B-002

LLM を利用した深層強化学習エージェントによるウェブアプリのバグの発見
Using LLM-Based Deep Reinforcement Learning Agents to Detect Bugs in Web Applications

酒井 佑旗[1)]　田原 康之[1)]　大須賀 昭彦[1)]　清 雄一[1)]
Yuki Sakai　Yasuyuki Tahara　Akihiko Ohsuga　Yuichi Sei

1　はじめに

ソフトウェア開発においてテストは非常に重要な工程である．特にウェブアプリのブラックボックス GUI テストには大きなコストがかかる．特にテストシナリオの作成，シナリオに沿ったテストの実行がボトルネックとなる．そのため自動化を試みる研究が取り組まれている．またシナリオを作成しないテスト手法に探索テストが挙げられる．探索テストは実施者の直感や経験を活かしてバグを発見するアプローチである．探索テストにも自動化の流れがあり，深層強化学習（DRL）を用いたアプローチが中心である．近年の研究では GUI の状態に限らず，GUI 要素のインタラクション情報を学習に利用することで，エージェントの性能が向上するといった結果が示されている [1]．しかしウェブアプリにおいて GUI 要素すなわち HTML 要素のインタラクション情報はブラウザに依存する情報であり，ブラウザによっては取得できないこともある．そこで大規模言語モデル（LLM）を用いて HTML からインタラクション情報を推論させ，その結果を状態の一部とし，推論結果を使用しなかった場合と同等の精度となるか検証する．この仮説は LLM の広範な知識による推論と深層強化学習のロバストネスの相性に基づいている．

この仮説が立証された場合，さまざまな分野への応用が考えられる．LLM の広範な知識はウェブアプリに限定されないため，深層強化学習における状態の取得が難しい場面やエージェント学習時の手掛かりとしての活用が期待される．

2　ウェブアプリの自動テスト

本研究でテスト対象にウェブアプリを選定した理由は 2 つある．1 つ目はモバイルアプリよりもウェブアプリの方が自動テストに向いているという点，2 つ目は GUI 自動テストに関する研究にはモバイルを対象としたものが多い点である．

まずは 1 つのウェブアプリの方が自動テストに向いている点だが，その理由としてリリーススパンの違いが挙げられる．モバイルアプリはアプリストアを経由して配布される．通常ストアへのリリースには審査が必要であり，リリースまでに数日かかることがある．iOS 向けアプリを配布する App Store の場合は平均して 90%が 1 時間以内にレビューが完了する [2]．しかし平均であるため，初回審査時などの場合数日かかることもある．一方 Android 向けの Google Play では，最大 7 日程度かかることがある [3]．このようにモバイルアプリはリリースに一定の日数を要する．リリースに時間がかかるということは，バグ修正のリリースにも時間がかかるということである．従って新機能リリース前には十分なテストが求められる．一方ウェブアプリはサーバーにファイルをアップロードするだけでリリースが完了する．自動テストによってテスト工程を省力化することで，より高頻度なリリースが可能となる．従ってウェブアプリは自動テストに向いている．

次に 2 つ目のブラックボックス GUI テストの自動化に関する研究の対象はウェブアプリがほとんどである点だが，近年のスマートフォンの普及の影響もあると推測される．そこで本研究ではモバイルアプリを対象として提案された ARES[1] を参考にし，DRL の状態空間に HTML 要素のインタラクション情報を含める手法を検証する．モバイルアプリの GUI 状態を取得する場合，Appium を利用することが一般的であり，タップ，ロングタップ，スワイプなどのインタラクション情報が取得可能である．これらのインタラクションはモバイルアプリ特有のものであり，ウェブアプリには存在しない．そこで本研究ではインタラクションの種類を Selenium で取得可能なクリックに限定することとした．このような理由から本研究ではウェブアプリを探索対象とすることした．

3　関連研究

3.1　DRL を使用した GUI テスト

GUI ブラックボックステストの自動化はモンキーテストフレームワーク [4] や Q 学習ベースのアプローチ [5] が提案されてきた．最近では深層強化学習（DRL）を利用した手法も提案されている．Eskonen らはウェブアプリにおいて GUI のスクリーンショットを DRL の入力とした画像ベースの手法を提案し，ランダムな探索や Q 学習ベースの手法よりも高い探索精度を達成した [6].

