

LLM を活用した探索的テスト効率化の提案

小楠 聡美
株式会社 HBA
ogusuxs@hba.co.jp

佐々木 健人
株式会社 HBA
sasakixkn2@hba.co.jp

要旨

筆者らは、ソフトウェアのシステムテストに LLM を活用したいと考えていた。

また、ここ近年要員不足が慢性化しており、業務効率化の必要性が高まっている。そこで今回、効率よく業務するための LLM の活用手段として、探索的テストに活用することを考えた。

本件では、探索的テストにおける LLM の効率的な活用手段と、その実践例、およびその効果を報告する。

1. はじめに

筆者らの業務環境では、システムテストを請け負うことが多い。システムテストは主にブラックボックステストの実行フェーズとなるため、その業務割合が多くなるほど、人的リソースも必要となる業務である。

一方で、コロナ禍明けの影響もあり、ソフトウェア開発現場でも要員が流動的になり要員不足が慢性的に発生している [1]。

筆者らは要員不足の要因とそれによって起こりうる結果が、図 1 に示したような関係になっていると考えた。

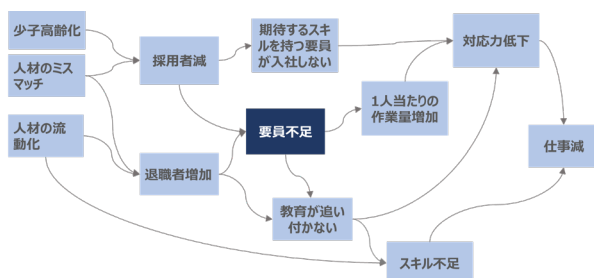


図 1 要員不足の要因とそれによって起こりうる結果

要員不足が発生すると、一人当たりの作業量が増えた

り、依頼された業務をできなくなったりするなど、業務への対応力が低下するだけでなく、教育に割く工数も十分に取ることができなくなりうる。その結果、既存顧客への要求に対応できなくなったり、新規顧客案件が獲得できなくなったりするなど、仕事減へとつながる。要員が不足しているからと言って、顧客から求められる品質の水準が下がるわけではないからである。しかし、要員不足は IT 業界全体的な傾向であり、すぐに解決することは難しい。

そのため、現状の要員が不足している状況でも対応力を向上できる施策がないか検討した。

2. 対応力向上施策の検討

要員不足が起こっている状態で、要員数を増やすことなく対応力の向上を実現するには、人的な作業負担を減らす必要がある。このテスト工程における人的負担を減らすために LLM を活用できないかと筆者らは考えた。

ブラックボックステストは主に、網羅型のテストとピンポイント型のテストに分類できる。網羅型のテストは記述式テストのアプローチにより、仕様に対して網羅性の高いテストを実施できるが、この網羅型のテストケース作成などに LLM を活用しようとしても正確なテストケースの出力は信頼性の面で難しい。数多く出力されたテストケースがすべてそのまま使用できることはなく、結局そのテストケースが間違えていないか、テスト仕様書のレビューに時間を割く必要が出てくる。そこで筆者らは、ピンポイント型である探索的テストへの LLM の活用に着目した。

探索的テストとは「テスト担当者がテストアイテムや以前のテストの結果の知識や調査情報を使用して、テストを動的に設計、および実行する」ピンポイント型のテスト手法である。網羅型の記述的テストのように事前にテストケースを作成するのではなく、テスト実行者の知見・経験に基づきながら潜在的な不具合を検知するため、熟練者向きのテストと言われている。

要員不足により熟練者が不足している現場で LLM が

熟練者の代わりとして機能することができれば、効率的にソフトウェアの故障を見つけ、工数削減につながると考えた。また、探索的テストではテストケースを作成せずに、故障が発生しそうな箇所を狙えばよいと、網羅型のテストのように作成したテストケースのレビューに時間を割くこともない。このような点においても、探索的テストに LLM を活用するほうが効果的と考え、探索的テストの手順、または観点の導出に LLM を使用して初心者でも簡単にシステムの故障を検出できる手段を検討した。

