自然言語のテストシナリオから
保守性の高いテストスクリプトを自動生成する手法
～Web エージェント技術を活用したアプローチ～

Automatic Generation of Maintainable Test Scripts
from Natural Language Test Scenarios
- An Approach Utilizing Web Agent Technology -

NTT 株式会社 コンピュータ＆データサイエンス研究所
Computer and Data Science Labs., NTT, Inc.
○切貫 弘之　但馬 将貴　若林 慧
○Hiroyuki Kirinuki　Masaki Tajima　Kei Wakabayashi

Abstract Recent advances in web agent technologies enable automated browser operations via natural language, but challenges remain in reliability, efficiency, and element identification, especially as web pages grow more complex and dynamic. To address these issues, we propose a novel framework that generates deterministic Gherkin test cases and JavaScript step definitions from natural language instructions. Our tool enhances reproducibility, maintainability, and accuracy through modular step reuse, scenario abstraction, and hierarchical element exploration. Our evaluation demonstrates that our tool enables higher abstraction and reuse in test generation, and achieves better web element identification accuracy compared to naive set-of mark prompting technique.

1. はじめに

近年、大規模言語モデル（LLM）の進展により、自然言語の指示を通じて Web ブラウザを操作する自律型 Web エージェントが実現している[1]。これらのエージェントは Web コンテンツを認識し、タスク実施に必要な手順を推論することで、一連の操作を実行できる。

このような LLM ベースの Web エージェントを Web アプリケーションのエンドツーエンド（E2E）テストへ応用する試みも報告されている[2, 3]。しかし、2025 年現在の Web エージェントは安定かつ再現可能なテスト実行が困難であるため、実用的な自動テスト実行には不十分である。たとえば、エージェントは同じ指示に対して異なる振る舞いをしたり、予期しない画面状態からのリカバリに失敗したりすることがある。実際に、代表的な OSS の Web エージェントである browser-use[4]は、既存データセットを用いたオフライン評価により 89%の精度を自ら報告しているが、実在の Web サイト上の現実的なシナリオを用いた評価では、30%しかタスクを完了できないことが調査により示されている[5]。

また、Web エージェントによるリアルタイムのテスト実行は、操作ごとに LLM の呼び出しを行う必要があり、時間的・金銭的コストが高いため、何度も実行する回帰テストに適用しにくい。したがって、E2E テスト自動化に Web エージェントを利用する場合、リアルタイム実行に依存せず、レビューおよび保守が可能なテストスクリプトを生成する方法が適していると考える。

NTT 株式会社 コンピュータ＆データサイエンス研究所
Computer and Data Science Labs., NTT, Inc.
〒108-0075 東京都港区港南 1 丁目 2-70 29F　Tel: 03-5860-6843　e-mail:hiroyuki.kirinuki@ntt.com
1-2-70 Konan, Minato-ku, Tokyo, Japan.

【キーワード：】ソフトウェアテスト、テスト自動化、End-to-End テスト、大規模言語モデル、Web エージェント

Web エージェントを活用した E2E テストスクリプト生成の主要な技術課題として、次の 2 つが挙げられる。1 つ目は、既存手法のように画面全体のスクリーンショットや Document Object Model（DOM）を LLM に与える手法では、複雑な Web ページにおける操作対象の特定が困難な点である。2 つ目は、エージェントの行動を単純にスクリプト化するだけでは、従来のレコード＆リプレイツールと同様に、保守性や再利用性に乏しいテストスクリプトになりうる点である。

これらの課題を解決するために、本研究では自然言語によるテストシナリオを Gherkin 形式のテストケースおよび実行可能な JavaScript のテストステップ定義へ変換する新しいテストスクリプト生成手法を提案する。提案手法は LLM を用いてシナリオ抽象化、入力値パラメータ化、テストステップ再利用を行うことで、保守性・再利用性の高いテストスクリプトを生成する。さらに、操作すべき Web 要素をより正確に特定するため、探索範囲を狭めながら DOM ツリーを探索する戦略（hierarchical tree exploration: HTE）を導入した。

