BAB 2
LANDASAN TEORI

2.1 Reinforcement Learning

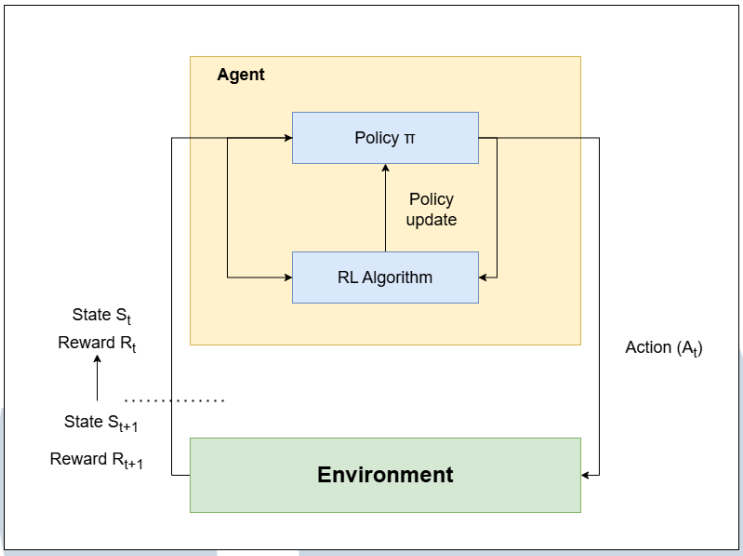
Reinforcement Learning (RL) merupakan salah satu pendekatan dalam *machine learning* (ML) yang mempelajari bagaimana suatu agen berinteraksi dengan lingkungan (*environment*) untuk memperoleh strategi pengambilan keputusan melalui proses *trial-and-error*. Agen mempelajari strategi tersebut berdasarkan sinyal *reward* yang diterima dari lingkungan sebagai umpan balik terhadap *action* yang dilakukan. Salah satu keunggulan RL adalah kemampuannya menangani permasalahan pengambilan keputusan sekuensial pada lingkungan yang dinamis maupun tidak pasti.

RL terdiri atas beberapa komponen utama, yaitu agen (*agent*), lingkungan (*environment*), *state*, *action*, *reward*, dan *policy*. Agen merupakan entitas yang bertugas mengambil keputusan melalui interaksi dengan lingkungan. Pada setiap *timestep*, agen mengamati kondisi lingkungan yang direpresentasikan sebagai *state* (S_t), kemudian memilih suatu *action* (A_t) berdasarkan *policy* (π) yang dimilikinya. Setelah *action* dieksekusi, lingkungan berpindah ke *state* berikutnya (S_{t+1}) dan memberikan sinyal *reward* (R_{t+1}) kepada agen. *Reward* merupakan sinyal evaluasi yang menunjukkan kualitas *action* yang dipilih dan dapat bernilai positif, nol, maupun negatif.

Tujuan utama RL adalah mempelajari suatu *policy* yang mampu memaksimalkan *expected cumulative reward* atau *return*. Secara umum, *return* pada *timestep t* didefinisikan sebagai

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \tag{2.1}$$

dengan G_t menyatakan *return* pada *timestep t*, R_{t+k+1} merupakan *reward* yang diterima pada *timestep* berikutnya, dan $\gamma \in [0,1]$ merupakan *discount factor* yang mengatur kontribusi *reward* di masa depan terhadap nilai *return*. Dengan memaksimalkan nilai *return*, agen diharapkan mampu memperoleh strategi pengambilan keputusan yang semakin baik seiring berlangsungnya proses pembelajaran.



Gambar 2.1. Siklus interaksi umum pada *Reinforcement Learning*.

Gambar 2.1 menunjukkan siklus interaksi umum pada *reinforcement learning*. Pada setiap *timestep* t, lingkungan menyediakan *state* (S_t) kepada agen. Berdasarkan *policy* (π), agen memilih suatu *action* (A_t). Lingkungan kemudian memproses *action* tersebut dan menghasilkan *state* baru (S_{t+1}) beserta *reward* (R_{t+1}). Informasi tersebut digunakan oleh agen untuk memperbarui mekanisme pembelajarannya sehingga *policy* yang dimiliki mampu menghasilkan *return* yang semakin baik seiring berlangsungnya proses pembelajaran [9, 10].

2.2 Multi-Armed Bandit

Multi-Armed Bandit (MAB) merupakan salah satu formulasi permasalahan pengambilan keputusan sekuensial paling sederhana dalam *reinforcement learning*. Berbeda dengan RL secara umum, MAB tidak memodelkan transisi *state*, sehingga keputusan agen hanya didasarkan pada estimasi *reward* dari setiap lengan (*arm*) yang tersedia. Pada setiap *timestep*, agen harus memilih satu dari K buah *arms* yang masing-masing memiliki distribusi *reward* yang tidak diketahui. Tujuan utama agen adalah memperoleh *expected cumulative reward* sebesar mungkin melalui keseimbangan antara eksplorasi dan eksploitasi.

Secara matematis, performa suatu algoritma MAB umumnya dievaluasi menggunakan metrik *regret*. *Regret* menyatakan kerugian kumulatif akibat agen tidak selalu memilih lengan dengan *expected reward* tertinggi. Dalam satu episode sepanjang T *timesteps*, *regret* dirumuskan sebagai

$$R(T) = \sum_{t=1}^{T} (\mu^* - \mu_{a_t}), \tag{2.2}$$

dengan $\mu_a = \mathbb{E}[r \mid a]$. Notasi $\mathbb{E}[\cdot]$ menyatakan operator ekspektasi (*expectation*), yaitu nilai rata-rata teoritis dari suatu peubah acak. Dengan demikian, μ_a merepresentasikan *expected reward* yang diperoleh apabila agen memilih lengan a, sedangkan μ^* merupakan *expected reward* dari lengan optimal. Pada setiap *timestep* t, agen memilih suatu lengan a_t dan menerima *reward* r_t dari lingkungan.

