客観的指標に基づく ローカル LLM のコード修復能力とモデル特性の分析

赤澤拓空[1] 松本和幸[2] 吉田稔[2]
[1]徳島大学大学院 創成科学研究科 理工学専攻 知能情報システムコース
[2]徳島大学大学院 社会産業理工学研究部
c612535058@tokushima-u.ac.jp {matumoto, mino}@is.tokushima-u.ac.jp

概要

近年，大規模言語モデルを用いた自動プログラム修正が高い性能を示しているが，クラウド型モデルにはプライバシーやコストの課題がある．本研究では，セキュアなローカル環境で動作する 10 種のモデルを対象に，コード修復能力とフィードバック応答特性を評価した．実験の結果，モデルサイズと修復性能は必ずしも相関せず，エラー情報を活用して性能を向上させる「学習者型」と，逆に低下させる「混乱型」という対照的な特性を確認した．特に中規模モデルが大規模モデルを凌駕する事例が観測され，パラメータ数よりもモデルの特性が重要であることが示唆された．本稿では，これらの知見に基づき，実環境でのモデル選択指針について議論する．

1 はじめに

ソフトウェア開発のライフサイクルにおいて，バグの発見と修正は多大なコストと時間を要する工程である．近年の大規模言語モデルの飛躍的な性能向上により，これらを自動化する自動プログラム修正技術が実用段階に入りつつある．ChatGPT や Claude といったクラウド型モデルを用いた手法は高い修正能力を示しているが，企業内の機密コードを外部サーバーへ送信する必要があるため，セキュリティやプライバシーの観点から導入が困難な場合が多い．この課題に対し，自社環境で構築・運用可能なローカル LLM（オープンソース LLM）が注目されている．ローカル LLM はデータ流出のリスクがなく，推論コストを固定化できる利点がある．しかし，クラウド型モデルと比較してパラメータ数が制限されるため，本来のプログラミング能力や，複雑な指示に従う能力には課題が残るとされてきた．既存研究の多くは，初期生成の正解率（Pass@1）に焦点を当てているが，実際の開発現場では，コンパイルエラーやテスト結果をもとに修正を繰り返すプロセスが一般的である．したがって，単発の生成能力よりも，外部からのフィードバックを受け入れ，自身の出力を改善する「自己修復能力」の評価が重要となる．そこで本研究では，3.8B から 46.7B までの多様なパラメータ数を持つ 10 種類の代表的なローカル LLM を選定し，テスト駆動型のフィードバックを与えた際の修復挙動を分析する．単純な再生成やエラーログの提示に加え，思考を促すプロンプトを用いた提案手法（Strategic Feedback）を適用することで，各モデルが持つ潜在的な修正能力を明らかにする．さらに，モデルサイズと修正性能の相関関係を調査し，スケーリング則だけでは説明できないモデル固有の特性について考察を行う．

2 関連研究

LLM を用いたコード修復では，ユニットテスト結果に基づくリランキング[1]や，言語的な自己内省フレームワーク[2]など，高性能なクラウド型モデルを対象とした手法が先行し，フィードバックの有効性が示されてきた．これらは，テスト駆動評価や言語的な報酬が，モデルの自己修正能力を引き出す上で重要であることを示唆している．最新の研究では，フィードバックの質やモデルの専門化に焦点が移っている．Dai らは多様な形式を網羅したベンチマークを通じ，テスト結果が最も有効な情報である一方，反復修正による改善は早期に停滞することを指摘した[3]．また，Sun らは大規模な実証研究により，パラメータ数の増大よりも，コードへの特化が修復性能を左右することを発見した[4]．さらに，小規模な特化モデルが大規模モデルを補完する独自の修正能力を持つという相互補完性も明らかにしている．本研究は，これらの知見を基盤とし，リソース制約下にあるローカル LLM の応答特性を「学習者型」と「混乱型」に分類・評価する点で独自性を有する．

3 提案手法

LLM が誤ったコードを修正するためには，単に「間違っている」という事実を知るだけでは不十分であり，「どこが」「なぜ」間違っているかを理解し，修正方針を立てる必要がある．本研究では，ローカル LLM の自己修復能力を段階的に検証するため，フィードバック情報の粒度と質が異なる二つの手法を提案する．これらは，人間の開発者がバグを修正する際の認知プロセス（事実の確認，原因の推論，修正の実行）をモデル上で再現することを意図している．