本研究の評価では、提案手法が自然言語シナリオからテストスクリプトを生成する際のテストステップの削減効果、および HTE の有効性を検証した。その結果、提案手法は生成されたテストケースの抽象化と再利用を促進し、現実的なテストシナリオにおいて、HTE は既存の Set-of-Mark（SoM）プロンプティング手法[7]と比較して Web 要素特定精度が向上することが示された。

2. 提案手法

提案手法は、OpenAI 社や Anthoropic 社が提供する汎用的なマルチモーダル大規模言語モデル（MLLM）で利用可能である。したがって、これらの基盤モデルの進化に柔軟に適応可能であり、ユーザが LLM 実行環境を管理したり、自前でモデルを学習したりする必要がない。

2.1 概要

本手法は、以下の 4 つのフェーズから構成される。(1) 次の操作の生成、(2) 操作対象要素の特定、(3) 操作の実行と観察、(4) テストスクリプト生成である。図 1 にワークフローの全体像を示す。本ワークフローは、ユーザが自然言語で与えたテストシナリオを実行可能なテストスクリプトへと変換するプロセスである。

提案手法は Web ブラウザで動作するツールとして実装されており、ユーザはテスト対象 Web アプリケーションの URL と、自然言語で記述したテストシナリオを入力する。システムは、Gherkin 形式のテストケースと、対応する JavaScript のテストステップ定義を出力する。

(1) 次の操作の生成

ここでは、現在の状態とテストシナリオに基づいて次に実行すべき操作を決定する。既存手法 SeeAct[6]のアプローチに従い、現在のページのスクリーンショットと過去の操作履歴を MLLM にプロンプトとして与えることで、次の操作を生成する。ここでの操作とは、「ID 入力フォームに～を入力する」といった自然言語のテキストである。提案手法は現在、クリック・入力・ドロップダウンリスト選択の 3 種類に対応している。提案手法は SeeAct の方式に加え、現状観察による失敗からのリカバリの仕組みを組み込んでいる。フェーズ 3 で操作後の画面の変化を MLLM に観測させることで、意図しない画面遷移などに気づき、自律的にテスト実行を軌道修正できる場合がある。また、操作対象とした要素が実際には Playwright で操作できないといったエラーが発生した場合には、エラーの原因を分析して代わりの操作を生成する。これらの動的なリカバリ機構によってテストシナリオ実行の信頼性を高めている。

MLLM がテストシナリオの完了を判断した場合、次の操作の代わりにアサーションを生成する。現在、特定の文字列が画面上に存在することを検証するアサーションと、特定の画面要素の内部テキストが特定の文字列と一致することを検証するアサーションの 2 種類のみ生成可能である。これらのアサーションに使用する文字列は、シナリオ完了時点での画面から抽出される。

(2) 操作対象要素の特定

次に、フェーズ 1 で生成された自然言語のテキストから、操作すべき Web 要素をブラウザ上で特定する。**Web エージェントでよく用いられる SoM は、操作されうる Web 要素をスクリーンショット上でラベル付きでハイライトし、その画像を MLLM に与えて操作対象要素のラベルを回答させる手法である。SoM の問題点として、画面中に操作すべき要素が多数存在していたり、大きな**