Secara umum, tujuan MAB adalah menemukan *policy* yang memaksimalkan *expected cumulative reward*, yang secara ekuivalen dapat dipandang sebagai upaya meminimalkan *cumulative regret*. Hubungan tersebut dapat dinyatakan sebagai

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t=1}^{T} r_t\right]. \tag{2.3}$$

dengan π menyatakan suatu *policy* yang memetakan kondisi yang diamati agen ke suatu *action*, sedangkan π^* merupakan *optimal policy*, yaitu *policy* yang menghasilkan nilai *expected cumulative reward* maksimum. Notasi $\arg\max$ menunjukkan proses pencarian *policy* yang memberikan nilai objektif terbesar.

Kualitas suatu algoritma MAB sangat dipengaruhi oleh kemampuannya menangani dilema *exploration-exploitation*. *Exploration* merupakan strategi agen untuk mencoba lengan yang belum banyak dieksplorasi guna memperoleh informasi baru mengenai distribusi *reward*. Sebaliknya, *exploitation* merupakan strategi memilih lengan yang diperkirakan memiliki *expected reward* tertinggi berdasarkan informasi yang telah diperoleh sebelumnya. Eksplorasi yang berlebihan dapat menyebabkan agen sering memilih lengan yang kurang menguntungkan, sedangkan eksploitasi yang terlalu dini berisiko membuat agen terjebak pada solusi suboptimal karena gagal menemukan lengan yang sebenarnya lebih baik [11, 12, 13, 14].

2.3 Non-Stationary Multi-Armed Bandit

Pada MAB klasik, distribusi *reward* setiap lengan (*arm*) diasumsikan bersifat stasioner, sehingga *expected reward* dari masing-masing lengan tidak berubah sepanjang proses pembelajaran. Asumsi tersebut memungkinkan agen memanfaatkan seluruh riwayat interaksi untuk membangun estimasi *reward* yang semakin akurat seiring bertambahnya jumlah observasi.

Namun, pada berbagai permasalahan dunia nyata seperti *financial trading*, sistem rekomendasi, maupun pengawasan medis, karakteristik lingkungan dapat berubah seiring waktu. Perubahan tersebut menyebabkan distribusi *reward* dari setiap lengan tidak lagi konstan, sehingga lengan yang sebelumnya memberikan *expected reward* tertinggi belum tentu tetap menjadi pilihan terbaik pada *timestep* berikutnya. Untuk memodelkan kondisi tersebut digunakan kerangka *Non-Stationary Multi-Armed Bandit* (NS-MAB), yaitu perluasan dari MAB klasik yang memperbolehkan distribusi *reward* berubah selama proses pembelajaran berlangsung.

Perubahan distribusi *reward* menimbulkan tantangan karena estimasi yang dibangun dari seluruh data historis dapat menjadi tidak lagi representatif terhadap kondisi lingkungan saat ini. Akibatnya, agen cenderung mempertahankan estimasi berdasarkan informasi lama sehingga respons terhadap perubahan lingkungan menjadi lebih lambat. Oleh karena itu, algoritma yang dirancang untuk lingkungan non-stasioner umumnya memerlukan mekanisme adaptasi agar estimasi *reward* dapat mengikuti perubahan distribusi secara lebih cepat.

Berbagai pendekatan telah dikembangkan untuk menangani permasalahan tersebut, seperti penggunaan *sliding window*, *discount factor*, maupun mekanisme pelacakan internal yang secara bertahap mengurangi pengaruh informasi historis yang sudah tidak lagi relevan (*forgetting*) [15].

2.4 Emotional-Cognition Integration Architecture (ECIA)

Emotional-Cognition Integration Architecture (ECIA) [3] merupakan arsitektur *reinforcement learning* yang terinspirasi oleh mekanisme afektif dan kognitif pada otak mamalia. Arsitektur ini dirancang untuk meningkatkan kemampuan pengambilan keputusan adaptif pada lingkungan yang bersifat non-stasioner.

Berbeda dengan algoritma *Multi-Armed Bandit* (MAB) klasik yang tidak memelihara informasi internal, ECIA memiliki mekanisme pembelajaran internal yang diperbarui secara berkelanjutan, seperti sistem emosi (*emotion system*), *episodic memory*, dan modulasi pembelajaran berbasis dopamin. Oleh karena itu, ECIA lebih tepat dikategorikan sebagai arsitektur *reinforcement learning*. Meskipun demikian, pada penelitian [3], performa ECIA dievaluasi melalui kerangka permasalahan *Non-Stationary Multi-Armed Bandit* (NS-MAB), sehingga proses interaksi agen dengan lingkungan tetap mengikuti paradigma MAB.

Berbeda dengan pendekatan MAB konvensional yang umumnya mengandalkan mekanisme statistik, ECIA mengintegrasikan sinyal afektif, memori berbasis konteks, dan modulasi laju pembelajaran melalui unit pengambilan keputusan terpusat. Integrasi tersebut memungkinkan agen menyesuaikan keseimbangan antara eksplorasi dan eksploitasi secara adaptif ketika karakteristik lingkungan berubah.

2.4.1 Non-Stationary Five-Armed Bandit

Mengikuti lingkungan evaluasi yang digunakan oleh [3], agen ECIA berinteraksi dengan lingkungan *non-stationary five-armed bandit*. Setiap episode terdiri atas $T = 200$ *timesteps*. Pada setiap *timestep* t, lingkungan mendefinisikan nilai harapan (*expected reward*) $\mu_t(a)$ untuk masing-masing dari lima buah *arm*. Nilai $\mu_t(a)$ dapat berubah terhadap waktu sesuai karakteristik lingkungan, sehingga distribusi *reward* pada setiap *arm* juga berubah selama proses pembelajaran.

