Quantized Online Gradient Descent for Constant-Time Caching with Dynamic Regret Guarantees

Isidoor Pinillo Esquivel[a], Sara Alouf[a], Damiano Carra[b], Giovanni Neglia[a]

[a] *Inria Centre at Université Côte d'Azur, 2004, route des Lucioles, BP 93, Sophia Antipolis Cedex, 06902, France*
[b] *University of Verona, Strada Le Grazie 15, Verona, 37134, Italy*

Abstract

Online learning provides a strong theoretical foundation for caching, but existing algorithms often rely on idealized arbitrary-precision computations and incur per-request overheads that scale with the catalog size, making them difficult to deploy in practice. To address this limitation, we introduce quantized Online Gradient Descent (qOGD), a caching algorithm that supports heterogeneous object sizes and a broad class of caching performance metrics. qOGD builds on OGD but is inherently quantized. Quantization yields constant amortized time per request and robustness to finite-precision implementations, two properties that make it well suited to practical implementations. We establish guarantees on cache update cost and dynamic regret, and show that qOGD matches the fundamental $O(\sqrt{CT})$ static-regret lower bound for caching. Experiments on real-world traces show that qOGD achieves runtimes and hit ratios comparable to those of state-of-the-art caching algorithms, while reducing cache-update costs.

Keywords: Caching, Online learning, Dynamic regret, Quantization

1. Introduction

Caching is a fundamental technique used across computer systems to reduce access time and improve overall performance by storing frequently used objects in faster storage locations. The caching problem arises from the need to efficiently manage limited high-speed memory resources while minimizing delays caused by accessing slower storage layers. This challenge appears at multiple levels of computing systems, including CPU caches that accelerate instruction and data retrieval, operating system page caches that reduce disk I/O operations, and network or web caches that decrease bandwidth usage and latency. Since cache memory is limited, systems must decide which objects should be stored, updated, or removed, making cache management a critical optimization problem. Ideally, caching policies should remain simple and intuitive while achieving strong performance in practical settings. In low-latency and high-throughput systems, these policies must also be computationally efficient and capable of scaling with growing workloads. Existing research on caching generally falls into three categories: policies designed for implementation efficiency and speed, approaches emphasizing empirical performance on real-world workloads, and theoretical frameworks that focus on providing formal performance guarantees.

The performance of caching mechanisms is often evaluated through competitive analysis and regret guarantees, two frameworks that offer complementary views of online decision-making under uncertain and dynamic request patterns. In competitive analysis [6, 2, 37, 24], an online algorithm is compared with an optimal offline algorithm that knows the entire request sequence in advance. The resulting competitive ratio is a multiplicative guarantee: it bounds the online cost by a constant factor of the offline optimum in the worst case. Consequently, even asymptotically, the average performance of a competitive algorithm need not converge to that of the offline optimum. Regret analysis instead provides an additive guarantee, comparing the cumulative cost of an adaptive algorithm with that of a comparator chosen in hindsight [51, 16, 32]. A sublinear regret bound implies that the time-average gap vanishes

Email addresses: isidoor.pinillo-esquivel@inria.fr (Isidoor Pinillo Esquivel), sara.alouf@inria.fr (Sara Alouf), damiano.carra@univr.it (Damiano Carra), giovanni.neglia@inria.fr (Giovanni Neglia)

over time, but only with respect to the best strategy within a prescribed comparator class. Thus, while competitive analysis emphasizes robustness to worst-case sequences, regret analysis captures adaptation to observed workloads, at the price of restricting the benchmark. Reconciling these perspectives is generally difficult and sometimes even impossible: for example, in smoothed online convex optimization, no algorithm can simultaneously achieve sublinear regret and a constant competitive ratio [5]. This paper follows the regret-analysis line of work.

Despite the strong theoretical guarantees offered by no-regret caching algorithms [18, 36, 34, 29, 40, 15, 9], their deployment remains challenging. Existing approaches typically incur non-constant per-request computational overhead, often scaling with the catalog size, which makes them difficult to apply in caching systems that serve large catalogs and require cache decisions at line rate. Moreover, they typically treat numerical precision as an implementation detail, implicitly relying on arbitrary- or high-precision state variables whose maintenance can be costly and ill-suited to efficient hardware implementation. This gap between theoretical optimality and practical feasibility motivates the search for caching algorithms that retain regret guarantees while remaining computationally lightweight and implementable in real-world systems.

In this paper, we introduce Quantized Online Gradient Descent (qOGD), a caching algorithm that explicitly incorporates numerical granularity into its design. Our approach achieves constant amortized per-request overhead, enabling efficient cache decisions, while maintaining its internal state through integer-valued counters and implementation-friendly integer arithmetic. By accounting for quantization directly in the update mechanism, qOGD bridges the gap between theoretical learning guarantees and practical deployability, making efficient no-regret caching feasible in high-speed systems with large catalogs.

Our main contributions are as follows:

- **Theoretical Guarantees**: While most prior work for caching is restricted to static regret, where the comparator is a fixed cache allocation chosen in hindsight, we establish dynamic regret guarantees for qOGD and we formally quantify how finite-precision impacts regret. In addition, we provide explicit bounds on qOGD cache update cost. These bounds hold under heterogeneous object sizes and arbitrary utility weights.
- **Practical Implementation**: Building on a novel relaxed quantized projection algorithm that may be of independent interest, we provide an implementation of qOGD that achieves $O(1)$ amortized time complexity per request. The algorithm maintains its state through integer-valued counters and avoids priority queues and other ordered data structures.
- **Experimental Evaluation**: We validate qOGD's static regret and switching cost guarantees experimentally. Evaluations on real-world traces demonstrate that qOGD achieves runtimes and hit ratios comparable to state-of-the-art heuristic caching algorithms, while reducing cache-update costs.

The remainder of this paper is organized as follows. In Section 2, we formalize the caching problem and motivate our approach. In Section 3, we introduce OGD with relaxed projections and derive its dynamic regret guarantees. In Section 4, we specify the quantized relaxed projection, establish the performance guarantees for qOGD, provide its optimized $O(1)$ implementation, and extend the analysis to integral caching. We evaluate qOGD against state-of-the-art baselines in Section 5 and discuss future work in Section 6 before concluding in Section 7.

2. Background and Motivation

In this section, we formalize caching with heterogeneous object sizes as an online learning framework and introduce OGD. In what follows, we use bold font to represent vectors. Inequalities between vectors are to be understood component-wise. Table 1 summarizes the main notation used throughout the paper. We use $O(\cdot)$ for standard asymptotic bounds and $O^{\dagger}(\cdot)$ to denote amortized per-request complexity, i.e., total running time divided by the number of requests.

2.1. Problem formulation

We consider a system in which requests for objects are served either by a remote server or by an intermediate cache of finite capacity C. Formally, we consider a sequence of requests for objects drawn from a catalog $[N] := \{1, 2, ..., N\}$. Objects may have different sizes, and we denote by s_n the size (in bytes) of object n.

In fractional caching [36, 35, 50, 33], the cache may store arbitrary fractions of each object. By contrast, in integral caching [15, 40, 34], each object must be stored in its entirety or not at all. We first formulate the problem

Table 1: Table of Notation.

Symbol	Description	Symbol	Description
N, T	Catalog size; Time horizon	S_T, R_T	Switching cost; Regret
n, t	Object index ($n \in [N]$); Round index ($t \in [T]$)	$\boldsymbol{e}(n) \in \mathbb{R}^N$	n-th standard basis vector
$[N]$	Set of integers $\{1, \dots, N\}$	z_n	n-th component of vector $\boldsymbol{z}$
C	Cache capacity	$\tilde{\boldsymbol{z}}$	Scaled version of variable $\boldsymbol{z}$
$\boldsymbol{s} \in \mathbb{R}^N$	Vector of object sizes (s_n is size of object n)	$(\boldsymbol{z})^+$	Component-wise $\max\{\boldsymbol{z}, \boldsymbol{0}\}$
$\boldsymbol{y}_t \in \mathbb{R}^N$	Fractional cache state vector at round t	$\lfloor \boldsymbol{z} \rfloor$	Component-wise floor function
$\tilde{\boldsymbol{f}}_t \in \mathbb{R}^N$	Scaled fractional cache state at round t	η	Step size
$\boldsymbol{z}_t \in \mathbb{R}^N$	Comparator vector at round t	$\tilde{\kappa}$	Tracks $\sum(\tilde{f}_n - 1)^+$
$\boldsymbol{u}_t \in \mathbb{R}^N$	Utility vector at round t	ε	Quantization granularity
r_t	Index of the object requested at round t	$\mathrm{UR}(\cdot)$	Unbiased Rounding
w_{t,r_t}	Utility weight (e.g., 1 or $1/s_{r_t}$)	$\mathrm{PR}(\cdot)$	Poisson Rounding function
$\mathcal{Y}$	Feasibility set	$P_{\mathcal{Y}}$	Euclidean projection onto $\mathcal{Y}$
$\mathcal{V}$	Comparator set	$P^R_{\mathcal{V}}$	Relaxed projection onto $\mathcal{V}$
$\tilde{\mathcal{V}}$	Scaled comparator set	$P^q_{\tilde{\mathcal{V}}}$	Quantized relaxed projection

in the fractional setting, and then discuss the integral setting in Section 2.3. We denote by $y_n \in [0, s_n]$ the amount of object n stored in the cache at time t, for example in bytes. The cache state is then given by $\boldsymbol{y}_t$, which belongs to the capped simplex:

$$\mathcal{Y} = \left\{ \boldsymbol{y} \in \mathbb{R}^N \,\middle|\, \sum_{n=1}^{N} y_n \leq C \text{ and } \boldsymbol{0} \leq \boldsymbol{y} \leq \boldsymbol{s} \right\}. \tag{1}$$

Let $r_t \in [N]$ denote the object requested at time t. The cache may serve the amount y_{t,r_t} of the requested object, while the remaining part can be retrieved from the remote server, incurring a cost $w_{t,r_t}(s_{r_t} - y_{t,r_t}) \in \mathbb{R}^+$. Equivalently, the cache provides a reward equal to $w_{t,r_t} y_{r_t}$. Two common choices for the weights w_{t,r_t} are found in the literature. When $w_{t,r_t} = 1/s_{r_t}$, the reward is the fraction of the requested object served locally, which we refer to as a partial hit. When $w_{t,r_t} = 1$, the reward is instead the amount of data served locally. The reward at time t can be expressed in a compact way by the scalar product $\langle \boldsymbol{u}_t, \boldsymbol{y}_t \rangle$, where $\boldsymbol{u}_t := w_{t,r_t} \boldsymbol{e}(r_t)$ and $\boldsymbol{e}(r_t)$ denotes the r_t-th standard basis vector. We refer to the vector $\boldsymbol{u}_t$ as the utility.

A caching policy $\mathcal{A}$ is a sequence of mappings $(\mathcal{A}_t)_{t=1}^{T-1}$, where $\mathcal{A}_t$ maps the sequence of past utilities and states $((\boldsymbol{u}_\tau, \boldsymbol{y}_\tau))_{\tau=1}^{t}$ to the next cache state $\boldsymbol{y}_{t+1}$. Over a sequence of T requests, the caching policy achieves an aggregate reward $\sum_{t=1}^{T} \langle \boldsymbol{u}_t, \boldsymbol{y}_t \rangle$. Caching policies may be studied in the framework of linear online learning [51, 16, 39, 32, 21]. In this framework, the two main performance metrics are the regret [32, 21] and the switching cost [30, 49]. Regret captures the reward loss relative to a reference cache evolution, called a comparator, while switching cost quantifies the cost of updating the cache state, in terms of the amount of data inserted or evicted from the cache. This cost may stem from memory-management [28, 25] or file fetching [40]. We provide formal definitions of these metrics below.

Definition 1 (Switching cost) The switching cost of a caching policy $\mathcal{A}$ measures the cumulative variation of the cache state, defined as half the ℓ_1-distance between consecutive cache states:

$$S_T(\mathcal{A}) := \frac{1}{2}\mathbb{E}\left[\sum_{t=1}^{T-1} \left\|\boldsymbol{y}_{t+1} - \boldsymbol{y}_t\right\|_1\right] = \frac{1}{2}\mathbb{E}\left[\sum_{t=1}^{T-1}\sum_{n=1}^{N} (y_{t+1,n} - y_{t,n})^+ + (y_{t,n} - y_{t+1,n})^+\right], \tag{2}$$

where the expectation is over the possible random choices of the caching policy $\mathcal{A}$.

Definition 2 (Regret) The regret of a caching policy $\mathcal{A}$ with respect to the comparator sequence $(\boldsymbol{z}_t)_{t=1}^{T} \in \mathcal{V}^T$, under the utility sequence $(\boldsymbol{u}_t)_{t=1}^{T}$ is:

$$R_T\left(\mathcal{A}, (\boldsymbol{z}_t)_{t=1}^{T}, (\boldsymbol{u}_t)_{t=1}^{T}\right) := \mathbb{E}\left[\sum_{t=1}^{T} \langle \boldsymbol{u}_t, \boldsymbol{z}_t - \boldsymbol{y}_t \rangle\right], \tag{3}$$

where the expectation is over the possible random choices of the caching policy $\mathcal{A}$.

Algorithm 1: OGD for caching [36]

1 choose $\eta \in \mathbb{R}^+$ and $\boldsymbol{y}_1 \in \mathcal{Y}$
2 **for** $t \leftarrow 1$ **to** T **do**
3 get $\boldsymbol{u}_t = w_{t,r_t}\mathbf{e}(r_t)$
4 $\boldsymbol{y}_{t+1} = P_{\mathcal{Y}}(\boldsymbol{y}_t + \eta \boldsymbol{u}_t) = \operatorname{argmin}_{\boldsymbol{y}\in\mathcal{Y}} \|\boldsymbol{y}_t + \eta \boldsymbol{u}_t - \boldsymbol{y}\|$

Algorithm 2: OGD with relaxed projection [19]

1 choose $\eta \in \mathbb{R}^+$ and $\boldsymbol{y}_1 \in \mathcal{Y}$
2 **for** $t \leftarrow 1$ **to** T **do**
3 get $\boldsymbol{u}_t = w_{t,r_t}\mathbf{e}(r_t)$
4 $\boldsymbol{y}_{t+1} = P^R_{\mathcal{V}}(\boldsymbol{y}_t + \eta \boldsymbol{u}_t)$

Note that the set of admissible comparator states, $\mathcal{V}$, is not necessarily equal to $\mathcal{Y}$. Often, the comparator is chosen as the best static state in hindsight, and the regret is bounded uniformly over all possible utility sequences, leading to the notion of worst-case static regret (e.g., [36, 40, 9, 29]).

Definition 3 (Worst-case static regret) We define worst-case static regret for an algorithm $\mathcal{A}$ for utilities in $\mathcal{U} \subset \mathbb{R}^N$ and comparators from $\mathcal{V} \subset \mathbb{R}^N$ as follows:

$$R_T^S(\mathcal{A}) := \sup_{(\boldsymbol{u}_t)_{t=1}^T \in \mathcal{U}^T} \left(\mathbb{E}\left[\sum_{t=1}^T \langle \boldsymbol{u}_t, \boldsymbol{z}^* - \boldsymbol{y}_t \rangle \right] \right), \tag{4}$$

where $\boldsymbol{z}^* = \operatorname{argmax}_{z\in\mathcal{V}} \left(\sum_{t=1}^T \langle \boldsymbol{u}_t, \boldsymbol{z} \rangle \right)$.

If the worst-case static regret is sublinear, namely, if $R_T^S(\mathcal{A})/T \to 0$ as $T \to \infty$, then the average per-request reward of algorithm $\mathcal{A}$ asymptotically matches that of the optimal static cache allocation in hindsight. Algorithms with sublinear regret are commonly referred to as no-regret algorithms.

A few papers on caching, such as [50], study worst-case dynamic regret, in which the comparator $(\boldsymbol{z}_t)_{t=1}^T$ is allowed to change over time. We state our results for the more general, comparator-dependent regret in Definition 2. This yields a more informative guarantee, since the bound can be evaluated for any chosen comparator sequence and explicitly reflects its variation over time. Worst-case static or dynamic regret guarantees can then be recovered, when desired, by taking the supremum over the relevant utility sequences and comparator class.

2.2. OGD for fractional caching

Paschos et al. [36] proposed the first caching algorithm within the online learning framework. Their method relies on *Online Gradient Descent* (OGD) and is presented in Algorithm 1. Starting from a feasible cache state $\boldsymbol{y}_1 \in \mathcal{Y}$, OGD updates the cache state after each request by taking a gradient step in the direction of the observed utility vector and projecting the resulting point back onto the feasible set. Formally, $\boldsymbol{y}_{t+1} = P_{\mathcal{Y}}(\boldsymbol{y}_t + \eta \boldsymbol{u}_t)$, where $\eta \in \mathbb{R}^+$ is the step size and $P_{\mathcal{Y}}$ is the Euclidean projection onto the feasible set $\mathcal{Y}$, that is, $\operatorname{argmin}_{\boldsymbol{y}\in\mathcal{Y}} \left\| \boldsymbol{y}_t + \eta \boldsymbol{u}_t - \boldsymbol{y} \right\|$.

When all objects have unit size, $\boldsymbol{s} = \mathbf{1}$ (the uniform-object-size setting), and the reward is measured in partial hits, we have $w_{t,r_t} = 1/s_{r_t} = 1$. In this scenario, Paschos et al. [36] proved that OGD with $\eta = \sqrt{C/T}$ achieves a static regret of order $O\left(\sqrt{CT}\right)$ [36]. As a result, OGD asymptotically attains the same average number of partial hits as the best static cache allocation in hindsight $\boldsymbol{z}^*$.

The main drawback of OGD is the computational bottleneck induced by the projection step. For arbitrary utility vectors $(\boldsymbol{u}_t)_{t=1}^T$, the gradient step may perturb several components of the previous cache state before projection. In this general case, computing the Euclidean projection onto the capped simplex can be costly: the best known algorithm has complexity $O\left(N^2\right)$ [44]. In the caching setting considered here, with a single request per timeslot, only one component changes before projection. Paschos et al. [36] showed that this special structure reduces the complexity to $O(N \log N)$, later improved to $O(N)$ [35]. This linear complexity is unavoidable for an implementation that explicitly materializes the cache state vector according to Line 4 in Algorithm 1, because, even in the single-request setting, the projection may alter all components of the state vector.

2.3. OGD for Integral Caching

A natural way to implement an integral caching policy is to run a fractional policy and then round the fractional cache state into an integral one. Because the reward is linear in the cache state, sampling object n with probability y_n/s_n preserves the same regret guarantees in expectation [40].

Table 2: Overview of representative rounding schemes.

Rounding Scheme	Switching Cost	Feasibility	Heterogeneous Object Size	Time ↑
Optimal Transport [40]	optimal	exact	×	$O\left(N^3\right)$
Madow [34, 40]	$O\left(CN\sum_{t=1}^{T-1}\left\|\boldsymbol{y}_{t+1}-\boldsymbol{y}_t\right\|_1\right)$	exact	×	$O(N)$
Poisson + OGD [15]	$O\left(\sum_{t=1}^{T-1}\left\|\boldsymbol{y}_{t+1}-\boldsymbol{y}_t\right\|_1\right)$	in expectation	✓	$O^{\dagger}(\log N)$
Poisson + qOGD §4.3	$O\left(\sum_{t=1}^{T-1}\left\|\boldsymbol{y}_{t+1}-\boldsymbol{y}_t\right\|_1\right)$	in expectation	✓	$O^{\dagger}(1)$

Madow sampling is a systematic sampling method with unequal inclusion probabilities and fixed sample size: it draws a single random offset and selects objects at unit-spaced positions along the cumulative inclusion-probability scale [26]. In the uniform-object-size setting, when the cache capacity C corresponds to an integer number of objects, Madow sampling can therefore be used to round a fractional state into a feasible integral cache state with exactly C stored objects [34, 40]. Its computational complexity is $O(N)$. However, Madow sampling does not guarantee the minimization of the *switching cost* (Definition 1), which, in the uniform-object-size setting, corresponds to the number of new objects inserted into the cache. To address this issue, Si Salem et al. [40] proposed a rounding scheme based on optimal transport, at the expense of a substantially higher computational cost of $O\left(N^3\right)$.

