International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 10 October 2026
Page No: 5723 -5728
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0743

Model-Based Q-Learning for Dynamic Cache Allocation in Multi-Tenant Edge Computing

Pooranakalai S[1], Safrin Suhaina A[2], Surbhi Kumari[3], Ms. R. Punithagowri[4]
[1]Department of Computer Science and Engineering, Erode Sengunthar Engineering College, Erode - 638 057, Tamil Nadu, India
[2,3,4]Assistant Professor, Department of Computer Science and Engineering, Erode Sengunthar Engineering College, Erode - 638 057, Tamil Nadu, India
Emails: *spooranakalai@gmail.com[1], safirnsuhaina515@gmail.com[2],skholistic012@gmail.com[3], punithagowri@email.com[4]*

Abstract
In multi-tenant edge computing, several Service Providers (SPs) share a limited cache. Because the traffic demand of each SP changes over time, a fixed cache partition can leave space unused for one SP while another SP suffers cache misses and longer response delay. This paper proposes a dynamic cache allocation framework that monitors the traffic demand and cache usage of every SP and moves unused cache space to the SPs that need it. Allocation decisions are supported by model-based reinforcement learning (MB-RL) built on Q-learning. A regression model estimates cost or performance from system conditions, experience replay reuses past experiences during learning, and an epsilon-greedy policy balances trying new actions with choosing learned ones. Every candidate action is checked against two hard rules before it is applied: the total allocation must not exceed the cache capacity, and every SP must keep a minimum cache share. Fair sharing is therefore enforced by construction rather than left to the learning signal. The paper presents the system architecture, the allocation procedure, a worked example with a 100 GB cache, and a planned evaluation based on cache utilization, response delay, cache hit ratio, and fairness. The framework is expected to improve cache usage, reduce unnecessary cache misses and response delay, and support fair sharing among tenants compared with fixed allocation.

Keywords: *Cache allocation; Edge computing; Experience replay; Model-based reinforcement learning; Multi-tenant; Q-learning.*

1. Introduction

Edge computing places storage and computation close to users so that popular content and services can be delivered with low delay and without repeatedly crossing the core network [1], [2]. An edge node, however, has limited cache space, and in a multi-tenant deployment this space is shared by several Service Providers (SPs). Caching is therefore not only a question of which content to keep, but also of how much space each SP should be given. The traffic demand of an SP is rarely constant. It rises and falls with time of day, popularity of content, and user activity. When the cache is split into fixed parts, an SP with high demand may not receive enough space, while another SP holds space it does not use. The result is poor cache utilization, avoidable cache misses, and longer response delay. Allocation that follows demand is attractive, but it must not let a busy SP starve the others. This paper proposes a dynamic cache allocation framework for multi-tenant edge computing. The system monitors the traffic of each SP, identifies which SP needs more cache and which has unused cache, and reallocates space accordingly. Reinforcement learning supports the choice of allocation action, and a minimum cache share is kept for every SP at all times. The main contributions are as follows: a demand-driven reallocation scheme with real-time traffic monitoring, dynamic reallocation, and a guaranteed minimum cache allocation; a model-based Q-learning formulation that combines a regression cost model, experience replay, and epsilon-greedy exploration; a constraint-checking step that validates capacity and minimum shares before any change is applied; and an evaluation plan built on cache utilization, response

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 10 October 2026
Page No: 5723 -5728
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0743

delay, cache hit ratio, and fairness. The rest of this paper is organized as follows. Section 2 reviews the related work. Section 3 states the problem. Sections 4 and 5 describe the proposed system and its architecture. Section 6 presents the allocation methodology with a worked example. Section 7 defines the performance metrics and the evaluation plan, Section 8 discusses limitations, and the final section concludes the paper.

2. Related Work

2.1.Edge computing and caching

Satyanarayanan [1] and Shi et al. [2] describe edge computing as a response to the latency and bandwidth limits of centralized clouds, and identify resource management at the edge as a central challenge. Mach and Becvar [3] survey mobile edge computing architectures and offloading, and Mao et al. [4] review the field from the communication perspective. Podlipnig and Boszormenyi [5] survey cache replacement strategies, which decide what to evict within a cache of given size. These strategies assume the size of the cache is fixed; the question of how to divide that size among competing tenants is left open.

