Universitatea din Bucuresti

Facultatea de Matematica si Informatica

Optimizarea plasarii continutului in cache-uri distribuite

O abordare bazata pe Reinforcement Learning

Profesor indrumator:
Stefan Iordache

Echipa:
Gabriel Stefan
David Cotiga
Andrei Copilau

15 ianuarie 2026

Cuprins

1 Introducere

Inspiratia din spatele acestui proiect vine din succesul recent al Reinforcement Learning in optimizarea unor sisteme hardware foarte complexe, cum ar fi chip design la Google sau thread scheduling in procesoarele moderne.

Echipa noastra si-a propus sa aplice acelasi principiu in Content Delivery Networks. Algoritmi euristici clasici de caching, precum LRU, LFU sau Random, folosesc reguli fixe si nu se pot adapta rapid la schimbari de trafic sau la comportamentul utilizatorilor. Scopul nostru a fost sa construim un agent RL care sa invete strategii dinamice de caching, adaptate la contextul retelei, si sa depaseasca metodele traditionale bazate pe reguli statice.

2 Environment

2.1 Setul de date si preprocesare

Pentru validare am folosit trace-uri reale de trafic din proiectul WikiBench, care contin cereri reale pentru continut Wikipedia. Aceste date reflecta distributii reale de popularitate a datelor si ofera un mediu realist pentru antrenarea agentului.

Datele au fost curatate si transformate intr-un format CSV pentru a putea fi folosite direct in simulare. Au fost impartite cronologic intr-un train/test split 70/30. Primele 70% din cereri au fost folosite pentru antrenare, iar ultimele 30% doar pentru evaluare.

Costurile de acces la Origin sunt simulate folosind un hash determinist al URL-ului pentru a introduce latenta variabila.

2.2 Arhitecutura CDN-ului

CDN-ul este gandit ca o retea ierarhica de noduri interconectate, definind un graf in care fiecare nod are o anumita capacitate, latenta si bandwidth. Modelul foloseste o topologie pe mai multe niveluri pentru a simula arhitectura si propagarea reala a continutului intr-un sistem CDN modern.

Topologia folosita in simulari este organizata in trei niveluri:

- **Tier 0 (Edge):** Reprezinta nodul situat cel mai aproape de utilizatori. Acesta este caracterizat prin cea mai mica latenta si cel mai mare bandwidth, insa are cea mai mica capacitate de stocare.
- **Tier 1 (Regional):** Nod intermediar care actioneaza ca un buffer intre Edge si Origin. Acesta are o capacitate de stocare mai mare decat nivelul Edge, dar are de asemenea penalizari mai mari de latenta si bandwidth.
- **Origin:** Sursa finala a continutului. Desi capacitatea de stocare este considerata nelimitata, accesarea acestui nod are cele mai mari costuri latenta si limitari de bandwidth.

Parametrii specifici utilizati in cadrul simularii pentru fiecare tip de nod sunt in tabelul de mai jos.

Tip Nod	**Capacitate**	**Latenta**	**Bandwidth**
Tier 0 (Edge)	1.0 MB	1 ms	1000 Mbps
Tier 1 (Regional)	2.0 MB	10 ms	500 Mbps
Origin	∞	100 ms	100 Mbps

Environmentul este implementat prin intermediul unei clase, care permite configurarea libera a retelei. Sistemul suporta un numar variabil de noduri si niveluri si noduri cu parametri diferiti.

3 Evolutia Environmentului

Ideea noastra initiala, in care agentul decidea ce item sa fie eliminat, a fost rafinata in urma mai multor experimente si iteratii ale environmentului, care au evidentiat multe limitari ale acestei formule. Ca urmare, putem spune ca procesul poate fi impartit in trei faze.

3.1 Faza 1: Control bazat pe evictii

In configuratia initiala, corespunzatoare primei faze a abordarii si ideei noastre initiale, agentul RL trebuia sa decida ce obiect sa fie eliminat din cache. La fiecare pas, agentul primea un set de 30 de candidati: 10 propusi de LRU, 10 de LFU si 10 selectati aleator, si trebuia sa aleaga unul singur pentru eviction.

In practica, politica a suferit un fenomen de colaps al actiunilor. Agentul a invatat ca anumiti indici de actiune, de exemplu o pozitie fixa in lista, duc frecvent la rezultate acceptabile, indiferent de obiectul real. Astfel, agentul nu a invatat o strategie de caching, ci doar a memorat pozitii in vectorul de actiuni.

Pentru a elimina aceasta problema, spatiul de actiuni a fost simplificat. In loc sa aleaga obiecte individuale, agentul a fost mutat la nivelul de alegere a unei euristici de evictie pe care sa o aplice pe un nod la alegere.

3.2 Faza 2: Evaluarea euristicilor

Pentru a evalua potentialul acestei abordari, au fost rulate benchmark-uri pe 1 milion de cereri din trace-ul Wikipedia. Rezultatele au aratat ca LFU obtine un hit rate de aproximativ 52.18%, in timp ce LRU atinge 40.30%, FIFO 34.91% si Random 34.89%.

