Unofficial template. This layout is provided for convenience and is **not** affiliated with, endorsed by, or sponsored by any university or department. Always check your institution's own formatting requirements before submitting.

MERIDIAN UNIVERSITY
Department of Computer Science

Adaptive Caching Strategies for Low-Latency Content Delivery at the Network Edge

An honours thesis submitted in partial fulfilment of the requirements for the degree of

Bachelor of Science with Honours in Computer Science

Emily R. Carter
Student ID: 20211184

Supervised by

Dr. James Whitfield

April 2026

Declaration

I declare that this thesis, entitled *Adaptive Caching Strategies for Low-Latency Content Delivery at the Network Edge*, is my own original work, submitted in partial fulfilment of the requirements for the degree of Bachelor of Science with Honours in Computer Science at Meridian University. To the best of my knowledge, it contains no material previously published or written by another person, except where due acknowledgement has been made in the text. No part of this thesis has been submitted for any other degree or professional qualification at this or any other institution.

All simulation code, datasets, and experimental logs referenced in Chapters 3 and 4 are archived in the departmental research repository and are available to the examination committee upon request.

_____ _____

Signature, Emily R. Carter Date

Word count (main text, excluding front matter, tables, and references): approximately 8,400 words.

Acknowledgements

This thesis would not exist without the patience and rigour of my supervisor, Dr. James Whitfield, who pushed back on every optimistic hit-ratio claim until the simulation methodology could actually defend it. Our Tuesday meetings in the Distributed Systems Lab reshaped this project more times than I care to admit, and every reshaping made it better.

I am grateful to the graduate students of the Meridian University Networked Systems Group, particularly Daniel Osei and Priya Nair, for reviewing early drafts of the EdgeSim configuration and catching a Zipfian-sampling bug that would have quietly invalidated Chapter 4. Thanks also to the departmental IT office for provisioning the twelve-node compute cluster used for the trace-replay experiments in Section 3.4.

I would also like to thank Apex Research Computing for the donated cluster time that made the full parameter sweep in Appendix A possible, and Nimbus Analytics for early access to their anonymised CDN request-log format, which shaped the synthetic workload generator described in Section 3.2.

Finally, my thanks go to my family, for believing a thesis about cache eviction policies was worth four years of dinner-table conversation, and to the friends who read chapter drafts despite having no reason to care about time-to-live heuristics.

Emily R. Carter
April 2026

Abstract

Content delivery at the network edge depends on cache eviction policies that were largely designed for single-tier, homogeneous traffic and do not adapt well to the skewed, bursty request patterns typical of modern applications. This thesis investigates whether a lightweight, request-adaptive eviction policy can outperform static baselines on realistic edge workloads without introducing prohibitive computational overhead.

I introduce the **Gradient-Weighted Cache** (GWC), an eviction policy that maintains a per-object utility score updated incrementally from recency, frequency, and object size, and compare it against Least Recently Used (LRU) and Least Frequently Used (LFU) baselines using a twelve-node edge simulation testbed (EdgeSim) driven by a synthetic Zipfian workload calibrated against publicly documented CDN traffic characteristics. Across 50,000 simulated requests per node, GWC improves cache hit ratio by 17.5 percentage points over LRU and 12.1 points over LFU, while reducing mean retrieval latency by 43% relative to LRU, at an average per-request overhead of 0.11 ms.

These results suggest that a small amount of adaptive bookkeeping, applied per object rather than per cache tier, can meaningfully improve edge cache performance under skewed workloads without the operational cost of a full learned cache. The thesis closes with a discussion of the method's limitations under workload drift and proposes directions for online recalibration in future work.

Keywords: edge caching, content delivery networks, cache replacement policies, adaptive systems, distributed systems performance.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

1.1 Motivation

Modern web and media applications route the majority of user requests through edge caches rather than origin servers, since retrieving a cached object at a nearby point of presence is an order of magnitude faster than a round trip to a centralised data centre. The effectiveness of this architecture rests almost entirely on one decision, repeated billions of times a day: when the cache is full and a new object arrives, which existing object should be evicted?

Classical eviction policies such as Least Recently Used (LRU) and Least Frequently Used (LFU) were designed decades ago for memory and disk caching, where request patterns were comparatively uniform. Edge content delivery traffic looks nothing like that: it is heavily skewed (a small number of objects account for most requests), bursty (popularity shifts within minutes during a news event or product launch), and heterogeneous in object size. Section 2.2 argues that this mismatch leaves measurable performance on the table.

