生成 AI の「図が思いどおりにならない」を解決する

- Diagrams as code ツールへの「推論と行動」（ReAct）適用 -

研 究 員：中川　幸人（アビームコンサルティング株式会社）
主　査　：石川 冬樹（国立情報学研究所）
副 主 査：徳本 晋（富士通株式会社）
アドバイザー：栗田 太郎（フリー株式会社）

研究概要
生成 AI に図の生成を依頼しても，要求と異なる出力や描画失敗が発生し，試行錯誤や手戻りが生じることが多い．これは，図が合意形成・理解促進・分析のための重要な表現手段であるにもかかわらず，生成結果の品質が安定しないという実務上の課題である．本研究では，コードから図を生成する Diagrams as code ツールに「推論と行動」（ReAct）フレームワークを適用することで，要求どおりの図を生成する手法「ReAct-Dac」を考案した．実験によって，図の要求充足率の向上を確認した．

1. はじめに

筆者はアーキテクトとして日々，UML や ArchiMate 等のモデリング言語でモデルを構築し，図として可視化して関係者に共有している．生成 AI によってモデルの図示に要する手間を減らせれば，議論や文書作成を加速できる．特に Diagrams as code（以下，Dac）はモデリング言語の記法に基づく図を得やすく，生成 AI に Dac ツール(例:PlantUML,Mermaid 等）へ入力するコードを作らせる用途と相性がよい．一方で，生成 AI が出力したコードは，Dac ツールで描画するとエラーとなる場合がある．描画できたとしても，要素や関係，制約，表記などが要求と一致しないことがある．

そこで本研究では，推論によるコード生成と Dac ツール実行を反復し，描画結果を根拠に生成コードを改善する手法「ReAct-Dac」を提案する．評価の結果，図の要求充足率の向上，描画エラーの減少，ならびにトークン数増加のトレードオフを確認した．

2. 背景と問題

2.1 背景

モデルの図示は，関係者間の共通認識を迅速に形成し，議論や意思決定を前進させ，手戻りを減らすうえで有効である．したがって，図作成の負荷を下げ，議論や文書作成を加速する手段として生成 AI を活用できることは魅力的である．

一方で，合意形成に用いる図は，読み手によって解釈がばらつかないことが重要である．UML や ArchiMate 等の記法に従えば，図形や線の意味があらかじめ定義されているため，関係者が同じ内容を同じように読み取りやすい．したがって，実務で用いる図では記法への準拠が重要となる．

生成 AI に図を生成させる方法は大きく二つある．第一は図を画像として生成させる方法であり，第二は Diagrams as code のコードを生成させ，ツールで描画する方法である．解釈のばらつきを抑えるという点では，後者は画像生成に比べて記法に沿った表現を得やすいため，アーキテクトの立場からは好ましい．両方式の特徴を表 1 に示す．

表 1 生成 AI による図生成方式の比較

図生成方式	メリット	デメリット
画像生成	・自由な発想を表現しやすく，想定外のアイデアを得やすい.	・自由度が高い分，記法に従わせにくく，解釈のブレが生じやすい.
Diagrams as code	・記法に沿った表現を得やすく，解釈のブレを抑えやすい. ・テキストとして扱えるため，レビューやバージョン管理が容易である.	・生成したコードがツールで描画エラーとなる，または描画できても図が要求を満たさない場合がある.

2.2 扱う問題

本研究は，Diagrams as code 方式の利点を維持したまま，「図が思い通りにならない」という課題の緩和を目的とする．Dac 方式では，生成 AI に図への要求を自然言語で与えてコードを生成させ，ツールで描画する．しかし出力されたコードは描画エラーとなって図が得られない場合がある．たとえ描画できても，要素・関係・制約などが要求と一致しないことが多い．結果として，利用者はプロンプトの調整やコードの修正を手作業で重ねることになり，図作成を省力化するという狙いが十分に達成できない．したがって，Dac の利点を維持したまま，要求どおりの図に安定的に到達させる手法が必要である．

この問題の主な原因は，AI が自ら生成したコードについて，要求充足を判断・確定できない点にあると考える．Dac では，コードは描画されて初めて図として解釈されるため，たとえ高性能な AI が妥当な推論に基づいてコードを生成・評価しても，描画結果を確認しない限り，要素・関係・制約などの要求が満たされているかは確定できない．実際，わずかなコードの誤りで描画エラーが生じ，図そのものが得られないことがある．

