Towards a methodology for parameter selection in adaptive cache management

Victoria Vysotska[1,†], Kyrylo Smelyakov[2,†], Anastasiya Chupryna[2,†] and Matvii Kuchapin[2,*,†]

[1] *Kharkiv National University of Internal Affairs, L. Landau Avenue 27 61080 Kharkiv, Ukraine*
[2] *Kharkiv National University of Radio Electronics, Nauky Ave. 14, Kharkiv, 61166, Ukraine*

Abstract
Caching effectiveness in practice depends less on which policy is chosen and more on how its parameters are configured. Static settings rarely track non-stationary workloads, while existing adaptive TTLs and hybrid eviction schemes are under-adopted because there is no reproducible way to select parameters. Building on prior work that introduced adaptive TTL and multi-criteria eviction, this paper shifts the focus from algorithm design to reproducible configuration. We formulate parameter selection as a multi-criteria decision process that maps workload characteristics and operational constraints to scenario-based presets and deterministic correction rules. The methodology provides engine-agnostic guidance for latency-sensitive APIs, reuse-oriented analytics, edge/IoT deployments, and mixed web services, including recommended ranges for lifetime bounds and eviction weights. Rather than reporting new experiments, the contribution consolidates prior empirical insights into practitioner-oriented, workload-aware presets and a lightweight operational validation checklist. The aim is to reduce trial-and-error, improve reproducibility across cache engines (e.g., Redis, Memcached), and lay groundwork for automated, self-tuning caching systems that maintain performance as workloads evolve.

Keywords
adaptive cache management; parameter selection; time-to-live (TTL); eviction policies; multi-criteria decision making; workload-aware caching; performance optimization; reproducibility

1. Introduction

Caching mechanisms are widely employed in data-intensive and high-load systems to reduce latency and pressure on primary storage. However, the effectiveness of such mechanisms depends largely not only on the chosen policy, such as TTL, LRU, LFU, but also on the configuration of key parameters that control these policies. Static parameterization often cannot adapt to workload dynamics. Overly conservative lifetimes lead to inefficient memory usage, while aggressive eviction reduces hit rates and increases tail latency. The presence of heterogeneous data objects, non-stationary access distributions, and strict resource constraints further complicates parameter selection.

Adaptive and hybrid caching strategies have been proposed to address these challenges [1-3]. Nevertheless, existing research primarily focuses on the design of algorithms rather than on systematic approaches to parameter tuning. Unlike algorithmic innovation, which introduces new eviction or TTL strategies, the contribution of this work lies in defining how such mechanisms can be systematically configured. In other words, the focus shifts from what algorithms to use to how they should be parameterized in order to match diverse workloads and constraints.

This paper addresses the identified gap. Building on our earlier work [4], which introduced an adaptive cache management strategy that integrates exponential-decay frequency, recency-aware aging, and a multi-criteria eviction score under different workload conditions, the present paper

CIAW-2025: Computational Intelligence Application Workshop, September 26-27, 2025, Lviv, Ukraine.
[1*] Corresponding author.
[†] These authors contributed equally.

victoria.a.vysotska@lpnu.ua (V. Vysotska); kyrylo.smelyakov@nure.ua (K. Smelyakov); anastasiya.chupryna@nure.ua (A. Chupryna); matvii.kuchapin@nure.ua (M. Kuchapin).
0000-0001-6417-3689 (V. Vysotska); 0000-0001-9938-5489 (K. Smelyakov); 0000-0003-0394-9900 (A. Chupryna); 0009-0006-1953-2893 (M. Kuchapin);

emphasizes configuration rather than new algorithm design. That prior study reported gains in hit ratio and eviction stability compared to LRU and LFU, but also revealed that cache performance remained highly sensitive to parameter choices such as TTL bounds and weight distributions. This sensitivity provides the motivation for the current study: to develop a reproducible methodology for selecting cache parameters in accordance with workload characteristics and operational constraints, thereby reducing reliance on repeated trial-and-error.

