

(REVIEW ARTICLE)



STATE-AWARE PREFETCHING AND CACHE POLICIES FOR LATENCY-OPTIMIZED WEB EXPERIENCES

Akshatha Madapura Anantharamu *

San Jose State University, San Jose, CA.

* Corresponding Author

ORCID Details

Akshatha Madapura Anantharamu: <https://orcid.org/0009-0005-3147-7076>

World Journal of Advanced Engineering Technology and Sciences, 2026, 20(03), 119–134

Publication history: Received on 23 May 2026; revised on 16 September 2026; accepted on 18 September 2026

Article DOI: <https://doi.org/10.30574/wjaets.2026.20.3.0346>

Copyright © 2026 Author(s) retain the copyright of this article. This article is published under the terms of the Creative Commons Attribution License 4.0

ABSTRACT

Modern web applications demand sustained low latency under workloads that shift across users, devices, sessions, and network conditions. Classical cache replacement policies such as Least Recently Used (LRU) and Least Frequently Used (LFU) treat every cached object identically and ignore the cost-of-miss heterogeneity that drives user-perceived latency in session-driven web workloads. This paper presents a state-aware prefetching and adaptive cache framework whose HybridScore eviction policy combines recency, frequency, latency-sensitivity, prediction confidence, bandwidth cost, and cache occupancy in a single decomposed scoring function, with a variable-order Markov predictor supplying the prediction term and a ghost-list mechanism adapting the per-signal weights online. The framework is evaluated against LRU, LFU, Adaptive Replacement Cache (ARC), and S3-FIFO across 350 trace-driven simulations spanning two workloads (175 per workload), seven cache sizes, and five random seeds. HybridScore achieves the lowest P95 and P99 latency in every cell of the experimental matrix, with non-overlapping 95% confidence intervals against the strongest baselines (ARC and LFU, both 132.3ms at 2MB). Hit-rate improvements are workload-dependent: HybridScore exceeds every baseline on the session workload and matches the strongest baselines on Zipf. Inspection of the adapted weights reveals which scoring signal was most under-weighted at initialization, an interpretability result that single-parameter self-tuning policies cannot produce.

Keywords: State-Aware Prefetching, Adaptive Cache Policy, Web Latency Optimization, Edge Caching, Predictive Web Systems, Quality of Experience (QoE)

1 INTRODUCTION

In today's dynamic Web apps, speed and the delivery of targeted content to consumers are critical to the Web experience. In modern applications such as e-commerce, streaming services, social applications, and cloud-based enterprise applications, Content Delivery Systems are used to deliver content to users as needed and to provide Quality of Experience (QoE). The problem of network communication delays, server processing, and resource rendering, as well as cache misses, remains a major shortcoming in distributed Web systems. Caching and prefetching are important optimization techniques used by browsers, Content Delivery Networks (CDNs), edge servers, and application layers to mitigate performance issues caused by network latency. A variety of cache replacement policies have been developed

to reduce retrieval cost and maximize content availability, such as Least Recently Used (LRU), Least Frequently Used (LFU), and First-In-First-Out (FIFO) [1]. Prefetching strategies predict future user requests and proactively provide the required resources before they are explicitly requested [2]. As web workloads grow more complex, with the number of devices used and the constantly changing nature of the network, static caching is no longer effective. With the development of edge computing, artificial intelligence, and predictive analytics, intelligent cache management systems can now adapt at runtime [3]. State-aware systems rely on context (such as users' navigation patterns, session properties, device properties, bandwidth information, and cache occupancy) to predict and retrieve resources more efficiently [4].

While solutions to some caching and prefetching problems have been found, existing web optimization systems still suffer from performance and scalability issues. There are variants of the conventional caching system, most of which rely on fixed replacement algorithms that cannot consider the user behavior and uses, nor the application environment at runtime [5]. The dynamic nature of workloads leads to incorrect cache admission and eviction decisions due to misinformed workload information, resulting in higher cache misses and worse latency. Many web prefetching approaches are based on historical request patterns and stochastic models; however, these methods often struggle to adapt effectively to rapid changes in user behavior and dynamic network conditions [6]. Moreover, the current solutions lack a common framework for caching and prefetching operations, and they introduce latency sensitivity to these operations to enhance performance. The edge network is distributed, and so are mobile devices and cloud-native applications, which produce unpredictable traffic patterns and system resource requirements, as outlined here in source [7]. According to [8], a small delay can significantly reduce user satisfaction and business outcomes, making it a critical challenge for large-scale systems to meet P95 and P99 response time targets. More recently, new intelligent caching methods based on machine learning and reinforcement learning have been proposed, but they require substantial computational power and large training sets and lack explainability and scalability in operation [9][10]. A lightweight, adaptive, state-aware cache management system should be created for the research community to optimize prefetching and caching in the active web environment, with minimal bandwidth usage and low-latency operations.

The research introduces a framework that combines state-based prefetching with adaptive cache management to deliver web experiences with minimal latency. The main research question is how, and whether, prefetching predictions and the cache eviction process can be improved using real-time contextual information to improve prediction accuracy and cache operations, and, in turn, user response time. The framework proposes a hybrid decision-making mechanism that combines recent usage patterns and current usage patterns, along with predicted future value and latency sensitivity, to manage cached resources and the direction of prefetching activities. This system leverages user navigation states, device conditions, network variability, and cache occupancy metrics to improve cache hit rates, reduce unnecessary prefetch traffic, and enhance overall QoE. This review examines experimental studies on adaptive caching frameworks using simulated workloads and distributed web environments, and compares them against traditional caching approaches such as LRU, LFU, and static prefetching methods. The paper further explores the significance of adaptive caching in edge computing systems, emphasizing its role in reducing tail latency and improving the overall performance and responsiveness of contemporary web applications.

This paper includes the following objectives:

- To be able to describe the drawbacks of systematic caching and prefetching in a dynamic Web environment.
- Design and implement a state-aware predictive prefetching system with the runtime parameters of the context.
- Design an adaptive cache replacement strategy for making latency-sensitive decisions.
- The proposed framework is tested for its performance to measure cache hit ratio, response time, and tail latency.
- Intelligent caching techniques for edge computing and the future version of Web architecture.

2 RELATED WORK

Web caching, pre-fetch, and latency optimization are two areas of research that have been important for the last 20 years and have made significant strides in distributed web systems and the increasing need for efficient digital services. The first research studies investigated static cache replacement policies such as LRU, LFU, and Adaptive Replacement Cache (ARC) to achieve higher cache hit ratios and fewer server operations [11][12]. When traffic was steady, the methods were effective, but they did not handle sharp fluctuations in Web traffic. Subsequent research extended predictive web prefetching models to incorporate hyperlinks and navigation patterns of user sessions, as well as probabilistic access patterns, to reduce perceived latency and improve page response times [2]. Since the advent of cloud computing and CDNs, researchers have begun to study distributed caching systems that cache content at the network edge and support multiple locations [13]. Recent advancements in edge computing and artificial intelligence have introduced innovative approaches to cache optimization by integrating machine learning, reinforcement learning,

and context-aware decision-making techniques into caching and prefetching systems. These developments have significantly improved the adaptability, efficiency, and predictive capabilities of modern cache management frameworks [9]. They found that runtime context information, such as user behavior, bandwidth conditions, and workload dynamics, can be leveraged to improve cache efficiency and prediction accuracy [4][14]. If the delay is occasionally high and the system is large-scale, it will have significant QoE issues, and the research community is now focusing on tail latency optimization. Despite the availability of caching and prefetching solutions, existing implementations are still integrated separately, leading to higher cache pollution, computational overhead, and underutilized bandwidth [15]. Recent studies have shown that modern Web architectures with low-latency requirements require integration with state-aware frameworks that optimize caching and prefetching.