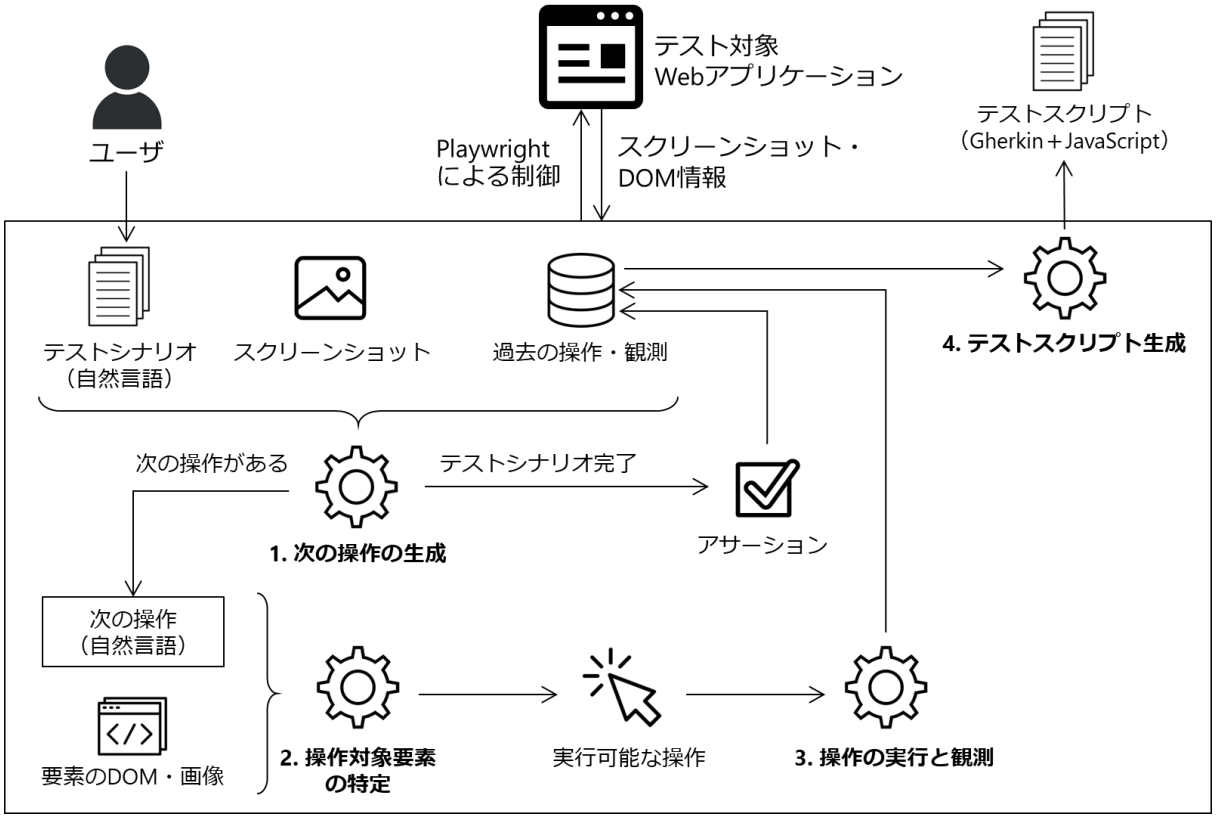

図 1: 提案手法のワークフローの全体像

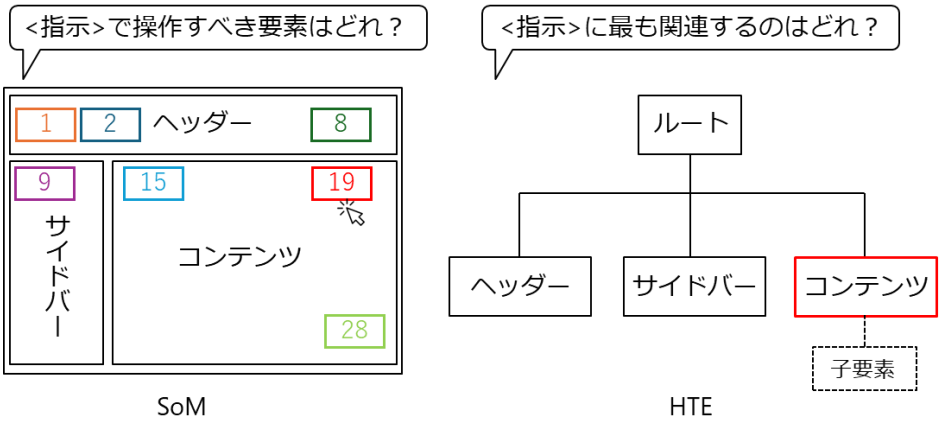

図 2: SoM と HTE の違い

画面であったりする場合、MLLM が把握すべきコンテキストが大きくなり、精度が下がる恐れがある。本研究では、この課題を解決するために、DOM ツリーを探索して操作対象を特定する手法 HTE を提案する。図 2 は SoM と HTE の違いを簡潔に示したものである。操作すべき要素が Web 画面中のコンテンツ領域にある場合、SoM では画面中のあらゆる操作可能な要素から対象を特定する必要があるが、HTE ではルート直下の 3 つの領域のいずれかの中で、コンテンツ領域にあることを特定できれば良い。

HTE は以下の手順で操作対象要素の特定を行う。

1. Web ページの DOM を解析し、各ノードが Web 要素を表すツリー構造を構築する。この際、不可視の要素や操作不能な要素はツリーから除外し、以降の処理対象としない。
2. 指定された操作に対して、ルートノードから探索を開始し、階層ごとに子ノード（すなわち各子要素に対応する画面領域）の画像および DOM を取得する。
3. MLLM にそれぞれの子ノードの画像と DOM 情報を与え、与えられた操作に最も関連する子ノードを選択させる。

