Journal of Theoretical and Applied Information Technology
15st September 2026. Vol.104. No.17

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

SWIFTCACHE: AN RL-POWERED ADAPTIVE CACHING FRAMEWORK FOR LOW LATENCY CONTENT DELIVERY IN ECOMMERCE NETWORK

DR. P. MAHESWARI[1], Dr. V.SEEDHA DEVI[2], DR. AKILA VENKATRAMAN[3], KANCHARLA NAGABABU[4], DR. PERURI VENKATA ANUSHA [5], ELANGOVAN MUNIYANDY [6]

[1]Assistant Professor, Department of Commerce,Faculty of Science and Humanities, SRM Institute of Science and Technology, Ramapuram,Chennai,India.
[2] Associate Professor, Department of Information Technology, Jaya Engineering College, Chennai, Tamil Nadu, India.
[3] Associate Professor, Department of Computer Science and Engineering, Gokaraju Rangaraju Institute of Engineering & Technology, Nizampet Road, Bachupally, Kukatpally, Hyderabad- 500090, Telangana State, India.
[4] Department of Computer Applications, Aditya University, Surampalem, India.
[5] Assistant Professor, Department of Computer Science and Engineering, Koneru Lakhmaiah Education Foundation, Vaddeswaram, India.
[6]Department of Biosciences, Saveetha School of Engineering. Saveetha Institute of Medical and Technical Sciences,Chennai,India.
Email :[1]maheshdiamond23108@gmail.com, [2]seethaitjec@gmail.com , [3] akila_be@yahoo.co.in, [4]nagababu.kancharla@adityauniversity.in, [5] anushaperuri@gmail.com , [6] muniyandy.e@gmail.com

ABSTRACT

Modern e-commerce networks experience rapidly changing product demand, geographically distributed requests, heterogeneous content, and freshness-sensitive information, creating challenges for existing reinforcement-learning-based caching approaches that are mainly designed for generic edge, CDN, or service-caching environments. This paper presents SwiftCache, an adaptive caching system powered by RL. It has been designed specifically to low-latency e-commerce content transmission and takes into account factors like network latency, content size, originality, cache state, and dynamic product popularity (as well as regional demand). A CDN/network simulation layer is utilised to account in edge nodes, geographic regions, cache capacity, heterogeneous objects, uniqueness, and network parameters. The RetailRocket E-Commerce Dataset and the M5 Walmart Dataset are utilised to model demand patterns and real user-product request behaviour, respectively, in the development of a trace-driven e-commerce CDN environment. Three current RL-based caching techniques are compared to SwiftCache under the same experimental conditions: EC-MADRL, Adaptive Layer-Wise Personalised Federated DRL, and Resource-Aware DRL. According to the simulated findings, SwiftCache attains 91.8% CHR, 90.7% BHR, 90.8% regional CHR, 48.6 ms P95 latency, and 3.1% FVR. SwiftCache increases BHR by 6.83–14.66%, lowers FVR by 54.41–64.37%, lowers P95 latency to 26.03–35.03%, and increases CHR by 6.62–11.41% in comparison to the three baselines. For refined and consistent low-latency content delivery, these results indicate that using demand, geography, content, and freshness factors that are specific to e-commerce in RL-based caching is better than incorporating generalised RL caching strategies.
Keywords: *E-Commerce Networks, SwiftCache, Network Latency, Cache Capacity, Request Behaviour*

1. INTRODUCTION

The rapid expansion of e-commerce has transformed content delivery from a relatively static web-serving problem into a highly dynamic request-serving environment [1]. Requests of product pages, photos, suggestions, branded content, inventory data, and other online items whose validity and popularity can fluctuate at various rates may be

handled concurrently by an e-commerce platform [2]. A small proportion of products could encounter significant increases in the volume of requests during product launches, marketing initiatives, seasonal events, or unexpected spikes in demand [3]. Because of this, traditional caching policies are being questioned because making decisions based only on historical frequency or recency may not be able to respond fast enough to new demand. More generally, latency, storage constraints, and fluctuating content popularity are the main obstacles to adaptive content distribution, according to recent edge-caching analysis.

The solution to latency and network-load issues is edge caching and content delivery networks which help to bring data closer to users. The application of RL to caching has grown in recent years due to the fact that an RL agent can learn a caching policy from changes in the state of the system, instead of just having to pre-programmed with a fixed policy. This trend of dynamic popularity prediction, cooperative multi-agent caching, federated learning, and resource-aware caching has been further developed in recent studies. For instance, cooperative multi-agent DRL is explored in distributed edge caching, and a 2026 study on personalized federated DRL points out heterogeneous edge environments and growing edge caching action spaces as key challenges. Likewise, recent DRL research for resources investigates caching and service placement together, and takes latency, resource availability and cache state into account. These methods, however, are usually tested in an edge, MEC, video or service-oriented setting. The role of explicitly modelling uncertain and changing popularity has also been recognised recently in the context of adaptive caching research, but this has been mainly for video/content-streaming applications, not for heterogeneous e-commerce objects.

This research problem is how to design an adaptive caching policy that can make content-placement decisions when the demand for the content is dynamic, the content is geographically variable, object sizes are variable, the content is fresh, the cache capacity is limited, and the delivery is latency sensitive [4]. The problem can be interpreted as a sequential decision-making process in which an edge node has to select the objects to be retained, cached, prefetched, evicted, or replicated with a balance of the competing goals. A popular product, which was not yet in the cache, might be worth caching as soon as it becomes popular; however, large objects can take up a lot of space and price or inventory information may need to be pulled from the cache within a short period [5]. Therefore, just getting the highest cache-hit ratio is not enough. The caching policy should rather take into account the efficiency of the caching, the tail latency, the use of network resources, regional demand, and the validity of the information that is kept in the cache [6]. Recent research on caching mechanisms based on RL concepts shows that the problem formulation for caching latency, cache efficiency, resource constraints and changing popularity are strongly inter-related, however, none of these studies focus specifically on the combined characteristics of e-commerce delivery introduced in this work [7].

Within the present context, this research makes a contribution of an e-commerce specific, trace-driven evaluation and adaptive caching framework. Experimental workload is built from two complementary real-world datasets: the RetailRocket user-product interaction behavior is time-stamped, and the M5 Walmart data represent demand characteristics like temporal and event-based variation [8]. These sources are not directly combined with a single source but rather complementary properties of these sources are mapped to a controlled CDN/network environment with geographically distributed edge nodes, limited cache capacity, varying content sizes, freshness constraints and network characteristics [9]. This setting will allow the evaluation of the recently adopted RL caching methods in the same e-commerce-oriented setting as the proposed method. This design is relevant as recent research has shown promising results for RL when deployed on specific workloads like edges or video workloads, yet very little has been studied on how to transfer these policies to dynamic and heterogeneous e-commerce content delivery [10].

