一般社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report

LLMを用いた自動プログラム修正における正答率の予測に向けて

石坂　颯大†　　柗本　真佑†　　楠本　真二†

安田　和矢††　伊藤　信治††　張潘タンフエン††

† 大阪大学大学院情報科学研究科
†† 株式会社 日立製作所
E-mail: †{s-isizak,shinsuke}@ist.osaka-u.ac.jp

あらまし　大規模言語モデル（LLM）を用いた自動プログラム修正（LLM-APR）の実現可能性が増している．LLM-APR の実践的利用における一つの課題として，LLM の出力に対する品質保証手段の確保が挙げられる．LLM-APR による修正は開発者による修正と同様，誤りを含むことがあるからである．この課題を解決するために，LLM-APR におけるプログラム修正の成否の予測が必要である．本研究では，LLM の品質保証を目的として，成否と相関を持つ新たな指標の発見に取り組む．実験の結果，最も高い相関係数でも 0.32 であったため，より高い相関を持つ指標の発見が一つの課題である．

キーワード　大規模言語モデル（LLM），自動プログラム修正（APR），品質保証

1. はじめに

大規模言語モデル（LLM，Large Language Model）を用いた，全自動によるプログラム修正（APR，Automated Program Repair）の実現可能性が増している．本稿ではこの LLM ベースの APR を LLM-APR と呼称する．LLM-APR ではバグを含むソースコードとバグの修正指示を自然言語のプロンプトとして与えることで，バグが取り除かれたソースコードを得る．GPT-3.5 や GPT-4，CodeLlama などの言語モデルは，自然言語の文章だけでなく大量のソースコードが学習データに含まれている．よってこれらのモデルは，APR [1] やコード生成 [2]，コード要約 [3] などのソースコードに関する様々なタスクにも適用できる．LLM-APR はソースコード解析に基づく既存の APR 技術 [4] [5] と比較して，高い修正性能を持つことが報告されている [6].

LLM-APR の利用における一つの課題として，修正結果に対する品質保証手段の確保が挙げられる．LLM-APR による修正は開発者による修正と同様，誤りを含むことがある．よって，その修正の成否を何かしらの手段によって予測すべきである．APR という状況においては，その成否は自動テストによって確認することもできる．修正後のソースコードに対して自動テストを適用すれば，その振る舞いが期待通りであるかを自動的に検証できる．しかしながら，テストの利用を前提とした評価手法は LLM-APR が持つ可搬性の犠牲に繋がる．LLM-APR はバグを含むソースコードとタスク指示のみが必須の入力データであり，テストを必須とする既存 APR と比較して高い可搬性を持つ．LLM-APR の可搬性を保つためには，テストとは異なる手段による成否の予測方法が必要である．

もし LLM-APR の成否を予測することができれば，LLM の性能評価の見直しにも繋がると考えられる．LLM は確率分布に基づく統計モデルであり，処理結果には一定のランダムさが含まれる．よって LLM の性能を確かめる際には，単一試行ではなく複数回の試行を行うことが一般的である．Chen ら [7] の提案する Pass@k は複数回試行を前提とした評価指標であり，k 個の解を選んだ際に 1 つ以上の正解を含む可能性を表している．Pass@k における k 個の選択は無作為が前提である．すなわち解集合の中で優先順位を付与できないという状況を想定している．もし成否の予測が可能となれば，解集合の中での優先順位の付与が可能となり，Pass@k よりも実践的な状況を想定した評価指標を得ることが可能となる．

本研究の目的は LLM-APR におけるプログラム修正の成否の予測である．そのために，本稿では成否と相関を持つ新たな指標の発見に取り組む．この指標を信頼度スコアと呼ぶ．信頼度スコアは，その獲得において LLM-APR の可搬性を損なわないという必須要件があり，自動テストの利用は考慮しない．本稿で提案する信頼度スコアは確信度，難易度，およびトークン確率の 3 種類である．確信度と難易度は LLM に直接問い合わせる値であり，LLM の内発的な指標であるといえる．取り組んだタスクの結果に自信があるほど，確信度は高く正答率（タスクが成功した割合）も高くなると考えられる．難易度も確信度とアイデアであり，タスクが容易だと感じるほど難易度は低く正答率は高くなる．LLM は問い合わせの表現や手順によって結果が異なることが知られており [8]，確信度とは別の問い合わせとして難易度を採用する．トークン確率は機械的に得られる値である．LLM は文章を生成するときにトークンと呼ばれる短い語を確率に基づいて選択する．この確率が高いほど出力に対して自信があると考えられる．さらに本稿では，信頼度スコアを用いた LLM の新たな評価指標 $\mathrm{Pass}_{\mathrm{conf}}@k$ の有効性についても実験的に確かめる．$\mathrm{Pass}_{\mathrm{conf}}@k$ は信頼度スコアに基づいて正解群を選択する際の Pass@k であり，正解群

