Buletin Sistem Informasi dan Teknologi Islam
Vol 7, No 3 (2026), pp 334-344
ISSN: 2721-0901
DOI: https://dx.doi.org/10.33096/busiti.v7i3.3507

Optimalisasi Automated Unit Testing pada Pemrograman Berbasis Web Menggunakan Pendekatan Reinforcement Learning

Optimization of Automated Unit Testing in Web-Based Programming Using a Reinforcement Learning Approach

Muh Akram Riyadi Ramadhan [a,1], Muhammad Faisal [a,2], Lukman [a,3*], Desy Anggraeni [a,4], Irnawaty Idrus[b,5], M. Agusalim[c,6], Soemitro Emin Praja[d,7]

[a]*Department of Informatics, Universitas Muhammadiyah Makassar, Makassar, Indonesia*
[b]*Department of Architecture, Universitas Muhammadiyah Makassar, Indonesia*
[c]*Department of Water Resources Engineering, Universitas Muhammadiyah Makassar, Indonesia*
[d]*Department of Urban and Regional Planning, Universitas Muhammadiyah Makassar, Indonesia*

[1]*105841117423@student.unismuh.ac.id;* [2]*muhfaisal@unismuh.ac.id;* [3]*lukman@unismuh.ac.id;*
[4]*desianggreani@unismuh.ac.id;* [5]*irnawatyidrus@unismuh.ac.id;* [6]*m.agusalim@unismuh.ac.id;*
[7]*soemitroeminpraja.ft@unismuh.ac.id*
**corresponding author*

Informasi Artikel

Diserahkan : 9 Mei 2026
Diterima : 24 Juli 2026
Direvisi : 30 Agustus 2026
Diterbitkan : 31 Agustus 2026

Kata Kunci:
Reinforcement Learning
Pengujian Unit Otomatis
Pengujian Perangkat Lunak
Pengujian Adaptif
Pemrograman Berbasis Web,

ABSTRAK

Pengujian perangkat lunak penting dalam pengembangan aplikasi web, namun automated unit testing konvensional berbasis aturan statis terbatas menghadapi kompleksitas sistem modern. Penelitian ini memformulasikan prioritisasi kasus uji sebagai Markov Decision Process: agen Reinforcement Learning mengamati state, memilih kasus uji, dan menerima reward multiobjektif yang memadukan cakupan kode dan deteksi kesalahan dikurangi biaya eksekusi. Tiga algoritma (DQN, DDQN, PPO) dibandingkan terhadap baseline Random dan Greedy menggunakan lima seed dan uji Mann–Whitney U. Pada System Under Test berbasis Flask, DDQN memperoleh performa terbaik dengan APFD 0,8504, NAPFD 0,8447, dan F1 0,4183, mengungguli baseline secara signifikan (nilai p terkecil $7{,}15 \times 10^{-144}$) dengan latensi keputusan 3,0033 ms, kurang dari separuh PPO. Greedy mencapai cakupan tertinggi namun deteksi terendah, menegaskan cakupan bukan indikator memadai. Validasi eksternal pada BugsInPy (82 bug web) mengonfirmasi arah keunggulan serupa dengan margin lebih kecil. Reinforcement Learning merupakan pendekatan menjanjikan dan dapat direproduksi untuk automated unit testing adaptif pada aplikasi web.

***Keywords*:**
Reinforcement Learning
Automated Unit Testing
Test Case Prioritization
Software Testing
Adaptive Testing

ABSTRACT

Software testing is crucial in web application development, but conventional automated unit testing based on static rules has limitations when dealing with the complexity of modern systems. This study formulates test case prioritization as a Markov Decision Process: a Reinforcement Learning agent observes the state, selects a test case, and receives a multi-objective reward that combines code coverage and error detection, minus the execution cost. Three algorithms (DQN, DDQN, PPO) were compared against the Random and Greedy baselines using five seeds and the Mann–Whitney U test. On a Flask-based System Under Test, DDQN achieved the best performance with an APFD of 0.8504, a NAPFD of 0.8447, and an F1 score of 0.4183, significantly outperforming the baselines (smallest p-value: 7.15×10^{-144}) with a decision latency of 3.0033 ms—less than half that of PPO. The Greedy algorithm achieved the highest coverage but the lowest detection rate, confirming that coverage is not a sufficient indicator. External validation on BugsInPy (82 web bugs) confirmed a similar trend of superiority, albeit by a smaller margin. Reinforcement learning is a promising and reproducible approach for adaptive automated unit testing of web applications.

I. Pendahuluan

Pengujian perangkat lunak merupakan aktivitas krusial dalam pengembangan sistem berbasis web karena menjamin kesesuaian fungsionalitas yang diimplementasikan dengan spesifikasi dan perilaku operasional yang diharapkan. Namun, aplikasi web modern semakin dicirikan oleh arsitektur yang heterogen, perubahan kode yang cepat, serta konteks eksekusi yang dinamis, sehingga menurunkan efektivitas pendekatan pengujian berbasis aturan (*rule-based testing*) yang bersifat statis. Kondisi ini menimbulkan persoalan skalabilitas dan adaptabilitas yang mempersulit pemeliharaan pengujian sekaligus membatasi cakupan deteksi kesalahan, khususnya pada tingkat pengujian unit. Karena itu, sejumlah penelitian menekankan pentingnya generasi kasus uji otomatis dan prioritisasi kasus uji (*test case prioritization*, TCP) untuk meningkatkan efisiensi pengujian sekaligus menekan eksekusi redundan pada sistem berskala besar yang terus berevolusi [1], [2].

Kemajuan *deep learning* menunjukkan kemampuan yang kuat dalam memodelkan pola perilaku kompleks pada lingkungan perangkat lunak berdimensi tinggi, sehingga membuka peluang strategi pengujian yang lebih adaptif dan berbasis data. Keberhasilan pendekatan berbasis pembelajaran mesin dan pembelajaran mendalam dalam menangani permasalahan dunia nyata yang kompleks telah ditunjukkan pada beragam domain, *mulai dari visualisasi analitik untuk prioritisasi kasus uji berbasis machine learning*, [3] *personalisasi berbasis machine learning pada sistem kompleks* [4], *hingga pengujian keamanan berbasis deep reinforcement learning pada perangkat IoT* [5]. Dalam konteks ini, Reinforcement Learning (RL) menonjol karena karakteristiknya selaras dengan prioritisasi pengujian, yaitu pengambilan keputusan berurutan (*sequential decision-making*) di bawah ketidakpastian dengan umpan balik yang tertunda: agen mempelajari kebijakan optimal melalui interaksi langsung dengan lingkungan, bukan melalui aturan yang ditetapkan sebelumnya [6]. RL telah terbukti efektif pada beragam persoalan optimasi stokastik, kendali adaptif, serta sistem yang mengoptimasi diri (*self-optimizing*), dan adopsi *deep reinforcement learning* terus meningkat pada domain penjadwalan, optimasi, dan sistem pendukung Keputusan [7], [8]. Kesesuaian struktural inilah yang menjadikan RL kandidat menjanjikan untuk mengotomatiskan siklus pengujian unit yang adaptif.

