Prioritisasi Kasus Uji *Advantage Actor-Critic* untuk Deteksi Dini Cacat Operasional Layanan *AI*

Advantage Actor-Critic-based Test Case Prioritization for Early Detection of Operational Defects in AI Services

[1]Eko Agustina*, [2]Lulu Chaerani Munggaran
[1,2]Program Studi Magister Manajemen Sistem Informasi, Fakultas Teknologi dan Rekayasa, Universitas Gunadarma
[1,2]Jl. Margonda Raya No.100, Pondok Cina, Beji, Kota Depok, Jawa Barat, Indonesia
*e-mail: eko.agustina86@gmail.com

(***received*:** 1 September 2026, ***revised*:** 5 October 2026, ***accepted*:** 7 October 2026)

Abstrak

Layanan *artificial intelligence* berbasis API dapat mengalami *semantic defect* dan *runtime failure* ketika beroperasi pada kondisi *high-load*, sehingga diperlukan strategi prioritisasi kasus uji yang mampu mendukung deteksi *operational defect* sedini mungkin. Penelitian ini bertujuan menganalisis efektivitas *adaptive test case prioritization* berbasis Advantage Actor-Critic (A2C) pada layanan AI dalam kondisi *high-load*. Penelitian menggunakan layanan *intent classification* berbasis DistilBERT dan ELECTRA-small dengan *test suite* dari Banking77. A2C dibandingkan dengan *random prioritization* dan *performance-based prioritization* menggunakan *full test suite prioritization* selama sepuluh siklus dan lima *run*, sehingga menghasilkan 300 unit eksperimen. Efektivitas dievaluasi menggunakan Average Percentage of Faults Detected (APFD), Test to First Failure (TTF), dan *Response Time*. Pada DistilBERT, A2C memperoleh rata-rata APFD tertinggi sebesar 0,883816 dan TTF terendah sebesar 1,000, sedangkan pada ELECTRA-small A2C memperoleh rata-rata APFD tertinggi sebesar 0,674268 dengan TTF sebesar 1,000, yang identik dengan *performance-based prioritization*. *Performance-based prioritization* menghasilkan *Response Time* terendah pada kedua model, yaitu 3.760,845 ms pada DistilBERT dan 5.984,625 ms pada ELECTRA-small. Pengujian *Friedman* menunjukkan perbedaan signifikan pada ketiga metrik untuk kedua model ($p < 0{,}001$). Hasil *Wilcoxon signed-rank* dengan koreksi Holm menunjukkan bahwa sebagian besar pasangan metode berbeda signifikan, dengan pengecualian TTF A2C terhadap *performance-based prioritization* pada ELECTRA-small yang identik, serta *Response Time* A2C terhadap *random prioritization* yang tidak berbeda signifikan. Hasil ini menunjukkan bahwa keunggulan A2C terutama terletak pada efektivitas pembentukan urutan prioritas untuk mendeteksi *operational defect* lebih awal, bukan pada pencapaian *Response Time* terendah.

Kata kunci: *high-load testing, operational defect, reinforcement learning, software testing, test case prioritization*

Abstract

API-based artificial intelligence (AI) services may experience semantic defects and runtime failures under high-load conditions, necessitating test case prioritization strategies that support the early detection of operational defects. This study aims to evaluate the effectiveness of adaptive test case prioritization using Advantage Actor-Critic (A2C) for AI services under high-load conditions. The experiments were conducted using intent classification services based on DistilBERT and ELECTRA-small, with a test suite derived from Banking77. A2C was compared with random prioritization and performance-based prioritization using full test suite prioritization over ten cycles and five runs, yielding 300 experimental units. Effectiveness was evaluated using Average Percentage of Faults Detected (APFD), Test to First Failure (TTF), and response time. For DistilBERT, A2C achieved the highest mean APFD of 0.883816 and the lowest TTF of 1.000. For ELECTRA-small, A2C achieved the highest mean APFD of 0.674268, with a TTF of 1.000, identical to that of performance-based prioritization. Performance-based prioritization yielded the lowest response times for both models: 3,760.845 ms for DistilBERT and 5,984.625 ms for ELECTRA-small. The Friedman tests indicated statistically significant differences across the three metrics for both models ([test statistics and p-

 ISSN:2302-8149
e-ISSN:2540-9719

values to be inserted]). Wilcoxon signed-rank tests with Holm correction showed that most method pairs differed significantly. Exceptions included TTF for A2C versus performance-based prioritization on ELECTRA-small, which produced identical results, and response time for A2C versus random prioritization, which did not differ significantly. These findings indicate that A2C's primary advantage lies in its ability to generate effective test case prioritization sequences for earlier operational defect detection rather than achieving the lowest response time.
Keywords: *high-load testing, operational defect, reinforcement learning, software testing, test case prioritization*

1 Pendahuluan

Pemanfaatan *Artificial Intelligence* (AI) dalam layanan digital terus berkembang untuk mendukung berbagai fungsi, seperti prediksi, klasifikasi, rekomendasi, dan pemrosesan bahasa alami. Kapabilitas tersebut umumnya diimplementasikan melalui *Application Programming Interface* (API) agar model AI dapat diakses dan diintegrasikan dengan sistem lain secara modular [1]. Karakteristik layanan berbasis *machine learning* berbeda dari perangkat lunak konvensional karena perilakunya tidak hanya ditentukan oleh logika program, tetapi juga dipengaruhi oleh data, model, proses pelatihan, serta kondisi operasional ketika sistem dijalankan [2]. Kondisi tersebut menyebabkan kualitas layanan AI tidak cukup dinilai hanya berdasarkan performa prediksi pada tahap pengembangan model. Evaluasi juga perlu memperhatikan kemampuan layanan dalam menerima *request*, menghasilkan *response* yang sesuai, mempertahankan kinerja, dan menangani kegagalan yang dapat muncul selama proses operasional [3].

Tantangan pengujian menjadi semakin kompleks ketika layanan AI dijalankan pada kondisi *high-load*. Peningkatan tekanan terhadap sumber daya dapat memperbesar latensi, memperlambat pemrosesan, dan meningkatkan kemungkinan terjadinya *timeout* maupun kegagalan layanan selama eksekusi [4]. Dalam penelitian ini, permasalahan yang muncul selama operasional direpresentasikan melalui *operational defect* yang dibatasi menjadi *semantic defect* dan *runtime failure*. *Semantic defect* menunjukkan ketidaksesuaian antara keluaran semantik yang dihasilkan layanan dengan keluaran yang seharusnya, sedangkan *runtime failure* berkaitan dengan ketidakmampuan layanan menyelesaikan pemrosesan *request* secara valid, seperti akibat *timeout*, *error response*, atau bentuk kegagalan eksekusi lainnya [5]. Deteksi terhadap kedua bentuk cacat tersebut perlu dilakukan sedini mungkin agar informasi mengenai ketidaksesuaian perilaku layanan dapat diperoleh pada bagian awal proses pengujian, terutama ketika seluruh *test case* harus dieksekusi pada kondisi beban tinggi.

Kebutuhan deteksi dini tersebut dapat didukung melalui *Test Case Prioritization* (TCP), yaitu mekanisme pengurutan *test case* berdasarkan tingkat prioritas tertentu sehingga kandidat yang dinilai lebih penting dapat dieksekusi lebih awal [6]. Pendekatan ini tidak harus mengurangi jumlah *test case*, tetapi dapat digunakan untuk menentukan urutan eksekusi terhadap keseluruhan *test suite*. Pendekatan prioritisasi statis, seperti *random prioritization* maupun prioritisasi yang hanya menggunakan informasi historis tertentu, memiliki mekanisme yang relatif sederhana, tetapi tidak secara langsung menyesuaikan strategi berdasarkan umpan balik terbaru dari lingkungan pengujian. Perubahan pola kegagalan, karakteristik waktu respons, dan kondisi eksekusi dapat menyebabkan prioritas yang dibentuk berdasarkan kondisi sebelumnya kehilangan relevansi ketika lingkungan berubah [7]. Mekanisme prioritisasi yang mampu memperbarui strategi berdasarkan pengalaman pengujian terbaru karena itu diperlukan untuk mendukung pengujian pada lingkungan yang bersifat dinamis.

Reinforcement learning menyediakan mekanisme pembelajaran yang memungkinkan strategi prioritisasi diperbarui berdasarkan hasil eksekusi sebelumnya. Formulasi TCP menggunakan *reinforcement learning* telah dikembangkan melalui pendekatan *pointwise*, *pairwise*, dan *listwise ranking*, yang menunjukkan bahwa informasi hasil eksekusi dapat dimanfaatkan untuk membentuk urutan prioritas secara adaptif [6]. Riwayat eksekusi juga diketahui menjadi sumber informasi penting dalam pembentukan prioritas karena mampu merepresentasikan perilaku *test case* berdasarkan pengalaman pengujian sebelumnya [8]. Salah satu algoritma yang relevan dalam konteks tersebut adalah *Advantage Actor-Critic* (A2C), yang menggabungkan *actor* untuk menghasilkan keputusan berdasarkan *policy* dan *critic* untuk mengevaluasi keputusan sebagai dasar pembaruan strategi. Perbandingan beberapa algoritma *reinforcement learning* pada tugas pengujian perangkat lunak

menunjukkan bahwa A2C memiliki karakteristik yang kompetitif untuk TCP serta *prediction time* yang relatif rendah dibandingkan beberapa algoritma *actor-critic* lainnya [9].

Meskipun penerapan *reinforcement learning* pada TCP terus berkembang, penelitian sebelumnya masih didominasi konteks *continuous integration*, *regression testing*, *test suite minimization*, dan pengujian perangkat lunak konvensional. Evaluasi *adaptive test case prioritization* berbasis A2C pada layanan AI berbasis API dalam kondisi *high-load* masih terbatas, terutama dalam kaitannya dengan kualitas operasional selama *runtime*. Oleh karena itu, efektivitas A2C dalam memprioritaskan *test case* untuk mendeteksi *operational defect* lebih awal pada kondisi beban tinggi, dibandingkan dengan prioritisasi statis, masih perlu dikaji. Meskipun penelitian ATRL-TCP dan pendekatan berbasis histori menunjukkan potensi strategi adaptif dan informasi eksekusi dalam meningkatkan prioritisasi, konteksnya belum secara khusus mencakup layanan AI berbeban tinggi [10].

Meskipun reinforcement learning telah menunjukkan potensi dalam *test case prioritization*, penelitian terdahulu masih banyak berfokus pada konteks *regression testing* dalam lingkungan *Continuous Integration*. Pendekatan *test case prioritization* berbasis reinforcement learning telah dikembangkan melalui berbagai strategi *ranking*, seperti *pointwise*, *pairwise*, dan *listwise*, dengan memanfaatkan data historis eksekusi untuk menentukan prioritas pengujian [6]. Selain itu, mekanisme *adaptive reward* berbasis *sliding window* telah diperkenalkan untuk menyesuaikan pemanfaatan informasi historis terhadap kondisi pengujian yang berubah [11]. Penggunaan berbagai algoritma dan *framework deep reinforcement learning*, termasuk Advantage Actor-Critic (A2C), juga telah dievaluasi dalam penerapan reinforcement learning untuk *software testing [9]*. Dengan demikian, penelitian terdahulu telah memberikan landasan metodologis yang kuat bagi pengembangan *adaptive test case prioritization*, tetapi penerapan dan evaluasi mekanisme tersebut pada layanan *artificial intelligence* berbasis API, khususnya dalam kondisi *high-load*, masih belum banyak dikaji.

Penelitian ini memperluas konteks RL-based test case prioritization tersebut dengan menempatkan layanan *intent classification* berbasis model AI sebagai *system under test* pada *high-load environment*. Kontribusi penelitian tidak terletak pada pengenalan algoritma A2C atau formulasi dasar RL-TCP yang telah dikembangkan dalam penelitian sebelumnya, tetapi pada penerapan mekanisme *adaptive prioritization* tersebut untuk mendukung deteksi dini *operational defect* pada layanan AI. *Operational defect* dalam penelitian ini mencakup *semantic defect* dan *runtime failure*, sehingga evaluasi tidak hanya mempertimbangkan kemampuan ranking dalam menempatkan *test case* yang mengungkap permasalahan lebih awal melalui APFD dan TTF, tetapi juga mengevaluasi responsivitas layanan melalui *Response Time*. Untuk memperkuat evaluasi perbedaan antar-metode, penelitian ini juga menggunakan *Friedman test* yang dilanjutkan dengan *Wilcoxon signed-rank test* dan koreksi Holm pada unit analisis berpasangan**.** Selain itu, penelitian menerapkan *full test suite prioritization* sehingga seluruh *test case* tetap dieksekusi dan perbedaan antar-metode hanya berasal dari urutan eksekusi yang dihasilkan. Untuk memperluas validasi empiris, pendekatan yang sama diterapkan pada dua model *intent classification*, yaitu DistilBERT dan ELECTRA-small, dengan *test suite* dan prosedur eksperimen yang sama. Penggunaan dua model memungkinkan penelitian menilai apakah efektivitas prioritisasi A2C tetap terlihat ketika karakteristik model AI yang menjadi *system under test* berubah. Dengan karakteristik tersebut, penelitian ini memberikan bukti empiris mengenai relevansi *adaptive* A2C-based *test case prioritization* ketika objek pengujian merupakan layanan AI yang memiliki perilaku prediksi sekaligus karakteristik operasional yang dipengaruhi oleh tekanan beban.