4. 選択された子ノードを次の親として、3. の選択処理を再帰的に繰り返し、最終的に到達した葉ノードを操作対象要素とする。

HTE は直接の子要素だけに注目することで、MLLM に与えるコンテキストを減らせるため、画面全体のスクリーンショット上で操作可能な Web 要素をハイライトして MLLM に選択させる SoM と比較して特定精度を向上させることが期待できる。

ここで得られた操作対象に加え、操作の種類と入力値をフェーズ 1 で得られた操作内容から LLM を用いて抽出することで、操作を（操作対象，操作種類，入力値）の 3 つ組として定義する。

(3) 操作の実行と観察

ここでは、フェーズ 2 で定義された操作を Playwright によって実行し、その結果として生じる状態変化を観察する。各操作の実行後、提案手法はページ遷移や DOM の更新など、発生した状態変化を監視する。これらの変化は LLM によって要約され、操作と同様に記録される。この情報は、フェーズ 1 で次の操作を決定する際に活用される。

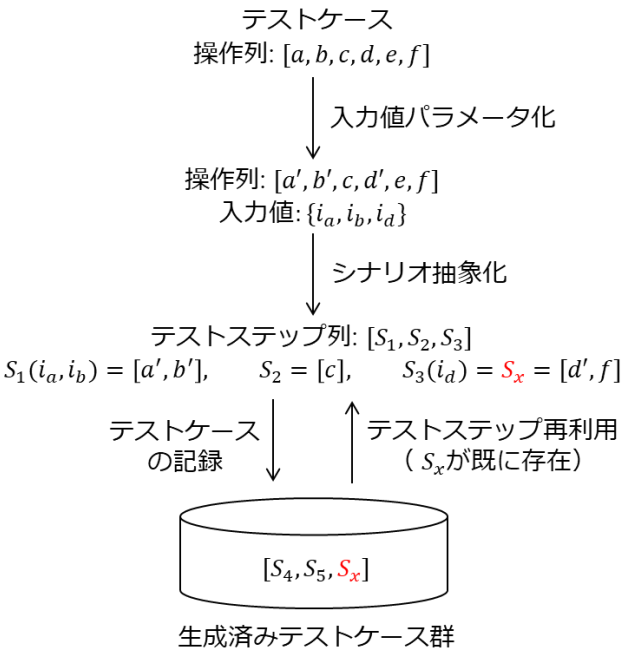

図 3: テストケース抽象化の例

(4)テストスクリプト生成

最後に、これまでのフェーズで実行された操作をテストスクリプトへ変換する。本手法では、第 1 節で述べた動機に基づき、以下の 3 種類のテストケース抽象化を導入している。