Dalam ranah pengujian perangkat lunak, RL mulai dieksplorasi untuk otomatisasi pengujian, sebagai contoh, kerangka kerja PyTester memanfaatkan RL untuk menghasilkan kasus uji secara adaptif dari spesifikasi tekstual [9]. Di luar pengujian unit, RL juga telah dieksplorasi untuk pengujian berorientasi keamanan, seperti *penetration testing* pada perangkat IoT dan jaringan kendaraan (VANET), yang menegaskan keluwesan paradigma ini di beragam domain pengujian perangkat lunak,[10] [11]. Namun, sebagian besar studi yang ada masih berfokus pada generasi kasus uji secara terpisah dan belum mengintegrasikan siklus pengujian unit adaptif yang secara eksplisit menautkan cakupan kode dengan deteksi kesalahan pada aplikasi web nyata [12], [13], [14]. Tiga celah metodologis masih menonjol. Pertama, perbandingan yang adil antaralgoritma *deep* RL khususnya metode berbasis nilai (DQN, DDQN) versus berbasis kebijakan (PPO) dalam satu kerangka eksperimen yang seragam masih jarang dilakukan. Kedua, fungsi reward umumnya berfokus tunggal pada cakupan kode dan mengabaikan lokasi kesalahan [15], sehingga diperlukan *reward* multiobjektif yang menyeimbangkan peningkatan cakupan dan keberhasilan deteksi bug. Ketiga, banyak studi belum menyertakan *baseline* non-pembelajaran yang eksplisit maupun uji signifikansi statistik dengan pengulangan seed berganda [16], [17], sehingga reproduktibilitas dan validitas kesimpulannya masih terbatas.

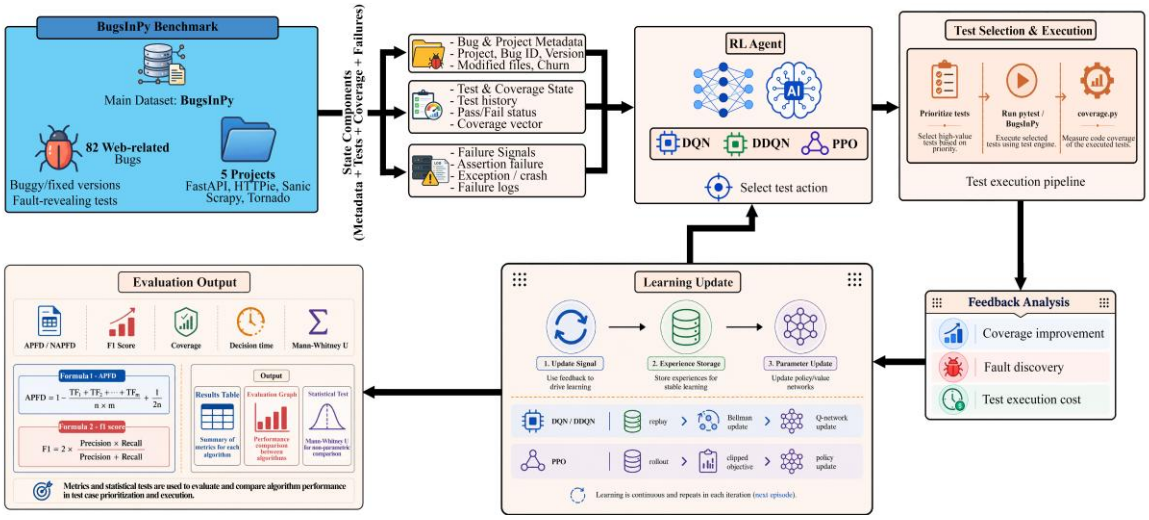
Untuk menjawab celah tersebut, penelitian ini mengusulkan kerangka kerja pengujian unit otomatis berbasis RL yang adaptif untuk aplikasi berbasis web. Prioritisasi kasus uji diformulasikan sebagai *Markov Decision Process* (MDP) dengan fungsi reward multiobjektif yang memadukan peningkatan cakupan kode dan keberhasilan deteksi kesalahan, sehingga agen mengarahkan eksplorasi ke kasus uji yang paling informatif. Kerangka ini dievaluasi pada dua lingkungan yang saling melengkapi: (i) aplikasi web nyata berbasis Flask yang dilengkapi kesalahan tersemai (*seeded faults*) sebagai System *Under Test* terkendali, dan (ii) *benchmark bug* Python BugsInPy sebagai validasi eksternal untuk menguji generalisasi pada bug dunia nyata. Pada lingkungan terkendali, metode RL berbasis nilai khususnya DDQN mengungguli *baseline* dalam efektivitas deteksi kesalahan (APFD dan NAPFD ≈ 0,85; keunggulan terhadap baseline bersifat deskriptif (selisih absolut besar, baseline tidak memiliki replikasi seed), sedangkan perbedaan antar-algoritma RL signifikan pada lima seed independen) sekaligus mempertahankan efisiensi komputasi yang lebih baik dibandingkan PPO, evaluasi pada BugsInPy menegaskan arah keunggulan yang sama pada bug nyata, meskipun dengan margin yang lebih kecil. Kerangka kerja ini juga dirancang agar kompatibel dengan alur *continuous integration/continuous deployment* (CI/CD) untuk mendukung pengujian berkelanjutan.

Kontribusi utama penelitian ini dirumuskan sebagai berikut. *Pertama*, pengujian unit otomatis diformulasikan sebagai persoalan prioritisasi kasus uji dalam kerangka *Markov Decision Process*, lengkap dengan definisi *state*, *action*, dan *reward* multiobjektif berbasis cakupan kode serta deteksi kesalahan. *Kedua*, dikembangkan *System Under Test* berupa aplikasi web nyata berbasis Flask beserta modul pemantauan (*web analytics*) sebagai sumber umpan balik pengujian, dan dimanfaatkan *benchmark* bug Python BugsInPy sebagai validasi eksternal, sehingga pengukuran cakupan kode dan deteksi bug bersifat objektif dan dapat direproduksi. *Ketiga*, dilakukan perbandingan tiga algoritma *Deep Reinforcement Learning* (DQN, DDQN, dan PPO) terhadap dua *baseline* non-pembelajaran (*Random* dan *Greedy*) menggunakan lima *seed* independen

dan uji signifikansi statistik (*Mann–Whitney U*). *Keempat*, dianalisis secara kritis *trade-off* antara cakupan kode, kecepatan deteksi bug, dan efisiensi komputasi termasuk temuan bahwa strategi berbasis cakupan semata (*Greedy*) memperoleh cakupan tertinggi namun deteksi kesalahan terburuk yang menghasilkan panduan pemilihan algoritma untuk praktik pengujian berkelanjutan.

II. Metode

Untuk menggambarkan alur kerja dan integrasi komponen sistem yang diusulkan, dirancang suatu arsitektur operasional berbasis *Reinforcement Learning* yang memformulasikan pengujian unit otomatis sebagai prioritisasi kasus uji adaptif. Arsitektur ini menghubungkan lingkungan pengujian (aplikasi web *under test* dan *benchmark* bug), representasi *state*, agen RL pengambil keputusan, eksekusi pengujian, serta modul umpan balik yang mengukur cakupan kode, deteksi kesalahan, dan biaya eksekusi. Gambar 1 menunjukkan interaksi antarkomponen tersebut selama proses *automated testing* berlangsung.



Gambar 1. Arsitektur kerangka kerja pengujian unit otomatis berbasis Reinforcement Learning yang diusulkan, dari sumber data (Flask SUT dan BugsInPy) hingga evaluasi (APFD/NAPFD/F1) dalam kerangka Markov Decision Process.