Andrea らは DRL ベース Android アプリテストフレームワークである ARES を提案した [1]．図 1 に ARES の概要図を示す．アプリの GUI は Appium を経由して xml 形式で取得される．アプリの GUI 要素の可視性とインタラクションの有効性からなるベクトルを状態とし，エラーの発見や新しい要素の探索を正の報酬として学習を行なった．結果として Q 学習ベースの手法よりも高い探索精度を達成した．

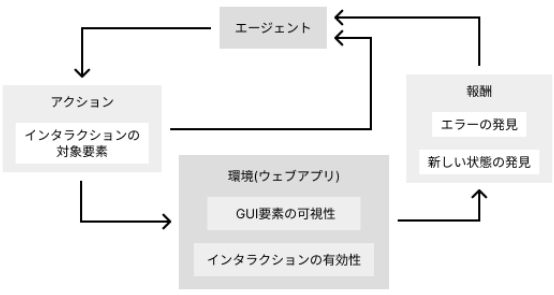

図 1　ARES の概要図

自動テスト実行時の GUI の取得にはモバイルアプリ

1)　電気通信大学

は Appium，ウェブアプリは Selenium が利用される．しかしフレームワークフレームワークによって取得可能な情報や実行可能な操作が異なる．そのためウェブアプリとモバイルアプリの GUI テストの自動化には異なるアプローチが必要である．そこでモバイルアプリ向けに提案された手法をウェブアプリに応用することに価値がある．

3.2 GUI テスト特化 LLM エージェント

yoon らは LLM を利用した GUI テストフレームワークを提案した [7]．フレームワークはプランナー，アクター，オブザーバー，リフレクターの 4 種類のエージェントで構成される．まず初めにプランナーがテスト内容の多様性，リアリズム，難易度，重要性を考慮して抽象度の高いテストケースを生成する．次にアクターが生成されたテストケースを達成するための具体的なアクションを決定・実行する．オブザーバーはアクション後の状態を観察し，状態を JSON で出力する．最後にリフレクターが実行内容を振り返り，プランナーへフィードバックを行う．この手法は探索カバレッジや機能カバレッジにおいて有意な結果を示した．yoon らは LLM として OpenAI API を使用したため，金銭的なコストを課題としてあげた．

3.3 報酬関数としての LLM

kwon23 らは強化学習の報酬関数として LLM を利用する手法を提案した [8]．強化学習の報酬設計は人間の望ましい行動を報酬関数を通じて指定することが難しいこと，効果的な報酬関数を設計するためには専門的な知識を要することから困難であると言える．そこで kwon23 らは LLM を利用することで，自然言語をインターフェースとして利用できるようにし，報酬関数の設計難易度を下げることに成功した．

ユーザーは望む行動の例や説明をテキストプロンプトとして LLM に提供し，LLM がこのプロンプトに基づいて報酬信号を出力することで，強化学習エージェントの行動を更新する．複数のタスクにおいて，提案手法が従来手法よりも優れた性能を示した．

本手法は報酬関数として LLM を利用している．強化学習ではアクションの実行や状態の取得にも専門的な知識を必要とすることがる．これらは学習に重要な影響を与えるため LLM の応用が期待される．

4 提案手法

4.1 強化学習手法

本研究では深層強化学習アルゴリズムには，Proximal Policy Optimazation (PPO)[9] を利用する．DRL アルゴリズムに関する実装には，Stable Baselines3[10] を使用した．本研究では強化学習手法の変更による性能の向上を目的としておらず，LLM の推論結果での代替による性能の変化を検証することが目的である．従って学習の安定性を長所とする PPO を採用する．PPO はその使いやすさと優れたパフォーマンスにより，OpenAI のデフォルトの強化学習アルゴリズムに採用されている [11]．

