

[1] M. Schäfer, S. Nadi, A. Eghbali and F. Tip, "An Empirical Evaluation of Using Large Language Models for Automated Unit Test Generation," in IEEE Transactions on Software Engineering, vol. 50, no. 1, pp. 85-105, Jan. 2024.

[2] Hourani, H., Hammad, A.M., & Lafi, M.," The Impact of Artificial Intelligence on Software Testing," IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology, pp. 565-570, Apr. 2019.

[3] Divyani Shivkumar Taley and Bageshree Pathak, "Comprehensive study of software testing techniques and strategies: A review," International Journal of Engineering Research and Technology, Vol. 9, Issue 8, pp. 817–822, Aug. 2020.

[4] Free Software Foundation, Inc., "This file documents the use of the GNU compilers", Free Software Foundation, 1988.

[5] GitHub, "virtual/{fmt}," https://github.com/fmtlib/fmt, Apr.2014, (accessed 2024/12).