- **入力値パラメータ化:** 得られた操作列に対し、入力値にあたる箇所をパラメータ化し、処理と入力値を分離することで、同じテストケースを複数の入力データセットで実行するデータ駆動テストを可能にする。ここでは、操作からパラメータ化すべき箇所を分析するために LLM を用いる。
- **シナリオ抽象化:** 入力値がパラメータ化された操作列を、ログインやユーザ登録といった高レベルなテストステップへとまとめることで、テストステップの可読性と再利用性を高める。このプロセスでは、LLM の能力を活用し、実行された操作列から再利用可能なパターンを抽出する。さらに、観察結果に基づき、失敗した操作や冗長な操作はテストステップから除外される。
- **テストステップ再利用:** 複数のテストケース間でテストステップを再利用し、重複コードを削減して保守性を高める。入力値の違いを無視し、操作対象と操作種類が一致するテストステップが過去に生成済みである場合、そのテストステップは再利用される。

抽象化の手順の例を示したものが図 3 である。操作a~fで構成されるテストケースがあるとする。ここで、操作a, b, dが入力値を伴う場合、これらが入力値i_a, i_b, i_dをとるようにパラメータ化する。この際、操作a, b, dを入力値の部分をプレースホルダに置き換えた操作a', b', d'に変換する。次に、シナリオ抽象化によって、一連の意味を持つ操作をまとめ、テストステップとして定義する。図の例では、操作eは冗長または失敗した操作として、テストステップから除外されている。ここで、テストステップS_3とそれを構成する操作全ての操作対象および操作種類が一致するS_xが過去に生成されていたとする。この場合、S_3は新たに定義されず、S_xを利用する。

図 4 は、提案手法が生成した Gherkin 形式のテストケースの一例である。この例は、ケーススタディ（3.1 節）で用いた PetClinic アプリケーションにおけるペット追加のテストケースである。テストケースの内容が `Scenario Outline` で示され、その中では一連のユーザ操作が高レ

```
Scenario Outline: Verify Adding a Pet Visit for an Owner
  and Confirming the Visit Details Displayed
    Given Open the page
    When Search for an owner by entering <lastName> in the
      last name field and submitting the search
    When Add a visit to the pet by clicking 'Add Visit',
      entering <description>, and submitting the form
    Then Assert that the string "<assertString>" exists within the page.
    Examples:
    | lastName | description     | assertString     |
    | Franklin | Routine checkup | Routine checkup |
```

図 4:生成された Gherkin 形式のテストケースの例

```
When("Add a visit to the pet by clicking 'Add Visit', entering {}, and submitting the
form", async function (description) {
  await this.page.locator("//body[1]/div[1]/div[1]/table[2]/tbody[1]/tr[1]/td[1]
    /table[1]/tbody[1]/tr[1]/td[2]/a[1]").click();
      // Click the "Add Visit" link in the "Pets and Visits" section.
  await this.page.locator("#description").fill(description);
      // Type <description> into the "Description" input field.
  await this.page.locator("//body[1]/div[1]/div[1]/form[1]/div[2]/div[1]/
    button[1]").click();
      // Click the "Add Visit" button located below the "Description" input field.
});
```

図 5: テストステップに対応する JavaScript の定義

ベルなテストステップとして抽象化されている。入力値は <lastName>、<description>、<assertString> というプレースホルダによりパラメータ化が行われており、Examples に書かれた入力値がテスト実行時に用いられる。Scenario Outline や各テストステップを説明するテキストは、記録された操作をもとに LLM が生成しており、テストケースの可読性向上に寄与している。

図 5 は、図 4 のテストケース中のテストステップ「Add a visit to the pet by clicking “Add Visit”, entering <description>, and submitting the form」の JavaScript による定義を示している。このテストステップは 3 つの操作によって構成されており、description が引数として与えられる。

テストステップ定義で用いられるロケータは、id, name, text といった頑健性が高いものが優先されるが、それらが無い場合は XPath が用いられる。一般的に XPath によるロケータは可読性が低いが、フェーズ 1 で生成されたテキストをコメントとして付与することで、操作の内容が理解しやすくなっている。