In the heterogeneous-object-size setting, no rounding scheme can, in general, both preserve the marginal inclusion probabilities y_n/s_n and always output feasible integral cache states. Appendix A illustrates this claim by exhibiting a feasible fractional cache state whose prescribed marginals are incompatible with any distribution over feasible integral cache states. Existing work addresses this issue by relaxing the feasibility requirement and imposing the capacity constraint only in expectation, which is often referred to as a soft cache constraint. This model has been studied in prior analytical work [17, 7, 31, 14, 4], and practical cache management schemes often allow temporary violations of the cache capacity [41].

Poisson rounding, namely, the independent sampling of each object with probability y_n/s_n, satisfies the soft cache constraint. Carra et al. [15] showed that, when the sampling is correlated over time via permanent random numbers, OGD with Poisson rounding achieves an amortized complexity of $O(\log N)$ while incurring small switching costs.

Table 2 summarizes some representative rounding schemes, with an assessment of their switching costs, feasibility, and complexity. Efficient rounding implementations may update the integral cache state incrementally, using not only the current fractional state $\boldsymbol{y}_{t+1}$, but also the integral state obtained from the previous fractional state $\boldsymbol{y}_t$. In this case, the resulting complexity is not a property of the rounding scheme alone: it also depends on how the underlying fractional policy updates $\boldsymbol{y}_t$, for instance how many components can change after a request. Whenever this dependence is essential, Table 2 reports the corresponding fractional-state update policy together with the rounding scheme. The table reports that Poisson rounding, when combined with qOGD—the quantized version of OGD proposed in this paper—achieves constant amortized time per request.

2.4. Motivation

Traditional caching algorithms, such as LRU and LFU, are widely deployed because they require only a constant number of operations per request for data-structure updates, admission, and eviction; accordingly, their per-request complexity is $O(1)$. By contrast, existing no-regret caching policies are computationally more demanding. Table 3 summarizes representative works, together with their regret guarantees and per-request complexity. In the integral setting, most works have a per-request time complexity of at least $O(\log N)$, while in the fractional setting it is often $O(N)$. Although logarithmic complexity may correspond to only a few operations per request, it can still be prohibitive for caches that must serve very large catalogs at line rate.

For each algorithm, Table 3 also indicates whether it can handle heterogeneous object sizes and whether it provides dynamic regret guarantees. Support for heterogeneous sizes is particularly important in applications such as web caching and Content Delivery Networks (CDNs), where objects are inherently non-uniform; policies restricted to uniform-size objects are therefore not directly applicable. Finally, most existing works provide regret guarantees only in the static setting, where the time horizon is known in advance and performance is measured against a single best

Table 3: Overview of online-learning-based caching algorithms. "Static Regret" and "Time/Request" columns assume uniform object sizes i.e. $s = \mathbf{1}$. Static regret refers to expected worst-case static regret in the number of hits. The "Utility Weights" column indicates whether the regret guarantees allow cache hits to have heterogeneous utility weights, rather than assigning the same reward to every hit. Note that OGB [15] and qOGDP §4.3 are implemented with Poisson rounding and therefore only achieve feasibility in expectation.

Caching Policy		Static Regret	Time/Request ↑	Dynamic Regret	Heterogeneous Object Size	Utility Weights	Fractional/ Integral
MW	[18]	$O\left(\sqrt{CT \log N}\right)$	$O(NC)$	×	×	×	Integral
OGD	[36]	$O\left(\sqrt{CT}\right)$	$O(N \log N)$	×	×	✓	Fractional
FTPL	[11]	$O\left((\log N)^{1/4} \sqrt{CT}\right)$	$O(N \log N)$	×	×	×	Integral
OMD_{NE}	[40]	$O\left(C \sqrt{T \log N}\right)$	$O(N)$	×	×	✓	Fractional
NOC	[50]	$O\left(\sqrt{CT}\right)$	$O(N)$	✓	×	✓	Fractional
SAcache	[38]	$O\left((\log N)^{1/4} \sqrt{CT}\right)$	$O(\log C)$	×	×	×	Integral
L-NFPL	[9]	$O\left(\sqrt{CT}\right)$	$O^{\dagger}\left(1 + \log C \sqrt{C/T}\right)$	×	×	✓	Integral
OFTRL_{L2}	[29]	$O\left(\sqrt{CT}\right)$	$O\left(N^3\right)$	×	✓	✓	Fractional
OFTPL	[29]	$O\left((\log N)^{1/4} \sqrt{CT}\right)$	$O(N \log N)$	×	✓	✓	Integral
OGB	[15]	$O\left(\sqrt{CT}\right)$	$O^{\dagger}(\log N)$	×	✓	✓	Integral
qOGD	§4.2	$O\left(\sqrt{CT}\right)$	$O^{\dagger}(1)$	✓	✓	✓	Fractional
qOGDP	§4.3	$O\left(\sqrt{CT}\right)$	$O^{\dagger}(1)$	✓	✓	✓	Integral

static cache allocation chosen in hindsight. To the best of our knowledge, only [50] considers a dynamic comparator and establishes dynamic regret guarantees, albeit with $O(N)$ computational complexity.

Our work is motivated by the search for a caching policy that provides regret guarantees while remaining computationally efficient and simple. Our proposed solution, qOGD, achieves amortized constant time per request in both the fractional and integral settings, while also supporting heterogeneous object sizes. In addition, we establish dynamic regret guarantees for qOGD.

Our scheme also implicitly addresses an aspect that is often overlooked, namely finite numerical precision. Existing OGD-based schemes assume either that each object can be divided into arbitrarily small portions that can be added to or removed from the cache in the fractional setting, or that the probability y_n/s_n associated with each object can vary by an arbitrarily small amount in the integral setting. In practical implementations, however, numerical values are represented with finite precision, which may introduce errors not captured by the theoretical analysis. Our quantized approach resolves this issue by restricting object portions in the fractional case, and object probabilities in the integral case, to a finite set of values, effectively yielding an algorithm based on the more robust use of integer arithmetic. Note that, in this quantized setting, "fractional caching" still refers to the ability to store fractions of an object, but no longer arbitrary fractions.

3. Beyond Euclidean Projection: Dynamic Regret for OGD with Relaxed Projections

As discussed in Section 2.2, the projection step is one of the main computational bottlenecks of OGD for caching. This motivates the search for cheaper alternatives. Such alternatives, however, cannot be chosen arbitrarily: the regret bound in OGD relies on specific properties of the Euclidean projection, and changing the projection may alter the analysis or even lead to linear regret. In that case, the average regret would not vanish, and the policy could maintain a constant asymptotic gap relative to the comparator. We therefore investigate a broader class of projection operators that still guarantee sublinear regret, possibly with slightly worse constants than those obtained with the Euclidean projection. This broader class allows us to choose, for our algorithm, a projection that leads to a computationally efficient caching policy. In the following, we introduce *relaxed projections* and formulate the dynamic regret guarantees for OGD with relaxed projection.

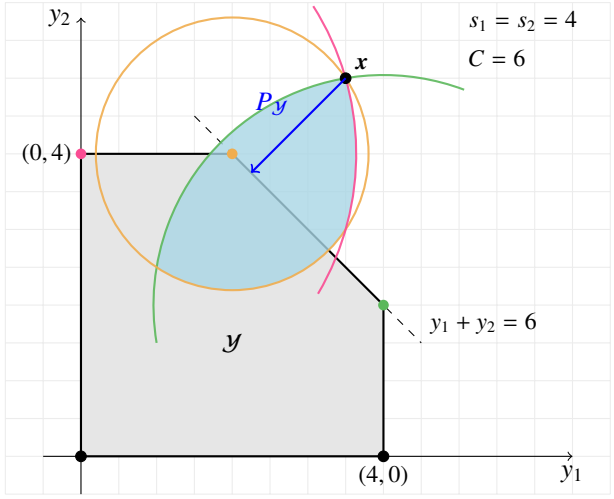


Figure 1: Unlike the Euclidean projection $P_{\mathcal{Y}}$ that maps $\boldsymbol{x}$ to a point on the capped simplex, the relaxed projection $P^R_{\mathcal{Y}}(\boldsymbol{x})$ lives in the blue area: any point in the blue area is closer to any point in $\mathcal{Y}$ than $\boldsymbol{x}$ is.

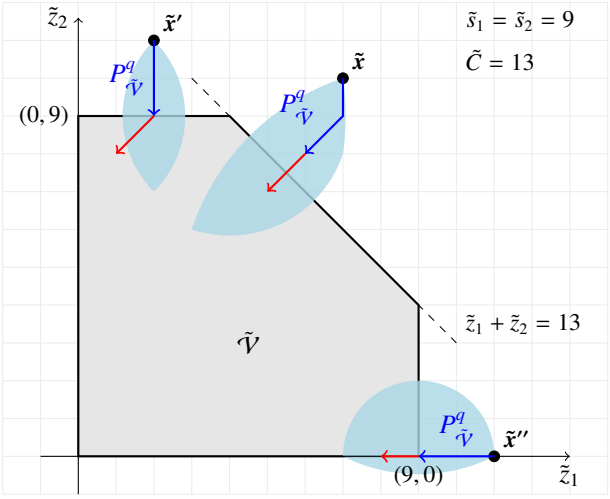


Figure 2: Quantized projection $P^q_{\tilde{\mathcal{V}}}$ in rescaled coordinates with $\varepsilon = 0.5$. Notice that $P^q_{\tilde{\mathcal{V}}}$ maps $\tilde{\boldsymbol{x}}$, $\tilde{\boldsymbol{x}}'$ or $\tilde{\boldsymbol{x}}''$ from the grid to the grid, after which a feasibility adjustment (the red arrows) brings the points inside $\tilde{\mathcal{V}}$.

Definition 4 (Relaxed Projection) $P^R_{\mathcal{V}} : \mathbb{R}^N \to \mathbb{R}^N$ is a relaxed projection onto $\mathcal{V} \subset \mathbb{R}^N$ if and only if for any $\boldsymbol{x} \in \mathbb{R}^N$ and any $\boldsymbol{z} \in \mathcal{V}$, $\left\| P^R_{\mathcal{V}}(\boldsymbol{x}) - \boldsymbol{z} \right\| \leq \|\boldsymbol{x} - \boldsymbol{z}\|$.

In practice, a relaxed projection maps the perturbed state to another state that is closer to all points in $\mathcal{V}$, but does not necessarily belong to $\mathcal{V}$. As an example, Fig. 1 shows a two-dimensional capped simplex and an external point $\boldsymbol{x}$. The Euclidean projection is represented by the straight line, while the blue area represents all possible realizations of the relaxed projection—i.e., all points that satisfy the condition in Definition 4. Relaxed projections are very similar to quasi-nonexpansive operators [8, Chapter 4]. [22, 19] define pointwise versions of relaxed projections and refer to them as approximate or infeasible projections. Since we use the terms "feasible" and "infeasible" to describe cache states in this work, we use the term "relaxed projection" for $P^R_{\mathcal{V}}$ to avoid confusion.

We are now ready to present a result that provides the dynamic regret bound for OGD (described in Algorithm 1), but where we replace the Euclidean projection with the relaxed projection, resulting in Algorithm 2.

Theorem 1 (Dynamic regret of OGD with relaxed projections)**.** *Let the cache state $\boldsymbol{y}_t$ be determined by Algorithm 2, where the projection step uses a relaxed projection $P^R_{\mathcal{V}}$ onto $\mathcal{V}$, with utility $\boldsymbol{u}_t = w_{t,r_t}\boldsymbol{e}(r_t) \in \mathbb{R}^N$, and step size $\eta \in \mathbb{R}^+$. Let $\boldsymbol{y}_1 = 0$. Then, for any comparator $\boldsymbol{z}_t \in \mathcal{V}$:*

$$R_T\left(OGD, (\boldsymbol{z}_t)_{t=1}^T, (w_{t,r_t}\boldsymbol{e}(r_t))_{t=1}^T\right) \leq \frac{1}{2\eta}\|\boldsymbol{z}_T\|^2 + \frac{1}{\eta}\sum_{t=1}^{T-1}\langle \boldsymbol{z}_t - \boldsymbol{z}_{t+1}, \boldsymbol{y}_{t+1}\rangle + \frac{\eta}{2}\sum_{t=1}^{T} w_{t,r_t}^2. \tag{5}$$

Proof. The bound follows by applying the inequality in Definition 4 and from dropping the term $-\left\|\boldsymbol{y}_{T+1} - \boldsymbol{z}_{T+1}\right\|^2$. The detailed proof can be found in Appendix B and is based on the proofs of [16, Thm. 11.4, p. 309] and [32, Thm. 15.1 p. 208]. □

Theorem 1 indicates that the regret bound of OGD with a relaxed projection consists of a final comparator term, a path-length term and a utility term. In the particular case when the comparator is static, the path length term disappears, and we find the static regret bound.

For classical OGD, dynamic regret bounds are obtained by controlling the path-length term. Since the Euclidean projection ensures that each iterate remains in the feasible set, $\boldsymbol{y}_{t+1} \in \mathcal{Y}$, we can write

$$\sum_{t=1}^{T-1}\langle \boldsymbol{z}_t - \boldsymbol{z}_{t+1}, \boldsymbol{y}_{t+1}\rangle \leq \sum_{t=1}^{T-1}\max_{\boldsymbol{y}\in\mathcal{Y}}\langle (\boldsymbol{z}_t - \boldsymbol{z}_{t+1})^+, \boldsymbol{y}\rangle \leq \sum_{t=1}^{T-1}\max_{\boldsymbol{y}\in\mathcal{Y}}\|\boldsymbol{y}\|\,\|\boldsymbol{z}_t - \boldsymbol{z}_{t+1}\|. \tag{6}$$

As $\mathcal{Y}$ is bounded, this yields a finite bound proportional to the path length of the comparator sequence, and is therefore meaningful for comparator classes with limited variation, e.g., when $\sum_{t=1}^{T-1}\|\boldsymbol{z}_t - \boldsymbol{z}_{t+1}\|$ grows sublinearly in T. The

same argument does not immediately extend to arbitrary relaxed projections. Indeed, even if the comparator set $\mathcal{V}$ is bounded, the relaxed projection does not necessarily return a point in $\mathcal{V}$; hence the boundedness of $\mathcal{V}$ alone does not control the norm of the iterates $(\boldsymbol{y}_t)_{t=1}^T$, which can be unbounded. In our algorithm, we select a computationally efficient relaxed projection whose output remains bounded.

4. Our Algorithm: Quantized OGD (qOGD)

In this section, we introduce our caching policy, quantized OGD (qOGD). As shown in Table 3, among caching policies with regret guarantees, qOGD achieves the most favorable per-request complexity, namely amortized constant time, while supporting heterogeneous object sizes.

The computational efficiency of qOGD relies on a quantized representation of cache states. Specifically, we fix a quantization granularity $\varepsilon > 0$ and restrict the amount of each object stored in the cache to be an integer multiple of ε. This quantization may reflect the operation of real caching systems: for example, in CDNs, objects are usually stored, fetched, or transmitted as chunks of bounded size [43, 42, 20, 3]. However, this operational interpretation is not required. More generally, quantization can be viewed as an algorithmic constraint on the cache allocations produced by the policy. Besides enabling computationally efficient updates, this constraint also allows the policy to be implemented using integer arithmetic. This avoids relying on arbitrarily precise fractional states or probabilities, an assumption that is implicit in many fractional caching policies but may be problematic in finite-precision implementations.

For ease of presentation, we rescale all variables by the quantization granularity ε, and denote rescaled quantities with a tilde. These quantities are illustrated in Fig. 2 for the example in Fig. 1. In particular, we define $\tilde{\boldsymbol{s}} = \lfloor \boldsymbol{s}/\varepsilon \rfloor + \mathbf{1}$. If the size s_n of object n is not a multiple of ε, then $\tilde{s}_n$ corresponds to the number of chunks needed to store the whole object, with the last chunk having size smaller than ε. If s_n is a multiple of ε, the definition instead adds one virtual chunk. This is a bookkeeping device: as shown below, qOGD's relaxed projection may produce a state that is not feasible, and a subsequent feasibility step removes one quantization unit from each positive component. The virtual chunk ensures that objects whose size is an exact multiple of ε can still be fully stored after this feasibility step. Similarly, we define the rescaled cache capacity as $\tilde{C} = \lfloor C/\varepsilon \rfloor + 1$. Since the OGD update depends on the gradient step $\eta \boldsymbol{u}_t$, we rescale and discretize this whole increment, rather than discretizing η and $\boldsymbol{u}_t$ separately. A deterministic rounding rule, such as always rounding down or to the nearest integer, would distort different components differently depending on their fractional parts and then introduce a bias that accumulates over time. To preserve the original OGD step in expectation, we therefore apply an unbiased rounding operation $\mathrm{UR}(\cdot)$, to $\tilde{\eta}\boldsymbol{u}_t$, where $\tilde{\eta} = \eta/\varepsilon$.

Definition 5 (Unbiased Rounding) For any $\lambda \in \mathbb{R}$, we define $\mathrm{UR}(\lambda)$ as $\lfloor \lambda \rfloor + 1$ with probability $\lambda - \lfloor \lambda \rfloor$ and $\lfloor \lambda \rfloor$ otherwise.

It is easy to check that $\mathrm{UR}(\cdot)$ is unbiased ($\mathbb{E}[\mathrm{UR}(\lambda)] = \lambda$) and has bounded variance ($\mathrm{Var}[\mathrm{UR}(\lambda)] \leq \frac{1}{4}$).

In the following, we first introduce a quantized projection that combines a relaxed projection with a feasibility adjustment (Section 4.1). We then use such a projection to define qOGD, first for fractional caching (Section 4.2) and then for integral caching (Section 4.3).

4.1. Quantized Projection

The key element in qOGD is the *quantized projection*, $P^q_{\tilde{\mathcal{V}}}$, over the scaled comparator set $\tilde{\mathcal{V}}$, which is defined as follows:

$$\tilde{\mathcal{V}} := \left\{ \tilde{\boldsymbol{z}} \in \mathbb{R}^N \;\middle|\; \sum_{n=1}^{N} \tilde{z}_n \leq \tilde{C} \text{ and } \mathbf{0} \leq \tilde{\boldsymbol{z}} \leq \tilde{\boldsymbol{s}} \right\}. \tag{7}$$

Observe that $\mathcal{Y} \subset \varepsilon\tilde{\mathcal{V}}$.[1]

Algorithm 3 outlines the quantized projection $P^q_{\tilde{\mathcal{V}}}$, which consists of two steps. First, the point is clipped to satisfy the box constraints of $\tilde{\mathcal{V}}$ (Line 1). Then, if needed, one quantization unit is repeatedly removed from every positive

[1] We could have denoted this set as $\tilde{\mathcal{Y}}$, because it can be seen as a rescaled version of $\mathcal{Y}$, but the notation $\tilde{\mathcal{V}}$ stresses that this set plays the role of the comparator set in Theorem 1.