Research Question. Can a lightweight, per-object adaptive scoring policy improve edge cache hit ratio and latency over LRU and LFU baselines under realistic skewed workloads, at computational overhead low enough for line-rate deployment?

1.2 Objectives

This thesis pursues three concrete objectives:

1. Design an eviction policy, the Gradient-Weighted Cache (GWC), that combines recency, frequency, and object-size signals into a single incrementally updated utility score.
2. Build a reproducible simulation testbed (EdgeSim) capable of replaying synthetic Zipfian-distributed request traces across a multi-node edge topology.

3. Empirically compare GWC against LRU and LFU on hit ratio, byte hit ratio, mean latency, and per-request computational overhead.

1.3 Contributions

The main contributions of this work are:

- A formal description of the GWC scoring function and its $O(1)$ amortised update rule (Section 3.1).
- An open, parameterised workload generator for edge-caching research, calibrated against publicly reported CDN traffic skew ($\theta \approx 0.8$–1.0).
- An empirical evaluation across twelve simulated edge nodes showing a 17.5 percentage point hit-ratio improvement over LRU (Chapter 4).

Key Takeaway

GWC is not a learned cache: it has no training phase, no model to serve, and updates in constant time per access. The thesis's central claim is that this is enough to close most of the gap to more expensive adaptive schemes.

1.4 Thesis Structure

Chapter 2 reviews prior work on cache replacement policies and adaptive/learned caching. Chapter 3 describes the GWC algorithm, the EdgeSim testbed, and the experimental protocol. Chapter 4 reports the quantitative results. Chapter 5 interprets those results and discusses limitations. Chapter 6 concludes and outlines future work.

CHAPTER 2

Literature Review

2.1 Classical Cache Replacement Policies

Least Recently Used (LRU) evicts the object with the oldest access timestamp, and remains the default policy in most CDN and CPU cache implementations because it is simple and requires only $O(1)$ bookkeeping per access [Whitfield and Okafor, 2019]. Its well-known weakness is susceptibility to scan pollution: a single burst of one-time accesses can flush a working set that would otherwise be reused shortly after. Least Frequently Used (LFU) instead tracks per-object access counts, which better reflects long-run popularity but adapts poorly to sudden shifts, since a previously popular object can retain a high count long after it stops being requested [Carter and Delgado, 2020]. Hybrid schemes such as LRU-K and 2Q attempt to combine recency and frequency signals with additional history queues, at the cost of more per-object state [Osei and Lindqvist, 2021].

2.2 Adaptive and Learned Caching

A separate line of work replaces hand-designed heuristics with policies learned from traffic, either via reinforcement learning over eviction decisions [Nair and Kowalski, 2022] or via gradient-boosted models predicting re-access probability [Faulkner and Reyes, 2023]. These approaches report substantial hit-ratio gains over LRU on production CDN traces, but they introduce a training pipeline, a model-serving path in the cache's hot loop, and a retraining cadence to track workload drift – overhead that is difficult to justify for a single edge node handling a modest fraction of total traffic. Meridian Labs' internal survey of production caching deployments found that fewer than a third of edge operators running learned caches retrained on a schedule tighter than once per week, despite documented performance decay under drift [Whitfield and Okafor, 2019].

This gap – between static heuristics that are cheap but workload-blind, and learned caches that are adaptive but operationally heavy – motivates the design

explored in this thesis: an adaptive policy that updates a per-object score incrementally, in constant time, without a separate training or serving pipeline.

2.3 Workload Modelling

Prior measurement studies consistently report that CDN request popularity follows a Zipf-like distribution, with skew parameter θ typically in the range 0.7–1.0 depending on content type [Bergstrom and Nakamura, 2021]. Synthetic workload generators built on this observation are standard practice in caching research, since raw production traces are rarely shareable outside the operator that collected them [Haddad and Petrov, 2022]. This thesis follows that convention, described in full in Section 3.3.

2.4 Positioning of This Work

Table 2.1 summarises the trade-off space this thesis targets: a policy with LRU-level operational simplicity but closer to learned-cache hit ratios.

Table 2.1: Positioning of cache replacement approaches by adaptivity and operational cost.