4.2 LLM による推論結果の利用

ウェブアプリは一般的にブラウザを介して操作される．同一のソースコードであってもブラウザによって挙動が異なることがあるため，ブラックボックスな GUI テストはブラウザごとに実施することが望ましい．ウェブアプリの自動テストフレームワークは Selenium が有名であり，Selenium にはブラウザごとにドライバーが用意されているため，それらを利用してブラウザごとにテストすることが可能である．

本研究では HTML 要素がクリック可能かどうかという，インタラクション情報を学習に利用する．先行研究はモバイルアプリを対象としていたため，テストフレームワークに Appium を使用していたが，ウェブアプリでは Selenium を使用する．そして Appium と Selenium では取得可能なインタラクション情報が異なる．

またウェブアプリにおいて要素がクリック可能かどうかという情報は，ランタイム情報であり，ブラウザを介して取得する必要がある．Chrome の場合，情報の取得に必要な Chrome Devtools Protocol コマンドを実行するためのインターフェースが Selenium に用意されている．一方 Safari, Firefox の場合，ブラウザの開発者ツールからは取得可能だが，Selenium 上にはインターフェースが存在しないため，取得することが難しい．加えて Selenium のインターフェース以前にブラウザ自体が対応していないことも考えられる．そこで本研究では LLM を利用して HTML からクリック可能かどうかを推論し，DRL の状態の一部として利用する手法を提案する．推論結果で代替可能であることが分かれば，さまざまなブラウザ GUI ブラックボックステストを効率的に行うことができるようになる．

4.3 機械学習モデルのロバストネスと LLM

本研究では DRL の状態の一部として LLM の推論結果を利用する．これは LLM の推論精度が 100%ではないという特徴と，機械学習モデルのロバストネスが補完関係にあり，相性が良いという仮説に基づいている．LLM の推論精度は 100%ではない．これは学習データの不完全性や LLM 自体が確率的なモデルである点，自然言語の曖昧さなどが理由である．一方機械学習モデルにはロバストネスという特徴がある．ロバストネスとは，モデルがノイズに対して頑健であることを指す．これは入力に多少のノイズが含まれていても，モデルの出力が安定していることを意味する．本研究ではこれら特徴の相性を活かして，LLM の推論結果を DRL の状態の一部として利用する．

5 実験方法

本研究ではまず初めにウェブアプリにおいても HTML 要素のインタラクション情報を利用することで学習が効率化されることを検証する．次に LLM を利用して HTML 要素のインタラクション情報を推論し，その結果を DRL の状態の一部として利用することで，推論結果を使用しなかった場合と同等の精度が得られるか検証する．

5.1 オリジナルのテスト対象のアプリ作成

本研究ではテスト対象としてオリジナルのウェブアプリを作成し利用した．これはウェブアプリのブラックボックス GUI テストの自動化に関する研究におけるテスト対象アプリが一般化されていないことと，DRL の状態の一部としての LLM の利用を検証するためである．

作成したウェブアプリを図 2 に示す．このウェブアプリは 3 つの状態を持つ．画面上部には 3 つのボタンがあり，クリックすることで状態が切り替わる．また現在の状態は画面下部にテキストで表示される．全ての状態は常に 1 アクションで遷移することができる．

ウェブアプリは Vue.js[12] を利用して作成した．ボタンは全て button タグで記述されており，クリック可能である．現在の状態の表示は div タグを利用している．本アプリはシングルページアプリケーションとして作成されており，URL は変化しない．また vue-cli-service の serve コマンドを利用してローカルサーバーを立ち上げ，Selenium 経由でアクセスすることで実行した．