Policy	Hit Rate	Byte Hit Rate
LFU	**52.18%**	**51.10%**
LRU	40.30%	39.15%
FIFO	34.91%	33.81%
Random	34.89%	33.79%

Simulat pe un Cache unificat (Edge + Regional) pentru a testa performanța pura de eviction.

Aceste rezultate indica faptul ca pentru acest trace, LFU este deja optima teoretic pentru eviction si invatarea unei politici mai bune de eliminare ofera un spatiu redus de optimizare pe care nu l-am putut gasi.

Spre exemplu, cel mai reusit experiment pe aceasta structura a folosit un agent RL bazat pe DQN cu 7 actiuni, fiecare corespunzand alegerii unei euristici pe un nod sau skip.

Aceasta abordare a obtinut un hit rate de 46.03%, sub performanta LFU de 52.18%. Agentul a avut dificultati in a invata momentele optime de schimbare a strategiei si a colapsat frecvent catre o singura euristica suboptima.

3.3 Faza 3: Controlul plasarii continutului

Dupa aceste doua faze, am hotarat ca e putin probabil sa putem obtine imbunatatiri semnificative prin optimizarea politicii de eviction. LFU este deja aproape optim pentru acest trace, iar agentii RL nu au reusit sa il depaseasca atunci cand trebuiau doar sa aleaga intre euristici. Din acest motiv, am schimbat formularea problemei: am fixat politica de eviction la LFU si am mutat invatarea catre decizia de plasare a continutului in ierarhia de cache (Edge, Regional sau Skip).

Am implementat un discrete action space care contine trei optiuni:

- Cache at Edge: obiectul este stocat pe nodul Edge
- Cache at Regional: obiectul este stocat pe nodul Regional
- Skip: obiectul nu este stocat si este servit direct din Origin

Observation space-ul folosit de agent este structurat in felul urmator:

- Caracteristici ale nodurilor (12 in total): pentru fiecare nod (Edge si Regional in cazul nostru) sunt urmarite 6 metrice:
  - Occupancy: procentul de cache utilizat
  - Item Count: numarul de obiecte stocate, normalizat
  - Avg Size: dimensiunea medie a obiectelor din cache
  - Avg Frequency: popularitatea medie a obiectelor stocate
  - Avg Recency: cat de recent a fost accesat continutul
  - Cache Pressure: raportul dintre dimensiunea cererii curente si spatiul liber

- Caracteristici ale cererii (2 in total):
  - Size: dimensiunea normalizata a obiectului cerut
  - Frequency Hint: un indicator al popularitatii istorice a URL-ului
- Caracteristici globale (3 in total):
  - Total Hit Rate: performanta globala
  - Edge Hit Ratio: procentul de hit-uri servite de Edge
  - Regional Hit Ratio: procentul de hit-uri servite de Regional

Inainte de formularea finala a functiei de reward, au fost testate mai multe variante, inclusiv rewarduri binare (`+1` pentru hit, `-1` pentru miss) si rewarduri bazate pe economii absolute de latenta. Acestea s-au dovedit nepotrivite: rewardurile binare erau prea rare pentru a oferi un semnal de invatare util, iar cele bazate pe latenta absoluta introduceau o varianta foarte mare, deoarece cererile mari dominau procesul de optimizare.

Functia de reward finala foloseste economia relativa de latenta:

$$R = \frac{L_{origin} - L_{actual}}{L_{origin}} \times 10.$$

Aceasta forma normalizeaza recompensa in functie de costul cererii, astfel incat atat obiectele mici cat si cele mari contribuie comparabil la procesul de invatare. In consecinta, varianta gradientilor este redusa, iar antrenarea devine mai stabila.

4 Baseline-uri

Pentru a evalua performanta agentului RL in problema de plasare, am definit mai multe politici euristice care decid unde este stocat fiecare obiect.

- SizeSplit (Median): obiectele mai mici decat dimensiunea mediana sunt stocate pe Edge, iar cele mai mari pe Regional.
- PercentileSplit (P90): doar cele mai mici 10% dintre obiecte sunt stocate pe Edge, restul fiind directionate catre Regional.
- EdgeFirst: toate obiectele sunt stocate pe Edge pana cand cache-ul este plin, dupa care sunt redirectionate catre Regional.
- Probabilistic (10%): obiectul are o probabilitate de 10% sa fie stocat pe Edge, altfel pe Regional.

Policy	**Hit Rate**	**Reward (Latency)**
SizeSplit (Median)	**52.06%**	**4,472,646**
PercentileSplit (P90)	46.62%	4,275,183
EdgeFirst	43.10%	4,017,288
Probabilistic (10%)	42.67%	3,721,054

Tabela 1: Rezultate baseline pe 1 milion de requesturi

5 Agenti implementati

5.1 DQN

Agentul a fost implementat folosind biblioteca Stable Baselines 3. DQN a fost ales ca punct de plecare deoarece este algoritmul standard pentru probleme cu discret action spaces, iar in cazul nostru decizia este explicit o alegere intre trei optiuni fixe (Edge, Regional, Skip). In plus, DQN invata direct o functie $Q(s, a)$ care estimeaza valoarea pe termen lung a fiecarei actiuni, ceea ce se potriveste cu problema noastra de a evalua unde este mai profitabil sa fie plasat un obiect in cache.

