DRL Approach : giải thích kiến thức

1.NOMA : Non-Orthogonal Multiple Access

- Đây là **kỹ thuật truy nhập vô tuyến phi trực giao**, được nghiên cứu nhiều trong **5G và 6G**.

- Khác với **OMA (Orthogonal Multiple Access)** như TDMA, FDMA, OFDMA, nơi mỗi người dùng chiếm một kênh riêng (không chồng lấn), thì **NOMA cho phép nhiều người dùng chia sẻ cùng một tài nguyên tần số / thời gian**, nhờ vào kỹ thuật phân tách tín hiệu ở phía thu.

Tức là ở các bài báo trước chia time slot thì sẽ là TDMA thuộc OMA , thì NOMA sẽ cho phép trong cùng 1 khung thời gian có nhiều người dung có thể cùng truy cập sử dụng , 

+ Các user **có thể truyền cùng lúc trong cùng một time slot**.

+UAV nhận được **tín hiệu chồng lấn** và sử dụng **SIC (Successive Interference Cancellation)** để giải mã từng user.

Điểm khác biệt chính:

* **OMA**: người dùng A và B sẽ được cấp **tài nguyên riêng biệt** → không giao thoa.
* **NOMA**: người dùng A và B cùng **truyền trong một kênh**, nhưng được phân biệt bằng **công suất và kỹ thuật giải mã**.

**Mô hình hoạt động của NOMA**

Có hai thành phần chính:

**a) Bên phát (Base Station / UAV / MEC server)**

* Sử dụng **Superposition Coding** để xếp chồng tín hiệu nhiều người dùng.
* Ví dụ: tín hiệu của User 1 và User 2 sẽ được cộng lại thành một tín hiệu hỗn hợp, nhưng phân bổ **công suất khác nhau**.

**b) Bên thu (người dùng / thiết bị di động)**

* Sử dụng **Successive Interference Cancellation (SIC)** để giải mã.
* Cách giải:
  1. Người dùng có kênh “tốt hơn” (gần trạm) giải mã tín hiệu mạnh trước, sau đó loại bỏ nó khỏi tín hiệu hỗn hợp.
  2. Tiếp tục giải mã tín hiệu còn lại.
  3. Người dùng ở xa hơn thì nhận trực tiếp tín hiệu đã được phân bổ công suất lớn hơn.

**Ý tưởng chính của bài báo**

1. **Vấn đề**:
   * VEC cần xử lý nhiều tác vụ với yêu cầu thời gian thực.
   * Hạn chế: tài nguyên biên (edge) hữu hạn, nhiễu giao tiếp lớn, nhu cầu kết nối đông.
2. **Đóng góp**:
   * Đề xuất kiến trúc **NOMA-based VEC** → cho phép nhiều phương tiện cùng chia sẻ tài nguyên truyền thông, cải thiện hiệu quả phổ và giảm độ trễ.
   * Xây dựng **mô hình truyền V2I** tính cả nhiễu trong-biên và giữa các biên.
   * Tối ưu hóa hợp tác tài nguyên (CRO): kết hợp **offloading quyết định** + **phân bổ tài nguyên**.
3. **Phương pháp**:
   * **Offloading** → mô hình hóa như **trò chơi tiềm năng chính xác (EPG)**, giải bằng **MAD4PG (thuật toán học tăng cường đa tác tử)**.
   * **Phân bổ tài nguyên** → chia thành các bài toán lồi, giải bằng **gradient iterative + KKT**.
4. **Kết quả**:
   * Mô phỏng dựa trên dữ liệu thực tế, chứng minh giải pháp đề xuất vượt trội hơn so với baseline.

**1. Khái niệm V2I**

* **V2I (Vehicle-to-Infrastructure)** là loại kênh truyền trong mạng **vehicular network**.
* Nó mô tả việc **xe (vehicle)** kết nối và trao đổi dữ liệu với **hạ tầng (infrastructure)** như:
  + Trạm gốc (BS – Base Station)
  + Road Side Unit (RSU – thiết bị ven đường)
  + Hoặc **UAV acting as edge server**

👉 Đây là đường truyền chính để xe **offload dữ liệu** hoặc **nhận kết quả xử lý** từ edge/cloud.

**🔹 2. Đặc điểm của kênh V2I**

1. **Đặc thù di động cao**:
   * Xe di chuyển nhanh → gây **Doppler effect** (dịch tần).
   * Tín hiệu thay đổi nhanh → kênh fading ngắn hạn.
2. **Môi trường phức tạp**:
   * Đường phố nhiều vật cản (nhà cao tầng, cây cối).
   * Kênh thường phân loại:
     + **LoS (Line-of-Sight)**: có tầm nhìn trực tiếp đến RSU/UAV.
     + **NLoS (Non-Line-of-Sight)**: bị chắn, phải phản xạ, khúc xạ.
3. **Mô hình kênh phổ biến**:
   * **Path loss** (suy hao theo khoảng cách):

PL(d)=PL(d0)+10αlog⁡10(dd0)+XσPL(d) = PL(d\_0) + 10\alpha \log\_{10}\left(\frac{d}{d\_0}\right) + X\_\sigmaPL(d)=PL(d0​)+10αlog10​(d0​d​)+Xσ​

* + - α\alphaα: hệ số suy hao đường truyền.
    - XσX\_\sigmaXσ​: shadowing (nhiễu log-normal).
  + **Small-scale fading**: thường dùng **Rayleigh** (NLoS) hoặc **Rician** (LoS).

1. **Băng thông & tốc độ**:
   * V2I thường triển khai trong **5G NR-V2X** hoặc **IEEE 802.11p (DSRC)**.
   * Hỗ trợ tốc độ cao để phục vụ ứng dụng thời gian thực (video, radar, AI).