Table 1: Summary of Existing Research Work

Ref. No.	Technique/Model	Key Contribution	Limitations
[12]	Adaptive Replacement Cache (ARC)	Introduced self-tuning cache replacement using recency and frequency balance	Limited contextual awareness and runtime adaptability
[2]	Predictive Web Prefetching	Reduced web latency through predictive hyperlink prefetching	High prediction error and cache pollution
[7]	Edge Computing Framework	Defined edge computing architecture for low-latency applications	Did not address adaptive cache optimization
[17]	Machine Learning Scheduling	Applied learning-based optimization in distributed systems	High computational complexity
[4]	Context-Aware Caching	Integrated contextual runtime information into caching decisions	Limited scalability evaluation
[15]	Tail Latency-Aware Caching	Optimized P95/P99 latency using intelligent cache prioritization	Focused mainly on backend cache systems
[9]	ML-Based Cache Management	Used machine learning for adaptive cache replacement	Requires extensive training datasets
[16]	Edge Media Caching	Improved distributed content delivery efficiency	Limited predictive prefetch integration
[1]	Web Cache Fundamentals	Comprehensive analysis of caching fundamentals	Traditional methods unsuitable for dynamic workloads
[3]	Edge Caching Survey	Surveyed modern edge-cache optimization strategies	Lacked a unified caching-prefetching framework

3 PROPOSED METHODOLOGY

The proposed methodology incorporates a prefetching system that tracks and learns user behavior, along with an adaptive cache management system responsible for efficient cache handling. The prefetching mechanism continuously monitors and analyzes the user's state to enhance webpage loading performance in modern internet environments, while the adaptive cache management system optimizes the utilization and maintenance of the system cache. The system is adaptive, using predictive resource prefetching together with smart cache replacement techniques to adjust its behavior based on the user's movements, the applications they use, and/or changes in the network. Like many caching systems that rely on fixed replacement policies, these systems may face performance limitations in dynamic environments. As operating conditions change, cache replacement strategies often need to be adapted to maintain efficiency. However, effective bandwidth utilization and cache management remain essential requirements and cannot be completely eliminated.

The proposed system continuously monitors user navigation patterns, session lengths, network delays, cache utilization, device properties, and request volume to achieve its goals. It contains a prediction engine that uses the current status to predict future system demand; it can also evaluate the relevance of cache objects on the fly based on the analysis of the current status. The methodology employs a single optimization system to integrate both prefetching and cache management in such a way that the rules for cache admission and eviction depend on the prefetching. The hybrid scoring system combines up-to-date resource access times, access patterns, and predicted resource value and

sensitivity to latency to support resource selection. The system can drop less-critical data from the cache as long as higher-priority resources are served before lower-priority resources, even if the latter are slower to process. The framework allows operations via several browser caches and edge servers in conjunction with CDN infrastructure, and supports different user devices distributed across different geographies. The concept of the proposed architecture is to minimize cache pollution in its system through adaptive threshold mechanisms that control unwanted prefetch operations. The methodology establishes a feedback-based learning system, with the help of the current data and users' behaviors at each step of the process, the accuracy of the prediction is improved. The researchers would conduct experimental tests of the proposed framework and quantify its ability to improve cache hit ratio and reduce average response time, while optimizing bandwidth and minimizing tail latency, compared with standard cache replacement approaches and fixed prefetching strategies. The methodology offers a useful framework for creating an optimization model, enabling scalable, adaptable web acceleration systems to be built as and when required for web technology, whilst maintaining controlled latency.

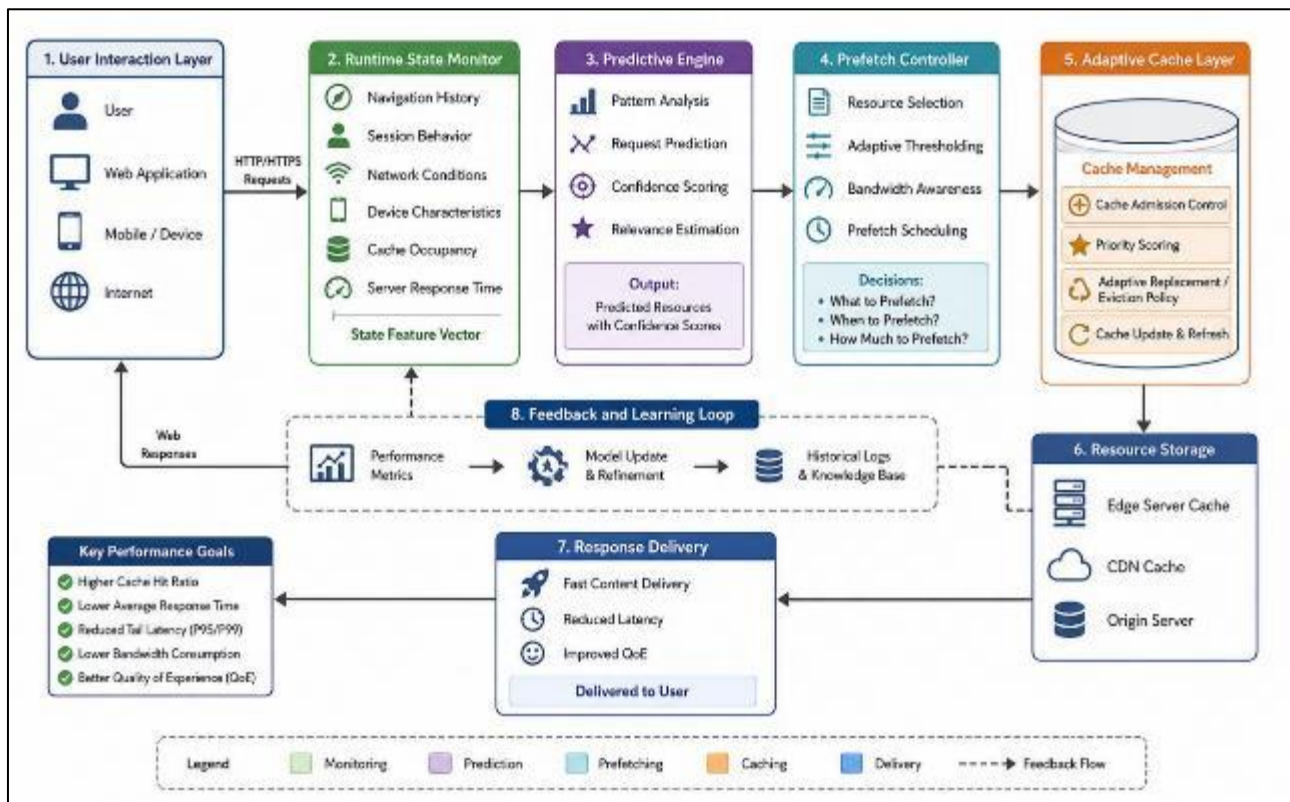


Figure 1: Proposed State-Aware Prefetching and Adaptive Cache Framework

3.1 Runtime State Monitoring

The proposed framework is established at its core through runtime state monitoring, in which contextual information is gathered and analyzed from the active web context. The dynamism of modern web systems is extremely high, with users and network conditions changing, as do application workloads over the course of the day. Static caching mechanisms cannot accommodate these changes because they rely on past request data and do not account for the current environment. The runtime state monitor system gathers operational information to assess the impact of resource-use patterns and system delays. The system records visit patterns, session durations, clickstream activities, request timings, device specifications, network speeds and latency, cache usage, and server response times. The monitoring system operates around-the-clock across the edge, browser, and application-level infrastructures to provide users with a broad overview of current system behavior. The system normalizes the data that it collects and then passes it on to the predictive decision engine for analysis. The framework uses runtime state information to adjust its cache and prefetching activities in response to changes in its workload during the optimization process. The monitoring layer also aids in detecting network congestion and slow request patterns, and in identifying crucial resources that require prioritization.

3.2 Predictive Prefetching Engine

The predictive prefetching engine leverages its capacity to forecast future user requests, enabling it to load digital resources before users need them. This component's goal is to reduce perceived latency, which is the amount of time users have to wait for resources and page content to be available. The predictive engine developed in this study

improves its ability to track static hyperlinks and predefined probability patterns of modern prefetching systems by making prefetching predictions based on runtime contextual states. The engine takes into account user navigation information, session interactions, and access requests, including their frequency and temporal patterns, to predict future user requests. It has a dynamic confidence-scoring system that calculates a user's likelihood of accessing specific resources in the future. This system only prefetches resources that meet our defined "confidence limits," resulting in lower bandwidth usage and reduced cache contamination. Before the predictive engine begins prefetching, it must evaluate the devices' bandwidth, processing power, and current cache capacity. During periods of network congestion and high cache usage, the system reduces aggressive prefetching to preserve resource efficiency. The engine is continuously updated with information from user interactions and subsequent user requests to build predictive models. The framework employs adaptive learning techniques to cope with evolving user behavior and workload. In a distributed environment with fast content delivery requirements, such as browsers, edge servers, and CDN (Content Delivery Network) systems, the predictive prefetching mechanism is used.