3.1 Detailed Feedback

Detailed Feedback は，モデルが生成したコードがテストに失敗した際，実行環境から得られる客観的なエラー情報を網羅的にフィードバックする手法である．具体的には，以下の 3 点の情報を構造化してプロンプトに追加する．

スタックトレース (Traceback): Python インタプリタが出力するエラー発生箇所と呼び出し履歴．これにより，モデルは例外の種類（TypeError，IndexError など）と発生行を特定できる．

入力値と期待される出力: 失敗したテストケースにおける関数の引数と，本来返すべき正解の値．

実際の出力 (Stdout/Stderr): 誤ったコードが実際に出力した値や, 実行時に発生した警告メッセージ．

この手法の狙いは，モデルに「コードが間違っている」という判定結果だけでなく，「どのような入力に対して，どのような誤った挙動をしたか」という証拠（Evidence）を提示し，データに基づいた修正を促す点にある．

3.2 Strategic Feedback

Strategic Feedback は，前述の詳細なエラーログに加え, モデルの推論プロセスを明示的に誘導する「思考誘導プロンプト（Chain-of-Thought Prompt）」を付与する手法である．中規模以下のローカル LLM においてエラーログのみを与えると，ログの内容を深く理解せずに表面的な記号の置換（例えば変数のリネームなど）のみを行い，論理的な修正に至らないケースが散見される．これを防ぐため，本手法ではエラーログの直後に以下の 2 段階の指示を追加する．

エラー原因の言語化: 「提供されたトレースバックと入出力の差分に基づき，なぜエラーが発生したのかを説明せよ」と指示し，修正コードを書く前に原因分析を行わせる．

修正プランの立案: 「コードの論理を修正するためのステップを箇条書きで示せ」と指示し，修正方針を明確化させる．このプロセスを経ることで，モデルは「反射的な修正」から「論理的な思考に基づいた修正」へとシフトし，複雑なアルゴリズムのバグに対しても有効な修正案を導出可能となる．

4 実験設定

4.1 データセット

本実験では，コード生成および修復能力の評価標準として広く用いられている HumanEval データセットを採用した.このデータセットは，関数のシグネチャ，機能説明（Docstring），標準解，および複数のユニットテストを含む 164 個の Python プログラミング課題から構成されている．

各課題の難易度は標準的なアルゴリズム実装レベルであるが，エッジケースを含む多様なテストケースが用意されており，モデルの論理的正確性を厳密に評価するのに適している．

4.2 評価対象モデル

ローカル環境(オンプレミス)での運用を想定し，パラメータ数が 3.8B から 46.7B までの範囲にある，計 10 種類のオープンソース LLM を選定した．

これらは Llama 系，Mistral 系，Command R 系など，異なるアーキテクチャや学習データを持つ代表的なモデル群である．

4.3 比較手法

モデルの自己修復能力を多角的に分析するため，フィードバック情報の質と量が異なる 4 つの実験条件を設定した.ベースラインとなる Zeroshot は初期生成のみの評価である.Simple Feedback は，テストの合否（Pass/Fail）のみを伝え，具体的なエラー内容は伏せた状態で再生成を促す条件である．

提案手法である Detailed Feedback は, エラートレースバック等の客観情報を与える条件，Strategic Feedback はそれに加えて思考誘導プロンプト（Chain-of-Thought）を与える条件である.これにより，単なる情報の付与だけでなく，推論の誘導が修復に与える影響を分離して評価可能とした．

4.4 評価指標

修復能力の定量的な評価には，以下の指標を用いる．

Pass@k (k=1，5，10) 本研究における Pass@k は，同一入力を並列生成する従来の定義とは異なり，「最大 k 回のフィードバック・修正ループを行う中で，k 回目までに正解コードに到達できた問題の割合」と定義する.特に Pass@10 は, モデルが対話を通じて最終的に解に到達できるかを示す指標であり，実用上の「デバッグ完了率」に相当する.

平均試行回数 (Average Attempts) 正解に到達した問題集合において，成功するまでに要した平均の修正回数である.この値が小さいほど, 少ないフィードバックで効率的にバグを修正できたことを示す.