探索的テストを活用することで、図 2 のような効果が生まれることを期待した。

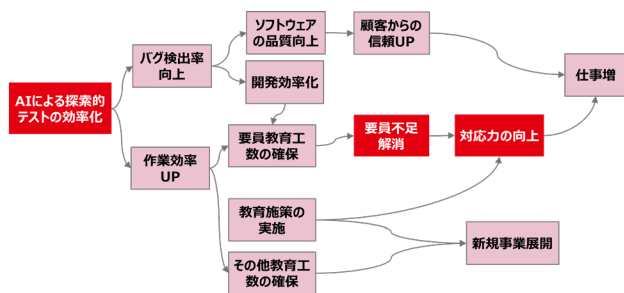


図 2. 探索的テストへの LLM 適用で期待される効果

3. 解決手段と期待する効果

3.1. 解決手段と期待する効果

LLM (RAG) にテストベースを読み込ませて、仕様漏れ、および非機能観点の欠陥の可能性のある箇所を洗い出し、そこをもとに探索的テストを行うことで効果的にシステムの欠陥や故障を検出できるのではないかと考えた。具体的には、表 1 のようなテストベースを読み込ませて一定の質問をすることで効果的に探索的テストの観点やテストのフォーカルポイントを見つけることができるのではないかと仮定した。

表 1. テストベースとそこから見つかるテスト観点例

No	取り込むテストベース	検出可能なテストの観点	今回検証
1	要求定義書、仕様書	仕様の欠陥、仕様漏れ	○
2	要求定義書、仕様書、ソースコード	仕様漏れ、実装漏れ	○
3	過去の故障・欠陥情報、ソースコード	類似欠陥 (ソースコードの欠陥)	
4	過去の故障・欠陥情報、仕様書	類似欠陥 (仕様の欠陥)	
5	テストチャーター、仕様書	仕様の欠陥、仕様漏れ	
6	リスク、仕様	仕様の欠陥、仕様漏れ	

3.2. 検証方法

今回の検証では、LLM を使用して探索的テストを行い期待通りの結果を得られるかの検証を行う場合と、LLM を使用してテストのフォーカルポイントを探すと同じ観点で人間がレビューした場合で、違いがあるかどうかを検証した。

3.3. 今回使用するテストベース

解決手段の効果を確認するために、いくつかのテストベースの組み合わせを用意した。

1. 仕様書とソースコード
2. 要求定義書と仕様書とソースコード

3.4. 今回使用する LLM のシステム

本検証を行うにあたり、社内で使用している CHAT GEAR というシステムを使用した。これは、ChatGPT4.0 をベースとした LLM をローカル環境に構築したものであるが、RAG の機能を持つため、このシステムに仕様書等の外部ファイルを読み込み、これらの情報を検索して生成 AI で回答を生成することができる。そのため、今回の検証のように、テストベースを読み込ませて仕様漏れを見つけるのに RAG 機能を持つこのシステムが最適であった。もう1つの大きな理由として、社内業務情報や顧客の業務環境で LLM を使うにあたり、セキュリティの都合で使用が制限されるケースがある。このような場合、LLM をローカル環境に構築した CHAT GEAR であれば学習内容が外部に漏れる可能性はないため、業務情報の仕様書といった機密情報を使いたい今回のようなケースにあっているのと、今後様々な環境で利用しやすいと思ったためである。

3.5. CHAT GEAR による検証手順

今回の CHAT GEAR を使用した検証の手順は以下のとおりである。

<CHAT GEAR による検証手順>

- ① CHAT GEAR に、以下の情報を読み込ませる。
 - ・要求定義書
 - ・仕様書
 - ・ソースコード
- ② CHAT GEAR で探索的テストを実行するフォーカルポイントを探す。
- ③ 実際に該当ポイントを探索的テストする。

CHAT GEAR では、質問の仕方によって返ってくる答えが変わることがよくあり、そこが難しい。この質問の仕方にスキルが必要ということになるが、今回の施策の目的は「要員不足の状況でも対応力を向上させたい」ということである。つまり、熟練者がいない、あるいは不足している状況でもこの手法を取り入れることで探索的テストを効率的に行いたい。初心者しかいなくてもこの手法を使って探索的テストで故障を見つけないのである。そのためには、なるべく手順や質問の仕方を一定にさせて誰でも同じ結果を出せるようになる必要がある。そこで今回は、この質問の仕方も一定の方法で、よい回答が導き出せる手順についても確立した。

3.6. 人間による検証手順

弊社の社員 3 名 (以下メンバーと呼ぶ) に対し、実際に該当テストベースの機能一覧や画面遷移図を仕様理解のために読み込んだ後、一部の仕様書を仕様漏れの観点で読み込んで、実際にシステムの該当機能のテストをしてもらった。その手順は以下のとおりである。

