RAGAs による AI エージェントのコード生成における要件に対応した評価フレームワークの構築

長谷川 愛珠 [1] 倉光 君郎 [2]
[1] 日本女子大学大学院 理学研究科 [2] 日本女子大学 理学部
m2116064hm@ug.jwu.ac.jp kuramitsuk@fc.jwu.ac.jp

概要

大規模言語モデル（LLM）では，生成内容がユーザの質問に適切に答えているかを定量化する評価が重要課題である．近年，この背景から，RAG（Retrieval-Augmented Generation）を対象とした自動評価フレームワーク RAGAs（Retrieval-Augmented Generation Assessment）が提案され，忠実性や関連性に加えて，AI エージェントに適したワークフローの「目標達成」を判定する指標（Agent Goal Accuracy）も提供されている．また，近年，ソフトウェア開発への AI エージェントの導入が加速している．AI エージェントは人間の代わりに複雑な処理を行える一方で，その生成コードは一般に分量が多く，構造も複雑であるため，ユーザの要件を満たしているかを人手で検証することは容易ではない．本研究は，RAG の評価として設計された RAGAs を，AI エージェントが生成するコードの要件充足評価へ転用し，ユーザの要件の段階的なフェーズに対応した評価方法を提案する．

1 はじめに

近年，LLM の発展により，コード生成やコード修正，自動プログラミング支援といったタスクにおいて，LLM の活用が急速に進んでいる．これに伴い，LLM が生成したコードの品質をどのように評価するかは，信頼性の高いシステム構築において重要な課題となっている．従来のコード評価手法では，単体テストや pass@k（k 回の生成のうち少なくとも 1 回正解する確率）などの正解コードとの一致率といった実行結果に基づく評価が主流であったが，これらの手法はテストケースの網羅性に依存しやすく，可読性や仕様理解の正しさといった側面を十分に捉えることが困難である．

このような背景のもと，近年では LLM 自身を評価者として用いる「LLM as a judge」に基づく出力ベース評価手法が注目を集めている．LLM as a judge は，人間のコードレビューに近い観点から生成コードを評価できる可能性が示されており，複数の研究においてその有効性が報告されている．特に，複数エージェントによる協調的な判定や，相対評価による人間の選好の模倣など，従来の自動評価手法では困難であった多面的なコード品質評価が可能となりつつある．

2 コードの評価

本節では，LLM as a judge を用いたコードの評価手法を説明する．

2.1 LLM as a judge

RAGAs とは，LLM 自身に自動的に評価をさせる LLM を用いた出力ベースの評価指標である．これは，人間に近い評価性能を発揮することが示されている [1]．LLM as a judge による複数エージェントを用いた，人間の判断を模倣するコード評価が行われてきた．

2.2 LLM as a judge を用いたコードの評価手法の例

例えば Codeagent の研究では，`\tool{}`を用いることで複数エージェント間の協調を通じて，より深い理解と解決策の提案を目指しているのが特徴である [2]．また，CodeJudgeBench の研究では，コードの正しさ・堅牢性・可読性・推論可能性などを多面的に測定し 26 の LLM エージェントによる協調的な判定を通じて，評価している [3]．さらに，Auto-J の研究では，「人間の選好（Human Preference）」を模倣した判断を行い，LLM を単なるスコアリングモデルとして用いるのではなく，各候補コードの正確性，可読性，堅牢性などを比較し複数の出力の中から「最も良いものを選ぶ」という評価をしている．最後に，

CodeJudge の研究では LLM が「スローシンキング」を実行することで，詳細かつ信頼性の高い評価に到達するための様々な方法を検証する．[4]

一方で，LLM が生成したコードが「ユーザの要件を満たしているか」という観点に基づく評価は，依然として体系的な手法が十分に確立されていない．コード生成タスクにおいては，単に動作するか否かだけでなく，ユーザの指示や意図が正しくプログラムとして反映されているかを評価することが極めて重要である．

そこで本研究では，自然言語処理分野において RAG の評価を目的として提案されたフレームワークである RAGAs を活用し，コード生成の出力ベース評価に適用する．また，ユーザの要件の段階的なフェーズに対応した評価方法を提案する．

3 RAGAs を用いた実験手法

AI エージェントの出力が，ユーザの要件を満たしているか否かを評価する手法として我々は評価フレームワークである RAGAs に注目した．RAGAs は多様なメトリクスにより段階的な評価を可能にする．