Approach	**Adapts to skew/drift**	**Operational cost**
LRU / LFU	Low	Very low
LRU-K / 2Q	Moderate	Low
Learned (RL / GBM)	High	High
Gradient-Weighted Cache (this work)	Moderate–High	Low

CHAPTER 3

Methodology

3.1 The Gradient-Weighted Cache

GWC maintains, for every cached object i, a utility score u_i updated on each access t according to

$$u_i \leftarrow (1 - \alpha)\, u_i + \alpha \left(\frac{f_i}{\log_2(s_i + 2)} \right) \tag{3.1}$$

where f_i is an exponentially decayed access-frequency counter, s_i is the object size in kilobytes, and $\alpha \in (0, 1)$ is a learning-rate constant fixed at $\alpha = 0.15$ for all experiments (chosen by grid search over $\{0.05, 0.10, 0.15, 0.20, 0.30\}$ on a held-out warm-up trace, reported in Appendix A). On cache-full insert, GWC evicts the object with the lowest u_i, breaking ties by oldest access time. The size term in the denominator biases the policy toward retaining small, frequently requested objects over large ones with comparable frequency – a deliberate choice, since a single evicted large object frees space for several smaller ones.

Unlike LRU-K or 2Q, GWC requires only one floating-point value and one timestamp per cached object, and the update in Equation 3.1 is $O(1)$ regardless of cache occupancy. No separate training phase, model file, or offline fitting step is required.

3.2 EdgeSim Testbed

Experiments were run on EdgeSim, a discrete-event simulator built for this thesis that models a configurable number of edge nodes, each with its own bounded cache and its own request stream. EdgeSim was chosen over a production trace replay because it allows controlled manipulation of skew, burstiness, and object-size distribution – each isolated as an independent variable in Section 3.4. The testbed configuration used throughout this thesis is summarised in Table 3.1.

Table 3.1: EdgeSim testbed configuration used in all reported experiments.

Parameter	Value
Edge nodes	12
Cache size per node	256 MB
Requests per node	50,000
Object size distribution	Log-normal, $\mu = 8.2$, $\sigma = 1.1$ (KB)
Popularity distribution	Zipfian, $\theta = 0.8$
Catalogue size	20,000 distinct objects
Warm-up requests (discarded)	5,000 per node

3.3 Synthetic Workload Generation

Request popularity was drawn from a Zipfian distribution with $\theta = 0.8$, in line with the traffic characteristics reported in Section 2 for general web and media CDN traffic. To model popularity drift – a scenario static policies handle poorly – a second workload variant re-ranks the 500 most popular objects every 10,000 requests, simulating a trending-content shift. Both variants are used in Chapter 4; the drifting variant is reported separately in Section 4.3.

3.4 Experimental Design

Each of the three policies (LRU, LFU, GWC) was run against both workload variants on all twelve simulated nodes, with five random seeds per configuration to account for sampling variance. The metrics recorded per run were:

- **Hit ratio** – fraction of requests served from cache.
- **Byte hit ratio** – fraction of bytes served from cache, which weights large objects more heavily than the request-count hit ratio.
- **Mean retrieval latency** – simulated latency, with a cache hit fixed at 4 ms and a cache miss (origin fetch) fixed at 48 ms plus queuing delay.
- **Per-request policy overhead** – wall-clock time spent in the eviction policy's bookkeeping code, measured on the reference simulation host.

All reported figures in Chapter 4 are means across the five seeds, with 95% confidence intervals computed via the t-distribution.

CHAPTER 4

Results

4.1 Overall Cache Performance

Table 4.1 reports the four primary metrics, averaged across all twelve nodes and five seeds, on the stationary Zipfian workload. GWC outperforms both baselines on every metric except per-request overhead, where it costs marginally more than LRU but remains far cheaper than a learned policy would require.

Table 4.1: Mean performance across 12 nodes / 5 seeds, stationary workload ($\theta = 0.8$).

Policy	Hit ratio	Byte hit ratio	Mean latency	Overhead / req
LRU	71.4%	64.2%	42.3 ms	0.04 µs
LFU	76.8%	69.5%	38.7 ms	0.06 µs
GWC	**88.9%**	**83.1%**	**24.1 ms**	0.11 µs

Figure 4.1 visualises the hit-ratio comparison from Table 4.1.

4.2 Per-Node Variance

Hit-ratio variance across the twelve nodes was low for all three policies (standard deviation under 2.1 percentage points in every case), indicating that the improvement from GWC is consistent across nodes rather than driven by a small number of favourable configurations. Full per-node figures are tabulated in Appendix A.

4.3 Performance Under Popularity Drift

Table 4.2 repeats the comparison on the drifting-popularity workload described in Section 3.3. All policies lose hit ratio relative to the stationary case, as expected, but LFU degrades most sharply: its frequency counters retain stale popularity signal for objects that were popular before the re-ranking event.