の優先順位を付与するという実践的な LLM 利活用の状況を想定した評価指標である．

実験の結果，トークン確率が最も高い相関係数を持っておりその値は 0.32 であった．成否の予測という目的に対しては十分とはいえず，より高い相関を持つ指標の発見が一つの課題である．難易度と確信度はどちらも無相関であった．LLM 自身に問い合わせる内発的な指標は成否の予測には利用できないといえる．特に確信度は 80 以上と答えるケースが 99% を占めていた．タスクの内容に依らず常に高い確信度と答えており，LLM 自身が感じていた真の確信の度合いは抽出できていないと考えられる．もしくは，LLM の自身の行為に対する自己認知能力の不足という可能性も考えられる．$\mathrm{Pass}_{\mathrm{conf}}@k$ についても Pass@k を上回る結果は見られなかった．最も高い相関係数でも 0.32 程度であり，適切な解の優先順位が付与できていないことが原因だと考えられる．

2. 準　　備

2.1 LLM を用いたプログラム修正（LLM-APR）

LLM-APR は LLM の登場により近年盛んに研究されている．LLM-APR では LLM にバグを含むソースコードとその修正の指示を含むプロンプトを与えることで修正済みソースコードを得る．従来，自動プログラム修正は様々なソースコード解析技術を組み合わせて実現されていた．例として探索ベースの手法 [4] や意味論ベースの手法 [9] などが挙げられる．他方 LLM-APR では LLM にソースコードを与えることで LLM がそれを理解し，修正する．例えば Hossain らは LLM にバグを限局・修正させている [10]．また Xia ら [6] は 9 つの LLM モデルを用いて，Defects4J [11] を含む 5 つのデータセットについて従来の APR 技術と比較した．結果すべてのデータセットにおいて既存の APR 技術を大幅に上回ることを明らかにした．このように LLM-APR は修正精度の観点で期待されている．

また LLM-APR は従来の APR 手法と比較して，精度以外にも利便性と可搬性の点で優位性を持つ．まず自然言語を用いた自然な対話インタフェースが可能であり，APR に関する特殊な知識やパラメタ調整が一切必要ない．さらに，自動テストが不要という可搬性上の利点もある．従来の探索的 APR はテストの結果を探索の目標関数としており，テストが必須だった．一方 LLM-APR は大規模言語モデルの推論に基づいてプログラムを修正するため，テストが必須でない．

2.2 LLM-APR の課題

しかし LLM-APR による修正は開発者による修正と同様，誤りを含むことがある．しかも従来の APR と違い，プログラム修正の成否を確かめることができない．先述の通り LLM-APR は大量データに基づく推論を動作の核としているためである．

もし開発者の記述した自動テストが存在していれば，成否を自動的かつ客観的に確かめることも可能である．しかし成否判定に自動テストを用いることは，LLM-APR の利点の一つであるテストが不要という性質を損なうことになる．自動テストが不要という LLM-APR の可搬性上の利点を保つためには，自動テスト以外の手法によってプログラム修正の成否を見積もる手法が求められると考えられる．

またプログラム修正の成否の見積もり結果は，LLM-APR により得られた解の優先順位付けにも活用できる．LLM は一定のランダムさが含まれるため，タスクに取り組む際は複数回の試行を繰り返すことが一般的である [7]．しかし試行回数が多いほど解の成否確認に要するコストが大きくなってしまう．得られた複数の解に対して，正答率との関連を持つような値によって優先順位を付与できれば，解の正しさを目視確認するコストを低減させることも可能である．

3. 信頼度スコア

本研究の目的は，LLM を用いたプログラム修正における正答率の予測である．そのために正答率の代用となる，すなわち正答率との相関を持つ可能性のある指標を考える．本論文ではこの指標を信頼度スコアと呼ぶこととする．信頼度スコアに求められる性質は以下の 3 つである．