Tugas prioritisasi pengujian dimodelkan sebagai *Markov Decision Process* (MDP) yang dinyatakan oleh tuple (S, A, P, R, γ), dengan transisi (s_t, a_t, r_t, s_{t+1}). Pada setiap langkah, lingkungan menyajikan himpunan kandidat kasus uji C_t yang belum dieksekusi, agen memilih satu kandidat untuk dijalankan, lalu menerima *reward* berdasarkan cakupan kode dan kesalahan yang terungkap. Tujuan agen adalah memaksimalkan ekspektasi imbalan *discounted*:

$$J(\pi) = \mathbb{E}_\pi \left[\sum_{t=0}^{T} \gamma^t \, r(s_t, a_t) \right] \tag{1}$$

dengan $\gamma \in (0,1]$ faktor diskon dan T anggaran pengujian (*test budget*). Berbeda dengan formulasi berbasis kendala, model ini tidak memerlukan *safety cost* eksplisit karena biaya eksekusi telah diintegrasikan langsung sebagai komponen *reward*.

State. State s_t menggabungkan progres pengujian global (fraksi anggaran terpakai dan cakupan kumulatif) dengan vektor fitur setiap kandidat. Untuk aplikasi web Flask, fitur kandidat mencakup identitas *endpoint*, waktu respons, ukuran cakupan saat ini, dan potensi cakupan baru, untuk BugsInPy, fitur mencakup *churn*, jumlah berkas termodifikasi, jumlah *hunk*, jumlah *test*, dan potensi cakupan baru. Action. Aksi $a_t \in C_t$ memilih kasus uji (atau *request* HTTP) berikutnya untuk dieksekusi. Karena kandidat berkurang seiring eksekusi, ruang aksi bersifat dinamis. Reward multiobjektif. *Reward* memadukan peningkatan cakupan, deteksi kesalahan, dan biaya eksekusi:

$$r_t = \lambda^{k_t} (w_{\text{cov}} \Delta\text{Cov}_t + w_{\text{fault}} 1[\text{fault}_t]) - w_{\text{cost}} \text{cost}_t \tag{2}$$

dimana ΔCov_t pertambahan cakupan kode, $1[\text{fault}_t]$ indikator kasus uji mengungkap kesalahan (respons HTTP 5xx pada Flask SUT atau *fault-revealing test* pada BugsInPy), dan $cost_t$ biaya eksekusi (waktu respons). Bobot ditetapkan $w_{\text{cov}} = 0{,}7$, $w_{\text{fault}} = 3{,}0$, $w_{\text{cost}} = 0{,}1$, dengan faktor peluruhan posisi $\lambda = 0{,}97$ dan k_t urutan eksekusi. Peluruhan λ^{k_t} memberi bobot lebih besar pada deteksi dini sehingga selaras dengan metrik APFD, sementara $w_{\text{fault}} \gg w_{\text{cov}}$ mengarahkan agen memprioritaskan penemuan kesalahan di atas cakupan semata. Kedua agen melakukan aproksimasi fungsi nilai aksi $Q_\theta(s, a)$ melalui jaringan penilai (scoring network) atas fitur kandidat, dilatih secara off-policy menggunakan experience replay buffer D dan target network berparameter θ^- yang diperbarui berkala. Pada DQN, target temporal difference dan fungsi kerugian adalah:

$$y_t = r_t + \gamma \max_{a' \in \mathcal{C}_{t+1}} Q_{\theta^-}(s_{t+1}, a'), \qquad L(\theta) = \mathbb{E}_{(s,a,r,s') \sim \mathcal{D}}[(Q_\theta(s, a) - y_t)^2] \tag{3}$$

DDQN mengurangi bias *overestimation* dengan memisahkan pemilihan dan evaluasi aksi: aksi maksimum dipilih oleh jaringan daring θ, tetapi dievaluasi oleh target network θ^-:

$$y_t^{\text{DDQN}} = r_t + \gamma Q_{\theta^-}\left(s_{t+1}, \arg\max_{a' \in \mathcal{C}_{t+1}} Q_\theta(s_{t+1}, a')\right) \tag{4}$$

Eksplorasi dikendalikan melalui kebijakan ε-greedy: dengan probabilitas ε agen memilih kandidat secara acak, selebihnya memilih $a_t = \arg\max_{a \in \mathcal{C}_t} Q_\theta(s_t, a)$. PPO mengoptimalkan kebijakan π_θ secara *on-policy* dengan arsitektur *actor–critic*. Dengan rasio probabilitas $\rho_t(\theta) = \pi_\theta(a_t \mid s_t)/\pi_{\theta_{\text{old}}}(a_t \mid s_t)$ dan estimasi *advantage* $\hat{A}_t$, fungsi objektif (*clipped surrogate*) adalah:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t\left[\min\left(\rho_t(\theta)\hat{A}_t, \text{clip}(\rho_t(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_t\right)\right] \tag{5}$$

Objektif total menggabungkan kerugian kebijakan, kerugian nilai, dan bonus entropi:

$$L(\theta) = -L^{\text{CLIP}}(\theta) + c_v \mathbb{E}_t[(V_\theta(s_t) - R_t)^2] - c_e \mathbb{E}_t\left[H\left(\pi_\theta(\cdot \mid s_t)\right)\right] \tag{6}$$

diman $\epsilon = 0{,}2$, koefisien nilai $c_v = 0{,}5$, dan koefisien entropi $c_e = 0{,}02$. Sebagai pembanding non-pembelajaran digunakan Random (memilih kandidat secara seragam) dan Greedy (additional-coverage, memilih kandidat dengan tambahan cakupan terbesar) [18]. Performa diukur dengan APFD, NAPFD (APFD ternormalisasi untuk anggaran terbatas), F1 deteksi kesalahan, dan waktu keputusan (*decision time*) per siklus:

$$APFD = 1 - \frac{\sum_{i=1}^{m} TF_i}{n\,m} + \frac{1}{2n}, NAPFD = p - \frac{\sum_{i=1}^{m} TF_i}{n\,m} + \frac{p}{2n} \tag{7}$$

A. Dataset

Penelitian ini menggunakan dua sumber data yang saling melengkapi. Pertama, sebuah aplikasi web nyata berbasis Flask (*mini-shop e-commerce*) dikembangkan sebagai System *Under Test* (SUT). Aplikasi ini terdiri atas dua belas endpoint HTTP mencakup penelusuran produk, pencarian, keranjang, *checkout*, pemesanan, rekomendasi, dan diskon dan tujuh di antaranya sengaja menyimpan bug terdokumentasi (B1–B7) yang memunculkan respons HTTP 5xx pada masukan tertentu, sehingga berfungsi sebagai *ground-truth* deteksi kesalahan [19]. Ketujuh bug tersebut merepresentasikan kelas kegagalan yang beragam dan realistis *KeyError*, *AttributeError*, *ValueError*, *ZeroDivisionError*, dan *IndexError* yang lazim ditemukan pada aplikasi web nyata. Kedua, *benchmark BugsInPy* digunakan sebagai validasi eksternal. BugsInPy memuat 501 *bug* nyata dari 17 proyek Python sumber terbuka, penelitian ini difokuskan pada lima proyek berbasis web (*FastAPI*, *Sanic*, Tornado, *HTTPie*, dan *Scrapy*) yang mencakup 82 *bug* web beserta test pengungkap kesalahan (*fault-revealing test*) dan *patch* perbaikannya. Setiap *bug* menyediakan versi *buggy* dan *fixed*, sehingga pengukuran cakupan kode (melalui coverage.py) dan efektivitas deteksi *bug* bersifat