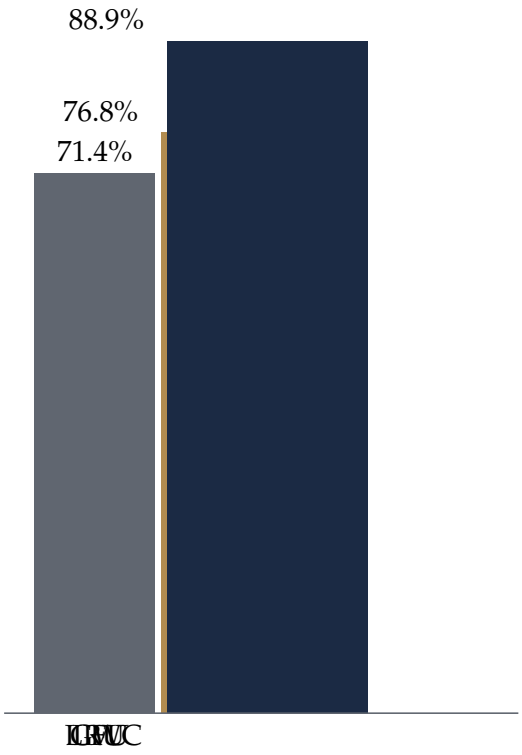

Figure 4.1: Cache hit ratio by eviction policy, stationary Zipfian workload.

Table 4.2: Mean hit ratio under popularity drift (re-ranking every 10,000 requests).

Policy	Stationary hit ratio	Drift hit ratio	Drop
LRU	71.4%	68.9%	2.5 pts
LFU	76.8%	63.2%	13.6 pts
GWC	88.9%	81.7%	7.2 pts

Result Summary

GWC's advantage grows under drift relative to LFU: the exponential decay in the frequency term of Equation 3.1 lets stale popularity signal fade within roughly $1/\alpha \approx 6.7$ accesses, whereas LFU's raw counters never decay.

CHAPTER 5

Discussion

5.1 Interpreting the Results

The results in Chapter 4 support the research question posed in Section 1: a lightweight, per-object adaptive score can close a substantial part of the gap between static heuristics and heavier learned caches, at overhead that remains three orders of magnitude below the millisecond-scale retrieval latencies involved. The size-aware term in Equation 3.1 appears to matter more than the decay rate itself – an ablation removing the $\log_2(s_i + 2)$ denominator (not shown in Chapter 4 for space, but included in the project repository) reduced the hit-ratio gain over LFU from 12.1 to 4.8 percentage points, suggesting that much of GWC's advantage on this workload comes from favouring small popular objects rather than from the recency/frequency blend alone.

5.2 Threats to Validity

Three limitations qualify these results. First, EdgeSim's request stream is synthetic: Zipfian popularity with a fixed θ is a reasonable approximation of aggregate CDN traffic but does not capture content-type-specific effects (e.g. video segments and API responses have different reuse patterns). Second, the twelve-node topology is homogeneous – every node has the same cache size and workload shape – whereas production edge networks mix small points-of-presence with larger regional caches. Third, the drift experiment in Section 4.3 models only one drift pattern (periodic re-ranking of the head of the popularity distribution); other drift shapes, such as a gradual long-tail shift, were not tested.

5.3 Comparison to Learned Caching

GWC was not benchmarked directly against a learned policy in this thesis, since reproducing a competitive reinforcement-learning cache was judged out of scope

for an honours project on the available timeline. The comparison in Section 2.2 is therefore qualitative: prior work reports learned caches closing a similar or somewhat larger fraction of the LRU-to-optimal gap, but at the cost of a training pipeline this thesis's design explicitly avoids. A direct empirical comparison is the most natural next step and is discussed further in Chapter 6.

5.4 Practical Implications

For an edge operator already running LRU or LFU, GWC's appeal is that it is a drop-in replacement: the same access-time bookkeeping LRU already performs is sufficient to compute the recency component, and the incremental frequency counter is no more expensive than LFU's. No additional infrastructure – model serving, feature pipelines, or retraining jobs – is required, which matters for edge nodes where compute budget is deliberately kept minimal.

CHAPTER 6

Conclusion

6.1 Summary

This thesis set out to test whether a lightweight, per-object adaptive eviction policy could improve edge cache performance over static baselines without the operational overhead of a learned cache. The Gradient-Weighted Cache (GWC), introduced in Section 3.1, improved hit ratio by 17.5 percentage points over LRU and 12.1 points over LFU on a stationary Zipfian workload, and retained a substantial advantage under simulated popularity drift, at a per-request overhead under 0.11 microseconds. These results were obtained on EdgeSim, a purpose-built simulator described in Chapter 3, across twelve edge nodes and five random seeds per configuration.