- 正答率と相関を持つ．
- LLM-APR の可搬性を損なわない．
- 多大な計算コストを要さない．

2 つ目の要件は，すなわち自動テスト等の LLM-APR 適用に不要な追加データを要さないという意味である．3 つ目に関しては正答率の代用という性質上，計算コストは低いことが望ましい．以降では，提案する 3 つの信頼度スコアについて説明する．

3.1 確　信　度

確信度とは LLM 自身が出力する内発的な指標であり，与えられたタスクの解に対する自信の度合いである．人が何かしらのタスクに取り組む際と同様，解が成功に近いほど高い確信度となると考えられる．逆に解が正しくないときほど自信がない，つまり低い確信度となると考えられる．確信度は先述の 3 つの要件を満たす指標であり，LLM にタスクを取り組ませた後にその確信度はどうであったかを自然言語で問い合わせる．値は 0 から 100 の範囲で自己申告させる．

確信度は Xiong ら [12] によって提案された指標であり，本研究と同じ目的で用いられている．Xiong らの研究では，プログラム修正等のソフトウェアタスクではなく，常識・数値計算等のタスクにおいて確信度が LLM から取り出せるかを確かめている．実験の結果，LLM が出力した確信度のほとんどは 80 以上の極めて高い値に限定されており，正答率と比較して高い，つまり過信傾向であることが報告されている．

3.2 難　易　度

難易度は確信度と同様，LLM に直接的問い合わせることで得られる LLM 自身の内発的な指標である．この指標も確信度と同様に，人が正答率を予測する方法に着想を得ている．例えば，タスクが易しいと感じるときほど正答率は高くなり，逆に難しいと感じるほど正答率が低くなると考えられる．この指標は取り組んだタスクをどう感じたか，という LLM 自身の印象や主観を表すという点で，確信度と共通のアイデアを持つ．Xiong ら実験結果では，LLM が出力する確信度は 80 以上の狭い値域に限られていしまっていた．難易度は確信度とやや似

た指標ではあるが，LLM では尋ねる文章の構造や表現による結果の大幅な変化が指摘されている [8]．よって我々は自信の度合いではなく，タスクが難しかったかを LLM に問い合わせるという代替手法を考える．値の範囲は確信度と同様 0 から 100 とする．

3.3 トークン確率

LLM 内部から機械的に得られるトークン単位の確率を用いた指標を考える．LLM が文章を生成するとき，LLM はトークンと呼ばれる短い語の候補の中から確率に基づいてトークンを選択し，文章を生成する．例えば「私は学生です」という文章を生成するとき，「私」の後には「は」という語が続く確率が高いと判断している．これはすなわちこの確率が高いほど LLM が自信をもってその文章を生成しているとみなせるため，信頼度スコアとして利用できると考えた．

本研究ではタスクに対するトークン確率を次のように定義する．文章を生成する n 個のトークンを $t_1, t_2, ..., t_n$ とし，t_i が選ばれる確率を p_{t_i} とする．

$$\text{トークン確率} = \prod_{i=1}^{n} (p_{t_i})^{\frac{1}{n}}$$

各トークンの p_{t_i} の総乗を取ることで文章全体の確率となる．単なる総乗だとトークン数によって値が左右されてしまうため $1/n$ 乗して平均を取る．

LLM 評価においてはトークン確率の対数が用いられる [6] [13] 場合もあるが，本研究では対数化せず用いる．対数に変換する理由は確率が極めて小さくなる場合に比較が困難になるからである．今回のタスクにおけるバグの性質上，LLM は与えられたバグを含むソースコードの一部だけを変更して出力を行うため，修正部分のトークン確率が小さくなりそれ以外の部分の確率は 1 に近い値をとると予想される．そのためトークン確率の対数をとる必要はないと考えた．また，このように定義することでほかの信頼度スコアと同じように 0 から 100 の値での表現が可能となる．

4. 実験設計

4.1 実験概要

3. 節で説明した 3 つの信頼度スコアが，LLM-APR の成否の予測に利用可能かを実験的に確かめる．実験では，まずバグを含むソースコードと修正指示を LLM に与え，プログラム修正タスクに取り組ませる．LLM-APR が生成した解となるソースコードに自動テストを適用し，その成否を確かめる．LLM の内発的な信頼度スコア（確信度と難易度）を確かめる際には，LLM へ与えるプロンプト内にスコアを自己申告させるよう指示する．内発的ではない信頼度スコア（トークン確率）は，LLM 実行時の API にトークン確率を出力させるオプションを加える．実験で用いた ChatGPT の場合，`logprobs=True` がそのオプションに該当する．