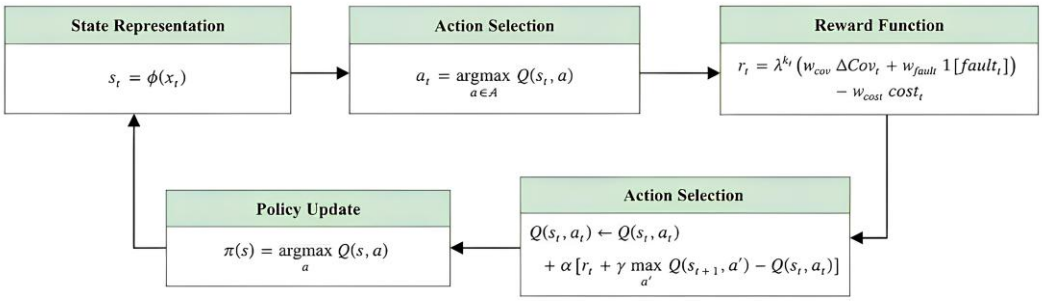
objektif serta dapat direproduksi. Kombinasi SUT terkendali dan *benchmark bug* nyata ini selaras secara metodologis dengan tujuan *automated* unit testing pada pemrograman berbasis web. Contoh data *BugsInPy* yang digunakan ditampilkan pada Tabel 1.

Tabel 1. Sampel data dataset BugsInPy yang digunakan pada pengujian.

No	Proyek	ID Bug	Berkas Kode Termodifikasi	Fault-Revealing Test	Churn (LOC)	Versi Python
1	FastAPI	1	fastapi/applications.py	tests/test_jsonable_encoder.py	223	3.8.3
2	FastAPI	2	fastapi/routing.py	tests/test_ws_router.py	7	3.8.3
3	HTTPie	1	httpie/downloads.py	tests/test_downloads.py	36	3.7.3
4	HTTPie	2	httpie/client.py	tests/test_redirects.py	2	3.7.3
5	Sanic	1	sanic/app.py	tests/test_blueprints.py	2	3.8.3
6	Sanic	2	sanic/server.py	tests/test_app.py	20	3.8.3
7	Tornado	1	tornado/websocket.py	tornado/test/websocket_test.py	11	3.7.0
8	Tornado	2	tornado/http1connection.py	tornado/test/httpclient_test.py	5	3.7.0
9	Scrapy	1	scrapy/spidermiddlewares/offsite.py	tests/test_spidermiddleware_offsite.py	8	3.8.3
10	Scrapy	2	scrapy/utils/datatypes.py	tests/test_utils_datatypes.py	5	3.8.3

B. *Reinforcement learning model*

Gambar 2 menyajikan siklus pembelajaran *Reinforcement Learning* yang digunakan dalam penelitian ini. Proses berlangsung iteratif: agen mengamati *state* lingkungan, memilih aksi berdasarkan kebijakan, menerima *reward* sebagai umpan balik, lalu memperbarui estimasi nilai Q (yang mencerminkan keuntungan jangka panjang pasangan *state–action*) dan kebijakannya. Siklus ini diulang hingga agen memperoleh strategi prioritisasi pengujian yang stabil dalam memaksimalkan *reward* kumulatif. Gambar 2 mengilustrasikan prinsip pembelajaran berbasis nilai (DQN/DDQN), PPO mengikuti siklus serupa tetapi memperbarui kebijakan secara langsung melalui objektif terpotong berbasis gradien.



Gambar 2. Alur kerja model RL berbasis Q-Learning untuk pengujian unit otomatis.

Gambar 2 menyajikan siklus pembelajaran *Reinforcement Learning* yang digunakan dalam penelitian ini dalam kerangka *Markov Decision Process*. Pada setiap langkah, agen mengamati *state* $s_t = \phi(x_t)$ yang merangkum progres pengujian dan fitur tiap kandidat kasus uji, lalu memilih aksi $a_t = \arg\max_a Q(s_t, a)$ untuk menentukan kasus uji berikutnya. Lingkungan mengembalikan *reward multiobjektif* yang memadukan penambahan cakupan dan keberhasilan deteksi bug serta dikurangi biaya eksekusi, dengan faktor peluruhan posisi λ^{k_t} sehingga deteksi dini dihargai lebih tinggi. Berdasarkan *reward* tersebut, agen memperbarui estimasi nilai Q dan menyempurnakan kebijakan π, siklus ini berulang hingga strategi prioritisasi pengujian menjadi stabil. Gambar 2 mengilustrasikan pembaruan berbasis nilai (DQN/DDQN), sedangkan PPO memperbarui kebijakan secara langsung melalui objektif terpotong berbasis gradien.

1. *Formulasi Prioritisasi Pengujian dan Algoritma Pembelajaran*

Proses pengujian diformulasikan sebagai prioritisasi kasus uji dalam kerangka *Markov Decision Process* (MDP). Pada setiap langkah, agen mengamati *state* berupa vektor fitur yang merangkum progres pengujian (fraksi kasus uji yang telah dieksekusi, *code coverage* berjalan, dan fraksi bug yang telah ditemukan) beserta atribut tiap kandidat kasus uji (*endpoint*, ukuran perubahan kode, waktu eksekusi, dan potensi penambahan *coverage*)[20].*Action* berupa pemilihan kasus uji berikutnya untuk dieksekusi. *Reward* didefinisikan sebagai kombinasi berbobot antara penambahan *coverage* dan keberhasilan pengungkapan bug, dikurangi biaya eksekusi, dengan faktor peluruhan posisi sehingga deteksi dini bernilai lebih tinggi. Tiga algoritma dibandingkan: *Deep Q-Network* (DQN) yang mempelajari fungsi nilai aksi melalui *experience replay* dan *target network*, *Double DQN* (DDQN) yang memisahkan pemilihan dan evaluasi aksi untuk mengurangi bias *overestimation* [21], serta *Proximal Policy Optimization* (PPO) yang

mengoptimalkan kebijakan stokastik melalui objektif terpotong (*clipped surrogate*) [22]. Kebijakan diwakili jaringan saraf penilai per-kandidat sehingga mendukung ruang aksi berukuran variabel.

2. Protokol Eksperimen dan Evaluasi

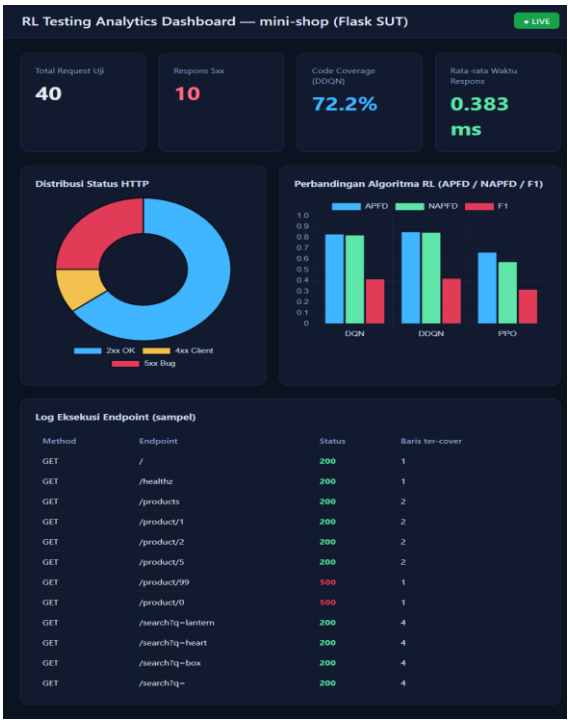
Sistem diuji pada dua lingkungan komplementer. Lingkungan utama adalah aplikasi web Flask nyata yang berperan sebagai *System Under Test* dengan tujuh bug tersemai terdokumentasi, *code coverage* diukur secara dinamis menggunakan instrumentasi coverage.py dan status HTTP direkam untuk setiap *request*. Lingkungan kedua adalah *benchmark* BugsInPy padanan Defects4J untuk ekosistem *Python* yang menyediakan bug nyata beserta *test* pengungkap kesalahan sebagai validasi eksternal. Setiap algoritma dilatih dan dievaluasi menggunakan lima *seed* independen untuk mengukur variabilitas, dengan evaluasi sebanyak 300 siklus pengujian pada anggaran 50% dari *suite*. Empat metrik dilaporkan: *Average Percentage of Faults Detected* (APFD) beserta versinya yang dinormalisasi terhadap anggaran (NAPFD), *F1-score* deteksi *bug* (dengan *Precision* = TP/b dan *Recall* = TP/m, di mana TP adalah jumlah *fault-revealing test case* yang terdeteksi dalam anggaran b = ⌊0,5 × B⌋, dan m adalah jumlah total *fault-revealing test case* dalam *suite*; F1 bernilai nol apabila tidak ada *bug* terdeteksi dalam anggaran), dan latensi pengambilan keputusan agen. Perbedaan antar algoritma diuji signifikansinya menggunakan uji *Mann Whitney* U dua sisi, dan seluruh eksperimen dapat direproduksi melalui penetapan *seed* acak yang tetap.