Pentru evaluare, am inceput cu un baseline folosind parametrii impliciti ai DQN.

Metric	**Value**
Steps	500,000
Hit Rate	42.23%
Byte Hit Rate	41.12%
Average Latency	64.70 ms
Edge Actions	55.0%
Regional Actions	0.0%
Skip Actions	45.0%

Tabela 2: DQN baseline (hiperparametrii default)

Agentul ignora complet nodul Regional. In termeni de Q-values, reteaua a invatat sistematic $Q(s, \text{Edge}) > Q(s, \text{Skip}) > Q(s, \text{Regional})$. Rewardurile obtinute prin Regional sunt intarziate fata de Edge, iar cu un learning rate mic acestea nu au fost propagate eficient, ducand la subestimarea valorii Regional.

Pentru a testa daca problema este lipsa de explorare, am crescut rata de explorare si dimensiunea bufferului de replay.

Metric	**Value**
Steps	500,000
Exploration Fraction	0.3
Replay Buffer Size	200,000
Hit Rate	42.80%
Byte Hit Rate	41.57%
Average Latency	64.08 ms
Edge Actions	69.8%
Regional Actions	0.0%
Skip Actions	30.2%

Tabela 3: DQN cu exploration rate mare

Comportamentul a ramas identic: nodul Regional este complet ignorat, indicand ca problema nu este explorarea, ci propagarea rewardului prin Q-learning.

In continuare, am variat doar learning rate-ul. Rezultatele arata o tranzitie majora intre regimuri de caching complet diferite:

Exp	**LR**	**Hit Rate**	**Latency (ms)**	**Policy**
Exp 3	10^{-3}	48.89%	62.26	100% Regional
Exp 4	5×10^{-4}	50.49%	57.47	99.6% Regional
Exp 7	2×10^{-4}	49.35%	61.80	99.8% Regional
Exp 8	1.5×10^{-4}	42.62%	64.27	0% Regional

Tabela 4: Sensitivity la learning rate

Pentru learning rate-uri mici, contributiile rewardului intarziat al nodului Regional nu sunt propagate eficient prin update-urile Q-learning, ceea ce duce la subestimarea sistematica a $Q(s, \text{Regional})$. Pentru learning rate-uri mari, bootstrap-ul devine instabil si amplifica excesiv aceste rewarduri, facand ca Regional sa domine complet si sa suprime Edge si Skip. Ca urmare, exista doar o fereastra foarte ingusta de valori ale learning rate-ului in care se obtine o politica mixta stabila.

Cele mai bune rezultate au fost obtinute pentru unde learning rate de 3×10^{-4}, cu doua configuratii:

Exp 6	
Learning Rate	3×10^{-4}
Batch Size	32
Hit Rate	51.68%
Latency	56.12 ms
Policy	60% Regional, 38% Skip, 2% Edge

Exp 10	
Learning Rate	3×10^{-4}
Batch Size	128
Hit Rate	51.70%
Latency	55.25 ms
Policy	96% Regional, 4% Edge, 0% Skip

Tabela 5: Cele mai bune configuratii pe DQN

Learning rate-ul 3×10^{-4} reprezinta un punct critic in care valoarea nodului Regional devine invatabila fara a suprima complet Edge. Cresterea batch size-ului la 128 a redus varianta gradientilor, stabilizand estimarile Q si eliminand comportamentul de skip. Aceasta a dus la o politica Cache Everything care maximizeaza utilizarea cache-ului si minimizeaza latenta.