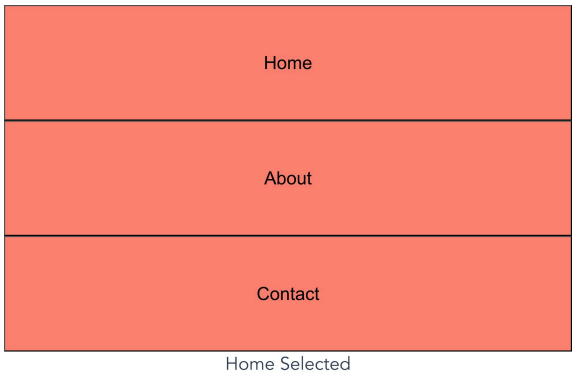

図 2　テスト対象のアプリ

5.2　実験 1 ウェブアプリへの応用

まず初めにオリジナルウェブアプリにおける自動テストエージェントの学習を行った．学習エージェントの学習を通して，ウェブアプリにおいてもインタラクション情報を活用することで学習が早くなり，精度が向上することを検証した．システムの概略図を図 3 に示す．ウェブアプリは Selenium を通して操作・情報取得することで OpenAI の Gym 環境 [13] に変換される．そして Stable Baselines3 を利用して PPO による学習を行う．Selenium はブラウザ操作を行うためのフレームワークであり，一般的にウェブアプリの自動テストに用いられる．また Gym は OpenAI が提供する DRL アルゴリズムの開発や比較を容易にするためのオープンソースのツールキットである．エージェントと環境間の標準的な API を提供する．OpenAI Gym ではカスタム環境を作成することができ，OpenAI Gym のインターフェースに準拠することができる．Stable Baseline3 は DRL アルゴリズムを提供する PyTorch ベースのライブラリである．Stable Baseline3 は A2C，DDPG，DQN，HER，PPO，SAC，TD3 といったアルゴリズムに対応している．特に PPO に関しては Schulman らによる Proximal Policy Optimization Algorithms[9] を参考としている．Stable Baselines3 は学習環境と学習アルゴリズム間のインターフェースとして OpenAI Gym の環境を採用している．したがってウェブアプリを学習環境として利用するために，Selenium を通して OpenAI Gym のインターフェースに準拠させた．

学習時のパラーメータは次のように設定した．以下に記載していない項目に関しては Stable Baselines3 のデフォルト値を使用した．

ネットワークを更新　4048 ステップごと
ミニバッチサイズ　48
学習率　$3e-4$
テスト周期　10epochs

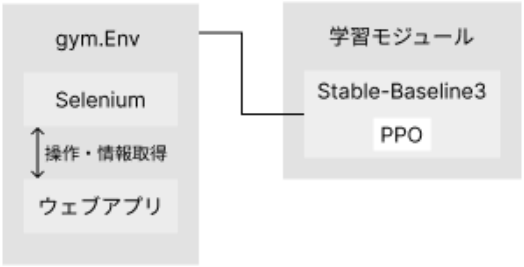

図 3　実験 1 システム概要図

オリジナルウェブアプリをラップした学習環境について説明する．

エピソード　1 エピソード 3 ステップ．1 ステップで 1 つの HTML 要素をクリックする．情報はエピソード終了時にリセットされる．
行動空間　HTML 要素ディクショナリにおける，クリック対象の HTML 要素のインデックス．
状態空間　$2 \times n$ 次元ベクトル．各行の 1 次元目は要素が画面内にあるか，2 次元目はクリック可能かどうかを表す．n は HTML 要素数である．
報酬　1 つ目の状態を発見した場合 +0.1，2 つ目を発見したら +0.2，3 つ全て発見したら +1.0 を与える．

新しい状態であるかどうかの判定には状態空間を表す行列の 1 列目を使用した．行動空間における 1 列目は各 HTML が画面内にあるかどうかを表す．従って 1 列目を抽出したベクトルを作成し，既存のベクトルと比較し，重複しなかった場合に新しい状態を発見したと判断することができる．発見した状態はエピソード終了時にリセットした．

本研究では HTML 要素がクリック可能かどうかの判断に，Click イベントのリスナーがアタッチされているかどうかという情報を利用した．Selenium 経由で取得される DOM Element には，クリック可能かどうか判定するためのインターフェースがない．button や div などの HTML タグで大まかにクリック可能かどうか判定することも可能だが，button が disabled 状態であったり，div タグに Click イベントがアタッチされていたりと単純に判定することはできない．そこで HTML 要素にアタッチされたイベントリスナーのうち，Click イベントリスナーを持つ場合にクリック可能と判定することにした．HTML 要素にアタッチされたイベントリスナーはランタイム情報であり，ブラウザに付随する開発者用ツール経由で取得する必要がある．Selenium にはブラウザごとにドライバーが用意されているが，開発者用ツールの API を利用できるかどうかはブラウザによって異なる．Chrome ドライバには開発者用 API を呼び出すインターフェースがあるが，Firefox や Safari にはない．したがって本研究では Chrome ドライバを使用した．