Penelitian ini bertujuan menganalisis efektivitas *adaptive test case prioritization* berbasis Advantage Actor-Critic dalam mendukung deteksi dini *operational defect* pada layanan AI dalam kondisi *high-load*. A2C dibandingkan dengan *random prioritization* dan *performance-based prioritization* melalui *full test suite prioritization*, sehingga seluruh *test case* tetap dieksekusi dan perbedaannya terletak pada urutan pengujian. Efektivitas metode diukur menggunakan Average Percentage of Faults Detected (APFD) dan Test to First Failure (TTF), sedangkan *Response Time* digunakan untuk mengevaluasi responsivitas layanan selama pengujian. Perbedaan kinerja antar-metode selanjutnya diuji secara statistik untuk menentukan apakah perbedaan yang diperoleh bersifat signifikan.

 ISSN:2302-8149
e-ISSN:2540-9719

2 Tinjauan Literatur

Perkembangan *test case prioritization* berbasis pembelajaran menunjukkan pergeseran dari strategi yang mengandalkan aturan dan histori secara statis menuju pendekatan yang mampu menyesuaikan prioritas berdasarkan perubahan data pengujian. Informasi histori eksekusi memiliki peran penting dalam proses tersebut karena memberikan indikasi mengenai perilaku *test case* pada siklus sebelumnya. Evaluasi terhadap kumpulan fitur yang komprehensif menunjukkan bahwa fitur histori eksekusi memberikan pengaruh terbesar terhadap efektivitas TCP, sedangkan penggunaan fitur yang memerlukan analisis kode dapat menambah biaya pengumpulan data. Model berbasis *machine learning* juga perlu diperbarui secara berkala karena efektivitasnya dapat menurun ketika karakteristik sistem berubah [9]. Kebutuhan pembaruan tersebut menjadi salah satu alasan penggunaan *reinforcement learning*, karena agen dapat menerima informasi baru secara bertahap dan memperbarui strategi prioritisasinya tanpa selalu membangun model kembali dari awal [6]. Temuan-temuan tersebut menunjukkan bahwa kemampuan memanfaatkan histori sekaligus beradaptasi terhadap hasil eksekusi terbaru menjadi aspek penting dalam pengembangan TCP untuk lingkungan pengujian yang dinamis.

Penerapan *reinforcement learning* pada TCP kemudian berkembang melalui pemodelan proses prioritisasi sebagai permasalahan pemeringkatan. Bagherzadeh et al. mengevaluasi *pointwise*, *pairwise*, dan *listwise ranking* dengan berbagai algoritma RL serta menggunakan histori eksekusi yang diputar kembali untuk mendukung pembelajaran secara adaptif [6]. Hasilnya menunjukkan bahwa konfigurasi *pairwise ranking* secara umum memberikan akurasi yang lebih baik dibandingkan *pointwise* dan *listwise*, sedangkan konfigurasi *actor-critic* termasuk di antara pendekatan yang kompetitif dalam aspek akurasi maupun waktu prediksi. Kinerja tersebut tetap bergantung pada informasi yang tersedia karena data yang hanya memuat histori eksekusi tidak selalu menghasilkan kebijakan prioritisasi yang memadai, sementara penambahan fitur yang lebih kaya dapat menghasilkan ranking yang lebih dekat dengan urutan optimal [6]. Kondisi tersebut memperlihatkan bahwa efektivitas RL-TCP tidak hanya ditentukan oleh algoritma, tetapi juga oleh formulasi ranking dan kecukupan informasi yang membentuk pengalaman pembelajaran agen.

Aspek lain yang menentukan kualitas pembelajaran adalah mekanisme *reward* dan cakupan histori yang digunakan untuk membentuk umpan balik. Yang et al. mengembangkan *dynamic sliding window* agar informasi historis yang digunakan dalam perhitungan *reward* dapat menyesuaikan pola eksekusi, sehingga tidak hanya bergantung pada hasil terbaru maupun keseluruhan histori yang semakin panjang [11]. Evaluasi menggunakan NAPFD, *recall*, TTF, dan durasi menunjukkan bahwa *dynamic sliding window* dapat meningkatkan efektivitas prioritisasi pada sebagian besar kondisi yang diuji, dengan pendekatan berbasis masing-masing *test case* menghasilkan kinerja yang lebih baik dibandingkan penyesuaian jendela pada tingkat *test suite* dalam beberapa indikator [11]. Meskipun demikian, efektivitas mekanisme tersebut tetap dipengaruhi oleh karakteristik data, terutama ketika tingkat kegagalan rendah menyebabkan *reward* menjadi jarang dan menghambat proses pembelajaran. Keterbatasan ini memperlihatkan bahwa penggunaan histori tidak cukup dipandang dari jumlah data yang tersedia, tetapi perlu mempertimbangkan relevansi informasi terhadap perubahan perilaku *test case* serta kualitas sinyal evaluasi yang diterima agen.

Pemilihan algoritma dan *framework* implementasi turut memengaruhi hasil RL-TCP. Mindom et al. membandingkan beberapa algoritma DRL yang diimplementasikan melalui Stable-Baselines3, Tensorforce, dan Keras-rl pada tugas pengujian perangkat lunak, termasuk TCP [9]. Hasil eksperimen menunjukkan bahwa implementasi algoritma yang sama dapat menghasilkan performa yang berbeda ketika dijalankan menggunakan *framework* yang berbeda, sedangkan Stable-Baselines secara umum lebih sesuai untuk permasalahan TCP pada dataset yang dievaluasi [9]. A2C yang dikombinasikan dengan *pairwise ranking* melalui Stable-Baselines menghasilkan performa yang kompetitif dan mampu melampaui beberapa baseline RL yang digunakan dalam evaluasi, meskipun hasil tersebut tidak menunjukkan bahwa satu algoritma selalu menjadi pilihan terbaik pada seluruh konfigurasi dan dataset. Temuan tersebut penting karena pemilihan A2C perlu didasarkan pada kesesuaiannya terhadap mekanisme prioritisasi dan karakteristik lingkungan pengujian, bukan pada asumsi bahwa algoritma tersebut secara universal lebih unggul.

Penelitian yang lebih baru memperluas kemampuan adaptasi RL-TCP dengan menyesuaikan informasi maupun proses pembelajaran berdasarkan karakteristik data. ATRL-TCP mengatasi

 ISSN:2302-8149
e-ISSN:2540-9719

keterbatasan strategi yang hanya berfokus pada fitur individual atau hubungan relatif antartest case melalui mekanisme perpindahan perhatian berdasarkan kekayaan histori dan tingkat kesalahan [10]. Evaluasi pada empat dataset menunjukkan bahwa pendekatan tersebut mengungguli RETECS dan PRM serta memiliki kemampuan generalisasi yang baik, tetapi fungsi *reward* yang digunakan masih berorientasi pada posisi *test case* pendeteksi kesalahan dan belum mempertimbangkan waktu eksekusi secara langsung [10]. Perkembangan lain dilakukan melalui Meta-DRL yang menggabungkan *pairwise ranking* dengan A2C, DQN, PPO, dan TRPO serta menyesuaikan parameter pelatihan berdasarkan perkembangan performa agen [12]. Pendekatan tersebut dilaporkan mampu meningkatkan adaptabilitas sekaligus mengurangi waktu pelatihan sekitar 60% dibandingkan pendekatan pembanding dalam konteks CI [12]. Kedua penelitian tersebut memperlihatkan bahwa arah pengembangan RL-TCP saat ini semakin menekankan kemampuan adaptasi, baik terhadap karakteristik informasi pengujian maupun konfigurasi proses pembelajaran.

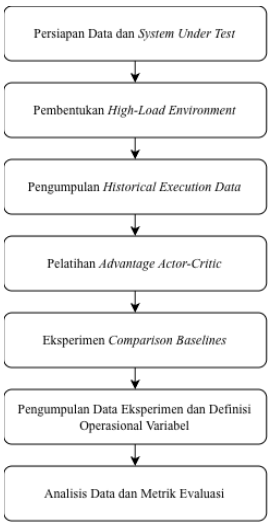
Sintesis penelitian tersebut menunjukkan bahwa RL telah memberikan dasar yang kuat bagi pembentukan TCP yang adaptif, tetapi ruang lingkup empiris penelitian utama yang ditinjau masih didominasi oleh regression testing dalam Continuous Integration [9]. Evaluasi yang dilakukan umumnya berorientasi pada kemampuan memprioritaskan *test case* yang mengalami kegagalan berdasarkan histori eksekusi dan informasi kode, dengan metrik seperti APFD, NRPA, NAPFD, TTF, *recall*, waktu pelatihan, atau waktu prediksi. Dalam studi-studi utama yang ditinjau, belum ditemukan evaluasi RL-TCP yang secara khusus menempatkan layanan AI berbasis API sebagai *system under test* pada kondisi *high-load* serta mengintegrasikan kesalahan keluaran semantik dan kegagalan selama runtime sebagai satu sasaran deteksi cacat operasional**.** Kesenjangan tersebut juga mencakup evaluasi dampak urutan prioritas terhadap responsivitas layanan, karena penelitian RL-TCP terdahulu lebih banyak menggunakan waktu sebagai biaya eksekusi atau *prediction/training time*, bukan sebagai perilaku operasional layanan yang sedang diuji. Batasan konteks ini membuka kebutuhan untuk menguji apakah kemampuan adaptif RL-TCP tetap relevan ketika objek pengujian memiliki perilaku prediksi sekaligus karakteristik operasional yang dipengaruhi oleh tekanan beban.

Berdasarkan celah penelitian tersebut, penelitian ini memposisikan *Advantage Actor-Critic* sebagai mekanisme *adaptive test case prioritization* untuk mendukung deteksi dini operational *defect* pada layanan AI dalam kondisi *high-load*. Pemilihan A2C didukung oleh hasil penelitian terdahulu yang menunjukkan bahwa algoritma *actor-critic* memiliki performa kompetitif dalam TCP, sedangkan A2C juga menunjukkan waktu prediksi yang relatif rendah dan performa prioritisasi yang baik pada konfigurasi tertentu [9]. Berbeda dari penelitian yang berorientasi pada CI regression testing atau pelaksanaan sebagian *test suite*, penelitian ini menerapkan *full test suite prioritization* dengan menggunakan histori eksekusi sebagai informasi pembelajaran dan mengevaluasi A2C terhadap *random prioritization* serta *performance-based prioritization*. Evaluasi diarahkan pada APFD dan TTF untuk mengukur kemampuan menempatkan *test case* yang mengungkap *semantic defect* maupun *runtime failure* lebih awal, sedangkan *Response Time* digunakan untuk menilai perilaku operasional layanan selama pengujian. Posisi tersebut memperluas kajian RL-TCP dari prioritisasi kegagalan pada *regression testing* menuju pengujian layanan AI yang mempertimbangkan deteksi dini cacat sekaligus kondisi operasional layanan.

3 Metode Penelitian

Penelitian ini menerapkan pendekatan eksperimental untuk mengevaluasi efektivitas *adaptive test case prioritization* berbasis *reinforcement learning* dengan algoritma Advantage Actor-Critic (A2C) dalam mendukung deteksi dini *operational defect* pada layanan AI pada kondisi *high-load*. Pelaksanaan penelitian dilakukan melalui serangkaian tahapan yang disusun secara sistematis dan berurutan untuk memastikan proses eksperimen berjalan secara terarah. Rangkaian tahapan penelitian yang digunakan ditunjukkan pada Gambar 1.

 ISSN:2302-8149
e-ISSN:2540-9719



Gambar 1 Tahapan penelitian

Gambar 1 menunjukkan alur pelaksanaan penelitian yang menjadi dasar dalam menjalankan seluruh proses eksperimen. Setiap tahapan dilaksanakan secara berurutan dan memiliki keterkaitan dengan tahapan berikutnya. Uraian mengenai masing-masing tahapan penelitian disajikan sebagai berikut:

a. Persiapan Data dan System Under Test

Dataset Banking77 dari repositori PolyAI/banking77 digunakan sebagai sumber data untuk *fine-tuning* model dan pembentukan *test suite*. Dataset dibagi menjadi 70% data *training*, 15% data *validation*, dan 15% data *test* dengan mengadaptasi skema pembagian dataset yang digunakan pada penelitian sebelumnya [13]. Data *training* digunakan untuk melakukan *fine-tuning* dua model *intent classification*, yaitu DistilBERT dan ELECTRA-small, sedangkan data *validation* digunakan untuk memantau proses pelatihan. Data *test* dipisahkan dari proses pelatihan dan digunakan sebagai sumber pembentukan *test suite*. Proses *fine-tuning* kedua model dilakukan menggunakan Python 3.11.9 dan Hugging Face Transformers pada perangkat dengan prosesor Apple M2 CPU 8-core, RAM 8 GB, dan SSD 256 GB.