3.1 概要

RAGAs の評価には以下の 4 つのパラメータが用いられる．

- user_input（プロンプト）
  ユーザが RAG システムに入力した指示．
  例：東京タワーはいつ建設されたか
- retrieved_contexts（取得されたコンテキスト）
  ユーザの指示に基づいて，RAG システムの検索機能が外部データベースから検索・取得してきた文書のリスト．
  例：東京タワーは 1958 年に竣工し，主にテレビ・ラジオ放送の電波送信を目的として建設された．
- response（モデルの出力）
  user_input と retrieved_contexts を入力として，LLM が最終的に生成した出力テキスト．
  例：東京タワーは 1958 年に完成した．
- reference（期待する出力）
  ユーザの指示に対する理想的な正解．
  例：1958 年に竣工した．

これらは LLM に入力され，LLM が生成物の評価を行う．RAGAs はこれらのパラメータを評価用 LLM に入力し，LLM による推論や判定に基づき，組み込み関数を用いてスコアを算出する．また，対象ユースケースが大幅に増え，RAG だけに留まらず AI エージェントや SQL や Nvidia Metrics など 7 つのユースケースの評価が追加され，RAGAs によって AI エージェントの生成物の評価も可能になった．我々は，RAGAs に搭載された AI エージェントのメトリクスである Agent Goal Accuracy に着目した．

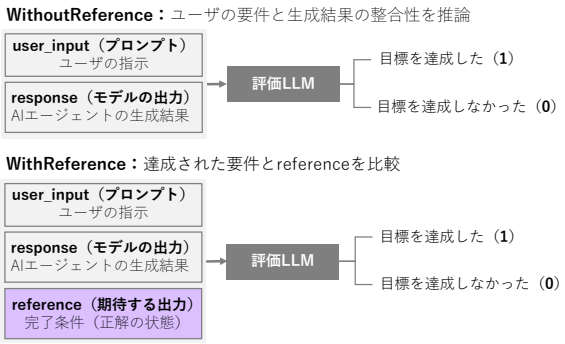

図 1 WithReference と WithoutReference の概要

3.2 Agent Goal Accuracy

Agent Goal Accuracy[1)]は，AI エージェントの生成物がユーザの要件を満たしているかを評価するメトリクスである．これは二値指標であり，「1」は AI エージェントが要件を満たしたことを示し，「0」は AI エージェントが要件を満たさなかったことを示す．

3.2.1 評価方法

Agent Goal Accuracy の WithReference と WithoutReference の評価方法の概要を図 1 に示す．

WithoutReference

WithoutReference はユーザーの要件を特定し満たしている要件から AI エージェントの性能を評価する．ここでは，user_input の人間とのやり取りから望ましい結果が推測される．

- user_input
  AI エージェントに対する満たすべき要件の定義（ユーザの指示）．
  例：ジョンに会議の依頼メッセージを送信
- response
  AI エージェントが実行した「実際の結果」や「行動の履歴」．

1) https://github.com/vibrantlabsai/ragas/blob/main/src/ragas/metrics/_goal_accuracy.py

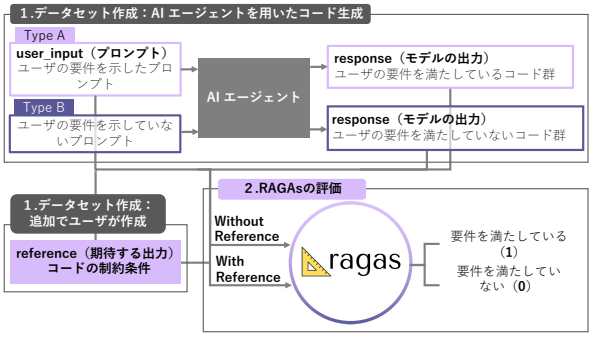

図 2　実験の概要

1. 投票処理後にメモリ上のデータが適切にクリアされること
2. 管理者でも投票者を特定できない仕組みになっていること
3. アクセスログ、エラーログ、データベースログにIPアドレスやセッション情報が記録されていないこと
4. ユーザー入力がシステムプロンプトに影響を与えないこと
5. 悪意のあるプロンプトや特殊文字を送信して適切にサニタイズされること
6. 通信（WSS使用）とデータ保存時の暗号化が機能していること
7. 不要なデータが自動削除される仕組みがあること

