ソフトウェアテストシンポジウム 2026
JaSST'26: Japan Symposium on Software Testing 2026

LLM のテスト分析・テスト設計能力を測定するためのベンチマーク手法

吉川 努
株式会社ベリサーブ

谷﨑 浩一
株式会社ベリサーブ

上野 彩子
株式会社ベリサーブ

増田 聡
東京都市大学

あらまし 本研究は，ソフトウェアテストにおけるテスト分析およびテスト設計に着目し，LLM の能力を測定するためのベンチマーク手法を提案する．従来の研究がユニットテストやコード生成に偏っていたのに対し，本手法はシステムテストレベルを対象とした．さらに，混同行列の考え方を適用し，テスト観点導出力（単一・組み合わせ）および生成されたテストケースにより検出可能な欠陥の範囲（欠陥識別力）を正解率・適合率・再現率・F 値により定量化するベンチマークを設計した．社員情報検索システムおよびテスト管理ツールを題材に評価した結果，LLM は単一のテスト観点導出では高い性能を示す一方，観点の組み合わせや具体的なテストケース生成では性能が低下することを確認した．本研究は，LLM のテスト分析・テスト設計能力を評価するための枠組みを示し，今後の評価手法に関する研究への適用や実務における LLM 活用を判断するための基盤を提供する．

キーワード ベンチマーク手法，大規模言語モデル，テスト分析能力，テスト設計能力，混同行列

Benchmarking Large Language Models on Test Analysis and Test Design

Abstract This study proposes a benchmark for evaluating LLMs in software test analysis and test design. Unlike prior studies that focus on unit testing and code generation, it targets system-level testing and introduces evaluation metrics based on the concept of a confusion matrix, quantifying test view derivation capability (single and combinational) and defect identification capability—i.e., the range of defects detectable by generated test cases—using accuracy, precision, recall, and F-score. Applied to an employee search system and a test management tool, the results showed that the LLMs performed well in deriving single test views, while its performance declined for combination of test views and concrete test case generation. This method provides a framework for assessing LLMs capabilities in test analysis and test design and serves as a foundation for future evaluation research and for guiding practical adoption of LLMs.

Keyword Benchmarking, Large Language Model;LLM, Test Analysis Capability, Test Design Capability, confusion matrix

1. はじめに

ソフトウェアテストにおいてテスト分析やテスト設計は高度な自然言語処理能力と創造性が必要なタスクであり，従来は人の経験や知識を基に人の手で実行されてきた．特にシステムテストと呼ばれるテストレベルでは，システム全体の機能や構造，入力条件や悪条件，ユーザがシステムを使用する際の環境やユーザの行動など，さまざまな要素を考慮してテストを作成する必要があり，人の経験や知識への依存が大きい．しかし近年，大規模言語モデル（Large Language Model．以下，LLM と呼ぶ）をベースとした生成 AI の登場により，人の手を介さずにシステムテストレベルのテスト分析やテスト設計を行うことが可能となりつつある．

LLM を特定のタスクに活用するためには，そのタスクの実行に当たっての LLM の能力を測る必要がある．そのためのさまざまなベンチマーク手法が提案されており，ソフトウェアテストの能力を測るベンチマークも存在する．既存のベンチマークは，LLM が生成したプログラムコードやテストコードを実行してテストを Pass するかどうかを計測することで，LLM のソフトウェアテストの能力を測定できる．

既存のベンチマークは主にユニットテストレベルのテストコード生成を対象としており，システムテストレベルのテスト分析・テスト設計の能力を直接評価するものではない．システムテストレベルの GUI テストや欠陥修正を扱う試みも一部存在するが，体系的な評価枠組みとしては十分に確立されていない．システムテストレベルでの純粋なテスト分析・テスト設計の能力を評価するベンチマークは提案されておらず，LLM がどの程度の能力を有しているかを測る方法がないのが現状である．

そこで本研究では，LLM のシステムテストレベルの

テスト分析・テスト設計能力を測定可能にすることを目的とし，テスト分析・テスト設計のベンチマーク手法を提案する．その提案手法を LLM によるテスト分析・テスト設計の成果物に適用し，LLM の能力がどの程度のレベルにあるかを確認する．