Algorithm 3: Quantized projection $P^q_{\tilde{\mathcal{V}}}$

input: $\tilde{\boldsymbol{x}} \in \mathbb{N}^N$ ▷ point to be projected

1 $\tilde{\boldsymbol{x}} \leftarrow \min\{(\tilde{\boldsymbol{x}})^+, \tilde{\boldsymbol{s}}\}$

2 **while** $\sum_{n=1}^N (\tilde{x}_n - 1)^+ \geq \tilde{C}$ **do**

3 $\quad \tilde{\boldsymbol{x}} \leftarrow (\tilde{\boldsymbol{x}} - \mathbf{1})^+$

4 **return** $\tilde{\boldsymbol{x}}$

Algorithm 4: qOGD

1 choose $\varepsilon, \eta \in \mathbb{R}^+$ and set $\boldsymbol{y}_1 = \tilde{\boldsymbol{f}}_1 \leftarrow \mathbf{0}$

2 **for** *round* $t \leftarrow 1$ **to** T **do**

3 $\quad$ get r_t, w_{t,r_t}

4 $\quad \tilde{\boldsymbol{f}}_{t+1} \leftarrow \tilde{\boldsymbol{f}}_t + \mathrm{UR}(\tilde{\eta} w_{t,r_t})\, \boldsymbol{e}(r_t)$ ▷ gradient step

5 $\quad \tilde{\boldsymbol{f}}_{t+1} \leftarrow P^q_{\tilde{\mathcal{V}}}\left(\tilde{\boldsymbol{f}}_{t+1}\right)$ ▷ projection step

6 $\quad \boldsymbol{y}_{t+1} \leftarrow \varepsilon\left(\tilde{\boldsymbol{f}}_{t+1} - \mathbf{1}\right)^+$ ▷ feasibility step

Algorithm 5: qOGD optimized

1 choose $\varepsilon, \eta \in \mathbb{R}^+$, set $\boldsymbol{y} = \tilde{\boldsymbol{f}} \leftarrow \mathbf{0}, \tilde{\kappa} \leftarrow 0, \mathcal{H} = \emptyset$

2 **for** *round* $t \leftarrow 1$ **to** T **do**

3 $\quad$ get r_t, w_{t,r_t}

4 $\quad \tilde{\kappa} \leftarrow \tilde{\kappa} - \left(\tilde{f}_{r_t} - 1\right)^+$

5 $\quad \tilde{f}_{r_t} \leftarrow \min\left\{\tilde{f}_{r_t} + \mathrm{UR}\left(\tilde{\eta} w_{t,r_t}\right), \tilde{s}_{r_t}\right\}$ ▷ update $\mathcal{H}$

6 $\quad \tilde{\kappa} \leftarrow \tilde{\kappa} + \left(\tilde{f}_{r_t} - 1\right)^+$

7 $\quad y_{r_t} \leftarrow \varepsilon\left(\tilde{f}_{r_t} - 1\right)^+$ ▷ update cache state

8 $\quad$ **while** $\tilde{\kappa} \geq \tilde{C}$ **do**

9 $\quad\quad$ **for** $n \in \mathcal{H}$ $\left(= \{n \mid \tilde{f}_n > 0\}\right)$ **do**

10 $\quad\quad\quad \tilde{\kappa} \leftarrow \tilde{\kappa} - \left(\tilde{f}_n - 1\right)^+$

11 $\quad\quad\quad \tilde{f}_n \leftarrow \tilde{f}_n - 1$ ▷ update $\mathcal{H}$

12 $\quad\quad\quad \tilde{\kappa} \leftarrow \tilde{\kappa} + \left(\tilde{f}_n - 1\right)^+$

13 $\quad\quad\quad y_n \leftarrow \varepsilon\left(\tilde{f}_n - 1\right)^+$ ▷ update cache state

component (Line 3) until the stopping condition in Line 2 is satisfied. The procedure is well defined and halts in finite time for all $\tilde{C} > 0$. Indeed, while the loop condition holds, the vector $\tilde{\boldsymbol{x}}$ has at least one positive component, and the update $\tilde{\boldsymbol{x}} \leftarrow (\tilde{\boldsymbol{x}} - \mathbf{1})^+$ strictly decreases $\sum_{n=1}^N \tilde{x}_n$. Since all operations preserve integer-valued coordinates, $P^q_{\tilde{\mathcal{V}}}$ is indeed quantized. Fig. 2 provides some examples of quantized projections.

The key property of the quantized projection is that it preserves the relaxed-projection inequality required by Theorem 1. We state this property formally below.

Theorem 2 (Quantized Relaxed Projection)**.** *The output $P^q_{\tilde{\mathcal{V}}}$ of Algorithm 3 is a relaxed projection onto $\tilde{\mathcal{V}}$.*

The proof, given in Appendix C, shows that $P^q_{\tilde{\mathcal{V}}}$ can be decomposed into elementary operations, each of which is a relaxed projection with respect to $\tilde{\mathcal{V}}$. The result then follows by composition.

As illustrated by the examples in Fig. 2, the output of the quantized projection $P^q_{\tilde{\mathcal{V}}}$ does not necessarily belong to $\tilde{\mathcal{V}}$. Moreover, even if it did, simply rescaling it by ε would not be sufficient to obtain a feasible cache allocation, because $\mathcal{Y} \subset \varepsilon\tilde{\mathcal{V}}$: the scaled set allows one extra quantization unit (one extra chunk) in the capacity and object-size constraints. As a result, the corresponding cache allocation may violate feasibility, for example by exceeding the cache capacity. We therefore introduce an additional feasibility adjustment.

Proposition 1 (Feasibility adjustment) For any $\tilde{\boldsymbol{x}} \in \mathbb{N}^N$, $\varepsilon(P^q_{\tilde{\mathcal{V}}}(\tilde{\boldsymbol{x}}) - \mathbf{1})^+$ belongs to $\mathcal{Y}$.

Proof. For ease of notation, let $\tilde{\boldsymbol{f}} = P^q_{\tilde{\mathcal{V}}}(\tilde{\boldsymbol{x}})$ denote the output of Algorithm 3. Feasibility is achieved if both the box constraints and the cache capacity constraint of $\mathcal{Y}$ are satisfied (see (1)). Line 1 of Algorithm 3 ensures $\mathbf{0} \leq \tilde{\boldsymbol{x}} \leq \tilde{\boldsymbol{s}}$. This condition is maintained in Line 3 after any update $\tilde{\boldsymbol{x}} \leftarrow (\tilde{\boldsymbol{x}} - \mathbf{1})^+$. Therefore $\mathbf{0} \leq \tilde{\boldsymbol{f}} \leq \tilde{\boldsymbol{s}}$ and $\mathbf{0} \leq \varepsilon(\tilde{\boldsymbol{f}} - \mathbf{1})^+ \leq \varepsilon(\tilde{\boldsymbol{s}} - \mathbf{1})^+ \leq \boldsymbol{s}$ (recall that $\tilde{\boldsymbol{s}} = \lfloor \boldsymbol{s}/\varepsilon \rfloor + \mathbf{1}$). The proposed adjustment then satisfies the box constraints. The output of Algorithm 3 satisfies the exit condition of the while loop in Line 2, namely, $\sum_{n=1}^N (\tilde{f}_n - 1)^+ < \tilde{C}$. Since the sum is an integer and $\tilde{C} = \lfloor C/\varepsilon \rfloor + 1$, we have: $\sum_{n=1}^N (\tilde{f}_n - 1)^+ \leq \tilde{C} - 1 \leq \frac{C}{\varepsilon}$. Multiplying both sides of the inequality by ε yields $\sum_{n=1}^N \varepsilon(\tilde{f}_n - 1)^+ \leq C$. Thus, the proposed adjustment satisfies the cache capacity constraint. Having both conditions satisfied implies that $\varepsilon(\tilde{\boldsymbol{f}} - \mathbf{1})^+ \in \mathcal{Y}$, completing the proof. □

4.2. Fractional qOGD

qOGD is described in Algorithm 4. The algorithm has a simple structure: at each request, it combines the basic steps described above, namely the quantized gradient update, the quantized projection and the feasibility adjustment. In the following, we establish qOGD's guarantees in terms of the switching cost and regret.

Theorem 3 (Switching cost of qOGD)**.** *The switching cost of qOGD (Algorithm 4), with granularity $\varepsilon \in \mathbb{R}^+$ and step size $\eta \in \mathbb{R}^+$, satisfies*

$$S_T(qOGD) \le \eta \sum_{t=1}^{T-1} w_{t,r_t}. \tag{8}$$

Proof. Because the cache starts empty ($\boldsymbol{y}_1 = \mathbf{0}$), the total amount of data evicted over time cannot exceed the total amount of data inserted. Thus, the switching cost (Definition 1) is bounded by the expected amount of data added. We can bound the increase of $\boldsymbol{y}_t$ as follows:

$$\mathbb{E}\left[\sum_{n=1}^{N} (y_{t+1,n} - y_{t,n})^+\right] \le \varepsilon \mathbb{E}\left[\sum_{n=1}^{N} \left(\tilde{f}_{t+1,n} - \tilde{f}_{t,n}\right)^+\right] \le \varepsilon \mathbb{E}\left[\mathrm{UR}\left(\tilde{\eta} w_{t,r_t}\right)\right] = \eta w_{t,r_t}, \tag{9}$$

where the first inequality uses $((a)^+ - (b)^+)^+ \le (a-b)^+$, the second one follows from Line 4, and the final equality uses the unbiasedness of UR$(\cdot)$. Summing this bound over all requests completes the proof. □

Theorem 4 (Dynamic regret of qOGD)**.** *The regret of qOGD (Algorithm 4) with granularity $\varepsilon \in \mathbb{R}^+$ and step size $\eta \in \mathbb{R}^+$, satisfies*

$$R_T\left(qOGD, (\boldsymbol{z}_t)_{t=1}^T, (w_{t,r_t}\boldsymbol{e}(r_t))_{t=1}^T\right) \le \frac{1}{2\eta}\|\boldsymbol{z}_T\|^2 + \frac{1}{\eta}\sum_{t=1}^{T-1} \max_{\boldsymbol{y}\in\mathcal{Y}} \langle (\boldsymbol{z}_t - \boldsymbol{z}_{t+1})^+, \boldsymbol{y} + \varepsilon\mathbf{1}\rangle + \frac{\eta}{2}\sum_{t=1}^{T} w_{t,r_t}^2 + \varepsilon\sum_{t=1}^{T}\left(w_{t,r_t} + \frac{\varepsilon}{8\eta}\right), \tag{10}$$

for any comparator policy $\boldsymbol{z}_t \in \varepsilon\tilde{\mathcal{V}}$.

Sketch of the proof. The sequence $(\tilde{\boldsymbol{f}}_t)_{t=1}^T$ is generated by an OGD with relaxed projection $P^q_{\tilde{\mathcal{V}}}$ and utility $\tilde{\boldsymbol{u}}_t = \frac{\mathrm{UR}(\tilde{\eta} w_{t,r_t})}{\tilde{\eta}}\boldsymbol{e}(r_t)$ instead of $\boldsymbol{u}_t = w_{t,r_t}\boldsymbol{e}(r_t)$. Therefore, Theorem 1 applies with $\mathcal{V} = \tilde{\mathcal{V}}$. Rescaling the resulting bound and accounting for the feasibility adjustment $\boldsymbol{y}_t = \varepsilon(\tilde{\boldsymbol{f}}_t - 1)^+$ yields (10). For the full derivation, see Appendix D. □

The bound makes explicit the price of quantization. As expected, for a fixed comparator, a coarser quantization, corresponding to a larger ε, increases the regret bound. However, a larger ε also expands the comparator set $\varepsilon\tilde{\mathcal{V}}$, which may contain a better-performing comparator. Formally, as $\varepsilon \to 0$, the additional quantization terms vanish, and we recover the regret bound of Theorem 1 after upper-bounding the latter's path-length term using the first inequality in (6). For $\varepsilon > 0$, quantization affects the bound in two ways. First, it slightly enlarges the path-length term, replacing $\boldsymbol{y}$ by $\boldsymbol{y} + \varepsilon\mathbf{1}$. This effect is limited when the granularity is small relative to the scale of the cache allocations and, as we commented for (6), for comparator classes with limited variation, e.g., when $\sum_{t=1}^{T-1}\|\boldsymbol{z}_t - \boldsymbol{z}_{t+1}\|$ grows sublinearly in T, the path-length terms grows sublinearly. The main additional contribution is the final term, which is absent from the non-quantized OGD bound. We observe that η should not be chosen much smaller than ε, since most gradient increments would otherwise be rounded to zero. For $\varepsilon = O(\eta)$, the final term scales as $O(\varepsilon T)$: for fixed granularity this yields a linear regret, whereas choosing ε to decrease with T makes this contribution sublinear.

In particular, by choosing ε appropriately, the static regret bound of qOGD recovers the same $O\left(\sqrt{CT}\right)$ dependence as OGD with Euclidean projection. Similarly, the asymptotically optimal regret against S-switch comparators can be recovered (see Appendix E).

Corollary 1 (Static regret of qOGD)**.** *Let the cache state $\mathbf{y}_t$ be generated by Algorithm 4 with scaled step size $\tilde{\eta} \in \mathbb{R}^+$ and the granularity $\varepsilon = \sqrt{\frac{s_{max}C}{2\tilde{\eta}T\left(\frac{\tilde{\eta}w_{max}^2}{2} + w_{max} + \frac{1}{8\tilde{\eta}}\right)}}$, where $w_{max} \ge \max_{t\in[T]} w_{t,r_t}$ and $s_{max} \ge \max_n s_n$. Then, for any fixed comparator policy $\boldsymbol{z} \in \mathcal{Y}$:*

$$R_T\left(qOGD, (\boldsymbol{z})_{t=1}^T, (w_{t,r_t}\boldsymbol{e}(r_t))_{t=1}^T\right) \le \sqrt{\left(w_{max}^2 + \frac{2w_{max}}{\tilde{\eta}} + \frac{1}{4\tilde{\eta}^2}\right) s_{max}CT}. \tag{11}$$

Proof. We apply Theorem 4. Since the comparator is fixed, the path-length term vanishes. Moreover, $w_{t,r_t} \le w_{\max}$ for all t. For any comparator $\boldsymbol{z} \in \mathcal{Y} \subseteq \left\{\boldsymbol{y} \in [0, s_{\max}]^N \;\middle|\; \sum_{n=1}^N \frac{y_n}{s_{\max}} \le C\right\}$, we have $\|\boldsymbol{z}_T\|^2 = \sum_{n=1}^N z_{T,n}^2 \le \sum_{n=1}^N s_{\max} z_{T,n} \le s_{\max}C$ [36]. Substituting these bounds into Theorem 4 yields the following:

$$R_T\left(\mathrm{qOGD}, (z)_{t=1}^T, (w_{t,r_t}\boldsymbol{e}(r_t))_{t=1}^T\right) \le \frac{s_{\max}C}{2\eta} + \frac{\eta T w_{\max}^2}{2} + \varepsilon T w_{\max} + T\frac{\varepsilon^2}{8\eta} = \frac{s_{\max}C}{2\varepsilon\tilde{\eta}} + \varepsilon T\left(\frac{\tilde{\eta}w_{\max}^2}{2} + w_{\max} + \frac{1}{8\tilde{\eta}}\right). \quad (12)$$

The right-hand side is minimized by choosing $\varepsilon = \sqrt{\frac{s_{\max}C}{2\tilde{\eta}T\left(\frac{\tilde{\eta}w_{\max}^2}{2}+w_{\max}+\frac{1}{8\tilde{\eta}}\right)}}$. Substituting this value into (12) gives the final bound in (11). □

Corollary 1 shows that qOGD has sublinear static regret in the time horizon T. Hence, its average regret vanishes, and its average per-request reward asymptotically matches that of the best static cache allocation.

Remark 1 The term $\varepsilon/(8\eta)$ in Theorem 4 comes from bounding the variance introduced by unbiased rounding. When $w_{t,r_t} = 1$, so that the reward is the amount of data served locally, and $\tilde{\eta} \in \mathbb{N}$, the increment $\tilde{\eta}w_{t,r_t}$ is already integer-valued and $\mathrm{UR}(\tilde{\eta}w_{t,r_t}) = \tilde{\eta}w_{t,r_t}$; in this case, the variance term disappears. As a consequence, in the same setting, the static regret in Corollary 1 can be bounded by $\sqrt{\left(w_{\max}^2 + \frac{2w_{\max}}{\tilde{\eta}}\right)s_{\max}CT}$, when $\varepsilon = \sqrt{\frac{s_{\max}C}{2\tilde{\eta}T\left(\frac{\tilde{\eta}w_{\max}^2}{2}+w_{\max}\right)}}$.

Optimized Implementation of qOGD

Algorithm 4 gives a compact representation of qOGD, which is convenient for defining the policy and proving its regret guarantees. However, it is not directly an efficient implementation. We provide it in Algorithm 5 by using sparse dictionary-based representations of the nonzero components of $\tilde{\boldsymbol{f}}$ and $\boldsymbol{y}$, which are updated in place. To avoid an $O(N)$ scan for checking the while-loop condition in Line 2 of Algorithm 3, we maintain the quantity $\sum_{n=1}^N(\tilde{f}_n - 1)^+$ incrementally through the variable $\tilde{\kappa}$, which represents the current cache occupancy in rescaled units. As observed in prior work [36, 15], each request induces a sparse gradient step that can increase only the component associated with the requested object r_t. Therefore, after the gradient step, only this component may violate the box constraint and needs to be clipped.

Theorem 5 (Equivalence)**.** *Algorithm 5, with granularity $\varepsilon \in \mathbb{R}^+$ and step size $\tilde{\eta} \in \mathbb{R}^+$, generates the same cache state as Algorithm 4 when the same rounding decisions are made.*

Proof. Algorithm 5 only needs to enforce the upperbound constraint $\tilde{\boldsymbol{f}} \le \tilde{\boldsymbol{s}}$ on $\tilde{f}_{r_t}$, since all other components already satisfy the constraint, either from the previous iteration or, at $t = 1$, by initialization. Tracking $\sum_{n=1}^N(\tilde{f}_n - 1)^+$ when an element changes can be achieved by changing the contribution of this element to $\tilde{\kappa}$. Because $\tilde{\boldsymbol{f}} \in \mathbb{N}^N$ is quantized, we can replace $\tilde{\boldsymbol{f}} \leftarrow (\tilde{\boldsymbol{f}} - \mathbf{1})^+$ with $\tilde{f}_n \leftarrow \tilde{f}_n - 1$ if $\tilde{f}_n > 0$. □

By Theorem 5, the optimized implementation in Algorithm 5 inherits the dynamic regret guarantee of Theorem 4 and, in particular, the $O\left(\sqrt{CT}\right)$ static regret bound of Corollary 1. The following theorem shows that, when the utility weights w_{t,r_t} are uniformly bounded, Algorithm 5 runs in $O(T)$ total time, and therefore has constant amortized time per request.

Theorem 6 (Time Complexity of qOGD)**.** *Algorithm 5, with granularity $\varepsilon \in \mathbb{R}^+$ and step size $\tilde{\eta} \in \mathbb{R}^+$, runs in $O\left(\sum_{t=1}^T \lceil\tilde{\eta}w_{t,r_t}\rceil\right)$ total time.*

Proof. The set $\mathcal{H} = \{n \mid \tilde{f}_n > 0\}$ is often inherently maintained by the cache to look up requests, in which case maintaining $\mathcal{H}$ comes at no additional cost. If this is not the case, $\mathcal{H}$ can be implemented as a separate hash table that also stores $\tilde{f}_n$ and is updated in $O(1)$ time whenever $\tilde{f}_n$ is modified.

In each round t, Lines 3 to 7 update $\tilde{f}_{r_t}$, $\tilde{\kappa}$, and y_{r_t}, taking $O(1)$ time, yielding and hence $O(T)$ over all rounds. Inside the while-loop, the algorithm iterates over $\mathcal{H}$ rather than scanning the full catalog of size N. To bound the total work across all while-loops, consider the potential $\Phi = \sum_{n=1}^N \tilde{f}_n = \sum_{n\in\mathcal{H}} \tilde{f}_n \ge 0$. Each iteration of the inner for loop (Lines 10 to 13) for a given $n \in \mathcal{H}$ takes $O(1)$ time and decrements $\tilde{f}_n$ by 1, thereby decreasing Φ by exactly 1. Since this is the only operation that decreases Φ, and since Φ starts at zero and remains nonnegative, the total number of such decrements cannot exceed the cumulative increase of Φ. The only operation that increases Φ is the gradient step at Line 5. At round t, this step increases Φ by at most $\mathrm{UR}(\tilde{\eta}w_{t,r_t}) \le \lceil\tilde{\eta}w_{t,r_t}\rceil$. Therefore, the total work performed across all while-loops is at most $O(\sum_{t=1}^T\lceil\tilde{\eta}w_{t,r_t}\rceil)$. Combining this with the $O(T)$ work outside the while loop yields an overall runtime of $O(\sum_{t=1}^T\lceil\tilde{\eta}w_{t,r_t}\rceil)$. □

4.3. Integral qOGD or Quantized OGD with Poisson Rounding (qOGDP)

In this section, we extend qOGD to integral caching by running the fractional qOGD policy from Section 4.2 and applying Poisson rounding to obtain an integral cache state, which we denote by $\boldsymbol{Y}_t$, while keeping $\boldsymbol{y}_t$ for the underlying fractional state. We formally define Poisson rounding next and refer to the resulting policy as Quantized OGD with Poisson Rounding (qOGDP).