Agen kemudian memilih satu *action* berupa *arm* $a_t \in \{0,1,2,3,4\}$. Setelah *action* dipilih, lingkungan menghasilkan satu nilai *reward* secara stokastik hanya untuk *arm* yang dipilih tersebut.

$$r_t \sim \mathcal{N}\left(\mu_t(a_t), \sigma\right), \tag{2.4}$$

Dengan r_t menyatakan *reward* yang diterima agen pada *timestep* t, $\mu_t(a)$ sebagai *expected reward* dari *arm* a pada waktu t, dan σ menyatakan tingkat *environmental noise*. Nilai *reward* yang diterima agen diperoleh melalui proses sampling stokastik dari distribusi tersebut, sehingga nilainya tidak selalu sama dengan nilai harapan $(\mu_t(a_t))$.

Berbeda dengan lingkungan MAB stasioner, nilai $\mu_t(a)$ pada lingkungan non-stasioner berubah seiring waktu. Akibatnya, *arm* yang memberikan *expected reward* tertinggi pada suatu *timestep* belum tentu tetap menjadi *arm* terbaik pada *timestep* berikutnya. Perubahan inilah yang menjadi sumber utama ketidakstasioneran lingkungan dan menuntut agen untuk terus memperbarui estimasinya terhadap kualitas masing-masing *arm*.

Setelah menerima *reward*, agen ECIA memproses informasi tersebut melalui mekanisme internalnya untuk memperbarui estimasi, keadaan emosional, serta strategi pemilihan *action* pada *timestep* berikutnya. Berdasarkan karakteristik perubahan distribusi *reward*, lingkungan evaluasi pada penelitian [3] dibagi menjadi

tiga tipe, yaitu Environment A, Environment B, dan Environment C.

1. **Environment A (Sudden Strategy Reversal)**
   (a) **Mekanisme:** Lingkungan *piecewise-stationary* dengan satu titik perubahan (*change point*) yang menyebabkan *arm* optimal berpindah secara tiba-tiba.
   (b) **Tujuan Evaluasi:** Menguji kemampuan agen beradaptasi terhadap perubahan strategi yang mendadak ketika *arm* terbaik sebelumnya tidak lagi optimal.
2. **Environment B (Predictable Alternation)**
   (a) **Mekanisme:** Distribusi *reward* mengalami pergantian secara periodik sehingga dua *arm* optimal berosilasi mengikuti pola yang teratur.
   (b) **Tujuan Evaluasi:** Menguji kemampuan agen mengenali pola perubahan yang dapat diprediksi dan menyesuaikan strategi eksplorasi serta eksploitasi secara efisien.
3. **Environment C (Stochastic Disruptions)**
   (a) **Mekanisme:** Lingkungan *piecewise-stationary* dengan titik perubahan yang terjadi secara acak disertai lonjakan *reward* sementara (*reward spikes*).
   (b) **Tujuan Evaluasi:** Menguji ketahanan agen pada lingkungan yang sangat tidak terprediksi dengan perubahan *arm* optimal dan sinyal *reward* semu.

2.4.2 Konfigurasi Environment B

Penelitian ini menggunakan *Environment B* sebagai lingkungan evaluasi. Mengikuti konfigurasi yang digunakan oleh [3], lingkungan ini dirancang untuk menghasilkan pola perubahan *reward* yang bersifat periodik sehingga menguji kemampuan agen dalam mengenali pola (*pattern recognition*) dan menyesuaikan strategi pengambilan keputusan secara adaptif.

Karakteristik *Environment B* dirumuskan sebagai berikut.

1. Setiap 40 *timesteps*, *arm* optimal bergantian secara periodik antara *arm* 0 dan *arm* 4.

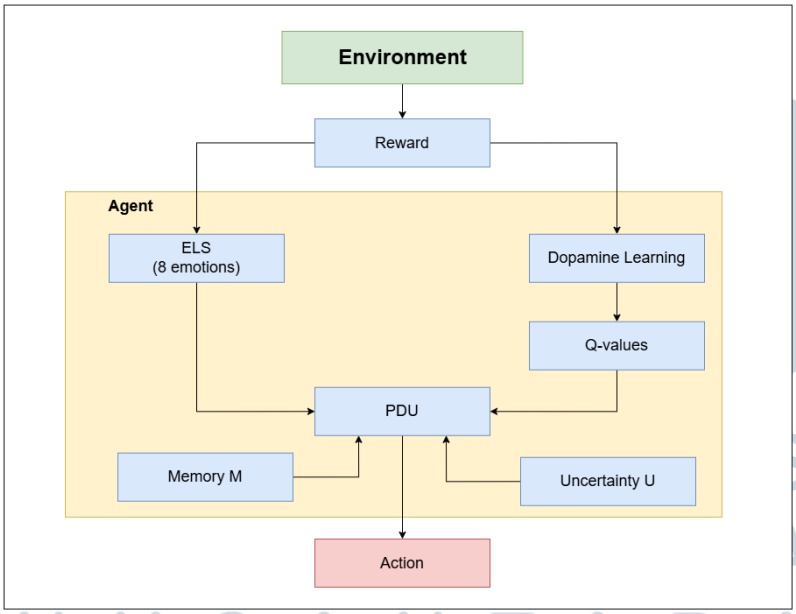
2. Nilai harapan (*expected reward*) dirancang kontras. *Arm* optimal memiliki nilai harapan sebesar $\mu = 0.95$, sedangkan *arm* yang berseberangan memiliki nilai harapan sebesar $\mu = 0.01$.

3. Tiga *arm* lainnya, yaitu *arm* 1, *arm* 2, dan *arm* 3, mempertahankan nilai harapan tetap sebesar $\mu = 0.05$ sepanjang episode.

4. Distribusi *reward* menggunakan *environmental noise* sebesar $\sigma = 0.05$, sehingga variasi *reward* relatif kecil dan pola pergantian *arm* optimal tetap dapat diamati secara jelas.

2.4.3 Komponen Utama Arsitektur ECIA

ECIA terdiri atas empat komponen utama yang saling berinteraksi untuk memproses informasi dari lingkungan dan menghasilkan keputusan tindakan secara adaptif, yaitu *External Limbic System* (ELS), *Hippocampus-Inspired Episodic Memory System*, *Dopamine-Inspired Adaptive Learning Module*, dan *Prefrontal Decision Unit* (PDU). Hubungan antar komponen beserta aliran informasi di dalam arsitektur ECIA ditunjukkan pada Gambar 2.2.