本研究の貢献は以下の 2 点である．

①システムテストレベルのテスト分析・設計能力を測定するための新たなベンチマーク手法を提案したこと．

②提案手法を実際に適用した事例を通じて，現状の LLM の能力と限界を明らかにしたこと．

以降，2 章で関連研究を述べ，3 章で提案するベンチマーク手法について解説する．4 章でベンチマーク手法の適用事例と結果を示す．5 章で考察を行い，6 章で結論を述べる．

2. 関連研究

LLM の一般的な言語処理能力や情報処理能力を測るベンチマークとして，自然な文章を生成する能力や論理的関係性の判断能力を測るベンチマーク[1]，一般常識に対する回答の正確性を測るベンチマーク[2]，数学問題や算数問題の解決能力を測るベンチマーク[3][4]が提案されている．これらのベンチマークでは，言語処理や情報処理のタスクとそのタスクにおける正解のセットを用意し，LLM の出力と正解との一致度で LLM の能力を測定する．新たなベンチマーク手法を検討する上で，正解セットとの一致度を測る際に活用できると考えられる．

LLM による特定のタスクの実行能力を測るベンチマークも提案されており，ソフトウェアのコード生成の能力を測るベンチマークとして HumanEval[5]，MBPP[6]が知られる．両者ともコードを生成するタスクと対応する単体テストコードが用意されており，LLM が出力したコードを用いて単体テストを自動実行し，結果が Pass となれば出力が正しいと判定される．これらを日本語向けにしたベンチマークも公開されている[7][8]．

より実践的なソフトウェア開発能力を測るベンチマークとして SWE-Bench[9]や DevBench[10]が提案されている．SWE-Bench は欠陥の修正や機能実装などのソフトウェア開発の過程で発生するさまざまなタスクを含み，ソフトウェア開発に対する LLM の能力を評価する方法を提供している．DevBench は設計，環境構築，実装，受入テスト生成，ユニットテスト生成といったソフトウェア開発ライフサイクル全体で発生するタスクを対象とし，LLM の能力を評価する方法を提供している．これらのベンチマークでは，LLM へのインプットとして仕様や課題を用意し，その仕様や課題に対してコード生成などのソフトウェア開発におけるタスクを LLM に実行させ，出力の正否を判定して LLM の能力を測定する．出力の正否の判定には，事前に用意されたテストコードの自動実行によるテスト結果の評価，有識者による成果物の内容の評価，LLM に成果物を評価させる LLM-as-a-Judge といったさまざまな方法が用いられる．テスト分析・テスト設計の能力を測定するベンチマークを検討するに当たっては，成果物に含まれるテストの内容の充実度を測定する必要があるため，テストコードの自動実行による評価は難しい．一方で，有識者による評価や LLM-as-a-Judge は活用可能であると考えられる．

テストケース作成の能力についてもさまざまなベンチマークが提案されている．TESTEVAL[11]は Python プログラムを対象に，単体テストコードによるプログラムコードの実行パスのカバレッジを評価する．TestGenEval[12]，UnLeakedTestbench[13]も同様に Python プログラムに対する単体テストのテストコード生成を対象とし，カバレッジに加えてミューテーションスコアによる欠陥検出力も評価する．TestBench[14]は Java プログラムを対象に，クラスレベルに対する単体テストケースの生成能力を，構文・コンパイルの正しさ，カバレッジ率，欠陥検出率などの観点で評価する．SWT-Bench[15]は SWE-Bench をテスト生成のベンチマークに拡張したもので，GitHub リポジトリ上に登録された実際の欠陥データを用い，LLM が生成したテストコードが欠陥修正前のプログラムコードに対して Fail し，修正後のプログラムコードに対して Pass するかを確認することによって，テストコードの欠陥再現能力を評価する．対象となるテストレベルは欠陥の種類に依存し，単体テストから結合テストレベルが中心である．

以上のように，LLM の能力を測定するためのベンチマークは数多く提案されているが，既存のベンチマークは主に単体テストのコード生成の能力に焦点を当てており，システムテストレベルでのテスト分析・テスト設計の能力を直接評価する枠組みは十分に検討されていない．DevBench は受入テスト生成を含むが，既存リポジトリに含まれるテストケースに Pass するかを基準にコード生成の能力を評価する枠組みであり，純粋なテスト分析・テスト設計の能力を測定するものではない．本研究はこの点を補うものである．

3. ベンチマーク手法の設計

3.1. 対象とするテストプロセス

本研究では，知識・思考プロセスに焦点を当てるため，テスト分析とテスト設計を対象とした．これらはテスト活動の品質を大きく左右する上流工程であり，

近年の LLM が人間に匹敵する水準の知的成果を示す領域であるため対象として適切であると考えた.一方，テスト実装プロセスは，与えられたテストケースを基に形式的に手順を定める工程であり，知識や思考の深さが相対的に求められにくいため，対象からは除外した.