図 3　7 種類のユーザの要件

ファイル名	内容
README.md	システム概要・使い方
client.html	投票操作を行う投票者画面
admin.html	投票結果表示を行う管理者画面
server.js	Node.jsサーバー（投票データ暗号化・匿名性担保・不要データ自動削除）
client.js	クライアント用JS（WSS通信・サニタイズ対応）
package-lock.json	依存パッケージを固定するロックファイル
package.json	プロジェクト設定（名称・依存・スクリプトなど）

表 1　AI エージェントで生成した際のファイル構成

例：ジョン様，こんにちは．お元気でいらっしゃるでしょうか．重要な事項について話し合うため，会議を設定させていただきたいと思います．ご都合がよろしいでしょうか．よろしくお願いいたします．

WithReference

WithReference は LLM がユーザの要件を特定し満たしている要件から LLM の性能を評価するために，user_input と response に加えて reference をパラメータとする．reference は理想的な結果として使用される．reference とワークフロー終了時に満たされた要件を比較することで算出する．

- reference
  要件が満たされたとみなすための「完了条件（正解の状態）」．
  例：ジョンに会議の依頼メッセージを送信した

WithoutReference と WithReference の特徴は，WithoutReference はユーザの指示とモデルの出力から LLM が評価するため，LLM 自身に評価基準を委ねていることである．一方で，WithReference はユーザ側が reference を用いて要件を明示するため，人間側が評価基準を設ける．

4　実験

自然言語タスクにおいて用いられている RAGAs の評価指標をコード生成タスクに適用し，その有効性を検証した．

具体的には，Agent Goal Accuracy の評価に必要なデータセットを AI エージェントで生成し，そのデータに対する RAGAs の評価結果と，人手による評価の結果を比較し，有効性を検証した．上記に関する実験概要を図 2 に示す．

4.1　データセット作成

本実験では，ユーザの要件として，ソフトウェア開発における重要な要件であるセキュリティ要件に着目し，匿名性を担保した投票システムを題材にデータセットを作成した．セキュリティ要件は設計段階での考慮が不可欠であり，記述が抽象的かつ曖昧になりやすいという特徴を持つ．本研究では，匿名性の確保が本質的な課題となる匿名投票システムを題材とすることで，セキュリティ要件を中心とした要件記述の分析に適したデータセットを構築した．

RAGAs はユーザの要件の段階的なフェーズに対応した評価が可能か検証するため，全てのユーザの要件を満たしているデータセットと 7 種類のユーザの要件をそれぞれ満たしていないデータセットを作成した．7 種類のユーザの要件を図 3 に示す．

user_input

user_input はユーザの要件を満たしているコード群を生成した際の入力プロンプトを用いた．

response

Visual Studio Code[2] のエージェントモードで生成した．エージェント内の LLM は GPT-4.1[3] を使用した．全てのユーザの要件を満たしているコード群と 7 種類のユーザの要件をそれぞれ満たしていないコード群を生成した．それぞれのファイル構成は表 1 のように統一した．

生成したコードがユーザの要件を満たしているかどうかを判定する方法としては，サーバー起動テストを行い，人手で内容を確認し，要件を満たしているか否かを判定した．

2）https://code.visualstudio.com/
3）https://openai.com/index/gpt-4-1/

ファイル名	全ての要件を満たす		要件1欠如		要件2欠如		要件3欠如		要件4欠如		要件5欠如		要件6欠如		要件7欠如	
	Without Reference	With Reference	Without Reference	With Reference	Without Reference	With Reference	Without Reference	With Reference	Without Reference	With Reference	Without Reference	With Reference	Without Reference	With Reference	Without Reference	With Reference
README.md	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
index.html	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
admin.html	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
server.js	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
client.js	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
package-lock.json	0	0	1	0	1	0	1	0	0	0	1	0	0	0	1	0
package.json	1	0	0	0	1	0	0	0	1	0	1	0	1	0	1	0

図 4　RAGAs の評価結果

reference

reference は 7 種類のユーザの要件である．これは，「ユーザの要件を満たしているコード群」が満たしているセキュリティ要件である．

4.2　評価

ユーザの要件を満たしているコード群と，ユーザの要件を満たしていないコード群のそれぞれに対して，RAGAs を用いて評価を行い，各ファイルごとに 0/1 の判定を行った．ここで，ユーザの要件を満たしているコード群はすべて 1 の判定をされるのが好ましく，対して要件を満たしていないコード群はすべて 0 の判定をされるのが好ましい．判定結果を正解と比較することで RAGAs の評価はユーザの要件を段階的なフェーズに対応した評価を行えるか考察を行う．