2.2.Reinforcement learning for caching and allocation

Sutton and Barto [6] give the standard treatment of reinforcement learning. Watkins and Dayan [7] introduced Q-learning, which learns action values from experience without a model of the environment. Lin [8] proposed experience replay, which stores past transitions and reuses them for learning, and Mnih et al. [9] used it to stabilize deep Q-learning. Sutton [10] showed that a learned model of the environment can generate additional simulatedexperience to speed up learning, and Moerland et al. [11] survey model-based reinforcement learning. Zhu et al. [12] review the use of deep reinforcement learning for mobile edge caching and discuss open issues.

2.3.Fair sharing and research gap

Ghodsi et al. [13] study fair allocation of shared resources among users and show why guarantees to each user matter when demand differs. Existing learning-based caching work mainly focuses on content placement and does not commonly describe an allocation step that moves unused space between tenants while enforcing a minimum share for each. The present work combines model-based Q-learning with explicit capacity and minimum-share checks for this purpose. As shown in Table 1 summarizes the closest studies.

Table 1 Summary of related studies and their relevance to this work

Ref.	Main contribution	Relevance to this work
[1], [2]	Vision and challenges of edge computing	Motivates shared edge cache
[5]	Survey of cache replacement strategies	Fixed-size cache assumption
[7]	Q-learning algorithm	Core learning method
[8]	Experience replay	Reuse of past experience
[10]	Learned model for planning	Model-based learning idea
[12]	Deep RL for mobile edge caching	Closest application area
[13]	Fair multi-resource allocation	Fairness among tenants

3. Problem Statement

When cache space is fixed, an SP with high demand may not get enough space, while another SP may have unused space. This reduces cache utilization and increases response delay. The system needs to use the available cache efficiently by reallocating unused space when demand changes, while maintaining fair sharing. The research problem is how to decide, at run time, how much of a shared edge cache each SP should hold, given changing
demand, a fixed total capacity, and a minimum share for every SP.

The objectives of this work are therefore:

- to use cache space efficiently
- to adjust allocation when traffic demand changes
- to reduce unnecessary cache misses and response delay
- to maintain a minimum cache share for every SP

4. Proposed System

4.1.Overview and new features

The proposed system continuously observes the

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 10 October 2026
Page No: 5723 -5728
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0743

traffic demand and cache usage of each SP. When it detects that one SP needs more cache and another holds unused cache, it selects an allocation action with MB-RL and Q-learning, validates it, applies it, and observes the effect on cache usage and response delay. This result is fed back to improve later decisions. Figure 1 shows the eight steps of this cycle. Three features distinguish the system. Real- time traffic monitoring checks the traffic demand of each SP. Dynamic reallocation moves unused cache space to an SP that needs more. Minimum cache allocation ensures that every SP receives a basic amount of cache.

4.2.Algorithmic components

Five techniques work together. Their roles are listed in Table 2. MB-RL helps the system choose cache allocation actions. Q-learning learns from feedback and improves future decisions. A regression model estimates cost or performance from system conditions. Experience replay reuses past experiences during learning. Epsilon-greedy selection balances trying new actions against choosing learned actions.

Table 2 Algorithms and techniques used

Technique	Use in the system
Model-based RL (MB-RL)	Helps the system choose cache allocation actions
Q-learning	Learns from feedback and improves future decisions
Regression model	Estimates cost or performance from system conditions
Experience replay	Reuses past experiences during learning
Epsilon-greedy	Balances trying new actions and choosing learned actions

4.3.Novel contribution

The key feature of this work is that learning and fairness are kept separate. The learning components propose allocation actions to reduce delay and misses, while a rule-based checker guarantees that the total allocation never exceeds capacity and that every SP keeps its minimum share. A poorly chosen action, for example during exploration, therefore cannot break the fair-sharing guarantee.

5. System Architecture

The architecture, shown in Figure 2, has five parts. The SPs generate requests that are served from the shared edge cache. A traffic and cache-usage monitor measures demand and usage for each SP and forms the state. The MB-RL allocation agent contains the Q-learning component, the regression cost model, the experience replay buffer, and the epsilon-greedy action selector. A constraint checker validates the selected action against the capacity and minimum-share rules. The cache controller applies valid actions to the shared cache. Cache utilization, response delay, and hit ratio measured after the change return to the monitor as feedback.