3.2. 評価する LLM の能力と評価方法

本研究では，LLM が生成したテストケースを起点として，以下の三つの能力を評価する対象とする.

(1)テスト観点導出力（単一）

テストベースから「何をテストするか」という観点を識別・分解できているかを評価する．これはテスト分析プロセスに対応する．単一のテスト観点は，仕様に記載された機能や条件を意味的に分類した結果として得られるものであり，LLM の意味理解能力が強く反映される.

(2)テスト観点導出力（組み合わせ）

複数のテスト観点を組み合わせて構造化できているかを評価する．これはテスト設計の初期段階に相当し，単なる列挙ではなく，テスト観点間の関係性や直交性を考慮した構成力が求められる.

(3)欠陥識別力

生成されたテストケースによって，どの程度の欠陥を検出し得るかを評価する．これはテスト設計の最終段階に相当する．ここでいう欠陥識別力とは，実際にテストを実行した結果ではなく,「そのテストケースを実施した場合に検出可能な欠陥の範囲」を指す．これはテストケース生成の妥当性・有効性を表す指標となる.

これら三つの能力は，人が作成したテストケースを基に構築したテスト観点および検出可能な欠陥の正解セットと，同一条件下で LLM が作成したテストケースから得られた結果とを比較して評価する.

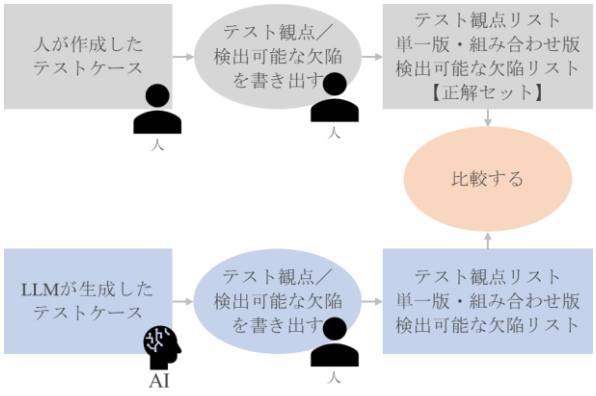

図 1 比較イメージ

3.3. 正解セットの構築

評価に用いる正解セットは，人が作成したテストケース群を基に構築する．具体的には，人のテストケースから以下を抽出する.

・単一のテスト観点リスト
・組み合わせテスト観点リスト
・テストケースによって検出可能な欠陥のリスト

正解セットはテスト対象に対して作成し，重複する欠陥や，テストベースから導出不可能なものは除外する.

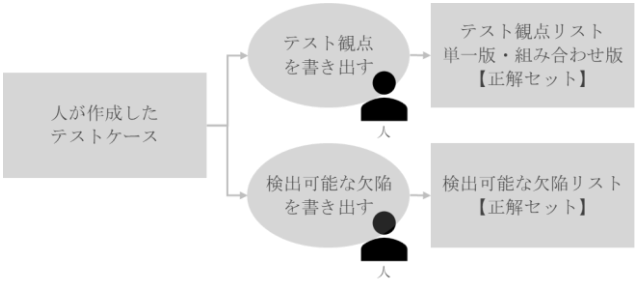

図 2 正解セット作成イメージ

3.4. 混同行列による評価方法

LLM の出力と正解セットとの関係は，二値分類問題として整理し，混同行列に基づいて評価する．本研究では，テスト観点および検出可能な欠陥に対して「有効か無効か」を基準として分類を行う.

	実際は正(有効)	実際は負(無効)
正(有効)と予測	TP(True Positive) 真陽性	FP(False Positive) 偽陽性
負(無効)と予測	FN(False Negative) 偽陰性	TN(True Negative) 真陰性

一般的な混同行列では，正解セットに含まれるものを正（Positive），含まれないものを負（Negative）として扱う.しかし，テスト分析・テスト設計においては，正解セットに含まれていないからといって，その出力が直ちに無効であるとは限らない．人が作成した正解セットには，含まれていなくても，実務上有効と判断できるテスト観点や欠陥が存在し得るためである.