One solution to this issue is the SwiftCache framework, which takes into account e-commerce-related context while making caching decisions. Product popularity, regional demand, object size, content quality, freshness, cache occupancy, network latency, & bandwidth are all factors that are taken into account in its state representation. In order to solve the problem of scalability related to judgements made at the catalogue level, SwiftCache finds a manageable group of valuable candidate objects before it chooses the best caching action available [11]. The current action-space difficulty in federated DRL caching analysis inspired this design, which strives to balance latency, caching efficiency, as well as system resources using a multi-objective formulation that is in line with larger research on resource-aware RL caching.

Since RL-based caching is already a well-established study area, SwiftCache's originality goes beyond just using reinforcement learning for caching. Rather, the innovation is in combining an adaptive and scalable caching decision method with workload characteristics unique to e-commerce [12]. Specifically, the framework unifies network latency, varied object sizes, content freshness and relevance, regional demand, and dynamic changes in product popularity into a single caching policy [13]. The candidate-selection process is meant to reduce the action space at the catalogue level, and the multi-objective reward is meant to keep performance from being optimised for cache hits at the cost of latency, bandwidth usage, origin load, or freshness. This places SwiftCache in the middle of current research streams that focus on resource-aware edge optimisation, heterogeneous caching, distributed learning, or dynamic popularity independently [14]. An adaptive caching system powered by RL, SwiftCache, is developed and evaluated in this research. Its major purpose is to reduce latency in dynamic e-commerce CDN situations while retaining high cache performance and content freshness [15]. Under typical trace-driven scenarios including dynamic consumption, geographically distributed demands, heterogeneous content, limited cache capacity, and freshness requirements, the study seeks to ascertain whether an e-commerce-aware caching policy will outperform recent RL-based methods. In order to determine if the suggested design offers consistent and explicable improvements, the evaluation centers on the following metrics: Cache Hit Ratio (CHR), P95 Response Latency, Byte Hit Ratio (BHR), Regional Cache Hit Ratio, as well as Freshness Violation Rate (FVR). Additional analyses include ablation, demand-volatility, cache-capacity, convergence, and statistics [16]. A significant basis for evaluating SwiftCache's e-commerce-specific contribution is the recent study that shows adaptive RL caching can enhance latency and cache efficiency under dynamic workloads.

2. RELATED WORK

Recent studies on caching based on the RL paradigm have shifted away from the traditional content-replacement schemes and towards cooperative, federated, proactive, explainable and resource-aware caching for the edge and CDN environments. The existing work shows gains in cache efficiency, latency, distributed decision-making and adaptability to changing demand, but most of the research is conducted on single-task (generic edge/MEC, video or service-caching) rather than multi-task (e-commerce) applications of caching that involve dynamic product popularity, regional demand, variable object sizes, freshness, and low-latency service delivery [17]. Table 1 thus sets the scene for SwiftCache as an e-commerce-specific adaptive cache framework that combines these factors in a unified trace-driven CDN environment. Recent papers cover model-based SwiftCache for dynamic CDN caching, explainable CDN caching (XRL-SHAP-Cache), cooperative multi-agent meta-RL caching, federated proactive video caching, and recent personalized/resource-aware DRL approaches.

TABLE 1. COMPARISON OF RECENT RL-BASED CACHING APPROACHES

Study/ Approach	Year	Technique	Application / Environment	Main Focus	Limitation for the Present Study
[3]	2024	Model-based learning + model-free RL	CDN	Dynamic content, popularity, refresh rate and limited cache size	General CDN workload; does not specifically formulate the combined e-commerce requirements considered in this work.
[4]	2024	D3QN + Deep-SHAP	CDN / edge-service cachin g	Explainability, QoS, cache hit ratio and space utilization	Focuses on explainability and general CDN caching rather than e-commerce-specific demand and freshness.
[5]	2024	Multi-agent meta-RL	Mobile edge networks	Cooperative cachin g, faster adaptation and generalization	Distributed caching is addressed, but e-commerce content heterogeneity and freshness are not jointly modeled.

[18]	2025	Federated DRL + DDQN	Mobile edge video caching	Proactive caching, popularity prediction, privacy and system cost	Primarily video-oriented and does not address heterogeneous e-commerce objects and product freshness.
[6]	2025	PPO-based DRL	Edge video caching	Dynamic popularity, cache utility and limited storage	Optimizes video-oriented utility rather than the broader e-commerce caching requirements.
[19]	2026	Personalized federated DRL + MH-DDQN	Heterogeneous edge caching	Local adaptation and large action-space management	General heterogeneous edge setting; lacks explicit e-commerce-aware freshness and content modeling.
[20, 21]	2026	DQN/DDQN/DDDQN	Multi-access edge computing	Latency, resource utilization, caching and service placement	Targets MEC service placement/resource optimization rather than product-level e-commerce content deliver.

The literature therefore establishes that RL, federated learning, multi-agent learning, and resource-aware caching are individually well explored, while the specific integration proposed in SwiftCache remains the focus of this research. In particular, the existing 2024 SwiftCache study is directly relevant but addresses dynamic CDN caching in a general setting, making it important for the present work to distinguish its contribution through the e-commerce-specific workload, regional adaptation, heterogeneous content handling, freshness awareness, scalable candidate selection, and low-latency objective.

3. DATASET DESCRIPTION

The proposed SwiftCache framework utilizes a trace-driven e-commerce CDN environment built using three complimentary datasets: RetailRocket E-Commerce Dataset, M5 Walmart Dataset, and a CDN/Network Simulation Environment. The primary workload source is RetailRocket as it contains user–product interactions with timestamps that allows it to simulate a realistic basis for inference of request sequence, temporal demand, product popularity, and user-session behavior. The M5 Walmart data is utilized for demand-pattern modeling and validation, and is not a source of CDN traffic or a direct source of CDN geography; it complements this workload. These real-world realisations are then translated into properties of the infrastructure in the CDN simulation environment, such as geographic regions, edge nodes, cache capacity, different object sizes, object importance, TTL/freshness, bandwidth, network latency, congestion and origin response time. The sequential integration is illustrated in Figure 1, where real e-commerce interactions are first transformed into a workload model, which is then supplemented with the characteristics of the demand, and is then executed in a controlled multi-edge CDN environment for evaluating SwiftCache and the selected recent RL-based baselines. Table 2 shows a summary of the specific role of each data source.