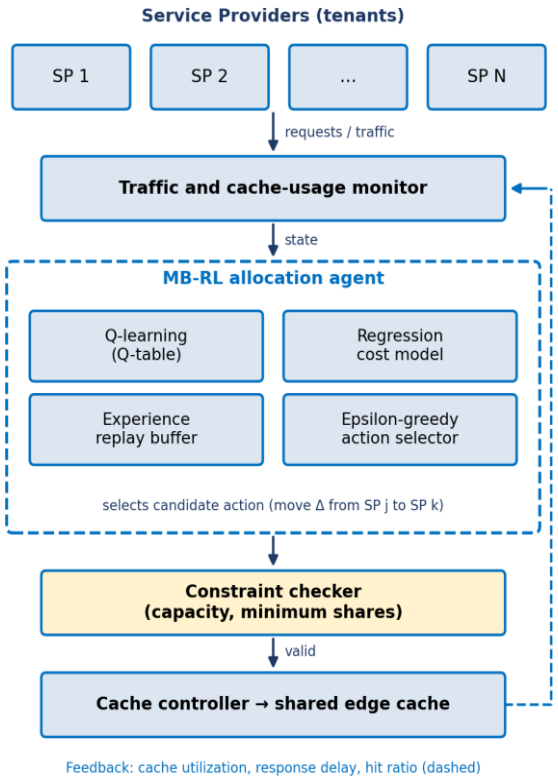


Figure 2 Proposed system architecture

6. Allocation Methodology

6.1.Capacity and minimum-share constraints

Let N be the number of SPs, C the total cache capacity, ci the cache allocated to SP i, and mi the minimum share of SP i. Every allocation must satisfy:

$$\sum_{i=1}^{N} c_i = C \qquad (1)$$

$$c_i \geq m_i, \quad \forall\, i \qquad (2)$$

Because the minimum shares must fit in the cache, the system requires that the sum of mi does not

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 10 October 2026
Page No: 5723 -5728
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0743

exceed C. The surplus, C minus the sum of mi, is the part of the cache that can be moved between SPs freely.

6.2.State, action, and reward

The state at decision time t contains, for each SP, its current traffic demand, its cache usage, and its current allocation. In a tabular Q-learning design these values are discretized into levels. An action moves a block of Δ GB from a donor SP j to a receiver SP k, or leaves the allocation unchanged. The effect of a move is:

$$cj \leftarrow cj - \Delta,\ ck \leftarrow ck + \Delta \quad (3)$$

An action is feasible only if the donor keeps its minimum share, that is cj − Δ ≥ mj. Infeasible actions are removed before selection (step 5 of Figure 1), so equations (1) and (2) hold after every change. After the change is applied, the system observes the average response delay D, the miss ratio M = 1 − H (H is the cache hit ratio), and the idle cache fraction I. The reward is the negative of a weighted cost:

$$r = -(w1D + w2M + w3I) \quad (4)$$

The weights w1, w2, and w3 are non-negative and set during evaluation. The idle fraction I rewards moving unused space to SPs that need it, and D and M reward lower delay and fewer misses.

6.3.Model-based Q-learning

The agent keeps an action-value function Q(s, a). After each real step it computes the temporal-difference error and updates the value:

$$\delta = r + \gamma \max_{a'} \underline{Q}(s', a') - \underline{Q}(s, a) \quad (5)$$

$$\underline{Q}(s, a) \leftarrow \underline{Q}(s, a) + \alpha\delta \quad (6)$$

Here α is the learning rate and γ is the discount factor. The model-based part is a regression model that estimates the cost of taking action a in state s from observed system conditions:

$$\hat{c}(s, a) = f\theta(s, a) \quad (7)$$

The regression model is fitted to past observations of state, action, and measured cost. Once fitted, it lets the agent estimate the outcome of candidate actions and generate extra simulated experience for additional Q updates, so fewer real changes to the cache are needed. Experience replay stores real transitions and samples them repeatedly for learning, which makes better use of each observation and reduces the effect of short-term fluctuations in traffic. Action selection is epsilon-greedy. With probability ε the agent chooses a random feasible action to explore. With probability 1 − ε it chooses the feasible action with the highest estimated value. A large ε early on allows the agent to explore, and ε is reduced as learning progresses.