3. 提案

3.1 ツール実行結果を手がかりに生成内容を更新する

人間が Dac で作図する際は，コードを書いて描画し，結果を見て修正するという流れを自然にとる．描画結果は，不足要素，余計な要素，誤った関係や制約など，要求からの逸脱を具体化するため，次に直すべき点を決める根拠になる．したがって生成 AI でも，ツール実行を組み込み，描画結果を手がかりに生成内容を改善していく手順が有効である．

3.2 ReAct の適用

ReAct[1]は，推論（Reasoning）と行動（Acting，外部ツールの実行）を組み合わせ，得られた結果を根拠に次の推論を進める枠組みである．本研究では，Dac コード生成を推論，Dac ツールの実行を行動に対応付ける．ツール実行により描画結果という観測可能な根拠が得られるため，推論のみでは不確実だった要求充足の判断と修正方針を具体化でき，要求どおりの図へ近づけやすくなる．

3.3 提案 ReAct-Dac

以上を踏まえ，本研究では ReAct を Dac 方式の図生成に適用し，要求充足率の向上を目指す手法 ReAct-Dac を提案する．ReAct-Dac は，次の手順で生成するコードを改善する．

1. 自然言語要求に基づき Dac コードを生成する
2. Dac ツールで実行し，描画結果を得る
3. 描画結果を要求と照合し，修正点を特定する
4. 修正点を 1 のプロンプトに追加して再実行（修正がなければ終了）

ReAct-Dac コミュニケーション図

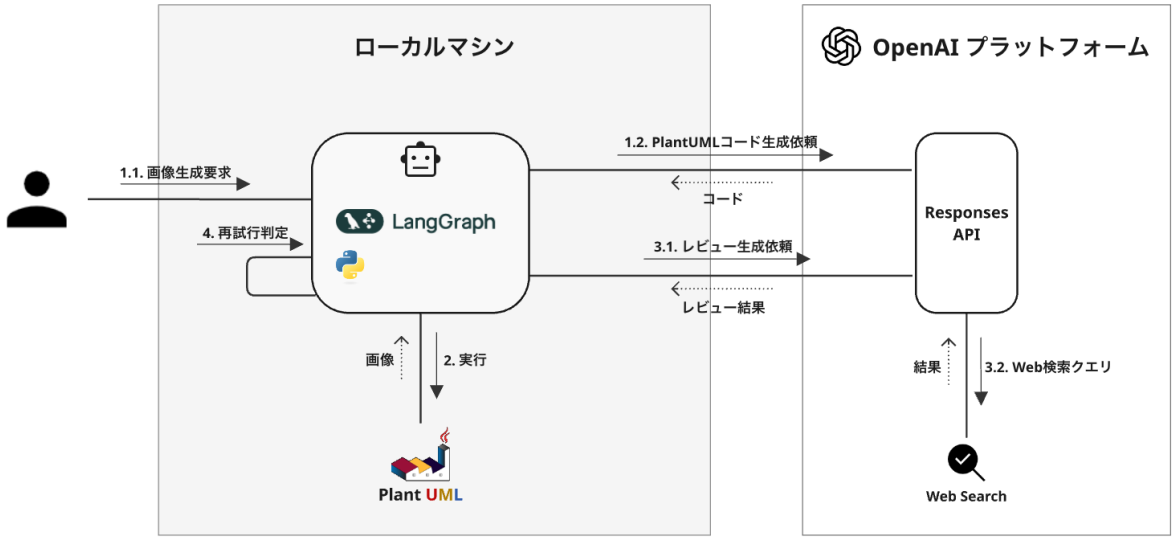


図 1 ReAct-Dac のハイレベルコミュニケーション図

本手法は，Dac の利点である記法に基づく解釈の安定性とテキストとしての管理容易性を保ったまま,描画結果という観測可能な根拠を用いて生成内容を改善する.これにより，利用者の手作業によるプロンプト調整やコード修正の負担を減らし，要求どおりの図をより安定的に得ることを目指す.