<人間による検証手順>

- ① 関連テストベースを以下の目的で読み込む。
 - ・仕様全体理解
 - ・仕様漏れの検出
- ② 実際に該当機能を探索的テストする。

このようにして、CHAT GEAR を使用して行った探索的テストと人間が行った探索的テストの結果の違いを検証した。

仕様漏れの観点から探索的テストを実行し故障を発見するには、十分に仕様を理解しているか、仕様書を読み込んで仕様漏れを探す必要がある。今回は筆者らも含めて全員軽くシステムの概要を知る程度、あるいは初めてみるレベルの知識しか持たないメンバーに協力してもらった。仕様やコンテキストを熟知していないメンバーが CHAT GEAR を活用して故障を発見できた場合、この手法はテスト初心者しかいない現場でも熟練者向けの探索的テストで成果を出すのに効果があると言えると考えたためである。比較のため、人間による検証についても同知識程度のメンバーに依頼した。3 名の協力メンバーの仕様書読み込みにかかった時間を計測して平均し、全仕様書読み込みにかかる時間を計測した。

4. 結果

今回の施策試行の結果は以下の通りであった。

4.1. CHAT GEAR による検証結果

CHAT GEAR から仕様漏れやソースコード漏れを探し、実際にその機能まわりをテストしてみたところ、15 分の CHAT GEAR とのやり取りと 5 分のテスト実行で 1 件の故障を見つけ、30 分で 2 件の故障を見つけた。

4.2. 人間による検証結果

弊社 3 名に、手順に従って該当機能を 15 分間テストしてもらったが、仕様漏れおよびシステムの故障を検出することはできなかった。

4.3. 仕様読み込みにかかる時間の検証

各協力者による仕様書の読み込み時間の平均から割り出した、該当仕様書の読み込みにかかる時間は以下の通りの結果となった。

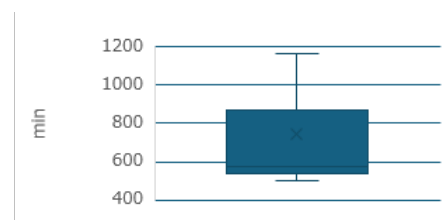


図 3 仕様書読み込みにかかる時間

該当システム全体の仕様書の読み込みを時間に換算すると、平均で 987 分(2.2 日)の時間がかかり、ソースコードまで読み込んだとすると 62,945 分(139.8 日)の時間がかかる試算となった。一方、LLM を使用して探索的テストを行った場合、LLM に質問してから探索的テストをして故障発見まで 15 分 1 件(CHAT GEAR でフォーカルポイントを見つけるのにかかった時間は 10 分に 1 件)であり、表 2 の結果のようになった。

表 2 検証にかかった時間と検出数

	仕様漏れ検出(レビュー)時間(分)	仕様漏れ検出件数(件)	テスト実行時間(分)	故障検出数(件)
CHAT GEAR	20	0	10	2
人間	987	0	15	0

5. 考察

今回の結果をまとめると以下のとおりである。

<結果>

- ① LLM を活用して、仕様漏れの観点からの故障を効率的に検出できた。
- ② 仕様を把握していなくても、仕様漏れの観点からの故障を検出できた。

①について、CHAT GEAR による検証手順に沿って探索的テストを行うことにより、レビュー時間とテスト時間を合わせて 30 分で 2 件の故障を検出できた。また、メンバーが仕様書を読み込んでも故障を 1 件も検出できなかったことから、この手法はテスト初心者にも探索的テストで故障を検出するのに有効だと言える。冒頭でも述べた通り、LLM の結果は信用できない場合も多く、間違えた結果を示している可能性もあるが、探索的テストの性質上、やってみて故障が見つからなければ、次の観点を探せばよいだけであり、正確なレビューなども必要としない。したがって、LLM から網羅型のテストケースを作成するよりも探索的テストに利用するほうが、効果的な活用ができると考えられる。