Definition 6 (Poisson rounding function (PR)) Given a cache state $\boldsymbol{y} \in \mathbb{R}^N$, we define its Poisson rounding function based on object sizes $\boldsymbol{s} \in \mathbb{R}^N$, with $\boldsymbol{s} \geq \boldsymbol{0}$, and $\boldsymbol{V} \in [0,1]^N$, with $V_n \sim \text{Uniform}(0,1)$, as follows:

$$\forall n \in [N]: \qquad \text{PR}(\boldsymbol{y}, \boldsymbol{V}, \boldsymbol{s})_n := \begin{cases} s_n, & \text{if } V_n \leq \frac{y_n}{s_n}, \\ 0, & \text{otherwise.} \end{cases} \tag{13}$$

qOGDP can be implemented by sampling the random vector $\boldsymbol{V}$ once at initialization and then applying the Poisson rounding step $\boldsymbol{Y}_t = \text{PR}(\boldsymbol{y}_t, \boldsymbol{V}, \boldsymbol{s})$ immediately after the feasibility adjustment in Algorithm 4. A constant-amortized-time implementation is obtained by modifying the optimized fractional implementation in Algorithm 5. Instead of updating the fractional cache state through $y_n \leftarrow \varepsilon(\tilde{f}_n - 1)^+$ in Lines 7 and 13, we update y_n with the corresponding Poisson-rounded value, namely $y_n \leftarrow s_n \mathbf{1}\{V_n \leq \varepsilon(\tilde{f}_n - 1)^+ / s_n\}$. This operation depends only on the component currently being updated and therefore preserves the time-complexity analysis of Algorithm 5. Moreover, the random thresholds V_n do not need to be stored explicitly: each V_n can be generated on demand by hashing the object identifiers.

We now present some properties of qOGDP. As discussed in Section 2.3, the first three properties are standard consequences of applying Poisson rounding to a fractional caching policy; we state them here for completeness.

Theorem 7 (Properties of Poisson rounding)**.** *Let $\boldsymbol{y}_t$ be generated by Algorithm 4. Let $\boldsymbol{Y}_t = PR(\boldsymbol{y}_t, \boldsymbol{V}, \boldsymbol{s})$, with $V_n \sim Uniform(0,1)$ potentially dependent across components and independent of all other randomness. Then, the following properties hold:*

1. *Unbiasedness:* $\mathbb{E}[\boldsymbol{Y}_t \mid \boldsymbol{y}_t] = \boldsymbol{y}_t$,
2. *Regret is preserved:* $\mathbb{E}\left[\sum_{t=1}^{T} \langle \boldsymbol{u}_t, \boldsymbol{z}_t - \boldsymbol{Y}_t \rangle\right] = \mathbb{E}\left[\sum_{t=1}^{T} \langle \boldsymbol{u}_t, \boldsymbol{z}_t - \boldsymbol{y}_t \rangle\right]$.
3. *Feasibility on average:* $\boldsymbol{0} \leq \boldsymbol{Y}_t \leq \boldsymbol{s}$,
$$\mathbb{E}\left[\sum_{n=1}^{N} Y_{t,n} \,\middle|\, \boldsymbol{y}_t\right] \leq C. \tag{14}$$
4. *High-probability occupancy control: if the random variables $(V_n)_{n=1}^N$ are independent, then for any $\delta > 0$:*
$$P\left(\sum_{n=1}^{N} Y_{t,n} \geq (1+\delta)C\right) \leq \exp\left(-\frac{3C}{2\|\boldsymbol{s}\|_\infty} \frac{\delta^2}{3+\delta}\right).$$
5. *Switching cost is preserved:* $\mathbb{E}\left[(Y_{t,n} - Y_{t+1,n})^+ \mid y_{t,n}, y_{t+1,n}\right] = (y_{t,n} - y_{t+1,n})^+$,
$\mathbb{E}\left[(Y_{t+1,n} - Y_{t,n})^+ \mid y_{t,n}, y_{t+1,n}\right] = (y_{t+1,n} - y_{t,n})^+$.
6. *Minimal switching cost: for any unbiased rounding scheme,* $\mathbb{E}\left[\|\boldsymbol{Y}_{t+1} - \boldsymbol{Y}_t\|_1 \mid \boldsymbol{y}_{t+1}, \boldsymbol{y}_t\right] \geq \left\|\boldsymbol{y}_{t+1} - \boldsymbol{y}_t\right\|_1$.

Proof. The first three properties are well known and we prove them in Appendix F.

4. High-probability occupancy control: As the components V_n are independent, so are the components of $\boldsymbol{Y}_t$ when conditioned on $\boldsymbol{y}_t$. For any n, $Y_{t,n} - \mathbb{E}[Y_{t,n} \,|\, \boldsymbol{y}_t]$ is a zero-mean conditional random variable such that

   - $\mathbb{E}\left[\left(Y_{t,n} - \mathbb{E}[Y_{t,n} \,|\, \boldsymbol{y}_t]\right)^2 \,\middle|\, \boldsymbol{y}_t\right] = \text{Var}[Y_{t,n} \,|\, \boldsymbol{y}_t] = s_n y_{t,n}\left(1 - \frac{y_{t,n}}{s_n}\right) \leq \|\boldsymbol{s}\|_\infty y_{t,n} \quad \Rightarrow \quad \sum_{n=1}^{N} \text{Var}[Y_{t,n} \,|\, \boldsymbol{y}_t] \leq \|\boldsymbol{s}\|_\infty C,$
   - $\left|Y_{t,n} - \mathbb{E}[Y_{t,n} \,|\, \boldsymbol{y}_t]\right| \leq s_n \leq \|\boldsymbol{s}\|_\infty$.

   Using the law of total probability, (14), and Bernstein's inequality, we can write

$$P\left(\sum_{n=1}^{N} Y_{t,n} \geq (1+\delta)C\right) = \mathbb{E}\left[P\left(\sum_{n=1}^{N} Y_{t,n} \geq (1+\delta)C \,\middle|\, \boldsymbol{y}_t\right)\right] \overset{(14)}{\leq} \mathbb{E}\left[P\left(\sum_{n=1}^{N}\left(Y_{t,n} - \mathbb{E}\left[Y_{t,n} \,\middle|\, \boldsymbol{y}_t\right]\right) \geq \delta C \,\middle|\, \boldsymbol{y}_t\right)\right]$$
$$\overset{\text{Bernstein}}{\leq} \mathbb{E}\left[\exp\left(-\frac{\frac{1}{2}(\delta C)^2}{\|\boldsymbol{s}\|_\infty C\left(1+\frac{\delta}{3}\right)}\right)\right] = \exp\left(-\frac{3C}{2\|\boldsymbol{s}\|_\infty}\frac{\delta^2}{3+\delta}\right).$$

5. Switching cost: Since $\boldsymbol{V}$ is shared over time (every component V_n is drawn only once from its distribution), we have
$$(Y_{t,n} - Y_{t+1,n})^+ = \begin{cases} s_n, & \text{if } \frac{y_{t+1,n}}{s_n} < V_n \leq \frac{y_{t,n}}{s_n}, \\ 0, & \text{otherwise.} \end{cases}$$
The non-zero case occurs with probability $(y_{t,n} - y_{t+1,n})^+ / s_n$, yielding the expected value $(y_{t,n} - y_{t+1,n})^+$. The second equality follows by symmetry.

6. Minimal switching cost: For any unbiased rounding scheme, that is, $\mathbb{E}\left[\boldsymbol{Y}_t \mid \boldsymbol{y}_t\right] = \boldsymbol{y}_t$, the following bound holds (use Jensen's inequality and the convexity of the switching cost to get the result):
$$\mathbb{E}\left[\|\boldsymbol{Y}_{t+1} - \boldsymbol{Y}_t\|_1 \mid \boldsymbol{y}_{t+1}, \boldsymbol{y}_t\right] \geq \left\|\mathbb{E}\left[\boldsymbol{Y}_{t+1} - \boldsymbol{Y}_t \mid \boldsymbol{y}_{t+1}, \boldsymbol{y}_t\right]\right\|_1 = \left\|\boldsymbol{y}_{t+1} - \boldsymbol{y}_t\right\|_1. \tag{15}$$
As Poisson rounding achieves the right-hand side, it has minimal switching cost. □

By Theorem 7, qOGDP has the same guarantees in terms of regret and switching cost as qOGD. In particular, the static regret of qOGDP is $O\left(\sqrt{CT}\right)$; see Corollary 1. While qOGDP guarantees feasability only in expectation, Theorem 7 shows that the probability of exceeding the cache capacity C by more than δC decrease exponentially fast with δ.

5. Experiments

We evaluate qOGD against baselines using synthetic and open-source traces. First, we detail the experimental setup in Section 5.1. We then use synthetic traces to validate the static regret/switching cost bounds and analyze the impact of quantization in Section 5.2. Finally, in Section 5.3, we report results on open-source traces, comparing utility, runtime and memory usage.

5.1. Experimental Setup

Algorithms. We compare qOGDP against LRU, SIEVE [48], Greedy-Dual-Size (GDS) [13], and a heuristic based on Follow-The-Leader (FTL) [32], which we simply refer to as FTL. LRU evicts the least recently requested object. SIEVE keeps objects in FIFO order but protects recently re-referenced ones with a one-bit "visited" flag, achieving LRU-like recency awareness with much lower update overhead. GDS and FTL, like qOGD, can account for the weight associated with each object. GDS generalizes LRU by replacing pure recency with an aging priority that also accounts for each object's weight and size, so that eviction favors objects that are old, large, and/or have low weight per unit of cache space. FTL extends Freq/Size [10] to general weights. It is based on the continuous knapsack relaxation of the caching problem. At each round, the value of an object is given by its cumulative reward over past requests. The policy computes the fractional knapsack optimum and then constructs an integral cache by keeping only the objects whose fractional variable is equal to one. In our experiments we consider two different choices of utility weights leading to two variants of qOGD, qOGDP, GDS, and FTL:

_b variant: $w_{t,r_t} = 1$, the reward is the fraction of bytes served locally.
_o variant: $w_{t,r_t} = 1/s_{r_t}$, the reward is the fraction of objects served locally.

Note that both variants coincide in the uniform-object-size case and that FTL_b coincides with plain LFU, namely LFU without aging or counter resets.

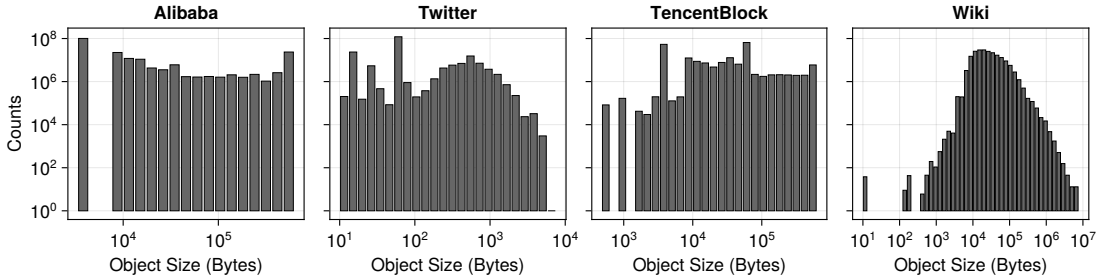


Figure 3: Histogram of the object sizes in each of the open-source traces.

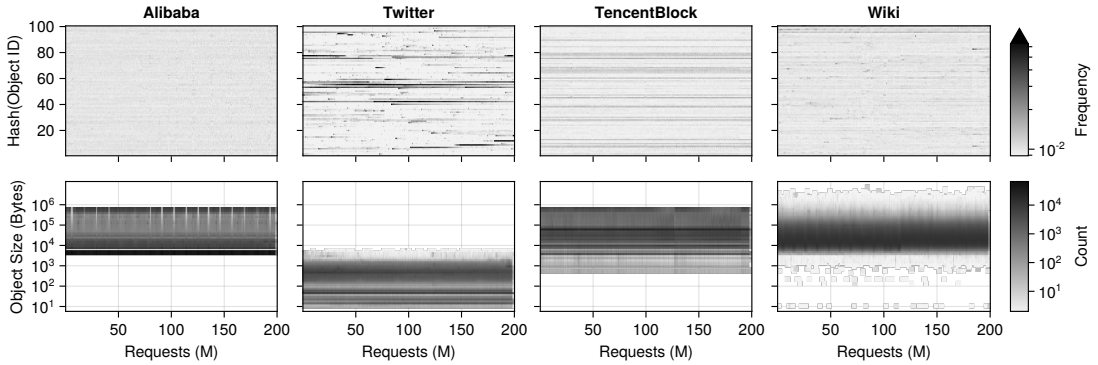


Figure 4: Top: Trace sketches showing the temporal evolution of object popularity. Object IDs are hashed and requests are aggregated into windows of 10^5 requests. Object-size heatmaps for the same traces, using logarithmic size bins with 10 buckets per order of magnitude and the same aggregation windows of 10^5 requests.

Implementation details. All algorithms are implemented in Julia. The implementations of qOGD and qOGDP follow Algorithm 5; qOGDP additionally applies the Poisson rounding step in Lines 7 and 13, as described in Section 4.3. qOGDP and FTL use "ghost" caches to store non-zero metadata associated with objects that are not currently cached. GDS and FTL rely on heap-based data structures, whereas LRU and SIEVE are implemented using linked hash tables.

Traces. We use both synthetic and open-source traces for evaluation. Two synthetic traces, Round Robin and Zipfian (with parameter $a = 1$), are used to evaluate the effect of granularity, switching cost and static regret guarantees. The open-source traces consist of the first 200 million requests of 4 different traces from [1], namely,

Alibaba [23]: (`alibabaBlock_107`). Workloads from Elastic Block Storage virtual disks (Ultra Disk). It has a catalog size of 4.55 TB with 74.96 million unique objects. The average requested object size is 82.52 KB and the harmonic mean is 6.72 KB.

Twitter [46]: (`cluster_52`). Key-value requests from Twitter production clusters (Twemcache/Memcached). It has a catalog size of 1.17 GB with 7.46 million unique objects. The average requested object size is 173.27 B, while the harmonic mean is 50.01 B.

TencentBlock [47]: (`TencentBlock_1063`). Requests from Cloud Block Storage, which provides virtual disk storage. It has a catalog size of 1.10 TB with 11.59 million unique objects. The average requested object size is 59.25 KB and the harmonic mean is 10.85 KB.

Wiki [45]: (`wiki_2019t`). Image upload and text web requests from a CDN cache server. It has a catalog size of 421.49 GB with 18.03 million unique objects. The average requested object size is 33.32 KB and the harmonic mean is 19.37 KB.

The object-size distribution for each of the four traces is shown in Fig. 3. The frequency and size of requested objects, aggregated over bins of 100,000 requests, are reported in Fig. 4. The four traces exhibit markedly different

characteristics. Average object sizes differ by almost three orders of magnitude between Twitter and TencentBlock; the range of object sizes also varies substantially, from about three orders of magnitude for Twitter to about six for Wiki. The traces also differ in their popularity dynamics, ranging from stable to strongly shifting patterns. Alibaba shows a relatively homogeneous popularity pattern over time and across objects. TencentBlock is more heterogeneous across objects, but its popularity remains largely stable over time. Wiki exhibits a temporal popularity shift, although a weak one, whereas Twitter shows the strongest popularity shift among the four traces.

Metrics. We consider several metrics in our experimental analysis.

Normalized static regret: The regret computed on the trace with respect to the best static cache allocation in hindsight, divided by the time horizon.

Byte write ratio: The ratio of bytes inserted in cache to the total requested bytes. It is closely related to the switching cost, which also accounts for evictions.

Byte hit ratio: The ratio of bytes served by the cache to the total requested bytes.

Object hit ratio (/rate): The ratio of the cumulative object fractions served by the cache to the number of requests.

Runtime: The wall-clock time needed to simulate an algorithm on a trace.

Cache entries: The number of objects with stored metadata at a given time, either in the cache or in the ghost cache; this serves as a proxy for memory usage.

Occupancy: The ratio of the amount of number of bytes stored in the cache at a given time to the cache capacity.

Observe that the byte hit ratio and the byte write ratio sum to 1 in policies that always add the requested object to the cache, such as LRU, GDS, and SIEVE.

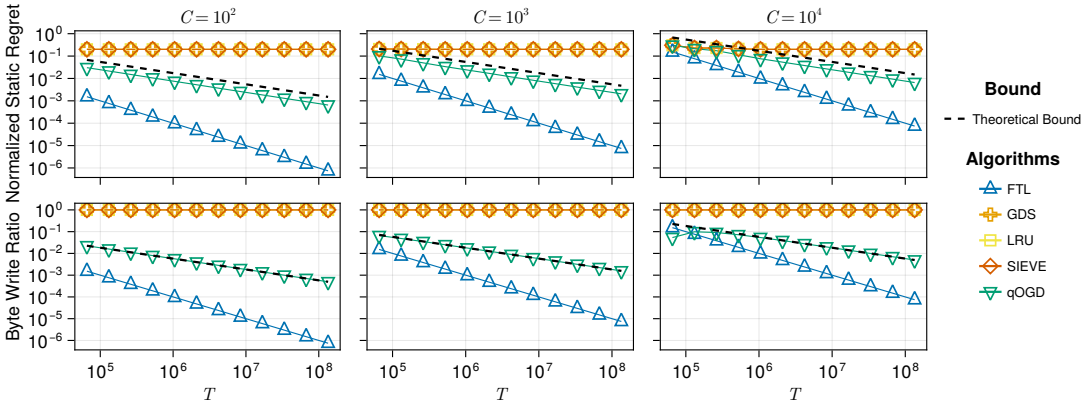
5.2. Results: Synthetic Traces

Validating theoretical bounds. To examine the tightness of the bounds derived in Corollary 1 and Theorem 3, we measure the normalized static regret and the byte write ratio on the synthetic traces. We consider a uniform-object-size-case ($\boldsymbol{s} = \mathbf{1}$) and set $\tilde{\eta} = w_{\max} = 1$. We set the granularity as specified in Remark 1. Thereby, the normalized static regret bound is $\sqrt{3C/T}$ and the byte write ratio bound is $\sqrt{C/3T}$, since the switching-cost bound in (8) reduces to $\tilde{\eta}\varepsilon T$). We consider cache sizes $C \in \{10^2, 10^3, 10^4\}$ and numbers of requests T doubling from 2^{16} to 2^{27}.

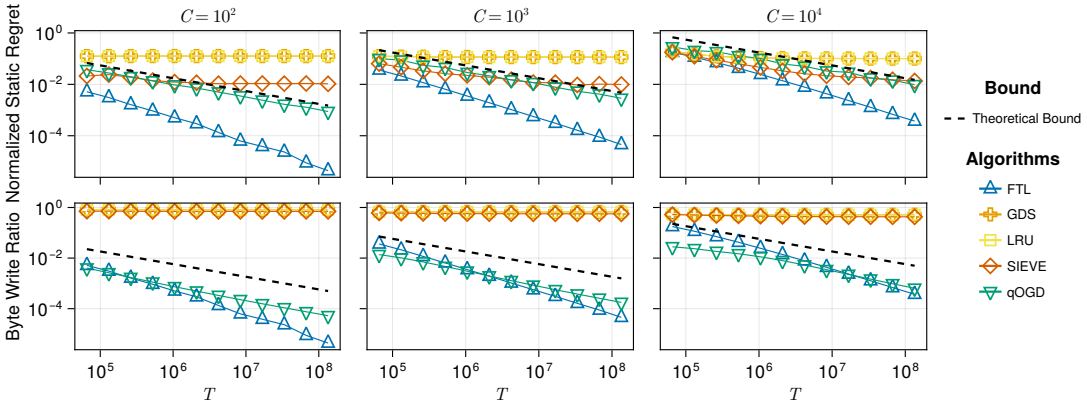
Results on the Round Robin trace (Fig. 5a) show that the normalized static regret bound is tight within a factor of 3, and that the byte write ratio bound is attained by this trace. Traditional heuristics such as GDS, LRU, and SIEVE fail on this scan workload, while both FTL and qOGD perform well. In this particular trace, FTL stores the first C requested objects and never changes them, which is optimal because all objects are requested with the same frequency. By contrast, qOGD reacts to the sequence of requests and tends to shift mass toward more recently requested objects; this is suboptimal for this specific trace, but qOGD still satisfies the regret guarantee.

