RL Project

1. Overview of the Cache Replacement MDP

We simulate a small cache of size CCC serving requests for NNN distinct pages.

At each time step:

1. A page request arrives.
2. The agent must decide (if miss + cache full) **which page to evict**.
3. The reward is **+1 for hit**, **−1 for miss**.
4. The next request is drawn from a stationary distribution.

This scenario is fully Markov if you define the state correctly.

2. State Space $\mathcal{S}$

The environment's internal situation at time ttt is fully determined by:

- The **current cache contents**
- The **identity of the requested page**

We encode each as one-hot/binary vectors.

2.1 Cache Status Vector

Let:

- NNN = number of unique pages (e.g., 10)
- $\mathbf{c}_t \in \{0,1\}^N where : ct[i] = 1$ if page i is currently in the cache otherwise 0

This is a **binary membership vector**, not recency counters, which keeps state small and fully Markov.

2.2 Request Vector

Let current request be page index $p_t \in \{0, \ldots, N-1\}$

Represent as a one-hot vector:

$$\mathbf{p}_t \in \{0,1\}^N, \quad p_t[i] = 1[i = \text{requested page}]$$

2.3 Full State

Define:

$$s_t = (\mathbf{c}_t, \mathbf{p}_t)$$

which is a vector in:

$$S = \{0,1\}^{2N}$$

This is the *entire* information needed to produce the next state given an action, assuming future requests follow a stationary distribution.

We are assuming that at a time only 1 page is requested denoted by p_t. (one-hot vector)

We are not representing p_t as a single index because neural networks dont like categorical integers better to convert them into vectors.

- **State**:
  - `cache_slots`: array of length `C`, each entry is a page index in `{0,...,N-1}` or a special "empty" token.
  - `current_request`: page index or one-hot vector.
  - In report: you can encode this as a concatenated one-hot representation or as discrete features.
- **Action**:
  - $\mathcal{A} = \{0, 1, \ldots, C-1\}$
  - $a_t = j$ evict / overwrite slot `j` if eviction is required.
- **Transition cases**:
  1. **Hit**:

- `current_request` already in `cache_slots`.
- `cache_slots` unchanged.
- `a_t` ignored in env.

2. **Miss & not full**:
   - Place `current_request` into first empty slot.
   - `a_t` ignored.
3. **Miss & full**:
   - Evict page in `cache_slots[a_t]`, replace with `current_request`.

- **Reward**:
  - `+1` for hit, `1` for miss (same as before).

✅ Transition Dynamics $P(s_{t+1} \mid s_t, a_t)$

Given current state:

$s_t = (\mathbf{c}_t, \mathbf{p}_t)$

and action a_t:

Step 1 — Determine Hit/Miss

Let requested page be p_t.

Hit:

$$\text{hit}_t = 1 \quad \text{if } c_t[p_t] = 1$$

Miss:

$$\text{hit}_t = 0$$

Step 2 — Cache Update Rule

Define new cache vector:

- If hit : $c_{t+1} = c_t$
- If miss and cache not full:

  Insert page:

  $c_{t+1}[p_t] = 1$
- If miss and cache full:
  - Evict page indicated by action

    $c_{t+1}[a_t] = 0$

    a_t is chosen from one of the cache pages
  - Insert request page

    $c_{t+1}[p_t] = 1$

Other entries remain as in c_t

Step 3 Sample Next Request

We assume a stationary request distribution D:

$$p_{t+1} \sim D$$

Can be uniform over pages

Step 4 — Construct Next State

$s_{t+1} = (\mathbf{c}_{t+1}, \mathbf{p}_{t+1})$

Final transition function

$$P(s_{t+1} \mid s_t, a_t) = \begin{cases} 1 & \text{if caches updated deterministically, and } p_{t+1} \text{ is sampled according to } D \\ 0 & \text{otherwise} \end{cases}$$

Cache evolution is deterministic given action and hit/miss.

Only the next request is stochastic.

✅ 5. Reward Function $R(s_t, a_t)$

Very simple and direct:

$$r_t = \begin{cases} +1 & \text{if } p_t \in \text{cache} \\ -1 & \text{otherwise} \end{cases}$$

Equivalent to maximizing cache hit rate.

You can argue:

- hit -
- miss → cost

Some people use 0 for hit and –1 for miss; either is valid.

We stick with (+1 / –1) because it stabilizes policy gradient normalization.

✅ 6. Episode Structure

Define an episode horizon T (e.g., 100 requests):

- Start with:

$$\mathbf{c}_0 = \mathbf{0}, \quad p_0 \sim D$$

- Run for $t = 0, 1, \ldots, T-1$.

Discount factor:

$$\gamma = 0.99$$

The return used in REINFORCE is:

$$G_t = \sum_{k=t}^{T-1} \gamma^{k-t} r_k$$