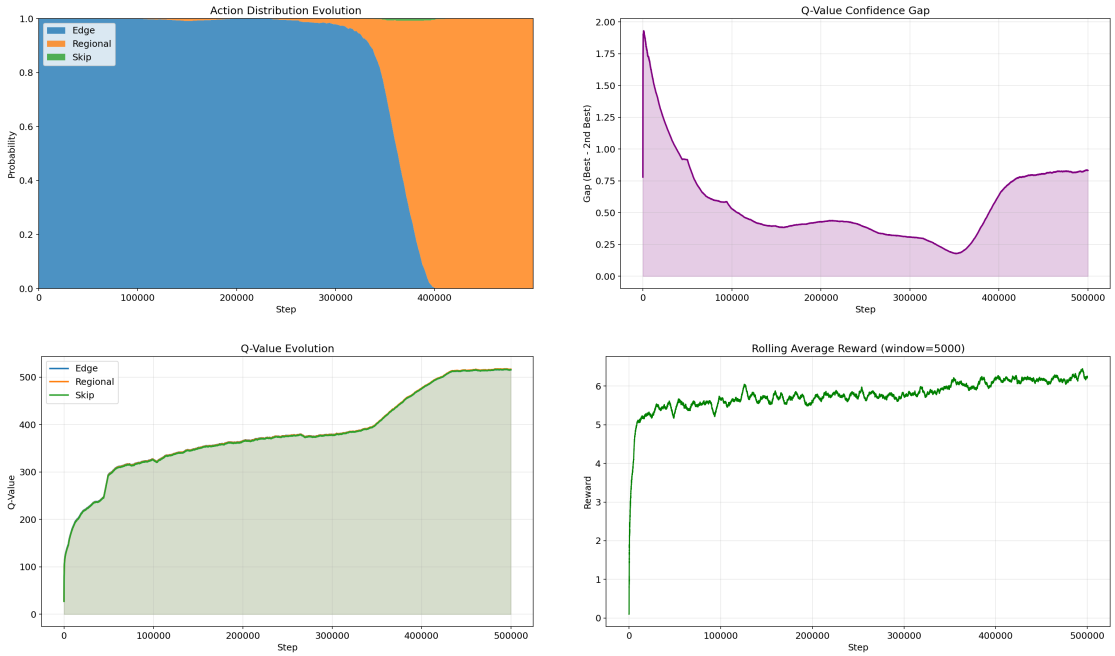

Figura 1: Analiza DQN

Aceste grafice arata ca DQN invata printr-o tranzitie de faza intre doua politici extreme (Edge-only si Regional-only), fara a putea mentine o combinatie stabila intre cele doua. Pe masura ce valorile Q pentru actiuni se apropie si se intersecteaza, politica sare brusc de la un tier la altul, ceea ce poate fi observat in distributia actiunilor alese de agent si in graficul Q-value confidence gap. Acest comportament este cauzat de feedback-ul Q-learning-ului: o actiune usor mai buna este aleasa mai des, apare mai des in replay buffer si ajunge sa domine complet. Desi rewardul mediu continua sa creasca, DQN nu poate invata o strategie ierarhica stabila, ceea ce explica sensibilitatea extrema la learning rate si instabilitatea sa in acest environment.

5.2 PPO

Agentul PPO a fost implementat tot folosind libraria Stable Baselines 3. Am ales PPO deoarece optimizeaza direct o politica stocastica, ceea ce il poate face mai potrivit decat DQN pentru probleme in care este necesar un echilibru fin intre mai multe actiuni, cum e in cazul nostru Edge, Regional, Skip si unde colapsul politicii este un risc major.

La fel ca la DQN, am inceput experimentele cu un baseline cu hiperparametrii default. Politica a colapsat intr-un regim Always Edge, similar cu DQN la learning rate mic. Cu doar 100k pasi si entropie mica, PPO nu a explorat suficient pentru a descoperi valoarea nodului Regional.

Metric	Value
Steps	100,000
Learning Rate	3×10^{-4}
Entropy Coef	0.01
Hit Rate	40.82%
Byte Hit Rate	39.74%
Average Latency	66.26 ms
Edge Actions	99.0%
Regional Actions	0.0%
Skip Actions	1.0%

Tabela 6: PPO baseline

Cresterea entropiei si a duratei de antrenare a eliminat total colapsul politicii. Agentul a invatat o distributie complexa asupra actiunilor, folosind Regional mai mult decat Edge si skip-uind selectiv. Entropia mai mare a fortat explorarea continua, iar antrenarea mai lunga a permis propagarea recompenselor intarziate ale nodului Regional, ceea ce a dus la invatarea unei strategii ierarhice corecte.

Metric	Value
Steps	500,000
Entropy Coef	0.05
Hit Rate	51.82%
Byte Hit Rate	51.72%
Average Latency	58.69 ms
Edge Actions	27.9%
Regional Actions	47.1%
Skip Actions	24.9%

Tabela 7: PPO cu entropie mai mare

Cresterea capacitatii retelei a degradat performanta, introducand un bias spre Edge. Ca si in cazul DQN si Discrete SAC, retelele mici generalizeaza mai bine pentru acest environment. Modelele cu capacitate mare tind sa supra-invete corelatii accidentale din advantage sau Q-values, favorizand actiunile cu reward imediat, cum ar fi Edge in cazul nostru. Retelele mai mici au varianta mai redusa, invata functii mai netede si generalizeaza mai bine, ceea ce conduce la politici mai stabile si mai echilibrate.

Metric	Value
Network	[256, 256]
Hit Rate	42.80%
Edge Actions	83.8%
Regional Actions	13.1%
Skip Actions	3.1%

Tabela 8: PPO cu large network

Reducerea learning rate-ului a produs o tranzitie de faza opusa baseline-ului: politica a abandonat complet Edge si a favorizat Regional si Skip. Ca si in DQN, PPO prezinta regimuri complet diferite in functie de learning rate. Aceasta se intampla deoarece un learning rate prea mic face ca actualizarile politicii sa fie dominate de reward-urile mai frecvente si mai stabile.