4.2 実験の流れ

実験は以下 4 ステップで構成される．

Step1. まず題材となるメソッドに対してバグを人為的に埋め込み，バグを含むソースコードを得る．具体的にはミューテーション解析 [14] によりバグを埋め込む．題材データセットとミューテーションについては 4.5 節で説明する．

Step2. LLM への入力となるプロンプトを作成する．プロンプトの中身はソースコードが満たすべき仕様，バグを含むソースコード，および自然言語でのプログラム修正の指示である．LLM 自身に問い合わせる信頼度スコア（確信度と難易度）を計測する際は，プログラム修正の指示に加えて，各種信頼度スコアを問い合わせる指示を加える．トークン確率は LLM に直接問い合わせず，API 実行時のオプションの付与により値を得る．

また実験では 2 種類のプロンプト方法を試みる．Vanilla は上記の最小限かつ単一のプロンプトを LLM に与える方法である．Chain of Thought（CoT）は Vanilla に加えて，ステップバイステップでなぜその解に至ったかを LLM 自身に説明させる方法である．CoT は推論ステップ推論能力の向上に寄与すると指摘されている [15]．手軽で効果的な方法であり，様々なタスクにおいても有用であると考えられることから，実験で行うプロンプトの一種として採用した．

Step3. LLM へ Step2 で作成したプロンプトを入力し，LLM-APR を 10 回繰り返す．この際，同一条件の試行（例えばバグを埋め込むメソッド：`hasChildNode()`，信頼度スコア：確信度，プロンプト：Vanilla）に対しては，同一のプロンプトを与える．複数種類の解を得るために，解のゆらぎを調整するパラメタ temperature は 0.7 とした．この値は常識や算数の問題などの一般的なタスクに対して確信度を誘出する実験を行っている既存研究，Xiong らの研究 [12] に従っている．

Step4. LLM の出力を解析し，ソースコードが正しく修正できているか確認し，信頼度スコアの値を取り出す．ソースコードの修正の正しさについては，事前に用意した単体テストを適用し確認する．すなわちバグ修正の成否を機能的な振る舞いの正しさと捉えており，ミューテーションで書き換えた箇所と同一箇所を修正しているかは考慮していない．また CoT プロンプトで実験を行った際の推論過程の正しさについては考慮しない．ステップバイステップで推論を行う行為自体が，修正の成否や信頼度スコアに影響すると考えられるためである．

4.3 具体的なプロンプトと LLM の出力の例

具体的なプロンプトの一例，および LLM が出力した解を，図 1 に示す．この図は信頼度スコアとして確信度を，プロンプト方法として Vanilla を選んだ際の内容である． 図 1(a) の前半で修正指示とフォーマットの指定，注意書きが書かれており，後半で実際に仕様とバグを含むソースコードを与えている． 図 1(a) の `###code###` の "`return childNodes == EmptyNodes;`" にバグが含まれている． 図 1(b) の例では正しくバグを修正できており，また確信度は高い値が得られていることから，確信度を正答率の代替として利用できる可能性を示している．

4.4 LLM モデル

LLM のモデルは gpt-3.5-turbo を使用した．このモデルではテキストの入出力が可能であり，出力されるトークンごとの対

```
Fix the source code to
follow the specification,
and provide your
confidence in your
answer. Use the following
format.

[FIXED PROGRAM]
[CONFIDENCE]

Note that:
Provide only the answer
and confidence, do not
provide any explanation.
The confidence indicates
how likely you think
your fixed source code
is correct.

Now, please fix the
source code below.

###specification###
Internal test to check if
 a nodelist object has
been created.

###code###
protected boolean
hasChildNodes() {
    return childNodes ==
    EmptyNodes;
}
```

(a) プロンプト

```
protected boolean
hasChildNodes() {
    return childNodes !=
    EmptyNodes;
}

100%
```

(b) 出力結果

図 1: 実験で用いたプロンプトと出力の一例

数確率が得られる．このモデルは確信度を誘出する実験を行っている Xiong らの研究 [12] でも用いられていた．本研究では LLM の代表例としてこのモデルを用いて実験を実施する．

4.5 データセット

データセットとして Java プロジェクト jsoup(注1) を用いる．本研究では jsoup の JavaDoc を仕様として用いる．jsoup は GitHub でのスター数が 10,000 を超えており，多くの人に使用されていることから，適切な JavaDoc やテストが用意されていると考え，データセットとして選定した．