The proposed methodology formulates parameter selection as a multi-criteria decision-making process. It considers how workload properties such as access asymmetry, volatility, read/write ratio, and object size distribution influence the relative importance of parameters including recency, frequency, and memory size. It also incorporates operational constraints such as service-level requirements, memory capacity, and processor load, and shows how scenario-based recommendations can be used to generalize prior findings into practical guidelines

The main contributions of this paper are as follows:

- a formalized framework for parameterization of adaptive cache management, independent of a specific cache engine and compatible with exponential-decay TTL models and multi-criteria eviction scoring;
- scenario-based guidelines, including tabular recommendations, for selecting weighting coefficients (w_a, w_f, w_m) and lifetime boundaries (T_{base}, T_{max}) in different deployment contexts;
- a lightweight protocol for deployment and tuning, incorporating profiling, initial calibration, and runtime monitoring, designed to minimize repeated manual adjustments.

In summary, this study advances adaptive cache management from the level of algorithmic design to that of a methodological framework for parameter selection. By codifying parameter choice as a structured process rather than an ad hoc practice, the work facilitates faster adoption in practical systems and establishes a foundation for future research on automated, self-tuning cache management. Beyond its theoretical contribution, the methodology aims to reduce operational trial-and-error and provide practitioners with engine-agnostic, scenario-driven rules that accelerate time-to-value. This practical focus ensures that adaptive caching strategies can be deployed more reliably in production without requiring exhaustive tuning for each new workload.

2. Related Work

Previous studies have revealed both the potential and limitations of adaptive caching. In our earlier study [4], we presented an adaptive strategy that combines dynamic TTL assignment with exponential decay and a multi-criteria eviction score that balances freshness, frequency, and memory consumption. Evaluation demonstrated improvements in hit rate, memory utilization, and eviction stability compared to LRU, LFU, and random eviction, but also revealed a critical limitation: cache performance was highly sensitive to parameter tuning, particularly weighting factors and TTL boundaries. This observation motivated the present work, which shifts the focus from algorithmic design to the development of a systematic methodology for parameter selection for heterogeneous scenarios.

Classic replacement policies such as LRU and LFU are still widely used due to their simplicity and low overhead. However, their reliance on a single criterion often leads to suboptimal behavior under dynamic loads. TTL-based approaches provide additional control in cases where data validity depends on time, but static assignments often result in premature deletion or prolonged storage of stale items. Adaptive TTL variants, such as exponential decay functions and workload-aware policies, increase flexibility but still lack generalized parameter selection rules and often depend on special tuning [5].

Hybrid strategies attempt to combine several criteria to increase adaptability. Examples include composite relevance and frequency scores [6] or adaptive replacement caches (ARC) that respond to changes in popularity distribution [7]. Despite their effectiveness, these approaches primarily emphasize algorithmic novelty rather than systematic parameterization.

More recent work continues to refine adaptive eviction in specialized contexts. CAKE introduces layer-aware eviction indicators tailored for large language models [8], while MadaKV adapts eviction weights for multimodal inference [9]. Both highlight the role of adaptive weighting but remain domain-specific. Similarly, SIEVE (2024) provides a lightweight algorithm that improves miss ratios relative to LRU [10], and Cold-RL (2025) applies reinforcement learning to cache eviction in NGINX by incorporating multiple features while retaining LRU fallback for stability [11].

Survey studies, such as Advancements in Cache Management: A Review of Machine Learning Techniques for Cache Replacement [12], highlight the growing research interest in adaptive and learning-based caching. However, this review also identifies a persistent shortcoming: the lack of reproducible, scenario-driven methodologies for parameter configuration, which continues to limit practical adoption. Similar reproducibility challenges have also been observed in applied ML domains such as road accident detection [13].

In summary, while the literature demonstrates significant advances in adaptive and hybrid cache management, it consistently leaves unresolved the problem of parameter selection. The present work addresses this gap by introducing a methodological framework that formalizes parameter configuration as a reproducible process, thereby extending prior algorithmic contributions with systematic and practice-oriented guidance.

3. Methodological Framework

3.1. Problem Statement

The effectiveness of adaptive cache management strategies strongly depends on how their parameters are configured. While prior research has introduced algorithms that combine recency, frequency, and memory consumption into unified eviction scores, as well as adaptive TTL mechanisms with exponential decay functions [5-7], the selection of configuration parameters remains largely unsystematic. In particular, two groups of parameters are critical:

- weighting coefficients (w_a, w_f, w_m), which determine the relative importance of access recency, frequency of use, and memory footprint within the eviction score;
- lifetime boundaries (T_{base}, T_{max}), which control the dynamic adjustment of TTL values and thereby influence data freshness and resource utilization.