Model DistilBERT dan ELECTRA-small hasil *fine-tuning* masing-masing diimplementasikan sebagai layanan AI berbasis API menggunakan FastAPI. Kedua layanan dijalankan secara terpisah dalam kontainer Docker pada home server Asus A450C dengan prosesor Intel Celeron 1007U 2-core, Intel HD Graphics, RAM 7 GB, dan penyimpanan 60 GB. Setiap endpoint layanan menerima masukan berupa teks, melakukan tokenisasi dan inferensi menggunakan model yang bersesuaian, kemudian mengembalikan *predicted_label_id* dan *predicted_label_name*. Setiap data pada subset test Banking77 dikonversi menjadi satu *test case* yang terdiri atas *test_id*, *input_request*, *expected_label_id*, dan *expected_label_name*. *Test suite* yang terbentuk dari subset test digunakan secara identik pada kedua layanan, sehingga perbedaan hasil eksperimen dapat diamati berdasarkan karakteristik model dan urutan eksekusi *test case*, bukan karena perbedaan data pengujian. Seluruh *test case* selanjutnya digunakan secara konsisten dalam pengumpulan data historis dan eksperimen komparatif pada kedua model.

b. Pembentukan High-Load Environment

Lingkungan high-load dibentuk dan divalidasi sebelum pengumpulan *historical execution data* serta eksperimen komparatif. Kalibrasi dilakukan secara terpisah pada masing-masing *system under test* menggunakan *environment-validation subset* yang dipisahkan dari *test suite* utama agar data pengujian utama tidak digunakan pada tahap penentuan beban. *Test runner* mengirimkan sejumlah request secara paralel kepada layanan dengan memvariasikan tingkat *concurrency* dan *timeout threshold*. Nilai *concurrency* digunakan untuk mengatur jumlah request yang diproses secara bersamaan, sedangkan *timeout threshold* menentukan batas waktu penerimaan respons sebelum eksekusi dikategorikan mengalami *timeout*.

 ISSN:2302-8149
e-ISSN:2540-9719

Setiap skenario kalibrasi dievaluasi berdasarkan *response time*, *runtime failure*, penggunaan CPU, dan penggunaan memori. Peningkatan *response time* digunakan untuk mengidentifikasi tekanan terhadap responsivitas layanan [4]. *Runtime failure* direpresentasikan melalui *timeout*, *error response*, atau respons tidak valid, sedangkan penggunaan CPU dan memori digunakan sebagai indikator tekanan sumber daya [14]. Konfigurasi dipertimbangkan sebagai kandidat *high-load* ketika layanan menunjukkan tekanan sumber daya dan peningkatan *response time*, tetapi masih mampu memproses sebagian besar request serta tidak mengalami kegagalan secara menyeluruh.

Kalibrasi dilakukan pada masing-masing *system under test* karena karakteristik operasional layanan dapat berbeda antar-model. Konfigurasi *high-load* yang telah ditetapkan untuk setiap model selanjutnya digunakan secara konsisten pada seluruh pengumpulan *historical execution data* dan eksperimen komparatif yang menggunakan model tersebut.

c. Pengumpulan Historical Execution Data

Historical execution data dikumpulkan secara terpisah untuk setiap *system under test* dengan mengeksekusi seluruh *test suite* utama sebanyak sepuluh siklus pada konfigurasi *high-load* yang telah ditetapkan. Setiap siklus merupakan satu kali eksekusi seluruh kandidat *test case*. Hasil eksekusi dicatat melalui *historical execution logger* yang terintegrasi dengan test runner dan basis data.

Data yang dicatat mencakup *model_name*, *cycle_id*, *test_case_id*, *input request*, *expected label*, *actual label*, *semantic_defect_status*, *runtime_failure_status*, *operational_defect_status*, *response time*, kode HTTP, dan waktu pencatatan. *Semantic defect* ditetapkan ketika hasil prediksi berbeda dari label yang diharapkan, sedangkan *runtime failure* mencakup *timeout*, *error response*, atau respons tidak valid. *Operational defect* bernilai positif apabila terjadi *semantic defect* dan/atau *runtime failure*.

Data historis kemudian digunakan untuk pembentukan *state* dan pelatihan *initial policy* A2C serta sebagai histori bagi *performance-based prioritization*. Sebaliknya, *random prioritization* tidak menggunakan data historis dalam pembentukan ranking. Data dari masing-masing *system under test* digunakan secara terpisah untuk menjaga kesesuaian pembelajaran dengan karakteristik layanan yang diuji.

d. Pelatihan Advantage Actor-Critic

Pelatihan A2C dilakukan secara offline pada masing-masing *system under test* menggunakan *historical execution replay environment* yang dibentuk dari data eksekusi historis masing-masing model. Proses pelatihan tidak menjalankan *test case* secara langsung pada layanan AI, tetapi memanfaatkan hasil eksekusi historis sebagai pengalaman pembelajaran untuk membentuk *state*, menghasilkan *action*, menentukan *optimal ranking*, menghitung *reward*, serta memperbarui jaringan *actor* dan *critic*. Pemanfaatan histori eksekusi ini mengadaptasi pendekatan *test case prioritization* berbasis *reinforcement learning* [6].

State merepresentasikan informasi historis setiap *test case* yang diberikan kepada jaringan *actor*. Komponen yang digunakan terdiri atas *verdict* (v), *execution time* (e), *execution history* (h), dan *age* (a), sehingga direpresentasikan melalui Persamaan (1).

$$s_t = [v, e, h, a] \quad (1)$$

Representasi *verdict* ditentukan berdasarkan nilai *operational_defect_status*, yaitu 1 apabila ditemukan *operational defect* dan 0 apabila tidak ditemukan. Selanjutnya, *execution time* direpresentasikan menggunakan rata-rata *response_time* berdasarkan histori eksekusi yang tersedia. *Execution history* direpresentasikan sebagai urutan nilai *operational_defect_status* dari siklus-siklus sebelum siklus acuan, sedangkan komponen *age* menunjukkan jumlah siklus yang telah dilalui sejak *test case* tersedia.

Jaringan *actor* menerima *state* dan menghasilkan *action* berupa skor prioritas kontinu. Rentang skor prioritas dinyatakan melalui Persamaan (2).

$$a_t \in (0,1] \quad (2)$$

Nilai *action* yang lebih tinggi menunjukkan tingkat prioritas yang lebih tinggi sehingga *test case* ditempatkan lebih awal dalam urutan eksekusi. Ketepatan skor tersebut dievaluasi menggunakan *optimal ranking* yang dibentuk dari hasil aktual pada siklus historis yang sedang diputar kembali. *Test case* yang mengungkap *operational defect* ditempatkan lebih awal, sedangkan *response_time* digunakan sebagai dasar pengurutan ketika beberapa *test case* memiliki status *operational defect* yang sama [6].

 ISSN:2302-8149
e-ISSN:2540-9719

Posisi masing-masing *test case* pada *optimal ranking* dinormalisasi ke rentang 0 sampai 1 dan digunakan sebagai acuan pembentukan *reward*. Semakin dekat skor prioritas yang dihasilkan *actor* terhadap posisi optimal yang telah dinormalisasi, semakin tinggi *reward* yang diberikan. Fungsi *reward* pada penelitian ini dinyatakan melalui Persamaan (3).

$$r = 1 - (norm(optimal_rank) - action)^2 \quad (3)$$

dengan (r) sebagai *reward*, (norm(optimal_rank)) sebagai posisi *test case* dalam *optimal ranking* yang telah dinormalisasi, dan (action) sebagai skor prioritas yang dihasilkan oleh *actor*.

A2C menggunakan *critic* untuk mengestimasi nilai *state* sebagai dasar evaluasi keputusan *actor*. *Reward* dan estimasi nilai *state* digunakan untuk menghasilkan *temporal-difference error* serta *advantage*. Parameter jaringan *actor* diperbarui agar kebijakan yang dihasilkan semakin sesuai dengan tujuan pembelajaran melalui Persamaan (4) [15].

$$\theta_{t+1} = \theta_t + \alpha A_t \nabla_\theta \ln \pi_\theta(a_t \mid s_t) \quad (4)$$

dengan θ menyatakan parameter jaringan *actor*, α menyatakan *actor learning rate*, A_t menyatakan nilai *advantage*, dan $\pi_\theta(a_t \mid s_t)$ menyatakan *policy* yang menghasilkan *action* berdasarkan *state*.

Jaringan *critic* diperbarui dengan meminimalkan perbedaan antara estimasi nilai *state* dan nilai target yang dibentuk dari *reward* serta estimasi *state* berikutnya. Fungsi *critic loss* dinyatakan melalui Persamaan (5) [15].

$$L_{\text{critic}}(\omega) = \frac{1}{2}[V_\omega(s_t) - (r_t + \gamma V_\omega(s_{t+1}))]^2 \quad (5)$$

dengan ω menyatakan parameter jaringan *critic*, r_t menyatakan *reward*, γ menyatakan *discount factor*, sedangkan $V_\omega(s_t)$ dan $V_\omega(s_{t+1})$ merupakan estimasi nilai *state* saat ini dan *state* berikutnya.

Konfigurasi jaringan dan *hyperparameter* ditetapkan sebelum pelatihan untuk menjaga konsistensi kondisi pembelajaran. Pemilihannya mengacu pada konfigurasi dan hasil *tuning* penelitian terdahulu yang relevan serta disesuaikan dengan lingkungan penelitian. Jaringan *actor* dan *critic* menggunakan satu *hidden layer* dengan 64 *hidden units* dan aktivasi Tanh, sesuai dengan konfigurasi yang digunakan pada penelitian terdahulu [16].

Optimizer Adam digunakan pada kedua jaringan dengan *learning rate* sebesar 0,0003 untuk *actor* dan 0,001 untuk *critic* [16]. *Discount factor* (γ) ditetapkan sebesar 0,9 dan *entropy coefficient* sebesar 0,01 untuk mendukung konvergensi dan eksplorasi *policy* [16] . Pelatihan dibatasi hingga 200 episode dengan mengadaptasi anggaran pembelajaran pada penelitian terdahulu untuk eksperimen *test case prioritization* [9] . Batas ini digunakan untuk menjaga anggaran pembelajaran tetap dan memungkinkan perbandingan yang konsisten. Pelatihan dihentikan lebih awal apabila reward tidak mengalami peningkatan selama 100 episode berturut-turut untuk menghentikan proses ketika tidak terdapat perbaikan yang terukur pada policy. Rincian konfigurasi pelatihan yang digunakan dalam penelitian ini ditampilkan pada Tabel 1.

Tabel 1 Konfigurasi pelatihan A2C

No	Hyperparameter	Nilai
1	Actor network architecture	1 hidden layer, 64 hidden units per layer
2	Critic network architecture	1 hidden layer, 64 hidden units per layer
3	Hidden activation function	TanH
4	Actor optimizer	Adam
5	Critic optimizer	Adam
6	Actor learning rate	0.0003
7	Critic learning rate	0.001
8	Discount factor(γ)	0.9
9	Entropy coefficient	0.01
10	Jumlah episode training	200
11	Kriteria penghentian training	Berhenti saat reward tidak membaik selama 100 episode berturut-turut

 ISSN:2302-8149
e-ISSN:2540-9719

Satu episode didefinisikan sebagai satu rangkaian pemrosesan seluruh kandidat *test case* yang berasal dari satu siklus historis. Setiap *test case* diproses untuk menghasilkan skor prioritas dan *reward*, sedangkan parameter *actor* dan *critic* diperbarui selama proses pembelajaran. Setelah seluruh kandidat pada satu episode selesai diproses, total *reward* dicatat dan *environment* di-*reset* untuk menjalankan episode berikutnya. Model hasil pelatihan kemudian disimpan sebagai *initial A2C model checkpoint* yang mencakup parameter *actor* dan *critic*, status *optimizer*, konfigurasi *hyperparameter*, parameter normalisasi *state*, *random seed*, dan jumlah episode yang telah dijalankan. *Checkpoint* yang sama digunakan sebagai kondisi awal pada setiap pengulangan eksperimen komparatif.

e. Eksperimen Comparison Baselines

Eksperimen utama membandingkan metode A2C dengan dua metode *static test case prioritization*, yaitu *random prioritization* dan *performance-based prioritization*. Eksperimen dilakukan secara terpisah pada masing-masing System Under Test (SUT), yaitu DistilBERT dan ELECTRA-small. Pada setiap SUT, ketiga metode menggunakan *test suite* yang sama dan konfigurasi *high-load environment* yang telah ditetapkan untuk SUT tersebut serta menerapkan *full test suite prioritization*, sehingga seluruh *test case* tetap dieksekusi. Dengan demikian, perbedaan antar-metode dalam satu SUT hanya terletak pada mekanisme pembentukan urutan prioritas. Eksperimen dilaksanakan selama sepuluh siklus dan diulang sebanyak lima kali dengan mengadaptasi rancangan penelitian sebelumnya [9]. Dengan tiga metode prioritisasi, diperoleh 150 unit eksperimen pada setiap SUT atau 300 unit eksperimen untuk kedua SUT.