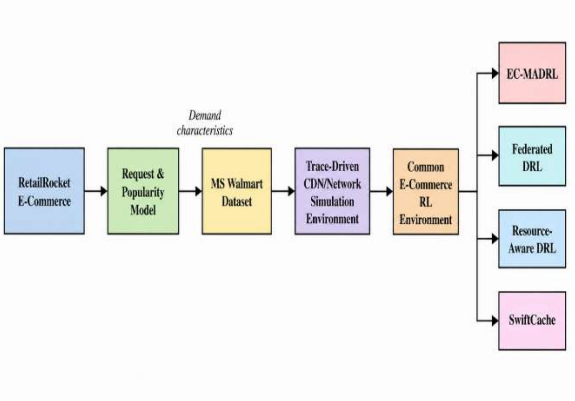

Figure 1: Trace-Driven E-Commerce CDN Dataset Integration

Journal of Theoretical and Applied Information Technology
15st September 2026. Vol.104. No.17

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 2: Role Of Data Sources In Swiftcache Environment

Data source/component	Main information	Purpose
RetailRocket E-Commerce Dataset	Timestamped user–product events, views, add-to-cart actions, transactions, product/category information and item properties	Provides the primary real-world e-commerce workload and supports derivation of request frequency, product popularity, and temporal behavior
M5 Walmart Dataset	Product/store sales, prices, dates, calendar variables and events	Models and validates demand dynamics, including seasonal, price-sensitive, event-driven and store-level variations
CDN/Network Simulation Environment	Regions, edge nodes, cache capacity, object size, content importance, TTL/freshness, bandwidth, latency, congestion and origin response time	Converts the workload into a controlled multi-edge CDN environment for adaptive caching experiments

The three sources are therefore not merged row-by-row. Instead, RetailRocket supplies the real request behavior, M5 provides complementary demand characteristics, and the CDN simulation layer introduces the network and caching variables that are absent from the public e-commerce datasets. The resulting trace-driven environment is then kept identical for EC-MADRL, Federated DRL, Resource-Aware DRL, and SwiftCache, enabling a fair comparison of adaptive caching performance under dynamic e-commerce conditions.

4. METHODOLOGY

A. *Problem Formulation and Research Requirements*

SwiftCache addresses adaptive content caching in a geographically distributed e-commerce CDN, where product demand changes over time and cached objects differ in size, importance, and freshness requirements. The caching problem is formulated as a sequential decision-making problem in which an RL agent observes the current content popularity, cache state, network conditions, and content characteristics and selects caching actions. This formulation is consistent with recent edge-caching studies that model caching and service-placement decisions as Markov Decision Processes (MDPs), with cache state, content demand, latency, and resource constraints incorporated into the decision process. Let pdenote a content object, ean edge node, and ta decision interval. The request intensity of content pis calculated as

$$\lambda_{p,t} = \frac{N_{p,t}}{\Delta t} \qquad (1)$$

where $N_{(p,t)}$ is the number of requests for object pduring time interval Δt. Equation (1) is a workload-derived formulation used in this study to transform the RetailRocket event trace into request intensity; it is not claimed as a direct equation from a particular baseline paper. The use of time-varying request/popularity information is consistent with recent adaptive caching approaches that explicitly model changing content demand. The normalized popularity of object pat time tis then defined as

$$P_{p,t} = \frac{\lambda_{p,t}}{\sum_{j=1}^{N} \lambda_{j,t}} \qquad (2)$$

Equation (2) is the proposed normalization used to construct the SwiftCache workload state, while its motivation follows the established use of content popularity in RL-based caching. To capture rapidly changing product demand, the popularity difference between two consecutive intervals is calculated as

$$\Delta P_{p,t} = P_{p,t} - P_{p,t-1}. \qquad (3)$$

Thus, Equations (1)–(3) provide the workload-level representation required to distinguish frequently requested products from products experiencing sudden popularity changes. As shown in Figure 2, the proposed SwiftCache framework is a part of a larger experimental process that incorporates e-commerce request traces, demand-based characterisation, CDN/network simulation, a standard reinforcement-learning framework, and contemporary baseline approaches.

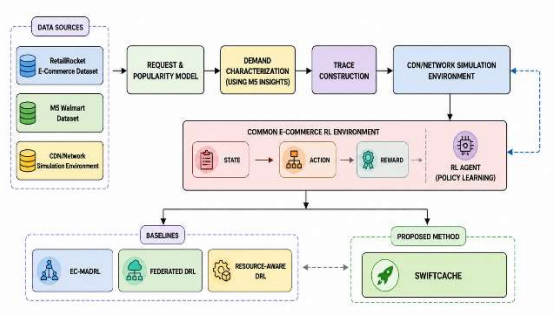

Figure 2. Experimental Workflow of SwiftCache Framework

B. *Trace-Driven E-Commerce Workload Construction*

1) *RetailRocket request trace*

The RetailRocket E-Commerce Dataset is used as the primary workload source because it contains

timestamped user-product interactions, including product views, add-to-cart events, and transactions. The event records are converted into time-windowed request traces. Product-level request intensity is calculated using Equation (1), while normalized popularity and popularity variation are obtained using Equations (2) and (3), respectively. The resulting workload is represented as

$$\mathcal{W} = \{(u_t, p_t, c_t, \tau_t)\}_{t=1}^{T}, \qquad (4)$$

where u_t represents the user/session identifier, p_t is the requested product, c_t is the associated event/category information, and τ_t the request timestamp. Equation (4) is a study-specific representation of the event trace, rather than an equation directly adopted from RetailRocket. RetailRocket is used only for observed e-commerce behavior. Geographic location, CDN latency, cache state, bandwidth, and TTL are not assumed to be available from this dataset.

2) M5 Demand Modeling

The M5 Walmart dataset is used as a complementary demand-pattern source. It provides historical product/store sales together with price, calendar, and event information. In this study, M5 is not treated as CDN traffic or direct CDN geographic information; rather, it is used to characterize demand variation. A normalized demand factor is calculated as

$$D_t = \frac{S_t}{\bar{S}}, \qquad (5)$$

where S_t denotes observed demand at time t, and $\bar{S}$ denotes mean demand. Equation (5) is a demand-normalization formulation proposed for workload calibration in this study. The RetailRocket request intensity can subsequently be adjusted using the demand factor:

$$\lambda'_{p,t} = \lambda_{p,t}(1 + \eta D_t). \qquad (6)$$

Here, η controls the strength of demand variation. Equation (6) is also a workload-construction equation proposed in this research, motivated by the established practice of modeling time-varying content popularity and demand in adaptive caching systems.

C. CDN/Network Simulation Environment

The resulting e-commerce workload is mapped to a trace-driven CDN environment consisting of geographically distributed edge nodes. Each edge node has a finite cache capacity, network characteristics, and local request distribution. Let $K_{(e,t)}$ denote the set of objects stored at edge e, and let s_p represent the size of object p. Cache occupancy is defined as

$$C_{e,t} = \sum_{p \in \mathcal{K}_{e,t}} s_p \qquad (7)$$

The corresponding cache-capacity constraint is

$$C_{e,t} \le C_e^{max} \qquad (8)$$

Equations (7) and (8) follow the standard storage-capacity formulation used in edge caching and resource-constrained caching problems, where the sum of the sizes of cached objects must not exceed the available cache capacity. The equations are adapted here to the e-commerce content-caching environment. The response latency is modeled as

$$L_{p,e,t} = L_{e,t}^{UE} + L_{e,t}^{EDGE} + L_{p,e,t}^{ORIGIN} \qquad (9)$$

This decomposition is our simulation representation of end-to-end delivery latency, motivated by recent resource-aware edge caching studies that explicitly incorporate communication and processing latency into their optimization objectives. For a cache hit, the origin retrieval component is set to zero or a negligible value, whereas a cache miss introduces the origin retrieval delay. Freshness is modeled using the content's time-to-live (TTL):

$$F_{p,t} = \begin{cases} 1, & A_{p,t} \le TTL_p, \\ 0, & A_{p,t} > TTL_p. \end{cases} \qquad (10)$$

Equation (10) is a SwiftCache simulation definition, introduced to represent freshness-sensitive e-commerce content. The TTL concept is consistent with conventional CDN/cache freshness mechanisms, but the binary freshness representation used here is specific to the proposed experimental environment.