Metric	**Value**
Learning Rate	10^{-4}
Hit Rate	48.88%
Edge Actions	0.0%
Regional Actions	67.0%
Skip Actions	33.0%

Tabela 9: PPO cu learning rate mic

Reducerea `clip_range` a limitat cat de mult politica noua poate devia fata de cea veche la fiecare update. Astfel, PPO nu mai face salturi agresive catre o singura actiune atunci cand aceasta pare temporar avantajoasa, ci ajusteaza probabilitatile incremental. In acest environment, unde Edge, Regional si Skip au tradeoff-uri subtile intre reward imediat si castig pe termen lung, aceasta stabilizare a permis invatarea unui echilibru fin intre cele trei optiuni, ducand la cea mai buna performanta globala. Aceasta performanta a intrecut si Discrete SAC, despre care vom vorbi mai tarziu.

Metric	**Value**
Clip Range	0.1
Entropy Coef	0.05
Hit Rate	**51.97%**
Byte Hit Rate	**51.81%**
Average Latency	**55.41 ms**
Edge Actions	51.4%
Regional Actions	30.6%
Skip Actions	18.0%

Tabela 10: PPO cu clip range redus

Reducerea `gae_lambda` face estimarea avantajelor mai mica, punand mai mult accent pe recompensa imediata si mai putin pe efectele pe termen lung. Valoarea actiunii Edge nu provine total din recompensa imediata la plasare, ci din hit-urile viitoare pe acel obiect. Un `gae_lambda` mai mic taie aceste contributii viitoare din A(s,Edge), subestimand valoarea Edge si impingand politica sa o abandoneze in favoarea Regional si Skip.

Metric	Value
GAE Lambda	0.90
Hit Rate	48.24%
Edge Actions	0.0%
Regional Actions	49.4%
Skip Actions	50.6%

Tabela 11: PPO cu GAE lambda redus

PPO a depasit limitarile DQN deoarece optimizeaza direct o politica stocastica, nu Q-values. Configuratia PPO Clip a obtinut cel mai bun rezultat global, cu cea mai mica latenta si cel mai mare hit rate dintre toate experimentele cu agenti RL, apropiindu-se foarte mult de baseline-ul euristic *median size split*.