相関係数 (Correlation Analysis) 「モデルサイズが大きければコード修復能力も高い」というスケーリング則の妥当性を検証するため，モデルのパラメータ数と Pass@10 の間の統計的な相関関係を分析する．本実験で扱うモデルサイズは離散的であり，外れ値の影響を受けやすいため，変数の分布を仮定しない非パラメトリックな手法であるスピアマンの順位相関係数を用いる．係数が 1 に近いほど強い正の相関（サイズ依存）があり，0 に近いほど相関がないことを示す.

フィードバック改善率 各モデルが外部からのフィードバックをどの程度有効に活用できたかを定量化するため，ベースライン（Zeroshot）に対する性能の伸び率を定義する．具体的には，Zeroshot 設定における Pass@1（初期生成の正解率）に対し，Strategic Feedback を用いて最大 10 回修正を行った際の Pass@10（最終的な到達正解率）がどの程度向上したかを，以下の式で算出する.

$$FIR = \frac{Pass@10_{Strategic} - Pass@1_{Zeroshot}}{Pass@1_{Zeroshot}}$$

この値が正に大きいほど，モデルが初期の誤りを自己修正する能力が高い「学習者型（Learner Type）」であることを意味する．一方，負またはゼロに近い値は，フィードバックが機能していないか，かえって性能を阻害している「混乱型（Confused Type）」であることを示唆する客観的な指標として用いる.

4.5 実験手順

本実験は以下のフローで行った. まず, HumanEval データセットから問題文と関数シグネチャ，およびユニットテストを抽出する前処理を行った．次に，各モデルに対して初期コードを生成させ，テストを実行した．テストに失敗した場合は，提案手法に基づいたフィードバックを生成し，再入力を通じて修正コードを取得した．このプロセスを最大 10 回繰り返し，得られたログから Pass@k および試行回数を算出し，モデル特性の分析を行った.

5 実験結果

5.1 手法ごとの修復成功率の比較

主要モデルにおける各手法の修復成功率（Pass@10）の推移を図 2 に示す．単純な再試行を行う Simple Feedback では，Zeroshot と比較して有意な性能向上は見られなかった．これに対し，詳細なエラー情報を与える Detailed Feedback，および思考誘導を行う Strategic Feedback においては，モデルによる性能差が顕著となった. 特に GPT-OSS (20B) は, 手法の高度化に伴い性能が単調増加し，最終的に 0.91 という高い正解率を達成した.

一方で，Llama 3 (8B) などの一部モデルでは，Detailed Feedback の時点で性能が Zeroshot を下回る現象（Performance Degradation）が確認された.

5.2 修復効率と試行回数の分析

正解に到達するまでに要した平均試行回数を分析した結果，修復の『効率性』においてもモデル特性が顕著に現れた. 学習者型の GPT-OSS 等は, 思考誘導を行う Strategic Feedback を用いることで, Detailed 設定時と比較して成功までの平均試行回数が減少した.これは, 思考の誘導が場当たり的な修正（Trial and Error）を抑制し，一度の修正の質を向上させたことを示唆している．一方，混乱型のモデルでは試行が上限の 10 回に達しても解に到達できない，あるいは極めて初期の段階で誤った修正を繰り返す停滞状態が観測された.

5.3 フィードバック応答特性によるモデル分類

各段階における Pass@10 の推移に基づき，ローカル LLM を以下の二つの特性に分類した.

学習者型 (Learner Type): GPT-OSS や DeepSeek Coder などが該当する.これらは提示情報の粒度が細かくなるほど，正解率が単調増加する特性を示した.

混乱型 (Confused Type): Command R や Mistral などが該当する.これらは Detailed Feedback の段階で性能が頭打ち，または低下する傾向が見られた.

具体的な数値として，GPT-OSS は Zeroshot 時の 0.56 から Strategic 設定で 0.91 まで向上したのに対

し, Command R は 0.75 から 0.68 へと低下しており，応答特性の違いが顕著に現れた.