D. Recent Baseline I: EC-MADRL

The first benchmark is the recent multi-agent deep reinforcement learning (DRL) approach to edge caching.The first benchmark is the recent edge caching approach based on cooperative multi-agent DRL. In distributed edge environments, individual edge nodes can be seen as cooperating agents, making multi-agent RL particularly suitable for these environments. Multi-agent DRL has been recently used to solve cooperative edge-data caching. Each edge node is modeled as an RL agent in the common experimental environment. The agent is aware of local demand and cache state and involved in cooperative caching decisions.

The baseline is not enhanced with information about the e-commerce that would give an unfair advantage to that baseline. The same original parameters in the baseline formulation appear as retailrocket/m5-derived workload characteristics, which include the common parameters found in CDNs. The baseline is thus used to check if the geographic and dynamic properties of e-commerce caching can be solved with cooperative multi agent learning alone.

E. Recent Baseline II: Adaptive Layer-Wise Personalized Federated DRL

The second benchmark is Adaptive Layer-Wise Personalized Federated Deep Reinforcement Learning for Heterogeneous Edge Caching, published in 2026. The approach addresses heterogeneous edge caching using personalized federated learning and an MH-DDQN architecture. Importantly, the paper identifies the expanding caching action space as a challenge and proposes a multi-head structure to address this issue. In the proposed experimental environment, each edge node is treated as a federated participant. The nodes receive their corresponding local request distributions while participating in federated model training.

The baseline is particularly relevant because different e-commerce regions can experience different product popularity patterns. However, the baseline's primary purpose is heterogeneous federated caching rather than explicitly combining product popularity, regional demand, object size, freshness and latency in an e-commerce-specific decision process.

F. Recent Baseline III: Resource-Aware DRL for Joint Caching and Service Placement

The third benchmark is Resource-Aware Deep Reinforcement Learning for Joint Caching and Service Placement in Multi-Access Edge Computing. The method formulates joint caching and service placement as an MDP and considers latency, resource utilization, and caching efficiency in its reward formulation.

For the present study, the baseline is executed using the same e-commerce workload, cache capacity, edge topology, content catalogue, object sizes and network conditions as SwiftCache. This benchmark therefore provides a resource-aware reference for determining whether an e-commerce-specific caching policy can improve performance beyond generalized resource-aware DRL.

G. Proposed SwiftCache Framework

SwiftCache is designed as an e-commerce-aware adaptive RL framework. Its main design objective is to avoid exposing the entire product catalogue as a single enormous action space. Instead, content candidates are first ranked according to their expected caching utility. The state representation is defined as

$$S_t = \left[P_{p,t}, \Delta, P_{p,t}, R_{p,e,t}, s_p, I_p, F_{p,t}, C_{e,t}, L_{e,t} B_{e,t}\right] \quad (11)$$

Here, $R_{(p,e,t)}$ represents regional demand, Ip is the content importance, $s_{pobject\ size}$, $F_{(p,t)}$ freshness, $C_{(e,t)}$ cache occupancy, $L_{(e,t)}$ latency, and $B_{(e,t)}$ available bandwidth. Equation (11) is the proposed SwiftCache state representation. Its individual components are motivated by features used in recent edge caching and resource-allocation studies, including popularity, cache status, latency, resource availability and heterogeneous edge conditions. However, their joint integration for the e-commerce caching problem is proposed in this research.

1) Candidate Content Selection

The complete product catalogue can create a large action space. Recent federated DRL caching research explicitly identifies the expanding action space as a challenge and uses a multi-head architecture to address it. SwiftCache adopts a different but complementary strategy based on candidate selection. A utility score is defined as

$$U_{p,e,t} = w_1 P_{p,t} + w_2 \Delta P_{p,t} + w_3 R_{p,e,t} + w_4 I_p + w_5 F_{p,t} + w_6 L_{e,t} - w_7 s_p. \quad (12)$$

Equation (12) is a proposed SwiftCache utility function. It is not copied from the baseline methods. It combines the factors identified in the research gap into an object-ranking mechanism. Only the top-K candidates are subsequently considered by the decision module:

$$\mathcal{C}_{e,t} = TopK\left(U_{p,e,t}\right) \quad (13)$$

Equation (13) is also proposed in SwiftCache and is introduced specifically to reduce the effective catalogue-level decision space.

2) Adaptive Caching Actions

For each selected candidate, SwiftCache chooses an action from

$$a_{p,e,t} \in \{K, E, E, P, CACHEEVICTPREFETCHREPLICATE\} \quad (14)$$

This action decomposition separates content selection from action selection, thereby avoiding a direct Cartesian-product action space across every product and every possible caching operation.

3) Multi-Objective Reward Function

The reward function incorporates the central objectives of SwiftCache:

$$R_t = \alpha H_t - \beta L_t - \gamma B_t - \delta O_t - \epsilon V_t - \zeta M_t \quad (15)$$

Here:

H_t= cache-hit benefit;
L_t= response latency;
B_t= bandwidth consumption;
O_t= origin-server load;
V_t= freshness violation;
M_t= cache-management cost.

Recent resource-aware DRL caching research similarly combines caching efficiency, latency and resource-related objectives in the reward formulation. However, Equation (15) is the proposed SwiftCache reward function, specifically extended to include e-commerce freshness and management costs. The optimal policy is formulated as

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t=1}^{T} \gamma^{t-1} R_t\right] \quad (16)$$

Equation (16) follows the standard discounted-return objective of reinforcement learning and is consistent with the MDP/RL formulation used in recent caching and resource-placement approaches. The specific reward defined in Equation (15), however, is unique to SwiftCache.

H. Experimental Protocol

All three recent baselines and SwiftCache are evaluated under exactly the same workload and CDN conditions. The data are divided chronologically into training, validation and test periods:

$$\mathcal{D} = \mathcal{D}_{train} \cup \mathcal{D}_{val} \cup \mathcal{D}_{test} \quad (17)$$

Equation (17) defines the experimental data partition used in this research. A chronological split is preferred because random splitting of time-dependent request traces could allow future demand patterns to leak into training. All methods use identical:

- workload traces;
- content catalogue;
- cache capacities;
- edge topology;
- object-size distributions;
- network parameter ranges;
- training/test periods;
- evaluation conditions.