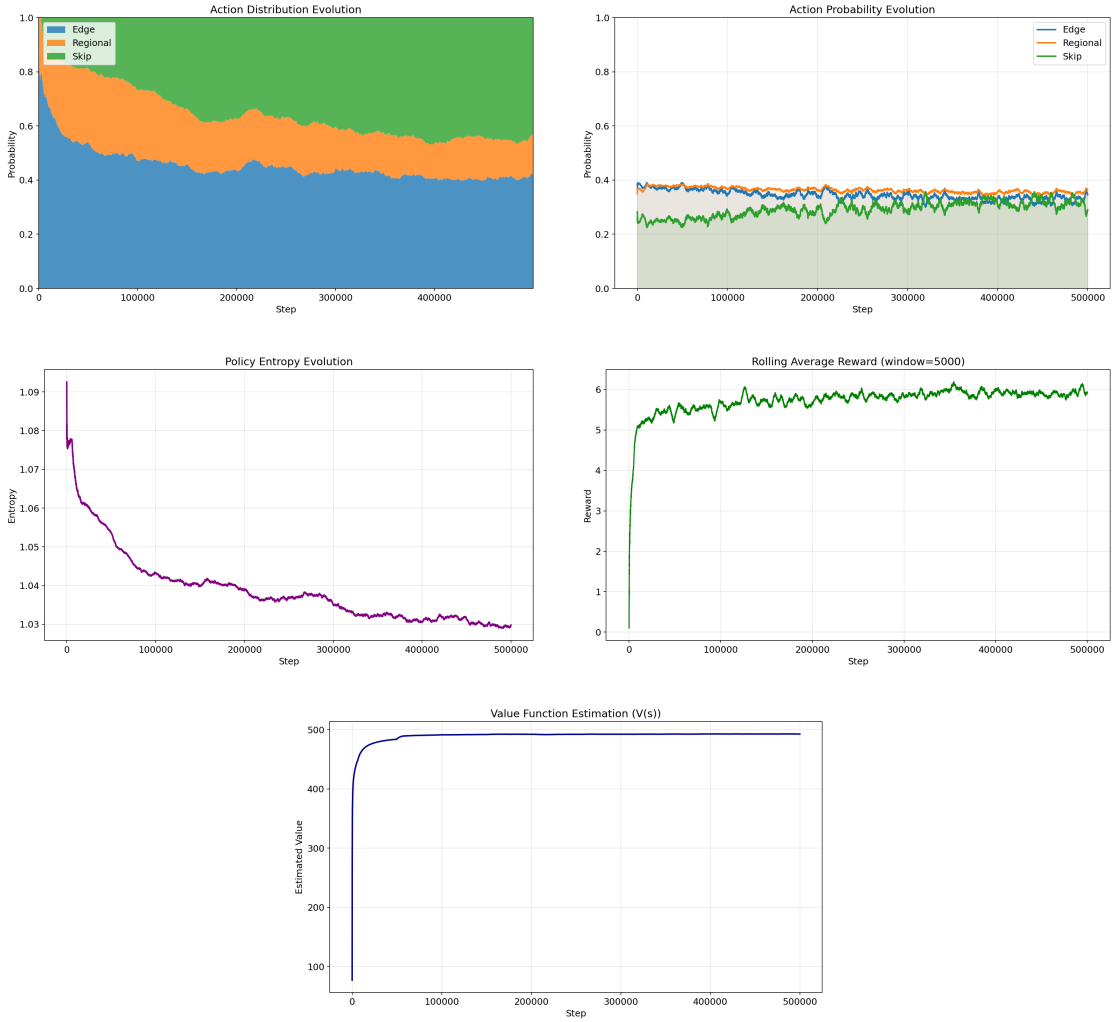

Figura 2: Analiza PPO

Graficele arata ca PPO converge catre o politica stocastica stabila, in care toate cele trei actiuni raman active pe termen lung. Distributia actiunilor se stabilizeaza fara a colapsa spre un singur tier, ceea ce indica faptul ca agentul a invatat un trade-off real dintre Edge,

Regional si Skip. Scaderea lenta a entropiei arata ca explorarea este redusa gradual, nu fortat, permitand descoperirea valorii nodului Regional fara pierderea Edge. Cresterea si stabilizarea rewardului si a functiei de valoare confirma ca PPO a invatat o evaluare corecta a beneficiilor pe termen lung ale fiecarei decizii de cache.

5.3 Discrete SAC

In final, am implementat si evaluat un agent Discrete SAC. Soft Actor-Critic este in forma sa clasica un algoritm off-policy pentru action space-uri continue, care invata simultan o politica $\pi(a|s)$ si doua retele $Q(s, a)$, maximizand reward-ul asteptat impreuna cu o componenta de entropie.

In cazul nostru, actiunile sunt discrete, deci am folosit o varianta Discrete SAC. Aceasta pastreaza aceeasi idee de baza ca SAC, si anume optimizarea unei politici stocastice regularizate cu entropie, dar inlocuieste actiunile continue cu o distributie peste actiunile discrete. Astfel, actorul produce direct probabilitati $\pi(a|s)$ pentru cele 3 actiuni, iar criticii estimeaza $Q(s, a)$ pentru fiecare actiune.

Implementarea poate fi sumarizata in doua etape: o implementare custom, relativ utila ca prototip dar instabila, si o implementare cu ajutorul librariei CleanRL.

5.3.1 Implementarea custom

Primele experimente au evidentiat probleme majore de implementare, cauzate de bug-uri de infrastructura (topologie configurata gresit, memory leak-uri) si de o arhitectura initial implementata incorect. Aceste erori au facut rezultatele initiale irelevante.

Metric	**Value**
Training Steps	500,000
Eval Steps	500,000
Hit Rate	49.63%
Byte Hit Rate	49.49%
Edge Actions	63.0%
Regional Actions	37.0%
Skip Actions	0.0%
Edge Occupancy	99.9%
Regional Occupancy	100.0%

Tabela 12: Baseline Custom Discrete SAC

Acest experiment reprezinta baseline-ul pentru Discrete SAC implementat de noi. Acest rezultat arata ca Discrete SAC este promitator si poate propaga corect rewardurile in environemtnul nostru.

5.3.2 Implementare Discrete SAC din CleanRL

Pentru a obtine rezultate reproductibile, am trecut la folosirea librariei CleanRL, o implementare standardizata de Discrete SAC. Aceasta a eliminat instabilitatile si a produs politici mai consistente intre rulari.

Metric	Value
Training Steps	500,000
Eval Steps	500,000
Hit Rate	48.48%
Byte Hit Rate	48.75%
Average Latency	61.41 ms
Edge Actions	72.4%
Regional Actions	19.0%
Skip Actions	8.6%

Tabela 13: Discrete SAC (CleanRL) baseline

Cresterea numarului de steps, a batch size-ului si reducerea frecventei de update au redus varianta gradientilor, permitand invatarea unor Q-values mai netede si la rezultate mai bune.

Metric	Value
Training Steps	1,000,000
Eval Steps	500,000
Batch Size	256
Update Frequency	8
Hit Rate	**51.79%**
Byte Hit Rate	**52.06%**
Average Latency	**56.13 ms**
Edge Actions	43.5%
Regional Actions	8.1%
Skip Actions	48.4%

Tabela 14: Discrete SAC configuratie

Aceasta configuratie obtine cel mai mare hit rate dintre toate rularile SAC stabile.

Cresterea target entropy a marit presiunea de explorare, ceea ce a impins politica catre actiunea cu varianta minima (*Skip*), degradand utilizarea cache-urilor.