そこで本研究では，LLM が出力した結果について，人手によるレビューを行い，正解セットに含まれない場合であっても，有効と判断されたものは True Positive として扱う．これを図示すると，正解セットと LLM の出力が重なる領域（TP2）に加え，LLM のみが導出したが有効と判断された領域（TP1）が存在する構造となる．一方，LLM が導出したものの，重複や誤りなどにより無効と判断されたものは False Positive とする.

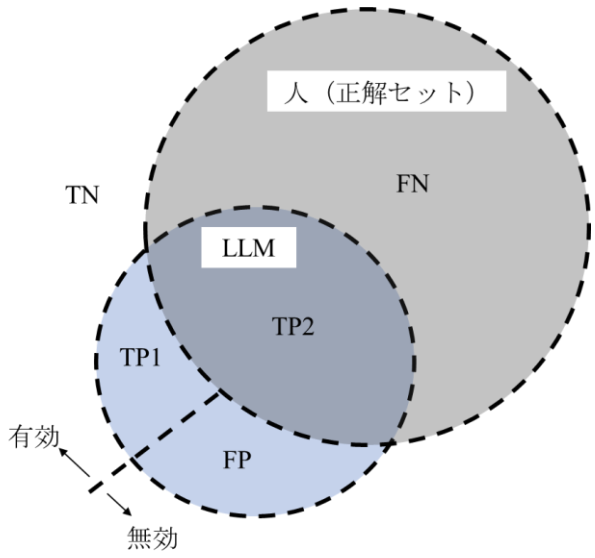

図 3 出力判定のイメージ

以上より，本研究における混同行列の各要素は以下のように定義される．

・TP（True Positive）：LLM が導出し，有効と判断されたテスト観点／欠陥（正解セットに含まれるもの(TP2)，および含まれないが有効と判断されたもの(TP1)を含む）

・FP（False Positive）：LLM が導出したが，無効と判断されたもの（重複，誤り，実務上意味を持たないものなど）

・FN（False Negative）：正解セットに含まれるが，LLM が導出できなかったもの

・TN（True Negative）：正解セットにも LLM の出力にも含まれず，かつ無効と判断されるもの

この定義に基づき，正解率，適合率，再現率，F 値を算出することで，正解セットとの単純な一致度だけでなく，LLM が導出したテストケースに含まれる有効なテスト観点や欠陥も含めて評価することが可能となる．

3.5. 評価手順

評価は以下の手順で実施する．

1.テストベースおよびプロンプトを LLM に入力し，テストケースを生成させる．

2.生成されたテストケースから，テスト観点および検出可能な欠陥を人手で抽出する．

3.抽出結果を正解セットと比較し，TP，FP，FN，TN を判定する．

4.各指標（正解率・適合率・再現率・F 値）を算出する．

この評価を複数の LLM モデル・バージョンに対して適用することで，モデルの進化による能力差や特性の違いを可視化する．

各指標の意味は以下の通りとなる．

正解率：「有効」「無効」含めてどのくらい出せたか

適合率：「有効」と判定したものがどのくらい信用できるか

再現率：「有効」をどのくらい出せたか

F 値：適合率と再現率のバランス

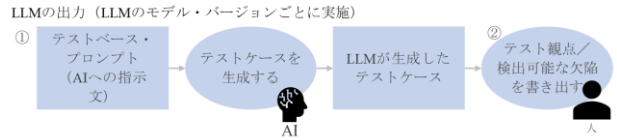

図 4 LLM の出力方法

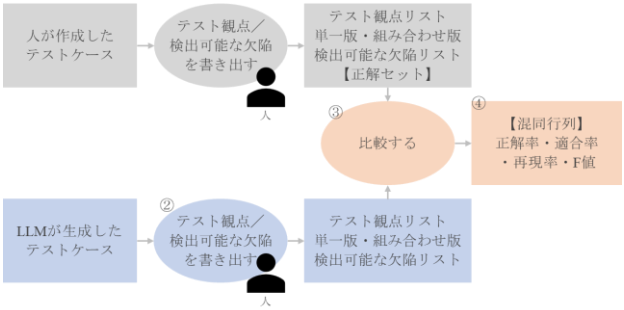

図 5 指標の算出

4. 適用事例

本章では提案したベンチマーク手法を実際に LLM に対して適用した事例と結果について示す．適用したシステムは，(1)テスト管理ツール，および(2)社員情報検索システムである．この結果を通じて，提案手法が LLM のテスト分析・設計においてどのような特性を持つのか，人との差異を可視化できるかを考察する．