Gambar 2.2. Komponen utama dan aliran informasi dalam arsitektur ECIA

A *External Limbic System* (ELS)

External Limbic System (ELS) merupakan modul yang bertanggung jawab mengubah informasi *reward* menjadi representasi keadaan emosional agen. Modul ini terinspirasi oleh sistem limbik pada otak mamalia dan mempertahankan delapan dimensi emosi yang dikelompokkan ke dalam empat dimensi fungsional sebagaimana dirangkum pada Tabel 2.1.

Tabel 2.1. Dimensi Fungsional ECIA

No.	Functional dimension	Emotions
1	Valence	*Joy*, *Fear*.
2	Temporal	*Hope*, *Sadness*.
3	Exploration	*Curiosity*, *Anger*.
4	Self-Evaluation	*Pride*, *Shame*.

Berdasarkan *reward* yang diterima beserta riwayat performa agen, ELS memperbarui *emotion vector*. Nilai emosi tersebut selanjutnya digunakan untuk memodulasi sensitivitas risiko, perilaku eksplorasi, serta proses evaluasi aksi pada *Prefrontal Decision Unit* (PDU).

B Hippocampus-Inspired Episodic Memory System

Modul ini berfungsi menyimpan pengalaman-pengalaman penting yang dialami agen selama proses pembelajaran. Setiap pengalaman disimpan bersama informasi konteks, aksi yang dipilih, *reward*, serta keadaan emosional yang menyertainya. Ketika agen menghadapi situasi yang serupa pada masa berikutnya, memori tersebut dapat diambil kembali untuk memberikan bias terhadap proses evaluasi aksi sehingga keputusan yang dihasilkan tidak hanya bergantung pada *reward* saat ini, tetapi juga pada pengalaman yang relevan di masa lalu.

C Dopamine-Inspired Adaptive Learning Module

Modul ini terinspirasi oleh mekanisme dopaminergik pada otak mamalia yang mengodekan *reward prediction error*. Berbeda dengan pendekatan RL konvensional yang menggunakan *learning rate* tetap, ECIA menyesuaikan *learning rate* dasar secara adaptif berdasarkan besar *prediction error* dan kondisi emosional

agen. Mekanisme tersebut memungkinkan proses pembelajaran berlangsung lebih responsif terhadap perubahan distribusi *reward* pada lingkungan non-stasioner.

D *Prefrontal Decision Unit* (PDU)

Prefrontal Decision Unit (PDU) merupakan pusat pengambilan keputusan pada ECIA. Modul ini mengintegrasikan seluruh informasi yang dihasilkan oleh modul-modul sebelumnya, meliputi nilai aksi (*Q-values*), pengaruh emosional, kontribusi *episodic memory*, serta komponen eksplorasi. Integrasi tersebut menghasilkan skor evaluasi bagi setiap aksi yang tersedia. Secara matematis, aturan evaluasi aksi dinyatakan sebagai

$$A^* = \arg\max \left[Q(s,a) + \eta E(s,a) + M(s,a) + U(s,a) + \xi \right] \tag{2.5}$$

dengan $Q(s,a)$ merupakan nilai aksi yang dipelajari, $E(s,a)$ menyatakan kontribusi keadaan emosional, $M(s,a)$ menunjukkan pengaruh *episodic memory*, $U(s,a)$ merupakan komponen eksplorasi berbasis ketidakpastian (*uncertainty*), sedangkan ξ merupakan komponen eksplorasi acak (*random exploration*). Aksi dengan skor evaluasi tertinggi kemudian dipilih untuk dieksekusi pada *timestep* berikutnya.

2.4.4 Mekanisme Pengambilan Keputusan ECIA

Pada setiap *timestep*, ECIA menjalankan serangkaian proses yang membentuk siklus pengambilan keputusan adaptif. Alur tersebut dapat diringkas menjadi lima tahapan berikut.

1. **Environment Reward Observation**

   Agen mengeksekusi aksi yang telah dipilih pada *timestep* sebelumnya dan menerima *reward* dari lingkungan. Nilai *reward* tersebut menjadi masukan utama bagi seluruh modul internal ECIA.

2. **Emotion Update**

   External Limbic System mengevaluasi *reward* yang diterima beserta riwayat performa agen untuk memperbarui *emotion vector*. Keadaan emosional yang baru akan memengaruhi sensitivitas risiko dan kecenderungan eksplorasi agen pada proses pengambilan keputusan berikutnya.

3. **Learning and Memory Update**

   Agen menghitung *reward prediction error* berdasarkan perbedaan antara nilai yang diprediksi dan *reward* yang benar-benar diterima. Nilai tersebut digunakan untuk memperbarui *Q-values*, menyesuaikan *learning rate* adaptif, serta menentukan apakah pengalaman tersebut perlu disimpan ke dalam *episodic memory*.

4. **Action Evaluation**

   Prefrontal Decision Unit mengintegrasikan nilai aksi yang telah dipelajari, keadaan emosional, informasi dari *episodic memory*, dan komponen eksplorasi sesuai Persamaan 2.5. Setiap aksi kemudian memperoleh skor evaluasi.