Improper configuration of these parameters can lead to significant performance degradation. For example, assigning excessive weight to recency may cause frequent eviction of long-term popular items, while overemphasizing frequency may result in stale objects occupying memory for extended periods. Likewise, overly narrow TTL ranges can trigger high eviction rates and write amplification, whereas excessively broad ranges may cause cache pollution by retaining low-value entries.

The challenge, therefore, lies in formulating parameter selection as a multi-criteria decision-making problem, in which system designers must balance several conflicting objectives. These include cache hit ratio, read and write latency, memory utilization, and CPU overhead. Furthermore, the importance of each objective varies depending on deployment scenarios: web applications require balanced responsiveness, real-time systems prioritize data freshness, analytics pipelines emphasize frequency-based reuse, and IoT or embedded devices operate under stringent memory constraints.

Despite substantial progress in adaptive caching research [1-3, 12], the literature still lacks a reproducible methodology for parameter selection that generalizes across heterogeneous workloads. Current approaches typically rely on empirical trial-and-error, workload-specific heuristics, or machine-learning models that require extensive retraining. This absence of systematic guidelines hinders both reproducibility and portability of adaptive cache mechanisms.

The present work addresses this gap by developing a methodological framework for parameter selection in adaptive cache management. The framework provides scenario-driven rules, supported by empirical evidence, enabling practitioners to configure (w_a, w_f, w_m) and (T_{base}, T_{max}) effectively without repeated costly experimentation. In doing so, it establishes one of the first reproducible methodologies that bridges empirical evaluation with scenario-based parameterization, advancing adaptive caching from ad hoc tuning toward systematic deployment.

3.2. Adaptive Lifetime Model

The adaptive lifetime model determines the residency duration of cache entries as a function of recent access activity. Static TTL assignments are prone to inefficiencies, while the adaptive approach provides workload-sensitive adjustment. In our earlier study [4], the model was formalized as

$$TTL = T_{base} + (T_{max} - T_{base}) \times f_n, \tag{1}$$

where TTL is the adaptive time-to-live assigned to a cache entry, T_{base} is the guaranteed minimum residency time (lower bound), T_{max} is the maximum residency time (upper bound), and $f_n \in [0,1]$ is the normalized access activity score, typically computed using exponential decay with $\lambda \geq 0$.

The parameters T_{base} and T_{max} provide explicit control over the lower and upper bounds of residency, while the method of computing f_n through exponential decay introduces responsiveness to workload volatility. A shorter λ for the decay function makes the system adapt quickly to recent changes, whereas a longer λ produces smoother, more stable behavior.

The adaptive lifetime model is considered in this work as a fundamental component whose effectiveness depends primarily on how its parameters are configured, rather than on the novelty of its formulation.

3.3. Adaptive Eviction Model

When cache capacity is reached, the system must decide which entries to discard. Single-criterion rules such as LRU or LFU are often insufficient under heterogeneous or time-varying workloads. To capture multiple objectives within one decision, a unified eviction score was introduced in earlier work [4] and is recalled here as a tunable component rather than re-derived in detail:

$$S = w_a \times 1/(a_n + \varepsilon) + w_f \times f_n + w_m \times m_{max}/m, w_a, w_f, w_m \geq 0, w_a + w_f + w_m = 1, \tag{2}$$

where S is the eviction score of an entry (higher implies lower priority for retention); w_a, w_f, w_m are non-negative weights for recency, frequency, and memory footprint (typically normalized); a_n is the normalized access-age term; f_n is the normalized frequency term; m is the entry's memory size (bytes); m_{max} is the maximum entry size observed (bytes); ε is a small constant preventing division by zero.

By adjusting the relative weights, the model can approximate established policies such as LRU when recency is emphasized, LFU when frequency is prioritized, or size-aware strategies when memory footprint dominates. At the same time, it supports intermediate hybrids that balance these objectives. This flexibility provides continuity with classical approaches while extending them into a broader design space where trade-offs can be explicitly managed. Within the scope of this study, the eviction score is considered not as a new algorithm but as a configurable mechanism whose effectiveness depends on systematic parameterization aligned with workload characteristics and resource constraints. In combination with the adaptive lifetime model, it forms one of the two core building blocks of the proposed methodological framework, supporting reproducible and scenario-driven cache management.

3.4. Parameterization Approach

The parameterization of adaptive cache models requires a systematic and reproducible procedure that translates workload characteristics into effective configurations. The approach is presented as an operational protocol rather than as a new experiment.