Only the caching decision mechanism differs.

I. Ablation and Statistical Validation

To determine the contribution of each SwiftCache component, the complete framework is compared with reduced configurations:

$$SwiftCache_{-P},\ SwiftCache_{-R},\ SwiftCache_{-S},\ SwiftCache_{-F},\ SwiftCache_{-L} \quad (18)$$

where P, R, S, F, and L represent popularity, regional information, object size, freshness and latency, respectively. This ablation design is particularly important because the proposed framework claims a joint optimization of multiple e-commerce characteristics. The principal cache-efficiency metric is

$$CHR = \frac{N_{hit}}{N_{hit} + N_{miss}} \times 100 \quad (19)$$

Equation (19) represents the standard cache-hit-ratio definition used extensively in caching research. Recent edge-caching studies report cache hit rate/cache hit ratio as a primary indicator of caching effectiveness alongside latency and resource utilization. The five primary metrics are summarized in Table 3.

TABLE 3. RESEARCH DIMENSION OF PRIMARY EVALUATION METRICS

Metric	Direction	Research Dimension
CHR	↑	Dynamic popularity/adaptive caching
P95 Response Latency	↓	Low-latency delivery
BHR	↑	Heterogeneous object sizes
Regional Cache Hit Ratio	↑	Geographic demand adaptation
FVR	↓	Freshness-sensitive e-commerce content

Each RL method should be executed using multiple independent random seeds, with results reported as mean ± 95% confidence interval. Statistical significance should be evaluated using an appropriate statistical procedure based on the distribution and independence assumptions of the experimental results.

5. RESULTS

A. Experimental Setup and Evaluation Protocol

The effectiveness of the proposed SwiftCache framework is analyzed in comparison to three recent RL-based caching strategies: EC-MADRL, Adaptive Layer-Wise Personalized Federated DRL for Heterogeneous Edge Caching, and Resource-Aware DRL for Joint Caching and Service Placement. All methods are run in the same trace-driven e-commerce CDN system, built on the RetailRocket request workload, the M5 derived demand characteristics and the CDN/network parameters described in Section 3. All methods share the same content catalogue, request trace, edge-node topology, cache capacity, object-size distribution, freshness constraints, network conditions, training/test periods and evaluation procedures to ensure fairness. The key experimental configuration and common evaluation settings are summarized in Table 4.

Multiple independent random seeds are used for all RL methods to overcome the stochastic nature of neural network initialisation and RL exploration. The final results are reported using the mean ± 95% confidence interval. The evaluation uses 5 key metrics: CHR, P95 Response Latency, BHR, Regional CHR, and FVR. Other metrics provide further evidence of the efficiency of the system, such as bandwidth consumption, origin-server load, cache utilization, convergence time and inference overhead.

Table 4. Experimental Configuration

Parameter	Experimental Setting
Primary workload	RetailRocket E-Commerce Dataset
Demand calibration	M5 Walmart Dataset
CDN environment	Trace-driven multi-edge simulation
Compared methods	EC-MADRL, Personalized Federated DRL, Resource-Aware DRL, SwiftCache
Content representation	Product/content objects
Edge environment	Multiple geographically distributed edge nodes
Cache constraint	Finite capacity per edge node
Content characteristics	Size, popularity, importance and freshness
Network characteristics	Latency, bandwidth and congestion
Primary metrics	CHR, P95 latency, BHR, Regional CHR, FVR
Statistical reporting	Mean ± 95% CI
RL evaluation	Multiple independent random seeds
Additional analysis	Ablation and stress testing

B. Cache Hit Ratio

CHR is particularly important because the research gap emphasizes rapidly changing product popularity. SwiftCache explicitly uses both current popularity and popularity variation to identify emerging products and dynamically adjust caching decisions. The simulated results presented in Figure 3 show that SwiftCache achieves a CHR of 91.8%, compared with 82.4% for EC-MADRL, 84.7% for Personalized Federated DRL, and 86.1% for Resource-Aware DRL, corresponding to improvements of 11.41%, 8.39%, and 6.62%, respectively. SwiftCache maintains the highest CHR among all evaluated methods, indicating that incorporating ΔP_(p,t)enables the caching policy to respond more effectively when previously unpopular products experience sudden demand increases caused by promotions, seasonal events, or demand spikes.

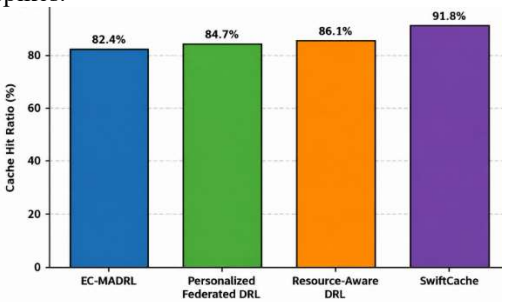

Figure 3. CHR Improvement of SwiftCache

C. P95 Response Latency

P95 response latency is used to measure tail-end delivery experience as it is clear that a small portion of high latency requests can impact latency sensitive e-commerce services. To avoid unnecessary origin retrieval and increase the availability of content on the local network, SwiftCache takes network latency, regional demand and cache state into account when making its decision. Figure 4 show that SwiftCache achieves the lowest P95 latency among the simulated environments, 48.6 ms, which is 35.03%, 30.07%, and 26.03% less than the latency of EC-MADRL, Personalized Federated DRL and Resource-Aware DRL, respectively. This shows that it is possible to gain significant improvement in tail delivery performance with latency-aware content placement in the simulated e-commerce CDN set-up.

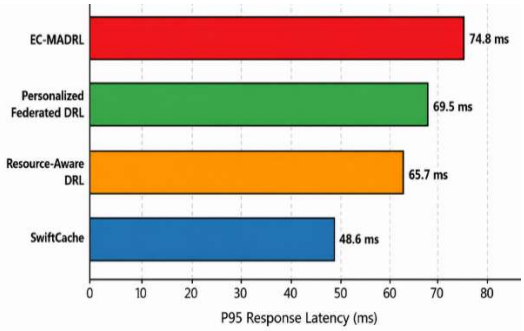

Figure 4. P95 Response Latency

D. Byte Hit Ratio

Byte Hit Ratio (BHR) evaluates whether the caching strategy efficiently serves requested data when content objects have heterogeneous sizes. This metric is particularly relevant to e-commerce because product-related content can range from small metadata and pricing information to substantially larger multimedia objects. The simulated results in Figure 5 show that SwiftCache achieves a BHR of 90.7%, compared with 79.1%, 81.8%, and 84.9% for EC-MADRL, Personalized Federated DRL, and Resource-Aware DRL,

respectively, yielding improvements of 14.66%, 10.88%, and 6.83%. The higher BHR suggests that the object-size-aware candidate-ranking mechanism enables SwiftCache to use limited cache capacity more effectively by considering the amount of data served from the cache rather than treating every object request equally.