5. **Action Selection**

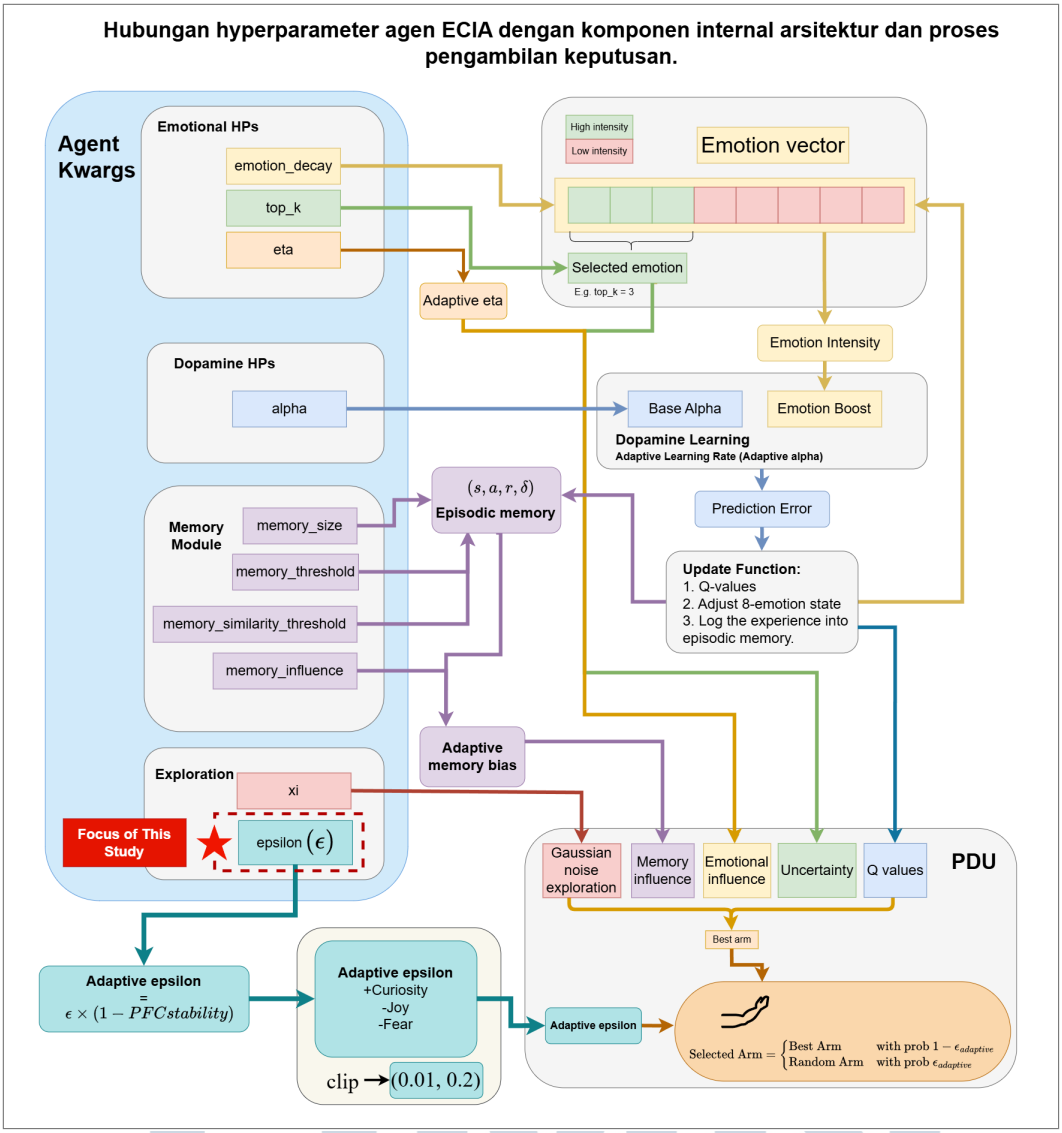
   Aksi dengan skor evaluasi tertinggi dipilih untuk dieksekusi pada *timestep* berikutnya. Setelah aksi dijalankan, lingkungan menghasilkan *reward* baru sehingga siklus pembelajaran kembali dimulai.

2.4.5 Hyperparameter ECIA

ECIA mengelompokkan parameter konfigurasi agen ke dalam sebuah *dictionary* `optimized_params` yang kemudian diteruskan sebagai `agent_kwargs` pada saat inisialisasi agen [16]. Parameter-parameter tersebut mengendalikan perilaku internal agen, seperti pembentukan emosi, mekanisme memori, pembelajaran adaptif, serta strategi eksplorasi.

Berbeda dengan parameter lingkungan, *hyperparameter* tersebut mendefinisikan karakteristik agen (*solution space*) sehingga nilainya ditentukan sebelum proses pelatihan dimulai.

Hubungan antara kelompok *hyperparameter* dengan modul-modul penyusun arsitektur ECIA ditunjukkan pada Gambar 2.3.



Gambar 2.3. Hubungan Hyperparameter Agen ECIA dengan Komponen Internal Arsitektur

Terlihat bahwa setiap kelompok parameter memengaruhi modul yang berbeda, namun seluruhnya berkontribusi terhadap proses pemilihan *action* pada *Prefrontal Decision Unit* (PDU). Dalam penelitian ini, fokus optimasi dibatasi pada *exploration coefficient* (ε).

A Exploration Coefficient (ε)

Parameter ε merupakan *hyperparameter* yang berfungsi sebagai *exploration coefficient*. Berbeda dengan pendekatan ε-*greedy* konvensional yang menggunakan probabilitas eksplorasi tetap, ECIA membentuk nilai *adaptive epsilon* yang

diperbarui pada setiap *timestep*. Proses pembentukan nilai tersebut mengikuti dua tahap.

Tahap pertama menghitung nilai eksplorasi dasar berdasarkan stabilitas *Prefrontal Decision Unit* (PDU):

$$\varepsilon' = \varepsilon\,(1 - \texttt{pfc\_stability}) \tag{2.6}$$

Selanjutnya, nilai tersebut dimodulasi menggunakan sinyal emosi dan dibatasi agar tetap berada pada rentang eksplorasi yang diizinkan.

$$\begin{aligned}\varepsilon_{\text{adaptive}} = \text{clip}\Big(&\varepsilon' + \texttt{curiosity\_boost} \\ &- \texttt{fear\_penalty} - \texttt{joy\_exploitation}, 0.01, 0.20\Big)\end{aligned} \tag{2.7}$$

Persamaan pertama menunjukkan bahwa nilai exploration coefficient ε dimodulasi oleh tingkat stabilitas kognitif agen sehingga menghasilkan *intermediate value* ε' Semakin tinggi nilai `pfc_stability`, semakin kecil nilai ε' yang dihasilkan.