The process begins with scenario identification, where workloads are profiled using available telemetry data and domain expertise. Characteristics such as freshness versus reuse orientation, capacity load, access pattern variability, read–write asymmetry, and object size distribution are analyzed to assign the deployment to a representative scenario class (e.g., latency-sensitive API, burst-loaded service, or memory-constrained device). This classification determines the subsequent parameter choices. Comparable methodology for scenario-based parameterization has been demonstrated in comparative studies of clustering algorithms for market segmentation [14-18].

Based on the identified scenario, initial parameters are selected for the adaptive lifetime model, the effective freshness sensitivity determined by the λ of the activity signal, and the eviction weights. These seed values are drawn from scenario-based presets and adjusted against operational constraints such as global lower bounds on minimum lifetime, namespace-specific upper bounds, and projected memory utilization.

To verify plausibility, the configuration undergoes a lightweight validation checklist rather than controlled experiments. This checklist typically includes:

- ensuring that latency percentiles remain within service-level targets;
- confirming that eviction and write rates do not exceed acceptable thresholds;
- checking that CPU and memory overhead remain within budget.

When deviations are observed, deterministic rules are applied. Instead of iterative fine-tuning, structured mappings between observed metric deviations and parameter adjustments provide stability and prevent oscillations. Since lifetime and eviction mechanisms interact, adjustments are coordinated to preserve trade-offs between reuse, responsiveness, and memory efficiency.

Once parameters are refined, the configuration is documented in a scenario-to-parameter log, recording the scenario label, final parameter values, justification for adjustments, and observed operational metrics. This log builds a cumulative knowledge base that supports reproducible configuration of future deployments without repeating manual trial-and-error.

4. Operational Validation Checklist

The use of predefined scenarios in adaptive cache management requires a structured mechanism to confirm that the selected parameters behave as expected in real operating conditions. Since no new experiments are proposed in this paper, the purpose of this section is not to describe a controlled evaluation protocol in detail, but to provide a checklist for operational validation. This checklist is intended to be used as a reproducible and simple process that allows system developers and practitioners to verify that a given configuration meets service level requirements, resource constraints, and stability criteria in production environments. By shifting the focus from experimental measurements to operational validation, the methodology ensures reliable and consistent deployment of adaptive cache management across different systems.

The first element of operational validation concerns compliance with service level objectives. Cache configuration directly affects response times, especially at higher percentiles such as P95 and P99, which are often critical in latency-sensitive environments. When applying preset settings based on scenarios, operators must confirm that these latency thresholds remain within the acceptable values defined by the application domain. Hit rate stability is equally important, as it reflects the cache's ability to deliver consistent benefits across varying traffic intensities. Unlike testing, which measures improvements relative to a baseline, operational validation only requires assurance that system-level goals will not be compromised after the configuration is deployed.

The second aspect of validation relates to the efficient use of resources. Caching is often subject to strict limitations on memory and processor power, especially in multi-user or embedded environments. For this reason, operational monitoring must confirm that load levels remain balanced and that memory is neither underutilized nor overloaded. Excessive eviction frequency, which manifests as uncontrolled outflow, can lead to increased write amplification and ultimately reduce the expected performance benefits of caching. Similarly, if eviction policies are not configured properly, large objects can monopolize cache capacity and displace smaller, frequently used records. Therefore, validation includes verifying that memory allocation is proportional, that the eviction rate remains below acceptable thresholds, and that the CPU load caused by caching operations does not interfere with other critical tasks on the same system.

The third element of the checklist involves corrective adjustment using deterministic rules. Since the methodology defines parameterization as a multi-criteria decision-making process, each parameter is linked to observable system metrics that indicate when adjustment is required. For example, if eviction rates exceed acceptable limits, increasing the lower bound of the lifetime (T_{base}) stabilizes cache residency. If stale objects are observed, decreasing the upper bound (T_{max}) restores freshness. If large objects constantly occupy most of the capacity, increasing the weight assigned to memory volume (w_m) in the displacement estimate prevents monopolization. These adjustments are not discovered through iterative trial and error, but are prescribed by structured comparisons between observed deviations and parameter updates. Thus, the validation process avoids fluctuations and maintains system stability.