3. 評価

本節では、提案手法の評価を 2 つの観点から行う。1 つ目は Spring PetClinic アプリケーションを対象とした自然言語シナリオからの実行可能テスト生成に関するケーススタディ、2 つ目は複雑な実環境の Web アプリケーションを用いた Web 要素特定精度の評価である。また、本評価で用いた MLLM は OpenAI API の gpt-4o である。

3.1 ケーススタディ：自然言語シナリオからのテスト生成

Spring PetClinic アプリケーションを対象に、提案手法が高レベルの自然言語シナリオから実行可能なテストスクリプトを生成できるか、また生成されたスクリプトがどの程度具体的なテ

表 1: PetClinic のテストシナリオに対して生成されたテストスクリプトの操作数とステップ数

画面	#	テストシナリオ(概略)	操作数	ステップ数
オーナー検索ページ	1	オーナー検索で 1 件ヒットした場合、オーナーページが表示される。	3	1
	2	オーナー検索で 2 件以上ヒットした場合、オーナー検索結果ページに一覧表示される。	3	1
	3	オーナー検索で何も入力しなかった場合、全オーナーが検索結果ページに表示される。	2	1
	4	オーナー追加ページへ移動する。	2	1
オーナー検索結果ページ	5	オーナー名をクリックしてオーナーページへ移動する。	3	1
オーナーページ	6	ペット追加ページへ移動する。	4	1
	7	ペット編集ページへ移動する。	6	2
	8	オーナー編集ページへ移動する。	4	1
	9	訪問データ追加ページへ移動する。	4	2
オーナー追加/編集ページ	10	入力欄を埋めてオーナーを追加する。	8	1
	11	入力欄が空欄のままオーナーを追加する。	3	2
	12	入力欄を埋めてオーナーを編集する。	7	2
	13	入力欄が空欄のままオーナーを編集する。	7	3
ペット追加/編集ページ	14	入力欄を埋めてペットを追加する。	9	2
	15	入力欄が空欄のままペットを追加する。	5	3
	16	入力欄を埋めてペットを編集する。	6	2
	17	入力欄が空欄のままペットを編集する。	6	3
訪問データ追加ページ	18	入力欄を埋めて訪問データを追加する。	6	2
	19	入力欄が空欄のまま訪問データを追加する。	5	2
ヘッダ	20	獣医一覧ページへ移動する。	1	1
	21	エラーサンプルページへ移動する。	1	1

スト手順を抽象化・簡略化できるかを評価した。

(1) 実験設定

2 名の開発者がそれぞれ独立に PetClinic 向けの E2E テストケースを作成し、詳細な手順は記さず「何をテストするか」のみを記述した。2 名が同一内容のテストケースを作成した場合は、一方のみを採用した。その結果、表 1 に示す 21 のテストシナリオが得られた。これらのシナリオを入力として提案手法を適用し、テストスクリプトを生成した。

各機能に対して提案手法に与えたのは自然言語のテストシナリオと、検索機能のテストに必要な事前登録済みのオーナー情報のみであり、追加の仕様情報や操作手順は与えていない。テストシナリオには曖昧な記述も含まれていたが、生成スクリプトがテストシナリオと矛盾しなければ有効と見なした。例えば、テストシナリオに「オーナーを編集」としか書かれていない場合、どのフィールドを編集しても問題ないとした。各シナリオについて有効なテストスクリプトが得られるまで最大 3 回試行した。

(2) 結果と考察

表 1 は各テストシナリオに対して生成されたテストスクリプトの操作数および抽象化された Gherkin 形式のテストステップ数を示している。提案手法は全テストシナリオに対して有効なテストスクリプトを生成できた。本評価では、「操作」をクリックや入力といった単一の操作、「ステップ」を複数の操作から成るひとまとまりの手順（例：オーナー登録）と定義している。実装の仕様上、生成されたテストスクリプトは必ず対象 URL を開く単一の命令で始まり、アサーションで終わる。これらはユーザ操作に該当しないため、操作数およびステップ数としてカウントし