**🔹 3. V2I trong VEC / UAV-MEC**

Trong bài toán **Vehicular Edge Computing (VEC)** hay **UAV-assisted MEC**:

* **Xe → V2I link → RSU / UAV / MEC server**
  + Xe gửi dữ liệu (ảnh, sensor, thông tin cảm biến) để offload tác vụ.
* **Server xử lý → trả kết quả qua V2I**
  + Ví dụ: cảnh báo nguy hiểm, bản đồ giao thông, kết quả AI.

👉 Hiệu năng của hệ thống phụ thuộc mạnh vào **chất lượng kênh V2I**:

* Nếu kênh yếu → độ trễ offloading cao, tỷ lệ mất gói tăng.
* Nếu có NOMA → nhiều xe có thể dùng **cùng kênh V2I**, tăng hiệu quả phổ.

**🔹 4. Ví dụ thực tế**

* Xe chạy trên cao tốc, có **RSU cách 500m**.
* Xe muốn offload ảnh camera 10 MB lên MEC server đặt tại RSU.
* Kênh V2I LoS, mô hình Rician fading.
* Với **OMA (TDMA)** → mỗi xe gửi lần lượt.
* Với **NOMA** → nhiều xe có thể gửi **đồng thời trên cùng V2I channel**, RSU giải mã bằng SIC → giảm độ trễ toàn hệ thống.

**🔹 1. Bài toán offloading**

* **Offloading = quyết định có offload tác vụ không, và offload đến đâu.**
* Trong VEC / UAV–MEC: mỗi xe (hoặc thiết bị) có một tác vụ tính toán. Nó có thể:
  1. **Local computing**: xử lý ngay trên xe (ít trễ truyền nhưng hạn chế tài nguyên).
  2. **Offload lên edge node (RSU/UAV)**: cần truyền qua kênh V2I, nhưng xử lý nhanh hơn.
  3. **Offload sang edge khác / cloud**: nếu edge gần bị quá tải, có thể offload sang edge khác → nhưng có thêm chi phí truyền dẫn liên-edge.

👉 Bài toán “offloading” trong bài báo chính là **tìm chính sách (policy) cho từng xe: offload hay không, offload đến đâu**, sao cho tối ưu (ví dụ: giảm trễ, tăng tỷ lệ hoàn thành task trước deadline).

**🔹 2. Bài toán resource allocation**

* Sau khi biết **xe nào offload đến đâu**, cần **phân bổ tài nguyên truyền thông và tính toán**.
* Bao gồm:
  1. **Tài nguyên truyền thông (wireless resources)**:
     + Công suất phát (power allocation).
     + Băng thông hoặc time slot (nếu dùng OMA).
     + Thứ tự giải mã & công suất (nếu dùng NOMA).
  2. **Tài nguyên tính toán (computing resources)** tại edge/UAV:
     + CPU cycles cho từng tác vụ.
     + Chia sẻ GPU/FPGA nếu có.

👉 Nói gọn: “resource allocation” = **kế hoạch phân bổ tài nguyên vô tuyến + tài nguyên tính toán** cho các tác vụ đã được quyết định offload.

\*Potential Game

1. Ký hiệu và Trong lý thuyết trò chơi (game theory):

= action (chiến lược/hành động) của người chơi . = tập hợp các action của tất cả những người chơi khác trừ .

Ví dụ: nếu có 3 người chơi, thì:

Như vậy, chiến lược toàn cục được viết là:

tức là tập hợp mọi hành động của tất cả người chơi.

🔹 2. Vai trò trong potential game Trong một potential game, ta có:

= hàm lợi ích (utility) của người chơi , phụ thuộc cả vào hành động của chính nó và hành động của các người chơi khác.

Hành động của chính nó: → xe i quyết định có offload không, offload đến đâu. Hành động của người khác: → những xe khác quyết định offload thế nào, chiếm tài nguyên ra sao, gây nhiễu như thế nào. 👉 Như vậy, chính là tác động từ môi trường xung quanh mà mỗi xe phải cân nhắc.

🔹 3. Vai trò trong bài toán offloading VEC Trong bài báo:

Người chơi = các xe (vehicles). Chiến lược = quyết định offloading của xe i (xử lý local, offload lên edge nào, công suất truyền). Chiến lược = quyết định của tất cả xe còn lại. Ví dụ:

Nếu xe khác đều offload lên RSU A, thì RSU A bị quá tải, offloading của xe i vào RSU A sẽ có utility thấp. Ngược lại, nếu nhiều xe chọn offload phân tán (một số lên UAV, một số xử lý local), thì utility của xe i khi chọn RSU A có thể cao hơn. 👉 Vậy:

= quyết định của bản thân. = “ngữ cảnh” hay “điều kiện môi trường” do các xe khác tạo ra. Kết hợp → xác định utility của xe i. 🔹 4. Ví dụ đơn giản minh họa Giả sử có 2 xe (2 người chơi), mỗi xe có 2 lựa chọn:

: xử lý local.

: offload lên RSU.

Nếu cả hai cùng offload (): RSU bị tắc → utility mỗi xe = thấp.

Nếu một offload, một local: xe offload được lợi (utility cao).

Nếu cả hai local: utility trung bình (không quá tốt vì xử lý chậm).

Ở đây:

quyết định của xe 1, nhưng utility còn phụ thuộc vào . Tương tự cho xe 2.

✅ Tóm lại:

= action/chiến lược của xe . = tập hợp các action của tất cả xe khác. Trong potential game, utility của xe i luôn phụ thuộc vào cả và .

Trong bài toán offloading VEC, chính là policy offloading của từng xe, còn chính là quyết định offloading của các xe khác, ảnh hưởng đến nhiễu và phân chia tài nguyên.