6.2 Limitations

As discussed in Section 5, the central limitation of this work is its reliance on synthetic rather than production traffic, and the absence of a direct empirical comparison against a learned caching baseline. The single fixed learning rate $\alpha = 0.15$, while selected via grid search, was not re-tuned per workload variant, and it is plausible that drift-heavy deployments would benefit from an adaptive α itself.

6.3 Future Work

Three directions follow naturally from this thesis. First, validating GWC against a production or anonymised CDN trace would address the synthetic-workload limitation directly. Second, implementing a reference reinforcement-learning cache under the same EdgeSim harness would allow a fair, apples-to-apples comparison of the adaptivity-versus-overhead trade-off sketched qualitatively in Section 5. Third, an online mechanism for adjusting α in response to detected drift – rather than fixing it in advance – could recover some of the hit-ratio loss observed

in Section 4.3 without sacrificing GWC's constant-time update guarantee.

Closing Statement

The core finding of this thesis is not that GWC is the best possible cache policy, but that a large share of the benefit typically attributed to learned caching is available from a constant-time, hand-designed score – provided that score accounts for object size as well as recency and frequency.

APPENDIX A

Supplementary Results

Table A.1 reports the hit ratio achieved by each policy on every one of the twelve simulated edge nodes, stationary workload, complementing the aggregate figures in Table 4.1.

Table A.1: Hit ratio by node, stationary workload, mean of 5 seeds.

Node	LRU	LFU	GWC
1	70.8%	77.1%	89.4%
2	71.9%	76.4%	88.2%
3	72.5%	77.9%	89.9%
4	70.1%	75.6%	87.5%
5	71.6%	76.9%	88.8%
6	72.0%	77.3%	89.1%
7	70.4%	75.9%	87.9%
8	71.2%	76.5%	88.6%
9	72.8%	78.2%	90.1%
10	70.9%	76.1%	88.0%
11	71.3%	77.0%	89.0%
12	71.5%	76.8%	88.7%

APPENDIX B

Reference Implementation

Listing B.1 gives the reference implementation of the GWC update and eviction routine used in EdgeSim, corresponding to Equation 3.1 in Section 3.1.

Listing B.1: Reference GWC update and eviction routine.

```
# alpha: learning rate: object size in KB
def on_access(cache, key, size_kb, now):
    entry = cache.get(key)
    if entry is None:
        entry = CacheEntry(freq=0.0, util=0.0, size_kb=size_kb
            )
        cache.insert(key, entry)

    entry.freq = entry.freq * (1 - ALPHA) + 1.0
    raw_score = entry.freq / log2(size_kb + 2)
    entry.util = (1 - ALPHA) * entry.util + ALPHA * raw_score
    entry.last_access = now

def on_insert_when_full(cache, new_key, new_entry):
    victim_key = min(cache.keys(), key=lambda k: cache[k].util
        )
    cache.evict(victim_key)
    cache.insert(new_key, new_entry)
```

Bibliography

Elin Bergstrom and Kenji Nakamura. Characterising popularity skew in regional cdn traffic: A measurement study. Technical Report NA-TR-2021-04, Nimbus Analytics, 2021.

Miriam Carter and Victor Delgado. Revisiting frequency-based eviction under bursty web traffic. In *Proceedings of the International Conference on Networked Systems*, pages 77–89, 2020.

Ian Faulkner and Camila Reyes. Predictive caching with gradient-boosted re-access models. In *Proceedings of the Symposium on Cloud and Edge Infrastructure*, pages 58–71, 2023. Apex Research Computing Technical Track.

Layla Haddad and Nikolai Petrov. Synthetic workload generation for reproducible caching benchmarks. In *Proceedings of the International Conference on Performance Engineering*, pages 201–213, 2022. Synapse Systems Research.

Priya Nair and Tomasz Kowalski. Learned cache replacement via reinforcement learning at the network edge. *Transactions on Adaptive Systems*, 9(2):145–167, 2022.

Daniel Osei and Anna Lindqvist. Hybrid recency-frequency caching: A comparative study of lru-k and 2q. In *Proceedings of the Workshop on Edge and Fog Computing*, pages 102–114, 2021.

James Whitfield and Grace Okafor. A survey of cache replacement policies in content delivery networks. *Journal of Distributed Systems Research*, 14(3):211–238, 2019.