評価に用いた LLM は，gpt-4o-mini[4] である．

4.2.1　評価結果

RAGAs の評価結果を図 4 に示す．全体的に見比べると「README.md」，「package.json」，「package-lock.json」で「1」になっている場合と「0」になっている場合が見受けられる．つまり，満たしていないユーザの要件によって評価結果に違いが出た．この結果から，ユーザ要件の違いに応じて評価結果が変化することが確認され，RAGAs による段階的フェーズ対応評価が可能であると考えられる．

4.2.2　考察

実験を通して，RAGAs は自然言語からユーザの要件を満たすかどうかの判定ができるだけでなく，コードからも判定できることが分かった．そして，ソフトウェア開発における AI エージェントの生成コードの評価を段階的なフェーズに対応した評価ができることが示唆された．具体的には，ユーザの要件を 7 つに細分化しそれぞれの要件を満たしていないデータを AI エージェントで生成し，全てのユーザの要件を満たすデータと比較した．その結果，満たしてない要件が違うと，評価結果にも差が現れた．

今後の展望

今後は，ユーザの要件を満たしているか否かを単に判定するだけでなく，要件文書を参照する過程において，どの段階に問題が存在するのかを分割して評価することを目指す．RAGAs は段階的な評価を可能にするフレームワークであるため，将来的には，要件記述そのものの品質，適切な情報が検索されているか，およびそれらに基づいた回答が生成されているかといった各要素を個別に評価できると考えられる．

5　関連研究

LLM ベースのエージェントは，ソフトウェア開発を含む多様なタスクで研究が進み [5, 6]，マルチエージェント開発支援なども提案されている [7, 8]．

6　むすびに

本研究は，AI エージェントが出した生成物がユーザの要件を満たしているか評価するために，RAGAs によるコードの評価が適応できるか検証した．RAGAs の Agent Goal Accuracy の WithReference と WithoutReference で評価した結果，RAGAs が様々なユースケースに対応したメトリクスが整備されていることと同じように，段階的なフェーズに対応した評価が可能であることを提案した．

謝辞

本研究を進めるにあたり，有益な議論および貴重なコメントをいただいた NTT 株式会社の秋信有花氏，佐藤美唯氏に感謝いたします．

参考文献

[1] Ruiqi Wang, Jiyu Guo, Cuiyun Gao, Guodong Fan, Chun Yong Chong, and Xin Xia. Can llms replace human evaluators? an empirical study of llm-as-a-judge in software

4）https://openai.com/ja-JP/index/gpt-4o-mini-advancing-cost-efficient-intelligence/

engineering. **Proceedings of the ACM on Software Engineering**, Vol. 2, No. ISSTA, pp. 1955–1977, 2025.

[2] Xunzhu Tang, Kisub Kim, Yewei Song, Cedric Lothritz, Bei Li, Saad Ezzini, Haoye Tian, Jacques Klein, and Tegawendé F Bissyandé. Codeagent: Autonomous communicative agents for code review. **arXiv preprint arXiv:2402.02172**, 2024.

[3] Hongchao Jiang, Yiming Chen, Yushi Cao, Hung-yi Lee, and Robby T Tan. Codejudgebench: Benchmarking llm-as-a-judge for coding tasks. **arXiv preprint arXiv:2507.10535**, 2025.

[4] Weixi Tong and Tianyi Zhang. Codejudge: Evaluating code generation with large language models. **arXiv preprint arXiv:2410.02184**, 2024.

[5] Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. The rise and potential of large language model based agents: A survey. **Science China Information Sciences**, Vol. 68, No. 2, p. 121101, 2025.

[6] Junwei Liu, Kaixin Wang, Yixuan Chen, Xin Peng, Zhenpeng Chen, Lingming Zhang, and Yiling Lou. Large language model-based agents for software engineering: A survey. **arXiv preprint arXiv:2409.02977**, 2024.

[7] Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. Chatdev: Communicative agents for software development. In **ACL**, 2024.

[8] Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. Metagpt: Meta programming for a multi-agent collaborative framework. In **ICLR**, 2024.