# 1) Exact Potential Game (EPG) là gì?

**Định nghĩa (trực giác):** Một trò chơi là **exact potential game** nếu tồn tại một hàm **potential** (phụ thuộc vào cấu hình chiến lược ) sao cho **mỗi khi một người chơi i đổi chiến lược đơn phương**, **mức thay đổi lợi ích của họ** bằng **mức thay đổi của** :

(Nếu làm theo hướng **tối thiểu hoá chi phí** , ta có phiên bản tương tự với dấu bằng cho **chi phí**: với một potential cho chi phí.)

**Nôm na:** khi mỗi “tay chơi” tự tối ưu lợi ích của mình bằng đổi chiến lược, thì **hàm**  cũng tăng (nếu tối đa hoá) hoặc giảm (nếu tối thiểu hoá) **một lượng y hệt**. Thế nên, **hành vi cục bộ** đẩy cả hệ thống **tiến về một điểm tốt** theo .

### Vì sao hay dùng trong offloading?

* **Bảo đảm tồn tại nghiệm cân bằng thuần (pure-strategy NE).**
* **Best-response / cải thiện dần** (mỗi xe tự đổi lựa chọn offload nếu có lợi) **hội tụ** tới một điểm dừng (cực trị địa phương của ), tương ứng **Nash equilibrium**.
* Cho phép **tối ưu phân tán**: mỗi xe ra quyết định dựa trên “lợi ích riêng”, nhưng vẫn **ăn khớp** với một **mục tiêu hệ thống** (ví dụ tối đa tỉ lệ phục vụ, tối thiểu tổng độ trễ).

## Ví dụ cực đơn giản (để “thấy” EPG)

Giả sử **2 thiết bị** chọn **kênh 1** hay **kênh 2**.

* **Thông lượng hệ thống** (tổng throughput) là **hàm potential** .
* Mỗi thiết bị cũng “quan tâm” đúng **tổng throughput** (trò chơi “lợi ích đồng nhất”):

Nếu thiết bị 1 đổi kênh làm tổng throughput tăng 0.8 Mbps, thì lợi ích của thiết bị 1 cũng **tăng đúng 0.8**. Vậy → **exact potential**.

Đây là ví dụ “trơn tru” nhất để cảm khái niệm: **một đổi nhỏ cục bộ** = **một đổi y hệt** ở “mục tiêu chung”.

## Ví dụ gần với offloading (kiểu **congestion game**)

* Có **1 RSU/UAV** làm điểm offload chung và **1 lựa chọn local**.
* Nếu **m** xe cùng offload lên RSU, **độ trễ offload** cho mỗi xe là (tăng theo m vì nhiễu/tắc nghẽn).
* **Độ trễ local** là hằng số .

Ta xét **bài toán tối thiểu hoá chi phí** (độ trễ). Đặt **potential cho chi phí** kiểu Rosenthal:

Nếu một xe **chuyển từ local → offload**, tổng chi phí hệ thống đổi:

còn **chi phí của chính xe đó** cũng đổi **đúng bằng** . ⇒ → **exact potential (phiên bản “cost”)**. Hệ quả: nếu mỗi xe **giảm chi phí riêng** bằng best response, thì cũng **giảm cùng lượng**, nên **hội tụ**.

Trong các bài VEC/UAV-MEC, khi mô hình hoá offloading như **congestion game** (chi phí tăng theo số offloader trên một liên kết/kênh), ta thường được **EPG** “miễn phí”.

# 2) Nash Equilibrium (NE) là gì?

**Định nghĩa:** Một cấu hình chiến lược $\mathbf{a}^\\*$ là **Nash equilibrium** nếu **không người chơi nào** có thể **đơn phương đổi chiến lược** để **cải thiện lợi ích** (hoặc giảm chi phí) của mình:

$$u\_i(a\_i^\\*, a\_{-i}^\\*) \;\ge\; u\_i(a\_i, a\_{-i}^\\*) \quad \forall i,\;\forall a\_i.$$

(Cho bản tối thiểu chi phí thì “” tương ứng.)

**Trực giác:** tại NE, **ai đổi riêng cũng không có lợi thêm** → hệ ở trạng thái “tự khoá”.

### Quan hệ giữa **EPG** và **NE**

* Trong **EPG**, **mọi cực trị địa phương** của potential (cực đại nếu tối đa hoá / cực tiểu nếu tối thiểu hoá) đều là **pure NE**.
* **Best-response dynamics** (mỗi lượt 1 xe đổi theo lợi ích riêng) **phải dừng** tại một điểm không ai muốn đổi tiếp → đó chính là **NE**.
* Vì vậy, mô hình hóa offloading là **EPG** giúp đảm bảo **tồn tại NE** và **dễ tìm** bằng các phương pháp phân tán/học tăng cường (ví dụ **MAD4PG** trong bài báo).

## Ví dụ nhỏ minh hoạ NE (tiếp nối congestion)

Dùng thiết lập trên (chi phí offload tăng theo m, chi phí local cố định).

* Nếu nhưng :
  + Khi **chưa ai offload**, một xe **muốn** offload (rẻ hơn local).
  + Khi đã có **1 xe offload**, xe thứ hai **không muốn** offload (vì đắt hơn local).
  + Cấu hình **1 offload + 1 local** là **NE**:
    - Người đang offload không muốn về local (vì ).
    - Người đang local không muốn sang offload (vì ).

Đây chính là “điểm dừng” tự nhiên mà **best-response** sẽ đạt tới.