表 2 ReAct-Dac の特徴

区分	特徴	詳細
利用価値	一度の指示で，レビューと修正が反復された完成版の図画像を取得できる.	利用者は,所望の図の説明や図への要求記述を一度与えるだけでよい．その後は AI が内部で，コード生成，描画，結果確認，再生成を繰り返し，要求を満たすように図をブラッシュアップしたうえで最終画像を返す.
実装上の工夫	コード生成とレビュー依頼のプロンプトを分離し，レビューを利用者視点で行う.	手順 1 のコード生成プロンプトには,図への要求記述と,必要に応じて前回特定したコードの修正点を与える．一方，手順 3 のレビュー依頼プロンプトには，図への要求記述と描画結果を与え，コードを参照せずに評価するよう指示する.これにより，コードの出来ではなく，最終的に得られた図画像が要求を満たしているかどうかを,利用者の視点で確認できる.

各プロンプトは，付録 1 に掲載した ReAct-Dac のソースコード中に記載している（コード生成プロンプトは 134 行目以降，レビュー依頼プロンプトは 242 行目以降）.

4. 実験

本章では，提案手法 ReAct-Dac の有効性を，ベースラインである一回完結方式と比較して評価する．ベースラインは，生成 AI が自然言語要求から Dac コードを生成し，Dac ツールで描画して終了する方式である．描画結果を踏まえた再生成を行わないため，以降これを一回完結方式と呼ぶ．

主要評価指標は要求充足率とし，併せて費用（API トークン課金）の増加を測定する．

4.1 研究課題

本研究は以下の研究課題に答える．

- RQ1（機能要求）
  ReAct-Dac は，一回完結方式と比べて，モデルの要求充足率を改善できるか．
- RQ2（非機能要求）
  ReAct-Dac は，一回完結方式に比べて，トークン数はどれだけ増加するか．

4.2 実験内容

4.2.1 実行環境

本研究では，Dac ツールとして PlantUML を用い，使用する OpenAI モデルの違いを 2 つの実行環境として扱う．いずれの環境でも，生成された PlantUML コードを同一条件で実行し，その描画結果を評価に用いる．

表 3 実行環境一覧

実行環境名	使用する OpenAI モデル	LLM に与えるパラメータ	使用する Dac ツール
E1	gpt-4.1-2025-04-14（非推論モデル）	temperature=0.2	PlantUML 1.2026.2beta1（Java 1.7）
E2	gpt-5.2-2025-12-11（推論モデル）	reasoning={"effort": "medium"}	

参考までに，2026 年 1 月時点の OpenAI 公式のモデル解説の抜粋を以下に示す．

表 4 OpenAI モデル情報（抜粋）

モデル名	Knowledge Cutoff	推論トークンの生成有無
gpt-4.1	2024-06-01	なし
gpt-5.2	2025-08-31	あり

4.2.2 タスク設計

図種と要求種別の組合せにより，4 つのタスクを設計する．図種は，UML 2.5.1 における図の大別である構造図と振る舞い図を対象にするため，構造図からクラス図，振る舞い図からシーケンス図を選定する．要求種別は下記の二つとする．

- モデル要求
  要素・関係・制約など，モデルの意味内容に関する要求である．
- スタイル要求
  レイアウト，色，フォント，形状など，図の表現スタイルに関する要求である．

要求種別を分ける理由は，実務上，モデル要求とスタイル要求とで要求の必須性が異なるためである．モデル要求は図の意味内容の正しさに関わる必須要件であり，満たされなければ図は利用に耐えない．一方，スタイル要求は図の表現品質に関わる付加的要件であり，満たされない場合でも図の意味内容自体は保持されうる．

以上より，図種 2 種類（クラス図，シーケンス図）と要求種別 2 種類（モデル要求，スタイル要求）を組み合わせ，計 4 タスク（2×2）を用意する．

表 5 タスク一覧

タスク名	作成対象の図種	要求種別
CLS_M	クラス図	モデル要求
CLS_M+S		モデル要求＋スタイル要求
SEQ_M	シーケンス図	モデル要求
SEQ_M+S		モデル要求＋スタイル要求

また，タスクの複雑さについては，図の大きさ（要素数）とスタイル指定の難度の両面から，経験上，実務で用いるのに十分な水準となるように設定した．タスクの詳細は付録 2 に示す．

4.2.3 実行方法

各タスクを，2 つの実行環境（gpt-4.1，gpt-5.2）で実行する．各タスクは一回完結方式と ReAct-Dac の 2 手法でそれぞれ 2 回ずつ試行する．これは，生成 AI の出力に含まれるランダム性の影響を低減するためである．したがって，総試行数は以下となる．