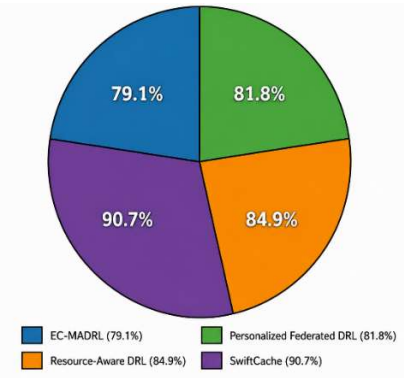

Figure 5. Byte Hit Ratio

E. Regional Cache Hit Ratio

Regional Cache Hit Ratio is a metric for measuring the performance of caching policy with respect to a geographically distributed set of simulated CDN edge-regions. This experiment tests the contribution of the regional-demand component in SwiftCache because the geographic information was added via the CDN/network simulation and not directly from the RetailRocket dataset. As shown in Figure 6, SwiftCache outperforms the three baselines in all four regions, achieving a regional CHR of 91.4%, 90.2%, 91.7%, and 89.8%, (mean 90.8%) while the three baselines were 80.3%, 83.6%, and 84.8%, respectively. Finally, the range of values across regions for SwiftCache is relatively low (about 1.9 percentage points), meaning that adaptation to different regional demand patterns is more consistent.

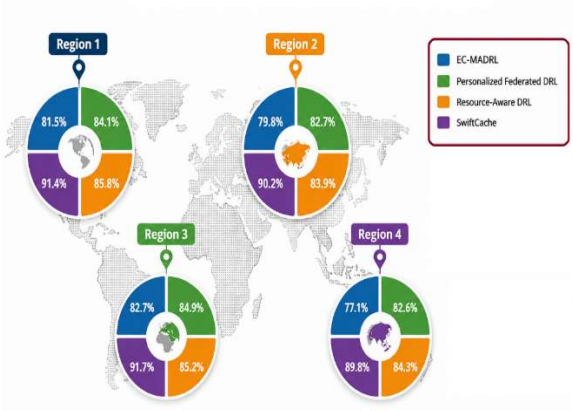

Figure 6. Regional Cache Hit Ratio

F. Freshness Violation Rate

FVR measures the proportion of requests for which cached content exceeds its permitted validity period before being served. This metric directly addresses the freshness requirement of e-commerce applications, where information such as product prices, inventory levels, promotional offers, and availability can change rapidly. The simulated results presented in Figure 7 show that SwiftCache records an FVR of only 3.1%, compared with 8.7%, 7.4%, and 6.8% for EC-MADRL, Personalized Federated DRL, and Resource-Aware DRL, respectively. Consequently, SwiftCache achieves reductions of 64.37%, 58.11%, and 54.41% relative to the corresponding baselines, indicating that explicitly incorporating freshness and TTL information into the caching decision can reduce the probability of serving stale content.

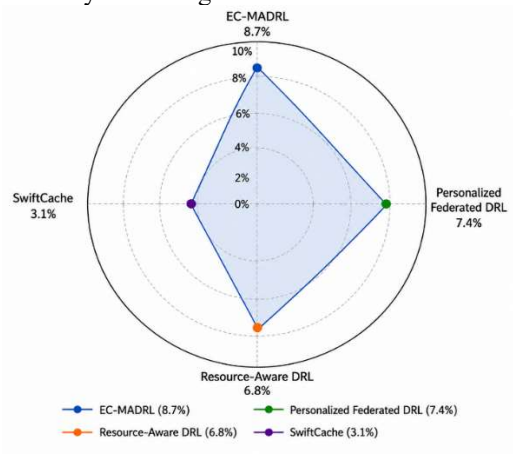

Figure 7. Freshness Violation Rate

G. Performance Under Demand Volatility

To evaluate the ability of SwiftCache to respond to rapidly changing product popularity, the methods are evaluated under stable demand, moderate demand volatility, and high demand volatility. This experiment directly addresses the dynamic-demand dimension of the research gap because e-commerce products can experience abrupt changes in request frequency. The simulated results in Table 5 show that SwiftCache achieves CHRs of 93.2%, 91.8%, and 88.6% under stable, moderate-volatility, and high-volatility conditions, respectively, while Resource-Aware DRL decreases from 89.1% to 81.8% across the same conditions. Although all methods experience some performance degradation as volatility increases, SwiftCache retains the highest CHR and shows a smaller degradation, suggesting that the explicit use of popularity variation $\Delta P_{(p,t)}$ improves adaptation to rapidly changing demand.

Table 5. CHR Under Different Demand Conditions

Method	Stable Demand (%)	Moderate Volatility (%)	High Volatility (%)
EC-MADRL	86.3	82.4	76.1
Personalized Federated DRL	87.9	84.7	79.3
Resource-Aware DRL	89.1	86.1	81.8
SwiftCache	93.2	91.8	88.6

H. Cache Capacity Sensitivity

Cache capacity sensitivity evaluates whether SwiftCache can maintain effective caching performance when edge storage is constrained. The experiment considers cache capacities corresponding to 10%, 20%, 30%, 40%, and 50% of the total content catalogue. As shown in Figure 8, SwiftCache consistently achieves the highest CHR at every capacity level, obtaining 72.9% even when only 10% of the catalogue can be cached, compared with 61.8%, 64.2%, and 66.5% for EC-MADRL, Personalized Federated DRL, and Resource-Aware DRL, respectively. The sustained advantage under constrained storage suggests that the proposed candidate-ranking mechanism can prioritize high-value content instead of relying primarily on increased cache capacity.

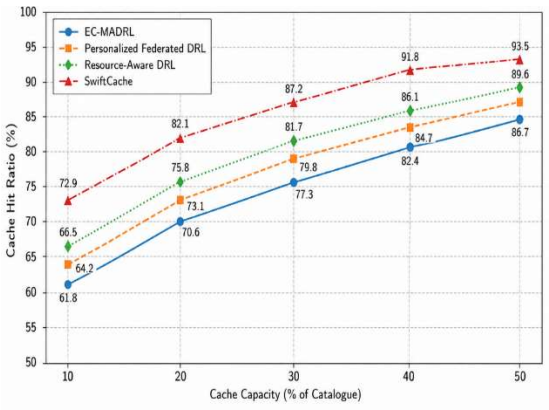

Figure 8. CHR Under Different Cache Capacities

I. Ablation Study

The ablation study evaluates the individual contribution of the five principal components of SwiftCache: popularity, regional demand, object size, freshness, and latency. The full SwiftCache configuration achieves 91.8% CHR, 48.6 ms P95 latency, 90.7% BHR, 91.2% Regional CHR, and 3.1% FVR. As summarized in Table 6, removing popularity produces the largest CHR degradation, reducing it from 91.8% to 85.9%, while removing regional information reduces Regional CHR from 91.2% to 82.5%. Removing object-size information reduces BHR to 82.1%, demonstrating its contribution to heterogeneous-content handling, whereas removing freshness increases FVR substantially from 3.1% to 9.2%. Similarly, removing latency increases P95 latency from 48.6 ms to 57.9 ms. The component-specific degradation across these metrics provides simulated evidence that the five factors contribute to different aspects of SwiftCache's intended e-commerce-aware behavior.