Setiap kombinasi metode prioritisasi, siklus, dan pengulangan diperlakukan sebagai satu unit eksperimen. Khusus untuk A2C, setiap pengulangan pada masing-masing SUT dimulai dari initial model checkpoint yang sama untuk memastikan pembaruan policy pada satu pengulangan tidak memengaruhi pengulangan lainnya. Checkpoint A2C untuk DistilBERT dan ELECTRA-small digunakan secara terpisah sesuai hasil pelatihan masing-masing SUT. Konfigurasi high-load environment juga mengikuti hasil kalibrasi masing-masing SUT dan dipertahankan konsisten selama eksperimen pada SUT tersebut. Dengan rancangan ini, hasil evaluasi antar-metode dan antar-pengulangan dapat dibandingkan secara konsisten. Mekanisme pembentukan ranking yang digunakan oleh masing-masing metode prioritisasi disajikan pada Tabel 2.

Tabel 2 Mekanisme pembentukan ranking metode prioritisasi

Metode	Mekanisme Ranking	Adaptasi Selama Eksperimen
A2C	*Actor* menghasilkan skor setiap *test case*, kemudian skor diurutkan menurun	*Policy* diperbarui setelah hasil satu siklus tersedia
Random	Seluruh *test case* diacak menggunakan *random seed* yang telah ditentukan	Tidak melakukan pembelajaran
Performance-based	Rata-rata *response time* historis diurutkan dari nilai terendah ke tertinggi	Menggunakan histori awal dan tidak diperbarui selama eksperimen

A2C membentuk *state* menggunakan histori yang tersedia sebelum siklus pengujian, kemudian jaringan *actor* menghasilkan skor prioritas untuk setiap *test case*. Parameter *actor* dan *critic* tidak diperbarui pada saat *ranking* dibentuk maupun selama eksekusi satu siklus. Pembaruan *policy* dilakukan setelah seluruh hasil siklus tersedia dan digunakan untuk membentuk *ranking* pada siklus berikutnya. Proses tersebut dilakukan secara terpisah pada masing-masing SUT menggunakan *policy* awal yang sesuai dengan model yang sedang diuji.

Random prioritization tidak memanfaatkan histori maupun hasil siklus sebelumnya. Sebaliknya, *performance-based prioritization* menggunakan rata-rata *response_time* historis dan *test_case_id* sebagai pembeda apabila terdapat nilai yang sama. *Ranking performance-based* tetap selama sepuluh siklus eksperimen.

Ranking dari setiap metode ditempatkan pada antrian eksekusi. *Test runner* mengambil *test case* sesuai posisi *ranking* dan mengirim sejumlah *request* secara paralel berdasarkan nilai *concurrency* pada konfigurasi *high-load* SUT yang sedang diuji. *Worker* yang telah menyelesaikan satu *request* mengambil *test case* berikutnya dari antrian.

 ISSN:2302-8149
e-ISSN:2540-9719

f. Pengumpulan Data Eksperimen dan Definisi Operasional Variabel

Hasil eksperimen utama dicatat menggunakan *main experiment logger* yang dipisahkan dari *historical execution logger*. Pemisahan dilakukan untuk mencegah data histori awal tercampur dengan hasil eksperimen utama. Pengumpulan data dilakukan secara terpisah untuk setiap SUT, yaitu DistilBERT dan ELECTRA-small, dengan struktur pencatatan dan definisi variabel yang sama. Data yang disimpan mencakup *model_name*, *run_id*, *cycle_id*, *method_name*, *test_case_id*, *ranking_position*, *priority_score*, *expected label*, *actual label*, *semantic_defect_status*, *runtime_failure_status*, *operational_defect_status*, *response_time*, serta waktu pencatatan.

Tabel 3 Definisi operasional variabel penelitian

Variabel	Definisi Operasional	Sumber Data
SUT/model	Model AI yang digunakan sebagai *System Under Test*, yaitu DistilBERT atau ELECTRA-small	*model_name*
Metode prioritisasi	Strategi pembentukan urutan eksekusi berupa A2C, random, atau performance-based	*method_name*, *ranking_position*
Semantic defect	Label intent hasil prediksi berbeda dari label yang diharapkan	*expected_label*, *actual_label*, *semantic_defect_status*
Runtime failure	Terjadi *timeout*, *error response*, atau respons tidak valid	runtime_failure_status
Operational defect	Terjadi *semantic defect* dan/atau *runtime failure*	operational_defect_status
APFD	Kemampuan ranking menempatkan *test case* pendeteksi *operational defect* lebih awal	*ranking_position*, *operational_defect_status*
TTF	Posisi pertama *test case* yang mengungkap *operational defect*	*ranking_position*, *operational_defect_status*
Response Time	Durasi sejak *request* dikirim hingga respons diterima atau *timeout*	response_time

g. Analisis Data dan Metrik Evaluasi

Data pada *main experiment log* dikelompokkan berdasarkan *model_name*, *run_id*, *cycle_id*, dan *method_name*. Analisis dilakukan secara terpisah untuk masing-masing *system under test*, yaitu DistilBERT dan ELECTRA-small, sehingga hasil dari setiap layanan dapat dievaluasi sesuai dengan karakteristik pengujiannya. Untuk setiap model, kombinasi *run_id* dan *cycle_id* digunakan sebagai unit analisis yang sama untuk ketiga metode prioritisasi, yaitu A2C, *random prioritization*, dan *performance-based prioritization*. Dengan lima pengulangan dan sepuluh siklus pada setiap pengulangan, diperoleh 50 unit analisis berpasangan untuk masing-masing model.

Efektivitas prioritisasi dievaluasi menggunakan *Average Percentage of Faults Detected* (APFD), *Test to First Failure* (TTF), dan *Response Time*. APFD digunakan untuk mengevaluasi kemampuan suatu urutan dalam menempatkan *test case* yang mengungkap *operational defect* pada bagian awal eksekusi [6]. Formulasi APFD yang digunakan dalam penelitian dinyatakan melalui Persamaan (6).

$$APFD(se) = 1 - \frac{\sum_{t \in se} idx(se,t) \times t.v}{|\ se\ | \times m} + \frac{1}{2 \times |\ se\ |} \quad (6)$$

dengan se sebagai urutan *test case* yang dihasilkan oleh metode prioritisasi, $idx(se,t)$ sebagai posisi *test case* t dalam urutan se, $|\ se\ |$ sebagai jumlah *test case* dalam urutan, m sebagai jumlah total kesalahan, dan $t.v$ sebagai hasil eksekusi (*execution verdict)*.

TTF mengukur posisi *test case* pertama yang berhasil mengungkap *operational defect*. Formulasi TTF dinyatakan melalui Persamaan (7).

$$TTF = rank(t_{fail,1}) \quad (7)$$

 ISSN:2302-8149
e-ISSN:2540-9719

dengan *TTF* menyatakan nilai *Test to First Failure*, sedangkan rank ($t_{\text{fail},1}$)merepresentasikan posisi *test case* pertama yang berhasil mengungkap *defect* dalam urutan *test suite* hasil prioritisasi. Nilai TTF yang lebih kecil menunjukkan bahwa cacat pertama berhasil teridentifikasi lebih dekat dengan awal urutan eksekusi.

Response Time digunakan untuk mengevaluasi responsivitas layanan selama pengujian dalam kondisi *high-load*. Waktu respons dihitung sebagai selisih antara waktu ketika respons diterima dan waktu ketika *request* dikirim, sebagaimana dinyatakan melalui Persamaan (8).

$$L = T_{response} - T_{start} \tag{8}$$

dengan L menyatakan waktu response, T_{start} menyatakan waktu ketika klien mengirim permintaan, dan $T_{response}$ menyatakan waktu ketika response terhadap permintaan tersebut diterima. Selain itu, untuk banyak permintaan, rata-rata *response time* dapat dihitung melalui persamaan (9).

$$L_{avg} = \frac{1}{n}\sum_{i=1}^{n}\left(T_{response,i} - T_{start,i}\right) \tag{9}$$

Nilai APFD, TTF, dan *Response Time* dikompilasi berdasarkan model, metode prioritisasi, siklus, dan pengulangan. Setiap metode pada masing-masing model memiliki 50 unit eksperimen dari sepuluh siklus dan lima pengulangan, sehingga terdapat 150 unit eksperimen pada DistilBERT dan 150 pada ELECTRA-small.

Analisis deskriptif mencakup rata-rata, minimum, maksimum, dan ukuran sebaran APFD, TTF, serta *Response Time*. APFD dan TTF digunakan untuk mengevaluasi kemampuan metode dalam menempatkan *test case* yang mengungkap *operational defect* di awal urutan eksekusi, sedangkan *Response Time* digunakan untuk mengevaluasi responsivitas layanan pada kondisi *high-load*.

Perbedaan kinerja antar-metode diuji menggunakan Friedman test secara terpisah untuk setiap model dan metrik, dengan mempertahankan pasangan berdasarkan kombinasi *run_id* dan *cycle_id*. Dengan demikian, nilai A2C, *random prioritization*, dan *performance-based prioritization* dari unit yang sama dibandingkan sebagai data berpasangan. Pengujian dilakukan pada tingkat signifikansi $\alpha = 0{,}05$.

Jika Friedman test menunjukkan perbedaan signifikan, analisis dilanjutkan dengan Wilcoxon signed-rank test untuk membandingkan pasangan A2C dengan *random prioritization*, A2C dengan *performance-based prioritization*, serta *random prioritization* dengan *performance-based prioritization*. Koreksi Holm diterapkan pada *p-value* untuk mengendalikan *multiple comparisons*. Hasil pengujian digunakan untuk mengidentifikasi pasangan metode yang memiliki perbedaan kinerja signifikan.

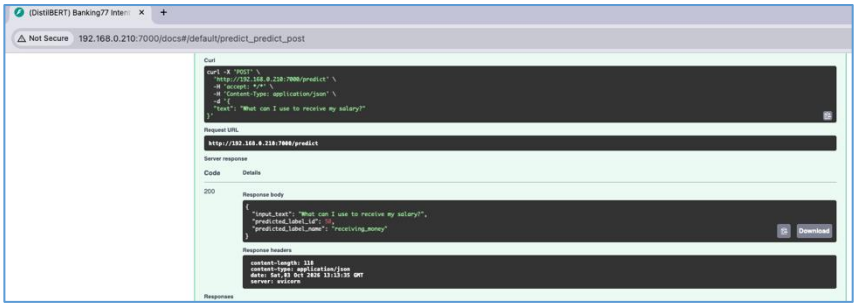
Analisis statistik dilakukan secara terpisah pada DistilBERT dan ELECTRA-small untuk mempertahankan karakteristik masing-masing layanan. Hasil kedua model kemudian dibandingkan secara deskriptif untuk melihat konsistensi pola kinerja metode prioritisasi pada *system under test* yang berbeda.

4 Hasil dan Pembahasan

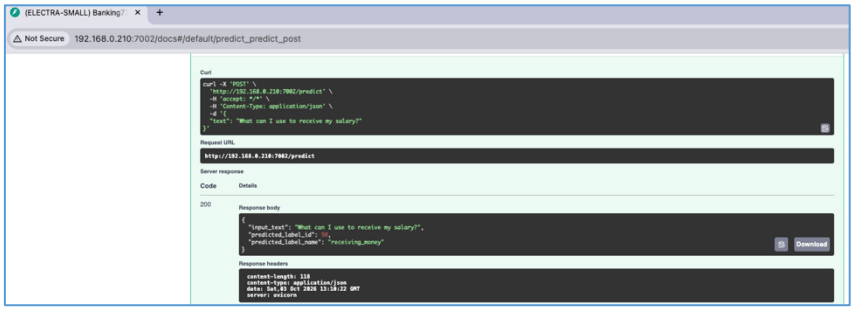
4.1 Persiapan Data dan *System Under Test*

Tahap persiapan menghasilkan dua layanan AI untuk klasifikasi intent, yaitu DistilBERT dan ELECTRA-small, yang digunakan sebagai *system under test*. Kedua model yang telah melalui proses *fine-tuning* diimplementasikan sebagai layanan berbasis API menggunakan FastAPI dan dijalankan secara terpisah dalam kontainer Docker. Setiap layanan menerima input berupa teks dan menghasilkan *predicted label* sebagai keluaran klasifikasi intent. Selanjutnya, hasil prediksi dibandingkan dengan *expected label* pada masing-masing *test case* untuk mengidentifikasi adanya ketidaksesuaian dalam proses klasifikasi. Implementasi kedua layanan tersebut ditunjukkan pada Gambar 2 dan Gambar 3.