Metric	Value
Eval Steps	1,000,000
Target Entropy Scale	0.98
Hit Rate	46.36%
Byte Hit Rate	44.98%
Average Latency	64.07 ms
Edge Actions	26.7%
Regional Actions	0.4%
Skip Actions	72.8%

Tabela 15: Discrete SAC cu entropie mare

Scaderea learning rate-ului a facut agentul mai atent la risc, favorizand Skip in detrimentul investitiilor in cache, ceea ce a dus la o politica suboptima.

Metric	Value
Eval Steps	1,000,000
Hit Rate	47.58%
Byte Hit Rate	46.51%
Average Latency	61.68 ms
Edge Actions	42.3%
Regional Actions	0.4%
Skip Actions	57.3%

Tabela 16: Discrete SAC cu learning rate redus

Cresterea capacitatii retelei a dus la un colaps al politicii spre *Always Edge*. Modelul supra-invata rewardul imediat asociat latentei mici a Edge, ignorand beneficiile pe termen lung ale Regional si Skip, ceea ce duce la performanta suboptima, asa cum am vazut si in cazul DQN si PPO.

Metric	Value
Eval Steps	1,000,000
Hit Rate	48.00%
Byte Hit Rate	46.86%
Average Latency	60.48 ms
Edge Actions	99.1%
Regional Actions	0.9%
Skip Actions	0.0%

Tabela 17: Discrete SAC cu retea mare

Ca si concluzionam, Discrete SAC cu batch size 256 si update frequency 8 reprezinta cel mai bun compromis intre explorare, selectivitate si utilizare eficienta a cache-ului. Marirea batch size-ului a redus variatia estimarii gradientului si a stabilizat actualizarile retelelor Q(s,a). Reducerea frecventei de update a limitat feedback-ul instabil dintre actor și critici,

prevenind supra-antrenarea pe tranziții corelate. Aceste efecte au produs estimări Q mai netede si mai consistente si a permis politicii SAC sa invete un trade-off bun intre Edge, Regional si Skip.