4.1. 適用したシステムと評価条件

4.1.1. テスト管理ツール

テスト管理ツールは，テストケースの作成・設定を行い，テスト実行の管理を行う Web アプリケーションである．本研究では，操作マニュアルをテストベースとし，「テストスイートを作成する機能」および「テストスイート設定を変更・削除する機能」を対象とした．人が作成したテストケースは 50 件であり，これらを基に，単一のテスト観点 76 件，組み合わせのテスト観点 57 件，テストケースによって検出可能な欠陥 69 件を抽出し，正解セットを構築した．

4.1.2. 社員情報検索システム

社員情報検索システムは，社員情報を管理し，条件指定により検索を行う Web アプリケーションである．本研究では，設計書をテストベースとし，「キーワード検索機能」を対象とした．人が作成したテストケースは 56 件であり，そこから単一のテスト観点 51 件，組み合わせのテスト観点 52 件，検出可能な欠陥 39 件を

抽出して正解セットを構築した.

4.2. 対象にした LLM のモデルと評価手順

適用した LLM のモデルは以下の通りとした.

・GPT-5
・GPT-5.1
・GPT-5.1 Thinking
・GPT-5.1 Pro

GPT-5 については社内向けのアプリケーションを使い，それ以外のモデルのアプリケーションについては ChatGPT を使用した.

それぞれのモデルに対して「3.5.評価手順」で述べた手順に基づいて実施した.

この評価を，複数の LLM モデル・バージョンに対して実施することで，モデル間の特性差も併せて分析した.

4.3. プロンプトの内容

LLM に与えるプロンプトの内容はそれぞれ以下の通りとなる.

```
Prompt（テスト管理ツール）：
あなたはソフトウェア開発プロジェクトにおいて、システムテストを担当している優秀なテストエンジニアです。

このプロジェクトでは、ソフトウェアテストにおけるテストスイートを管理するテスト管理ツールを開発しています。

添付ファイルの情報からシステムテストのテストケースを表形式で作成してください。
```

図 6 テスト管理ツールにおけるプロンプト

```
Prompt（社員情報検索システム）：

あなたはソフトウェア開発プロジェクトにおいて、システムテストを担当している優秀なテストエンジニアです。

このプロジェクトでは、社員データベースを開発しています。
以下の情報からシステムテストのテストケースを表形式で作成してください。

# 仕様
（省略。ここにテキストベースで仕様を記載する）
```

図 7 社員情報検索システムにおけるプロンプト

プロンプトは，LLM の能力結果を，プロンプト設計の工夫に依存させないようにするため，必要最小限の記述とした.

4.4. 適用結果

二つのシステムに共通して，以下の傾向が確認された.

まず，全体の傾向として，適合率について 83~100%となり LLM の各モデルはいずれも正解率や再現率と比べて高い数値を示した．一方で，正解率や再現率は 23~64%と適合率に比べて低く，必要なテスト観点や検出可能な欠陥を十分に網羅できていないことが確認された.

これらの傾向は，テスト管理ツールと社員情報検索システムという異なる題材においても概ね共通しており，提案したベンチマーク手法が特定のシステムに依存しない形で，LLM のテスト分析・テスト設計能力の特徴を可視化できることを示している.

また，正解率や再現率については，単一のテスト観点導出力が，テスト管理ツールで 38~53%，社員情報検索システムで 51~62%であったのに対して，組み合わせのテスト観点導出力や欠陥識別力がテスト管理ツールで 25~37%，社員情報検索システムで 31~47%と低い数値を示した.