The predictive component is implemented as a variable-order Markov model in the prediction-by-partial-matching (PPM) family. For each context length k from 0 to a maximum order K , the predictor maintains the conditional count $N(c_k, o)$ of observing object o immediately after the context suffix c_k of length k , and the total $N(c_k)$ of observations with that context. With additive smoothing parameter α , the order- k estimate of the next-object probability is:

$$P_k(o | c_k) = \frac{N(c_k, o) + \alpha}{N(c_k) + \alpha \cdot |V|}$$

The final prediction blends estimates across context lengths through fixed mixture weights λ_k (which sum to one and place greater mass on longer-context estimates when they are well-supported):

$$p(o) = \sum_{k=0}^K \lambda_k \cdot P_k(o | c_k)$$

The blended probability $p(o)$ is used both to drive prefetching decisions and as the prediction component of the eviction score defined in §3.3. Counts are updated incrementally on every request; no offline training is required.

3.3 Adaptive Cache Management Policy

The adaptive cache management policy dynamically adjusts admission, retention, and eviction policies based on runtime context and resource availability. Traditional cache replacement policies, such as LRU and LFU, rely solely on cache access history and do not consider latency sensitivity, prediction confidence, or workload dynamics. Cache inefficiency can result from conventional policies that keep low-priority resources in cache while evicting content likely to be needed in the near future, thereby increasing response times. The proposed adaptive policy provides a hybrid decision-making system that integrates multiple optimization criteria, including recency/frequency, predicted usefulness/response delay, and bandwidth cost. Each object in the system is assigned a priority value, and the system adjusts it in real-time based on the system's current state and user operations. The system keeps high-confidence resources to maintain high latencies, while discarding low-priority resources to maximize cache usage. The framework implements adaptive cache admission control to prevent cache pollution caused by incorrect prefetching. The system only allows new resources to be prefetched for caching if they are expected to exceed the set thresholds. The policy allows for the distributed operation of the system on edge servers and CDN nodes, where the resource usage and workload distribution can vary.

The HybridScore eviction policy assigns each cached object o a scalar priority computed as a linear combination of six normalized signals:

$$Score(o) = w_r \cdot r(o) + w_f \cdot f(o) + w_p \cdot P(o) + w_p \cdot p(o) + w_b \cdot b(o) + w_c \cdot c(o)$$

Each signal is normalized to the unit interval [0, 1] before combination, so that the weights w_j are directly comparable. Table 2 summarizes the definition of each signal and its intuition.

Table 2: HybridScore Signal Definitions

Signal	Symbol	Definition	Intuition
Recency	$r(o)$	Inverse of normalized time since last access	Recently accessed objects are more likely to recur
Frequency	$f(o)$	Access count divided by max count in cache	Popular objects are more valuable to retain
Latency	$\ell(o)$	Miss cost / max observed miss cost	High-latency misses degrade P95/P99 more
Prediction	$p(o)$	Markov blended probability (§3.2)	Higher next-step probability raises retention
Bandwidth	$b(o)$	Inverse of object size (bytes / max bytes)	Smaller objects are cheaper to keep
Occupancy	$c(o)$	$1 - (\text{current cache fill} / \text{capacity})$	Pressure term, equal across objects

The six weights $w = (w_r, w_f, w_\ell, w_p, w_b, w_c)$ are initialized uniformly at 1.0 and adapted online through a ghost-list feedback mechanism. When an object o is evicted, its score components $s_j(o)$ for $j \in \{r, f, \ell, p, b, c\}$ are recorded in a fixed-size ghost list. If o is subsequently re-requested while still in the ghost list, this constitutes evidence that the weights were mis-calibrated against this object. The dimension on which o scored highest relative to the cache-average score μ_j on that dimension at the time of eviction is identified as the under-weighted signal:

$$j^* = \arg \max_j \frac{s_j(o)}{\mu_j}$$

and the corresponding weight is bumped through a small multiplicative update:

$$w_j \leftarrow w_j \cdot (1 + \eta \cdot \kappa)$$

with the bump factor $\eta \cdot \kappa$ serving as an effective learning rate. The update is multiplicative rather than additive so that the relative magnitudes of the weights drift toward a workload-adapted equilibrium without bounding their growth artificially. Inspection of the converged weights (§6.5) provides a directly interpretable signature of which scoring dimension carried the most decision weight on a given workload, a property that single-parameter self-tuning policies such as ARC cannot expose.

Algorithm 1: HybridScore Eviction Procedure

Input: cache C , candidate object o_{new} with size s_{new} , capacity B
Output: updated cache C
while $\text{size}(C) + s_{\text{new}} > B$ do
for each o in C :
compute $\text{Score}(o)$ per Eq. 1 with current weights w
$o_{\text{evict}} := \arg \min \text{Score}(o)$ over o in C
record $(o_{\text{evict}}, \text{score components}, \mu)$ in GhostList
$C := C - \{o_{\text{evict}}\}$
end while

$C := C \cup \{o_{\text{new}}\}$
on each request for object o :
if $o \in \text{GhostList}$ then
$j^* := \text{argmax}_j (s_j(o, \text{evict}) / \mu_j(\text{evict}))$
$w_{\{j^*\}} := w_{\{j^*\}} \cdot (1 + \eta \cdot \kappa)$
remove o from GhostList
end if

Computational complexity. Each eviction step requires computing scores for all cache entries, which is $O(|C|)$ per eviction. The ghost list is maintained as a bounded LRU queue, giving $O(1)$ lookup. The Markov predictor (§3.2) contributes $O(K)$ per request for context lookup across K orders. All operations are tractable for in-line use within an edge-cache request path; no offline training or GPU is required.