総試行数：2 環境 × 4 タスク × 2 手法 × 2 回 ＝ 32 試行

4.2.4 評価方法

各タスクの自然言語要求から要求チェックリストを作成し，生成された図が各項目を満たすかを判定する．要求充足率は，チェックリストの充足率として次式で算出する．

要求充足率 ＝ 充足項目数 / 総項目数

各タスクについて 2 回の要求充足率を算出し，その平均を当該タスクの代表値とする．一回完結方式と ReAct-Dac の平均要求充足率を比較し，RQ1 に回答する．RQ2 に関しては，各試行における API トークン数を記録し，ReAct-Dac 適用による増加量を評価する．チェックリストは付録 3 に掲載する．

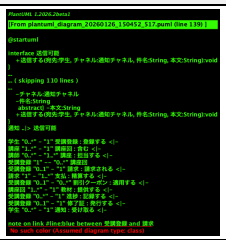
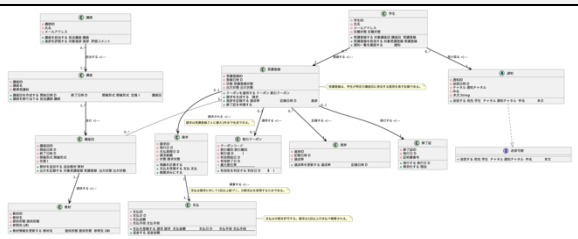
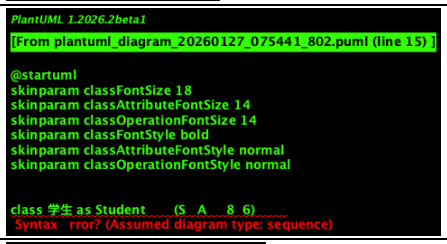
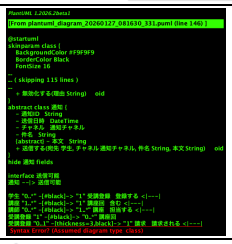
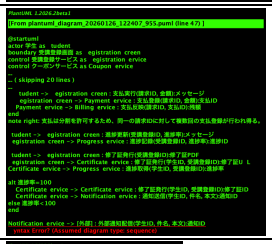
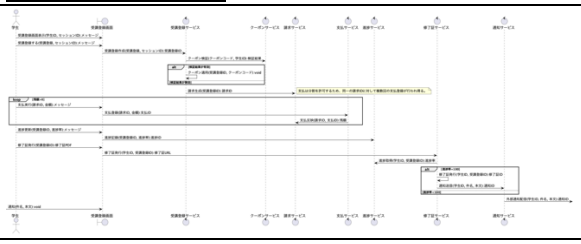
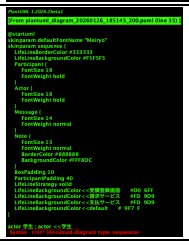
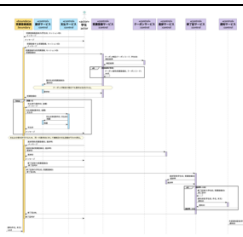
4.3 実験結果

4.3.1 実行環境 E1（gpt-4.1）

各タスクについて，一回完結方式および ReAct-Dac の実行結果を表 6 に示す．なお，表 6 には 2 回試行のうち代表として 1 回目の結果のみを掲載し，2 回目は傾向が同様であるため割愛した．

また，黒地に緑字で示した画像は描画失敗時のエラー出力画面であり，それ以外の画像は描画成功時に得られた結果図である．本文では，比較のために全体像を把握しやすい大きさで画像を掲載し，個々の結果の詳細を確認するための拡大画像は付録 4 に掲載した．