バグは Mutanerator(注2) により埋め込む．Mutanerator は Java プログラムへのミュータント作成ツールである．具体的にはソースコード中の "==" を "!=" に， "+" を "-" するといった変換によりバグを埋め込む．実際に Mutanerator を用いることで，jsoup の Element.java から合計 100 件のバグを含むソースを得た．1 個のメソッドの平均行数は約 6 行である．

（注1）：https://github.com/jhy/jsoup

（注2）：https://github.com/kusumotolab/Mutanerator

4.6 評価方法

評価の方法としては，正答率と信頼度スコアの相関の有無の確認，および新たに提案する LLM の評価指標 $\mathrm{Pass}_{\mathrm{conf}}@k$ の有用性の確認の 2 つを考える．

正答率を各信頼度スコアで代替可能かを確認するために，2 つの指標の相関を確かめる．正答率は成否の二値であり，信頼度スコアは 0 から 100 まで値となる．よって二値の変数と連続値の変数の相関となるので点双列相関係数（以下，相関係数）を用いる．これは二値と連続値の相関である．

さらに信頼度スコアの有用性を確かめるために，既存の LLM の評価指標である Pass@k と信頼度スコアを用いた新たな評価指標 $\mathrm{Pass}_{\mathrm{conf}}@k$ の比較を行う．Pass@k は Chen らの提案する評価指標であり [7]，LLM における複数回の試行を考慮している．直感的には，LLM から得られた複数解から k 個をピックアップした際に最低一つの解を得られる確率を表している．Pass@k は k 個のピックアップという操作を全ての可能性として考えている．これはすなわち，無作為による k 個の解のピックアップを前提としている．一方，正答率と相関を持つ指標が得られるのであれば，その指標の順にピックアップすることで，無作為よりも効率的な解の選択が可能となる．一方で $\mathrm{Pass}_{\mathrm{conf}}@k$ は信頼度スコアを反映する指標である．具体的には，解を信頼度スコアで並び替えてから k 個をピックアップした際に最低一つの解を得られる確率を表す．

なお実際にはすべてのタスクにおいてこの評価を行ったわけではなく，10 回の試行の中で成否が異なる場合にのみ評価の対象とした．10 回の試行の成否が 10 回とも同じとき Pass@k，$\mathrm{Pass}_{\mathrm{conf}}@k$ はともに 1 または 0 になるからである．実際に対象となったタスクは全体のうちの約 40 %である．

5. 実験結果

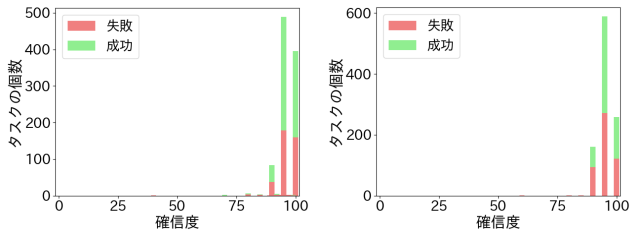
5.1 相　　関

正答率と信頼度スコアの相関係数を 表 1 に示す． 表 1 はプロンプト，信頼度スコア別の相関係数である．例えばプロンプト：Vanilla，信頼度スコア：確信度のとき相関係数は 0.03 である．各信頼度スコアについて相関係数を確認すると，まず確信度と難易度が無相関であると分かる．トークン確率は比較的相関が高いが，正答率の予測には不十分である．

信頼度スコア	Vanilla	CoT
確信度	0.03	0.08
難易度	0.14	0.11
トークン確率	0.32	0.22

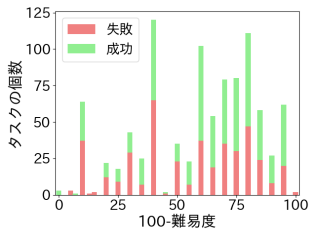
表 1: 正答率と信頼度スコアの相関係数

表 1 の結果に対する詳細な分析を行うために， 図 2 に信頼度スコアごとの成否の割合を積算グラフで表す． 図 2 の各図の位置は 表 1 の表と対応している． 図 2 のすべてのグラフにおいて横軸が信頼度スコア，縦軸がバグ修正タスクの個数を表す．例えば 図 2(a) では確信度が 100 %であると答えたタスク