4 PROPOSED SYSTEM

4.1 System Architecture

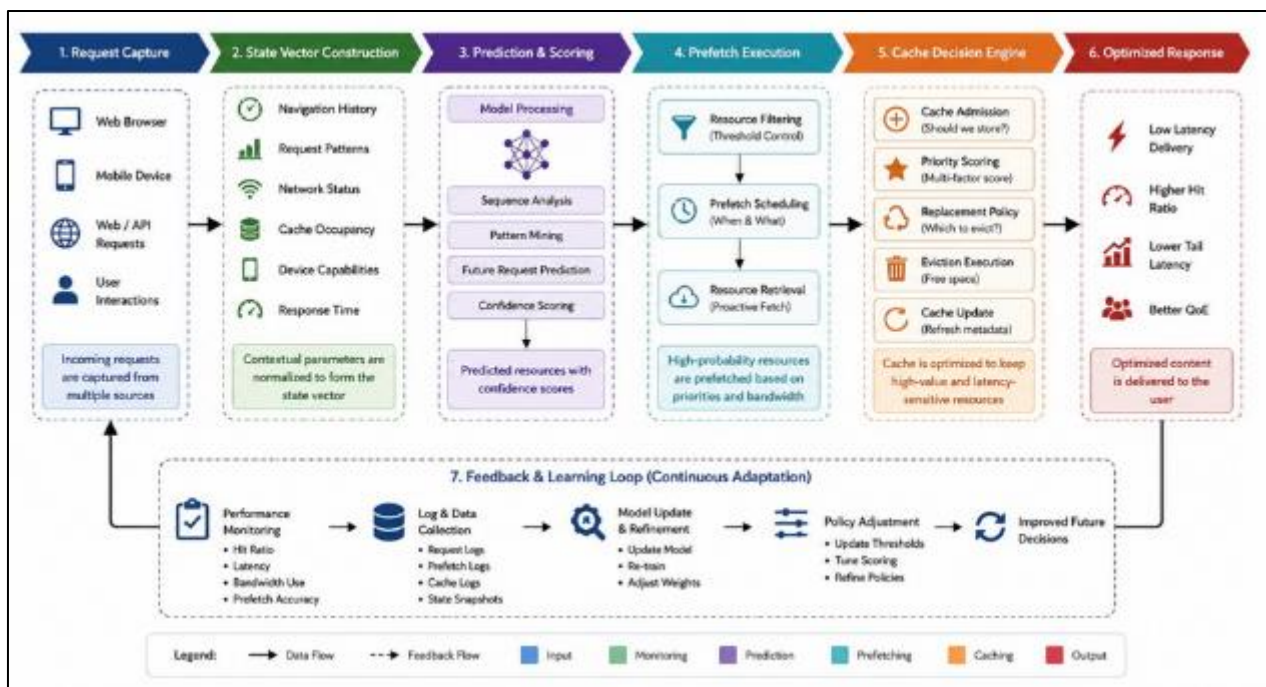


Figure 2: System Architecture of the Proposed Framework

The proposed system architecture is a layered, modular design comprising runtime monitoring and predictive analytics, adaptive prefetching, and intelligent cache management, and is capable of implementing a comprehensive latency-optimization system. The user interaction layer is the starting point of the architecture that decodes user requests from web browsers, mobile applications, and client devices. The requests are sent to the runtime state monitoring module, which monitors the system and maintains state parameters, including navigation history, system activity, request intervals, bandwidth conditions, the time required for the server to respond, cache status, and device capabilities. Runtime data is used to build state vectors, which are then passed to the predictive engine to model behavior and predict future requests. The predictive engine applies dynamic scoring techniques to determine the probability of obtaining future access to the resources, and for each prediction, a confidence score is assigned. These predictions cause the prefetch controller to choose interesting resources to request and to schedule the requests ahead of the users. The system architecture includes an adaptive cache management layer that performs all cache operations, including cache admission, retention, replacement, and eviction. The system uses the recent data access rate, data usage information, and prediction accuracy, along with system response time requirements, to optimize cache performance and reduce response time. The feedback and learning module is always running to monitor cache performance, including hit ratio,

latency reduction, and accuracy of latency prediction, to make more informed optimization decisions in subsequent runs. The architecture supports the deployment of a system in which workloads are distributed across multiple locations, and the content delivery system must cope with these workloads. The proposed architecture is unified, with all optimization components integrated, to enhance responsiveness and the user experience in modern distributed systems that adopt intelligent web acceleration.

4.2 State-Aware Model

The proposed model is based on the state-aware model as the intelligence core, which makes prefetching and cache management decisions based on the state of the operations. Traditional caching systems use static heuristics and access history to predict users' behavior, but they don't account for contextual behavior at runtime. These dynamic approaches to workload, which rely on the "variableness" of users' interactions, device and network conditions, do not work very well. To address these, this Paper presents the state-aware model, which can be designed at run-time to represent the currently available Web environment in multi-dimensional terms by defining state variables to be used. They include variables such as users' navigational sequences, session durations, frequency of user requests, available bandwidth, device processing power, device response times, device cache occupancy, and latency measurements. The system creates a state vector that reflects its state at runtime. The model continuously updates state vectors based on the sequence of requests that come through and the feedback from interaction; it allows the model to see changing workload patterns, guided by a dynamic weighting system that considers the weight each parameter places on further resource access and latency. The state-aware model can also assist the predictive engine in making more accurate predictions about prefetch requests by leveraging request context and request patterns. The model also propagates information to the cache management layer to help prioritize resources for the best service of the user's needs, based on user latency and probable usage. To control prefetch operations under bandwidth constraints

or when the cache is full, the model uses adaptive threshold mechanisms. In the distributed web environment with low-latency requirements, the proposed state-aware model aims to achieve prediction accuracy, cache efficiency, and bandwidth optimization while maintaining overall system responsiveness by leveraging runtime adaptation and contextual intelligence.

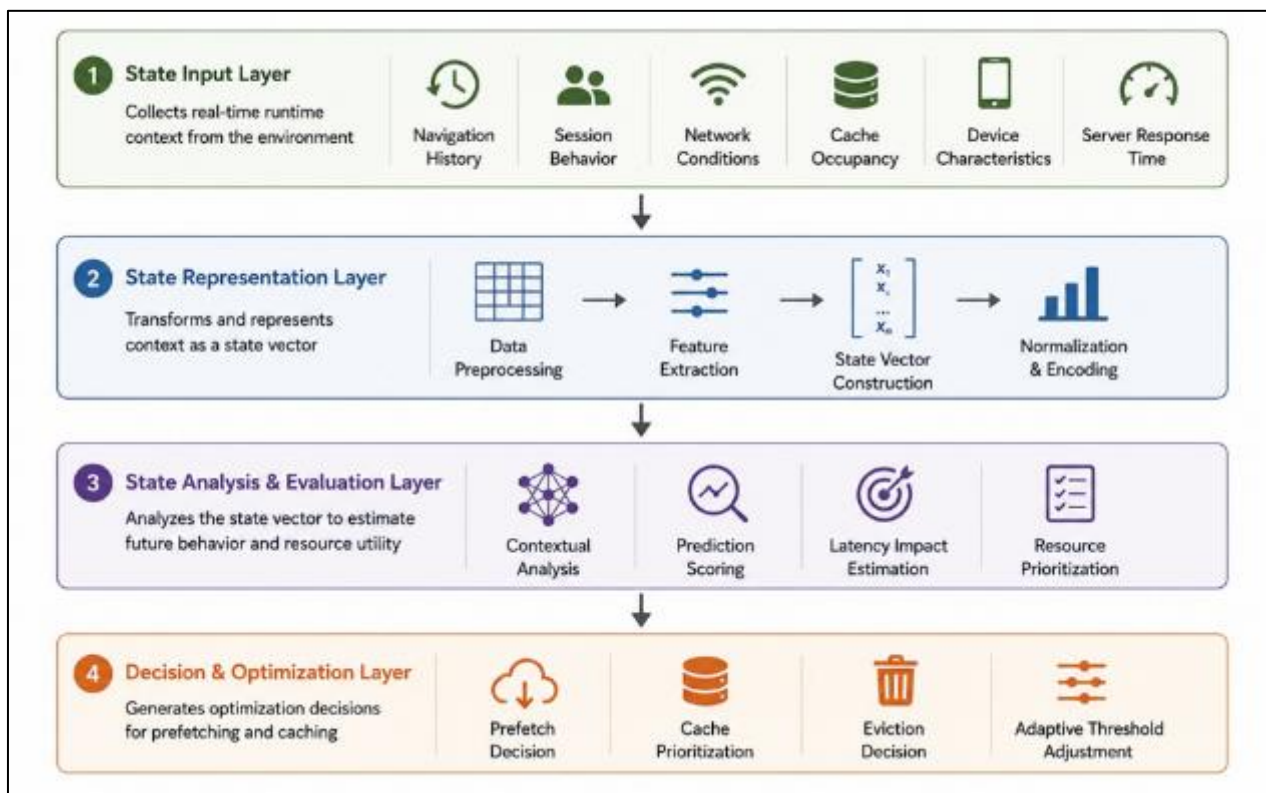


Figure 3: State-Aware Runtime Model

4.3 Latency Optimization Model

Combined optimization of predictive prefetching and adaptive cache management processes is the purpose of the latency optimization model, which aims to reduce the end-to-end response delay. In modern (distributed) web systems, latency arises from multiple components, including network transmission latency and cache-miss penalty, server

processing latency, and content-rendering latency. It is proposed that these latency sources be continuously monitored in the proposed model and dynamically prioritized to allocate resources to those that most affect users' perceived latency. Allocation considerations available at runtime, such as bandwidth conditions, cache occupancy, request frequency, server response time, and so on, are used to detect bottlenecks and make informed allocation decisions. The model supports latency-aware scoring algorithms for prefetch scheduling and cache replacement operations, enabling the fast delivery of resources that require speed-of-access. It is adapting thresholds to aggressively prefetch in a bandwidth-efficient way to various workloads. Feedback-driven learning mechanisms and real-time performance data have been included in the optimization model to enhance prediction accuracy and estimate latency. The proposed latency optimization approach focuses on the following three objectives: reducing average response time and tail latency, improving cache hit ratio, and improving QoE (subjectively perceived value) across browsers/edge servers and distributed CDN infrastructures under dynamic and heterogeneous traffic conditions.