5.4 パラメータ数と修復性能の相関

モデルサイズ（パラメータ数）と修復性能（Pass@10）の関係を図 3 に示す．全体のスピアマン順位相関係数は$\rho \approx 0.4$程度に留まり，強い相関は確認されなかった．特筆すべきは，20B の中規模モデルである GPT-OSS が，その 2 倍以上のサイズを持つ Mixtral (46.7B) や Command R (35B) を大きく上回る性能を示した点である．

この結果は，コード修復タスクにおいて「スケーリング則（Scaling Law）」は絶対的な支配要因ではなく，モデルのアーキテクチャや Instruction Tuning の質といった「モデルの性格」がより重要な決定要因であることを示唆している

6 考察

6.1 モデル特性とフィードバック受容性の関係

本実験で確認された「学習者型」と「混乱型」の二極化について，最新の研究知見を交えて考察する.

学習者型とモデルの専門化: GPT-OSS (20B) や DeepSeek Coder (6.7B) が示した高い受容性は，追加情報を文脈として正しく取り込み，内部知識と照合して修正案を洗練させる In-Context Learning 能力の高さに起因する.これは，Sun らが指摘した「モデルの規模（パラメータ数）よりも，コード特化型の微調整が APR 性能を左右する」という知見と整合する.つまり，コード構造への深い理解を持つモデルほど，フィードバックをノイズではなく論理的な手がかりとして正しく解釈できると言える.

混乱型と反復修正の限界点: 一方，Command R (35B) 等に見られた性能の停滞や低下は，Dai らが指摘した「反復修正における収穫逓減」の現象が，特定のモデルにおいては早期かつ顕著に現れることを示唆している.具体的には，プロンプトに含まれるエラーログの特定語彙に過剰反応して論理構造を破壊する現象やトークン増加による Attention の分散が原因と考えられる．これは大規模な汎用モデルであっても，APR という特化タスクにおいては複雑なフィードバックが逆に性能を阻害する負のフィードバック」として機能するリスクを示している.

6.2 スケーリング則の例外

一般に LLM の性能はパラメータ数に比例するとされるが, 本実験では 20B の中規模モデルが 46B の大規模モデルを凌駕する結果となった．これは，コード修復という特定タスクにおいては，単純な計算資源の量よりも，事前学習データの質（コード比率の高さ）や，論理的推論を強化するファインチューニングの有無が支配的な要因であることを示唆している．したがって，ローカル環境での運用においては，リソースを圧迫する巨大モデルよりも，タスク適応能力の高い中規模モデルを選定することが合理的である.

7 今後の課題

7.1 適応的なフィードバック生成

本研究では一律のフィードバックを与えたが，混乱型モデルに対しては情報の粒度を調整する必要がある．今後は，モデルの特性やエラーの種類に応じて，「詳細なログを与えるか」「単純な指摘に留めるか」を動的に切り替える適応的フィードバック（Adaptive Feedback）機構の構築が課題となる.

7.2 多言語および実用性の検証

本実験は Python のみを対象としたが, 実開発では Java や C++など多様な言語が用いられる．言語ごとの特性がモデルの修復能力に与える影響を検証する必要がある．また，HumanEval のようなアルゴリズム課題だけでなく, セキュリティ脆弱性の修正など, より実用的なタスクにおける有効性も検証していく予定である.

8 まとめ

本研究では，10 種類のローカル LLM を対象に，客観的指標に基づくコード修復能力の評価を行った. 実験の結果，単純な再生成では限界がある一方，思考誘導を用いた戦略的フィードバックにより，特定のモデルでは劇的な性能向上が可能であることを実証した．また，モデルサイズと修復性能の相関分析を通じ，パラメータ数よりもモデル固有の応答特性が重要であることを明らかにした. これらの知見は，プライバシーやコストの制約がある組織において，セキュアかつ効率的な自動プログラム修正システムを構築するための重要な指針となるものである.

謝辞

本研究は，JSPS 科研費 24K15193 の助成を受けたものである．

参考文献

[1] Bei Chen et al."CodeT: Code Generation with Generated Tests" in ICLR，2023.

[2] Noah Shinn et al."Reflexion: Language Agents with Verbal Reinforcement Learning" in NeurIPS，2023.

[3] Dekun Dai et al."FeedbackEval: A Benchmark for Evaluating Large Language Models in Feedback-Driven Code Repair Tasks" arXiv:2504.06939，2025.