# 3) Vai trò thực tế trong bài bạn đang đọc

* **Offloading** được mô hình thành **EPG** → đảm bảo có **NE** và có thể dùng **thuật toán phân tán** (mỗi xe là một agent).
* **MAD4PG** (multi-agent, distributional, DDPG) đóng vai trò **học chính sách best-response “mềm”** để **đi tới NE** trong môi trường nhiễu/động (V2I thay đổi, nhiễu intra/inter-edge, NOMA làm thay đổi can nhiễu & công suất…).
* Nhờ đặc tính **potential**, việc tối ưu cục bộ của từng xe **không phá hỏng** mục tiêu toàn cục (ví dụ **tối đa hoá service ratio / tối thiểu trễ hệ thống**).

## Tóm tắt nhanh

* **EPG**: có hàm sao cho **đổi chiến lược đơn phương** của người chơi làm **lợi ích của họ** đổi **bằng đúng** mức đổi của . → Đảm bảo **pure NE** tồn tại, **best-response hội tụ**.
* **NE**: trạng thái không ai muốn đổi chiến lược một mình.
* **Trong offloading VEC/UAV-MEC**: mô hình congestion ⇒ **EPG**; học phân tán (MAD4PG) ⇒ **tiệm cận NE**; phù hợp bối cảnh **NOMA** vì can nhiễu/phân bổ công suất làm chi phí offload phụ thuộc số người cùng truy cập.

Nếu bạn muốn, mình có thể **vẽ sơ đồ “best-response → tăng potential → dừng ở NE”** hoặc dựng **một bảng payoff nhỏ** để thấy các bước chuyển động cụ thể.

 **EPG là mô hình game đặc biệt** giúp ta chắc chắn có Nash equilibrium.

 **Nash equilibrium là trạng thái ổn định (lời giải)**.

 **Thuật toán học (như MAD4PG)** là **công cụ để tìm Nash** trong EPG.

## 🔹 1. MA-D4PG (Multi-Agent Distributed Distributional Deep Deterministic Policy Gradient)

* **Cơ sở:**
  + **DDPG**: thuật toán RL cho bài toán hành động liên tục (offloading tỷ lệ, phân bổ tài nguyên…).
  + **D4PG** (Distributional DDPG): cải tiến bằng cách học **phân phối của return** thay vì chỉ giá trị kỳ vọng. Điều này giúp chính sách ổn định và học tốt hơn.
  + **MA-D4PG**: mở rộng cho **nhiều agent** (multi-agent), mỗi xe hoặc mỗi node là 1 agent, cùng học nhưng **phân tán (distributed)**.
* **Ý nghĩa trong bài báo:**
  + Mỗi xe (agent) phải quyết định có offload hay không, hoặc offload bao nhiêu → hành động liên tục.
  + Mỗi agent **không thể biết chính xác hành động của các agent khác**, nhưng vì game là **EPG**, nên các agent cùng học sẽ dần hội tụ về Nash equilibrium.
  + MA-D4PG giúp:
    - Học trong môi trường nhiều agent.
    - Xử lý chính sách liên tục.
    - Học nhanh và ổn định hơn DDPG thường.

## 🔹 2. KKT (Karush-Kuhn-Tucker) conditions

* **Cơ sở:**
  + Là điều kiện cần và đủ để tìm nghiệm tối ưu cho **bài toán tối ưu lồi (convex optimization)** có ràng buộc.
  + Giống như mở rộng của đạo hàm = 0 trong tối ưu không ràng buộc.
* **Công thức tổng quát:** Với bài toán tối ưu:
* KKT conditions gồm:
  1. **Stationarity**: .
  2. **Primal feasibility**: .
  3. **Dual feasibility**: .
  4. **Complementary slackness**: .
* **Ý nghĩa trong bài báo:**
  + Sau khi tách bài toán lớn thành 2 subproblems:
    1. **Offloading** (game, giải bằng MA-D4PG).
    2. **Resource allocation** (phân bổ công suất/tần số cho truyền tải V2I).
  + Resource allocation là một **bài toán tối ưu lồi**, có thể giải bằng KKT và gradient descent.
  + Nhờ KKT, ta tìm được nghiệm tối ưu chính xác (không cần học).

## 🔹 Tổng kết liên hệ bài báo

* **Task offloading:** mô hình thành **Exact Potential Game**, giải bằng **MA-D4PG** để tìm Nash equilibrium.
* **Resource allocation:** mô hình thành **bài toán tối ưu lồi**, giải bằng **KKT + gradient-based method** để tìm nghiệm tối ưu.

👉 Một cái là **học (RL)**, một cái là **giải tích (optimization)**.

4. Vấn đề bài toán :

**1. NOMA thì không cần chia time slot?**

* **NOMA (Non-Orthogonal Multiple Access)** cho phép **nhiều người dùng truyền đồng thời trên cùng một tài nguyên vô tuyến** (cùng tần số, cùng time slot), nhờ vào kỹ thuật **SIC (Successive Interference Cancellation)** ở bộ thu.
* Do đó, lý thuyết thì **NOMA không cần tách người dùng bằng time slot hay subcarrier** như trong OMA.

**2. Vậy tại sao trong bài báo vẫn chia time slot?**

Trong các bài toán **Vehicular Edge Computing (VEC)** hoặc **UAV-MEC**, người ta hay chia **một khung thời gian (time frame)** thành **các time slot nhỏ** để:

* **Mô hình hóa quá trình offloading và truyền dữ liệu tuần tự**:  
  Ví dụ, trong một time slot, xe gửi task lên MEC server; slot sau MEC xử lý; slot tiếp theo gửi kết quả trả lại.
* **Đảm bảo phân tích độ trễ**: MEC system thường cần giới hạn delay → việc chia slot giúp mô hình hóa chính xác delay/latency.
* **Hỗ trợ resource allocation**: Dù NOMA cho phép truyền đồng thời, nhưng **khi có nhiều xe và nhiều edge server** → người ta vẫn chia slot để:
  + Lập lịch người nào được phục vụ trong giai đoạn nào.
  + Kết hợp **NOMA trong từng slot** (ví dụ: trong cùng 1 slot, 2 xe chia sẻ kênh bằng NOMA).