6.4.Allocation procedure

The system repeats the following steps whenever traffic demand changes.

- Check the total cache capacity and set a minimum share for each SP.
- Monitor the traffic demand and cache usage of all SPs.
- Identify which SP needs more cache and which SP has unused cache.
- Use MB-RL and Q-learning to select a suitable allocation action.
- Check that the total allocation does not exceed capacity and that minimum shares are maintained.
- Apply the allocation change and observe cache usage and response delay.
- Use the result as feedback to improve future decisions.
- Repeat the process whenever traffic demand changes.

6.5. Worked example

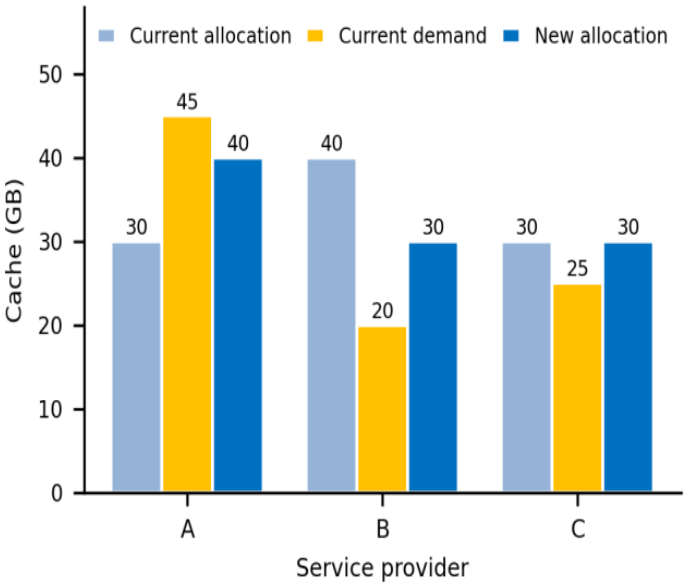
Consider a 100 GB cache shared by three SPs, as in Table 3 and Figure 3. SP A has a demand of 45 GB but holds 30 GB, while SP B has a demand of 20 GB but holds 40 GB. The agent selects the action that moves Δ = 10 GB from B to A. The checker confirms that B keeps at least its minimum share after the move and that the total stays at 100 GB. SP C already holds 30 GB against a demand of 25 GB and is unchanged. After the change A holds 40 GB, B holds 30 GB, and C holds 30 GB. The minimum shares are not specified in this example; the check passes for any minimum share of up to 30 GB per SP. As shown in Table 3 Worked example of one reallocation step.

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 10 October 2026
Page No: 5723 -5728
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0743

Figure 3 Cache allocation before and after the reallocation step.

Table 3 Worked example of one reallocation step

SP	Current allocation	Current demand	New allocation
A	30 GB	45 GB	40 GB
B	40 GB	20 GB	30 GB
C	30 GB	25 GB	30 GB
Total	**100 GB**	**-**	**100 GB**



7. Performance Metrics and Evaluation Plan

Four metrics are used to judge the system, as listed in Table 4. Cache utilization shows how much of the cache is being used. Response delay is the time taken to serve a request. The cache hit ratio is the share of requests served from cache:

$$H = \frac{requests\ served\ from\ cache}{total\ requests} \quad (8)$$

Fairness measures whether each SP receives its minimum cache share. It can be reported as the fraction of decision steps in which equation (2) holds for all SPs, and it is expected to be 100% by design. As shown in Table 4 Performance metrics.

Table 4 Performance metrics

Metric	Meaning
Cache utilization	How much cache is being used
Response delay	Time taken to serve a request
Cache hit ratio	Share of requests served from cache
Fairness	Whether each SP receives its minimum cache share

The evaluation is planned as a simulation of a shared edge cache with several SPs and time-varying traffic. The proposed method will be compared with a fixed allocation, and with a demand-proportional allocation that does not learn. Table 5 lists the planned experiments. No experimental results are reported in this paper. As shown in Table 5. Planned evaluation.