Persamaan kedua memodulasi lebih lanjut nilai ε' menggunakan sinyal yang diturunkan dari keadaan emosional agen. Aktivasi *curiosity* meningkatkan nilai eksplorasi, sedangkan *fear* dan *joy* menurunkannya. Terakhir, fungsi clip$(\cdot)$ memastikan bahwa nilai akhir $\varepsilon_{\text{adaptive}}$ selalu berada pada rentang $[0.01, 0.20]$ sesuai implementasi ECIA.

2.4.6 Evaluation Metric

Penelitian [3] mengevaluasi performa agen menggunakan tiga metrik, yaitu *overall mean reward*, *overall recovery time*, dan *overall recovery rate*. Penelitian ini menggunakan *overall mean reward* sebagai nilai objektif (*objective value*) yang dimaksimalkan selama proses optimasi karena metrik tersebut secara langsung merepresentasikan rata-rata *reward* yang diperoleh agen selama proses pembelajaran.

Pada setiap *timestep* t, agen menerima *instantaneous reward* r_t dari lingkungan. Selama satu episode yang terdiri atas $T = 200$ *timesteps*, seluruh *reward* yang diterima dirata-ratakan sehingga menghasilkan *episode mean reward*

R_k, yang dirumuskan sebagai berikut:

$$R_k = \frac{1}{T} \sum_{t=1}^{T} r_t \tag{2.8}$$

Untuk mengurangi pengaruh variasi acak selama evaluasi, setiap konfigurasi *hyperparameter* dijalankan menggunakan 12 *random seeds* berdasarkan deret Fibonacci sebagaimana diterapkan oleh [3]. Pada setiap *seed*, agen dievaluasi selama 300 episode sehingga setiap konfigurasi menghasilkan total $N = 3600$ episode independen.

Nilai *episode mean reward* dari seluruh episode kemudian dirata-ratakan kembali untuk memperoleh *overall mean reward* $\bar{R}$, yang dirumuskan sebagai berikut:

$$\bar{R} = \frac{1}{N} \sum_{k=1}^{N} R_k \tag{2.9}$$

Dengan demikian, notasi *reward* yang digunakan dalam penelitian ini dapat diringkas sebagai berikut:

1. r_t : *instantaneous reward* yang diterima agen pada *timestep* ke-t,
2. R_k : *episode mean reward* pada episode ke-k,
3. $\bar{R}$: *overall mean reward* dari seluruh episode evaluasi.

Semakin besar nilai $\bar{R}$, semakin baik performa konfigurasi *hyperparameter* yang diuji. Oleh karena itu, *overall mean reward* digunakan sebagai fungsi objektif yang dimaksimalkan pada proses optimasi *hyperparameter* dalam penelitian ini.

2.4.7 Pentingnya *Hyperparameter Optimization*

Pemilihan nilai *hyperparameter* memiliki pengaruh yang signifikan terhadap performa, stabilitas, dan kemampuan adaptasi suatu model ML maupun RL. Konfigurasi yang kurang sesuai dapat menyebabkan proses pembelajaran menjadi tidak stabil, menghasilkan performa yang rendah, atau menghambat kemampuan agen dalam beradaptasi terhadap lingkungan yang berubah. Penelitian oleh [2] menunjukkan bahwa perubahan kecil pada nilai *hyperparameter* dapat menghasilkan perbedaan performa yang signifikan.

Secara tradisional, penentuan *hyperparameter* dilakukan melalui *manual tuning*, yaitu proses *trial-and-error* berdasarkan pengalaman peneliti, atau menggunakan metode pencarian seperti *grid search* dan *random search*. Meskipun sederhana, pendekatan tersebut membutuhkan biaya komputasi yang tinggi atau sangat bergantung pada intuisi pengguna sehingga sulit direproduksi.

Untuk mengatasi keterbatasan tersebut, digunakan pendekatan *Hyperparameter Optimization* (HPO), yaitu proses pencarian konfigurasi *hyperparameter* secara otomatis menggunakan algoritma optimasi. Melalui HPO, proses pencarian dapat dilakukan secara lebih sistematis dan efisien sehingga diharapkan menghasilkan konfigurasi dengan performa yang lebih baik dibandingkan penentuan secara manual.

2.5 Hyperparameter Optimization

2.5.1 Hyperparameter

Dalam *machine learning* (ML) dan *reinforcement learning* (RL), parameter yang digunakan oleh suatu algoritma secara umum dibedakan menjadi dua kategori, yaitu *model parameter* dan *hyperparameter*. *Model parameter* dipelajari secara otomatis selama proses pelatihan, sedangkan *hyperparameter* merupakan parameter konfigurasi yang harus ditentukan sebelum proses pelatihan dimulai.

Nilai *hyperparameter* memengaruhi perilaku proses pembelajaran, seperti laju pembelajaran, strategi eksplorasi, kapasitas memori, maupun mekanisme pengambilan keputusan. Oleh karena itu, konfigurasi *hyperparameter* yang berbeda dapat menghasilkan performa model yang berbeda meskipun algoritma dan data pelatihannya sama.

Berdasarkan tipe domain nilainya, *hyperparameter* umumnya dikelompokkan menjadi tiga jenis sebagai berikut.

1. **Continuous**, yaitu *hyperparameter* yang bernilai riil pada suatu rentang kontinu, misalnya nilai epsilon (ε).
2. **Discrete**, yaitu *hyperparameter* yang hanya dapat mengambil nilai diskret, seperti jumlah neuron atau ukuran memori.
3. **Categorical**, yaitu *hyperparameter* yang terdiri atas sejumlah pilihan kategori, misalnya jenis fungsi aktivasi atau algoritma optimasi.

2.5.2 Search Space

Proses *Hyperparameter Optimization* (HPO) dilakukan pada suatu ruang pencarian (*search space*) yang dinotasikan sebagai Λ. *Search space* mendefinisikan seluruh konfigurasi *hyperparameter* yang diperbolehkan untuk dievaluasi selama proses optimasi, termasuk tipe data, batas nilai, maupun distribusi sampling setiap *hyperparameter*.