表 6 実行環境 E1 における各手法の実行結果（各 2 回試行のうち 1 回目）

タスク名	一回完結方式の実行結果	ReAct-Dac の実行結果
CLS_M		
CLS_M+S		
SEQ_M		
SEQ_M+S		

要求充足率の変化を図 2 に示す．要求充足率は，2 回試行した結果の平均値である．

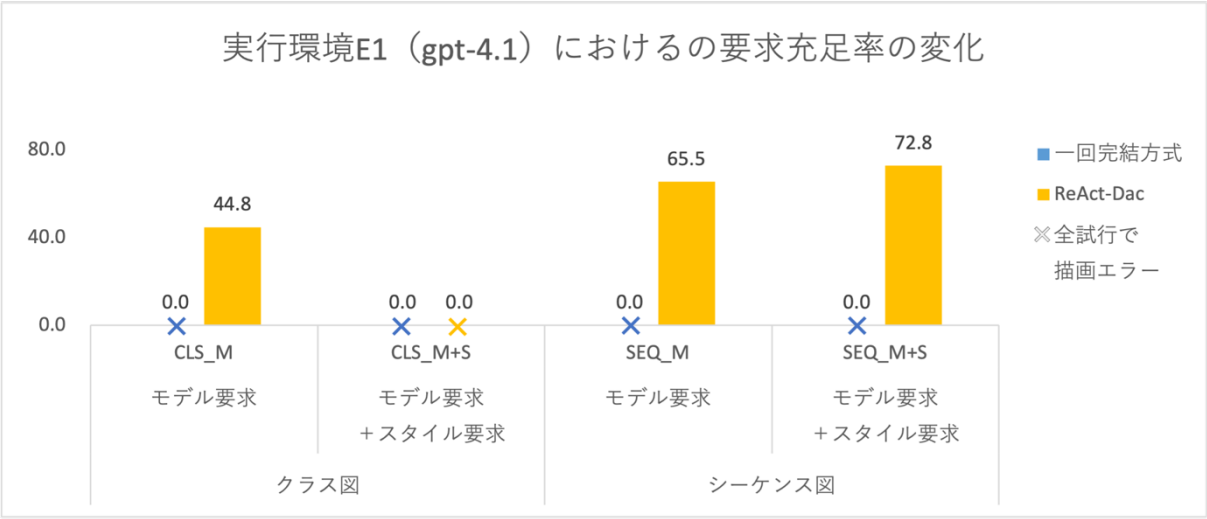


図 2 実行環境 E1（gpt-4.1）における要求充足率の変化

ReAct-Dac は，一回完結方式に比べて要求充足率が改善した（4 タスク平均で＋45.8pt）．クラス図では，モデル要求タスクは改善したが，モデル要求＋スタイル要求タスクは改善

しなかった．シーケンス図では，いずれのタスクも改善した．

トークン数の比較結果を表 7 に示す．トークン数は 2 回試行した結果の平均値である．

表 7 実行環境 E1（gpt-4.1）におけるトークン数比：ReAct-Dac/一回完結方式

タスク名	一回完結方式			ReAct-Dac			倍率 ReAct-Dac/一回完結方式		
	Input	Cached input	Output	Input	Cached input	Output	Input	Cached input	Output
CLS_M	5207	2560	2006	250460	129280	12887	**48.1**	**50.5**	**6.4**
CLS_M+S	5975	0	17763	279524	119488	13202	**46.8**	**NA**	**0.7**
SEQ_M	3593	1536	1368	64411	17664	4236	**17.9**	**11.5**	**3.1**
SEQ_M+S	4357	2048	2273	193047	94656	12493	**44.3**	**46.2**	**5.5**

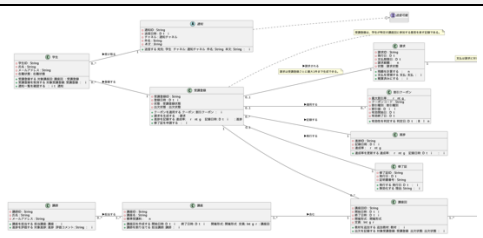
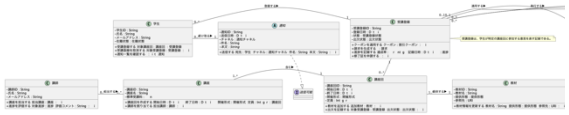
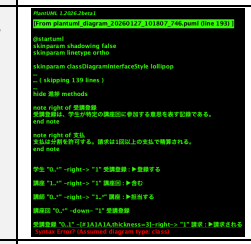
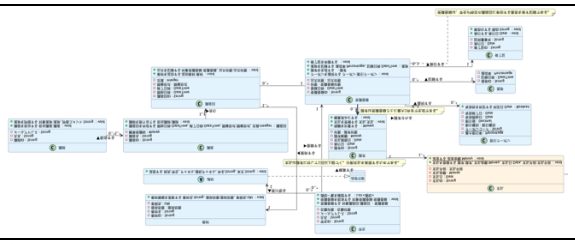
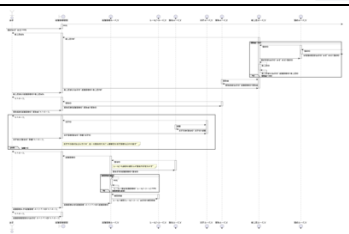
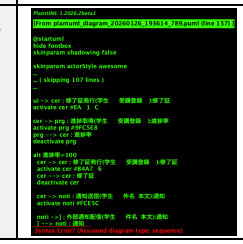
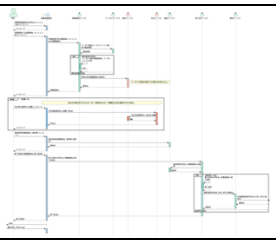
ReAct-Dac は，一回完結方式に対して，4 タスク平均のトークン数が Input で 39 倍，Cached input で 36 倍，Output で 4 倍であった．