Documentation is an important component of operational validation. Every setting and every decision must be recorded in a log of scenarios and parameters, which reflects the context of the workload, the settings selected, the deviations observed, the corrective actions taken, and the metrics obtained. Such documentation ensures that the knowledge gained during one deployment can be transferred to subsequent deployments without having to go through the same verification process again. Over time, the accumulation of these records forms a knowledge base that enhances reproducibility and allows for faster adaptation to changes in workloads. It is important to note that this approach is engine-independent and can be applied to various caching platforms, such as Redis, Memcached, or in-memory caching layers built into microservice architectures.

Thus, the operational validation checklist transforms parameter tuning from an informal and potentially unstable practice into a structured process based on service level compliance, resource efficiency, corrective adjustments, and documentation. Unlike experimental evaluation, which is designed to confirm performance improvements over baseline metrics, the checklist is designed to ensure the safety of practical implementation by ensuring that scenario settings are consistent with actual workloads and constraints. This emphasis on reproducibility and portability reinforces the broader methodological contribution of the paper and provides practitioners with a pragmatic path for implementing adaptive cache management in production systems.

5. Scenario-Based Guidelines

The operational validation checklist outlined in Section 4 ensures that cache configurations remain compliant with service-level objectives and resource constraints once deployed. However, practitioners also require clear entry points for selecting parameters before validation can take place. To address this, the following section translates the methodological framework into reproducible scenario-based guidelines that specify recommended ranges for lifetime boundaries and eviction weights. These guidelines do not claim to provide globally optimal values; rather, they function as practical presets that reduce reliance on ad hoc tuning and accelerate adoption in production systems.

The rationale for scenario-based recommendations arises from the diversity of application domains in which caching is deployed. Latency-sensitive APIs, analytics pipelines, edge or IoT environments, and heterogeneous microservice platforms impose distinct requirements that cannot be satisfied by a single configuration. By mapping these domains to representative scenarios and prescribing parameter presets tailored to their dominant characteristics, the methodology bridges theoretical models with operational practice. The subsections that follow present such mappings together with explanatory notes, allowing practitioners to select the most relevant profile and then confirm its suitability using the operational checklist.

5.1. Latency-Sensitive APIs

Latency-sensitive APIs and interactive microservices are characterized by bursty traffic and strict Service-Level Objectives (SLOs) concerning tail latency. In these environments, cache configuration directly affects response times, especially at higher percentiles such as P95 and P99.Consequently, eviction decisions must prioritize recency (w_a) to maintain responsiveness, while lifetime bounds (T_{base}, T_{max}) are kept narrow to ensure rapid churn and prevent memory saturation.

Typical configurations set the guaranteed minimum residency time (T_{base}) between 30 and 60 seconds, which ensures minimum residency but still allows rapid churn. The maximum residency time (T_{max}) is typically set between 300 and 600 seconds, limiting the maximum lifetime to avoid cache saturation. The decay coefficient λ is set relatively high, between 0.3 and 0.5, because a higher λ accelerates aging and makes the system adapt quickly to recent changes, thereby prioritizing recent access activity. The eviction weights (w_a, w_f, w_m) generally emphasize recency, with values around 0.6–0.7, 0.2–0.3, 0.1–0.2, respectively, to meet latency SLOs. Recommended parameter ranges are summarized in Table 1.

This recency-first configuration is essential for meeting strict tail latency SLOs, as it ensures the cache adapts rapidly to bursty traffic and prioritizes the most recently accessed items. While this aggressive eviction of older items might slightly lower the overall hit ratio by removing potentially reusable (but less recent) entries, this trade-off is necessary to maintain the responsiveness required in latency-sensitive applications.

Table 1
Recommended parameters for latency-sensitive API workloads

Parameter	Recommended Range	Notes
T_{base}	30–60 s	Ensures minimum residency but allows rapid churn
T_{max}	300–600 s	Limits lifetime to avoid cache saturation
λ	0.3–0.5	Higher λ accelerates aging, prioritizing recent accesses
w_a, w_f, w_m	0.6–0.7, 0.2–0.3, 0.1–0.2	Emphasizes recency to meet latency SLOs

These values are drawn from prior evaluation [4] and industry observations of web-service caching. They reflect conservative lifetimes and high recency weights, consistent with latency-sensitive design goals.

5.2. Reuse-Oriented Analytics

Analytical workloads, including recommendation engines, are characterized by intensive reuse of historical data. In such scenarios, frequency plays a dominant role, and longer retention periods are required to ensure the availability of valuable elements. The objective of this configuration is to maximize the cache hit ratio and minimize recomputation overhead.