ステップごとの流れを図 4 に示す．まず初めに学習モジュールが決定した行動を Selenium を通してウェブアプリで実行する．アクションが完了すると，Selenium を通してウェブアプリの HTML 要素を取得する．その後各要素にアタッチされているイベントリスナーを Chrome DevTools Protocol を利用して取得する．イベン

トリスナーに Click イベントが登録されている場合，クリック可能であると判断する．これらの情報をもとに状態空間を更新し，報酬を計算する．最後に状態，報酬を学習モジュールに返却し，諸々のネットワークの更新を行う．

本システムでは HTML 要素がユニークであることを DOM Element の outerHTML のハッシュ値を使用して判定した．この方法の場合複雑なウェブアプリでハッシュ値が重複する可能性が考えられるが，本研究で使用したウェブアプリでは outerHTML が重複する要素がない．したがって outerHTML のハッシュ値を各要素のユニーク ID として扱った．

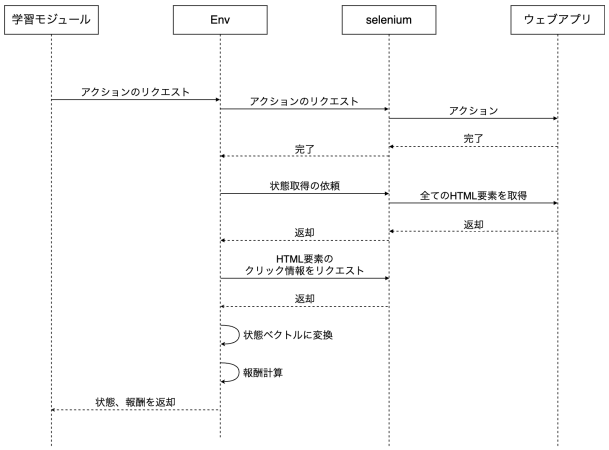

図 4　ステップごとの流れ

5.3　実験 2 ランダムノイズによる予備実験

本研究では LLM を利用して，HTML 要素がクリック可能かどうか推論させるが，LLM の推論精度は 100%ではない．これは LLM が確率的なモデルである点が理由として挙げられる．推論能力の高いモデルの使用やプロンプトの工夫によって推論精度を向上させることは可能であるが，100%の精度で推論することは不可能である．そこで実験 1 の状態空間のクリック可能かどうかを表す部分にランダムにノイズを加えることで，どの程度の精度の推論まで受け入れることができるのか検証した．LLM の推論の誤りはランダムノイズとは異なるので，大まかな目安として利用した．

本実験ではランダムノイズの割合は 0%，10%，20%とした．それぞれの割合における学習効率，精度を比較した．

5.4　実験 3 LLM によるクリック情報の推論

実験 2 ではランダムノイズの割合による学習効率，精度の違いを確認した．LLM の推論で達成すべき精度の目安が確認できたので，実験 3 ではモデルやプロンプトの改良による推論精度の向上を目指した．

本研究では強化学習の状態として推論結果を利用するので，ステップごとに推論させる必要がある．そのため OpenAI API などの有料サービスを利用することは金銭コスト的に相応しくない．したがってローカルで動かすことのできる LLM を利用して推論できる環境を構築した．モデルによって命令分の解釈が異なり，プロンプトに関してもその違いを考慮する必要がある．そこで実験 2 での結果を踏まえて基準を設け，その基準を超えたモデル・プロンプトの組み合わせを発見した段階で実験を終了した．

また，テスト対象の HTML 要素は複数のウェブサイトから手作業で 40 個抽出した．実際のウェブアプリにおける推論時の状況に揃えるため，クリック可能なものとクリック不可能なものを両方含むように抽出した．

ウェブサイト	概要
GitHub[1)]	ソースコード共有サイト
YouTube[2)]	動画共有サイト
文字数カウント[3)]	ツールサイト
レターファン[4)]	ビデオレター EC サイト