Results on the Zipfian trace (Fig. 5b) validate the theoretical bounds for both normalized static regret and byte write ratio, although the byte write ratio bound is loose. In this i.i.d. setting, FTL rapidly converges to the expected optimal static cache state, consistently with its $O(1/T)$ normalized regret rate on Zipfian traces [12]. SIEVE reaches a low but non-vanishing normalized static regret, whereas GDS and LRU perform poorly. Note that the strong performance of FTL on these two traces should be interpreted as trace-specific: FTL is not a no-regret policy under general adversarial request sequences (see [36, Proposition 1]).

Effect of granularity and step size. To examine the effect of granularity and step size on the performance of qOGD, we analyze the byte hit ratio and the byte write ratio obtained with different granularity $\varepsilon \in \{1, 4, 16, 64, 256\}$ and step sizes $\tilde{\eta}$ doubling from 1 until $\varepsilon\tilde{\eta} = 1024$ is reached. The experiments ran for $T = 10^7$ requests with cache sizes $C/s \in \{10^2, 10^3, 10^4\}$ where $s = 1024$ bytes is the uniform object size. We show the results in Fig. 6. For the same granularity ε, decreasing the step size $\tilde{\eta}$ always improves the byte write ratio (it decreases) as predicted by the bound on the switching cost ($\tilde{\eta}\varepsilon T$ in (8)). The effect of $\tilde{\eta}$ on the hit ratio is instead non-monotone. This can be intuitively explained as follows. When $\tilde{\eta}$ is too small, the caching policy reacts too slowly to the request stream and, in particular, takes a long time to fill the cache. Conversely, when $\tilde{\eta}$ is too large, the cache state becomes too sensitive to individual requests, leading to excessive cache updates and premature evictions of useful objects. As a result, intermediate values of $\tilde{\eta}$ provide the best trade-off between responsiveness and stability. Another interesting observation is that



(a) Round Robin, $N = 5C$



(b) Zipf $a = 1$

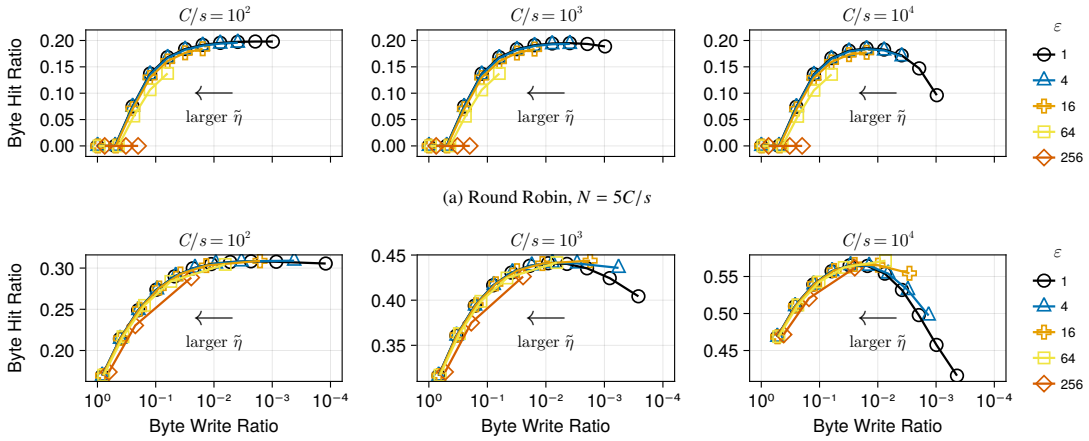
Figure 5: Normalized static regret and byte write ratio in uniform-object-size case ($\boldsymbol{s} = \mathbf{1}$). For qOGD, $\varepsilon = \sqrt{C/3T}$.

curves corresponding to different values of ε partially overlap. For instance, the performance obtained with $\tilde{\eta} = 4$ and granularity $\varepsilon = 16$ is close to that obtained with $\tilde{\eta} = 64$ and granularity $\varepsilon = 1$ for both traces. Intuitively, as long as the product $\tilde{\eta}\varepsilon$ is kept constant, qOGD mimics an OGD update with the same effective stepsize $\eta = \tilde{\eta}\varepsilon$, which explains the similar performance. This effect is visible in the non-Pareto-optimal region, where increasing $\tilde{\eta}$ degrades both the byte hit ratio and the byte write ratio. On the Pareto frontier, however, the precise value of ε becomes important, as it controls the granularity of the cache updates and therefore the attainable trade-off between hit ratio and write ratio.

5.3. Results: Open-Source Traces

Scalability with Catalog Size. To empirically validate the theoretical time complexity claims presented in Theorem 6, we evaluate the runtime of qOGDP against known baseline algorithms. Following the methodology in [15], we use the `wiki_2019t` trace subsampled to $T = 10^6$ requests with a uniform object size of $s = 1024$ and $w_{t,r_t} = 1$. This maintains a similar empirical popularity while ensuring a consistent number of requests. We vary the step size $\tilde{\eta} \in \{1, 4, 16\}$, the catalog size $N \in \{10^3, 10^4, 10^5\}$, and set the cache capacity as fraction of catalog size $C/s = 0.05N$. To maintain roughly equivalent switching costs and byte hit ratios for different step sizes, we scale the granularity accordingly by setting $\varepsilon = 64/\tilde{\eta}$. Runtime measurements are averaged over 20 runs.

As illustrated in Fig. 7, neither the catalog size N nor the cache size C significantly affect the runtime of qOGDP. Instead, the computational overhead is governed almost entirely by the step size $\tilde{\eta}$, an observation that is directly explained in Theorem 6. Among the baselines, GDS exhibits a clear $O(\log C)$ runtime. Although FTL is implemented using a heap, its updates become increasingly infrequent over time due to the algorithm's convergent nature, yielding an effective empirical runtime of $O(1)$. Finally, LRU and SIEVE both maintain steady $O(1)$ runtime, as expected.



(a) Round Robin, $N = 5C/s$

(b) Zipf $a = 1$

Figure 6: Byte hit ratio and byte write ratio achieved by qOGD, when configured with different granularity $\varepsilon \in \{1, 4, 16, 64, 256\}$ and step sizes $\tilde{\eta}$ doubling from 1 until $\varepsilon\tilde{\eta} = 1024$ is reached; $T = 10^7$ and uniform object size is $s = 1024$ bytes. Along each curve, increasing $\tilde{\eta}$ moves the points to the left (larger byte write ratio).

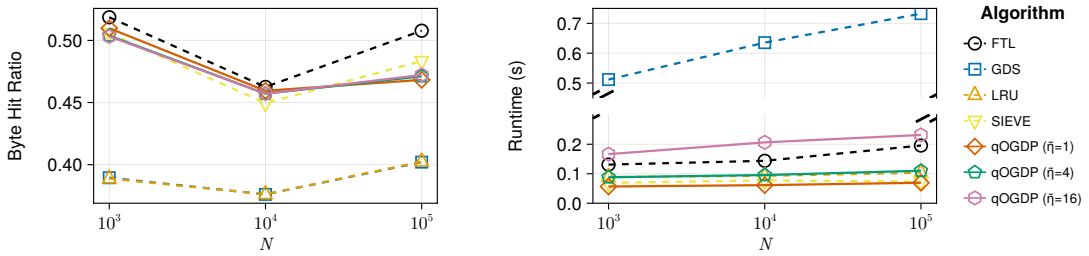


Figure 7: Byte hit ratio (left) and average runtime over 20 runs (right) on a subsampled `wiki_2019t` trace with varying catalog size N, cache size $C/s = 0.05N$, and $T = 10^6$. GDS has a clear $O(\log C)$ runtime while others are $O(1)$. qOGDP's runtime scales with $\tilde{\eta}$. The byte hit ratio is approximately the same across most algorithms, ensuring the runtime results are comparable.

Notably, qOGDP remains highly competitive with these constant-time algorithms, particularly at smaller step sizes (e.g., $\tilde{\eta} = 1$). The slight growth in runtime for constant-time algortihms occurs because dictionary lookups become more expensive as the key count grows, driven by CPU cache hierarchy misses.

Performance Curves. To investigate the practical performance of qOGDP, we compare the byte/object hit ratio and byte write ratio against baseline algorithms on a representative set of open-source traces. We also track the number of cache entries. For each cache capacity, a fraction $C_{\text{norm}} \in \{0.0001, 0.001, 0.01, 0.1\}$ of the catalog size, we generate a performance curve approximately with a range of $[0.001, 1]$ byte write ratio. This is achieved by varying the step size accordingly as byte write ratio is monotonically related to the step size. These curves effectively describe a Pareto front between the byte/object hit ratio and the byte write ratio. To limit the granularity's effect on the object/byte hit ratio, we restrict the following term to 0.01: $\varepsilon \sum_{t=1}^{T} w_{t,r_t} / \sum_{t=1}^{T} w_{t,r_t} s_{r_t}$, motivated by the regret guarantee. This term is proportional to the average and harmonic mean of the requested object size for byte and object hit ratio, respectively. While selecting granularity in this manner requires prior knowledge, it is justified by the stable behavior of the requested object size (see Fig. 4). Specifically, the parameters ε and $\tilde{\eta}$ are configured as follows:

$$\text{for qOGDP_b}: \begin{cases} \varepsilon = \left\lceil \frac{1}{100T} \sum_{t=1}^{T} s_{r_t} \right\rceil, \\ \tilde{\eta} = 2^{j-1}, \quad j \in [11], \end{cases} \qquad \text{for qOGDP_o}: \begin{cases} \varepsilon = \left\lceil \frac{T}{100 \sum_{t=1}^{T} (s_{r_t})^{-1}} \right\rceil, \\ \tilde{\eta} = 300 \left\lceil \frac{T}{100 \sum_{t=1}^{T} (s_{r_t})^{-1}} \right\rceil 2^{j-1}, \quad j \in [11]. \end{cases}$$

Fig. 8 presents the results of the experiments. The performance curves are smooth and vary continuously with

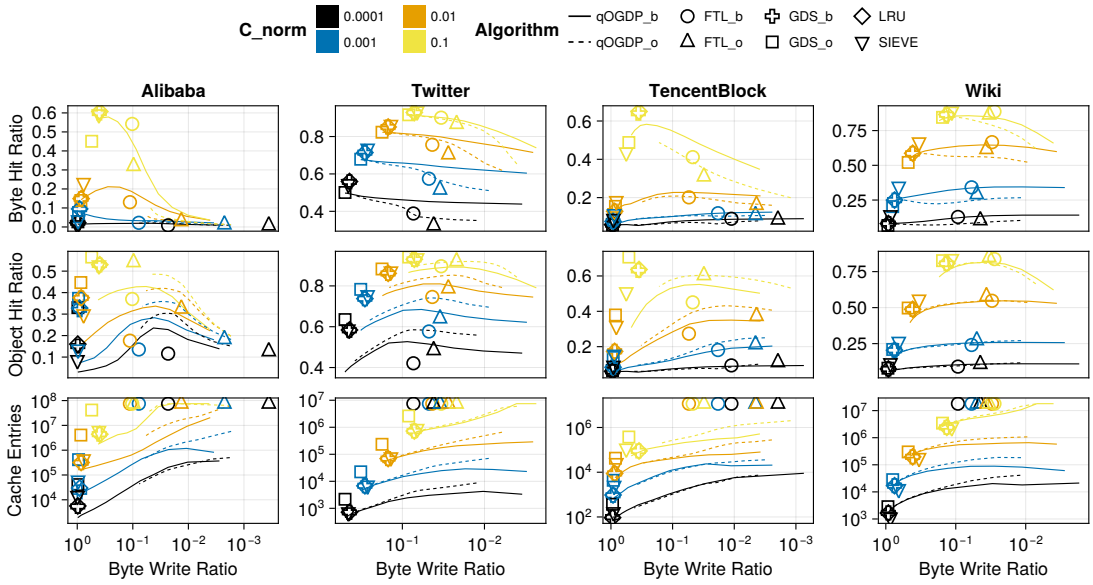


Figure 8: Performance curves of qOGDP describing object/byte hit ratio, byte write ratio and cache entries over a range of step sizes (along each curve, increasing $\tilde{\eta}$ moves the values to the left (larger byte write ratio)). The granularity is sufficiently small so it does not affect performance.

Table 4: Averaged results of Fig. 8. For each trace and cache size, qOGDP_b/qOGDP_o picks the step size yielding the best byte/object hit ratio.

Algorithm	Byte write ratio (%)	Byte hit ratio (%)
FTL_b	**5.35**	38.54
GDS_b	58.76	41.24
LRU	58.61	41.39
SIEVE	57.50	*42.50*
qOGDP_b	*19.02*	**42.56**

Algorithm	Byte write ratio (%)	Object hit ratio (%)
FTL_o	*2.78*	44.53
GDS_o	62.46	**49.13**
LRU	58.61	43.98
SIEVE	57.50	43.95
qOGDP_o	**2.74**	*48.78*

cache capacity. The **byte/object hit ratios** of qOGDP remain comparable to the baseline algorithms over a wide range of step sizes. Algorithms optimized for a specific objective perform better on their respective metrics. GDS_b is almost identical to LRU. qOGDP nearly dominates FTL at equivalent byte write ratios. We average these results in Table 4. On average, *qOGDP has a comparable if not the best hit ratio while achieving a lower (if not the best) byte write ratio*. The number of FTL **cache entries** effectively equals the number of unique objects in the catalog. Generally, the object variants hold more entries because they prefer to store smaller objects. Increasing the qOGDP step size yields large counters more frequently. This forces the fractional cache state to maintain fewer objects and reduces the total number of cache entries.

Metrics over Time. To inspect the behavior of the algorithms over time, we measure the byte hit ratio and byte write ratio in bins of 80,000 requests, and track the number of cache entries at the end of each bin. Among the results of Fig. 8, we focus on qOGDP_b (with $\tilde{\eta} = 16$), LRU and SIEVE, and show the time results with $C_{\text{norm}} = 0.01$ in Fig. 9.

At initialization, qOGDP_b uses more cache entries because it is not reducing any counters yet. After this phase, the metrics behave consistently over time. The byte hit ratio has consistent behavior across all algorithms with spikes in Alibaba and dips in Wiki, and the byte write ratio is closely linked to it. The occupancy plots show that the cache capacity constraint with qOGDP_b is not strictly enforced due to Poisson rounding. Infeasibility occurs both infrequently and in small proportions; the worst case across the traces is TencentBlock, where the maximum occupancy reaches 1.0258 and is infeasible 67 out of the 480 samples ($\approx 14\%$). In the Wiki trace, infeasibility occurs less than 1.5% of the time and stays within 0.2% of the cache size. Most of the variation in occupancy is due to the

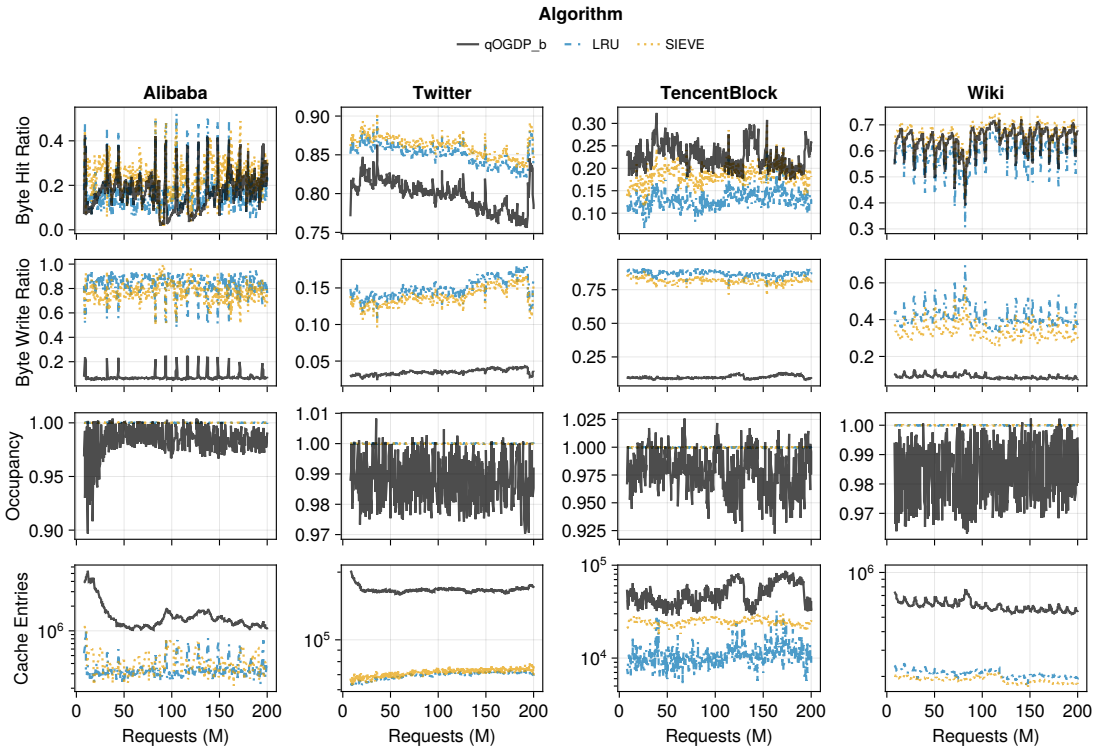


Figure 9: Metrics over time for qOGDP_b (with $\tilde{\eta} = 16$), LRU and SIEVE with $C_{\text{norm}} = 0.01$. The byte hit ratio and byte write ratio are computed in bins of 80,000 requests.

variation of the occupancy of the underlying qOGD.

6. Discussion and Future Work

To maintain the conciseness of this paper, we have deferred generalizations to this section, where we also discuss potential future work.

6.1. From amortized to worst-case time

In the presented implementation of qOGD Algorithm 5, computation clusters during the uniform eviction (Line 8). Its runtime depends on the step size yet can be faster than step-size-independent alternatives because linear scans are efficient. We present this simpler version because it is easier to modify and generalize.

A worst-case version of qOGD can be implemented similarly to SIEVE. Instead of performing the while loop all at once, computation is amortized by lazily executing it, achieving $O(\tilde{\eta})$ worst-case time per request. During one request, the dictionary is scanned to evict the same number of pieces added and a bit is used to keep track of objects already updated. Dictionary resizes make the scan inconsistent so they must be handled carefully or alternatively scanning using a linked list avoids this issue.

qOGDP can also be implemented similarly to OGB [15] to achieve a $O(1)$ worst-case time per request independent of step size. An implicit representation is used, quantization allows replacing the $O(\log(N))$ comparison-based data structures with $O(1)$ bucket queues similarly to how $O(1)$ LFU is implemented [27].

6.2. Regret refinements

OGD on bounded domains is nonequivalent under linear transformations, which results in different dynamic regret guarantees, exactly how online mirror descent (OMD) has different dynamic regret guarantees. OGD for

caching with uniform object sizes [36] has a symmetric parameterization as: $\left\{\boldsymbol{y} \in \mathbb{R}^N \;\middle|\; \sum_{n=1}^N y_n \leq C \text{ and } \mathbf{0} \leq \boldsymbol{y} \leq \mathbf{1}\right\}$. Keeping object information in separate components allows for sparse representations and local updates. In the case of heterogeneous object sizes, while we use $\mathcal{Y}$ (see (1)), Carra et al. [15] uses a different parameterization: $\left\{\boldsymbol{y} \in \mathbb{R}^N \;\middle|\; \sum_{n=1}^N s_n y_n \leq C \text{ and } \mathbf{0} \leq \boldsymbol{y} \leq \mathbf{1}\right\}$. Our parameterization simplifies projecting onto the set defined by $\sum_{n=1}^N y_n \leq C$ and quantizes naturally to $\mathbb{N}^N$. Both the approach in [15] and our quantized implementation are difficult to generalize to the other parameterization. We conjecture qOGD to be an OGD with a relaxed projection for both parameterizations, when $\varepsilon \to 0$, thereby obtaining both our dynamic regret guarantee and that of [15] simultaneously.