**3. Ví dụ minh họa**

Giả sử có 2 xe v1,v2 gửi task đến edge server e1:

* **OMA (time division)**:
  + Slot 1: v1 truyền task.
  + Slot 2: v2 truyền task.
* **NOMA**:
  + Slot 1: cả v1 và v2 **cùng truyền** (công suất khác nhau → server dùng SIC để giải mã).
* **Trong MEC bài báo**:  
  Một frame có thể chia:
  + Slot A: offloading (xe → edge server, dùng NOMA).
  + Slot B: tính toán (ở edge server).
  + Slot C: trả kết quả về xe.

👉 Vậy nên:

* **NOMA**: dùng để chia sẻ tài nguyên vô tuyến hiệu quả hơn trong **cùng 1 slot**.
* **Time slot**: vẫn cần để mô hình hóa tiến trình (offloading → computing → feedback) và lập lịch hệ thống.

## 🔹 1. **P – Transmission Power Allocation (công suất truyền)**

* Ký hiệu thường là hoặc .
* Đây là **công suất mà thiết bị (xe) dùng để truyền dữ liệu lên edge node (EN)**.
* Vì hệ thống dùng **NOMA**, nhiều xe truyền cùng lúc → công suất quyết định:
  + **Tốc độ truyền (data rate)**: công suất cao hơn → tốc độ cao hơn.
  + **Nhiễu liên biên (inter-user interference)**: công suất cao quá gây nhiễu cho xe khác.
* **Ràng buộc:**
* (không vượt quá công suất phát tối đa của xe).

👉 Trong tối ưu: **P được chọn để cân bằng giữa tốc độ truyền và mức nhiễu gây ra**.

## 🔹 2. **Q – Task Offloading Decision (quyết định offload)**

* Ký hiệu thường là hoặc biến nhị phân .
* Cho biết **xe i có offload task sang EN e hay không**.
* Có thể là:
  + **0/1** (offload hoặc không).
  + Hoặc **[0,1]** (thư giãn, coi như tỉ lệ phần task được offload).
* **Ràng buộc:**
* (mỗi task chỉ được offload đến **tối đa 1 EN**).

👉 Trong tối ưu: **Q quyết định hướng đi của task – xử lý cục bộ hay gửi lên EN nào**.

## 🔹 3. **C – Computation Resource Allocation (phân bổ tài nguyên tính toán)**

* Ký hiệu thường là hoặc .
* Là **số CPU cycles/s mà EN e phân bổ cho task của xe i**.
* Vì mỗi EN có năng lực tính toán hữu hạn (), ta có ràng buộc:
* **Ý nghĩa:** nếu nhiều task cùng offload đến 1 EN → phải chia sẻ CPU.
* Chia ít thì task xử lý chậm, trễ tăng.

👉 Trong tối ưu: **C được chọn để giảm trễ xử lý, nhưng vẫn công bằng giữa các task**.

## 🔹 4. Vai trò trong bài toán tối ưu

* **P** → quyết định **tốc độ truyền & mức nhiễu**.
* **Q** → quyết định **task được xử lý ở đâu** (local / EN).
* **C** → quyết định **tốc độ xử lý task ở EN**.

📌 Bài toán tối ưu chung:

Trong đó:

* Hàm mục tiêu có thể là **tối đa lợi ích hệ thống** (utility), **tối thiểu độ trễ trung bình**, hoặc **tối đa số task được hoàn thành trong deadline**.

👉 Tóm gọn:

| Biến | Ý nghĩa | Vai trò trong tối ưu |
| --- | --- | --- |
| **P** | Công suất truyền | Cân bằng tốc độ truyền và nhiễu |
| **Q** | Quyết định offload | Chọn EN hay local để xử lý task |
| **C** | Tài nguyên tính toán | Quyết định tốc độ xử lý task tại EN |

Bài toán CRO muốn **tối đa hóa f1**.  
Tức là tìm **(P,Q,C)(P, Q, C)(P,Q,C)** sao cho:

* Nhiều task nhất có thể được xử lý (không bị drop vì trễ hoặc thiếu tài nguyên).
* Đảm bảo giới hạn công suất phát, CPU cycles của EN, và hạn chót (deadline) của tác vụ.

👉 Vậy tóm gọn: **f1 trong CRO = maximize task success ratio toàn hệ thống qua nhiều time slots**.

| **Bước** | **Công thức chính** | **Kết quả** |
| --- | --- | --- |
| 1. Sinh task | Xác suất pp | Task với (D,F,Td)(D, F, T\_d) |
| 2. Quyết định | Chính sách offload QQ | Chọn local hoặc EN |
| 3. Offload | SINR → Rv,eR\_{v,e}, delay = D/R+F/CD/R + F/C | Tổng delay offload |
| 4. Local | delay = F/ClocalF / C\_{local} | Tổng delay local |
| 5. Deadline check | delay ≤ TdT\_d ? | Thành công / thất bại |
| 6. Ghi nhận | Đếm task thành công | Input cho f1 |
| 7. CRO tối ưu | Max f1(P,Q,C) | Chính sách tối ưu |

# Tổng quan ngắn

* **Mục tiêu**: học chính sách offloading phân tán cho mỗi edge node (mỗi node là một agent) sao cho tiến tới NE của potential game.
* **Kiến trúc**: một **learner** (trung tâm chịu trách nhiệm cập nhật mạng) + nhiều **distributed actors** (thu thập trải nghiệm song song) + **replay buffer** + **môi trường NOMA-VEC** (khi actor gửi action, hệ thống giải P2 để phân bố P và C → trả reward & trạng thái mới).
* **Thuật toán RL**: MA-D4PG (multi-agent distributional DDPG): actor deterministic, critic phân phối (distributional), dùng n-step returns, replay buffer, target networks, soft updates.