III. Hasil dan Pembahasan

Bagian ini menyajikan hasil eksperimen dari kerangka kerja *automated* unit testing berbasis *Reinforcement Learning* yang diusulkan. Evaluasi difokuskan pada empat aspek terukur: (i) konvergensi dan stabilitas pembelajaran antar algoritma, (ii) efektivitas deteksi kesalahan melalui APFD, NAPFD, dan F1, (iii) *trade-off* antara cakupan kode dan kecepatan deteksi bug, serta (iv) efisiensi komputasi yang diukur melalui latensi keputusan. Hasil dilaporkan pada dua lingkungan: aplikasi web *Flask* sebagai System *Under Test* terkendali dan *benchmark BugsInPy* sebagai validasi eksternal.

A. Observasi dan Deskripsi Sistem Web

Untuk mendukung eksperimen *Reinforcement Learning*, dikembangkan aplikasi web nyata berbasis *Flask* yang berperan sebagai *System Under Test* sekaligus dilengkapi modul *web analytics*. Aplikasi menyediakan dua belas *endpoint* HTTP (daftar produk, pencarian, keranjang, *checkout*, pemesanan, rekomendasi, diskon, dan autentikasi), tujuh di antaranya menyimpan bug tersemai yang memunculkan galat HTTP 5xx pada masukan tertentu sebagai *ground-truth* deteksi. Selama pengujian, agen RL mengirimkan *request* ke *endpoint*, mengamati status HTTP dan *code coverage*, serta memvisualisasikan hasil secara *real-time* melalui *dashboard* analitik. Gambar 3 menyajikan *dashboard* tersebut.

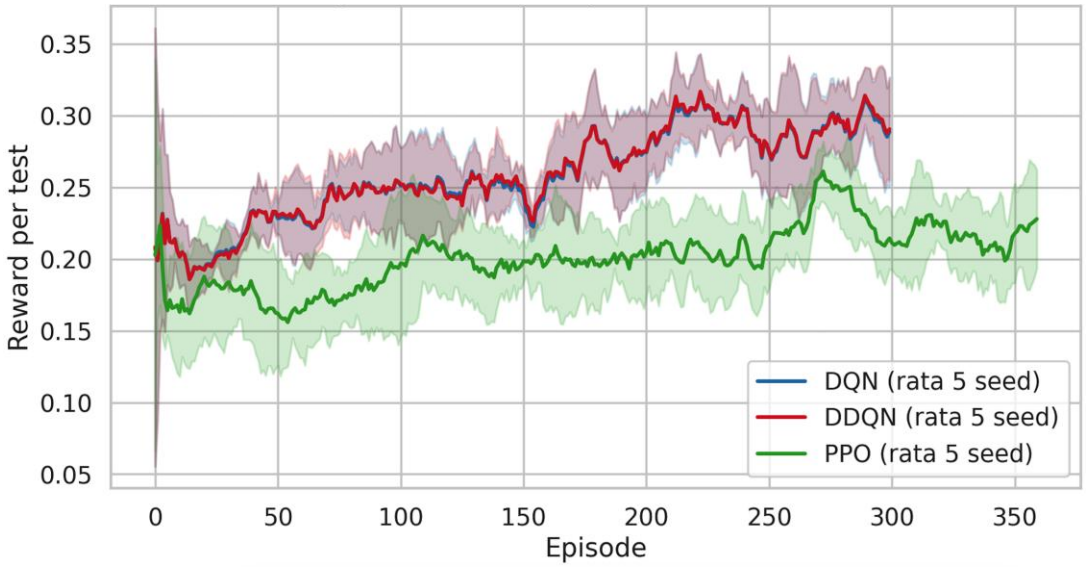


Gambar 3. Dashboard analitik pengujian berbasis RL pada aplikasi web (SUT): distribusi status HTTP, code coverage, jumlah respons pemicu-bug, dan waktu respons.

Gambar 3 menampilkan *dashboard* analitik yang memvisualisasikan interaksi agen *Reinforcement Learning* dengan aplikasi secara *real-time*. *Dashboard* menyajikan indikator utama (total *request* uji, jumlah respons 5xx, *code coverage*, dan rata-rata waktu respons HTTP), distribusi status HTTP (2xx, 4xx, dan 5xx), serta perbandingan capaian antar algoritma. Katalog pengujian terdiri atas 40 *request* pada beragam *endpoint*, sepuluh *request* di antaranya memicu galat HTTP 5xx yang berasal dari tujuh bug tersemai sebagai *ground-truth* deteksi. Cakupan kode yang dicapai kebijakan terbaik (DDQN) mencapai 72,2%. Mekanisme visualisasi *real-time* ini mendukung pemantauan dan interpretasi proses pengujian.

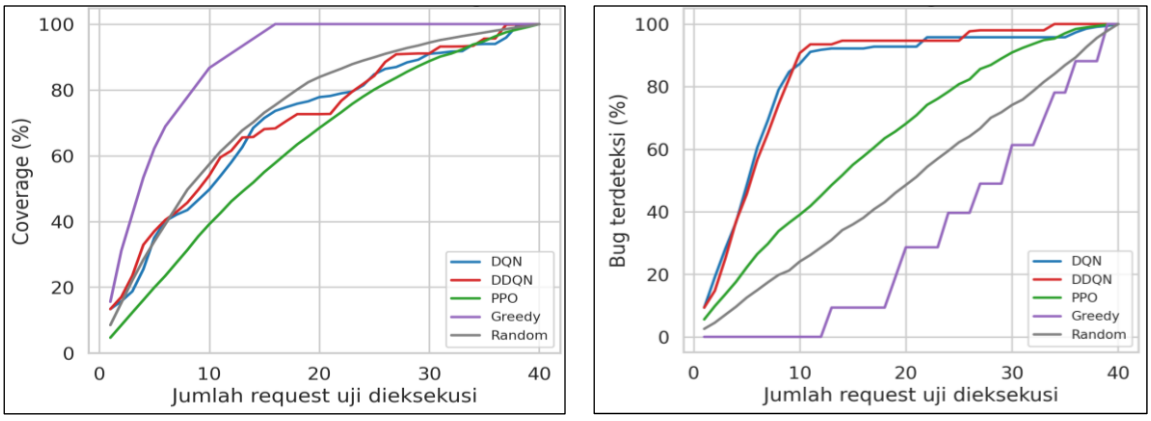
B. Hasil Analisis Eksperimen

Untuk mengevaluasi dinamika pembelajaran, dilakukan analisis konvergensi *reward per test* selama pelatihan pada algoritma *Deep Q-Network* (DQN), *Double Deep Q-Network* (DDQN), dan *Proximal Policy Optimization* (PPO). Gambar 4 digunakan untuk mengamati kestabilan pembelajaran dan efektivitas masing-masing algoritma dalam memaksimalkan *reward* seiring bertambahnya episode.