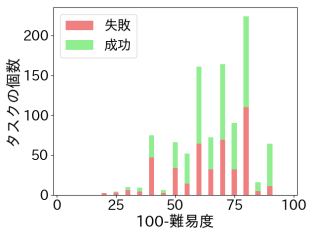


(a) 確信度・Vanilla

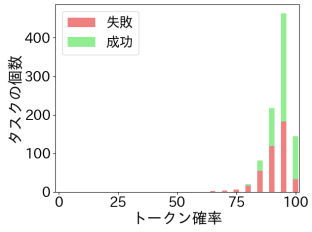
(b) 確信度・CoT



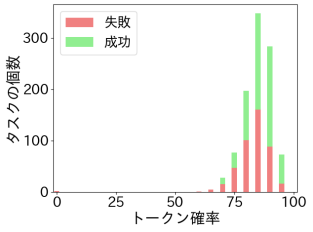
(c) 難易度・Vanilla



(d) 難易度・CoT



(e) トークン確率・Vanilla



(f) トークン確率・CoT

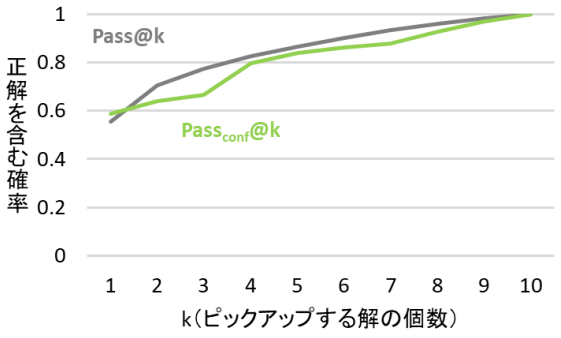
図 2: 信頼度スコア・プロンプト別の成否の積算グラフ

数が約 400 回であり，その正答率は約 50 %だと読み取れる．信頼度スコアが高くなるにつれて成功（緑）の割合が増える場合に相関係数が高くなる．

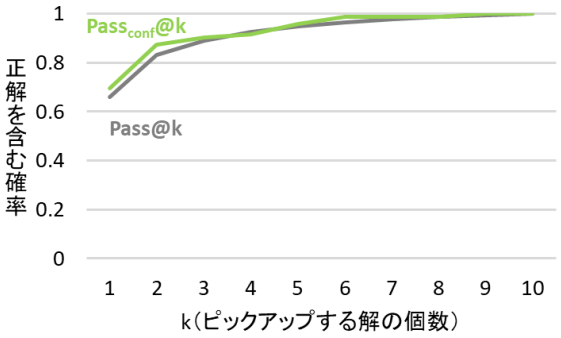
確信度については値が大きくなっても成功の割合が増える傾向はなく，80 以上の値に偏っている．これは Xiong らの研究の結果とも一致する [12]．次に難易度に着目する．難易度は難易度は難しいタスクほど正答率が低くなる，という逆相関となるため，横軸を 100 - 難易度としている．難易度は確信度より取り得る値の範囲は広いが，相関は弱いことが読み取れる．確信度・難易度に比べるとトークン確率は確率の値が大きくなるほど正答率が上がっている傾向がみられる．信頼度スコアの中ではトークン確率が正答率の予測に最も有用であると分かる．結論として，確信度と難易度は無相関であり，トークン確率も弱い正の相関であるため正答率の予測には有用ではないといえる．

5.2 複数解の優先度付け

図 3 に従来指標である Pass@k と提案する Pass$_{\text{conf}}$@k の比較を示す． 図 3(a) は Pass$_{\text{conf}}$@k の算出に難易度を，図 3(b) はトークン確率を用いた際の結果である．5.1 節の結果からもわかるように確信度に関しては正答率の予測ができていなかったため Pass@k との比較は省略している．横軸が k，縦軸が k 個の解に 1 つでも正解を含む確率を表す．例えば 図 3(a) では Pass$_{\text{conf}}$@1 は 60 %の確率で正しい答えを含む．