Pemilihan *search space* yang sesuai berpengaruh terhadap efektivitas proses optimasi. Rentang pencarian yang terlalu sempit dapat menyebabkan konfigurasi terbaik tidak pernah dievaluasi, sedangkan ruang pencarian yang terlalu luas meningkatkan biaya komputasi karena algoritma optimasi harus mengeksplorasi lebih banyak konfigurasi yang berpotensi memberikan performa rendah [1].

Selain rentang pencarian, setiap *hyperparameter* juga dapat didefinisikan menggunakan skala sampling yang berbeda sesuai karakteristiknya [2].

1. ***Linear Scale***, digunakan ketika perubahan nilai *hyperparameter* diperkirakan memberikan perubahan performa yang relatif seragam pada seluruh rentang pencarian.

2. ***Logarithmic Scale***, digunakan untuk *hyperparameter* yang sensitivitasnya berbeda pada setiap orde magnitudo, sehingga proses sampling dilakukan secara proporsional terhadap perpangkatan tertentu, seperti basis 10.

2.5.3 Formulasi Matematis *Hyperparameter Optimization*

Tujuan utama *Hyperparameter Optimization* (HPO) adalah mencari konfigurasi *hyperparameter* optimal λ^* pada *search space* Λ sehingga performa algoritma menjadi optimal terhadap suatu fungsi objektif.

Secara umum, mengikuti formulasi yang digunakan oleh [1], diberikan konfigurasi *hyperparameter* $\lambda \in \Lambda$ dan fungsi objektif $c(\lambda)$, maka permasalahan HPO dapat dituliskan sebagai:

$$\lambda^* \in \arg\min_{\lambda \in \Lambda} c(\lambda) \tag{2.10}$$

Pada formulasi tersebut, $c(\lambda)$ merepresentasikan nilai evaluasi (*objective value*) yang dihasilkan setelah model dijalankan menggunakan konfigurasi λ. Fungsi ini diperlakukan sebagai *black-box* karena algoritma optimasi hanya

mengamati nilai akhirnya tanpa memerlukan bentuk analitik maupun informasi gradien.

Apabila fungsi objektif berupa metrik yang ingin dimaksimalkan, maka formulasi yang ekuivalen dapat dituliskan sebagai:

$$\lambda^* = \arg\max_{\lambda \in \Lambda} f(\lambda) \tag{2.11}$$

dengan $f(\lambda)$ menyatakan metrik performa yang ingin dimaksimalkan, misalnya *accuracy*, *return*, ataupun *overall mean reward*.

Karena penelitian ini bertujuan memaksimalkan *overall mean reward*, maka fungsi objektif yang dioptimalkan adalah

$$\lambda^* = \arg\max_{\lambda \in \Lambda} \bar{R}(\lambda) \tag{2.12}$$

dengan $\bar{R}(\lambda)$ merupakan *overall mean reward* yang diperoleh agen ECIA menggunakan konfigurasi *hyperparameter* λ.

Meskipun demikian, implementasi berbagai *framework* HPO modern umumnya mengubah permasalahan maksimasi menjadi minimasi melalui transformasi sederhana, misalnya

$$c(\lambda) = -\bar{R}(\lambda), \tag{2.13}$$

sehingga

$$\arg\max_{\lambda \in \Lambda} \bar{R}(\lambda) = \arg\min_{\lambda \in \Lambda} [-\bar{R}(\lambda)]. \tag{2.14}$$

Transformasi tersebut tidak mengubah solusi optimum, melainkan hanya mengubah representasi matematis dari permasalahan optimasi. Oleh karena itu, algoritma HPO dapat tetap menggunakan kerangka minimasi tanpa memengaruhi konfigurasi *hyperparameter* optimal yang diperoleh.

2.5.4 Hyperparameter Optimization sebagai Black-Box Optimization

Permasalahan *Hyperparameter Optimization* (HPO) umumnya dipandang sebagai permasalahan *black-box optimization*. Pada paradigma ini, algoritma optimasi tidak memerlukan pengetahuan mengenai mekanisme internal model yang dioptimasi. Sebaliknya, algoritma hanya mengusulkan konfigurasi *hyperparameter*, menjalankan model menggunakan konfigurasi tersebut, kemudian menerima nilai *objective function* sebagai umpan balik untuk menentukan konfigurasi berikutnya.

Dalam penelitian ini, model yang diperlakukan sebagai *black box* adalah agen ECIA. Untuk setiap konfigurasi *hyperparameter* λ, agen ECIA dijalankan pada lingkungan *Non-Stationary 5-Armed Bandit* dan menghasilkan nilai *overall mean reward* sebagaimana telah dijelaskan pada Subbab 2.4.5. Nilai tersebut kemudian dikembalikan kepada algoritma HPO sebagai satu-satunya informasi yang digunakan untuk mengevaluasi kualitas konfigurasi λ.

Secara umum, proses optimasi berlangsung melalui tahapan sebagai berikut.

1. Algoritma HPO mengusulkan sebuah konfigurasi *hyperparameter* kandidat $\lambda \in \Lambda$.
2. Agen ECIA dijalankan menggunakan konfigurasi tersebut pada lingkungan evaluasi.
3. Setelah seluruh proses evaluasi selesai, diperoleh nilai *objective function*, yaitu *overall mean reward*.
4. Nilai tersebut dikembalikan kepada algoritma HPO sebagai umpan balik untuk menentukan konfigurasi kandidat berikutnya.

Proses tersebut diulang hingga anggaran evaluasi (*evaluation budget*) terpenuhi. Selama proses optimasi, algoritma HPO tidak memanfaatkan informasi mengenai struktur internal ECIA, bentuk analitik fungsi objektif, maupun informasi gradien. Seluruh keputusan optimasi hanya didasarkan pada pasangan masukan berupa konfigurasi *hyperparameter* dan keluaran berupa nilai *objective function*. Oleh karena itu, optimasi *hyperparameter* pada penelitian ini termasuk ke dalam kategori *expensive black-box optimization*.