 ISSN:2302-8149
e-ISSN:2540-9719



Gambar 2 Implementasi layanan DistilBERT



Gambar 3 Implementasi layanan ELECTRA-small

Test suite dibentuk dari data *test* Banking77, dengan setiap data dikonversi menjadi satu *test case* yang memuat identitas pengujian, *input request*, dan *expected label*. Struktur ini menghasilkan kumpulan *test case* yang siap dieksekusi pada endpoint layanan AI, sebagaimana ditunjukkan pada Gambar 4.

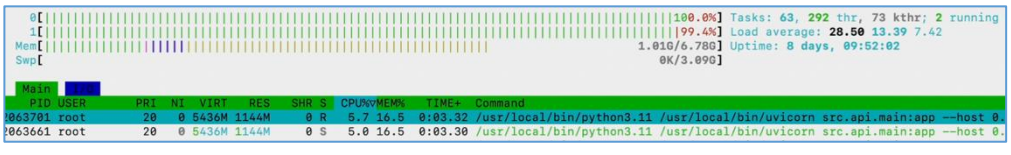
test_id	input_request	expected_label_id	expected_label_name
TC_00001	How do I link the new card?	13	card_linking
TC_00002	I made a card payment and a fee is showing up?	15	card_payment_fee_charged
TC_00003	I am here to change my adress.	30	edit_personal_details
TC_00004	How do I deposit cash into my account?	58	top_up_by_cash_or_cheque
TC_00005	I seem to have a problem here, I bought something about a 2 weeks ago and just now received the payment back into my account and I've already received the item. Can you tell me what is going on here?	53	reverted_card_payment?

Gambar 4 Test suite

4.2 Pembentukan *High-Load Environment*

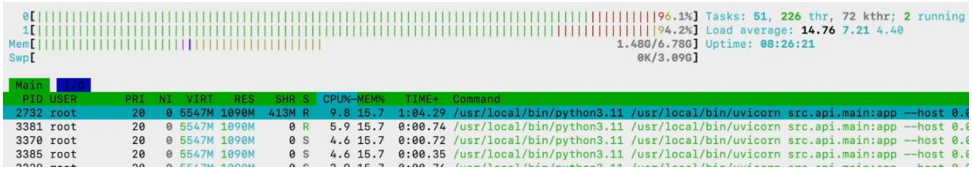
Kalibrasi high-load environment dilakukan secara terpisah pada DistilBERT dan ELECTRA-small menggunakan *environment-validation* subset yang terdiri atas 981 test case. Beberapa kombinasi concurrency dan timeout threshold diuji untuk mengidentifikasi kondisi yang memberikan tekanan komputasi dan peningkatan response time tanpa menyebabkan layanan gagal secara menyeluruh.

Pada model DistilBERT, penerapan kombinasi *concurrency* sebesar 25 dan *timeout threshold* 3 detik menghasilkan kondisi beban yang terlalu ekstrem, yang ditandai dengan 979 request gagal. Sementara itu, kombinasi *concurrency* 25 dengan *timeout threshold* 8 detik memberikan kondisi *high-load* yang lebih proporsional. Pada konfigurasi ini, penggunaan CPU mencapai 99–100%, penggunaan memori berada pada kisaran 1.010 MB, rata-rata *response time* mencapai 3.940,9 ms, serta *success rate* sebesar 96,33% dengan 36 request mengalami kegagalan. Berdasarkan karakteristik tersebut, konfigurasi ini ditetapkan sebagai *high-load environment* untuk model DistilBERT, sebagaimana ditunjukkan pada Gambar 5.



Gambar 5 Kondisi server pada concurrency 25 dan timeout threshold 8 detik (DistilBERT)

Hasil kalibrasi pada ELECTRA-small menunjukkan karakteristik operasional yang berbeda dibandingkan DistilBERT. Pada tingkat *concurrency* yang lebih rendah dengan *timeout threshold* sebesar 3 detik, sebagian besar request tidak berhasil diselesaikan sehingga konfigurasi tersebut tidak digunakan. Peningkatan *timeout threshold* menjadi 8 detik menghasilkan tingkat keberhasilan yang lebih proporsional. Pada *concurrency* 85, rata-rata *response time* tercatat sebesar 6.667,85 ms dengan *success rate* sebesar 96,84% atau terdapat 31 request yang gagal. Sementara itu, pengujian pada *concurrency* 90 menghasilkan *success rate* sebesar 95,31% dengan 46 request gagal, disertai penggunaan CPU pada kisaran 94–96% dan penggunaan memori sekitar 1,48 GB. Berdasarkan karakteristik tersebut, konfigurasi *concurrency* 90 dengan *timeout threshold* 8 detik ditetapkan sebagai *high-load environment* untuk ELECTRA-small, sebagaimana ditunjukkan pada Gambar 6.



Gambar 6 Kondisi server pada concurrency 90 dan timeout threshold 8 detik (ELECTRA-small)

Dengan demikian, konfigurasi akhir *high-load environment* ditetapkan secara spesifik untuk masing-masing model, yaitu *concurrency* 25 dan *timeout threshold* 8 detik untuk DistilBERT serta *concurrency* 90 dan *timeout threshold* 8 detik untuk ELECTRA-small. Konfigurasi tersebut selanjutnya dipertahankan secara konsisten pada pengumpulan *historical execution data* dan eksperimen komparatif untuk masing-masing model.

4.3 Historical Execution Data

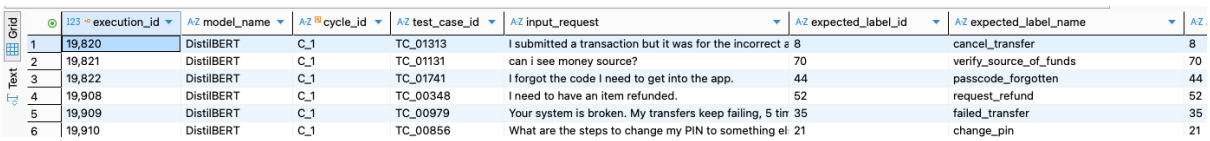
Pembentukan *historical execution data* dilakukan secara terpisah pada DistilBERT dan ELECTRA-small menggunakan *test suite* utama yang terdiri atas 1.963 *test case*. Setiap model dieksekusi selama sepuluh *historical cycle* menggunakan konfigurasi *high-load environment* yang telah ditetapkan pada Subbab 4.2. Dengan demikian, masing-masing model menghasilkan 19.630 *execution records* yang memuat hasil klasifikasi, status *operational defect*, *response time*, dan informasi eksekusi lainnya untuk membentuk histori setiap *test case*.

Pada DistilBERT, pembentukan data historis menggunakan *concurrency* 25 dan *timeout threshold* 8 detik, sedangkan pada ELECTRA-small menggunakan *concurrency* 90 dan *timeout threshold* 8 detik. Perbedaan konfigurasi tersebut mengikuti hasil kalibrasi masing-masing SUT sehingga setiap model diuji pada kondisi *high-load* yang telah ditetapkan untuk karakteristik layanan tersebut. Rincian pembentukan *historical execution data* disajikan pada Tabel 4.

Tabel 4 Konfigurasi historical execution data

Komponen	DistilBERT	ELECTRA-small
Sumber test case	Test suite utama	Test suite utama
Jumlah test case	1.963	1.963
Concurrency	25	90
Timeout threshold	8 detik	8 detik
Jumlah historical cycle	10	10
Jumlah execution records	19.630	19.630

Contoh *historical execution data* yang tersimpan pada *historical execution logger* untuk masing-masing model ditunjukkan pada Gambar 7 dan Gambar 8. Gambar tersebut memperlihatkan data hasil eksekusi yang disimpan untuk setiap *test case*, termasuk informasi hasil prediksi, status *operational defect*, *response time*, dan informasi eksekusi lainnya.



	execution_id	model_name	cycle_id	test_case_id	input_request	expected_label_id	expected_label_name	
1	19,820	DistilBERT	C_1	TC_01313	I submitted a transaction but it was for the incorrect a	8	cancel_transfer	8
2	19,821	DistilBERT	C_1	TC_01131	can i see money source?	70	verify_source_of_funds	70
3	19,822	DistilBERT	C_1	TC_01741	I forgot the code I need to get into the app.	44	passcode_forgotten	44
4	19,908	DistilBERT	C_1	TC_00348	I need to have an item refunded.	52	request_refund	52
5	19,909	DistilBERT	C_1	TC_00979	Your system is broken. My transfers keep failing, 5 tim	35	failed_transfer	35
6	19,910	DistilBERT	C_1	TC_00856	What are the steps to change my PIN to something el	21	change_pin	21

Gambar 7 Contoh historical execution data DistilBERT

 ISSN:2302-8149
e-ISSN:2540-9719

	execution_id	model_name	cycle_id	test_case_id	input_request	expected_label_id	expected_label_name
1	45	electra-small	C_1	TC_00410	I wrote a check to the account yesterday to add to my balai	6	balance_not_updated_after_cheque_or_cas
2	48	electra-small	C_1	TC_00667	I found an unauthorized card payment	16	card_payment_not_recognised
3	49	electra-small	C_1	TC_00928	How do you calculate exchange rates?	32	exchange_rate
4	53	electra-small	C_1	TC_01246	Can i receive transfers from my employer this way?	50	receiving_money
5	55	electra-small	C_1	TC_01323	Is there a fee to order a physical card?	43	order_physical_card
6	58	electra-small	C_1	TC_01395	Hello, I made a payment with my card and it's not working.	45	pending_card_payment

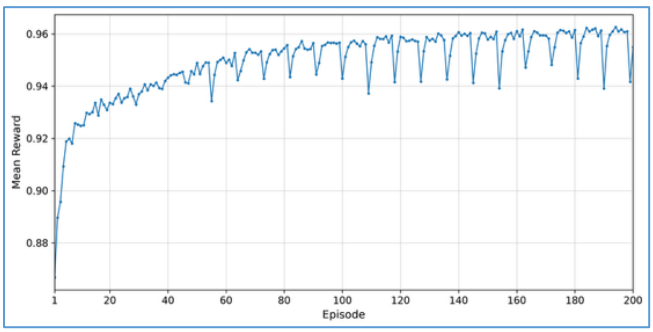
Gambar 8 Contoh historical execution data ELECTRA-small

Data historis yang dihasilkan dari masing-masing model selanjutnya digunakan sebagai sumber informasi awal untuk membentuk *state* dan melatih A2C, serta sebagai histori bagi *performance-based prioritization*. A2C memanfaatkan histori tersebut untuk proses *offline training*, sedangkan *performance-based prioritization* menggunakannya untuk membentuk *ranking* berdasarkan rata-rata *response time* historis. Sebaliknya, *random prioritization* tidak menggunakan data historis dalam pembentukan *ranking*. Pemisahan data berdasarkan model dilakukan agar proses pembelajaran dan pembentukan prioritas tetap sesuai dengan karakteristik SUT yang digunakan.

4.4 Pelatihan Advantage Actor-Critic

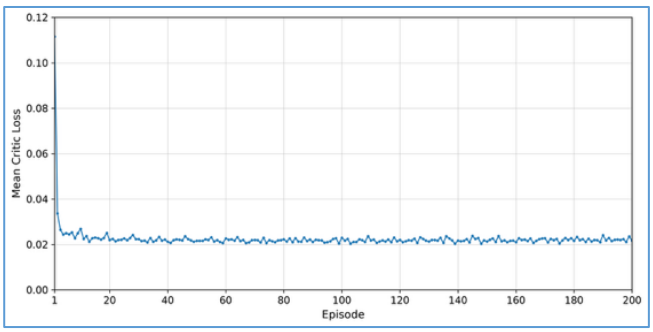
Pelatihan A2C dilakukan secara *offline* menggunakan *historical execution data* dari masing-masing *system under test*, yaitu DistilBERT dan ELECTRA-small, dengan maksimum 200 episode. Proses pelatihan dilakukan secara terpisah agar setiap model memiliki *policy* yang dibentuk berdasarkan karakteristik *historical execution data* model tersebut.

Pada DistilBERT, *mean reward* meningkat dari 0,866769 pada Episode 1 menjadi 0,925327 pada Episode 9 dan mencapai nilai tertinggi 0,962611 pada Episode 194. Peningkatan tersebut menunjukkan bahwa *actor* semakin mampu menghasilkan skor prioritas yang mendekati target dari *historical execution replay environment*, meskipun terdapat fluktuasi antar-episode. Perkembangan *mean reward* ditunjukkan pada Gambar 9.