4.3.2 実行環境 E2（gpt-5.2）

各タスクについて，一回完結方式および ReAct-Dac の実行結果を表 8 に示す．なお，表 8 には各手法 2 回試行のうち 1 回目の結果を示す．2 回目は概ね同様の傾向であったため割愛したが，タスク SEQ_M では 1 回目は描画成功，2 回目は描画失敗となり結果が一致しなかった．

また，黒地に緑字で示した画像は描画失敗時のエラー出力画面であり，それ以外の画像は描画成功時に得られた結果図である．本文では，比較のために全体像を把握しやすい大きさで画像を掲載し，個々の結果の詳細を確認するための拡大画像は付録 5 に掲載した．

表 8 実行環境 E2 における各手法の実行結果（各 2 回試行のうち 1 回目）
*体裁統一のため，一部をトリミングしている．

タスク名	一回完結方式の実行結果	ReAct-Dac の実行結果
CLS_M		
CLS_M+S		
SEQ_M		
SEQ_M+S		

要求充足率の変化を図 3 に示す．要求充足率は，2 回試行した結果の平均値である．

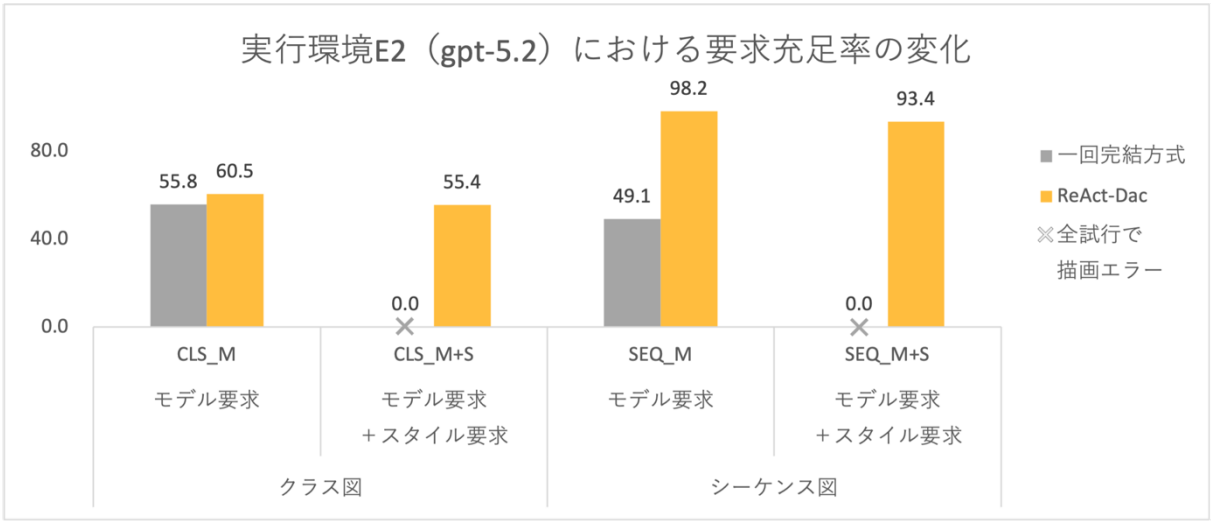


図 3 E2 環境（gpt-5.2）における要求充足率の変化

ReAct-Dac は，一回完結方式に比べて要求充足率が改善した（4 タスク平均で＋50.6pt）．

全てのタスクで改善が確認された．

トークン数の比較結果を表 9 に示す．トークン数は 2 回試行した結果の平均値である．

表 9 実行環境 E2（gpt-5.2）におけるトークン数比：ReAct-Dac/一回完結方式

タスク名	一回完結方式			ReAct-Dac			倍率 ReAct-Dac/一回完結方式		
	Input	Cached input	Output	Input	Cached input	Output	Input	Cached input	Output
CLS_M	9332	0	4825	212493	128640	29653	22.8	NA	6.1
CLS_M+S	10100	9984	10936	474300	309120	66313	47.0	31.0	6.1
SEQ_M	7718	3776	5173	48480	14528	13095	6.3	3.8	2.5
SEQ_M+S	8482	8320	8289	224853	143104	33180	26.5	17.2	4.0

ReAct-Dac は，一回完結方式に対して，4 タスク平均のトークン数が Input で 26 倍，Cached input で 17 倍，Output で 5 倍であった．

4.4 議論

4.4.1 RQ1 要求充足率を改善するか

ReAct-Dac は，実行環境 E1（gpt-4.1，非推論モデル）では 4 タスク中 3 タスクで改善が確認され，残る 1 タスクは変化がなかった．実行環境 E2（gpt-5.2，推論モデル）では 4 タスクすべてで改善が確認された．以上より，ReAct-Dac は要求充足率の向上に寄与することが示唆される．