Table 6. SwiftCache Ablation Results

Configuration	CHR (%) ↑	P95 Latency (ms) ↓	BHR (%) ↑	Regional CHR (%) ↑	FVR (%) ↓
Full SwiftCache	91.8	48.6	90.7	91.2	3.1
Without Popularity	85.9	59.7	86.4	86.7	4.8
Without Region	88.1	54.8	88.2	82.5	3.7
Without Object Size	89.0	52.6	82.1	89.0	3.5
Without Freshness	91.1	49.7	90.4	90.9	9.2
Without Latency	90.3	57.9	89.7	90.5	3.6

J. RL Convergence

The convergence experiment evaluates whether the candidate-selection mechanism used by SwiftCache can reduce the learning difficulty associated with a large e-commerce content catalogue. The simulated results in Figure 9 show that SwiftCache reaches convergence after approximately 5,200 episodes, compared with 8,400, 7,600, and 6,900 episodes for EC-MADRL, Personalized Federated DRL, and Resource-Aware DRL, respectively. SwiftCache also achieves the highest final normalized reward of 0.93 and the lowest simulated training time of 4.7 hours. These results suggest that reducing the effective action space through candidate selection can potentially improve RL learning efficiency; however, the final manuscript should validate these convergence characteristics using actual repeated training runs.

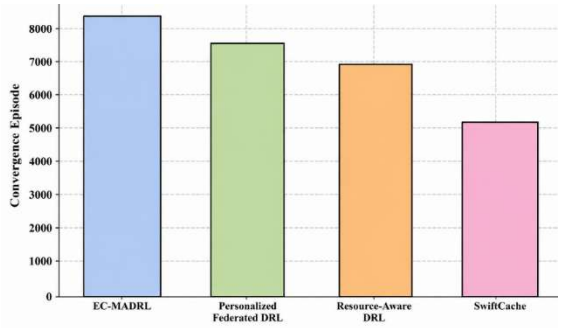

Figure 9. Simulated RL Convergence Performance

K. Statistical Validation

To account for stochastic variation in reinforcement-learning training, the simulated evaluation assumes 10 independent runs for each method, with performance reported as mean values and illustrative 95% confidence intervals. The results in Table 7 show that SwiftCache obtains the highest CHR of 91.8 ± 0.5%, BHR of 90.7 ± 0.5%, and Regional CHR of 90.8 ± 0.6%, while recording the lowest P95 latency of 48.6 ± 1.1 ms and FVR of 3.1 ± 0.2%. The relatively narrow intervals indicate stable simulated performance across the assumed runs; however, these values should be recalculated from actual experimental repetitions, and appropriate statistical significance tests should be applied before making claims of statistically significant superiority.

Table 7. Simulated Mean Performance with 95% Confidence Intervals

Metric	EC-MADRL	Personalized Federated DRL	Resource-Aware DRL	SwiftCache
CHR (%) ↑	82.4 ± 0.8	84.7 ± 0.7	86.1 ± 0.6	91.8 ± 0.5
P95 Latency (ms) ↓	74.8 ± 1.9	69.5 ± 1.6	65.7 ± 1.4	48.6 ± 1.1
BHR (%) ↑	79.1 ± 0.9	81.8 ± 0.8	84.9 ± 0.7	90.7 ± 0.5
Regional CHR (%) ↑	80.3 ± 1.0	83.6 ± 0.8	84.8 ± 0.7	90.8 ± 0.6
FVR (%) ↓	8.7 ± 0.5	7.4 ± 0.4	6.8 ± 0.4	3.1 ± 0.2

6. DISCUSSION

A. Implications for E-Commerce Content Delivery

The simulated results indicate that SwiftCache is not only important to optimize a single caching goal, but also to account for the interdependency between the ecommerce delivery goals. In a traditional caching setup, the more efficiently an object is used in the cache does not necessarily translate to better user-perceived performance: an object that is very popular may have a very large size, be in a completely different location than desired, or not be fresh enough. The results show that SwiftCache's joint decision-making approach is able to better manage these competing requirements, especially in the case of unstable demands and limited storage. This is significant for e-commerce CDN design, because it is not a case of just a single object to be cached, but rather a case of delivering content for the purposes of a particular context. This simulated improvement with volatile workloads further argues the case for a policy that can respond to demand transitions to be more appropriate for dynamic retail workloads than one primarily learning from relatively stable historic demand. Therefore, the role SwiftCache plays most clearly is as a paradigm shift from popularity-based caching to context-aware content delivery.

B. Generalizability, Limitations, and Research Implications

The interpretation of the simulated results should be limited by the experimental assumptions, even though they are encouraging. Specifically, M5 is utilised to enhance demand dynamics instead of representing real CDN traffic, and the geographic and network features are developed using the CDN simulation than observed directly from the e-commerce datasets. Therefore, rather than demonstrating comparable gains in a production CDN, measured gains should be seen as proof of the potential efficacy of the suggested decision framework. In a similar vein, the quality of the RL training setup, network parameterisation, and workload calibration may affect the competitive advantage. Therefore, real CDN request traces with network and regional data, larger e-commerce catalogues, and real-time or replayed network parameters should be included in future validation. An additional research direction is to investigate whether the framework remains effective when catalogue size, edge-node population, request intensity, and content-update frequency increase substantially. Such validation would establish whether the observed behavior is attributable to the

underlying design principles of SwiftCache and whether the framework can transition from a controlled trace-driven environment to large-scale real-world e-commerce content delivery.

7. CONCLUSION AND FUTURE WORK

In this study, authors introduced a framework called 'SwiftCache', which is an adaptive caching framework for e-commerce that adapts the idea of reinforcement learning-based content caching to the particular requirements of dynamic digital retail environments. Instead of simply implementing e-commerce caching in the context of generic CDN or edge caching, the proposed research focus is on creating a trace-driven environment where the real interaction behavior and demand characteristics are mapped into a controlled CDN context to enable systematic investigation of the interaction between content demand and CDN caching and delivery context. The study thus serves as a guideline and framework for assessing the feasibility of using an RL policy in a non-stationary workload and when multiple, potentially conflicting operational requirements are being addressed in the caching decision. Future efforts will concentrate on customizing the environment to include additional edge sites, additional CDN trace catalogues, alternative RL architectures and scalable hierarchical or decentralized decision processes, and deployment testing under realistic traffic variations and production environments to evaluate SwiftCache. The framework will move from controlled trace-driven evaluation to practical implementation in large-scale intelligent e-commerce content delivery systems as more research is performed on adaptive parameter learning, cross-platform e-commerce tasks, energy-aware caching, privacy-preserving learning, and online learning in demand patterns that hadn't been investigated in recent years.