Gambar 9 *Mean reward* selama training a2c pada DistilBERT

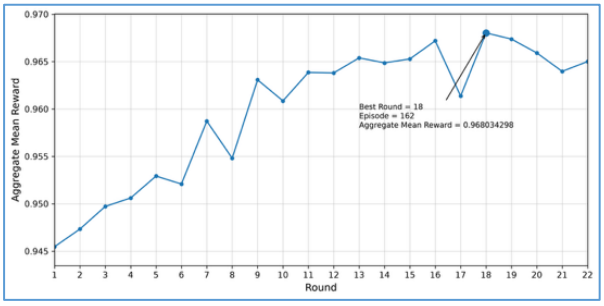
Perkembangan jaringan *critic* pada DistilBERT ditunjukkan melalui *mean critic loss* yang menurun dari 0,111614 pada Episode 1 menjadi 0,026519 pada Episode 3, kemudian relatif stabil di sekitar 0,02. Penurunan tersebut menunjukkan semakin kecilnya perbedaan antara estimasi nilai *state* dan *temporal-difference target* selama pelatihan. Perkembangan *mean critic loss* ditunjukkan pada Gambar 10.



Gambar 10 *Mean critic loss* selama training A2C pada DistilBERT

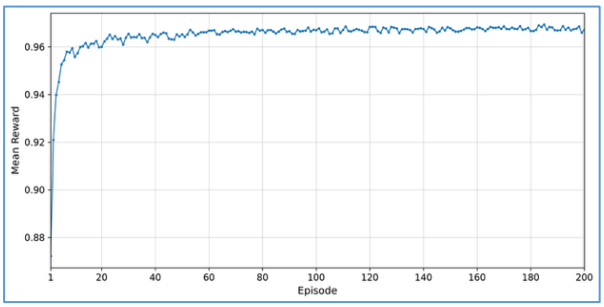
 ISSN:2302-8149
e-ISSN:2540-9719

Pemilihan *initial model checkpoint* DistilBERT didasarkan pada *aggregate mean reward* dari seluruh *historical cycle* C2–C10 dalam setiap *complete round*, bukan pada *reward* tertinggi dari satu episode. Nilai tersebut meningkat dari 0,945469146 pada Round 1 hingga mencapai 0,968034298 pada Round 18 yang berakhir di Episode 162. Karena tidak terdapat nilai agregat yang lebih tinggi pada *complete round* berikutnya, parameter pada Episode 162 ditetapkan sebagai *initial A2C model checkpoint* untuk eksperimen komparatif DistilBERT. Perkembangan *aggregate mean reward* per *round* ditunjukkan pada Gambar 11.



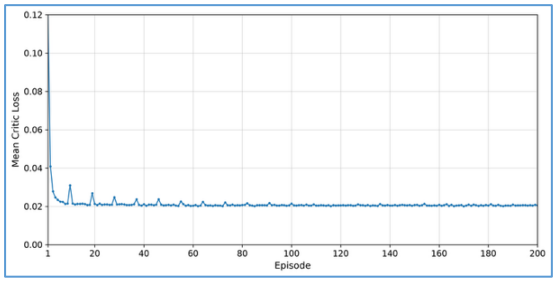
Gambar 11 *Perkembangan aggregate mean reward per round* pada DistilBERT

Pelatihan A2C selanjutnya dilakukan menggunakan *historical execution data* ELECTRA-small dengan konfigurasi jaringan dan *hyperparameter* yang sama. Perkembangan *mean reward* menunjukkan peningkatan yang relatif cepat pada fase awal pelatihan, dari 0,872122 pada Episode 1 menjadi 0,959390 pada Episode 9. Setelah fase awal tersebut, *mean reward* terus mengalami peningkatan dengan fluktuasi antar-episode dan mencapai nilai tertinggi sebesar 0,969339 pada Episode 185. Nilai *mean reward* pada episode-episode akhir tetap berada pada kisaran yang relatif stabil, dengan nilai 0,968563 pada Episode 198 dan 0,967165 pada Episode 200. Pola tersebut menunjukkan bahwa *actor* semakin mampu menghasilkan skor prioritas yang mendekati target dari *historical execution replay environment* ELECTRA-small. Perkembangan *mean reward* ditunjukkan pada Gambar 12



Gambar 12 *Mean reward* selama training A2C pada ELECTRA-small

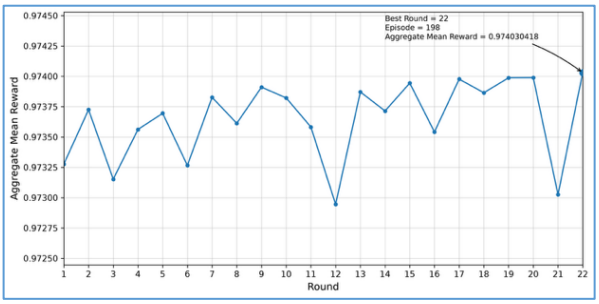
Perkembangan jaringan *critic* ELECTRA-small menunjukkan penurunan *mean critic loss* yang cukup besar pada fase awal pelatihan. Nilainya menurun dari 0,123075 pada Episode 1 menjadi 0,027832 pada Episode 3, kemudian bergerak relatif stabil di sekitar 0,020 pada episode-episode berikutnya. Pada Episode 198, *mean critic loss* tercatat sebesar 0,020489. Pola tersebut menunjukkan bahwa penyesuaian terbesar pada estimasi nilai *critic* terjadi pada fase awal pelatihan, kemudian stabil pada fase berikutnya. Perkembangan *mean critic loss* ditunjukkan pada Gambar 13.



Gambar 13 *Mean Critic Loss* Selama Training A2C pada ELECTRA-small

 ISSN:2302-8149
e-ISSN:2540-9719

Evaluasi *policy* ELECTRA-small dilakukan menggunakan *aggregate mean reward* pada setiap *complete round*. Pelatihan menghasilkan 22 complete round hingga Episode 198. Nilai *aggregate mean reward* mengalami fluktuasi antar-*round*, tetapi secara keseluruhan berada pada tingkat yang relatif stabil. Nilai tersebut meningkat dari 0,973276772 pada Round 1 dan mencapai nilai tertinggi sebesar 0,974030418 pada Round 22, yang berakhir pada Episode 198. Nilai tersebut menjadi hasil evaluasi agregat terbaik selama pelatihan karena tidak terdapat *complete round* sebelumnya yang menghasilkan nilai lebih tinggi. Perkembangan *aggregate mean reward* per *round* ditunjukkan pada



Gambar 14.

Gambar 14 *Perkembangan aggregate mean reward per round* pada ELECTRA-small

Berdasarkan hasil evaluasi tersebut, parameter model pada Episode 198 ditetapkan sebagai *initial A2C model checkpoint* untuk eksperimen komparatif ELECTRA-small. Pemilihan checkpoint didasarkan pada *aggregate mean reward* tertinggi dari *complete round* yang mencakup seluruh *historical cycle* C2–C10, bukan pada *mean reward* tertinggi dari satu episode. Dengan demikian, meskipun *mean reward* individual tertinggi tercapai pada Episode 185, Episode 198 dipilih karena menghasilkan *aggregate mean reward* tertinggi pada evaluasi seluruh *historical cycle*. *Checkpoint* tersebut selanjutnya digunakan sebagai kondisi awal A2C pada eksperimen komparatif ELECTRA-small.

4.5 Eksperimen Comparison Baselines

Eksperimen komparatif dilaksanakan secara terpisah pada dua *system under test*, yaitu DistilBERT dan ELECTRA-small, dengan membandingkan tiga metode prioritisasi, yaitu A2C, *random prioritization*, dan *performance-based prioritization*. Pada masing-masing model, setiap metode dijalankan selama sepuluh *cycle* pada lima run, sehingga menghasilkan 50 unit eksperimen per metode atau 150 unit eksperimen untuk setiap model. Setiap unit eksperimen mengeksekusi seluruh 1.963 *test case* sesuai prinsip *full test suite prioritization*.

Pada DistilBERT, eksperimen menggunakan konfigurasi *high-load* dengan *concurrency* 25 dan *timeout threshold* 8 detik, sedangkan ELECTRA-small menggunakan *concurrency* 90 dan *timeout threshold* 8 detik. Perbedaan konfigurasi tersebut mengikuti hasil kalibrasi masing-masing SUT, sedangkan mekanisme pembentukan *ranking*, jumlah *test case*, jumlah *cycle*, dan jumlah *run* dipertahankan sama agar perbandingan antar-metode dalam setiap model dapat dilakukan secara konsisten.

Berdasarkan rekapitulasi pelaksanaan eksperimen, setiap metode pada masing-masing model menghasilkan 50 unit eksperimen yang terdiri atas lima *run* dan sepuluh *cycle*. Setiap metode menghasilkan 98.150 *execution records*, yang diperoleh dari 50 unit eksperimen dengan masing-masing unit menjalankan 1.963 *test case*. Dengan demikian, DistilBERT menghasilkan 150 unit eksperimen atau 294.450 *execution records*, sedangkan ELECTRA-small menghasilkan jumlah yang sama. Secara keseluruhan, eksperimen pada kedua model menghasilkan 300 unit eksperimen dan 588.900 *execution records*, sebagaimana ditunjukkan pada Gambar 15.

SELECT * FROM (SELECT model_name AS "Model Nar Enter a SQL expression to filter results (use Ctrl+Space)

Grid

	Model Name	Metode	Total Experiment Units	Total Execution Records	Total Run	Total Cycles	Total Test Case per Cycles
1	DistilBERT	A2C	50	98,150	5	10	1,963
2	DistilBERT	Random Prioritization	50	98,150	5	10	1,963
3	DistilBERT	Performance-based Prioritization	50	98,150	5	10	1,963
4	electra-small	A2C	50	98,150	5	10	1,963
5	electra-small	Random Prioritization	50	98,150	5	10	1,963
6	electra-small	Performance-based Prioritization	50	98,150	5	10	1,963

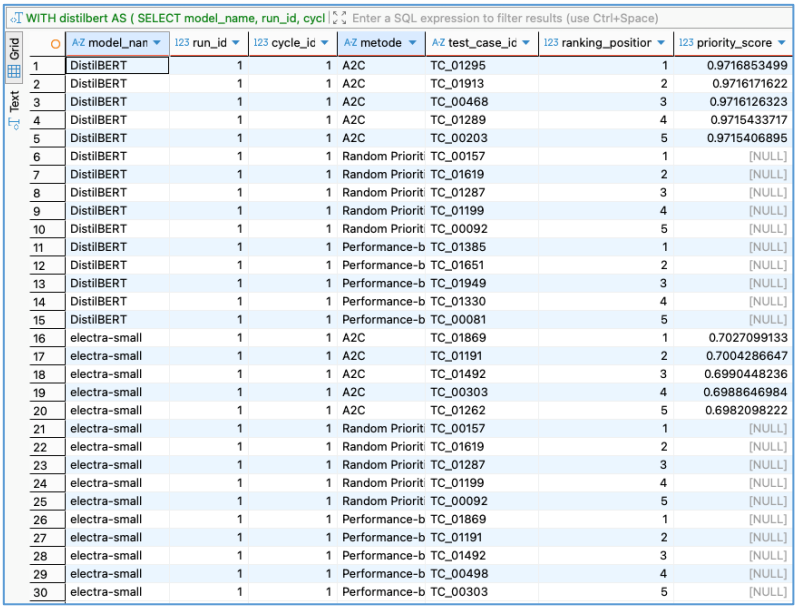
 ISSN:2302-8149
e-ISSN:2540-9719

Gambar 15 Hasil pelaksanaan eksperimen komparatif

4.6 Pengumpulan Data Eksperimen dan Definisi Operasional Variabel

Seluruh hasil eksperimen DistilBERT dan ELECTRA-small dicatat pada *main experiment log* yang terpisah dari *historical execution data*. Data yang disimpan mencakup *model_name*, *run_id*, *cycle_id*, *method_name*, *test_case_id*, *ranking_position*, *priority_score*, *expected label*, *actual label*, status *semantic defect*, *runtime failure*, *operational defect*, dan *response time*. Struktur tersebut memungkinkan setiap hasil eksekusi ditelusuri berdasarkan model, metode, *cycle*, *run*, dan posisi *ranking*.

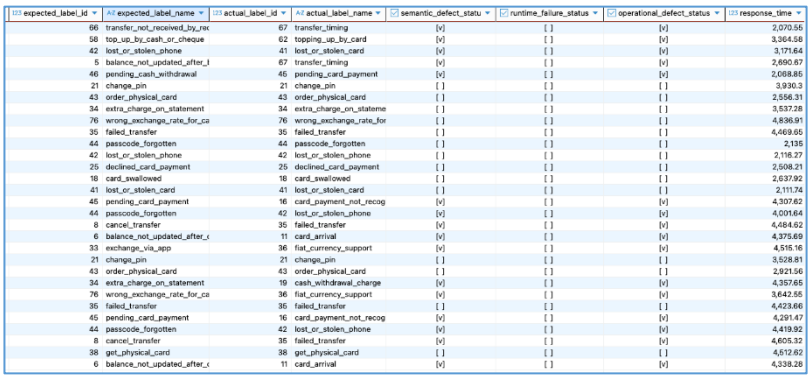
Data yang tersimpan dalam *main experiment log* mencakup informasi *ranking* dan hasil eksekusi *test case*. Informasi *ranking* yang dihasilkan oleh ketiga metode pada model DistilBERT dan ELECTRA-small meliputi model_name, run_id, cycle_id, method_name, test_case_id, ranking_position, dan priority_score. Pada metode A2C, atribut *priority score* digunakan sebagai dasar dalam pembentukan *ranking*, sedangkan atribut tersebut tidak digunakan pada metode *random prioritization* dan *performance-based prioritization*. Selain itu, hasil eksekusi *test case* mencakup *expected label*, *actual label*, status *semantic defect*, *runtime failure*, *operational defect*, serta *response time*. Contoh data *ranking* dan hasil eksekusi *test case* yang tersimpan dalam *main experiment log* ditunjukkan pada Gambar 16 dan Gambar 17.