# Diễn giải Algorithm 1 — Learner / Training loop (chi tiết dòng-một-dòng)

**Mục đích:** thu thập trải nghiệm từ nhiều actors, huấn luyện critic & policy (actor) trên minibatch, cập nhật target networks, đồng bộ tham số tới actors.

Dưới đây là pseudo-code kèm chú giải (mỗi dòng tương ứng với nội dung trong bài):

1. **Initialize the network weights;**
   * Khởi tạo tham số mạng: θ\_μ cho policy (actor) và θ\_Q cho critic.
   * Mạng policy: ánh xạ quan sát cục bộ o\_e^t → hành động a\_e^t (ở bài báo là offloading decision).
   * Mạng critic: đánh giá Q-value dựa trên (thường) thông tin cục bộ + action toàn cục Q(o\_e, a\_all | θ\_Q).
2. **Initialize the replay buffer B;**
   * Buffer lưu các transition (ở D4PG dùng n-step sequences): mỗi entry có dạng (o\_{t:t+N}, a\_{t:t+N−1}, r\_{t:t+N−1}, o\_{t+N}).
   * Kích thước tối đa |B| được cấu hình.
3. **Launch J distributed actors and replicate network weights to the actors;**
   * Tạo J actors chạy song song; copy tham số mạng policy (hoặc target policy) vào mỗi actor θ\_μ^j ← θ\_μ (hoặc θ\_μ0). Actors dùng policy để tương tác với môi trường.
4. **for iteration = 1 to max-iteration-number do** — outer training loop.
5. **for t = 1 to T do** — lặp qua từng time-slot trong 1 epoch/episode.
6. **for edge node e = 1 to E do** — (trong learner mỗi bước lấy sample để cập nhật; thực tế learner làm trên minibatch nhiều tác tử cùng lúc).
7. **Sample M transitions of length N from B randomly;**
   * Lấy M sequences (mỗi sequence dài N) để tính target distribution (n-step returns).
   * M là batch size, N là n-step bootstrap length.
8. **Construct the target distributions;**
   * Với mỗi sample i và mỗi edge node e, compute target theo Eq.(36):
   * Ở đây a\_{i+N} là hành động tại bước i+N do **target policy** tạo ra. (phần distributional: trong D4PG họ dùng distributional targets — bài báo tóm gọn bằng biểu thức này).
9. **Compute the policy and critic network loss;**
   * **Critic loss** (Eq.37): MSE giữa target Y\_i^e và Q(o\_i^e, a\_i | θ\_Q):
   * **Policy loss** được tính gián tiếp qua gradient của critic (actor không có loss trực tiếp): use deterministic policy gradient (Eq.38):
   * Thực tế cập nhật policy bằng step theo hướng ∇θµ J (maximize Q).
10. **Update the local policy and local critic networks;**
    * Áp dụng optimizer (Adam, RMSProp…) với learning rates α (policy) và β (critic) để cập nhật θ\_μ và θ\_Q.
11. **if t mod ttgt = 0 then Update the target networks;**
    * Soft update target networks θ\_μ0 và θ\_Q0 theo (39):
    * (bài báo ký n thay τ, nhưng ý là soft update). ttgt là chu kỳ cập nhật target.
12. **if t mod tact = 0 then Replicate network weights to the distributed actors;**
    * Sau mỗi tact bước, copy tham số policy (thường target policy θ\_μ0 hoặc θ\_μ) sang các actors: θ\_μ^j ← θ\_μ0 (Eq.40). Giúp actors dùng chính sách cập nhật để thu thập trải nghiệm mới.
13. Vòng for kết thúc; lặp tiếp iteration.

# Diễn giải Algorithm 2 — j-th Distributed Actor (chi tiết)

**Mục đích:** các actor chạy song song trong môi trường thật (hoặc mô phỏng), sinh trải nghiệm (trajectories) và lưu vào replay buffer để learner học.

Pseudo-code + chú giải:

1. **while Learner is not finished do** — actor chạy liên tục cho tới khi learner dừng huấn luyện.
2. **Initialize a random process N for exploration;**
   * Khởi tạo noise process (Ornstein-Uhlenbeck, Gaussian, v.v.) để thêm vào action cho khám phá.
3. **Receive the initial system state o1;**
   * Nhận trạng thái đầu của episode: o\_1 = {o\_1^1,...,o\_1^E}.
4. **for t = 1 to T do** — lặp từng time slot.
5. **for edge node e = 1 to E do** — cho mỗi edge node (agent) thực hiện:
6. **Receive a local observation o^t\_e;**
   * Actor j chỉ thấy quan sát cục bộ o^t\_e (khoảng cách các xe, kích thước task, deadline, ...).
7. **Select a action a^t\_e = μ(o^t\_e | θ^j\_μ) + N\_t;**
   * Lấy policy mạng actor (param θ^j\_μ) để sinh hành động deterministically, rồi thêm noise N\_t để khám phá.
   * Ở bài báo biểu diễn thêm điều khoản ε N\_t để điều chỉnh cường độ khám phá.

* **Ghi chú về action loại rời rạc (q∈{0,1}):** tác giả dùng actor deterministic với nhiễu. Thực tế có thể:
  + output là vector rời rạc qua argmax binarize, hoặc
  + output continuous relaxation rồi threshold. (Bài báo không nêu chi tiết mapping; triển khai thực tế cần xử lý chuyển continuous → rời rạc.)