5.5　実験 4 LLM の推論結果による代替

実験 3 で HTML 要素がクリック可能かどうかを判定すると言うタスクにおいて，精度高く推論可能な LLM とプロンプトを選定した．

実験 4 では実験 1 の状態空間のうち，クリック可能かどうかを表すベクトルを LLM の推論結果で代替して学習を行った．図 5 に実験 4 のシステム概要図を示す．LLM の推論は学習モジュール，学習環境とは独立したサーバーで API としてアクセス可能な形で実装した．推論には時間がかかるため，ステップの度に実行するとシミュレーション時間が悪化してしまう．そこでレスポンスをキャッシュすることで推論時間を削減し，トータルの学習時間を短縮した．

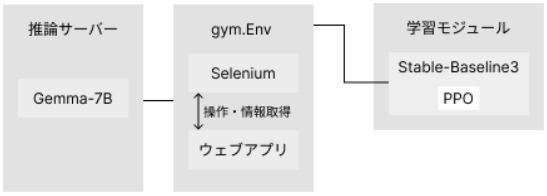

図 5　実験 4 システム概要図

6　実験結果

6.1　実験 1 ウェブアプリへの応用

実験 1 ではクリック情報の活用による学習効率，精度の変化を確認した．図 6 に平均エピソード報酬の移動平均の推移を示す．評価は毎エポック終了後に行い，5 エポック単位で移動平均を計算した．点線がクリック情報なしの場合，実線がクリック情報ありの場合を表す．HTML 要素がクリック可能かどうかを行動空間に含んだ場合の結果が，クリック情報を含まなかった場合と比べて，学習効率，精度共に向上していることが確認できる．これらの結果からウェブアプリにおいても，GUI 要素のインタラクション情報を行動空間に含むことで学習効率，精度が向上することがわかった．

4） wwwwwwhttps://github.com/
5） https://www.youtube.com/
6） http://www1.odn.ne.jp/megukuma/count.htm
7） https://letterfan.jp/

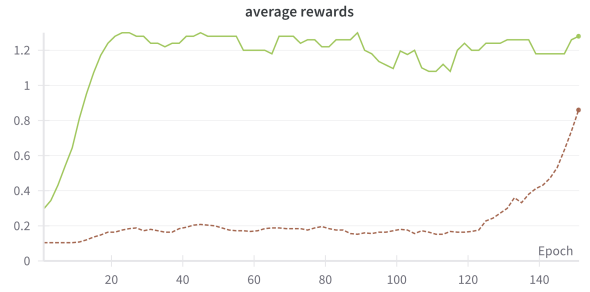

図 6　実験 1 平均報酬推移

6.2　実験 2 ランダムノイズによる予備実験

実験 2 では行動空間の一部を LLM の推論結果で代替するにあたって，目安となる推論精度を求める．図 7 に複数の割合でランダムノイズを含めた場合の平均エピソード報酬の移動平均を示す．評価は毎エポック終了後に行い，5 エポック単位で移動平均を計算した．各線はそれぞれ 0%，10%，20%のノイズを含めた場合の結果である．実線が 0%，点線が 10%，二点鎖線が 20%の場合を表す．図から 20%のノイズを含んだ場合と含まなかった場合で，精度に大きな差がないことがわかる．実験 3 では 80%以上の精度で推論可能な LLM，プロンプトを目指すこととした．

図 7　実験 2 ノイズ割合ごとの平均報酬推移

6.3　実験 3 LLM によるクリック情報の推論

実験 3 ではある HTML 要素がクリック可能かどうか 80%以上で推論することのできる LLM，プロンプトの組み合わせを見つけた．

モデル，プロンプトの検証は探索的に行った．モデルの選定にあたってローカルマシンで動作することと推論性能を重視した．その結果 google/gemma-7b を使用することとした．