Gambar 4. Konvergensi reward per test selama pelatihan (rata-rata 5 seed) untuk DQN, DDQN, dan PPO.

Gambar 4 menunjukkan konvergensi *reward per test* selama pelatihan. Ketiga algoritma meningkat dari fase eksplorasi awal menuju kebijakan yang stabil. Pada fase konvergen, metode berbasis nilai memperoleh *reward per test* yang lebih tinggi, yaitu 0,3010 untuk DDQN dan 0,2991 untuk DQN, dibandingkan 0,2314 untuk PPO. Pita bayangan merepresentasikan simpangan baku antar-*seed*, yang menandakan stabilitas proses pelatihan. Pola ini menunjukkan bahwa agen berbasis nilai mempelajari strategi prioritisasi pengujian secara lebih efisien.



(a) (b)

Gambar 5. Perbandingan kinerja metode pengujian berdasarkan (a) pertumbuhan code coverage dan (b) deteksi bug kumulatif.

Gambar 5 menyajikan dinamika (a) pertumbuhan *code coverage* dan (b) deteksi bug kumulatif terhadap jumlah *request* uji yang dieksekusi. Pada sepuluh *request* pertama, DDQN telah menemukan 0,9072 dari seluruh *bug* dan DQN 0,8728, sedangkan PPO mencapai 0,3905 dan *baseline* Random 0,2407. Sebaliknya, *baseline* Greedy tidak menemukan satu pun *bug* pada sepuluh *request* pertama meskipun mencapai *code coverage* keseluruhan tertinggi (0,89, Gambar 5a). Kontras ini menegaskan bahwa maksimalisasi cakupan kode semata tidak berbanding lurus dengan deteksi kesalahan dini: *request* yang memaksimalkan cakupan baris kode bukanlah *request* yang paling mungkin memicu kegagalan. Metode berbasis nilai (DDQN dan DQN) mengungguli seluruh pembanding dalam kecepatan deteksi, membuktikan kemampuan agen memprioritaskan *request* bernilai diagnostik tinggi.

Untuk membandingkan performa kuantitatif seluruh algoritma dan baseline secara menyeluruh, Tabel 2 merangkum metrik akhir (APFD, NAPFD, Coverage, F1, dan latensi keputusan) beserta hasil uji signifikansi statistik Mann–Whitney U.

Tabel 2. Perbandingan kuantitatif algoritma RL dan *baseline* pada SUT *Flask* (*mean* ± *std,* 5 *seed*).

Algoritma	APFD	NAPFD	Coverage	F1	Latensi Keputusan(ms)
DQN	0,8293 ± 0,0131	0,8203 ± 0,0117	0,718 ± 0,031	0,4119 ± 0,0047	3,0339 ± 0,1108
DDQN	0,8504 ± 0,0080	0,8447 ± 0,0131	0,722 ± 0,013	0,4183 ± 0,0023	3,0033 ± 0,0868
PPO	0,6603 ± 0,0212	0,5715 ± 0,0361	0,650 ± 0,021	0,3164 ± 0,0135	6,2668 ± 0,1103
Greedy	0,3199 ± 0,0979	0,1655 ± 0,1167	0,890 ± 0,000	0,1282 ± 0,0827	0,1775 ± 0,0156
Random	0,4922 ± 0,1407	0,3632 ± 0,1865	0,741 ± 0,034	0,2130 ± 0,1250	0,2911 ± 0,0265

Tabel 2 menunjukkan bahwa metode berbasis nilai memperoleh performa terbaik pada seluruh metrik deteksi. DDQN mencapai APFD 0,8504, NAPFD 0,8447, dan F1 0,4183, lebih tinggi daripada DQN (APFD 0,8293, NAPFD 0,8203, F1 0,4119). Relatif terhadap *baseline*, NAPFD DDQN lebih tinggi 132,58% dibandingkan *Random* dan 410,38% dibandingkan *Greedy*, sedangkan F1 DDQN lebih tinggi 96,37% dibandingkan *Random* dan 226,20% dibandingkan *Greedy.* Untuk perbandingan antar algoritma RL, uji Mann–Whitney U dua sisi dilakukan pada unit analisis yang benar-benar independen, yaitu rata-rata metrik per seed (n = 5 per algoritma): DDQN vs PPO pada F1 ($p = 0,0079$), DDQN vs DQN pada F1 ($p = 0,0317$), DDQN vs PPO pada NAPFD ($p = 0,0079$), DDQN vs DQN pada NAPFD ($p = 0,0317$), dan DDQN vs PPO pada Coverage ($p = 0,0079$); seluruhnya signifikan ($p < 0,05$). Keunggulan RL terhadap baseline Greedy dan Random bersifat deskriptif (selisih absolut sangat besar: NAPFD DDQN = 0,8447 vs Random = 0,3632), mengingat baseline tidak memiliki replikasi seed. PPO memperoleh APFD 0,6603, NAPFD 0,5715, dan F1 0,3164 lebih rendah daripada metode berbasis nilai serta menuntut latensi keputusan 6,2668 ms, yaitu 2,0867 kali latensi DDQN (3,0033 ms) dengan perbedaan yang signifikan ($p = 0,0079$)

Baseline Greedy memperoleh *code coverage* tertinggi (0,89) tetapi NAPFD terendah (0,1655) dan F1 terendah (0,1282), karena strategi berbasis cakupan semata mengabaikan lokasi kesalahan. Temuan ini menegaskan adanya *trade-off* antara cakupan kode dan deteksi bug, sekaligus keunggulan *Reinforcement Learning* khususnya DDQN dalam menyeimbangkan keduanya. Selain lebih unggul, metode berbasis nilai juga lebih stabil antar-*seed*: simpangan baku F1 DDQN sebesar 0,0023, jauh lebih kecil daripada simpangan baku F1 Random sebesar 0,1250, yang mencerminkan ketidakstabilan strategi non-pembelajaran.

C. Validasi External pada BugsInPy

Untuk menguji generalisasi kerangka kerja pada *bug* dunia nyata, seluruh algoritma dievaluasi ulang pada 82 *bug* web BugsInPy dengan protokol yang sama (lima *seed*, anggaran 50%). Tabel 3 merangkum hasilnya.

Tabel 3. Perbandingan kuantitatif pada benchmark BugsInPy (mean ± std, 5 seed).