The guaranteed minimum residency time (T_{base}) is set in the range of 5 to 10 minutes, specifically to avoid premature eviction of items that are useful in the long term. The maximum residency time (T_{max}) is set high, ranging from 1 to 4 hours, enabling long-term reuse. The decay coefficient λ is set low, in the range of 0.05–0.1, because a lower λ produces smoother, more stable behavior that preserves long-term popularity trends and smooths out fluctuations. Eviction weights (w_a, w_f, w_m) are shifted strongly towards frequency, with values around 0.2–0.3, 0.6–0.7, 0.1–0.2. Recommended parameter ranges are shown in Table 2.

This configuration maximizes the hit ratio and reduces overhead for recalculation, although it may increase the risk of cache pollution if the access distribution changes unexpectedly. When latency requirements become critical, the recency weight can be moderately increased to restore responsiveness.

Table 2
Recommended parameters for reuse-oriented analytics workloads

Parameter	Recommended Range	Notes
T_{base}	5–10 min	Avoids premature eviction of useful items
T_{max}	1–4 h	Enables long-term reuse
λ	0.05–0.1	Lower λ preserves long-term popularity trends
w_a, w_f, w_m	0.2–0.3, 0.6–0.7, 0.1–0.2	Prioritizes frequency for reuse-heavy workloads

This parameterization reflects the focus of analytical workloads on reuse and is consistent with findings from prior evaluation [4] and with caching strategies described in industry and academic studies.

5.3. Edge and IoT Environments

Edge and IoT workloads are typically subject to stringent memory constraints and exhibit high variability in access patterns. Caching in these domains is often subject to strict limitations on memory and processor power. To prevent a small number of large objects from monopolizing the limited cache capacity, eviction decisions must place strong emphasis on the memory footprint.

Typical T_{base} values are set low, 10–30 seconds, and T_{max} is set between 120–300 seconds. These short bounds are used to ensure rapid rotation, keeping memory available for rotation and preventing long-lived objects from persisting. The decay coefficient λ is set higher between 0.4–0.6, enforcing rapid aging, which is suitable for volatile access patterns. Eviction weights (w_a, w_f, w_m) are skewed toward memory, with distributions around 0.2–0.3, 0.2–0.3, 0.5–0.6. Recommended configurations are summarized in Table 3. Such settings prevent oversized entries from saturating limited capacity, although they may reduce hit ratios due to more frequent evictions. This trade-off is acceptable in contexts where predictability and fairness take precedence over maximizing reuse.

Table 3
Recommended parameters for edge/IoT workloads

Parameter	Recommended Range	Notes
T_{base}	10–30 s	Keeps memory available for rotation

T_{max}	120–300 s	Prevents long-lived objects from persisting
λ	0.4–0.6	Higher λ enforces rapid aging, suitable for volatile access patterns
w_a, w_f, w_m	0.2–0.3, 0.2–0.3, 0.5–0.6	Prioritizes memory fairness over hit ratio

These guidelines reflect practical strategies reported in prior evaluation [4] and in lightweight caching approaches for embedded and IoT systems. The focus is on predictable performance under capacity limitations, even at the expense of reduced hit ratios.

5.4. Mixed Web and Microservices

Heterogeneous environments, such as large-scale web platforms and multi-tenant microservice architectures, require balanced performance across multiple dimensions, including responsiveness, reuse, and memory efficiency. These workloads demand a setting where no single factor dominates and resilience is prioritized across scenarios.

A representative configuration uses moderate parameter values. T_{base} is set between 1–2 minutes, which prevents rapid churn without excessive retention. T_{max} is set higher, between 20–40 minutes, providing stability for moderately reused items. The decay coefficient λ is set to an intermediate range of 0.1–0.3. This intermediate λ is selected specifically to balance responsiveness (recency) and stability (frequency). Eviction weights (w_a, w_f, w_m) are distributed relatively evenly, with 0.3–0.4, 0.4–0.5, and 0.2–0.3, respectively. Recommended configurations are summarized in Table 4.

This balanced setup provides robustness under diverse and shifting workloads, without aggressively prioritizing a single criterion. Continuous monitoring and periodic reassessment remain essential to ensure sustained alignment with workload dynamics.