[4] Jiajun Sun et al."Empirical Evaluation of Large Language Models in Automated Program Repair" arXiv:2506.13186，2025

A 付録

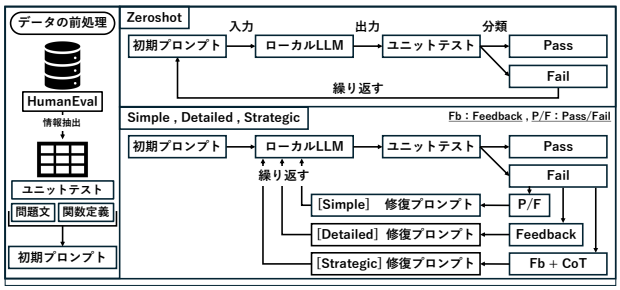
◆ **提案手法におけるプロンプトテンプレート**

DETAILED_FEEDBACK_PROMPT = """### Instruction:You are an expert Python debugger. You previously attempted to complete the function，but your generated code failed the unit tests with the following error.Carefully analyze the failure log and generate only the improved，corrected indented function body.Do NOT include the function signature or explanations.Problem Description (Docstring):{problem_description}Function Signature:Python{function_signature} Your Previous Attempt (Function Body):Python{previous_attempt_body}Unit Test Failure Log:{test_feedback}Corrected Function Body:"""

STRATEGIC_FEEDBACK_PROMPT = """### Instruction: You are an expert Python debugger. Your previous attempt failed. Follow these steps exactly:Analyze Failure Log: Carefully read the error log. What is the exact error type (e.g.，AssertionError，IndentationError)?Locate Error: Which line number or code segment is the root cause of this error?Formulate Hypothesis: Based on the error and the code，why did this error really happen?Propose Correction Plan: State the minimum changes needed to fix this specific error.Generate Code: Provide only the improved，corrected indented function body.Problem Description (Docstring):{problem_description}

Function Signature:Python{function_signature}Your Previous Attempt (Function Body):Python

{previous_attempt_body}Unit Test Failure Log:{test_feedback}1. Analyze Failure Log:2. Locate Error:3. Formulate Hypothesis:4. Propose Correction Plan:Corrected Function Body:"""



1 提案システムの概要図

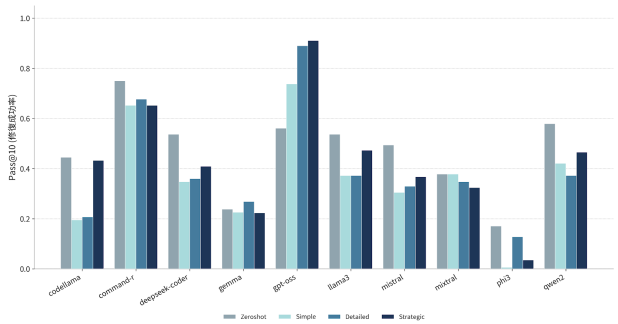
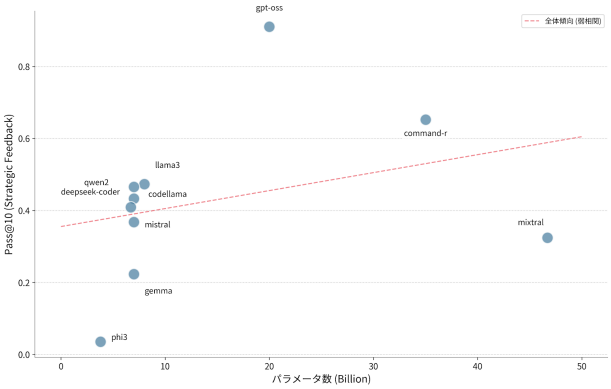


図 2 主要モデルにおける手法ごとの修復成功率



パラメータ数と修復性能の相関

表 1 評価対象モデル一覧

No.	モデル名	パラメータ数
1	GPT-OSS	20B
2	Llama3	8B
3	Qwen2	7B
4	Mistral	7B
5	Gemma	7B
6	Command R	35B
7	DeepSeek Coder	6.7B
8	CodeLlama	7B
9	Mixtral	46.7B
10	Phi3	3.8B