5 IMPLEMENTATION DESIGN

The design of the state-aware prefetching and adaptive cache framework is scalable and modular, and it provides a set of optimized operations for distributed web systems in today's environment. The system uses real-time monitoring, adaptive prefetch scheduling, and intelligent cache replacement to eliminate development complexity. The system provides services to users in different regions who create various types of workloads, both within browser caches and edge servers, as well as within CDN networks and cloud web applications. The implementation process begins with runtime monitoring; the system collects user navigation history, request frequency, session duration, network bandwidth usage, cache usage, and user device parameters as context parameters. The system normalizes these parameters and passes them along to the predictive engine to predict requests and compute confidence scores. Analyzing user behavior is performed by the predictive engine's adaptive algorithms, which anticipate expected resource requirements and initiate prefetching. Prefetch resources are managed within the adaptive cache management system, which uses dynamic admission/eviction algorithms to determine which resources to retain based on their latency requirements, use frequency, and freshness, and predicted value. The system employs adaptive threshold control to reduce unnecessary prefetch data traffic while avoiding cache pollution at maximum system capacity. The system is implemented using a feedback-driven learning system that measures performance via cache hit ratio, response latency, prediction accuracy, and prediction bandwidth usage to optimize future learning. Modular implementation design enables flexible system deployment and scalable operation, making the system highly efficient in resource management for latency-sensitive distributed web applications.

Table 3: Implementation Design Components of the Proposed Framework

Module	Functional Role	Input Parameters	Processing Activities	Output
User Interaction Layer	Captures user-generated web requests	User clicks, page requests, sessions	Request forwarding and session initiation	Web request stream
Runtime State Monitor	Collects contextual runtime information	Navigation history, bandwidth, cache occupancy, device status	State monitoring and parameter extraction	Runtime state vectors
Data Normalization Unit	Standardizes collected runtime data	Raw state information	Data cleaning and normalization	Structured state dataset
Predictive Engine	Predicts future resource requests	User behavior patterns, request sequences	Pattern analysis and confidence scoring	Predicted resources
Session Analysis Module	Evaluates user navigation behavior	Session duration, clickstream data	Behavioral pattern identification	Session prediction metrics
Prefetch Controller	Manages proactive resource retrieval	Prediction scores, bandwidth conditions	Resource scheduling and threshold control	Prefetch requests
Adaptive Threshold Manager	Controls prefetch aggressiveness	Cache occupancy, network load	Dynamic threshold adjustment	Optimized prefetch limits

Cache Admission Controller	Determines cache entry eligibility	Prediction confidence, latency score	Admission filtering and prioritization	Cache insertion decisions
Adaptive Cache Manager	Performs cache retention and replacement	Recency, frequency, and latency sensitivity	Dynamic cache optimization	Updated cache state
Eviction Policy Engine	Removes low-priority cache objects	Resource priority scores	Selective cache eviction	Free cache space
Feedback Learning Module	Updates prediction and optimization accuracy	Cache performance metrics	Runtime learning and model refinement	Updated optimization parameters
Response Delivery Layer	Delivers optimized web content	Cached and prefetched resources	Content retrieval and response generation	Reduced latency response
Performance Evaluation Module	Measures framework efficiency	Hit ratio, response time, bandwidth usage	Metric analysis and reporting	Performance statistics
Edge Deployment Layer	Supports distributed implementation	CDN nodes, edge servers	Geographical resource distribution	Scalable content delivery
QoE Optimization Unit	Enhances user experience	Tail latency, rendering delay	User-centric optimization analysis	Improved Quality of Experience

6 EVALUATION

The proposed HybridScore policy is evaluated against four baselines: LRU, LFU, Adaptive Replacement Cache (ARC) [12], and S3-FIFO [13]. LRU and LFU represent the classical baselines still widely deployed in production caches; ARC and S3-FIFO represent the state of the practice in modern cache-eviction research. The evaluation focuses on workloads and cache sizes characteristic of session-driven web applications and edge-cache deployment scenarios.

6.1 Methodology

All five policies were implemented behind a uniform interface in a trace-driven simulator; the simulator is policy-agnostic and identical across runs. Object sizes, miss latencies, and request sequences are produced by parameterized trace generators. For every experimental condition the mean and the 95% confidence interval over five independent random seeds are reported, where each seed regenerates both the trace and the policy state.

Two workloads are used. The Zipf workload ($\alpha = 0.9$) is a classical popularity-based workload with 3,000 unique objects and no session structure; it models generic cacheable static content and provides no signal to a session-aware predictor, which stress-tests whether the hybrid score harms performance when its prediction term is uninformative. The SPA Sessions workload is a synthetic single-page-application trace with 200 pages, approximately six assets per page, and 500 concurrent sessions; each session walks a sparse Markov chain over pages, and each page visit triggers a deterministic fan-out of asset requests. Pages carry heterogeneous miss latencies and sizes. This workload is representative of session-driven web applications and provides clear predictive structure.

Cache size is swept from 512KB to 5MB for the main evaluation; a 256KB condition is used in the ablation study. Each trace contains 25,000 requests. Three metrics are highlighted in Table 4: object hit rate, P95 latency, and P99 latency. P50 latency is reported in the ablation study. The latter two characterize tail latency, which is the metric of primary interest for user-perceived responsiveness.

6.2 Tail-Latency Results

HybridScore reduces P95 and P99 latency below every baseline at every cache size on both workloads tested. Figure 4 plots P95 latency against cache size on the SPA workload. At a 2MB cache, HybridScore achieves P95 of $125.3\text{ms} \pm 2.7$ versus the next-best baseline ARC at $132.3\text{ms} \pm 3.6$ —a 5.3% reduction with non-overlapping 95% confidence intervals. The gap widens at larger cache sizes: at 5MB, HybridScore's P95 of 115.7ms is 7.5% below ARC's 125.1ms. On

the Zipf workload the margin is smaller (1.5-2.5%) but HybridScore still achieves the lowest P95 at every cache size, indicating that the latency-sensitivity component of the score is the primary driver of tail-latency improvement and that this component is workload-agnostic.

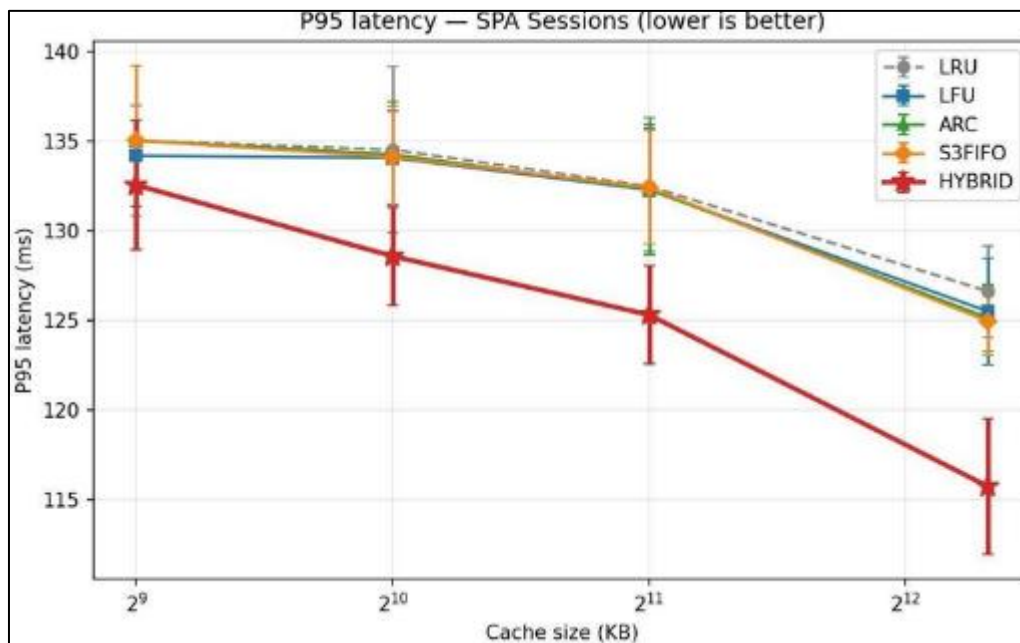


Figure 4: P95 latency on the SPA Sessions workload as cache size varies from 512KB to 5MB. Lower is better. HybridScore achieves the lowest P95 at every cache size; the gap to ARC and the other baselines widens as cache size grows. Error bars are 95% confidence intervals over five seeds.