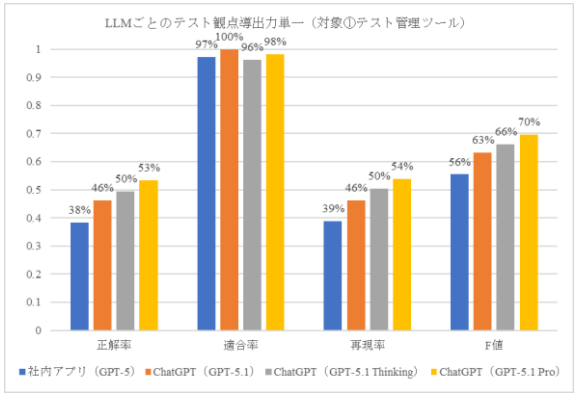

図 8 テスト管理ツール テスト観点導出力単一

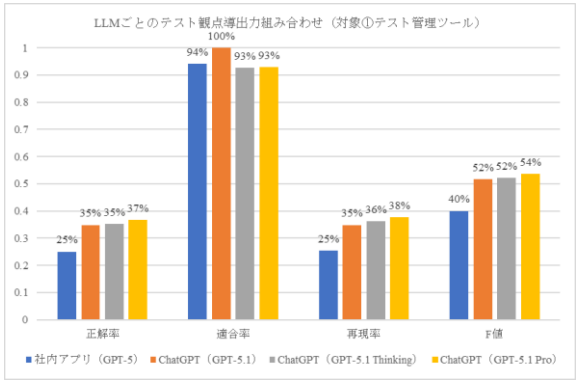

図 9 テスト管理ツール テスト観点導出力組み合わせ

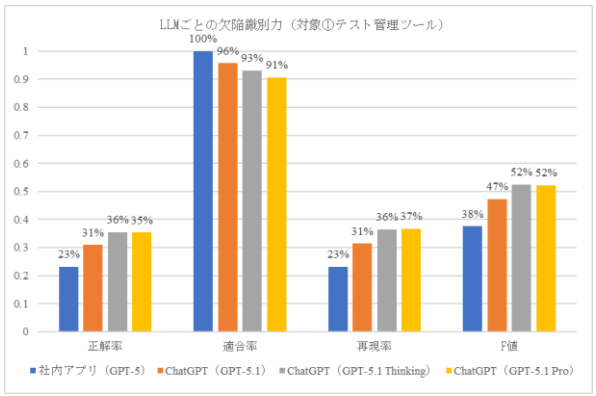

図 10 テスト管理ツール 欠陥識別力

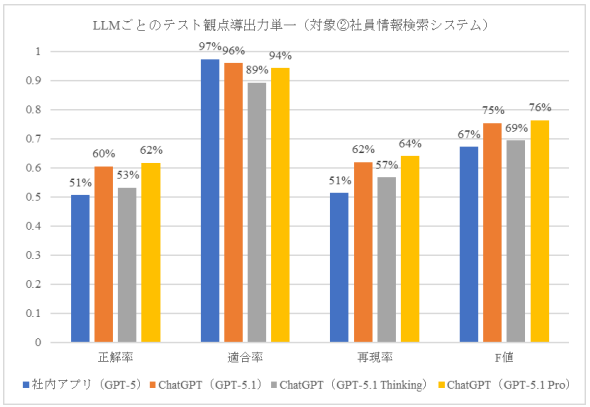

図 11 社員情報検索システム テスト観点導出力単一

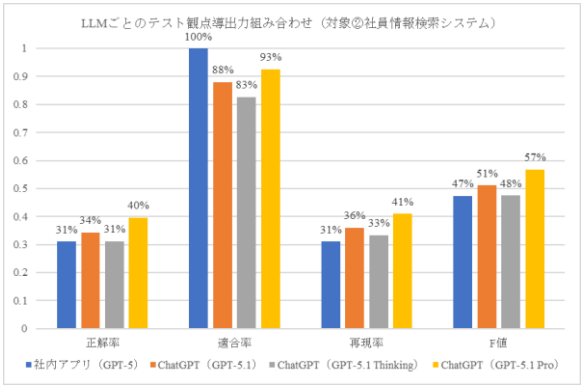

図 12 社員情報検索システム テスト観点導出力組み合わせ

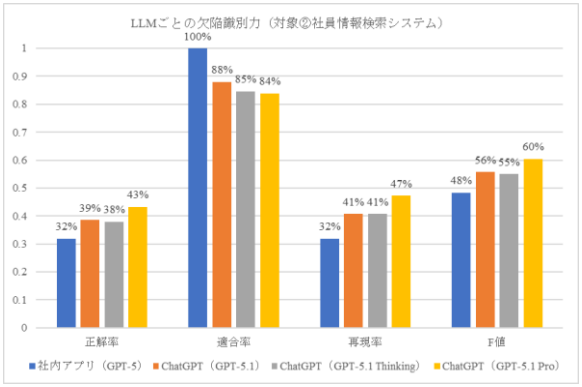

図 13 社員情報検索システム 欠陥識別力

5. 考察

5.1. LLM のテスト分析・テスト設計能力の特性

人がテスト分析やテスト設計を行う場合には，テストベースの記載を基にテスト観点を列挙・整理する．一方で，過去に遭遇した欠陥の経験などからどのような欠陥が発生し得るかを考えながら行う．それに対して，LLM は，適合率が高く，正解率・再現率が低いことからも代表的で妥当と思われるテスト観点を慎重に選択する傾向があると言える．