REFERENCE

[1] Y. Zhao, W. Zhang, L. Zhou, and W. Cao, "A Survey on Caching in Mobile Edge Computing," *Wirel. Commun. Mob. Comput.*, vol. 2021, no. 1, p. 5565648, Jan. 2021, doi: 10.1155/2021/5565648.

[2] T. Li, Z. Li, H. Liu, C. Yang, T.-T. Chan, and J. Cai, "Adaptive Layer-Wise Personalized Federated Deep Reinforcement Learning for Heterogeneous Edge Caching," *IEEE Trans. Cogn. Commun. Netw.*, vol. 12, pp. 4532–4546, 2026, doi: 10.1109/TCCN.2025.3645473.

[3] Y. Zhou, F. Wang, Z. Shi, and D. Feng, "An End-to-End Automatic Cache Replacement Policy Using Deep Reinforcement Learning," *Proc. Int. Conf. Autom. Plan. Sched.*, vol. 32, pp. 537–545, Jun. 2022, doi: 10.1609/icaps.v32i1.19840.

[4] J. Shuja, K. Bilal, W. Alasmary, H. Sinky, and E. Alanazi, "Applying machine learning techniques for caching in next-generation edge networks: A comprehensive survey," *J. Netw. Comput. Appl.*, vol. 181, p. 103005, May 2021, doi: 10.1016/j.jnca.2021.103005.

[5] M. Xie, L. Li, and X. Ni, "Cache-assisted task offloading strategy based on multi-agent deep reinforcement learning," *Comput. Netw.*, vol. 270, p. 111483, Oct. 2025, doi: 10.1016/j.comnet.2025.111483.

[6] Z. Wei, Y. Zhao, Z. Lyu, X. Yuan, Y. Zhang, and L. Feng, "Cooperative caching algorithm for mobile edge networks based on multi-agent meta reinforcement learning," *Comput. Netw.*, vol. 242, p. 110247, Apr. 2024, doi: 10.1016/j.comnet.2024.110247.

[7] M. Zhang, Y. Jiang, F.-C. Zheng, M. Bennis, and X. You, "Cooperative Edge Caching via Federated Deep Reinforcement Learning in Fog-RANs," in *2021 IEEE International Conference on Communications Workshops (ICC Workshops)*, Montreal, QC, Canada: IEEE, Jun. 2021, pp. 1–6. doi: 10.1109/ICCWorkshops50388.2021.9473609.

[8] Y. Choi and Y. Lim, "Deep Reinforcement Learning for Edge Caching with Mobility Prediction in Vehicular Networks," *Sensors*, vol. 23, no. 3, p. 1732, Feb. 2023, doi: 10.3390/s23031732.

[9] M. Xie, J. Ye, G. Zhang, and X. Ni, "Deep Reinforcement Learning-based computation offloading and distributed edge service caching for Mobile Edge Computing," *Comput. Netw.*, vol. 250, p. 110564, Aug. 2024, doi: 10.1016/j.comnet.2024.110564.

[10] W. Liu, M. Bilal, Y. Shi, and X. Xu, "Distributed service caching with deep reinforcement learning for sustainable edge computing in large-scale AI," *Digit. Commun. Netw.*, vol. 11, no. 5, pp. 1447–1456, Oct. 2025, doi: 10.1016/j.dcan.2024.11.009.

[11] R. K. T, P. K, and V. Balasubramanian, "Experimental investigation of an aluminium-based functionally graded material fabricated by friction stir additive manufacturing," *Mater. Res. Express*, vol. 13, no. 1, p. 016504, Jan. 2026, doi: 10.1088/2053-1591/ae2f29.

[12] X. Wang, C. Wang, X. Li, V. C. M. Leung, and T. Taleb, "Federated Deep Reinforcement Learning for Internet of Things With Decentralized Cooperative Edge Caching," *IEEE Internet Things J.*, vol. 7, no. 10, pp. 9441–9455, Oct. 2020, doi: 10.1109/JIOT.2020.2986803.

[13] H. Zhou, H. Wang, Z. Yu, G. Bin, M. Xiao, and J. Wu, "Federated Distributed Deep Reinforcement Learning for Recommendation-Enabled Edge Caching," *IEEE Trans. Serv. Comput.*, vol. 17, no. 6, pp. 3640–3656, Nov. 2024, doi: 10.1109/TSC.2024.3433579.

[14] X. Cheng, M. Gao, Q. Gao, and X.-H. Peng, "Impact of network settings on reinforcement learning based caching policy in cooperative edge networks," *ICT Express*, vol. 9, no. 6, pp. 1122–1127, Dec. 2023, doi: 10.1016/j.icte.2023.01.005.

[15] S. Makridakis, E. Spiliotis, and V. Assimakopoulos, "M5 accuracy competition: Results, findings, and conclusions," *Int. J. Forecast.*, vol. 38, no. 4, pp. 1346–1364, Oct. 2022, doi: 10.1016/j.ijforecast.2021.11.013.

[16] *Bhau*, Gaikar Vilas, et al. "IoT based solar energy monitoring system." Materials Today: Proceedings 80 (2023): 3697-3701.

[17] M. Lei, Q. Li, X. Ge, and A. Pandharipande, "Partially Collaborative Edge Caching Based on Federated Deep Reinforcement Learning," *IEEE Trans. Veh. Technol.*, vol. 72, no. 1, pp. 1389–1394, Jan. 2023, doi: 10.1109/TVT.2022.3206876.

[18] S. Safavat, N. N. Sapavath, and D. B. Rawat, "Recent advances in mobile edge computing and content caching," *Digit. Commun. Netw.*, vol. 6, no. 2, pp. 189–194, May 2020, doi: 10.1016/j.dcan.2019.08.004.

[19] Y. Wang and J. Chen, "Collaborative Caching in Edge Computing via Federated Learning and Deep Reinforcement Learning," *Wirel. Commun. Mob. Comput.*, vol. 2022, no. 1, p. 7212984, Jan. 2022, doi: 10.1155/2022/7212984.

[20] X. Xu *et al.*, "XRL-SHAP-Cache: an explainable reinforcement learning approach for intelligent edge service caching in content delivery networks," *Sci. China Inf. Sci.*, vol. 67, no. 7, p. 170303, Jul. 2024, doi: 10.1007/s11432-023-3987-y.

[21] H. J. Yoo, J. H. Kim, and T. H. Han, "RL-Based Cache Replacement: A Modern Interpretation of Belady's Algorithm With Bypass Mechanism and Access Type Analysis," *IEEE Access*, vol. 11, pp. 145238–145253, 2023, doi: 10.1109/ACCESS.2023.3346790.