今回の仕様漏れに関する故障の検出は、人間が行っても発見できると考えられる。同じ観点で仕様漏れを探し、

他のテストベースのレビューやソースコードをレビューすることで見つけることは可能だと考えられるが、そのためには豊富なコンテキスト知識、ソフトウェアの開発知識などが必要である。それが、探索的テストが熟練者向けと言われている理由である。そのため、プロジェクト背景をよく知らない業務経験の浅い人やテスト初心者には難しい。しかし今回の②の結果から、仕様書の詳細を把握できていない人でも仕様漏れに関する故障を発見できた。つまり、要員不足の状況や熟練者ではなくても簡単に探索的テストで故障を発見でき、要員不足の状況でも対応力を向上させることができると言える。

また、今回のシステムの仕様書類をすべて人間が読み込むと平均で 2.2 日かかる計算となった。仕様書を読み込むことなく故障を検出できたことから、この方法を実践すると少なくとも 2.2 日分の工数を、他の作業に充てるができるようになる。さらに、仕様漏れを見つけれずにやみくもにテスト実行を行うことに工数を費やす可能性もあるため、その分についても効率 UP につなげることができる。この時間を本来要員不足解消のために行いたいメンバーの教育や新規事業の展開検討の時間に充てる事が可能となる。

今回、誰でも同じ結果が出せるように質問の仕方についても工夫したことを上記で記載したが、何度も試行した結果、最も良い答えを得られたのは図 4 のフローに沿って行う場合であった。このフローに従って質問をすることで、同じような回答を得ることができたことから、初心者でも同じように探索的テストのフォーカルポイントは導出できると考えられる。

場合によってはソースコードが入手できない環境もあるが、その場合は図 4-③のチェックで NO だった場合は⑩に遷移すればよいだけである(図 5)。実装は行われている可能性もあるが、書いていない仕様への実装の可能性は仕様書に記載されている内容と比べると低く、やみくもにテストを行うよりも故障を検出できる確率は高いであろうと推測できる。また、要求定義書がある場合は、要求仕様書をもとに要求を満たしていない仕様があるか探し、ない場合はソースコードで実装がされているか探すといった上流から順にフローを流していくことで要求に沿った仕様、実装となっているか探すことができる。

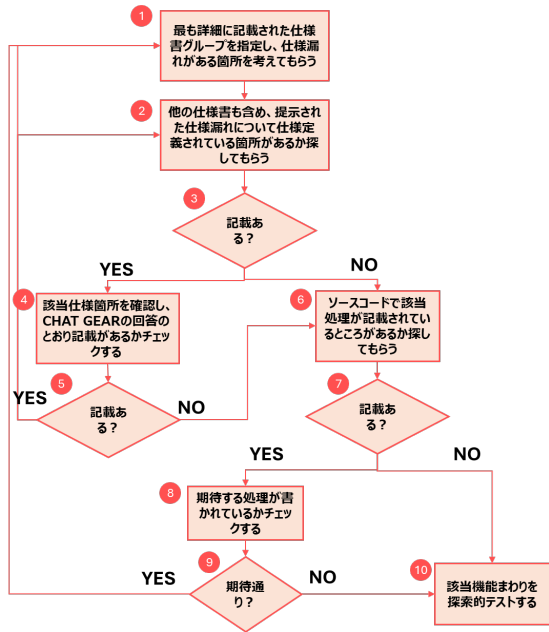


図 4 LMM 質問のフロー(ソースコードがある場合)

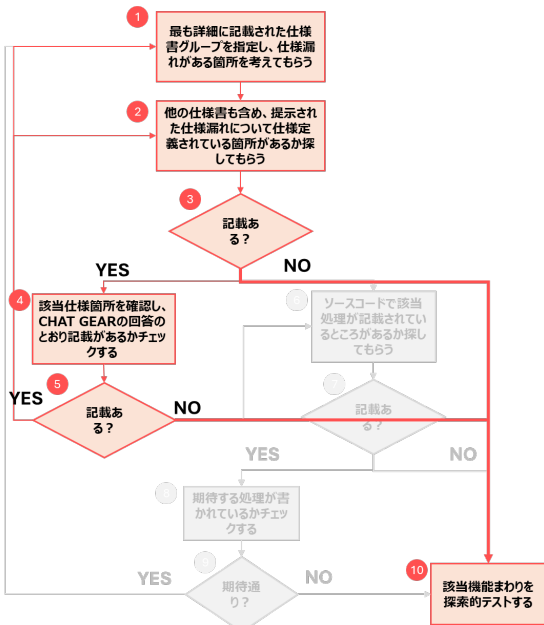


図 5 LMM 質問のフロー(ソースコードがない場合)