また，単一のテスト観点導出力（38~62%）が，組み合わせのテスト観点導出力や欠陥識別力（25~47%）

に比べて高い傾向にあることから，複数のテスト観点を組み合わせて構造化する工程やテストケースを生成する工程よりもテストベースの情報からテスト観点を意味的に整理・列挙することに優れていると考えられる．さらに単一のテスト観点導出力において，高い場合でも 60%台にとどまっており，人のテスト分析の能力に達しているとは言い難い．

以上より，現時点で今回適用したシンプルなプロンプトでは，LLM は完全に人を代替する存在ではなく，テスト分析・設計における思考のたたき台や補助として位置付けるのが妥当であると考えられる．

5.2. 本研究における課題と評価手法の妥当性

本研究では，LLM のテスト分析・テスト設計能力を評価するためのベンチマーク手法を提案し，二つの異なるシステム（テスト管理ツールおよび社員情報検索システム）に適用した．その結果，両システムにおいておおむね同様の傾向が確認されており，本研究で採用したベンチマーク手法は，特定の題材に依存せず，LLM の特性を一定程度安定して可視化できる可能性が示唆された．この点において，本評価手法自体の有効性は確認できたと考えられる．

一方で，本研究には幾つかの課題も存在する．第一に，対象としたシステムの範囲である．本研究では，いずれも Web アプリケーションの一部の機能を対象としており，LLM が知識を有していると想定されるため，比較的良好な分析結果が得られた可能性がある．しかし，より専門的かつドメイン依存性の高い領域，例えば金融，医療，組込みシステムなどに適用した場合，同様の傾向が維持されるかは不明である．今後は多様な領域や特性を持つシステムを取り上げて検証する必要がある．

第二に，評価回数および出力のばらつきに関する課題である．本研究では，各 LLM モデルについて原則 1 回の生成結果で評価しているが，LLM は生成のたびに異なる出力を示す特性を持つ．そのため，複数回の生成結果を用いた平均的な評価や分散の分析を行うことで，より安定した能力評価が可能になると考えられる．

第三に，能力指標の拡張可能性である．本研究では，混同行列に基づく正解率・適合率・再現率・F 値といった定量的指標を中心に評価を行った．しかし，実務上はこれに加え，テストケースの理解容易性，保守性といった定性的側面も重要である．今後はこれらを包括的に扱う評価フレームワークを構築することで，より現実に即した評価が可能になると考えられる．

最後に，LLM の進化速度に対応する課題である．モデルの性能は短期間で大幅に改善されるため，本研究で得られた結論はあくまで現時点でのスナップショットに過ぎない．将来的には，LLM の進化に合わせてベンチマーク手法を改良・更新し，継続的に評価できるように効率良く評価できる仕組みを構築していく必要がある．

これらの課題を踏まえ，今後はより多様なシステムや比較指標を取り入れた検証を進め，ベンチマーク手法の一般化を図る必要がある．

6. 結論

本研究は，ソフトウェアテストにおけるテスト分析およびテスト設計に着目し，LLM の能力を計測可能とするベンチマーク手法を提案し，その適用事例を通じて有効性を示した．

提案手法を適用した結果，LLM は単一のテスト観点導出において，テスト観点の組み合わせの導出や欠陥識別力と比較すれば相対的に良好な結果を示した．しかし，その性能は正解率や再現率が高くても 60%台程度にとどまっており，単一のテスト観点導出においても，人のテスト分析・設計能力に達しているとは言い難いことが明らかとなった．

一方で，LLM が生成したテスト観点やテストケースは適合率が高く，有効性の観点では一定の信頼性を有していることも確認された．このことは，LLM が無作為に多くの候補を列挙するのではなく，慎重に「当てにいく」振る舞いを示していることを意味する．その反面，再現率が低いことから，必要な観点や欠陥を十分に網羅できておらず，そのまま利用した場合にはテスト漏れを招くリスクが高い．

以上を踏まえると，LLM は，現状ではテスト分析・テスト設計を全面的に代替する存在というよりも，人が実施するテスト分析・設計を補助するツールとして位置付けるのが妥当である．特に，初期段階での観点洗い出しや思考のたたき台として活用し，人がレビューや補完を行う前提で用いることで，実務上の効率化に寄与する可能性がある．