またプロンプトは次のようになった．HTML を入力とし，クリック可能かどうかを 0 または 1 で返却する．入力とする HTML のうち，子要素は考慮しないように指示した．プロンプトではまず [INSTRUCTION] で命令を与え，次に [ADVICE] で推論のためのアドバイスを与える．次に [THINKING STEPS] で思考手順を示し，最後に [EXAMPLES] で具体例を示した．

```
[INSTRUCTION]
Determine whether the following HTML element is
      clickable or not.
Answer 1 if the element is clickable, and 0 if it is
      not.
Please answer with 0 or 1, and answer only at the
      beginning of your response. Do not include any
      explanations.
Think with following the steps.

[ADVICE]
- Do not consider child elements in your judgment.

[THINKING STEPS]
1. Check if the given HTML element's tag is inherently
      clickable like <button> tags, <select> and so on.
      If so, the given html element is clickable.
      Therefore return 1, and then finish thinking
      sequence.
2. If not inherently clickable, it become clickable due
      to an attribute. Examples include <a> tags with
      href attributes or <div> tags with onclick
      attributes and so on. If the element is clickable
      due to this, return 1.
3. When elements such as class names or text content
      suggest that they are clickable, they should be
      treated as clickable.

[EXAMPLES]
Here are some examples:
# Example 1:
Given HTML: <a href="https://www.example.com">Link</a>
Answer: 1
# Example 2:
Given HTML: <a>Link</a>
Answer: 0
# Example 3:
Given HTML: <p>Text</p>
Answer: 0
# Example 4:
Given HTML: <div><button>Click Me</button></div>
Answer: 0
# Example 5:
Given HTML: <button>Click Me</button>
Answer: 1
# Example 6:
Given HTML: <div onclick="alert('Clicked!')">Click Me</
      div>
Answer: 1

[ACTUAL]
GIVEN HTML:
```

手作業で抽出した検証データを対象とした推論結果は 5 回の平均で 81.5%となった．またオリジナルウェブアプリにおける推論精度は 5 回の平均で 100.0%であった．

6.4　実験 4 LLM の推論結果による代替

実験 4 では実験 3 で選定した LLM，プロンプトを用いて，HTML 要素がクリック可能かどうかを推論し，その結果を DRL の状態空間の一部として利用した．

図 8 に平均エピソード報酬の移動平均を示す．評価は毎エポック終了後に行い，5 エポック単位で移動平均を計算した．実線が LLM を利用しなかった場合，点線が LLM を利用した場合を示す．この結果から状態の一部に LLM の推論結果を利用した場合も，クリック情報を利用した場合と比べて同等の学習効率，精度であることがわかる．

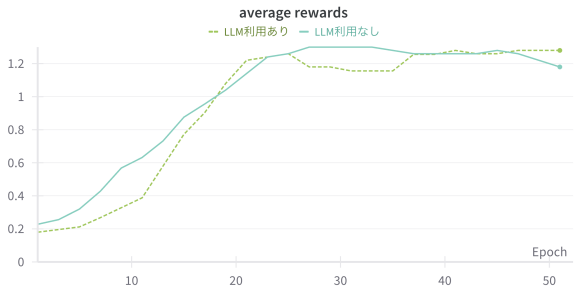

図 8　実験 4 平均報酬推移

7　考察

本研究の目的は LLM の推論結果を DRL の状態として活用できることを示すことである．ウェブアプリを単純にして最低限の構成にすることで，LLM の推論結果での代替が有効であることが確認された．本研究で用いたウェブアプリは構造が単純であるだけでなく，div タグや button タグといった HTML における基本的な要素のみで構成されていたため推論精度が高くなった．その結果，推論した情報で代替しなかった場合と比べて，遜色

のない学習効率，精度が得られた．しかし一般的なウェブアプリはさらに複雑であるため，本研究で作成したシステムでは十分な推論精度が得られず，学習が上手くいかない恐れがある．従ってより一般的なウェブアプリへ応用する場合，LLM やプロンプトの改善に限らず，モデルのファインチューニングや部分的な教師あり学習によって推論精度を向上させる必要がある．

またノイズを低減する一方で，DRL のロバストネスを向上させることも重要である．DRL にはロバストネスがあり，ノイズに対して頑健であることが知られているが，モデルによって能力に差がある．本研究では学習の安定性という観点で PPO を選定したが，さらにノイズに強いモデルを用いることで，より LLM ノイズに対してロバストなシステムを構築することができると考えられる．