ただし、注意点もある。今回の手法を適用する大前提として、詳細情報が記載された仕様書が存在することで

ある。抽象的な情報の記載しかない仕様書を読み込ませても、よい結果を得ることはできない。また、図4の過程で各質問内容を抽象的にすると、回答もあいまいで返ってきたことから、重要なポイントは以下の通りであると考えられる。

- ・ 質問はなるべく具体的に記載する。
- ・ 代名詞などは使用しない。
- ・ 範囲を指定する場合には、フォルダ名やファイル名は省略しない。

また、今回検証で CHAT GEAR に質問をしてみても最もよい回答を得られた結果について考察してみたところ、以下のことが考えられる。

1. 全体的になるべく具体的な表現で質問をする。

なるべく代名詞などは使わず、具体的、正確なファイル名を指定して質問する。曖昧な表現や代名詞などを使用して質問すると抽象的な答えや一般論の回答が返ってくる。

no 1 | 5/22 11:20 | 使用ファイル表示 | delete | copy

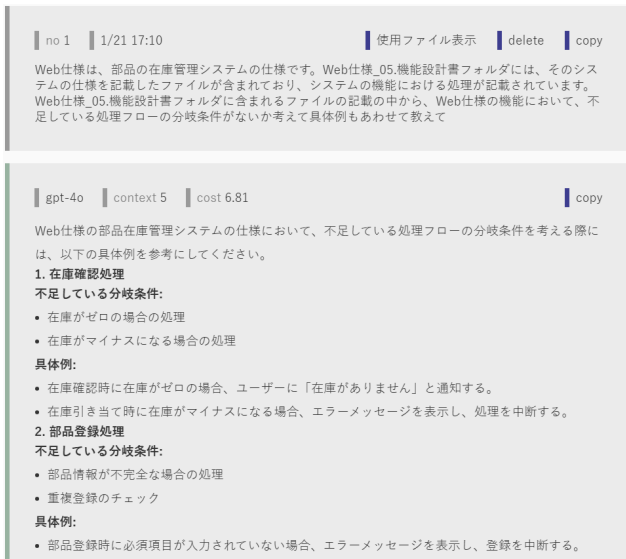
Web仕様の機能設計書の中から仕様の抜け漏れを探して

gpt-4o | context 5 | cost 5.769 | copy

仕様の抜け漏れを探すためには、提供された情報をもとに、各ファイルの内容を確認し、共通している要素や不足している要素を特定する必要があります。以下に、各ファイルの内容から考えられる仕様の抜け漏れを示します。

1. 共通要素の確認:
 - 各ファイルに共通して含まれている要素を確認します。例えば、`documentElement`や`namespaceURI`などの要素は複数のファイルに共通しています。これらの要素が仕様書に記載されているか確認します。
2. 特定の機能の確認:
 - `addEventListener`や`createComment`などの特定の機能が、仕様書に記載されているか確認します。これらはDOM操作に関連する重要な機能であり、仕様書に含まれているべきです。
3. データ操作関連の確認:
 - `getUint8`や`setInt16`などのデータ操作に関連する機能が、仕様書に記載されているか確認します。これらはバイナリデータの操作に重要です。
4. 正規表現やパターンマッチングの確認:
 - `RegExp`や`matchesSelector`などの正規表現やパターンマッチングに関連する機能が、仕様書に記載されているか確認します。これらはデータの検証やフィルタリングに使用されます。
5. エラーハンドリングの確認:

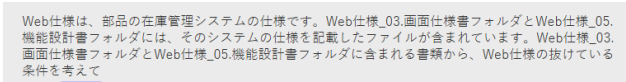
抽象的な質問をした結果例



具体的な質問をした結果例

2. 図 4-①の際には「調べて」「探して」「教えて」などの命令はせず、「考えて」と LLM 自身に考えさせるように指示する。

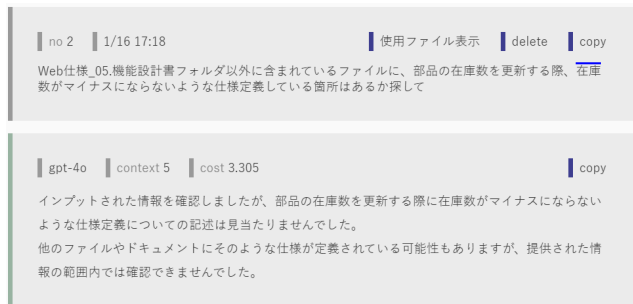
ここで探したいのは仕様漏れ、つまり、仕様書に記載されていないことを探したいため、LLM に結果を考えさせる必要があるためである。「探して」などで命令すると「ありませんでした」などの答えが返ってることが多い。