(a) 難易度



(b) トークン確率

図 3: Pass@k と Pass$_{\text{conf}}$@k の比較

図 3(a) では $k = 1$ のとき以外は Pass@k が Pass$_{\text{conf}}$@k より優れており，$k = 3$ のときその差は 0.11 と最大となる．すべての k に対しての Pass@k と Pass$_{\text{conf}}$@k の正解を含む確率の平均は 0.03 だけ Pass@k が高い． 図 3(b) ではわずかに Pass$_{\text{conf}}$@k が Pass@k より優れている．最も差が大きいのは $k = 2$ のときでその差は 0.04 である．すべての k に対しての Pass@k と Pass$_{\text{conf}}$@k の正解を含む確率の平均は 0.01 だけ Pass$_{\text{conf}}$@k が高い．よって Pass$_{\text{conf}}$@k は Pass@k と大きな差はなく，信頼度スコアによる複数解の優先度付けは信頼度スコアがトークン確率である場合にのみわずかにランダムな選択より優れるが，トークン確率の精度も優先度付けには有用でないといえる．

6. 課題と展望

6.1 正答率の予測に有用な信頼度スコアの発見

本研究の 3 つの信頼度スコアは正答率の予測に対しては実用的ではなかったため，より高い相関を持つ指標を考える必要がある．本論文では信頼度スコアとして，バグ修正タスクに特化しない指標を取り上げた．しかしながらバグ修正やプログラム生成，要約などプログラミングタスクに特化するという前提であれば，プログラミング特有の情報を活用できる可能性がある．例えばバグを修正させる前に，ソースコードの理解度を調べるというアイデアが考えられる．ソースコードのメソッド名や変数名，仕様などの情報の一部をマスクして

LLM にマスクされた情報を推論させる．これはメソッド名などの推論精度が高いほど，ソースコードを正しく理解できており，バグ修正の正答率も高い，という仮説に基づいている．

6.2 プロンプトエンジニアリングによる予測精度の向上

LLM へ与えるプロンプトの表現の違いにより LLM の出力は大幅に変化するため [8]，別のプロンプトを与えることで正答率の改善だけでなく，より正答率と高い相関を持つ信頼度スコアを取り出せる可能性がある．例えばタスクを解かせるときに，簡単なタスクと難しいタスクを与える手法が考えられる．本研究では確信度と難易度による正答率の予測に失敗したため，この手法では確信度や難易度の精度の向上を期待している．簡単なタスクと難しいタスクにより LLM が正しい数値の基準を理解することを期待している．Xiong らの研究でも LLM が複数の回答の候補がある場合のほうが，確信度誘出の精度が良いとされていた [12]．この手法は少数の例を提示することで LLM の出力の精度を向上させる [16]few-shot learning の一種であるといえる．

6.3 LLM によるデータセットの学習可能性

LLM が今回修正対象となったソースコードを学習しており，正確な評価ができていない可能性がある．ChatGPT においては学習データが公開されていないため，この可能性を否定できない．同じタスクにおいて修正の成否が分かれる場合が約 40 %あったこと，jsoup が今回のモデル gpt-3.5-turbo の学習以降に更新されたバージョンであることから，学習に含まれていないと考えている．このようにデータリーケージへの対策としては学習期間外のデータやオリジナルのデータを用いることが挙げられる．

7. おわりに

本研究では LLM-APR におけるプログラム修正の成否の予測を目的として成否と相関を持つ新たな指標（確信度スコア）の発見に取り組んだ．信頼度スコアとしては，確信度と難易度，トークン確率の 3 つについて実験した．評価には相関係数と $\mathrm{Pass}_{\mathrm{conf}}@k$ を用いた．$\mathrm{Pass}_{\mathrm{conf}}@k$ での評価では既存の評価方法 $\mathrm{Pass}@k$ と比較した．正答率との相関は非常に弱いという結果になった．結果として，ランダムな解のピックアップである $\mathrm{Pass}@k$ よりはトークン確率による優先度付けにおいてのみ $\mathrm{Pass}@k$ より $\mathrm{Pass}_{\mathrm{conf}}@k$ がわずかに優れていた．しかし，相関やその他の信頼度スコアでの優先度付けからおおむね信頼度スコアでの正答率の予測に有用でないと分かった．

今後の研究課題として，バグ修正のプログラミングタスクとしての特性の活用が挙げられる．本研究の信頼度スコアや評価方法などはタスク自体の特性に依存していない．そこでソースコードの一部の情報をマスクし，それを推論させてソースコードの理解度を調べてからバグを修正させるという実験を考えている．この手法は理解度による正答率の予測精度の向上も期待される．

加えて予測精度の向上のために few-shot learning の実施を検討している．本研究の結果から確信度と難易度は正答率と相関を持つ値ではなかった．そこで例として非常に簡単なタスクと難しいタスクを与えることで，LLM が確信度や難易度の評価のための基準を定められるのではないかと仮説を立てている．このようなステップの追加により，LLM に確信度や難易度という概念を理解させ，予測精度の向上を目指す．

謝辞 本研究の一部は，JSPS 科研費（JP24H00692, JP21H04877, JP21K18302）による助成を受けた．

文　献

[1] D. Sobania, M. Briesch, C. Hanna, and J. Petke, “ An Analysis of the Automatic Bug Fixing Performance of ChatGPT ,” In Proceedings of 2023 IEEE/ACM International Workshop on Automated Program Repair (APR), pp.23–30, 2023.