一回完結方式の結果に注目すると，実行環境 E1 では 4 タスクすべてが描画エラーとなった．実行環境 E2 では，モデル要求のみのタスクは描画に成功し，スタイル要求を含むタスクは依然として描画エラーのままであった．このことから非推論モデルよりも推論モデルの方が正確なコード生成が可能であり，かつスタイル要求のコード生成の難度が相対的に高いことが示唆される．これは記法上の制約や表現の微細な指定を多く含むことが原因と考えられる．

4.4.2 RQ2 トークン数はどの程度増加したか

ReAct-Dac は，コード生成とツール実行を反復するため，一回完結方式と比較してトークン数が増加した．実験では，4 タスク平均のトークン数倍率が実行環境 E1 で 23 倍，実行環境 E2 で 16 倍となった．これは，ReAct-Dac が生成した画像を Base64 でエンコードし，プロンプトに含めていることが主因と考えられる．したがって，要求充足率の改善と運用負担（コスト）の増加との間にはトレードオフが存在する．

4.4.3 ReAct-Dac に関する考察

ReAct-Dac は，Dac の利点である記法に基づく解釈の安定性とテキストとしての管理容易性を維持したまま，「要求どおりのモデルが得られない」問題を緩和できる．トークン数は増加するものの，RFP，要件定義書，設計書などの成果物において，描画可能で記法に沿った図を安定的に組み込めることは有用である．結果として，アーキテクトの負担を減らし，議論や文書作成の速度向上に資する可能性がある．

4.5 妥当性への脅威

本実験では，要求充足率をチェックリスト充足率として評価したため，チェックリストの粒度や解釈の違いにより結果が変動する可能性がある．したがって，採点基準と手順を明確化し，再現性を担保する必要がある．さらに，タスク数および試行回数が限られているため，結果が十分に収束していない可能性がある．今後はタスクの多様化と試行回数の増加により，結論の頑健性を高める必要がある．

5. おわりに

5.1 研究成果

本研究は，生成 AI に Diagrams as code のコード生成を依頼した際に，要求どおりの図が得られないという課題に対し，推論によるコード生成と Dac ツール実行を反復し，描画結果を根拠に生成コードを改善する手法 ReAct-Dac を提案した．PlantUML を対象に，ベースラインである一回完結方式と比較評価した結果，gpt-4.1 および gpt-5.2 の両環境で要求充足率が改善し，描画エラーも減少した．一方で，手順の追加に伴いトークン数は増加し，有効性の向上と運用負担の間にトレードオフが存在することを確認した．

5.2 今後の展望

今後は，トークン数を抑えるため，停止条件や修正方針の最適化に取り組む．併せて，図種および要求種別の拡張やタスク数の増加により，ReAct-Dac の適用条件と一般性を検証する．これらを通じて，より広いモデリング作業において，要求どおりの図を安定的に得るための支援へ発展させる．

参考文献

[1] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, Yuan Cao, ReAct: Synergizing Reasoning and Acting in Language Models, International Conference on Learning Representations (ICLR), -, -, 2023.