6.3. Common variations

We did not explicitly handle batching, subsampling [9], or varying constraints. Generalizing to these cases is straightforward given the presented lemmas and theorems. Both batching and subsampling can be achieved by modifying $\boldsymbol{u}_t$. Batching accumulates the utility of the full batch in $\boldsymbol{u}_t$, while subsampling unbiasedly replaces $\boldsymbol{u}_t$ with $\mathbf{0}$ to avoid an update. For batching, the regret analysis only requires changing the norm of $\boldsymbol{u}_t$. For subsampling, the analysis uses Lemma 8 and the dynamic regret guarantee with the sampled gradient, which is a similar but more general approach compared to that in [9]. Our general dynamic regret bound can be generalized to varying constraints, which, in a caching context, permits the cache capacity or object sizes to change over time. Batching makes the cache less reactive and can degrade performance when an object is requested multiple times within a single batch. This weakness can be mitigated by adding a small FIFO queue.

6.4. Picking the step size

Selecting the step size for qOGD is nontrivial. To use the dynamic regret guarantees for this selection, one needs to know the path length and the norm of a good comparator, which is hard without prior knowledge. A typical solution to this problem in online learning is to apply a learning-with-experts framework over multiple qOGD instances, each with a different step size, similar to the approach in [38]. The overhead of implementing multiple qOGD instances is manageable, as they can share data structures like hash maps for their counters. However, the potential switching between experts can be problematic. While fixed convex combinations of caching policies do not incur additional switching costs, this cost can scale with the catalog size when the weights of the combination vary [49]. Batching learning with experts could potentially make this approach practical. The resulting meta-learner can also integrate various caching policies other than qOGD to reach the best possible balance between reward and switching costs.

6.5. Feasibility adjustment

The feasibility adjustment in Proposition 1 was chosen for convenience, as it is simple to implement and its proof of feasibility is elegant. While the uniform adjustment is a natural choice for optimizing the byte hit ratio, other adjustments may yield better results for different objectives. Poisson rounding can be extended with a feasibility adjustment, though this incurs a regret penalty. For example, by evicting objects with a probability proportional to their size when the cache is full, the byte hit ratio degrades by a factor of the expected occupancy.

7. Conclusion

This work demonstrates that the gap between theoretical online-learning-based caching algorithms and practical high-performance caching systems is narrower than previously assumed. While prior work often treated numerical granularity as an implementation detail, we established it as a central feature of the algorithm design. We quantified the effect of quantization, showing that enforcing quantized states does not unrecoverably destroy dynamic regret guarantees. Ultimately, the techniques used to design qOGD offer a general methodology for designing other quantized OGD algorithms.

Acknowledgements

This research was supported in part by the French government, through the 3IA Côte d'Azur Investments in the Future project managed by the National Research Agency (ANR) with the reference number ANR-19P3IA-0002, the European Network of Excellence dAIEDGE under Grant Agreement No. 101120726, and the EU HORIZON MSCA 2023 DN project FINALITY (G.A. 101168816).

References

[1] 2025. cacheMon/cache_dataset. https://github.com/cacheMon/cache_dataset original-date: 2024-07-31T23:12:48Z.

[2] Anna Adamaszek, Artur Czumaj, Matthias Englert, and Harald Räcke. 2019. An O (log k)-Competitive Algorithm for Generalized Caching. *ACM Transactions on Algorithms* 15, 1 (Jan. 2019), 1–18. https://doi.org/10.1145/3280826

[3] Akamai Technologies. [n. d.]. Large File Optimization. https://techdocs.akamai.com/property-mgr/docs/large-file-optimization-lfo

[4] Abubakr O. Al-Abbasi and Vaneet Aggarwal. 2020. TTLCache: Taming Latency in Erasure-Coded Storage Through TTL Caching. *IEEE Transactions on Network and Service Management* 17, 3 (Sept. 2020), 1582–1596. https://doi.org/10.1109/TNSM.2020.2998175

[5] Lachlan L. H. Andrew, Siddharth Barman, Katrina Ligett, Minghong Lin, Adam Meyerson, Alan Roytman, and Adam Wierman. 2013. A Tale of Two Metrics: Simultaneous Bounds on Competitiveness and Regret. In *Proceedings of the 26th Annual Conference on Learning Theory (COLT)*. 741–763.

[6] Nikhil Bansal, Niv Buchbinder, and Joseph (Seffi) Naor. 2010. Towards the Randomized k -Server Conjecture: A Primal-Dual Approach: (Extended Abstract). In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, 40–55. https://doi.org/10.1137/1.9781611973075.5

[7] Soumya Basu, Aditya Sundarrajan, Javad Ghaderi, Sanjay Shakkottai, and Ramesh Sitaraman. 2018. Adaptive TTL-Based Caching for Content Delivery. *IEEE/ACM Transactions on Networking* 26, 3 (June 2018), 1063–1077. https://doi.org/10.1109/TNET.2018.2818468

[8] Heinz H. Bauschke and Patrick L. Combettes. 2017. *Convex Analysis and Monotone Operator Theory in Hilbert Spaces*. Springer International Publishing, Cham. https://doi.org/10.1007/978-3-319-48311-5

[9] Younes Ben Mazziane, Francescomaria Faticanti, Sara Alouf, and Giovanni Neglia. 2026. Efficient and Optimal No-Regret Caching Under Partial Observation. *IEEE Transactions on Networking* 34 (2026), 2020–2030. https://doi.org/10.1109/TON.2025.3636929

[10] Daniel S Berger, Nathan Beckmann, and Mor Harchol-Balter. 2018. Practical bounds on optimal caching with variable object sizes. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 2, 2 (2018), 1–38.

[11] Rajarshi Bhattacharjee, Subhankar Banerjee, and Abhishek Sinha. 2020. Fundamental limits on the regret of online network-caching. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 4, 2 (2020), 1–31. https://doi.org/10.1145/3392143

[12] Archana Bura, Desik Rengarajan, Dileep Kalathil, Srinivas Shakkottai, and Jean-Francois Chamberland. 2022. Learning to Cache and Caching to Learn: Regret Analysis of Caching Algorithms. *IEEE/ACM Transactions on Networking* 30, 1 (Feb. 2022), 18–31. https://doi.org/10.1109/TNET.2021.3105880

[13] Pei Cao and Sandy Irani. 1997. {Cost-Aware}{WWW} Proxy Caching Algorithms.

[14] Damiano Carra, Giovanni Neglia, and Pietro Michiardi. 2020. Elastic Provisioning of Cloud Caches: A Cost-Aware TTL Approach. *IEEE/ACM Transactions on Networking* 28, 3 (June 2020), 1283–1296. https://doi.org/10.1109/TNET.2020.2980105

[15] Damiano Carra, Giovanni Neglia, and Xufeng Zhang. 2025. Low-Complexity online learning for caching. *Computer Networks* 273 (Dec. 2025), 111743. https://doi.org/10.1016/j.comnet.2025.111743

[16] Nicolo Cesa-Bianchi and Gabor Lugosi. 2006. *Prediction, Learning, and Games* (1 ed.). Cambridge University Press. https://doi.org/10.1017/CBO9780511546921

[17] Nicaise Eric Choungmo Fofack, Philippe Nain, Giovanni Neglia, and Don Towsley. 2012. Analysis of TTL-based Cache Networks. In *Proceedings of the 6th International Conference on Performance Evaluation Methodologies and Tools*. IEEE, Cargèse, France. https://doi.org/10.4108/valuetools.2012.250250

[18] Amit Daniely and Yishay Mansour. 2019. Competitive ratio vs regret minimization: achieving the best of both worlds. In *Proceedings of the 30th International Conference on Algorithmic Learning Theory (Proceedings of Machine Learning Research, Vol. 98)*, Aurélien Garivier and Satyen Kale (Eds.). PMLR, 333–368. https://proceedings.mlr.press/v98/daniely19a.html

[19] Dan Garber. 2021. Efficient Online Linear Optimization with Approximation Algorithms. *Mathematics of Operations Research* 46, 1 (Feb. 2021), 204–220. https://doi.org/10.1287/moor.2020.1053

[20] Ehab Ghabashneh and Sanjay Rao. 2020. Exploring the interplay between CDN caching and video streaming performance. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*. IEEE Press, 516–525. https://doi.org/10.1109/INFOCOM41043.2020.9155338

[21] Andrew Jacobsen. 2024. Adapting to Non-stationarity in Online Learning. (2024). https://ajjacobs.github.io/thesis/Jacobsen_Andrew_PhD.pdf

[22] Sham M. Kakade, Adam Tauman Kalai, and Katrina Ligett. 2009. Playing Games with Approximation Algorithms. *SIAM J. Comput.* 39, 3 (Jan. 2009), 1088–1106. https://doi.org/10.1137/070701704

[23] Jinhong Li, Qiuping Wang, Patrick P. C. Lee, and Chao Shi. 2020. An In-Depth Analysis of Cloud Block Storage Workloads in Large-Scale Production. In *2020 IEEE International Symposium on Workload Characterization (IISWC)*. 37–47. https://doi.org/10.1109/IISWC50251.2020.00013

[24] Thodoris Lykouris and Sergei Vassilvitskii. 2021. Competitive Caching with Machine Learned Advice. *J. ACM* 68, 4 (Aug. 2021), 1–25. https://doi.org/10.1145/3447579

[25] Steven Lyons and Raju Rangaswami. 2024. To Cache or Not to Cache. *Algorithms* 17, 7 (July 2024), 301. https://doi.org/10.3390/a17070301

[26] William G Madow. 1949. On the theory of systematic sampling, II. *The Annals of Mathematical Statistics* 20, 3 (1949), 333–354.

[27] Dhruv Matani, Ketan Shah, and Anirban Mitra. 2010. An O(1) algorithm for implementing the LFU cache eviction scheme. https://doi.org/10.48550/arXiv.2110.11602 arXiv:2110.11602 [cs].

[28] Sara McAllister, Benjamin Berg, Julian Tutuncu-Macias, Juncheng Yang, Sathya Gunasekar, Jimmy Lu, Daniel S. Berger, Nathan Beckmann, and Gregory R. Ganger. 2021. Kangaroo: Caching Billions of Tiny Objects on Flash. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*. ACM, Virtual Event Germany, 243–262. https://doi.org/10.1145/3477132.3483568

[29] Naram Mhaisen, Abhishek Sinha, Georgios Paschos, and Georgios Iosifidis. 2022. Optimistic No-regret Algorithms for Discrete Caching.

Proceedings of the ACM on Measurement and Analysis of Computing Systems 6, 3 (Dec. 2022), 1–28. https://doi.org/10.1145/3570608 arXiv:2208.06414 [cs].

[30] Samrat Mukhopadhyay and Abhishek Sinha. 2021. Online Caching with Optimal Switching Regret. In *2021 IEEE International Symposium on Information Theory (ISIT)*. IEEE, Melbourne, Australia, 1546–1551. https://doi.org/10.1109/ISIT45174.2021.9517925

[31] Giovanni Neglia, Damiano Carra, and Pietro Michiardi. 2018. Cache Policies for Linear Utility Maximization. *IEEE/ACM Transactions on Networking* 26, 1 (Feb. 2018), 302–313. https://doi.org/10.1109/TNET.2017.2783623

[32] Francesco Orabona. 2025. A Modern Introduction to Online Learning. http://arxiv.org/abs/1912.13213 arXiv:1912.13213 [cs, math, stat].

[33] Nitish K. Panigrahy, Philippe Nain, Giovanni Neglia, and Don Towsley. 2022. A New Upper Bound on Cache Hit Probability for Non-Anticipative Caching Policies. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems* 7, 2-4 (Dec. 2022), 1–24. https://doi.org/10.1145/3547332

[34] Debjit Paria and Abhishek Sinha. 2021. LeadCache: Regret-Optimal Caching in Networks. In *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. S. Liang, and J. Wortman Vaughan (Eds.), Vol. 34. Curran Associates, Inc., 4435–4447. https://proceedings.neurips.cc/paper_files/paper/2021/file/2387337ba1e0b0249ba90f55b2ba2521-Paper.pdf

[35] Georgios S. Paschos, Apostolos Destounis, and George Iosifidis. 2020. Online Convex Optimization for Caching Networks. *IEEE/ACM Transactions on Networking* 28, 2 (April 2020), 625–638. https://doi.org/10.1109/TNET.2020.2968424

[36] Georgios S. Paschos, Apostolos Destounis, Luigi Vigneri, and George Iosifidis. 2019. Learning to Cache With No Regrets. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*. IEEE, Paris, France, 235–243. https://doi.org/10.1109/INFOCOM.2019.8737446

[37] Dhruv Rohatgi. 2020. Near-optimal bounds for online caching with machine learned advice. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 1834–1845.

[38] Zhenghao Sha, Kechao Cai, and Jinbei Zhang. 2025. SAcache: A Strongly Adaptive Online Caching Scheme for Non-Stationary Environments. *IEEE Networking Letters* 7, 1 (March 2025), 46–50. https://doi.org/10.1109/LNET.2024.3516321

[39] Shai Shalev-Shwartz. 2011. Online Learning and Online Convex Optimization. *Foundations and Trends® in Machine Learning* 4, 2 (2011), 107–194. https://doi.org/10.1561/2200000018

[40] Tareq Si Salem, Giovanni Neglia, and Stratis Ioannidis. 2023. No-regret Caching via Online Mirror Descent. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems* 8, 4 (Dec. 2023), 1–32. https://doi.org/10.1145/3605209

[41] Matt Smiley. 2022. How we diagnosed and resolved Redis latency spikes with BPF and other tools. https://about.gitlab.com/blog/how-we-diagnosed-and-resolved-redis-latency-spikes/

[42] Jim Summers, Tim Brecht, Derek Eager, and Bernard Wong. 2012. To chunk or not to chunk: implications for HTTP streaming video server performance. In *Proceedings of the 22nd International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV '12)*. Association for Computing Machinery, New York, NY, USA, 15–20. https://doi.org/10.1145/2229087.2229093

[43] Aditya Sundarrajan, Mingdong Feng, Mangesh Kasbekar, and Ramesh K. Sitaraman. 2017. Footprint Descriptors: Theory and Practice of Cache Provisioning in a Global CDN. In *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies (CoNEXT '17)*. Association for Computing Machinery, New York, NY, USA, 55–67. https://doi.org/10.1145/3143361.3143368

[44] Weiran Wang and Canyi Lu. 2015. Projection onto the capped simplex. https://doi.org/10.48550/arXiv.1503.01002 arXiv:1503.01002 [cs].

[45] Wikitech. 2025. Data Platform/Data Lake/Traffic/Caching — Wikitech,. https://wikitech.wikimedia.org/w/index.php?title=Data_Platform/Data_Lake/Traffic/Caching&oldid=2346221

[46] Juncheng Yang, Yao Yue, and K. V. Rashmi. 2020. A large scale analysis of hundreds of in-memory cache clusters at Twitter. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 191–208. https://www.usenix.org/conference/osdi20/presentation/yang

[47] Yu Zhang, Ping Huang, Ke Zhou, Hua Wang, Jianying Hu, Yongguang Ji, and Bin Cheng. 2020. OSCA: An Online-Model Based Cache Allocation Scheme in Cloud Block Storage Systems. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, 785–798. https://www.usenix.org/conference/atc20/presentation/zhang-yu

[48] Yazhuo Zhang, Juncheng Yang, Yao Yue, Ymir Vigfusson, and KV Rashmi. 2024. {SIEVE} is simpler than {LRU}: an efficient {Turn-Key} eviction algorithm for web caches. 1229–1246.

[49] Peng Zhao, Yu-Hu Yan, Yu-Xiang Wang, and Zhi-Hua Zhou. 2023. Non-stationary Online Learning with Memory and Non-stochastic Control. *Journal of Machine Learning Research* 24, 206 (2023), 1–70. http://jmlr.org/papers/v24/22-0218.html

[50] Shiji Zhou, Zhi Wang, Chenghao Hu, Yinan Mao, Haopeng Yan, Shanghang Zhang, Chuan Wu, and Wenwu Zhu. 2023. Caching in Dynamic Environments: A Near-Optimal Online Learning Approach. *IEEE Transactions on Multimedia* 25 (2023), 792–804. https://doi.org/10.1109/TMM.2021.3132156

[51] Martin Zinkevich. 2003. Online convex programming and generalized infinitesimal gradient ascent. 928–936.

Appendix A. A counterexample for marginal-preserving rounding with heterogeneous object sizes

We provide a simple example showing that, with heterogeneous object sizes, a feasible fractional cache state cannot always be rounded into feasible integral cache states while preserving the marginal inclusion probabilities. Consider a cache of capacity $C = 10$ and two objects with sizes

$$s_1 = 6, \qquad s_2 = 5.$$

An integral cache cannot store both objects simultaneously, since $s_1 + s_2 = 11 > C$. Hence, in any feasible integral cache state, at most one of the two objects can be present.

Now consider the fractional cache state

$$y_1 = 5, \qquad y_2 = 5.$$

This state is feasible for the fractional cache constraint and fills the cache exactly, since

$$y_1 + y_2 = 10 = C.$$

Suppose, by contradiction, that there exists a randomized rounding scheme that always produces feasible integral cache states and preserves the marginal inclusion probabilities y_n/s_n. Let A_n denote the event that object n is stored after rounding. Preserving the marginals would require

$$P(A_1) = \frac{y_1}{s_1} = \frac{5}{6}, \qquad P(A_2) = \frac{y_2}{s_2} = 1.$$

However, since the two objects cannot be stored together, the events A_1 and A_2 must be mutually exclusive in every feasible integral cache state. Therefore,

$$P(A_1) + P(A_2) \leq 1.$$

This contradicts the required marginals, since

$$\frac{5}{6} + 1 > 1.$$

Note how this example does not require independent rounding. Even if the rounding decisions are fully coordinated, the prescribed marginal inclusion probabilities may be incompatible with the requirement that every rounded cache state be feasible. Indeed, preserving the marginals means representing the fractional cache state as the expectation of a random feasible integral cache state. This is possible if and only if the fractional state belongs to the convex hull of feasible integral cache states. In the heterogeneous-object-size setting, the usual fractional cache polytope is generally larger than this convex hull: it enforces only the total capacity constraint and the per-object bounds, but it does not capture additional combinatorial constraints imposed by incompatible sets of objects. The example above exhibits exactly such a constraint: since the two objects cannot be stored together, their inclusion probabilities must sum to at most one, a condition that is not enforced by the fractional cache constraint.

Appendix B. Regret of OGD with Relaxed Projection

We first present preliminary lemmas and then prove Theorem 1.

