OpenEuler - Track 2 ‘Euler’s Eye’

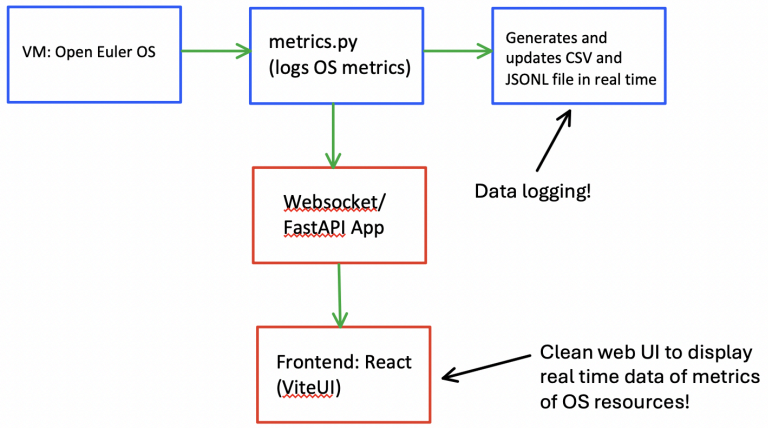
Created by Team Vertex:

- Aarav Parin
- Rory Condict
- Shashwat Bhatnagar
- Zeki Kam
- Zarif Ahmed

Tier 1 - “Forge the Eye”

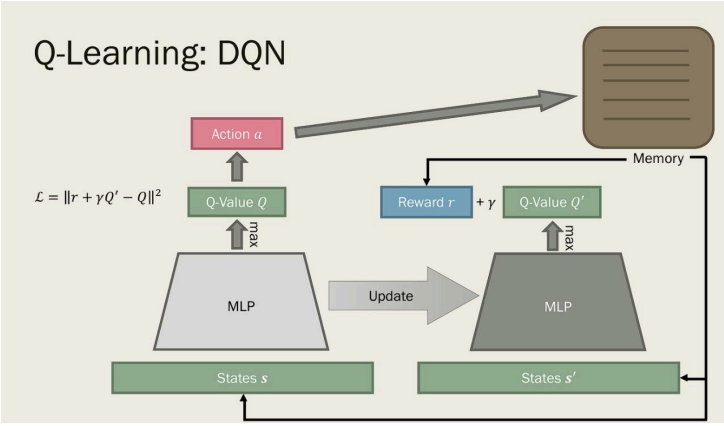
A lightweight monitoring agent that collects and exposes key system metrics (i.e. CPU usage (total and per core), memory usage, disk read and write speeds, network receive and transmission rate, GPU telemetry (usage, memory usage, temperature and power draw).]

^ A clean and modern UI that presents the collected data in a concise way.



^ A high level overview of the structure and flow of the project.

Tier 2 - “Build the Brain”



Executive Summary

Our platform leverages Deep Q-Learning to analyze openEuler system performance through real-time telemetry analysis and intelligent configuration tuning by adopting ML powered cache replacement technique

Problem Statement

Current cache replacement relies on deterministic algorithms that cannot capture complex, non-linear patterns in openEuler telemetry data. This results in suboptimal performance despite available CPU/GPU/memory/IO metrics.

Our Approach

- Formulated as Reinforcement Learning problem with states (telemetry features), actions (tuning knobs), and rewards (performance metrics)

- Implemented DQN with replay buffer and target network

- Optimized via Optuna hyperparameter tuning

- Outperforms deterministic heuristic baseline

States and Actions

State Space

- Suppose C slots for caching.
- State transitions happen on cache misses.
- Denote state as $\boldsymbol{s} = \{\mathcal{F}_0, \mathcal{F}_1, \dots, \mathcal{F}_c\}$
  - Features descriptors to represent large state space.
  - $\mathcal{F}_0$ is the feature vector of currently requested resource.
  - $\mathcal{F}_1, \dots, \mathcal{F}_c$ is the feature vector of each cache slot.
  - In our design, $\mathcal{F}_i = \{f_i^s, f_i^m, f_i^l\}$, where f_i^s, f_i^m, f_i^l represent the recent access times within short, middle, long terms, respectively.

Action Space

- Action is taken when caches conflict.
- Denote action as $a \in \{1, \dots, C\}$.
  - a indicates which cache slot should be replaced.

Results & Impact

Preprocessing & Feature Engineering

We have reduced the number of features from 7 to 3 via feature engineering. This results in improved stability and speed. Used Optuna for **Hyperparameter autotuning**.

Evaluation

Method Avg Reward Throughput ↑ Latency ↓

Key Takeaways

- 10–20% performance gain (replace with actual).
- Generalizable across workloads.
- Lightweight inference, production-ready.
- Lightweight inference, production-ready.

Future Work

- Double DQN, prioritized replay.
- Extend to continuous actions & more knobs.

Rewards

- Rewards evaluate the replacement choice.
- Denote rewards of a transition as $R(s, a, s') = H(s, s') + \lambda \cdot P(s, a, s')$.
  - $H(s, s') = \sum_i (n'_i - n_i)$ evaluates the total number of hits between two following cache misses.
    - n_i represent the number of hits at slot i for state s.
    - n'_i represent the number of hits at slot i for state s'.
  - $P(s, a, s')$ is the penalty of action a, resulting in the next cache miss.
    - i.e. $P(s, a, s') \neq 0$ if replacement at s by a causes the cache miss at s'.
    - It should decrease as cache hits $H(s, s')$ accumulate.
    - e.g. $P(s, a, s') = \mu + \frac{\psi}{H(s,s')}$ if a causes the next cache miss, where $\psi < 0$.