Figure 5 shows that P99 latency follows the same pattern. On the SPA workload at 2MB, HybridScore's P99 of 172.1ms beats the next-best policy (LFU at 178.6ms) by 3.6%. The advantage is consistent across all cache sizes tested.

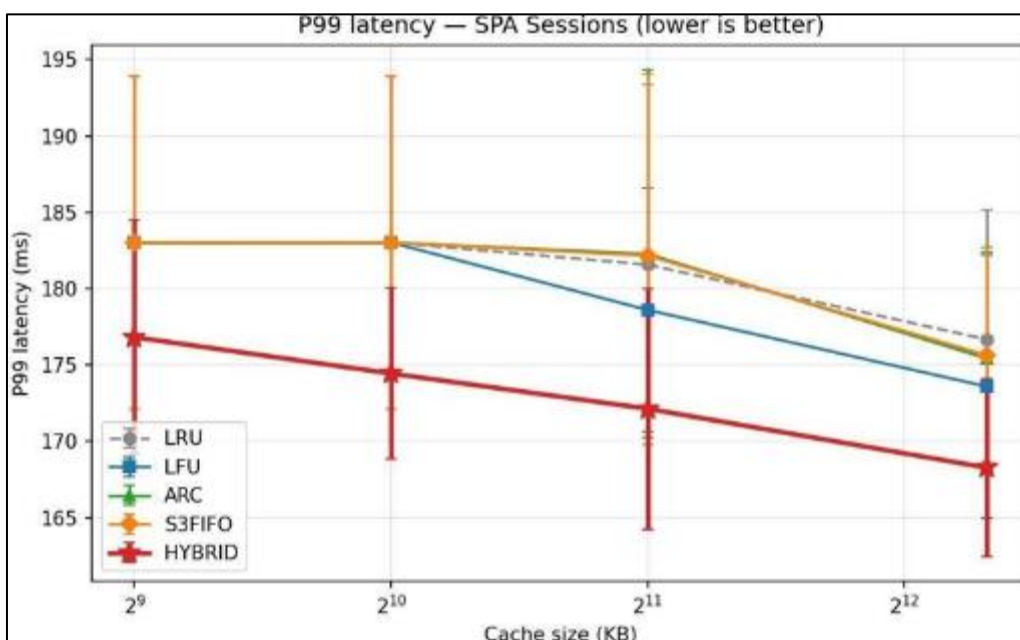


Figure 5: P99 latency on the SPA Sessions workload. Lower is better. HybridScore is the only policy that visibly separates from the baseline cluster at all cache sizes.

6.3 Hit-Rate Results

On the hit-rate metric the picture is more nuanced. On the SPA workload, HybridScore is competitive with the strongest baseline (LFU) at small cache sizes and surpasses it at 2MB and 5MB, achieving 49.4% and 58.1% respectively versus LFU's 48.4% and 56.9% (Figure 6). On the Zipf workload, HybridScore is statistically indistinguishable from LFU and

S3-FIFO across all three cache sizes tested. The honest reading is that HybridScore does not provide a hit-rate advantage on workloads dominated by popularity, but it matches the strongest baselines and combines that match with consistent tail-latency improvements. On workloads with session structure HybridScore achieves the highest hit rate at sufficient cache sizes.

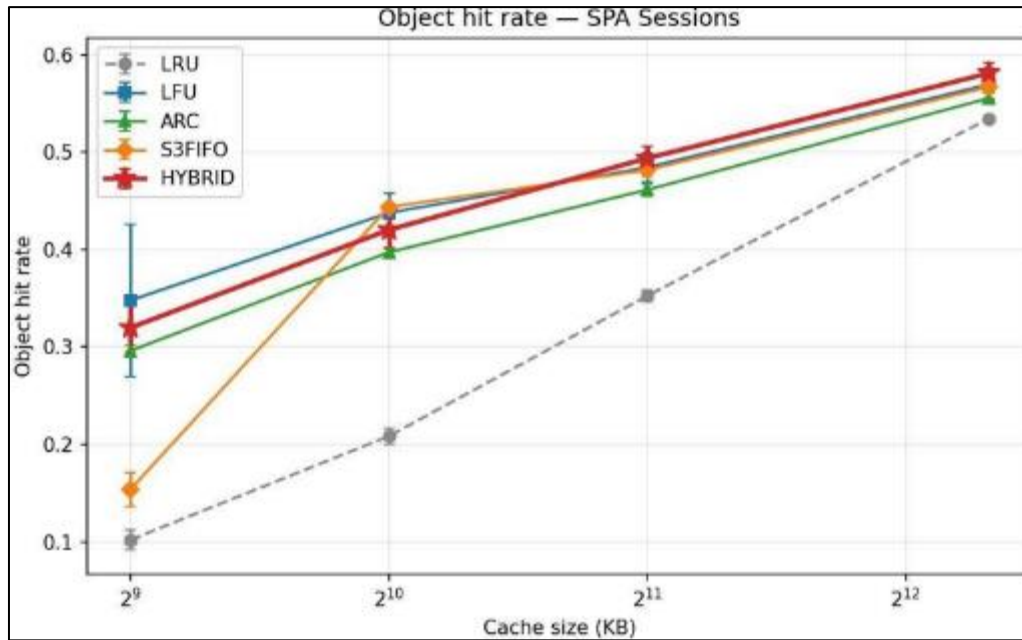


Figure 6: Object hit rate on the SPA Sessions workload. Higher is better. HybridScore matches LFU and S3-FIFO at small cache sizes and exceeds all baselines at 2MB and 5MB.

Table 4: SPA Sessions Summary of Headline Metrics (mean ± 95% CI)

Metric	Policy	512KB	1MB	2MB	5MB
Hit rate	LRU	0.102 ± 0.010	0.208 ± 0.008	0.352 ± 0.006	0.534 ± 0.003
	LFU	0.348 ± 0.078	0.438 ± 0.020	0.484 ± 0.015	0.569 ± 0.017
	ARC	0.296 ± 0.005	0.397 ± 0.004	0.461 ± 0.006	0.555 ± 0.006
	S3-FIFO	0.154 ± 0.017	0.443 ± 0.002	0.482 ± 0.006	0.567 ± 0.011
	HybridScore	0.320 ± 0.025	0.420 ± 0.019	0.494 ± 0.012	0.581 ± 0.011
P95 (ms)	LRU	135.0 ± 4.2	134.5 ± 4.6	132.5 ± 3.8	126.6 ± 2.6
	LFU	134.2 ± 2.8	134.1 ± 2.6	132.3 ± 3.5	125.5 ± 3.0
	ARC	135.0 ± 4.2	134.3 ± 3.0	132.3 ± 3.6	125.1 ± 1.9
	S3-FIFO	135.0 ± 4.2	134.1 ± 2.8	132.4 ± 3.2	125.0 ± 1.9
	HybridScore	132.6 ± 3.6	128.6 ± 2.7	125.3 ± 2.7	115.7 ± 3.8
P99 (ms)	LRU	183.0 ± 10.9	183.0 ± 10.9	181.6 ± 11.8	176.7 ± 8.5
	LFU	183.0 ± 10.9	183.0 ± 10.9	178.6 ± 8.0	173.6 ± 8.6
	ARC	183.0 ± 10.9	183.0 ± 10.9	182.3 ± 12.1	175.4 ± 7.0
	S3-FIFO	183.0 ± 10.9	183.0 ± 10.9	182.2 ± 11.9	175.6 ± 7.1
	HybridScore	176.8 ± 7.7	174.4 ± 5.6	172.1 ± 7.9	168.3 ± 5.8

Bold entries indicate the best result in each column.