1. **Execute actions a^t = {a^t\_e} in the NOMA-based VEC environment;`**
   * Môi trường thực hiện: **(i)** áp dụng quyết định offloading (Q^t), **(ii)** gọi module P2 (solver) để giải resource allocation (transmission power + CPU allocation) cho thời slot đó, **(iii)** tính SINR, thời gian tải lên, thời gian execution, kiểm tra deadline → xác định reward r^t theo Eq.(33): potential difference.
2. **Receive the reward r^t and the next system state o^{t+1};**
3. **Store (o^t, a^t, r^t, o^{t+1}) into replay buffer B;**
   * Lưu transition (hoặc nối vào sequence để tạo n-step transitions). Actors tiếp tục tương tác.
4. Loop tiếp tới t+1; mang về learner khi learner lấy mẫu.

# Giải thích các phần công thức quan trọng được dùng trong learner

* **Reward cục bộ (Eq.33)**:
  + r\_{a^t|o^t} = tổng service ratio toàn hệ nếu tất cả edge nodes thực hiện hành động a^t.
  + r\_{a^t\_{-e}|o^t} = tổng service ratio khi **loại bỏ** đóng góp của node e (tức đặt hành động của e thành null).
  + Kết quả là reward phản ánh **đóng góp biên** (marginal contribution) của node e → khớp với potential game, thúc đẩy agent tối ưu theo hàm potential.
* **Target distribution (Eq.36)**: n-step return cộng bootstrap bằng target critic: dùng để huấn luyện critic không chỉ trên 1-step mà n-step giúp ổn định & tăng hiệu quả.
* **Critic loss (Eq.37)**: MSE giữa target và giá trị dự đoán .
* **Policy gradient (Eq.38)**: actor cập nhật theo gradient của critic w.r.t action, rồi chain-rule qua policy.
* **Soft update (Eq.39)**: target networks làm mượt để tránh dao động.

# Cách hoạt động tích hợp với P2 (resource allocation)

1. Actor chọn Q^t (offloading decisions) cho tất cả edge nodes.
2. Môi trường (simulation step) sử dụng Q^t làm input cho **P2 solver**:
   * Giải **power allocation** (P3→P6→P9) bằng các bước dual/iterative (Eq.43–46) để tìm P^t.
   * Giải **computation allocation** (P8) bằng KKT (Eq.48 → closed form Eq.49) để tìm C^t.
3. Từ Q^t,P^t,C^t tính SINR, upload time, processing time → kiểm tra deadline → tổng service ratio Ψ^t\_e → reward r^t\_e theo Eq.33.
4. Trải nghiệm (o^t,a^t,r^t,o^{t+1}) lưu vào buffer.

**Nghĩa là:** learner/actor chỉ học phần **offloading policy**; phân bổ tài nguyên được giải bằng công thức/solver (không học), nhưng kết quả của solver ảnh hưởng trực tiếp đến reward dùng để huấn luyện policy.

# Thông số và chú ý khi triển khai (practical tips)

* **Kích thước minibatch M** và **độ dài N** (n-step) ảnh hưởng ổn định/tốc độ hội tụ (M thường 32–256; N thường 3–20).
* **Số actors J**: càng nhiều actor thu thập dữ liệu nhanh hơn nhưng cần đồng bộ tham số theo tact.
* **Khám phá (exploration)**: noise N\_t nên giảm dần theo thời gian; nếu action là rời rạc, dùng epsilon-greedy hoặc Gumbel-Softmax để nội suy.
* **Cập nhật target**: soft update factor τ nhỏ (0.001–0.01) thường ổn định. ttgt và tact điều chỉnh theo tài nguyên.
* **Reward scaling**: vì reward là service ratio contribution, có thể nhỏ; cần scale/normalize để learning ổn định.
* **Batching sequences**: lưu n-step sequences trong replay buffer hoặc lưu transitions và khi sample dựng sequence.
* **Action rời rạc (0/1)**: paper dùng deterministic actor + noise; thực thi thực tế cần map continuous→discrete (threshold/argmax/Gumbel-soft). Chọn phương pháp phù hợp cho việc học ổn định.
* **Chi phí tính toán**: mỗi env step cần solve P2 (iterative dual + KKT). Trong mô phỏng, P2 nên được cài đặt hiệu quả (vectorized) để nhiều actors/steps không làm chậm training quá mức.

# Khi huấn luyện xong — chính sách được dùng thế nào?

* Sau khi learner hội tụ: dùng **policy deterministic** a\_e = μ(o\_e | θ\_μ) (không thêm noise) để thực thi offloading online.
* Vì reward base on potential, agent policy đã học xu hướng tối đa hóa service ratio toàn hệ/đóng góp biên → tiến gần NE.

# Tóm tắt ngắn gọn (bằng bullet)

* **Algorithm 1 (learner)**: thu thập minibatch dài N từ buffer → tính n-step targets Y (Eq.36) → critic loss (Eq.37) → update critic → compute policy gradient (Eq.38) → update policy → soft update target → sync actors.
* **Algorithm 2 (actor)**: nhận quan sát cục bộ → sinh action µ(o)+noise → gửi lên environment → environment giải P2 (power+CPU) → trả reward (potential diff) + next state → lưu transition vào buffer.
* **Điểm then chốt đa-tác tử**: reward dùng hàm potential (marginal contribution) để khuyến khích hành vi hợp tác; critic học distributional n-step returns với thông tin toàn cục (qua a\_all trong Q), còn actor local học policy phân tán → phối hợp hiệu quả.