ていない。

全テストケースで行われる操作数を合計すると 95 個であったが、生成された JavaScript コードには 69 個しか含まれなかった。これは共通するテストステップが複数のテストケースで利用されたためである。たとえば、ページナビゲーションやフォーム入力を行うテストステップは複数のテストケースで利用されていた。さらに、95 個の操作が 35 個のテストステップに集約されており、シナリオ抽象化とテストステップ再利用がテストスクリプトの単純化に寄与することが確認できた。

(3) 操作失敗からのリカバリ

提案手法は操作実行の失敗からの回復能力を示した。例えば、ペット追加・編集ページでは生年月日の入力に直接の文字入力ではなく、入力欄をクリックしてカレンダーを開いて日付を選択する必要がある。フェーズ 1, 2 において、初回は日付を直接入力しようとして失敗したが、Playwright のエラー内容を分析することで、次の操作ではカレンダーを開いて日付を選択するように動作を修正した。失敗したアクションは出力されたテストスクリプトには含まれなかった。

(4) 抽象化における課題

提案手法は操作列を再利用可能なテストステップに抽象化する。本評価では、テストケース間で抽象化の方法が一貫せず、似た操作列が異なる形で抽象化されることが観察された。例えば、操作列$[a,b,c]$を持つテストケースT_1, T_2 がそれぞれ $T_1 = [a],[b,c]$、$T_2 = [a,b],[c]$ という 2 つのテストステップにまとめられた場合、最終的に$\{[a],[a,b],[b,c],[c]\}$という操作が重複する複数のテストステップが生成され、テストスクリプトが冗長になる。したがって、今後の改善として、シナリオ抽象化において過去に生成されたテストステップを参照することで抽象化の方針に一貫性を持たせるようにしたい。

3.2 Web 要素特定戦略の評価

(1) 実験設定

前節のケーススタディが比較的単純な Web アプリを対象としたのに対し、本実験では HTE の要素特定能力を、実在の複雑な Web サイトで評価した。比較対象は単純な SoM（set-of-mark prompting）とした。

自然言語からの E2E テスト生成を評価するための既存データセットは見つけられなかったため、本研究では SeeAct の評価にも用いられた Mind2Web データセット[1]のオンライン評価用サブセットを利用した。提案手法は動的に要素を探索する必要があるため、Web エージェントの評価でよく用いられるオフライン評価用データセットは利用できない。実験で用いるデータセットは実在する Web サイトの URL とタスクを提供する。可能な限り要素特定の精度のみを評価するため、本評価では各タスクの最初の操作が正しく実行できるかのみを判定した。なぜなら、もし最初の操作を誤るとその後の要素特定が無意味になり、有効な評価が行えないためである。

データセットの全 90 タスクからユニークな Web サイトごとに 1 タスクを抽出し、51 サイト・51 タスクを対象として SoM と HTE の成功率を比較した。これは、実環境を用いた評価には人的監視が必要になり、全てを対象とするにはコストが高く対象を絞る必要があったが、評価対象の多様性は確保したかったためである。Web 要素特定の前段階である操作生成フェーズは、SoM と HTE 共に提案手法のものを用いた。SoM による Web 要素特定フェーズでは、操作対象要素のハイライト付きスクリーンショットと DOM 情報を与えた。HTE では子要素の画像と DOM 情報を逐次与え、操作対象が特定されるまで繰り返した。各試行は 3 回行い、少なくとも 1 回成功すればそのタスクは成功と見なした。

(2) 結果と考察

タスク 51 件中 12 件は、操作生成フェーズでの失敗、もしくは iframe・shadow DOM に対応していないといった実装上の制約により評価対象外となった。残りの 39 件のうち、HTE と SoM の両方で成功したタスクは 33 件であった。HTE は SoM が失敗した 5 件で成功し、SoM が成功したが HTE は失敗したタスクは、Web 要素の画像取得エラー1 件のみであり、これは実装の問題に起因するものであった。これにより、HTE が複雑な Web ページにおいて SoM より高い精度で要素を識