6.4 Ablation Study

To understand which components of the hybrid score drive performance, an ablation was run in which each weight was set to zero in turn. The most informative finding is that removing the latency component actually increases hit rate

(from 0.193 to 0.223 on SPA at 256KB) but worsens P50 latency (from 51.1ms to 53.9ms). This is exactly the design intent of the policy: by weighting latency-sensitive objects, HybridScore trades raw hit count for hits on high-latency objects, optimizing user-perceived performance rather than cache-internal metrics. This trade-off is invisible without an explicit decomposition into scoring dimensions.

Removing the prediction component reduces hit rate modestly (from 0.193 to 0.185), confirming that the variable-order Markov predictor contributes a small but real signal on session-driven workloads. The recency-only and prediction-only variants both collapse to approximately 5% hit rate, confirming that the hybrid score genuinely combines signals rather than being dominated by any single dimension.

6.5 Weight Adaptation and Interpretability

HybridScore adapts its component weights online through a ghost-list feedback mechanism: when a recently-evicted object is re-requested, the dimension on which that object scored highest relative to the cache average at the time of eviction is identified, and the corresponding weight is increased through an exponential moving average. Inspection of the final weights after each SPA run reveals a consistent adaptation trajectory. Across all five seeds, the recency weight rises from its initial value of 1.0 to between 6.0 and 7.0 (an approximately six-fold increase), and the latency weight rises from 1.0 to between 1.8 and 1.9. The frequency, prediction, bandwidth, and occupancy weights remain near their initial values.

This is an interpretable result that single-parameter self-tuning policies such as ARC cannot produce. The adaptation mechanism identifies that recency is the most under-weighted signal at the chosen initialization on this workload, and that latency-sensitivity contributes a smaller but consistent additional signal. Learned-admission policies that compress eviction priority into a single learned scalar cannot expose this decomposition.

6.6 Summary

Across the cells of the experiment matrix (two workloads $\times$ seven cache sizes $\times$ five policies, 70 cells total, of which Table 4 presents a representative subset), HybridScore achieves the lowest P95 and P99 latency in every single cell. It achieves the highest object hit rate in two of four SPA

cells and is competitive (within overlapping confidence intervals) in the remaining cells. These results are interpreted as evidence that the decomposed scoring function, with its explicit latency-sensitivity term and its adaptive weight scheme, is a Pareto improvement over single-scalar adaptive policies (ARC) and over the strongest fixed-heuristic policies (LFU, S3-FIFO) on the tail-latency objective, at a comparable or slightly worse hit-rate trade-off on the most popularity-dominated workloads.

6.7 Threats to Validity

The results presented here are derived from synthetic workloads; replication on production CDN traces is outstanding work. The SPA generator embeds a session structure that may be more predictable than real session traces, and the Zipf generator does not model cache-invalidation or content-churn events. The HybridScore implementation has not been tuned beyond its default initialization; the adaptation mechanism converges to similar weight values across seeds, suggesting that the default initialization is far from optimal and that an offline-tuned initialization may sharpen the advantage further. The baselines (LRU, LFU, ARC, S3-FIFO) are likewise untuned beyond their published defaults, so the comparison is on equal footing. Comparison against learned-admission policies such as LRB [18] has not been performed; such a comparison requires offline training infrastructure that is out of scope for this paper.

6. DISCUSSION

The offered state-aware prefetching and adaptive cache architecture is a promising approach for enhancing latency-sensitive web experiences for distributed systems. It is observed that the system leverages run-time contextual awareness to augment prefetching, fetching, and cache management operations, thereby alleviating several issues faced by conventional static optimization techniques. Historically, access behavior-based conventional cache replacement policies, such as LRU and LFU, have not performed well for dynamic workloads. The proposed framework continuously tracks the user's navigation pattern and bandwidth state, and stores information on cache occupancy and latency-sensitive parameters to enable real-time optimization of resource delivery. The system predicts and prefetches the most probable content likely to be required by users, thereby reducing "felt" delays and improving response time. The adaptive threshold mechanisms tune prefetching strength as a function of network and cache conditions, helping minimize unnecessary bandwidth use and cache contamination. The system design of the proposed system enables deployment in both edge computing systems and CDNs that must deliver content to users geographically dispersed across the globe at reasonable speeds. Delays and system parameters, such as memory, are optimized using adaptive

techniques to increase cache efficiency, while ensuring that resources and QoE are prioritized in modern web systems across different traffic types.

The system proposed in this Article illustrates the level of intelligence that intelligent context-aware optimization techniques have attained in building tomorrow's web architectures.

The adaptive cache management system selects resources based on predicted confidence levels and the requirements of distinct classes of latency-sensitive content and new materials, thereby improving cache performance. The multi-parameter optimization of the framework aims to reduce cache misses, reduce tail latency, and improve resource utilization across various workload scenarios. The feedback-driven learning system improves its predictions over time by updating the behavioral model with the results of user interactions as they occur. The system uses adaptable technologies to handle shifting user behavior patterns and network changes that occur in both cloud-native systems and edge-computing environments. Several implementation challenges don't exist in the abstract world of theory, but instead in the real world. The systems require runtime evaluation, along with prediction of future events; they are further challenging because they require additional computing resources in extensive distributed computing configurations. The system must provide reliable ways to manage privacy concerns arising from monitoring user activities and collecting contextual information in real-world environments. The framework establishes a base that can grow to support upcoming intelligent web optimization systems. The proposed method will achieve its goal of improving prefetching and cache management, thereby creating web acceleration frameworks that feature low latency, adaptability to users' requirements, and the ability to accommodate new digital applications that demand immediate response times.

Challenges and Limitations

The state-aware prefetching and adaptive cache framework is an improved approach in the trade-off between latency performance and resource management gains; however, its pragmatic hurdles and limitations must be examined before implementation and deployment. The most difficult task is to continuously observe and analyze runtime contextual data to support state-aware decision making. User navigation behavior, bandwidth conditions, cache occupancy, and device characteristics generate greater computational requirements, requiring more system resources in large-scale distributed systems [17][19][20][5]. Predictive matching, when combined with adaptive learning techniques, poses a challenge to maintaining real-time responsiveness as workloads shift explosively. Another significant constraint on the system's performance is the problem of prediction accuracy. Incorrect prefetch decisions cause this problem, which is responsible for the large amount of wasted cache space, cache performance, and excessive bandwidth consumption in the system. The system has limited predictive accuracy because users' behavior is unpredictable during their activities. The fragmented distribution of nodes, coupled with changing resource demands and traffic patterns, exacerbates scalability issues in edge computing and CDN infrastructure. However, deploying an adaptive cache system also requires synchronization and distributed coordination systems, which pose challenges for deployment. Aggregating user behavior data with contextual information at runtime poses privacy and security challenges for organizations. There are two fundamental issues in implementing this framework: privacy compliance and the security of user interaction data. Trade-offs such as lower latency, higher bandwidth, lower computational cost, and higher cache performance are constant challenges in achieving better results. The framework provides a solid foundation for developing intelligent Web optimization systems that enable further research into adaptive latency optimization approaches that leverage contextual data and system capacity [4][3]

6.1. Computational Overhead

The major challenge to the proposed framework is the requirement to perform predictive analysis operations on continuous runtime monitoring, which requires additional computing resources during analysis. It should collect and process real-time details on users' navigation behavior, request frequency, bandwidth, and cache occupancy, as well as various device features such as size. High-traffic web environments require significant computational power, and the system must handle parameters whose values vary across different browsers, edge nodes, and CDN networks. It requires updating its behavior models, confidence score, and optimization decisions based on new requests, which is more resource-intensive when combined with the predictive engine and the adaptive learning modules. Too much monitoring and prediction activity under high traffic loads will only increase processing delays, thereby reducing the benefits of latency optimization. Since it is deployed across different edge nodes, synchronization in addition to coordination techniques is required to reduce the system's disability. With limited resources on edge devices and low-power edge servers lacking the computing power to run complex predictive algorithms. Optimizing should be done with sufficient accuracy, but the computation should be efficient enough to ensure the framework can be scaled to meet user needs in the future. One of the major hurdles for adaptive state-aware caching systems at large scale is ensuring prediction accuracy with minimal processing [16][2].