[2] J. Austin, A. Odena, M. Nye, M. Bosma, H. Michalewski, D. Dohan, E. Jiang, C.J. Cai, M. Terry, Q.V. Le, and C. Sutton, “Program synthesis with large language models,” ArXiv, p.arXiv:2108.07732, 2021.

[3] C.-Y. Su and C. McMillan, “Distilled gpt for source code summarization,” Automated Software Engineering, vol.31, pp.1–26, 2024.

[4] C. Le Goues, T. Nguyen, S. Forrest, and W. Weimer, “GenProg: A generic method for automatic software repair,” IEEE Transactions on Software Engineering, vol.38, no.1, pp.54–72, 2012.

[5] 柗本真佑，肥後芳樹，有馬諒，谷門照斗，内藤圭吾，松尾裕幸，松本淳之介，富田裕也，華山魁生，楠本真二，“高処理効率性と高可搬性を備えた自動プログラム修正システムの開発と評価，” 情報処理学会論文誌，vol.61，no.4，pp.830–841，2020.

[6] C.S. Xia, Y. Wei, and L. Zhang, “Automated program repair in the era of large pre-trained language models,” In Proceedings of International Conference on Software Engineering, pp.1482–1494, 2023.

[7] M. Chen, J. Tworek, and H.J. etal., “Evaluating large language models trained on code,” arXiv preprint arXiv:2107.03374, vol.abs/2107.03374, p.arXiv:2107.03374, 2021.

[8] J. White, Q. Fu, S. Hays, M. Sandborn, C. Olea, H. Gilbert, A. Elnashar, J. Spencer-Smith, and D.C. Schmidt, “A prompt pattern catalog to enhance prompt engineering with chatgpt,” ArXiv, vol.abs/2302.11382, p.arXiv:2302.11382, 2023.

[9] H.D.T. Nguyen, D. Qi, A. Roychoudhury, and S. Chandra, “Semfix: Program repair via semantic analysis,” In Proceedings of 2013 35th International Conference on Software Engineering (ICSE), pp.772–781, 2013.

[10] S.B. Hossain, N. Jiang, Q. Zhou, X. Li, W.-H. Chiang, Y. Lyu, H. Nguyen, and O. Tripp, “A deep dive into large language models for automated bug localization and repair,” In Proceedings of ACM Software Engineering., vol.1, no.FSE, pp.1471–1493, 2024.

[11] R. Just, D. Jalali, and M.D. Ernst, “Defects4J: a database of existing faults to enable controlled testing studies for java programs,” In Proceedings of the 2014 International Symposium on Software Testing and Analysis, pp.437–440, 2014.

[12] M. Xiong, Z. Hu, X. Lu, Y. Li, J. Fu, J. He, and B. Hooi, “Can llms express their uncertainty? an empirical evaluation of confidence elicitation in llms,” arXiv, p.arXiv:2306.13063, 2023.

[13] S. Farquhar, J. Kossen, L. Kuhn, and Y. Gal, “Detecting hallucinations in large language models using semantic entropy,” Nature, vol.630, no.8017, pp.625–630, 2024.

[14] Y. Jia and M. Harman, “An analysis and survey of the development of mutation testing,” IEEE Transactions on Software Engineering, vol.37, no.5, pp.649–678, 2010.

[15] J. Wei, X. Wang, D. Schuurmans, M. Bosma, b. ichter, F. Xia, E. Chi, Q.V. Le, and D. Zhou, “Chain-of-thought prompting elicits reasoning in large language models,” In Proceedings of Advances in Neural Information Processing Systems, vol.35, pp.24824–24837, 2022.

[16] Y. Wang, Q. Yao, J.T.-Y. Kwok, and L.M. shuanNi, “Generalizing from a few examples,” ACM Computing Surveys (CSUR), vol.53, pp.1–34, 2019.