LLM に結果を考えさせるための質問例

3. 図 4-②と図 4-⑥では、「調べて」「探して」「教えて」などの命令でよい。

ここでは、他のテストベースに記載あるか、記載されている仕様の中から探したいだけであるためである。2 で列挙された結果の中で、最も故障が発生するとインパクトが大きそうなものをピックアップし、他のテストベースに記載があるかどうかを探してもらう。その回答で「ない」と判断された場合は、仕様漏れの可能性が高い個所と判断する。同様に、入手できるのであればソースコード内にも該当する実装があるか探す。出力結果だけでわかりにくい場合は元資料を確認する。図 4-②あるいは図 4-⑥で該当結果が見つかった場合は 2 に戻って別の結果をピックアップし直して探す。



他テストベースに記載があるかの確認例

以上を簡単にまとめると、図 5 のようになる。

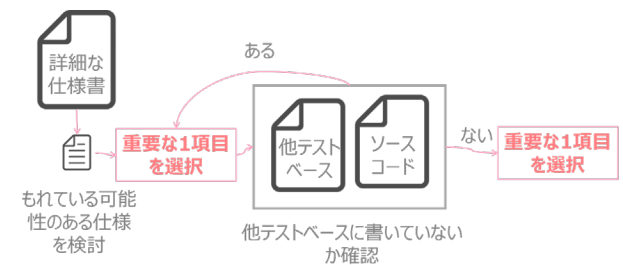


図 6 手順イメージ

最も具体的に書かれた仕様書を選び、その中で仕様漏れがない検討させる。そこで出てきた項目が他のテストベースに記載されていたり、仕様に書かれていないだけで実際に実装されていたりしないかを調べ、なければその仕様周りを探索的にテストする。こうすることで効率よく故障を検出できる。

今回用意できたテストベースの都合で仕様漏れの観点をメインに検証したが、表 1 に記載したような他のテストベースもしっかり揃っているシステムであれば、リスク観点の故障や仕様漏れ、仕様の欠陥を検出できたり、欠陥ベースのテスト技法にも活用できたりすると見込んでいる。リスクのテスト観点や過去の不具合のテスト観点についても同様に情報を読み込ませて、質問の仕方を工夫することでその観点のフォーカルポイントを抽出できると考えている。ただし、リスクは抽象的に表現されることも多いため、上述の重要ポイントを考慮すると、できるだけリスクをシステム寄りの表現に落とし込まないとよい結果が得られないのではないかと考えている。

6. まとめと今後の展望

今回の結果から、LLM を使用して仕様漏れを探し、探索的テストのインプットにするのは有効であることがわかった。ここから派生して、リスクベースドテストへの活用や欠陥ベースのテスト技法にも応用できる展望が見えた。

また、仕様書の読み込みを行わなくても短時間で仕様漏れ観点の故障を検出できたことから、この手法を用いて探索的テストを行うことで、熟練者ではなくても効果的に故障を検出できることがわかった。

以上から、今回提案した手法は要員不足の環境でも対応力向上に活用できると言える。

今回は用意できたテストベースが仕様書とソースコード類のみで、仕様漏れの観点からの活用しか検証できなかったため、今後は前述したリスクベースドテストや欠陥ベースのテスト技法への応用を検証したい。LLM を活用した探索的テストでカバーできる範囲を広げることで、人間による探索的テストと LLM を使った探索的テストの役割を分ける。LLM は性質上、読み込ませたテストベースに書いてある内容から答えを探すことを得意とするので、その範囲は LLM に任せ、人間はテストベースに書いていないような異常な操作をするなど役割分担しながら効率よく故障を検出できるようになる。

また、LLM の質問の仕方をより定型化し、より誰でも迷うことなく同じ結果を出せるようにしたいと考えている。

参考文献

- [1] 内閣府 令和6年度 年次経済財政報告
<https://www5.cao.go.jp/j-j/wp/wp-je24/h02-01.html>