これらの結果は，LLM の適用可能性と限界を同時に示すものであり，今後の研究や実務への応用に向けた基盤を提供する．さらに，本研究はテスト分析とテスト設計を統合的に評価する初の試みであり，従来の研究が扱ってこなかった工程全体の比較という観点で新規性を有する．今後，対象領域や比較する項目を拡張することで，提案手法はソフトウェアテストにおける LLM 研究の比較方法の基盤としての役割を果たし得る．本研究で得られた成果は LLM を活用したテスト開発の研究に対して出発点を与えるものであると考える．

文　献

[1] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman,“GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding,” Proc. Int. Conf. on Learning Representations (ICLR), pp. 1–20, May 2019.

[2] S. Lin, J. Hilton, and O. Evans,“TruthfulQA: Measuring How Models Mimic Human Falsehoods,” arXiv preprint arXiv:2109.07958v2, May 2022.

[3] R. Koncel-Kedziorski, S. Roy, A. Amini, N. Kushman, and H. Hajishirzi,“MAWPS: A Math Word Problem Repository,” Proc. NAACL-HLT 2016, pp. 1152–1157, San Diego, USA, June 2016.

[4] F. Shi, M. Suzgun, M. Freitag, X. Wang, S. Srivats, S. Vosoughi, H. W. Chung, Y. Tay, S. Ruder, D. Zhou, D. Das, and J. Wei,“Language Models are Multilingual Chain-of-Thought Reasoners,” arXiv preprint arXiv:2210.03057v1, Oct. 2022.

[5] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, Ł. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba,“Evaluating Large Language Models Trained on Code,” arXiv preprint arXiv:2107.03374v2, Jul. 2021.

[6] Google Research,“MBPP: Mostly Basic Python Problems,” GitHub repository, https://github.com/google-research/google-research/tree/master/mbpp, accessed Sept. 2025.

[7] D. Liang, M. Berant, Y. Chen, Q. Huang, J. D. Weston, K. Duh, and P. Liang,“Learning Dependency-based Compositional Semantics,” Proc. ACL 2010 Short Papers, pp. 590–596, Jul. 2010.

[8] Hugging Face,“MBPP-ja: Japanese Translation of Mostly Basic Python Problems,” Hugging Face Datasets, https://huggingface.co/datasets/llm-jp/mbpp-ja, accessed Sept. 2025.

[9] C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, and K. Narasimhan,“SWE-bench: Can Language Models Resolve Real-World GitHub Issues?,” Proc. Int. Conf. on Learning Representations (ICLR), pp. 1–20, Apr. 2024.

[10] B. Li, W. Wu, Z. Tang, L. Shi, J. Yang, J. Li, S. Yao, C. Qian, B. Hui, Q. Zhang, Z. Yu, H. Du, P. Yang, D. Lin, C. Peng, and K. Chen,“DevBench: A Comprehensive Benchmark for Software Development,” arXiv preprint arXiv:2403.08604v1, Mar. 2024.

[11] W. Wang, C. Yang, Z. Wang, Y. Huang, Z. Chu, D. Song, L. Zhang, A. R. Chen, and L. Ma,“TestEval: Benchmarking Large Language Models for Test Case Generation,” arXiv preprint arXiv:2406.04531v2, Jun. 2024.

[12] K. Jain, G. Synnaeve, and B. Roziere,“TESTGENEVAL: A Real World Unit Test Generation and Test Completion Benchmark,” Proc. Int. Conf. on Learning Representations (ICLR), pp. 1–20, Mar. 2025.

[13] D. Huang, J. M. Zhang, M. Harman, Q. Zhang, M. Du, and S.-K. Ng,“Benchmarking LLMs for Unit Test Generation from Real-World Functions,” ACM Trans. Softw. Eng. Methodol., vol. 34, no. 1, pp. 1–26, Aug. 2025.

[14] Q. Zhang, Y. Shang, C. Fang, S. Gu, J. Zhou, and Z. Chen,“TestBench: Evaluating Class-Level Test Case Generation Capability of Large Language Models,” arXiv preprint arXiv:2409.17561v1, Sep. 2024.

[15] N. Mündler, M. N. Müller, J. He, and M. Vechev,“SWT-Bench: Testing and Validating Real-World Bug-Fixes with Code Agents,” arXiv preprint arXiv:2406.12952v3, Feb. 2025.