別できる可能性が示唆された。さらに、SoM は特に操作対象が画面上で目立たない場合や大きな画面において失敗しやすいことが分かった。これは、スクリーンショットの解釈が MLLM にとって難しいためと考えられる。一方、HTE は Web 要素の部分的な画像を利用するためこの問題を回避しやすいと考えられる。

ただし、本評価は提案手法が既存の Web エージェント技術よりも優れていることを主張するものではない。例えば、SeeAct は、単純な SoM ではなく、操作対象要素を絞り込むために学習させた専用モデルを用いることで精度を高めている。HTE はこれらの既存手法と組み合わせることで、さらなる精度向上が期待できると考える。HTE の欠点として、DOM ツリーを探索するというアルゴリズムの性質上、SoM よりも MLLM への問い合わせ回数が多いため、計算コストが高くなる点がある。しかし、Web 要素特定はテストスクリプトの生成のためにのみ行えばよく、テストスクリプト生成後は MLLM を利用する必要がない。そのため、MLLM の利用コストは、Web エージェントの一般的な用途と比較すると問題になりにくいと考える。

4. まとめ

本論文では、Web アプリケーションを対象に、自然言語のテストシナリオから保守性と再利用性の高い E2E テストスクリプトを生成する手法を提案した。本手法は、LLM を用いたテストスクリプトの抽象化と操作対象 Web 要素特定のための探索手法 HTE を導入することで、Web エージェントを E2E テストに利用する際の課題に対処した。

また、評価実験により、提案手法がテストスクリプトの抽象化によって、簡潔かつ実行可能なスクリプトを生成できることを示した。また、HTE は Web エージェントでよく採用される単純な SoM よりも高精度で要素を特定できることを示した。

提案手法を用いてテストスクリプトを生成することで、テストを実行するたびに LLM を呼び出す必要がなくなり、時間的・金銭的コストを削減できる。また、自然言語のテストシナリオからテストスクリプトを生成することで、E2E テスト自動化の導入障壁を下げることが期待できる。

今後の課題として、シナリオ抽象化の一貫性向上や、より安定したテスト実行のための高度なプロンプティング技術や機械学習技術の導入などが挙げられる。また、有望な発展の方向性として、アプリケーションを自動的に探索してテストシナリオを生成することで、テスト設計の負担を軽減し、ユーザの入力なしにテストスクリプトを生成することが考えられる。

参考文献

[1] X. Deng, Y. Gu, B. Zheng, S. Chen, S. Stevens, B. Wang, H. Sun, and Y. Su, “Mind2Web: Towards a Generalist Agent for the Web,” arXiv:2306.06070, 2023.
[2] A. Chevrot, A. Vernotte, J.-R. Falleri, X. Blanc, and B. Legeard, “Are Autonomous Web Agents Good Testers?,” arXiv:2504.01495, 2025.
[3] S. Wang, S. Wang, Y. Fan, X. Li, and Y. Liu, “Leveraging Large Vision-Language Model for Better Automatic Web GUI Testing,” in IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 125-137, 2024.
[4] Müller, Magnus and Žunič, Gregor, “Browser Use: Enable AI to control your browser,” GitHub, URL: https://github.com/browser-use/browser-use, 2024
[5] X. Tianci, W. Qi, T. Shi, et al. “An Illusion of Progress? Assessing the Current State of Web Agents.” arXiv:2504.01382, May 24, 2025.
[6] B. Zheng, B. Gou, J. Kil, H. Sun, and Y. Su, “GPT-4V(ision) is a generalist web agent, if grounded,” in Proceedings of the 41st International Conference on Machine Learning (ICML), pp. 61349-61385, 2024.
[7] J. Yang, H. Zhang, F. Li, X. Zou, C. Li, and J. Gao, “Set-of-Mark Prompting Unleashes Extraordinary Visual Grounding in GPT-4V,” arXiv:2310.11441, 2023.