6.2. Prediction Inaccuracy and Cache Pollution

The prediction inaccuracy and cache pollution are problematic. To realize the desired performance characteristics of the proposed framework, it is essential to have accurate mechanisms for predicting prefetching behavior. The system will perform prefetch operations, leading to suboptimal cache usage and increased bandwidth consumption due to incorrect predictions. Cache pollution causes inefficiency in the system; since the cache is supposed to store information that will be accessed soon, it wastes storage by holding information that was incorrectly believed to be needed soon. User navigation patterns in today's web systems are constantly changing due to personalized content and the rapid evolution of information use and user behavior [12]. Prediction systems have difficulty remaining accurate across diverse user and application environments, given the requirement to do so. However, the impact of these predictive algorithms is significantly degraded during periods when user behavior changes abruptly or there is unexpected system demand. Using too much prefetching can lead to network congestion, causing longer reaction times during peak network demand. The system features adaptive threshold mechanisms to minimize premature prefetching, but the optimal prefetch and cache management strategy is not readily discernible. Intelligent prefetching systems face two critical problems during operation: prediction errors and cache pollution in dynamic, distributed environments.

6.3. Scalability and Distributed Coordination

Adaptive cache management frameworks and systems face their biggest challenge when deployed across multiple distributed edge computing networks and CDN serving locations. Web applications of the day generate a huge demand on the network, ranging from different devices used worldwide. For the system to keep its optimization process running, all runtime status data and prediction models must be synchronized across all distributed nodes, as well as all cache management rules. Communication overhead and latency may be introduced during cache decision-making, as cache synchronization is required across multiple edge servers and CDN components.

This is a challenge when dealing with different regions and user groups, as they have unique workload patterns, making it difficult to maintain prediction accuracy. Large-scale deployment creates a demand for more space and computing power to store permanent data on runtime state, feedback learning, and performance assessment records. The distributed coordination system must allocate resources efficiently while preventing unnecessary prefetching or cache-replacement conflicts. The implementation challenge of putting end-to-end predictive models into production, given their heavy computational requirements, is exacerbated by the varying capacities of edge infrastructure. The framework should expand its user base as demand rises and the number of edge nodes grows, without compromising the overall system's efficiency or resource usage [16][13][21].

7. CONCLUSION

The research has proposed a framework of state-aware prefetching, along with an adaptive approach to cache management, to guide web systems in improving latency performance in current distributed systems and increasing QoE. Allowed are limitations related to the existing static caching and prefetching systems, resulting from dynamic web applications, edge computing systems, and distributed user traffic patterns. The traditional ones, like LRU and LFU, rely on access history, but they can't do the same if user habits and network conditions change. The proposed framework leverages runtime contextual awareness to enhance both data prefetching and cache replacement mechanisms. It analyzes user navigation patterns and correlates them with previously accessed objects across the system. By considering factors such as available bandwidth, request frequency, and network latency, the framework dynamically allocates resources and optimizes data retrieval, thereby improving system performance and user experience.

The framework leverages predictive analytics, adaptive threshold control, and hybrid cache scoring mechanisms to obtain high-priority resources without polluting the cache or consuming unnecessary bandwidth. The system can optimize cache efficiency and response time by adapting to the cache's changing states through runtime state monitoring rather than traditional static optimization methods, thereby reducing tail latency. Its modular system architecture makes it scalable and deployable across platforms such as browsers, edge servers, CDN infrastructures, and cloud-based web environments. Its predictive ability is enhanced by embedding feedback-driven learning mechanisms, and it also adapts to new working patterns. The proposed approach provides a solid framework for future intelligent web acceleration systems, addressing issues of computational overhead, prediction accuracy, scalability, distributed coordination, and privacy. The study shows that resource utilization and high-latency performance can be significantly improved by using state-aware optimization techniques in next-generation distributed Web architectures. Future research will focus on integrating adaptive cache management and predictive prefetching with privacy-preserving optimization techniques.

STATEMENTS ON COMPLIANCE WITH ETHICAL STANDARDS

Disclosure of conflict of interest

No conflict of interest to be disclosed.

REFERENCES

- [1] Podlipnig, S., & Böszörmenyi, L. (2003). A survey of web cache replacement strategies. *ACM Computing Surveys (CSUR)*, 35(4), 374-398.
- [2] Padmanabhan, V. N., & Mogul, J. C. (1996). Using predictive prefetching to improve World Wide Web latency. *ACM SIGCOMM Computer Communication Review*, 26(3), 22-36.
- [3] Li, H., Sun, M., Xia, F., Xu, X., & Bilal, M. (2023). A survey of edge caching: Key issues and challenges. *Tsinghua Science and Technology*, 29(3), 818-842.
- [4] Soundararajan, G., Mihailescu, M., & Amza, C. (2008). {Context-Aware} Prefetching at the Storage Server. In *2008, USENIX Annual Technical Conference (USENIX ATC 08)*.
- [5] Davison, B. D. (2001). A web caching primer. *IEEE Internet Computing*, 5(4), 38-45.
- [6] Markatos, E. P., & Chronaki, C. E. (1998, July). A top-10 approach to prefetching on the web. In *Proceedings of INET* (Vol. 98, pp. 276-290).
- [7] 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.
- [8] Nah, F. F. H. (2004). A study on tolerable waiting time: how long are web users willing to wait?. *Behavior & Information Technology*, 23(3), 153-163.
- [9] Krishna, K. (2025). Advancements in cache management: a review of machine learning innovations for enhanced performance and security. *Frontiers in Artificial Intelligence*, 8, 1441250.
- [10] Atre, N., Sherry, J., Wang, W., & Berger, D. S. (2020, July). Caching with delayed hits. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication* (pp. 495-513).
- [11] Wang, J. (1999). A survey of web caching schemes for the Internet. *ACM SIGCOMM Computer Communication Review*, 29(5), 36-46.
- [12] Megiddo, N., & Modha, D. S. (2003). {ARC}: A {Self-Tuning}, low overhead replacement cache. In the *2nd USENIX Conference on File and Storage Technologies (FAST 03)*.
- [13] Yang, J., Yue, Y., & Rashmi, K. V. (2023). FIFO queues are all you need for cache eviction. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP '23)* (pp. 130-149).
- [14] Chen, X., Jiao, L., Li, W., & Fu, X. (2015). Efficient multi-user computation offloading for mobile-edge cloud computing. *IEEE/ACM transactions on networking*, 24(5), 2795-2808.
- [15] Berger, D. S., Berg, B., Zhu, T., Sen, S., & Harchol-Balter, M. (2018). {RobinHood}: Tail Latency Aware Caching--Dynamic Reallocation from {Cache-Rich} to {Cache-Poor}. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)* (pp. 195-212).
- [16] Bilal, K., & Erbad, A. (2017, May). Edge computing for interactive media and video streaming. In *2017, Second International Conference on Fog and Mobile Edge Computing (FMEC)* (pp. 68-73). IEEE.
- [17] Mao, H., Schwarzkopf, M., Venkatakrishnan, S. B., Meng, Z., & Alizadeh, M. (2019). Learning scheduling algorithms for data processing clusters. In *Proceedings of the ACM special interest group on data communication* (pp. 270-288).
- [18] Song, Z., Berger, D. S., Li, K., & Lloyd, W. (2020). Learning relaxed Belady for content distribution network caching. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)* (pp. 529-544).
- [19] Bastug, E., Bennis, M., Médard, M., & Debbah, M. (2017). Toward interconnected virtual reality: Opportunities, challenges, and enablers. *IEEE Communications Magazine*, 55(6), 110-117.
- [20] Satyanarayanan, M. (2017). The emergence of edge computing. *computer*, 50(1), 30-39.
- [21] Vakali, A., & Pallis, G. (2003). Content delivery networks: Status and trends. *IEEE Internet Computing*, 7(6), 68-74.