WITH distilbert AS (SELECT model_name, run_id, cycl | Enter a SQL expression to filter results (use Ctrl+Space)

	model_nan	run_id	cycle_ic	metode	test_case_ic	ranking_positior	priority_score
1	DistilBERT	1	1	A2C	TC_01295	1	0.9716853499
2	DistilBERT	1	1	A2C	TC_01913	2	0.9716171622
3	DistilBERT	1	1	A2C	TC_00468	3	0.9716126323
4	DistilBERT	1	1	A2C	TC_01289	4	0.9715433717
5	DistilBERT	1	1	A2C	TC_00203	5	0.9715406895
6	DistilBERT	1	1	Random Priorit	TC_00157	1	[NULL]
7	DistilBERT	1	1	Random Priorit	TC_01619	2	[NULL]
8	DistilBERT	1	1	Random Priorit	TC_01287	3	[NULL]
9	DistilBERT	1	1	Random Priorit	TC_01199	4	[NULL]
10	DistilBERT	1	1	Random Priorit	TC_00092	5	[NULL]
11	DistilBERT	1	1	Performance-b	TC_01385	1	[NULL]
12	DistilBERT	1	1	Performance-b	TC_01651	2	[NULL]
13	DistilBERT	1	1	Performance-b	TC_01949	3	[NULL]
14	DistilBERT	1	1	Performance-b	TC_01330	4	[NULL]
15	DistilBERT	1	1	Performance-b	TC_00081	5	[NULL]
16	electra-small	1	1	A2C	TC_01869	1	0.7027099133
17	electra-small	1	1	A2C	TC_01191	2	0.7004286647
18	electra-small	1	1	A2C	TC_01492	3	0.6990448236
19	electra-small	1	1	A2C	TC_00303	4	0.6988646984
20	electra-small	1	1	A2C	TC_01262	5	0.6982098222
21	electra-small	1	1	Random Priorit	TC_00157	1	[NULL]
22	electra-small	1	1	Random Priorit	TC_01619	2	[NULL]
23	electra-small	1	1	Random Priorit	TC_01287	3	[NULL]
24	electra-small	1	1	Random Priorit	TC_01199	4	[NULL]
25	electra-small	1	1	Random Priorit	TC_00092	5	[NULL]
26	electra-small	1	1	Performance-b	TC_01869	1	[NULL]
27	electra-small	1	1	Performance-b	TC_01191	2	[NULL]
28	electra-small	1	1	Performance-b	TC_01492	3	[NULL]
29	electra-small	1	1	Performance-b	TC_00498	4	[NULL]
30	electra-small	1	1	Performance-b	TC_00303	5	[NULL]

Gambar 16 *Main experiment log part 1*



expected_label_id	expected_label_name	actual_label_id	actual_label_name	semantic_defect_statu	runtime_failure_status	operational_defect_status	response_time
66	transfer_not_received_by_rec	67	transfer_timing	[v]	[ ]	[v]	2,070.55
58	top_up_by_cash_or_cheque	62	topping_up_by_card	[v]	[ ]	[v]	3,364.58
42	lost_or_stolen_phone	41	lost_or_stolen_card	[v]	[ ]	[v]	3,171.64
5	balance_not_updated_after_l	67	transfer_timing	[v]	[ ]	[v]	2,690.67
46	pending_cash_withdrawal	45	pending_card_payment	[v]	[ ]	[v]	2,068.86
21	change_pin	21	change_pin	[ ]	[ ]	[ ]	3,930.3
43	order_physical_card	43	order_physical_card	[ ]	[ ]	[ ]	2,556.31
34	extra_charge_on_statement	34	extra_charge_on_stateme	[ ]	[ ]	[ ]	3,537.28
76	wrong_exchange_rate_for_ca	76	wrong_exchange_rate_for	[ ]	[ ]	[ ]	4,836.91
35	failed_transfer	35	failed_transfer	[ ]	[ ]	[ ]	4,469.65
44	passcode_forgotten	44	passcode_forgotten	[ ]	[ ]	[ ]	2,135
42	lost_or_stolen_phone	42	lost_or_stolen_phone	[ ]	[ ]	[ ]	2,116.27
25	declined_card_payment	25	declined_card_payment	[ ]	[ ]	[ ]	2,508.21
18	card_swallowed	18	card_swallowed	[ ]	[ ]	[ ]	2,637.92
41	lost_or_stolen_card	41	lost_or_stolen_card	[ ]	[ ]	[ ]	2,111.74
45	pending_card_payment	16	card_payment_not_recog	[v]	[ ]	[v]	4,307.62
44	passcode_forgotten	42	lost_or_stolen_phone	[v]	[ ]	[v]	4,001.64
8	cancel_transfer	35	failed_transfer	[v]	[ ]	[v]	4,484.62
6	balance_not_updated_after_c	11	card_arrival	[v]	[ ]	[v]	4,375.69
33	exchange_via_app	36	fiat_currency_support	[v]	[ ]	[v]	4,515.16
21	change_pin	21	change_pin	[ ]	[ ]	[ ]	3,528.81
43	order_physical_card	43	order_physical_card	[ ]	[ ]	[ ]	2,921.56
34	extra_charge_on_statement	19	cash_withdrawal_charge	[v]	[ ]	[v]	4,357.65
76	wrong_exchange_rate_for_ca	36	fiat_currency_support	[v]	[ ]	[v]	3,642.55
35	failed_transfer	35	failed_transfer	[ ]	[ ]	[ ]	4,423.66
45	pending_card_payment	16	card_payment_not_recog	[v]	[ ]	[v]	4,291.47
44	passcode_forgotten	42	lost_or_stolen_phone	[v]	[ ]	[v]	4,419.92
8	cancel_transfer	35	failed_transfer	[v]	[ ]	[v]	4,605.32
38	get_physical_card	38	get_physical_card	[ ]	[ ]	[ ]	4,512.82
6	balance_not_updated_after_c	11	card_arrival	[v]	[ ]	[v]	4,338.28

Gambar 17 *Main experiment log part 2*

Pemeriksaan dilakukan untuk memastikan urutan *ranking* lengkap, tidak terdapat duplikasi *test case* maupun posisi *ranking*, serta data *operational defect* dan *response time* tersedia secara konsisten pada kedua model. Data yang telah tervalidasi kemudian digunakan untuk menghitung APFD dan

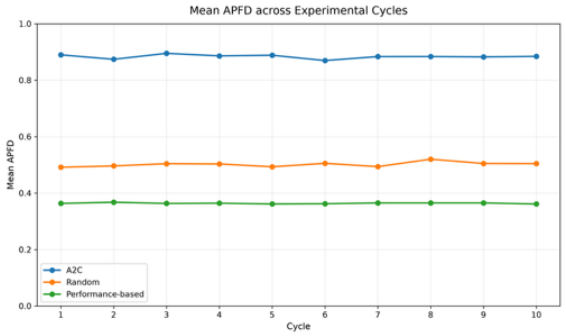
 ISSN:2302-8149
e-ISSN:2540-9719

TTF berdasarkan *ranking_position* dan *operational_defect_status*, sedangkan *response_time* digunakan untuk menghitung *Response Time* pada tahap analisis.

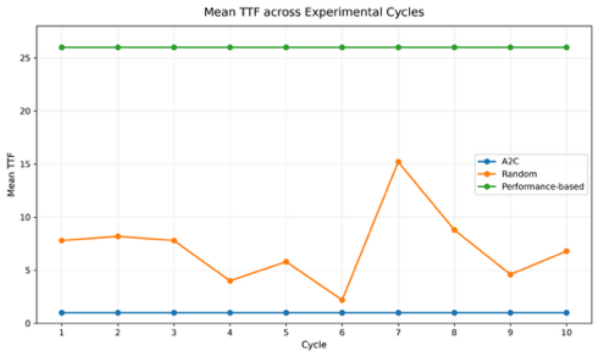
4.7 Analisis Data dan Metrik Evaluasi

Analisis hasil eksperimen dilakukan secara terpisah pada DistilBERT dan ELECTRA-small. Setiap model menghasilkan 150 unit eksperimen dari tiga metode prioritisasi, sepuluh *cycle*, dan lima *run*. Setiap unit menghasilkan nilai APFD, TTF, dan rata-rata *Response Time*, yang kemudian diagregasikan berdasarkan metode dan *cycle* untuk mengevaluasi efektivitas prioritisasi.

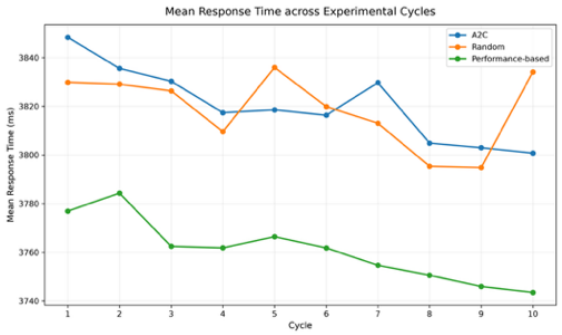
Pada DistilBERT, A2C menghasilkan rata-rata APFD sebesar 0,883815838, lebih tinggi dibandingkan *random prioritization* sebesar 0,501782458 dan *performance-based prioritization* sebesar 0,364008492. A2C juga menghasilkan rata-rata TTF 1,000 pada seluruh *cycle*, sedangkan *random prioritization* dan *performance-based prioritization* masing-masing menghasilkan 7,120 dan 26,000. Pada *Response Time*, A2C memperoleh rata-rata 3.820,484 ms, *random prioritization* 3.818,818 ms, dan *performance-based prioritization* 3.760,845 ms. Hasil tersebut menunjukkan bahwa keunggulan A2C pada DistilBERT terutama terdapat pada kemampuan menempatkan *test case* yang mengungkap *operational defect* pada posisi awal, bukan pada pencapaian *Response Time* terendah, sebagaimana ditunjukkan pada Gambar 18, Gambar 19, dan Gambar 20.



Gambar 18 Rata-Rata APFD di Seluruh Siklus pada DistilBERT



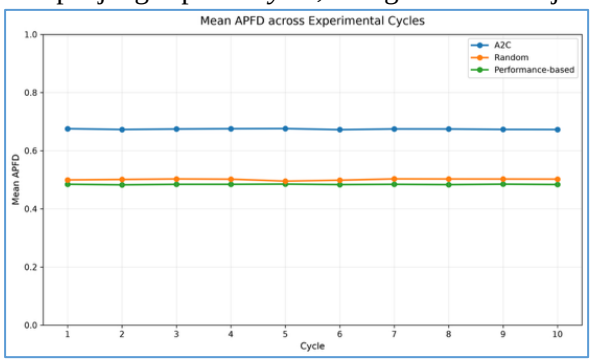
Gambar 19 Rata-Rata TTF di Seluruh Siklus pada DistilBERT



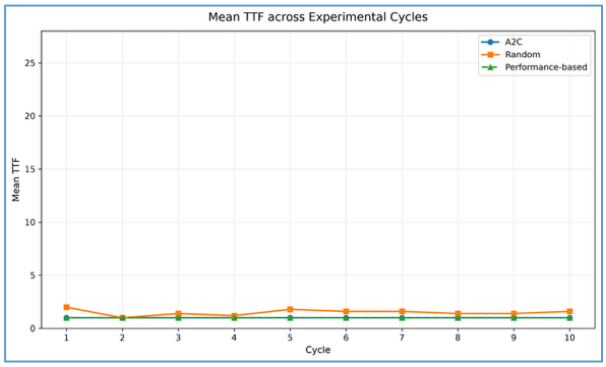
 ISSN:2302-8149
e-ISSN:2540-9719

Gambar 20 Rata-Rata response time di seluruh siklus pada DistilBERT

Pada ELECTRA-small, pola yang serupa juga terlihat pada APFD dan TTF. A2C menghasilkan rata-rata APFD sebesar 0,674268319, lebih tinggi dibandingkan *random prioritization* sebesar 0,501071145 dan *performance-based prioritization* sebesar 0,484010251. A2C menghasilkan rata-rata TTF 1,000 dengan standar deviasi 0,000, sedangkan *random prioritization* menghasilkan rata-rata 1,500 dan *performance-based prioritization* juga 1,000. Hasil per-*cycle* menunjukkan A2C mempertahankan TTF 1,000 sepanjang sepuluh *cycle*, sebagaimana ditunjukkan pada Gambar 21, dan Gambar 22.