Table 4
Recommended parameters for mixed web and microservice workloads

Parameter	Recommended Range	Notes
T_{base}	1–2 min	Prevents rapid churn without excessive retention
T_{max}	20–40 min	Provides stability for moderately reused items
λ	0.1–0.3	Intermediate λ balances responsiveness and stability
w_a, w_f, w_m	0.3–0.4, 0.4–0.5, 0.2–0.3	Balanced weights for heterogeneous demands

These ranges are presented as pragmatic compromises between responsiveness and reuse. They are informed by prior evaluation [4] and supported by reports on multi-tenant web and microservice workloads, where no single factor dominates and resilience across scenarios is prioritized.

6. Results and Discussion

The proposed methodology consolidates insights from prior evaluation [4] and published reports into a structured framework for parameter selection in adaptive caching. Rather than presenting new measurements, the emphasis lies in explaining how scenario-based presets are expected to behave under representative workload conditions. This section therefore interprets the rationale and anticipated effects of applying the guidelines introduced in Section 5, showing how systematic parameterization can shape cache behavior across different domains.

For latency-sensitive APIs, configurations dominated by recency weights are expected to improve responsiveness and reduce tail latency, even though they may shorten the residency of items that are useful in the long term. In reuse-oriented analytics, frequency-oriented parameterizations are likely to promote higher hit ratios and minimize recomputation overhead, though they may adapt more slowly to sudden shifts in access distributions. Edge and IoT workloads illustrate how memory-aware settings are anticipated to provide fairness and predictability by preventing large objects from monopolizing capacity, even if this results in lower aggregate hit ratios. Finally, in mixed web and microservice environments, balanced parameter ranges are expected to enhance robustness, ensuring that no single objective dominates and that performance remains stable under heterogeneous and shifting conditions. The deterministic adjustment rules embedded in the methodology provide an additional stabilizing effect. Instead of iterative fine-tuning, operators can rely on structured mappings between observed deviations such as excessive eviction rates or latency violations and parameter updates. This rule-based approach is expected to reduce oscillations and facilitate convergence toward suitable configurations.

Overall, the discussion supports the central premise that adaptive cache management benefits not only from algorithmic design but also from systematic parameterization. By coupling scenario-driven presets with lightweight operational validation, the methodology is anticipated to reduce trial-and-error, improve reproducibility, and enhance transferability across cache engines. In doing so, it offers a pathway for more reliable deployment of adaptive caching strategies in practice.

7. Conclusion

This paper has advanced adaptive cache management by shifting the focus from algorithm design to systematic parameterization. Building on prior evaluations [4] and insights from existing literature, it introduced a methodological framework that treats parameter selection as a reproducible process. The framework integrates two configurable components an adaptive lifetime model and a multi-criteria eviction model into scenario-driven guidelines and deterministic correction rules.

The guidelines provide practitioners with actionable starting points for configuring cache systems under diverse conditions, including latency-sensitive APIs, reuse-oriented analytics, memory-constrained IoT environments, and heterogeneous web or microservice platforms. Rather than prescribing globally optimal configurations, the recommendations function as reproducible presets that reduce reliance on ad hoc trial-and-error and accelerate practical adoption.

The broader contribution of the methodology lies in highlighting reproducibility, interpretability, and transferability as essential elements of cache management, alongside raw performance. By documenting parameter choices as scenario-to-parameter mappings and by embedding lightweight operational validation, the approach enables knowledge accumulation across deployments and supports engine-agnostic use in systems such as Redis or Memcached.

Future work may extend this foundation by exploring automated scenario detection, machine learning–based refinement of parameter mappings, and continuous self-tuning mechanisms that adapt to workload evolution. In this way, adaptive caching can progress toward becoming a robust, self-managing subsystem of modern data-intensive architectures.

Declaration on Generative AI

The authors don’t employed any Generative AI tools.

References

[1] A. Srivatsa, N. Fasfous, N. Anh Vu Doan, T. Wild, A. Herkersdorf, Exploring a Hybrid Voting-based Eviction Policy for Caches and Sparse Directories on Manycore Architectures, Volume 87, Microprocessors and Microsystems, 2021. doi: 10.1016/j.micpro.2021.104384.

[2] Z. Shi, Z. Fan, Optimized Caching Strategy: A Hybrid of Least Recently Used and Least Frequently Used Methods, 5th International Conference on Computer Network Security and Software Engineering (CNSSE '25), Association for Computing Machinery, New York, NY, USA, 2025, pp. 133–140. doi: 10.1145/3732365.3732389.