Algoritma	APFD	NAPFD	F1	Coverage	Latensi (ms)
DQN	0,5167 ± 0,0097	0,3714 ± 0,0184	0,4419 ± 0,0146	0,6477 ± 0,0267	1,1087 ± 0,0411
DDQN	0,5032 ± 0,0087	0,3541 ± 0,0235	0,4242 ± 0,0258	0,6207 ± 0,0309	1,0701 ± 0,0408
PPO	0,5095 ± 0,0110	0,3645 ± 0,0180	0,4314 ± 0,0169	0,6302 ± 0,0266	2,2611 ± 0,0649
Greedy	0,4583 ± 0,1571	0,2992 ± 0,2006	0,3620 ± 0,2284	0,6318 ± 0,0697	0,0733 ± 0,0776
Random	0,4929 ± 0,1391	0,3337 ± 0,2035	0,3957 ± 0,2197	0,6151 ± 0,0680	0,1141 ± 0,1029

Pada BugsInPy, ketiga algoritma RL tetap mengungguli *baseline* dalam efektivitas deteksi (NAPFD DQN 0,3714 dan F1 DQN 0,4419 versus *Random* 0,3337/0,3957 dan *Greedy* 0,2992/0,3620), sehingga arah keunggulan konsisten dengan hasil pada SUT terkendali. Namun, margin keunggulan jauh lebih kecil dan variabilitas *baseline* lebih tinggi, mencerminkan kompleksitas bug nyata. Uji *Mann Whitney* U menunjukkan keunggulan RL atas *baseline* tetap signifikan pada F1 (DDQN vs *Greedy*, $p = 1,28 \times 10^{-6}$, DDQN vs

Random, $p = 3{,}0 \times 10^{-3}$); catatan: unit analisis di sini adalah 82 versi bug independen—bukan replikasi seed—sehingga uji ini valid secara statistik, berbeda dengan SUT Flask di mana baseline tidak memiliki replikasi seed (perbandingan deskriptif). Perbedaan antar varian RL tidak signifikan (DDQN vs PPO pada F1, $p = 0{,}618$). Berbeda dengan SUT yang mengukur cakupan pernyataan secara dinamis cakupan pada BugsInPy diukur pada granularitas berkas, dan setiap versi hanya memuat satu kesalahan, sehingga sinyal *reward* lebih jarang. Kondisi ini menjelaskan margin yang lebih sempit dan menegaskan bahwa keunggulan RL bersifat umum namun bergantung pada kekayaan sinyal umpan balik.

C. *Justifikasi Pemilihan Metode*

Pemilihan *Reinforcement Learning* didasarkan pada karakteristik prioritisasi pengujian sebagai masalah pengambilan keputusan berurutan di bawah ketidakpastian dengan *reward* tertunda kondisi yang tidak tertangani optimal oleh heuristik statis. Hasil empiris memperkuat justifikasi ini secara terukur: RL mengungguli *baseline Random* dan Greedy yang merepresentasikan strategi non-pembelajaran (NAPFD 0,8447 berbanding 0,3632 dan 0,1655; perbandingan bersifat deskriptif pada SUT Flask—mengingat baseline tidak memiliki replikasi seed—dan signifikan secara statistik pada BugsInPy dengan unit analisis 82 bug independen). Dibandingkanmetode berbasis pencarian (misalnya algoritma genetika) yang mengoptimasi ulang setiap program dari awal, kebijakan RL bersifat adaptif dan dapat digeneralisasi antar-siklus tanpa optimasi ulang [23], [24], dibandingkan pembelajaran tersupervisi, RL tidak memerlukan label urutan optimal yang sulit diperoleh. Perbandingan antarvarian RL memberi *trade-off* praktis: metode berbasis nilai unggul dalam kecepatan (latensi 3,0033 ms untuk DDQN berbanding 6,2668 ms untuk PPO) dan akurasi deteksi, sedangkan PPO menawarkan eksplorasi yang lebih luas dengan biaya komputasi lebih tinggi.

D. *Ancaman terhadap Validitas*

Beberapa keterbatasan perlu diperhatikan. Pertama, aktivasi bug per siklus dimodelkan sebagai skenario regresi (subset bug aktif) menggunakan profil pengujian yang diukur secara nyata, walaupun test pengungkap kesalahan dan patch berasal dari sistem nyata, dinamika kemunculan bug antar-versi merupakan pendekatan (construct validity). Kedua, code coverage pada validasi BugsInPy diukur pada granularitas berkas, sedangkan cakupan dinamis penuh diukur pada SUT Flask melalui coverage.py, perbedaan granularitas ini turut menjelaskan margin keunggulan yang lebih kecil pada BugsInPy. Ketiga, latensi keputusan yang dilaporkan diukur sebagai waktu inferensi kebijakan agen per siklus. Keempat, generalisasi ke aplikasi web berskala industri dan lingkungan multi-pengguna masih perlu divalidasi lebih lanjut [25]. Keterbatasan ini menjadi arah pengembangan penelitian selanjutnya. Kelima, uji Mann–Whitney U dilakukan pada unit analisis rata-rata per seed ($n = 5$, independen) untuk perbandingan antar algoritma RL, sehingga menghindari inflasi signifikansi akibat korelasi antar siklus dalam run yang sama; perbandingan RL dengan baseline bersifat deskriptif karena baseline tidak memiliki replikasi seed.

IV. Kesimpulan dan saran

Berdasarkan hasil eksperimen, penerapan *Reinforcement Learning* pada *automated unit testing* aplikasi web terbukti meningkatkan efektivitas pengujian secara terukur. Pada *System Under Test* terkendali berbasis *Flask*, metode berbasis nilai khususnya DDQN memperoleh performa terbaik dengan APFD 0,8504, NAPFD 0,8447, dan F1 0,4183, mengungguli *baseline Random* dan Greedy secara deskriptif, dengan selisih absolut sangat besar (NAPFD DDQN = 0,8447 berbanding Random = 0,3632 dan Greedy = 0,1655)DDQN juga lebih efisien secara komputasi, dengan latensi keputusan 3,0033 ms atau kurang dari separuh latensi PPO (6,2668 ms). Sebaliknya, *baseline* Greedy mencapai *code coverage* tertinggi (0,8900) namun memperoleh NAPFD terendah (0,1655) dan tidak menemukan satu pun bug pada sepuluh *request* pertama, menegaskan bahwa cakupan kode bukan indikator yang memadai bagi kualitas pengujian. Validasi eksternal pada *benchmark* bug nyata BugsInPy menunjukkan arah keunggulan yang konsisten metode RL tetap mengungguli *baseline* dalam deteksi kesalahan meskipun dengan margin yang lebih kecil, yang mengindikasikan bahwa efektivitas pendekatan bergantung pada kekayaan sinyal umpan balik. Selain itu, integrasi *dashboard* analitik web mendukung pemantauan dan interpretasi proses pengujian secara *real-time*. [25] Untuk penelitian selanjutnya, pengembangan metode dapat diarahkan pada algoritma *Deep Reinforcement Learning* yang lebih canggih seperti A3C, Rainbow DQN, atau varian *actor–critic* untuk ruang aksi diskret guna menyeimbangkan stabilitas pembelajaran dan efektivitas eksplorasi pada lingkungan pengujian yang lebih kompleks dan non-stasioner [26]. Pengembangan *reward function* yang lebih kontekstual dengan mempertimbangkan tingkat keparahan kesalahan, prioritas jalur eksekusi kritis, tingkat kompleksitas *endpoint*, serta konsumsi sumber daya sistem berpotensi menghasilkan strategi pengujian yang lebih optimal dan efisien. Selain itu, penguatan evaluasi pada BugsInPy melalui jumlah bug dan episode yang lebih besar serta representasi *state* yang lebih kaya diperlukan untuk mempersempit kesenjangan performa terhadap SUT terkendali. Evaluasi lebih lanjut pada beragam *framework* aplikasi web, lingkungan *multi-user*, dan skenario pengujian *real-time* juga diperlukan

untuk mengukur kemampuan generalisasi model secara lebih komprehensif. Terakhir, integrasi *transfer learning*, *continual learning*, serta *pipeline* CI/CD berpotensi meningkatkan kemampuan adaptasi agen RL terhadap perubahan sistem secara berkelanjutan pada praktik pengembangan perangkat lunak modern.