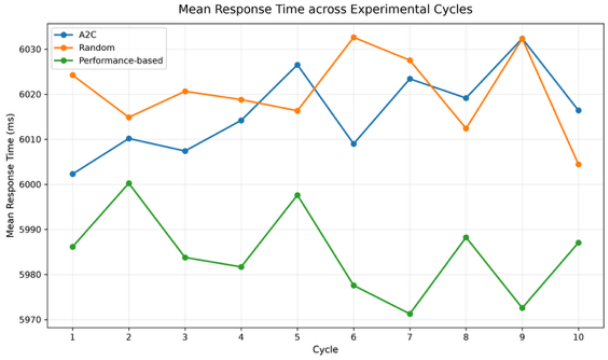


Gambar 21 Rata-Rata APFD pada ELECTRA-small



Gambar 22 Rata-Rata TTF pada ELECTRA-small

Pada metrik *Response Time*, ELECTRA-small menunjukkan pola yang berbeda dari APFD dan TTF. *Performance-based prioritization* menghasilkan rata-rata terendah sebesar 5.984,625 ms, sedangkan A2C sebesar 6.016,103 ms dan *random prioritization* sebesar 6.020,431 ms. Hasil tersebut menunjukkan bahwa A2C kembali memberikan keunggulan pada efektivitas prioritisasi, sementara *performance-based prioritization* menghasilkan respons layanan yang lebih rendah, sebagaimana ditunjukkan pada Gambar 23.



 ISSN:2302-8149
e-ISSN:2540-9719

Gambar 23 Rata-Rata Response Time pada ELECTRA-small

Pengujian perbedaan kinerja antar-metode dilakukan menggunakan *Friedman test* pada 50 unit analisis berpasangan untuk setiap model dan metrik. Perbedaan yang signifikan dianalisis lebih lanjut menggunakan *Wilcoxon signed-rank test* dengan koreksi Holm. Hasil pengujian statistik disajikan pada Tabel 5.

Tabel 5 Hasil pengujian statistik perbandingan metode

Model	Metrik	Friedman χ^2	Friedman *p*	A2C vs Random (Holm *p*)	A2C vs Performance-based (Holm *p*)	Random vs Performance-based (Holm *p*)
DistilBERT	APFD	100,000	$1{,}93\times10^{-22}$	$5{,}33\times10^{-15}$	$5{,}33\times10^{-15}$	$5{,}33\times10^{-15}$
DistilBERT	TTF	97,278	$7{,}52\times10^{-22}$	$7{,}26\times10^{-9}$	$4{,}61\times10^{-12}$	$1{,}44\times10^{-9}$
DistilBERT	Response Time	81,760	$1{,}76\times10^{-18}$	$7{,}50\times10^{-5}$	$5{,}33\times10^{-15}$	$5{,}33\times10^{-15}$
ELECTRA-small	APFD	100,000	$1{,}93\times10^{-22}$	$5{,}33\times10^{-15}$	$5{,}33\times10^{-15}$	$5{,}33\times10^{-15}$
ELECTRA-small	TTF	34,000	$4{,}14\times10^{-8}$	$5{,}81\times10^{-4}$	1,000	$5{,}81\times10^{-4}$
ELECTRA-small	Response Time	46,840	$6{,}74\times10^{-11}$	0,362	$3{,}82\times10^{-8}$	$1{,}29\times10^{-9}$

Hasil *Friedman test* menunjukkan adanya perbedaan yang signifikan pada APFD, TTF, dan *Response Time* untuk kedua model ($p < 0{,}001$). Pada DistilBERT, seluruh pasangan metode menunjukkan perbedaan yang signifikan setelah koreksi Holm. Pada ELECTRA-small, seluruh pasangan metode pada APFD juga menunjukkan perbedaan yang signifikan. Untuk TTF, A2C berbeda signifikan dengan *random prioritization*, sedangkan A2C dan *performance-based prioritization* tidak menunjukkan adanya perbedaan karena nilai TTF kedua metode identik pada seluruh unit analisis. Pada kondisi tersebut, seluruh selisih berpasangan bernilai nol sehingga uji *Wilcoxon signed-rank* tidak menunjukkan adanya perbedaan antara kedua metode. Pada *Response Time*, A2C tidak menunjukkan perbedaan signifikan terhadap *random prioritization*, sedangkan A2C dan *random prioritization* masing-masing berbeda signifikan dengan *performance-based prioritization.*

Secara keseluruhan, kedua model menunjukkan pola yang konsisten pada efektivitas prioritisasi. A2C menghasilkan APFD tertinggi pada DistilBERT maupun ELECTRA-small dan mempertahankan TTF sebesar 1,000 pada kedua model. Sebaliknya, *performance-based prioritization* menghasilkan *Response Time* terendah pada kedua model. Dengan demikian, keunggulan A2C terutama terletak pada kemampuannya membentuk urutan prioritas yang mendukung deteksi *operational defect* lebih awal, sedangkan metode tersebut tidak menunjukkan keunggulan pada *Response Time*.

5 Kesimpulan

Hasil penelitian menunjukkan bahwa *adaptive test case prioritization* berbasis Advantage Actor-Critic (A2C) mampu mendukung deteksi dini *operational defect* pada layanan AI dalam kondisi *high-load*. Pada DistilBERT, A2C menghasilkan rata-rata APFD sebesar 0,883816, lebih tinggi dibandingkan *random prioritization* sebesar 0,501782 dan *performance-based prioritization* sebesar 0,364008. Pada ELECTRA-small, A2C juga menghasilkan rata-rata APFD tertinggi sebesar 0,674268, dibandingkan *random prioritization* sebesar 0,501071 dan *performance-based prioritization* sebesar 0,484010. Hasil TTF menunjukkan bahwa A2C mempertahankan nilai 1,000 pada kedua model, sedangkan *random prioritization* menghasilkan 7,120 pada DistilBERT dan 1,500 pada ELECTRA-small. Pada DistilBERT, *performance-based prioritization* menghasilkan TTF sebesar 26,000, sedangkan pada ELECTRA-small nilainya identik dengan A2C, yaitu 1,000. Pengujian statistik menggunakan *Friedman test* menunjukkan adanya perbedaan yang signifikan pada

 ISSN:2302-8149
e-ISSN:2540-9719

APFD, TTF, dan *Response Time* pada kedua model ($p < 0{,}001$). Hasil *Wilcoxon signed-rank test* dengan koreksi Holm menunjukkan bahwa seluruh pasangan metode pada APFD berbeda signifikan pada kedua model. Pada TTF, seluruh pasangan metode pada DistilBERT berbeda signifikan, sedangkan pada ELECTRA-small A2C berbeda signifikan dengan *random prioritization*, tetapi tidak berbeda dengan *performance-based prioritization* karena nilai TTF kedua metode identik pada seluruh unit analisis. Untuk *Response Time*, *performance-based prioritization* menghasilkan nilai terendah pada kedua model, yaitu 3.760,845 ms pada DistilBERT dan 5.984,625 ms pada ELECTRA-small. Pada DistilBERT seluruh pasangan metode berbeda signifikan, sedangkan pada ELECTRA-small perbedaan antara A2C dan *random prioritization* tidak signifikan. Secara keseluruhan, hasil pada dua model menunjukkan pola yang konsisten bahwa keunggulan A2C terutama terletak pada efektivitas pembentukan urutan prioritas untuk mendukung deteksi *operational defect* lebih awal, yang tercermin melalui APFD dan TTF. Sebaliknya, *performance-based prioritization* memberikan *Response Time* terendah pada kedua model. Dengan demikian, penerapan A2C menunjukkan potensi sebagai pendekatan prioritisasi adaptif untuk pengujian layanan AI pada kondisi *high-load*, khususnya ketika tujuan utama pengujian adalah memperoleh informasi mengenai *operational defect* sedini mungkin. Validasi pada DistilBERT dan ELECTRA-small juga menunjukkan bahwa pola keunggulan A2C pada efektivitas prioritisasi tetap terlihat ketika model AI yang digunakan sebagai *system under test* berbeda.

Referensi

[1] Y. X. Tang, Z. L. Ren, H. Jiang, X. C. Li, and W. Q. Kong, "*An Empirical Comparison Between Tutorials and Crowd Documentation of Application Programming Interface*," *J. Comput. Sci. Technol.*, Vol. 36, No. 4, pp. 856–876, Jul. 2021, DOI: 10.1007/s11390-020-0042-0.

[2] G. Giray, "*The Pre-Print of the A Software Engineering Perspective on Engineering Machine Learning Systems: State of the Art and Challenges*," 2021. [Online]. Available: https://conferences.oreilly.com/software-architecture/sa-eu-

[3] S. Herbold and T. Haar, "*Smoke Testing for Machine Learning: Simple Tests to Discover Severe Bugs*," *Empir. Softw. Eng.*, Vol. 27, No. 2, Mar. 2022, DOI: 10.1007/s10664-021-10073-7.

[4] M. H. Moghadam, M. Saadatmand, M. Borg, M. Bohlin, and B. Lisper, "*An Autonomous Performance Testing Framework using Self-Adaptive Fuzzy Reinforcement Learning*," *Software Quality Journal*, Vol. 30, No. 1, pp. 127–159, Mar. 2022, DOI: 10.1007/s11219-020-09532-z.

[5] G. Bercaru, C. O. Truică, C. G. Chiru, and T. Rebedea, "*Improving Intent Classification using Unlabeled Data from Large Corpora*," *Mathematics*, Vol. 11, No. 3, Feb. 2023, DOI: 10.3390/math11030769.

[6] M. Bagherzadeh, N. Kahani, and L. Briand, "*Reinforcement Learning for Test Case Prioritization*," Mar. 2021, [Online]. Available: http://arxiv.org/abs/2011.01834

[7] C. Laaber, H. C. Gall, and P. Leitner, "*Applying Test Case Prioritization to Software Microbenchmarks*," *Empir. Softw. Eng.*, Vol. 26, No. 6, Nov. 2021, DOI: 10.1007/s10664-021-10037-x.

[8] A. S. Yaraghi, M. Bagherzadeh, N. Kahani, and L. Briand, "*Scalable and Accurate Test Case Prioritization in Continuous Integration Contexts*," Apr. 2022, DOI: 10.1109/TSE.2022.3184842.

[9] P. S. N. Mindom, A. Nikanjam, and F. Khomh, "*A Comparison of Reinforcement Learning Frameworks for Software Testing Tasks*," Jun. 2023, [Online]. Available: http://arxiv.org/abs/2208.12136

[10] Q. Su, X. Li, Y. Ren, R. Qiu, C. Hu, and Y. Yin, "*Attention Transfer Reinforcement Learning for Test Case Prioritization in Continuous Integration*," *Applied Sciences (Switzerland)*, Vol. 15, No. 4, Feb. 2025, DOI: 10.3390/app15042243.

[11] Y. Yang, C. Pan, Z. Li, and R. Zhao, "*Adaptive Reward Computation in Reinforcement Learning-Based Continuous Integration Testing*," *IEEE Access*, Vol. 9, pp. 36674–36688, 2021, DOI: 10.1109/ACCESS.2021.3063232.

[12] N. A. Alrakban, M. Alrashoud, and M. Abdullah-Al-Wadud, "*Optimizing Test Case Prioritization with Meta Deep Reinforcement Learning in Continuous Integration*," *IEEE Access*, Vol. 13, pp. 172435–172455, 2025, DOI: 10.1109/ACCESS.2025.3617387.

[13] S. Śmigiel, "*ECG Classification using Orthogonal Matching Pursuit and Machine Learning*," *Sensors*, Vol. 22, No. 13, Jul. 2022, DOI: 10.3390/s22134960.

[14] Y. Niu and S. S. Chai, "*Design and Optimization of a Test Case Generation Algorithm for Real-Time Embedded Systems based on adaptive Q-Learning*," *Automated Software Engineering*, Vol. 33, No. 2, Dec. 2026, DOI: 10.1007/s10515-026-00598-w.

[15] A. Jayanetti, S. Halgamuge, and R. Buyya, "*Deep Reinforcement Learning for Energy and Time Optimized Scheduling of Precedence-Constrained Tasks in Edge–Cloud Computing Environments*," *Future Generation Computer Systems*, Vol. 137, pp. 14–30, Dec. 2022, DOI: 10.1016/j.future.2022.06.012.

[16] Z. Wang, M. Goudarzi, M. Gong, and R. Buyya, "*Deep Reinforcement Learning-based Scheduling for Optimizing System Load and Response Time in Edge and Fog Computing Environments*," *Future Generation Computer Systems*, Vol. 152, pp. 55–69, Mar. 2024, DOI: 10.1016/j.future.2023.10.012.

[17] L. Giamattei, A. Guerriero, R. Pietrantuono, and S. Russo, "*Automated Functional and Robustness Testing of Microservice Architectures*," *Journal of Systems and Software*, Vol. 207, Jan. 2024, DOI: 10.1016/j.jss.2023.111857.