[3] S. Sharif, M. H. Y. Moghaddam, S. A. H. Seno, A hybrid Bi-level management framework for caching and communication in Edge-AI enabled IoT, Journal of Network and Computer Applications, Volume 232, 2024. doi: 10.1016/j.jnca.2024.104000.

[4] O. Shevchenko, M. Kuchapin, Z. Dudar, M. Shirokopetleva, Enhancing Redis Cache Efficiency Based on Dynamic TTL and Adaptive Eviction Mechanism, 2025 IEEE Open Conference of Electrical, Electronic and Information Sciences (eStream), Vilnius, Lithuania, 2025, pp. 1-6, doi: 10.1109/eStream66938.2025.11016870.

[5] S. Basu, et al, Adaptive TTL-Based Caching for Content Delivery, IEEE/ACM Transactions on Networking, Volume 26, pp. 1063-1077, 2018, doi: 10.1109/TNET.2018.2818468.

[6] A. Srivatsa, S. Nagel, N. Fasfous, N. Anh Vu Doan, T. Wild, A. Herkersdorf, HyVE: A Hybrid Voting-based Eviction Policy for Caches, 2020 IEEE Nordic Circuits and Systems Conference (NorCAS), Oslo, Norway, 2020, pp. 1-7, doi: 10.1109/NorCAS51424.2020.9265136.

[7] P. Singh, R. Kumar, S. Kannaujia, N. Sarma, Adaptive Replacement Cache Policy in Named Data Networking, 2021, pp. 1-5. doi: 10.1109/CONIT51480.2021.9498489.

[8] Z. Qin, et al, CAKE: Cascading and Adaptive KV Cache Eviction with Layer Preferences, ICLR 2025, arXiv, 2025. doi: 10.48550/arXiv.2503.12491.

[9] K. Li, et al, MadaKV: Adaptive Modality-Perception KV Cache Eviction for Efficient Multimodal Long-Context Inference, arXiv, 2025. doi: 10.48550/arXiv.2506.15724.

[10] Y. Zhang, et al, SIEVE is simpler than LRU: an efficient turn-key eviction algorithm for web caches, 21st USENIX Symposium on Networked Systems Design and Implementation (NSDI'24), USENIX Association, USA, 2024, pp. 1229–1246. doi: 10.5555/3691825.3691893.

[11] Aa. Gupta, A. Bhayani, Cold-RL: Learning Cache Eviction with Offline Reinforcement Learning for NGINX, arXiv, 2025. doi: 10.48550/arXiv.2508.12485.

[12] K. Keshav, Advancements in cache management: a review of machine learning innovations for enhanced performance and security, Frontiers in Artificial Intelligence, Volume 8, 2025. doi: 10.3389/frai.2025.1441250.

[13] O. Byzkrovnyi, L. Savulioniene, K. Smelyakov, P. Sakalys, A. Chupryna, Comparison of Potential Road Accident Detection Approaches, Vide Tehnologija Resursi - Environment, Technology Resources, Volume 3, pp. 50 – 55. doi: 10.17770/etr2023vol3.7299.

[14] D. Teslenko, A. Sorokina, K. Smelyakov, O. Filipov, Comparative Analysis of the Applicability of Five Clustering Algorithms for Market Segmentation, 2023 IEEE Open Conference of Electrical, Electronic and Information Sciences (eStream), Vilnius, Lithuania, 2023, pp. 1-6, doi: 10.1109/eStream59056.2023.10134796.

[15] V. Vysotska, Computer linguistic system modelling for Ukrainian language processing, in: CEUR Workshop Proceedings, vol. 3722, 2024, pp. 288–342.

[16] V. Vysotska, I. Kyrychenko, V. Demchuk, I. Gruzdo, Holistic adaptive optimization techniques for distributed data streaming systems, in: CEUR Workshop Proceedings, vol. 3668, 2024, pp. 120–132.

[17] V. Lytvyn, et al, Development of intellectual system for data de-duplication and distribution in cloud storage, Webology 16(2) (2019).

[18] V. Vysotska, et al, Information resource management technology based on fuzzy logic, in: Intellectual Systems of Decision Making and Problems of Computational Intelligence, Springer, Cham, 2020, pp. 164–182.