2.6 Library Optuna

Optuna merupakan *framework* Python *open-source* yang dirancang untuk melakukan *Hyperparameter Optimization* (HPO) secara otomatis pada berbagai model *machine learning*. Optuna menyediakan mekanisme untuk mendefinisikan *search space*, mengevaluasi konfigurasi *hyperparameter*, serta mengelola proses optimasi secara sistematis.

Salah satu karakteristik utama Optuna adalah paradigma *Define-by-Run*, yaitu pendekatan di mana *search space* didefinisikan secara dinamis selama program dijalankan menggunakan sintaks Python, bukan melalui konfigurasi statis di awal proses. Pendekatan ini memberikan fleksibilitas yang tinggi dalam mendefinisikan ruang pencarian yang bergantung pada kondisi tertentu.

Dalam Optuna, keseluruhan proses optimasi direpresentasikan sebagai sebuah *study*, sedangkan setiap evaluasi terhadap satu konfigurasi *hyperparameter* disebut sebagai sebuah *trial*. Pemilihan konfigurasi berikutnya dilakukan oleh suatu *sampler*. Pada penelitian ini digunakan *TPESampler*, yaitu *sampler* bawaan Optuna yang mengimplementasikan algoritma *Tree-structured Parzen Estimator* (TPE) [17].

2.7 *Tree-structured Parzen Estimator* (TPE)

Tree-structured Parzen Estimator (TPE) merupakan salah satu algoritma *Bayesian Optimization* yang banyak digunakan dalam permasalahan *Hyperparameter Optimization*. Berbeda dengan pendekatan *Bayesian Optimization* konvensional yang secara langsung memodelkan distribusi probabilitas fungsi objektif $p(y|x)$, TPE memodelkan distribusi kandidat *hyperparameter* berdasarkan hasil evaluasi historis melalui distribusi $p(x|y)$ dan $p(y)$.

Pada konteks penelitian ini, variabel x merepresentasikan konfigurasi *hyperparameter* $\lambda \in \Lambda$, sedangkan variabel y menyatakan nilai *objective function*, yaitu *overall mean reward* yang diperoleh setelah agen ECIA dievaluasi menggunakan konfigurasi tersebut. Dengan demikian, proses optimasi dilakukan pada ruang pencarian *hyperparameter* Λ yang telah didefinisikan sebelumnya.

Dalam implementasinya, distribusi probabilitas tersebut perlu diestimasi dari data historis hasil evaluasi. Optuna mengimplementasikan estimasi ini menggunakan *Gaussian Mixture Model* (GMM), yang berfungsi sebagai pendekatan terhadap *Kernel Density Estimation* (KDE) untuk membentuk model

distribusi pada setiap dimensi *hyperparameter*.

Seluruh hasil evaluasi historis kemudian dipisahkan ke dalam dua kelompok berdasarkan suatu nilai ambang y^*. Konfigurasi yang menghasilkan nilai fungsi objektif lebih baik daripada ambang dimasukkan ke dalam kelompok *good trials*, sedangkan sisanya dimasukkan ke dalam kelompok *bad trials*. Distribusi tersebut dinyatakan sebagai:

$$p(x|y) = \begin{cases} \ell(x), & \text{if } y < y^* \\ g(x), & \text{if } y \geq y^* \end{cases} \tag{2.15}$$

di mana $\ell(x)$ merepresentasikan distribusi konfigurasi *hyperparameter* yang menghasilkan performa lebih baik, sedangkan $g(x)$ merepresentasikan distribusi konfigurasi dengan performa yang relatif lebih rendah.

Nilai ambang y^* ditentukan menggunakan suatu kuantil γ dari seluruh nilai fungsi objektif yang telah diamati, sehingga berlaku:

$$p(y < y^*) = \gamma \tag{2.16}$$

Setelah kedua distribusi tersebut dibangun, TPE menghasilkan kandidat konfigurasi *hyperparameter* berikutnya dengan memilih konfigurasi yang memaksimalkan rasio:

$$\frac{\ell(x)}{g(x)} \tag{2.17}$$

Konfigurasi dengan nilai rasio yang tinggi memiliki probabilitas lebih besar untuk menghasilkan nilai fungsi objektif yang baik berdasarkan data historis. Setelah setiap *trial* selesai dievaluasi, distribusi $\ell(x)$ dan $g(x)$ diperbarui menggunakan seluruh hasil evaluasi yang telah tersedia, sehingga proses pencarian berikutnya dapat memanfaatkan informasi dari *trial* sebelumnya. Dengan mekanisme tersebut, TPE mampu mengarahkan proses pencarian menuju wilayah *search space* yang lebih menjanjikan dibandingkan metode pencarian acak [18, 19].

Pada implementasi Optuna, *TPESampler* menggunakan fungsi γ bawaan yang didefinisikan sebagai

$$\gamma(n) = \min(\lceil 0.1n \rceil, 25) \tag{2.18}$$

dengan n menyatakan jumlah *trial* yang telah selesai dievaluasi. Persamaan tersebut menunjukkan bahwa sekitar 10% konfigurasi terbaik, dengan batas maksimum 25 *trial*, digunakan untuk membangun distribusi $\ell(x)$ sebagai representasi wilayah pencarian yang menjanjikan [20, 21].

Secara umum, mekanisme TPE dapat dirangkum sebagai berikut:

1. Mengevaluasi sejumlah konfigurasi *hyperparameter*.
2. Memisahkan hasil evaluasi menjadi kelompok *good trials* dan *bad trials*.
3. Membangun distribusi probabilitas $\ell(x)$ dan $g(x)$ berdasarkan data historis.
4. Menghasilkan kandidat konfigurasi baru dengan memaksimalkan rasio $\ell(x)/g(x)$.
5. Mengevaluasi kandidat tersebut, memperbarui distribusi probabilitas, dan mengulangi proses hingga anggaran evaluasi terpenuhi.