Lemma 1 Let $\boldsymbol{y}_t$ be generated by a variant of OGD that uses a relaxed projection $P^R_{\mathcal{V}}$ instead of the Euclidean projection. Then, $\forall \boldsymbol{z}_t \in \mathcal{V}$, with $\boldsymbol{u}_t \in \mathbb{R}^N$ and $\eta \in \mathbb{R}^+$, we have

$$\left\|\boldsymbol{y}_{t+1} - \boldsymbol{z}_t\right\|^2 \leq \left\|\boldsymbol{y}_t - \boldsymbol{z}_t\right\|^2 + 2\eta\langle \boldsymbol{u}_t, \boldsymbol{y}_t - \boldsymbol{z}_t\rangle + \eta^2 \left\|\boldsymbol{u}_t\right\|^2. \tag{B.1}$$

Proof. By Definition 4 we can write

$$\begin{aligned}\left\|\boldsymbol{y}_{t+1} - \boldsymbol{z}_t\right\|^2 = \left\|P^R_{\mathcal{V}}(\boldsymbol{y}_t + \eta\boldsymbol{u}_t) - \boldsymbol{z}_t\right\|^2 &\leq \left\|\boldsymbol{y}_t + \eta\boldsymbol{u}_t - \boldsymbol{z}_t\right\|^2 \\ &= \langle \boldsymbol{y}_t - \boldsymbol{z}_t + \eta\boldsymbol{u}_t, \boldsymbol{y}_t - \boldsymbol{z}_t + \eta\boldsymbol{u}_t\rangle \\ &= \left\|\boldsymbol{y}_t - \boldsymbol{z}_t\right\|^2 + 2\eta\langle \boldsymbol{u}_t, \boldsymbol{y}_t - \boldsymbol{z}_t\rangle + \eta^2 \left\|\boldsymbol{u}_t\right\|^2,\end{aligned}$$

where we used $\langle \boldsymbol{a}, \boldsymbol{a}\rangle = \|\boldsymbol{a}\|^2$ and the distributivity property of the inner product to write the last two equalities. □

Lemma 2 $\forall \boldsymbol{z}_{t+1}, \boldsymbol{z}_t, \boldsymbol{y}_{t+1} \in \mathbb{R}^N$:

$$\left\|\boldsymbol{y}_{t+1} - \boldsymbol{z}_{t+1}\right\|^2 - \left\|\boldsymbol{y}_{t+1} - \boldsymbol{z}_t\right\|^2 = \|\boldsymbol{z}_{t+1}\|^2 - \|\boldsymbol{z}_t\|^2 + 2\langle \boldsymbol{z}_t - \boldsymbol{z}_{t+1}, \boldsymbol{y}_{t+1}\rangle. \tag{B.2}$$

Proof. The proof readily follows from the properties of the inner product:

$$\begin{aligned}\left\|\boldsymbol{y}_{t+1}-\boldsymbol{z}_{t+1}\right\|^2-\left\|\boldsymbol{y}_{t+1}-\boldsymbol{z}_t\right\|^2 &= \langle \boldsymbol{y}_{t+1}-\boldsymbol{z}_{t+1}, \boldsymbol{y}_{t+1}-\boldsymbol{z}_{t+1}\rangle - \langle \boldsymbol{y}_{t+1}-\boldsymbol{z}_t, \boldsymbol{y}_{t+1}-\boldsymbol{z}_t\rangle \\ &= \left\|\boldsymbol{y}_{t+1}\right\|^2-\left\|\boldsymbol{y}_{t+1}\right\|^2+\|\boldsymbol{z}_{t+1}\|^2-\|\boldsymbol{z}_t\|^2+2\langle \boldsymbol{z}_t-\boldsymbol{z}_{t+1}, \boldsymbol{y}_{t+1}\rangle \\ &= \|\boldsymbol{z}_{t+1}\|^2-\|\boldsymbol{z}_t\|^2+2\langle \boldsymbol{z}_t-\boldsymbol{z}_{t+1}, \boldsymbol{y}_{t+1}\rangle.\end{aligned}$$

□

Proof of Theorem 1. As there is no randomness in the algorithm's choices, the regret is simply $\sum_{t=1}^{T}\langle \boldsymbol{u}_t, \boldsymbol{z}_t-\boldsymbol{y}_t\rangle$. We know from Lemma 1 that

$$\begin{aligned}2\eta\sum_{t=1}^{T}\langle \boldsymbol{u}_t, \boldsymbol{z}_t-\boldsymbol{y}_t\rangle &\overset{(B.1)}{\leq} \sum_{t=1}^{T}\left\|\boldsymbol{y}_t-\boldsymbol{z}_t\right\|^2-\left\|\boldsymbol{y}_{t+1}-\boldsymbol{z}_t\right\|^2+\eta^2\|\boldsymbol{u}_t\|^2 \\ &= \sum_{t=1}^{T}\left(\left\|\boldsymbol{y}_t-\boldsymbol{z}_t\right\|^2-\left\|\boldsymbol{y}_{t+1}-\boldsymbol{z}_{t+1}\right\|^2\right)+\sum_{t=1}^{T}\left(\left\|\boldsymbol{y}_{t+1}-\boldsymbol{z}_{t+1}\right\|^2-\left\|\boldsymbol{y}_{t+1}-\boldsymbol{z}_t\right\|^2+\eta^2\|\boldsymbol{u}_t\|^2\right) \\ &\overset{(B.2)}{=} \sum_{t=1}^{T}\left(\left\|\boldsymbol{y}_t-\boldsymbol{z}_t\right\|^2-\left\|\boldsymbol{y}_{t+1}-\boldsymbol{z}_{t+1}\right\|^2\right)+\sum_{t=1}^{T}\left(\|\boldsymbol{z}_{t+1}\|^2-\|\boldsymbol{z}_t\|^2+2\langle \boldsymbol{z}_t-\boldsymbol{z}_{t+1}, \boldsymbol{y}_{t+1}\rangle+\eta^2\|\boldsymbol{u}_t\|^2\right) \\ &= \left\|\boldsymbol{y}_1-\boldsymbol{z}_1\right\|^2-\left\|\boldsymbol{y}_{T+1}-\boldsymbol{z}_{T+1}\right\|^2+\|\boldsymbol{z}_{T+1}\|^2-\|\boldsymbol{z}_1\|^2+\sum_{t=1}^{T}2\left\langle \boldsymbol{z}_t-\boldsymbol{z}_{t+1}, \boldsymbol{y}_{t+1}\right\rangle+\eta^2\|\boldsymbol{u}_t\|^2. \quad \text{(B.3)}\end{aligned}$$

As $\boldsymbol{y}_1=0$, by dropping $-\left\|\boldsymbol{y}_{T+1}-\boldsymbol{z}_{T+1}\right\|^2$, choosing $\boldsymbol{z}_{T+1}=\boldsymbol{z}_T$, and using $\boldsymbol{u}_t=w_{t,r_t}\boldsymbol{e}(r_t)$ the bound (B.3) becomes

$$2\eta\sum_{t=1}^{T}\langle \boldsymbol{u}_t, \boldsymbol{z}_t-\boldsymbol{y}_t\rangle \;\leq\; \|\boldsymbol{z}_T\|^2+\sum_{t=1}^{T-1}2\left\langle \boldsymbol{z}_t-\boldsymbol{z}_{t+1}, \boldsymbol{y}_{t+1}\right\rangle+\eta^2\sum_{t=1}^{T}w_{t,r_t}^2.$$

Dividing both sides of the inequality by 2η completes the proof. □

Appendix C. A Quantized Relaxed Projection

In this appendix, we prove that the quantized projection is a relaxed projection. We first formally define a "closer to" relation together with its properties. Then, we establish a few "closer to" relations and two preliminary lemmas that will be useful to prove the main result that $P^{q}_{\mathcal{V}}$ is a relaxed projection.

Definition 7 (Closer to) Consider two points $\boldsymbol{x}, \boldsymbol{z}\in\mathbb{R}^N$ and a set $\mathcal{V}\subseteq\mathbb{R}^N$. We say that $\boldsymbol{x}$ is closer to $\mathcal{V}$ than $\boldsymbol{z}$, which we denote by $\boldsymbol{x}\leftarrow_{\mathcal{V}}\boldsymbol{z}$, if and only if $\|\boldsymbol{x}-\boldsymbol{y}\|\leq\|\boldsymbol{z}-\boldsymbol{y}\|$ for any $\boldsymbol{y}\in\mathcal{V}$.

By Definition 7, relaxed projections onto $\mathcal{V}$ are functions that map closer to $\mathcal{V}$. For any $\boldsymbol{x}\in\mathbb{R}^N$, we have $P^{R}_{\mathcal{V}}(\boldsymbol{x})\leftarrow_{\mathcal{V}}\boldsymbol{x}$. The update in OGD with relaxed projection (see Section 3) can be rewritten using this definition as $\boldsymbol{y}_{t+1}\leftarrow_{\mathcal{V}}\boldsymbol{y}_t+\eta\boldsymbol{u}_t$.

Lemma 3 (Properties of "Closer to") The following properties of the relation "closer to" hold for $\boldsymbol{z}_k\in\mathbb{R}^N$ with $k\in\mathbb{N}$, $\lambda\in[0,1]$, and $\mathcal{V},\mathcal{W}\subset\mathbb{R}^N$ such that $\mathcal{V}\subset\mathcal{W}$:

$$\begin{aligned}&\text{Reflexivity:} && \boldsymbol{z}_1\leftarrow_{\mathcal{V}}\boldsymbol{z}_1. && \text{(C.1)}\\ &\text{Hereditary:} && \boldsymbol{z}_1\leftarrow_{\mathcal{W}}\boldsymbol{z}_2 \implies \boldsymbol{z}_1\leftarrow_{\mathcal{V}}\boldsymbol{z}_2. && \text{(C.2)}\\ &\text{Transitivity:} && \boldsymbol{z}_1\leftarrow_{\mathcal{V}}\boldsymbol{z}_2 \text{ and } \boldsymbol{z}_2\leftarrow_{\mathcal{V}}\boldsymbol{z}_3 \implies \boldsymbol{z}_1\leftarrow_{\mathcal{V}}\boldsymbol{z}_3. && \text{(C.3)}\\ &\text{Convexity:} && \boldsymbol{z}_1\leftarrow_{\mathcal{V}}\boldsymbol{z}_3 \text{ and } \boldsymbol{z}_2\leftarrow_{\mathcal{V}}\boldsymbol{z}_3 \implies \lambda\boldsymbol{z}_1+(1-\lambda)\boldsymbol{z}_2\leftarrow_{\mathcal{V}}\boldsymbol{z}_3. && \text{(C.4)}\\ &\text{Continuity:} && \boldsymbol{z}_k\leftarrow_{\mathcal{V}}\boldsymbol{z}_0 \implies \lim_{k\to\infty}\boldsymbol{z}_k\leftarrow_{\mathcal{V}}\boldsymbol{z}_0. && \text{(C.5)}\end{aligned}$$

Proof. Reflexivity and hereditary are trivial. Let $\boldsymbol{y} \in \mathcal{V}$.

$$\begin{aligned}
&\text{Transitivity:} && \|z_1 - \boldsymbol{y}\| \le \|z_2 - \boldsymbol{y}\| \le \|z_3 - \boldsymbol{y}\|. \\
&\text{Convexity:} && \|\lambda z_1 + (1-\lambda) z_2 - \boldsymbol{y}\| \;=\; \|\lambda(z_1 - \boldsymbol{y}) + (1-\lambda)(z_2 - \boldsymbol{y})\| \;\le\; \lambda\|z_1 - \boldsymbol{y}\| + (1-\lambda)\|z_2 - \boldsymbol{y}\| \\
& && \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \le\; \lambda\|z_3 - \boldsymbol{y}\| + (1-\lambda)\|z_3 - \boldsymbol{y}\| = \|z_3 - \boldsymbol{y}\|. \\
&\text{Continuity:} && \left\| \lim_{k\to\infty} z_k - \boldsymbol{y} \right\| = \lim_{k\to\infty} \|z_k - \boldsymbol{y}\| \le \lim_{k\to\infty} \|z_0 - \boldsymbol{y}\| = \|z_0 - \boldsymbol{y}\|\,.
\end{aligned}$$

□

Lemma 4 (Euclidean projection maps closer to $\mathcal{V}$) Let $\mathcal{V} \subset \mathbb{R}^N$ be non empty, closed and convex. For any $z \in \mathbb{R}^N$, the Euclidean projection $P_{\mathcal{V}}(z) := \operatorname{argmin}_{\boldsymbol{y}\in\mathcal{V}} \|z - \boldsymbol{y}\|$ maps closer to $\mathcal{V}$.

For later use, we establish a few "closer to" relations in Lemma 5 and two preliminary Lemmas 6 and 7.

Lemma 5 Recall that $\mathcal{Y}$ is the capped simplex (1). For any z in $\mathbb{R}^N$, the following "closer to" $\mathcal{Y}$ relations hold:

$$(z)^+ \leftarrow_{\mathcal{Y}} z. \tag{C.6}$$

$$\text{For any } \boldsymbol{b} \in \mathbb{R}^N \text{ such that } \boldsymbol{b} \ge \boldsymbol{s}: \qquad \min\{z, \boldsymbol{b}\} \leftarrow_{\mathcal{Y}} z. \tag{C.7}$$

$$\text{For any } \rho \in \mathbb{R}^+ \text{ such that } \sum_{n=1}^{N}(z_n - \rho) \ge C \text{ or } \rho = 0: \qquad z - \rho\mathbf{1} \leftarrow_{\mathcal{Y}} z. \tag{C.8}$$

Proof. Taking into account the different bounds that define the capped simplex $\mathcal{Y}$, we define three sets:

$$\mathcal{Y}_1 := \left\{\boldsymbol{y} \in \mathbb{R}^N \;\middle|\; 0 \le y_n\right\}, \qquad \mathcal{Y}_2 := \left\{\boldsymbol{y} \in \mathbb{R}^N \;\middle|\; y_n \le b_n\right\}, \qquad \mathcal{Y}_3 := \left\{\boldsymbol{y} \in \mathbb{R}^N \;\middle|\; \sum_{n=1}^{N} y_n \le C\right\}.$$

We know from Lemma 4 that the Euclidean projection over a set maps closer to this set, hence, all properties established in Lemma 3 hold for the Euclidean projection. By composing Euclidean projections over $\mathcal{Y}_1$ along all $n \in [N]$, (C.6) follows from transitivity (C.3) and the hereditary property (C.2) since $\mathcal{Y} \subset \mathcal{Y}_1$. Similarly, we can obtain (C.7), but this time considering Euclidean projections over $\mathcal{Y}_2 \supset \mathcal{Y}$.

To prove (C.8), we distinguish three cases:

- for $\rho = 0$, (C.8) is reflexivity (C.1);
- for $\rho \in \mathbb{R}^+$ such that $\sum_{n=1}^{N}(z_n - \rho) = C$, the point $z - \rho\mathbf{1}$ is the Euclidean projection of z over $\mathcal{Y}_3$. By Lemma 4, $P_{\mathcal{Y}_3}(z) \leftarrow_{\mathcal{Y}_3} z$. Then, (C.8) follows from the hereditary property (C.2) since $\mathcal{Y} \subset \mathcal{Y}_3$;
- for $\rho \in \mathbb{R}^+$ such that $\sum_{n=1}^{N}(z_n - \rho) > C$, there exist $\lambda \in [0,1]$ and $\rho^* \in \mathbb{R}^+$ such that $\rho = \lambda\rho^*$ and $\sum_{n=1}^{N}(z_n - \rho^*) = C$, implying that $z - \rho^*\mathbf{1} \leftarrow_{\mathcal{Y}} z$ (from second case). Then, by reflexivity (C.1) and convexity (C.4), we have (C.8):

$$\left.\begin{array}{r} z - \rho^*\mathbf{1} \leftarrow_{\mathcal{Y}} z \\ \text{(C.1)}: \quad z \leftarrow_{\mathcal{Y}} z \end{array}\right\} \overset{\text{(C.4)}}{\Rightarrow} \quad \lambda(z - \rho^*\mathbf{1}) + (1-\lambda)z \leftarrow_{\mathcal{Y}} z \quad \Leftrightarrow \quad z - \lambda\rho^*\mathbf{1} \leftarrow_{\mathcal{Y}} z \quad \Leftrightarrow \quad z - \rho\mathbf{1} \leftarrow_{\mathcal{Y}} z.$$

□

Lemma 6 For any point $z \in \mathbb{R}^N$ and any positive scalar $\rho \in \mathbb{R}^+$, the following holds:

$$((z)^+ - \rho\mathbf{1})^+ = (z - \rho\mathbf{1})^+. \tag{C.9}$$

Proof. It is trivial to prove the equality by considering all cases. □

Lemma 7 For any point $z \in \mathbb{R}^N$ and any positive scalars $\rho, \varepsilon \in \mathbb{R}^+$ such that $0 < \varepsilon < \rho$, the following implication holds:

$$\sum_{n=1}^{N}(z_n - \rho)^+ \ge C > 0 \quad \Rightarrow \quad \sum_{n=1}^{N}(z_n - \rho + \varepsilon)^+ \ge C + \varepsilon. \tag{C.10}$$

Proof. Since $\sum_{n=1}^{N}(z_n-\rho)^+ > 0$, there exists an n_+ in $[N]$ such that $z_{n_+}-\rho > 0$. Therefore, there is at least a term in $\sum_{n=1}^{N}(z_n-\rho+\varepsilon)^+$ to which ε is added, which implies the right-hand inequality in (C.10). □

The following theorem states that the projection performed in Line 3 of Algorithm 3 maps closer to $\tilde{\mathcal{V}}$. We have formally verified it in Lean[2] (see lean4web link).

Theorem 8. *For any point z in $\mathbb{R}^N$ and any scalar $\rho \in \mathbb{R}^+$ such that $\sum_{n=1}^{N}(z_n-\rho)^+ \geq C > 0$ or $\rho = 0$, the following holds:*

$$(z-\rho\mathbf{1})^+ \leftarrow_y z. \tag{C.11}$$

Proof. The case $\rho = 0$ boils down to (C.6). For $\rho > 0$, the idea to prove Theorem 8 is to use Lemma 5 infinitely often in small steps. We prove (C.11) by showing first that, for any $\varepsilon \in \mathbb{R}^+$ such that $0 < \varepsilon < \rho$,

$$(z-(\rho-\varepsilon)\mathbf{1})^+ \leftarrow_y z, \tag{C.12}$$

then the statement follows by invoking continuity (C.5).

We prove (C.12) by induction. Choose a large $l \in \mathbb{N}$ such that $\frac{\rho-\varepsilon}{l} \leq \frac{\varepsilon}{N}$ (for a reason that will become clear later), then show by induction that, for any $k \in \mathbb{N}$ such that $k \leq l$,

$$(z-(\rho-\varepsilon)\frac{k}{l}\mathbf{1})^+ \leftarrow_y z, \tag{C.13}$$

and obtain (C.12) with $k = l$.

Base case ($k = 0$): (C.13) follows from Lemma 5 as it is simply (C.6).
Induction hypothesis: assume (C.13) holds for $k = r$.
Induction step: $k = r+1 \leq l$,

$$\begin{aligned}\left(z-(\rho-\varepsilon)\frac{r+1}{l}\mathbf{1}\right)^+ &\overset{(C.9)}{=} \left(\left(z-(\rho-\varepsilon)\frac{r}{l}\mathbf{1}\right)^+ - \frac{\rho-\varepsilon}{l}\mathbf{1}\right)^+ \overset{(C.6)}{\leftarrow_y} \left(z-(\rho-\varepsilon)\frac{r}{l}\mathbf{1}\right)^+ - \frac{\rho-\varepsilon}{l}\mathbf{1} \\ &\overset{(C.8)}{\leftarrow_y} \left(z-(\rho-\varepsilon)\frac{r}{l}\mathbf{1}\right)^+ \\ &\leftarrow_y z,\end{aligned} \tag{C.14}$$

where the latter follows from the induction hypothesis ((C.13) with $k = r$). To complete the induction step, it remains to verify that the condition for (C.8) is satisfied, otherwise (C.14) will not hold. Recall that l is chosen such that $\frac{\rho-\varepsilon}{l} \leq \frac{\varepsilon}{N}$. Then, we have

$$\sum_{n=1}^{N}\left[\left(z_n-(\rho-\varepsilon)\frac{r}{l}\right)^+ - \frac{\rho-\varepsilon}{l}\right] \geq \sum_{n=1}^{N}\left[\left(z_n-(\rho-\varepsilon)\frac{r}{l}\right)^+ - \frac{\varepsilon}{N}\right] = \sum_{n=1}^{N}\left(z_n-(\rho-\varepsilon)\frac{r}{l}\right)^+ - \varepsilon \geq \sum_{n=1}^{N}(z_n-(\rho-\varepsilon))^+ - \varepsilon$$

where we used $r \leq l$ to write the last inequality. According to the statement, we have $\sum_{n=1}^{N}(z_n-\rho)^+ \geq C > 0$. Therefore, by the implication (C.10), the condition for (C.8) holds and the proof is completed. □

Proof of Theorem 2. To say that $P^q_{\tilde{\mathcal{V}}}$ is a relaxed projection onto $\tilde{\mathcal{V}}$ means that $\forall \tilde{\boldsymbol{x}} \in \mathbb{R}^N : P^q_{\tilde{\mathcal{V}}}(\tilde{\boldsymbol{x}}) \leftarrow_{\tilde{\mathcal{V}}} \tilde{\boldsymbol{x}}$. By transitivity (C.3), it suffices to show that each step maps closer to $\tilde{\mathcal{V}}$. Line 1 of Algorithm 3 projects onto $\mathbf{0} \leq \tilde{\boldsymbol{x}} \leq \tilde{\boldsymbol{s}}$, mapping closer by Lemma 5 ((C.6) and (C.7)). The update $\tilde{\boldsymbol{x}} \leftarrow (\tilde{\boldsymbol{x}}-\mathbf{1})^+$ in Line 3 corresponds to the map in Theorem 8 ($\rho = 1$), which maps closer to $\tilde{\mathcal{V}}$ whenever the while loop condition holds. □