# 📖 **Tổng quan bài báo (System + Algorithm + Optimization)**

## 1. **Mô hình hệ thống (System Model)**

* Hệ thống là **Vehicular Edge Computing (VEC)** kết hợp **NOMA (Non-Orthogonal Multiple Access)**.
* Thành phần chính:
  + **Xe (Vehicles)**: sinh tác vụ ngẫu nhiên theo xác suất mỗi slot.
  + **MEC Server**: gắn trên RSU (road-side unit), có khả năng xử lý task offloaded.
  + **Time Slot**: Hệ thống chia thời gian thành slot, trong mỗi slot các xe có thể sinh tác vụ mới và quyết định offload hay xử lý local.
* **Ba quyết định chính của mỗi xe** trong mỗi slot:
  1. **Công suất truyền**  khi offload.
  2. **Số lượng tác vụ offload** .
  3. **Tài nguyên tính toán local**  (CPU cycles).
* **Mục tiêu cục bộ** của xe: tối đa hóa tỉ lệ task xử lý thành công.
* **Mục tiêu hệ thống**: tối đa hóa tổng tỉ lệ task thành công trên tất cả xe, qua tất cả slot.

## 2. **Các công thức tính toán (phần chuẩn bị)**

* Bài báo đưa ra **các biểu thức toán học** để mô tả:
  + **Xác suất sinh tác vụ** và hàng đợi (queue dynamics).
  + **Độ trễ truyền tải & xử lý local**.
  + **Nhiễu trong NOMA** khi nhiều xe offload cùng lúc.
  + **Khả năng xử lý của MEC server**.

👉 Những công thức này không phải mục tiêu tối ưu trực tiếp, mà là **công cụ để tính reward và constraints** trong bài toán tối ưu.

## 3. **EPG (Expected Performance Gain)**

* **Định nghĩa**: thước đo hiệu quả kỳ vọng khi một xe chọn hành động .
* EPG tính đến:
  + Khả năng tác vụ thành công.
  + Tài nguyên tiêu thụ.
  + Ảnh hưởng nhiễu đến người khác.

👉 **Vai trò**: EPG chính là một phần của reward, giúp đánh giá xem hành động vừa chọn có “lợi” hay không.

## 4. **NE (Nash Equilibrium)**

* Vì nhiều xe cùng tham gia → đây là **multi-agent game**.
* Nếu một xe tăng công suất quá mức → gây nhiễu → toàn hệ giảm hiệu quả.
* Nash Equilibrium = trạng thái mà **không xe nào có thể cải thiện hiệu suất của mình bằng cách đơn phương thay đổi chiến lược**, khi những xe khác giữ nguyên chiến lược.

👉 **Vai trò**: NE là trạng thái hội tụ mong muốn của hệ thống, đảm bảo **ổn định + công bằng**.

## 5. **Bài toán tối ưu CRO (Centralized Resource Optimization)**

* Hàm mục tiêu:
* trong tất cả slot, trên tất cả xe.
* Các biến quyết định: .
* Ràng buộc: giới hạn tài nguyên MEC, giới hạn công suất, giới hạn độ trễ.

## 6. **Thuật toán giải quyết**

### (1) **MA4DPG (Multi-Agent 4-Dimensional Policy Gradient)**

* Đây là **thuật toán học tăng cường đa tác tử**.
* Gồm hai thành phần:
  + **Distributed Actor (xe)**: quyết định local dựa trên quan sát trạng thái + chính sách đã học.
  + **Centralized Critic (MEC)**: đánh giá EPG & reward, cập nhật cho actor.
* Mỗi vòng lặp:
  1. Xe sinh task → Actor chọn hành động.
  2. Task được xử lý local hoặc offload.
  3. Reward tính theo số task thành công.
  4. Critic đánh giá & cập nhật gradient chính sách.
  5. Lặp lại cho đến khi hội tụ (mong muốn đạt NE).

👉 **MA4DPG tối ưu đa tác tử** bằng cách để **mỗi xe là một agent** nhưng học nhờ critic chung, đảm bảo học được chính sách ổn định khi nhiều xe cùng tồn tại.

### (2) **j-th Distributed Actor (thuật toán phụ)**

* Là thành phần trong MA4DPG, đóng vai trò **tác tử local cho xe thứ j**.
* **Đầu vào**: trạng thái local của xe j (task queue, kênh truyền, tài nguyên MEC).
* **Đầu ra**: hành động .
* Sau khi hành động, reward nhận được từ critic giúp actor update chính sách.

👉 Vai trò của Distributed Actor là: **tự quyết định local**, không cần toàn bộ thông tin hệ thống, nhưng vẫn học tốt nhờ critic chung.

## 7. **Quy trình tổng quan (kết nối tất cả)**

1. **Khởi tạo**: Hệ thống chia thời gian thành slot, mỗi xe có xác suất sinh task.
2. **Distributed Actor (từng xe)** chọn hành động .
3. **Task thực thi**: xử lý local hoặc offload qua MEC (NOMA).
4. **Reward tính bằng EPG**: hiệu quả kỳ vọng của hành động.
5. **Centralized Critic** thu thập reward, cập nhật chính sách cho actor.
6. Hệ thống lặp lại nhiều vòng → chính sách hội tụ đến **Nash Equilibrium**.
7. **Kết quả cuối cùng**: Tối đa hóa **tỉ lệ task xử lý thành công** trên toàn hệ thống.

📌 **Tóm tắt dễ hiểu**:

* Mô hình VEC + NOMA → nhiều xe, task sinh ngẫu nhiên, phải quyết định offload.
* EPG = thước đo hiệu quả kỳ vọng.
* NE = trạng thái ổn định, không xe nào muốn thay đổi một mình.
* MA4DPG = RL đa tác tử để học chính sách.
* Distributed Actor = agent local cho mỗi xe, chọn hành động.
* Toàn hệ thống → tối ưu tỷ lệ task thành công.