Daftar Pustaka

[1] T. Chojnacki and L. Madeyski, "Test case prioritization: A systematic review using snowballing and TCPFramework with approach combinators," *Inf. Softw. Technol.*, vol. 194, p. 108062, 2026, doi: 10.1016/j.infsof.2026.108062.

[2] Q. Zhang, C. Fang, W. Sun, S. Yu, Y. Xu, and Y. Liu, "Test case prioritization using partial attention," *J. Syst. Softw.*, vol. 192, p. 111419, 2022, doi: 10.1016/j.jss.2022.111419.

[3] J. A. Silveira, L. Vieira, and N. Ferreira, "TPVis: A visual analytics system for exploring test case prioritization methods," *Comput. & Graph.*, vol. 124, p. 104064, 2024, doi: 10.1016/j.cag.2024.104064.

[4] M. Faisal *et al.*, "Assessing AI-Driven Personalization in Smart Cities Using Hybrid Machine Learning and MCDM Approach," *HighTech Innov. J.*, vol. 6, no. 3, pp. 770–792, Sep. 2025, doi: 10.28991/HIJ-2025-06-03-03.

[5] F. Pagano, M. Ceccato, A. Merlo, and P. Tonella, "Multi-agent deep reinforcement learning for penetration testing of IoT devices through their mobile companion app," *J. Syst. Softw.*, vol. 239, p. 112918, 2026, doi: 10.1016/j.jss.2026.112918.

[6] A. Abo-eleneen, A. Palliyali, and C. Catal, "The role of Reinforcement Learning in software testing," *Inf. Softw. Technol.*, vol. 164, p. 107325, 2023, doi: 10.1016/j.infsof.2023.107325.

[7] A. P. Marugán, "Applications of Reinforcement Learning for maintenance of engineering systems: A review," *Adv. Eng. Softw.*, vol. 183, p. 103487, 2023, doi: 10.1016/j.advengsoft.2023.103487.

[8] A. Seyyedabbasi, "A reinforcement learning-based metaheuristic algorithm for solving global optimization problems," *Adv. Eng. Softw.*, vol. 178, p. 103411, 2023, doi: 10.1016/j.advengsoft.2023.103411.

[9] W. Takerngsaksiri, R. Charakorn, C. Tantithamthavorn, and Y.-F. Li, "Pytester: Deep reinforcement learning for text-to-testcase generation," *J. Syst. Softw.*, vol. 224, p. 112381, 2025, doi: 10.1016/j.jss.2025.112381.

[10] M. R. Haris *et al.*, "Leveraging a Hybrid Fuzzy C-Means-PCA Model for Identifying Speech Therapy Needs in Children," *J. Data Sci. Intell. Syst.*, Jun. 2026, doi: 10.47852/bonviewJDSIS62028311.

[11] P. Garrad and S. Unnikrishnan, "Reinforcement learning in VANET penetration testing," *Results Eng.*, vol. 17, p. 100970, 2023, doi: 10.1016/j.rineng.2023.100970.

[12] Y. Ding, Y. Zhang, G. Yuan, S. Jiang, W. Dai, and Y. Zhang, "Integration test order generation based on reinforcement learning considering class importance," *J. Syst. Softw.*, vol. 205, p. 111823, 2023, doi: 10.1016/j.jss.2023.111823.

[13] Z. Qian, Q. Yu, H. Zhu, J. Liu, and T. Fu, "Reinforcement learning for test case prioritization based on LLEed K-means clustering and dynamic priority factor," *Inf. Softw. Technol.*, vol. 179, p. 107654, 2025, doi: 10.1016/j.infsof.2024.107654.

[14] T. Shu, C. Wu, and Z. Ding, "Boosting input data sequences generation for testing EFSM-specified systems using deep reinforcement learning," *Inf. Softw. Technol.*, vol. 155, p. 107114, 2023, doi: 10.1016/j.infsof.2022.107114.

[15] X. Wang and S. Zhang, "Cluster-based adaptive test case prioritization," *Inf. Softw. Technol.*, vol. 165, p. 107339, 2024, doi: 10.1016/j.infsof.2023.107339.

[16] E. A. da Roza, J. A. do Prado Lima, and S. R. Vergilio, "On the use of contextual information for machine learning based test case prioritization in continuous integration development," *Inf. Softw. Technol.*, vol. 171, p. 107444, 2024, doi: 10.1016/j.infsof.2024.107444.

[17] W. D. F. Mendonça, W. K. G. Assunção, and S. R. Vergilio, "Feature-oriented test case selection and prioritization during the evolution of highly-configurable systems," *J. Syst. Softw.*, vol. 217, p. 112157, 2024, doi: 10.1016/j.jss.2024.112157.

[18] A. Alwadaeen and Z. Alzamil, "Optimized greedy algorithm for efficient test case

prioritization," *Sci. Comput. Program.*, vol. 253, p. 103489, 2026, doi: 10.1016/j.scico.2026.103489.

[19] S. Alagarsamy, C. Tantithamthavorn, and A. Aleti, "A3Test: Assertion-Augmented Automated Test case generation," *Inf. Softw. Technol.*, vol. 176, p. 107565, 2024, doi: 10.1016/j.infsof.2024.107565.

[20] Y. Shi, B. Yin, and J.-A. Shi, "Markov model based coverage testing of deep learning software systems," *Inf. Softw. Technol.*, vol. 179, p. 107628, 2025, doi: 10.1016/j.infsof.2024.107628.

[21] M. Waltz and O. Okhrin, "Addressing maximization bias in reinforcement learning with two-sample testing," *Artif. Intell.*, vol. 336, p. 104204, 2024, doi: 10.1016/j.artint.2024.104204.

[22] L. Chavali, A. Krishnan, P. Saxena, B. Mitra, and A. Sreevallabh Chivukula, "Off-policy actor-critic deep reinforcement learning methods for alert prioritization in intrusion detection systems," *Comput. & Secur.*, vol. 142, p. 103854, 2024, doi: 10.1016/j.cose.2024.103854.

[23] N. Cardozo and I. Dusparic, "Auto-COP: Adaptation generation in Context-oriented Programming using Reinforcement Learning options," *Inf. Softw. Technol.*, vol. 164, p. 107308, 2023, doi: 10.1016/j.infsof.2023.107308.

[24] M. Zhao, R. Hou, H. Li, and M. Ren, "A hybrid grey wolf optimizer using opposition-based learning, sine cosine algorithm and reinforcement learning for reliable scheduling and resource allocation," *J. Syst. Softw.*, vol. 205, p. 111801, 2023, doi: 10.1016/j.jss.2023.111801.

[25] A. Sunba, J. Hassine, and M. Ahmed, "Testing reinforcement learning systems: A comprehensive review," *J. Syst. Softw.*, vol. 231, p. 112563, 2026, doi: 10.1016/j.jss.2025.112563.

[26] T. K. A. Choudhury, M. Behera, S. K. Dash, S. K. Pani, and J. Mishra, "AnoLSTM—A Deep Learning Approach for Test Cases Prioritization," in *Procedia Computer Science*, Elsevier BV, 2025, pp. 1793–1803. doi: 10.1016/j.procs.2025.04.431.