Table 5 Planned evaluation

Dimension	Planned checks
Baselines	Fixed equal allocation; demand-proportional allocation without learning
Workloads	Stable demand; gradual change; sudden surge in one SP
Main metrics	Utilization, response delay, hit ratio, fairness
Constraints	Capacity and minimum-share violations (expected zero)
Learning	Effect of ε, replay buffer size, regression accuracy
Sensitivity	Block size Δ, number of SPs, minimum share

8. Discussion And Limitations

Dynamic reallocation addresses the main weakness of fixed partitioning: space held by an SP with low demand can serve an SP with high demand. Keeping a minimum share protects each SP, and the check before each change keeps the total within capacity. Separating the learning signal from the constraints means that exploration cannot violate fair sharing. The work has several limitations. The approach is described and has not yet been evaluated, so its gains are expected rather than demonstrated. The reward weights and the block size Δ require tuning. Tabular

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 10 October 2026
Page No: 5723 -5728
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0743

Q-learning needs the state to be discretized, which can become coarse as the number of SPs grows. Moving cache space may also cause short-term misses while the contents of the cache are adjusted, an effect that the evaluation should measure.

Conclusion And Future Work

This paper presented a dynamic cache allocation framework for multi-tenant edge computing. The system monitors traffic, identifies changing cache needs, and reallocates unused cache to SPs that need it. MB-RL and Q-learning, supported by a regression model, experience replay, and epsilon-greedy selection, choose the allocation actions. Capacity and minimum-share rules are checked before any change is applied. The approach aims to improve cache usage, reduce delay, and support fair sharing. Future work includes implementing the simulation and reporting results against the baselines in Table 5, extending the state representation with function approximation for larger numbers of SPs, and studying weighted or priority-based minimum shares.

References

[1]. Satyanarayanan, M. (2017). The emergence of edge computing. Computer, 50(1), 30-39. https://doi.org/10.1109/MC.2017.9

[2]. Shi, W., Cao, J., Zhang, Q., Li, Y., & Xu, L. (2016). Edge computing: Vision and challenges. IEEE Internet of Things Journal, 3(5), 637-646. https://doi.org/10.1109/JIOT.2016.257919 Mach, P., & Becvar, Z. (2017).

[3]. Mobile edge computing: A survey on architecture and computation offloading. IEEE Communications Surveys & Tutorials, 19(3),1628-1656. https://doi.org/10.1109/COMST.2017.2682318

[4]. Mao, Y., You, C., Zhang, J., Huang, K., & Letaief, K. B. (2017). A survey on mobile edge computing: The communication perspective. IEEE Communications Surveys & Tutorials, 19(4), 2322-2358. https://doi.org/10.1109/COMST.2017.2745201

[5]. Podlipnig, S., & Boszormenyi, L. (2003). A survey of Web cache replacement strategies. ACM Computing Surveys, 35(4), 374-398. https://doi.org/10.1145/954339.954341

[6]. Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction (2nd ed.). MIT Press.

[7]. Watkins, C. J. C. H., & Dayan, P. (1992). Q-learning. Machine Learning, 8(3-4), 279-292. https://doi.org/10.1007/BF00992698

[8]. Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. Machine Learning, 8(3-4), 293-321. https://doi.org/10.1007/BF00992699

[9]. Mnih, V., Kavukcuoglu, K., Silver, D., et al. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529-533.https://doi.org/10.1038/nature14236 Sutton, R. S. (1990).

[10]. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In Proceedings of the 7th International Conference on Machine Learning (pp. 216-224). Morgan Kaufmann.

[11]. Moerland, T. M., Broekens, J., Plaat, A., & Jonker, C. M. (2023). Model-based reinforcement learning: A survey.

[12]. Foundations and Trends in Machine Learning, 16(1), 1-118. https://doi.org/10.1561/2200000086

[13]. Zhu, H., Cao, Y., Wang, W., Jiang, T., & Jin, S. (2018). Deep reinforcement learning for mobile edge caching: Review, new features, and open issues. IEEE Network, 32(6), 50-57. https://doi.org/10.1109/MNET.2018.1800109

[14]. Ghodsi, A., Zaharia, M., Hindman, B., Konwinski, A., Shenker, S., & Stoica, I. (2011). Dominant resource fairness: Fair allocation of multiple resource types. In Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI) (pp. 323-336).