強化学習はステップごとに状態を更新するため，LLM による推論を高頻度で行う必要がある．この問題を解決するために LLM をローカルできるものに限定した．このローカルマシンで実行できるモデルの採用により金銭的なコストは抑えられたが，その一方で推論に時間的コストがかかってしまった．推論結果をキャッシュすることで推論時間を削減したが，強化学習のステップ数の多さによる推論回数の多さによって，推論結果の平均が収束していくという長所を失ってしまう．高速に推論を行うことのできる環境を整えることや，推論結果のキャッシュのクリアタイミングの最適化によって，推論時間を削減することが求められる．

本研究の応用先としては様々考えられる．まず 1 つ目はコンシューマゲームのような実装がブラックボックスな環境である．ゲームを題材とした深層強化学習の研究は多く行われているが，それらは環境情報の取得が容易なシミュレーション環境を題材とすることが多い．OpenAI Gymnasium や MuJoCo，Fighting ICE などがその代表例である．これらの環境は環境情報を取得するための API が提供されているが，一般的なコンシューマゲームではそのような API は提供されておらず，値 1 つを取得するために独自の実装が必要となる．そこで LLM を用いて画像などから環境情報を推論することで，値の取得が簡素化され，深層強化学習の題材として扱うことが容易となる．2 つ目に自動運転など複雑な環境の情報を処理する必要のある分野が挙げられる．自動運転においてはカメラ映像から環境情報を取得する必要があり，画像や動画が入力とされることが多い．しかしこれらの情報は高次元であり，学習に時間がかかってしまう可能性が高い．そこで LLM を用いて画像や動画から単純化された一次元ベクトルとして情報を抽出することで，学習効率を向上させることができると考えられる．例としては現在の状況における危険度が考えられる．画像から危険度を推論し，その情報を状態空間に含めることで，危険度が高い場合には安全な行動を取るように学習させることができる．LLM には広範な知識が含まれているため，学習の効率化に役立つ可能性がある．

8 謝辞

本研究は JSPS 科研費 JP22K12157, JP23K28377, JP24H00714 の助成を受けたものです．

参考文献

[1] Andrea Romdhana, Alessio Merlo, Mariano Ceccato, and Paolo Tonella. Deep reinforcement learning for black-box testing of android apps. *ACM Trans. Softw. Eng. Methodol.*, Vol. 31, No. 4, jul 2022.

[2] Apple. App review. https://developer.apple.com/jp/distribute/app-review/. アクセス日: 2024-05-23.

[3] Google. Play console ヘルプ. https://support.google.com/googleplay/android-developer/answer/9859751?hl=ja. アクセス日: 2024-05-23.

[4] Thomas Wetzlmaier, Rudolf Ramler, and Werner Putschögl. A framework for monkey gui testing. In *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)*, pp. 416–423, 2016.

[5] David Adamo, Md Khorrom Khan, Sreedevi Koppula, and Renée Bryce. Reinforcement learning for android gui testing. In *Proceedings of the 9th ACM SIGSOFT International Workshop on Automating TEST Case Design, Selection, and Evaluation*, A-TEST 2018, p. 2–8, New York, NY, USA, 2018. Association for Computing Machinery.

[6] Juha Eskonen, Julen Kahles, and Joel Reijonen. Automating gui testing with image-based deep reinforcement learning. In *2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*, pp. 160–167, 2020.

[7] Juyeon Yoon, Robert Feldt, and Shin Yoo. Autonomous large language model agents enabling intent-driven mobile gui testing, 2023.

[8] Minae Kwon, Sang Michael Xie, Kalesha Bullard, and Dorsa Sadigh. Reward design with language models. In *The Eleventh International Conference on Learning Representations*, 2023.

[9] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[10] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, Vol. 22, No. 268, pp. 1–8, 2021.

[11] OepnAI. Proximal policy optimization. https://openai.com/index/openai-baselines-ppo/. アクセス日: 2024-05-29.

[12] Vue.js. https://ja.vuejs.org/. アクセス日: 2024-05-23.

[13] OepnAI. Oepnai gym. https://github.com/openai/gym. アクセス日: 2024-05-28.