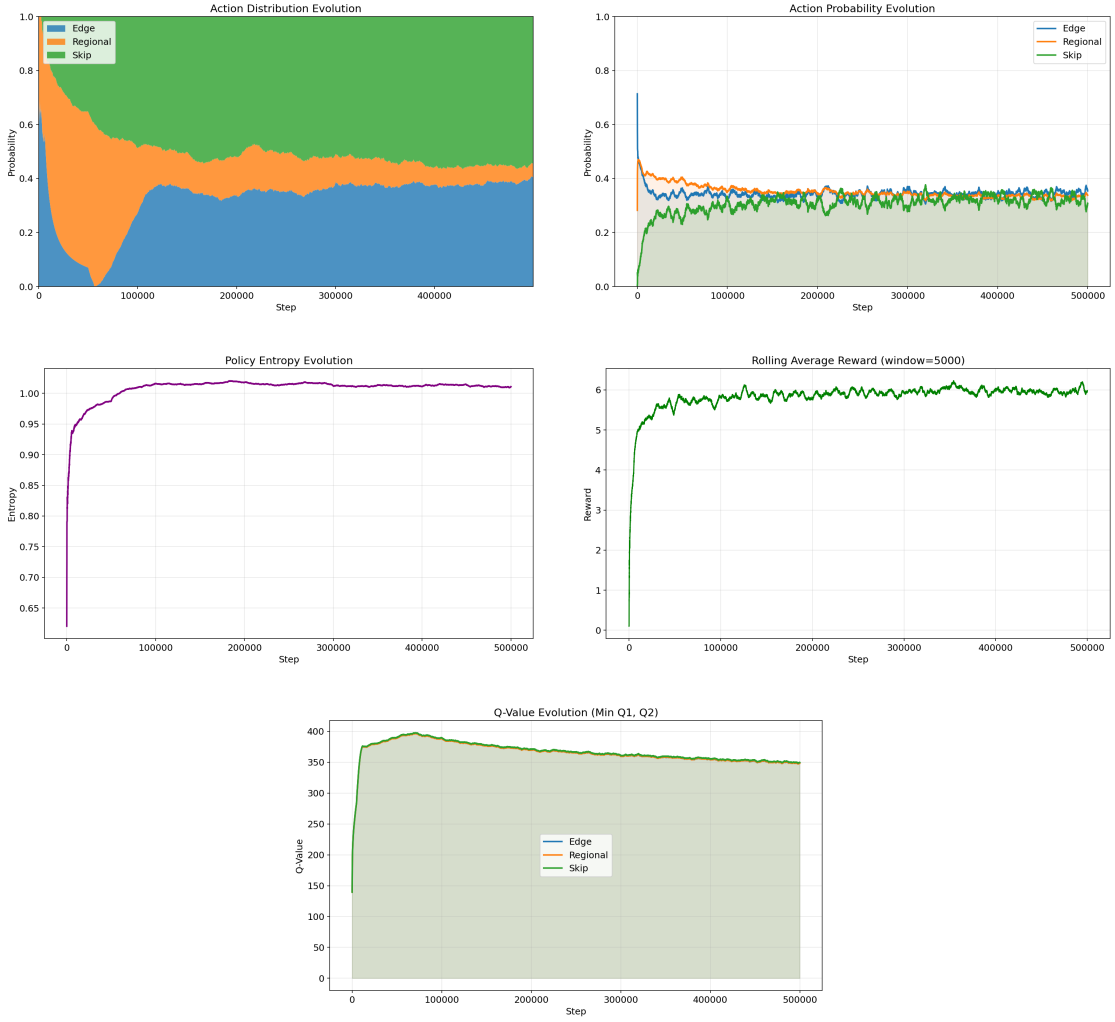

Figura 3: Analiza SAC

Aceste grafice arata ca agentul SAC CleanRL a invatat o politica stabila si bine echilibrata, fara colaps spre o singura actiune. Distributia si probabilitatile actiunilor raman apropiate, ceea ce indica o politica stocastica care face are un trade-off real intre Edge, Regional si Skip. Entropia scade gradual, semn ca explorarea este controlata si agentul converge in mod natural spre o strategie corecta. Rewardul si valorile Q se stabilizeaza, confirmand ca agentul a invatat corect valoarea pe termen lung a deciziilor.

6 Comparatii finale

Pentru a valida eficacitatea agentilor RL, am compilat cele mai bune configuratii obtinute pentru fiecare algoritm (DQN, PPO, Discrete SAC) si le-am comparat cu baseline-urile euristice si statice. Tabelul de mai jos prezinta ierarhia finala a performantei, ordonata descrescator dupa Hit Rate.

Loc	Metoda / Agent	Categorie	Hit Rate
1	SizeSplit (Median)	Euristica	52.06%
2	PPO	RL	51.97%
3	Discrete SAC (CleanRL)	RL	51.79%
4	DQN	RL	51.70%
5	Discrete SAC (Custom)	RL	49.63%
6	PercentileSplit(P90)	Euristica	46.62%
7	EdgeFirst	Euristica	43.10%
8	Probabilistic(10%)	Euristica	42.67%

Tabela 18: Comparatii finale

7 Concluzii si Directii Viitoare

In urma experimentelor realizate, am ajuns la concluzia ca optimizarea unui CDN nu este o problema usoara, iar metodele euristice clasice atunci cand sunt bine calibrate, raman extrem de competitive.

Faptul ca euristica *SizeSplit (Median)* a ocupat primul loc urmata la o diferenta foarte mica (sub 0.1%) de agentul PPO, ne arata un lucru interesant: agentul a reusit sa "redescopere" singur o regula optima. Diferenta majora este ca euristica a avut nevoie de o analiza statistica a intregului set de date pentru a calcula mediana, in timp ce agentul RL a invatat sa faca aceasta separare in mod dinamic, observand doar traficul curent.

Am observat ca algoritmii bazati pe Policy Gradient (PPO si intr-o oarecare masura SAC) sunt mult mai potriviti pentru aceasta problema decat cei bazati pe Q-learning. Decizia de a plasa un obiect pe Edge sau Regional necesita o politica stocastica, iar DQN fiind un algoritm determinist, nu a putut ramane stabil.

Pentru a concluziona, proiectul a demonstrat ca Reinforcement Learning este o solutie viabila pentru caching in retele ierarhice distribuite, capabila sa egaleze performanta unor reguli statice ideale, dar avand avantajul adaptabilitatii.

7.1 Directii de cercetare viitoare

Desi rezultatele sunt promitatoare, exista cateva directii prin care credem ca acest sistem ar putea fi imbunatatit semnificativ in viitor:

- **Arhitecturi Multi-Agent:** Un agent ar putea fi dedicat exclusiv deciziei de plasare, in timp ce un agent separat ar gestiona politica de eviction pe nodurile locale. Aceasta separare reduce complexitatea spatiului de actiuni si permite optimizarea independenta a celor doua sisteme critice in mod complementar.
- **Control Ierarhic (Meta-Control):** O solutie posibil mai eficienta este ca agentul RL sa ajusteze doar periodic parametrii macro, spre exemplu, pragul de dimensiune pentru split.

Bibliografie

[1] G. Urdaneta, G. Pierre, and M. van Steen. *Wikipedia Workload Analysis for Decentralized Hosting*. Elsevier Computer Networks, Vol. 53, No. 11, pp. 1830-1845, 2009. `http://www.globule.org/wikibench/`

[2] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor*. International Conference on Machine Learning (ICML), 2018.

[3] P. Christodoulou. *Soft Actor-Critic for Discrete Action Settings*. arXiv preprint arXiv:1910.07207, 2019.

[4] A. Raffin, A. Hill, A. Gleave, et al. *Stable-Baselines3: Reliable Reinforcement Learning Implementations*. Journal of Machine Learning Research, 22(268):1–8, 2021.

[5] S. Huang, R. F. J. Dossa, C. Ye, et al. *CleanRL: High-quality Single-file Implementations of Deep Reinforcement Learning Algorithms*. Journal of Machine Learning Research, 23(274):1–18, 2022.