[2] An unreviewed, shortened version of the Lean proof is available at lean4web link

Appendix D. Regret of qOGD

Lemma 8 (Unbiasedness dynamic regret) For any point $\boldsymbol{z}_t \in \mathbb{R}^N$, utility $\boldsymbol{u}_t \in \mathbb{R}^N$, and any independent random variables $\boldsymbol{Y}_t$ and $\boldsymbol{U}_t$, with $\mathbb{E}[\boldsymbol{U}_t] = \boldsymbol{u}_t$, we have

$$\mathbb{E}\left[\sum_{t=1}^{T}\langle \boldsymbol{u}_t, \boldsymbol{z}_t - \boldsymbol{Y}_t\rangle\right] = \mathbb{E}\left[\sum_{t=1}^{T}\langle \boldsymbol{U}_t, \boldsymbol{z}_t - \boldsymbol{Y}_t\rangle\right]. \tag{D.1}$$

Proof. By the linearity of expectation, we consider each term individually. For any t, we use the law of total expectation to write

$$\mathbb{E}[\langle \boldsymbol{U}_t, \boldsymbol{z}_t - \boldsymbol{Y}_t\rangle] = \mathbb{E}\Big[\mathbb{E}[\langle \boldsymbol{U}_t, \boldsymbol{z}_t - \boldsymbol{Y}_t\rangle \mid \boldsymbol{Y}_t]\Big] = \mathbb{E}\Big[\langle \mathbb{E}[\boldsymbol{U}_t \mid \boldsymbol{Y}_t], \boldsymbol{z}_t - \boldsymbol{Y}_t\rangle\Big] \overset{\text{ind.}}{=} \mathbb{E}\Big[\langle \mathbb{E}[\boldsymbol{U}_t], \boldsymbol{z}_t - \boldsymbol{Y}_t\rangle\Big] = \mathbb{E}[\langle \boldsymbol{u}_t, \boldsymbol{z}_t - \boldsymbol{Y}_t\rangle].$$

Summing over $t = 1$ to T yields the result. □

Proof of Theorem 4. We want to bound the regret of Algorithm 4 which can be written as follows:

$$R_T\left(\text{qOGD}, (\boldsymbol{z}_t)_{t=1}^T, (\boldsymbol{u}_t)_{t=1}^T\right) = \mathbb{E}\left[\sum_{t=1}^{T}\langle \boldsymbol{u}_t, \boldsymbol{z}_t - \boldsymbol{y}_t\rangle\right] = \varepsilon\mathbb{E}\left[\sum_{t=1}^{T}\left\langle \boldsymbol{u}_t, \tilde{\boldsymbol{z}}_t - (\tilde{\boldsymbol{f}}_t - \mathbf{1})^+\right\rangle\right] \tag{D.2}$$

where $\boldsymbol{z}_t = \varepsilon\tilde{\boldsymbol{z}}_t$ and $\boldsymbol{y}_t = \varepsilon(\tilde{\boldsymbol{f}}_t - \mathbf{1})^+$ (see Line 6 of Algorithm 4). The last expectation in (D.2) is the scaled regret. In order to leverage Theorem 1 that provides the regret guarantees of Algorithm 2, we define $\boldsymbol{U}_t = \frac{1}{\tilde{\eta}}\text{UR}(\tilde{\eta}w_{t,r_t})\boldsymbol{e}(r_t)$. Lines 4 and 5 of Algorithm 4 can then be rewritten using $\boldsymbol{U}_t$, namely, $\tilde{\boldsymbol{f}}_{t+1} \leftarrow P^q_{\tilde{\mathcal{V}}}\left(\tilde{\boldsymbol{f}}_t + \tilde{\eta}\boldsymbol{U}_t\right)$. This update step now resembles that of OGD with relaxed projection; see Algorithm 2, Line 4. It suffices then to consider $P^q_{\tilde{\mathcal{V}}}$ as relaxed projection, utility $\boldsymbol{U}_t$ instead of $\boldsymbol{u}_t$, and $\tilde{\eta}$ instead of η so that Theorem 1 applies to the cache representation $\tilde{\boldsymbol{f}}_t$ of Algorithm 4.

However, the scaled regret in (D.2) is expressed in terms of the feasible version of $\tilde{\boldsymbol{f}}_t$, that is $(\tilde{\boldsymbol{f}}_t - \mathbf{1})^+$. To rewrite the scaled regret in terms of $\tilde{\boldsymbol{f}}_t$, we use Lemma 8. Given that $\tilde{\boldsymbol{f}}_t$ does not depend on $\boldsymbol{U}_t$ in Algorithm 4, the random variables $(\tilde{\boldsymbol{f}}_t - \mathbf{1})^+$ and $\boldsymbol{U}_t$ are also independent. Also, we have that $\mathbb{E}[\boldsymbol{U}_t] = \boldsymbol{u}_t$ (by Definition 5), so Lemma 8 is indeed applicable. We can rewrite the scaled regret as follows:

$$\mathbb{E}\left[\sum_{t=1}^{T}\left\langle \boldsymbol{u}_t, \tilde{\boldsymbol{z}}_t - (\tilde{\boldsymbol{f}}_t - \mathbf{1})^+\right\rangle\right] \overset{\text{(D.1)}}{=} \mathbb{E}\left[\sum_{t=1}^{T}\left\langle \boldsymbol{U}_t, \tilde{\boldsymbol{z}}_t - (\tilde{\boldsymbol{f}}_t - \mathbf{1})^+\right\rangle\right] \le \mathbb{E}\left[\sum_{t=1}^{T}\left\langle \boldsymbol{U}_t, \tilde{\boldsymbol{z}}_t - \tilde{\boldsymbol{f}}_t\right\rangle\right] + \sum_{t=1}^{T} w_{t,r_t}, \tag{D.3}$$

where we used $\left(\tilde{\boldsymbol{f}}_t - \mathbf{1}\right)^+ \ge \tilde{\boldsymbol{f}}_t - \mathbf{1}$ and $\mathbb{E}[\langle \boldsymbol{U}_t, \mathbf{1}\rangle] = w_{t,r_t}$ to write the last inequality. We apply Theorem 1 to bound the last expectation in (D.3) (with $\tilde{\boldsymbol{z}}_t \in \tilde{\mathcal{V}}$) and rewrite (D.3) as follows:

$$\mathbb{E}\left[\sum_{t=1}^{T}\left\langle \boldsymbol{u}_t, \tilde{\boldsymbol{z}}_t - (\tilde{\boldsymbol{f}}_t - \mathbf{1})^+\right\rangle\right] \le \frac{1}{2\tilde{\eta}}\|\tilde{\boldsymbol{z}}_T\|^2 + \frac{1}{\tilde{\eta}}\sum_{t=1}^{T-1}\left\langle \tilde{\boldsymbol{z}}_t - \tilde{\boldsymbol{z}}_{t+1}, \tilde{\boldsymbol{f}}_{t+1}\right\rangle + \frac{\tilde{\eta}}{2}\mathbb{E}\left[\sum_{t=1}^{T}\left(\frac{1}{\tilde{\eta}}\text{UR}(\tilde{\eta}w_{t,r_t})\right)^2\right] + \sum_{t=1}^{T} w_{t,r_t}. \tag{D.4}$$

For the path length term, we use $\tilde{\boldsymbol{z}}_t - \tilde{\boldsymbol{z}}_{t+1} \le (\tilde{\boldsymbol{z}}_t - \tilde{\boldsymbol{z}}_{t+1})^+$ and $\tilde{\boldsymbol{f}}_{t+1} \le (\tilde{\boldsymbol{f}}_{t+1} - \mathbf{1})^+ + \mathbf{1}$. Then,

$$\left\langle \tilde{\boldsymbol{z}}_t - \tilde{\boldsymbol{z}}_{t+1}, \tilde{\boldsymbol{f}}_{t+1}\right\rangle \le \left\langle (\tilde{\boldsymbol{z}}_t - \tilde{\boldsymbol{z}}_{t+1})^+, (\tilde{\boldsymbol{f}}_{t+1} - \mathbf{1})^+ + \mathbf{1}\right\rangle. \tag{D.5}$$

Applying the expectation and variance bound of unbiased rounding (see Definition 5) yields:

$$\frac{1}{2\tilde{\eta}}\mathbb{E}\left[\text{UR}(\tilde{\eta}w_{t,r_t})^2\right] \le \frac{\tilde{\eta}^2 w_{t,r_t}^2 + 1/4}{2\tilde{\eta}} = \frac{\tilde{\eta}\, w_{t,r_t}^2}{2} + \frac{1}{8\tilde{\eta}}. \tag{D.6}$$

Combining (D.2) and (D.4) to (D.6) and unscaling with $\boldsymbol{z}_t = \varepsilon\tilde{\boldsymbol{z}}_t$, $\boldsymbol{y}_t = \varepsilon(\tilde{\boldsymbol{f}}_t - \mathbf{1})^+$ and $\eta = \varepsilon\tilde{\eta}$ yields:

$$R_T\left(\text{qOGD}, (\boldsymbol{z}_t)_{t=1}^T, (\boldsymbol{u}_t)_{t=1}^T\right) \le \frac{1}{2\eta}\|\boldsymbol{z}_T\|^2 + \frac{1}{\eta}\sum_{t=1}^{T-1}\langle (\boldsymbol{z}_t - \boldsymbol{z}_{t+1})^+, \boldsymbol{y}_{t+1} + \varepsilon\mathbf{1}\rangle + \frac{\eta}{2}\sum_{t=1}^{T} w_{t,r_t}^2 + \varepsilon\sum_{t=1}^{T}\left(w_{t,r_t} + \frac{\varepsilon}{8\eta}\right) \tag{D.7}$$

$$\le \frac{1}{2\eta}\|\boldsymbol{z}_T\|^2 + \frac{1}{\eta}\sum_{t=1}^{T-1}\max_{\boldsymbol{y}\in\mathcal{Y}}\langle (\boldsymbol{z}_t - \boldsymbol{z}_{t+1})^+, \boldsymbol{y} + \varepsilon\mathbf{1}\rangle + \frac{\eta}{2}\sum_{t=1}^{T} w_{t,r_t}^2 + \varepsilon\sum_{t=1}^{T}\left(w_{t,r_t} + \frac{\varepsilon}{8\eta}\right), \tag{D.8}$$

where the final step follows from $\boldsymbol{y}_{t+1} \in \mathcal{Y}$, completing the proof. □

Appendix E. Asymptotically Optimal Regret against S-Switch comparators

In this section, we analyze the regret against the comparator class $\mathcal{S}_S \subseteq \mathcal{Y}^T$ of S-piecewise static cache policies, i.e., sequences of cache states that switch allocations S times.

Lemma 9 (Lower bound for regret against S-switch comparators) In the uniform-object-size setting ($\boldsymbol{s} = \mathbf{1}$) with hits ($w_{t,r_t} = 1$) and $C < N/2$, for any online caching algorithm $\mathcal{A}$, the worst-case regret against $\mathcal{S}_S$ satisfies asymptotically (as $T \to \infty$) with $S = o(T)$:

$$\sup_{\substack{(r_t)_{t=1}^T \in [N]^T \\ (z_t)_{t=1}^T \in \mathcal{S}_S}} R_T(\mathcal{A}, (z_t)_{t=1}^T, (\boldsymbol{e}(r_t))_{t=1}^T) \geq \Omega(\sqrt{C(S+1)T}). \tag{E.1}$$

Proof. Let $K = S+1$ and partition $\{1, \dots, T\}$ into K consecutive blocks $I_1, \dots, I_K$ with lengths $L_k = |I_k| \in \{\lfloor T/K \rfloor, \lceil T/K \rceil\}$. Since $S = o(T)$, we have $\min_k L_k = \lfloor T/(S+1) \rfloor \to \infty$ as $T \to \infty$.

Apply the worst-case static regret lower bound of Paschos et al. [36, Corollary 1] within each block I_k (requires $C < N/2$). Specifically, for any online policy $\mathcal{A}$, the expected static regret over interval I_k with respect to a static cache configuration $z^{(k)}$ is at least $c\sqrt{CL_k}$ for some constant $c > 0$ and all sufficiently large L_k.

We define the global comparator sequence $(z_t)_{t=1}^T$ by setting $z_t = z^{(k)}$ for all $t \in I_k$. Because switches occur exclusively at block boundaries, the configuration changes at most $K - 1 = S$ times, ensuring $(z_t)_{t=1}^T \in \mathcal{S}_S$.

Summing the static regret lower bounds across all K blocks, we obtain

$$\sup_{\substack{(r_t)_{t=1}^T \in [N]^T \\ (z_t)_{t=1}^T \in \mathcal{S}_S}} R_T(\mathcal{A}, (z_t)_{t=1}^T, (\boldsymbol{e}(r_t))_{t=1}^T) \geq \sum_{k=1}^K c\sqrt{CL_k} \geq c\sum_{k=1}^K \sqrt{C\left\lfloor \frac{T}{K} \right\rfloor} = cK\sqrt{C\left\lfloor \frac{T}{K} \right\rfloor} = \Omega(\sqrt{CKT}). \tag{E.2}$$

Substituting $K = S + 1$ yields the bound $\Omega(\sqrt{C(S+1)T})$ and completes the proof. □

Corollary 2 (Regret of qOGD against S-switch comparators)**.** *Let the cache state $\mathbf{y}_t$ be generated by Algorithm 4 with scaled step size $\tilde{\eta} \in \mathbb{R}^+$ and granularity*

$$\varepsilon = \sqrt{\frac{s_{max} C(2S+1)}{2\tilde{\eta}T\left(\frac{\tilde{\eta} w_{max}^2}{2} + w_{max} + \frac{1}{8\tilde{\eta}}\right)}}, \tag{E.3}$$

where $w_{max} \geq \max_{t\in[T]} w_{t,r_t}$ and $s_{max} \geq \max_n s_n$. Then, for any comparator sequence $(z_t)_{t=1}^T \in \mathcal{S}_S$:

$$R_T(qOGD, (z_t)_{t=1}^T, (w_{t,r_t}\boldsymbol{e}(r_t))_{t=1}^T) \leq \sqrt{\left(w_{max}^2 + \frac{2w_{max}}{\tilde{\eta}} + \frac{1}{4\tilde{\eta}^2}\right) T s_{max} C(2S+1)} + \frac{CS}{\tilde{\eta}}. \tag{E.4}$$

Proof. We apply Theorem 4. Using Hölder's inequality with dual norms (ℓ_1, ℓ_∞), the path-length term is bounded by:

$$\begin{aligned}
\max_{\mathbf{y}\in\mathcal{Y}} \langle (\boldsymbol{z}_t - \boldsymbol{z}_{t+1})^+, \mathbf{y} + \varepsilon\mathbf{1}\rangle &\leq \left\|(\boldsymbol{z}_t - \boldsymbol{z}_{t+1})^+\right\|_1 \max_{\mathbf{y}\in\mathcal{Y}} \|\mathbf{y} + \varepsilon\mathbf{1}\|_\infty \\
&\leq (s_{\max} + \varepsilon)\left\|(\boldsymbol{z}_t - \boldsymbol{z}_{t+1})^+\right\|_1 \\
&\leq (s_{\max} + \varepsilon)C \text{ if } \boldsymbol{z}_t \neq \boldsymbol{z}_{t+1}, \text{ else } 0
\end{aligned} \tag{E.5}$$

since $\|\mathbf{y}\|_\infty \leq s_{\max}$ and a single switch can evict at most C amount of data (i.e., $\|(\boldsymbol{z}_t - \boldsymbol{z}_{t+1})^+\|_1 \leq C$). Summing over all rounds bounds the cumulative path length by $(s_{\max} + \varepsilon)CS$. Moreover, for any comparator $\boldsymbol{z}_T \in \mathcal{Y}$, we have $\boldsymbol{z}_T \in [0, s_{\max}]^N$ and $\sum_{n=1}^N z_{T,n} \leq C$. Thus, we can bound the squared norm as $\|\boldsymbol{z}_T\|^2 = \sum_{n=1}^N z_{T,n}^2 \leq \sum_{n=1}^N s_{\max} z_{T,n} \leq s_{\max} C$ [36]. Substituting these bounds into Theorem 4 with $\eta = \varepsilon\tilde{\eta}$ and $w_{t,r_t} \leq w_{\max}$ yields:

$$\begin{aligned}
R_T(\text{qOGD}, (z_t)_{t=1}^T, (w_{t,r_t}\boldsymbol{e}(r_t))_{t=1}^T) &\leq \frac{s_{\max}C}{2\eta} + \frac{(s_{\max}+\varepsilon)CS}{\eta} + \frac{\eta T w_{\max}^2}{2} + \varepsilon T w_{\max} + T\frac{\varepsilon^2}{8\eta} \\
&= \frac{s_{\max}C(2S+1)}{2\varepsilon\tilde{\eta}} + \varepsilon T\left(\frac{\tilde{\eta}w_{\max}^2}{2} + w_{\max} + \frac{1}{8\tilde{\eta}}\right) + \frac{CS}{\tilde{\eta}}.
\end{aligned} \tag{E.6}$$

The right-hand side is minimized by choosing ε as given in (E.3). Substituting this value into (E.6) gives the final bound in (E.4). □

Remark 2 Fix C, $s_{\max}$, $w_{\max}$ and set $\tilde{\eta} = 1$. If $S = o(T)$ asymptotically , the term CS in (E.4) is negligible compared to $O(\sqrt{s_{\max} C(S+1)T})$. Consequently, qOGD has regret $O(\sqrt{s_{\max} C(S+1)T})$ against $\mathcal{S}_S$. In particular, in the unit-size, unit-reward setting ($s_{\max} = 1, w_{\max} = 1$), and under the assumptions of Lemma 9, this matches the $\Omega(\sqrt{C(S+1)T})$ lower bound in Lemma 9 up to constant factors. We note that the choice of ε requires knowledge of the horizon T and of the number of comparator switches S. Additionally, when $w_{t,r_t} = 1$, the variance term in ε vanishes as noted in Remark 1.

Appendix F. Proof of Items 1–3 of Theorem 7

1. Unbiasedness: The probability that $V_n \le \frac{y_{t,n}}{s_n}$ is $\frac{y_{t,n}}{s_n}$. Thus, by Definition 6, $\mathbb{E}\left[Y_{t,n} \,\middle|\, \boldsymbol{y}_t\right] = s_n \cdot \frac{y_{t,n}}{s_n} + 0\left(1 - \frac{y_{t,n}}{s_n}\right) = y_{t,n}$. Therefore, $\mathbb{E}\left[\boldsymbol{Y}_t \,\middle|\, \boldsymbol{y}_t\right] = \boldsymbol{y}_t$.

2. Feasibility: The limits $\mathbf{0} \le \boldsymbol{Y}_t \le \boldsymbol{s}$ hold by Definition 6. Since $\mathbb{E}\left[Y_{t,n} \,\middle|\, \boldsymbol{y}_t\right] = y_{t,n}$, by linearity of expectation and $\boldsymbol{y}_t \in \mathcal{Y}$, we have $\mathbb{E}\left[\sum_{n=1}^{N} Y_{t,n} \,\middle|\, \boldsymbol{y}_t\right] = \sum_{n=1}^{N} y_{t,n} \le C$.

3. Regret: We use linearity of expectation, the law of total expectation, and $\mathbb{E}\left[\boldsymbol{Y}_t \,\middle|\, \boldsymbol{y}_t\right] = \boldsymbol{y}_t$ to write

$$\mathbb{E}\left[\sum_{t=1}^{T} \langle \boldsymbol{u}_t, \boldsymbol{z}_t - \boldsymbol{Y}_t \rangle\right] = \sum_{t=1}^{T} \mathbb{E}\left[\mathbb{E}\left[\langle \boldsymbol{u}_t, \boldsymbol{z}_t - \boldsymbol{Y}_t \rangle \,\middle|\, \boldsymbol{y}_t\right]\right] = \sum_{t=1}^{T} \mathbb{E}\left[\left\langle \boldsymbol{u}_t, \boldsymbol{z}_t - \mathbb{E}\left[\boldsymbol{Y}_t \,\middle|\, \boldsymbol{y}_t\right] \right\rangle\right] = \mathbb{E}\left[\sum_{t=1}^{T} \langle \boldsymbol{u}_t, \boldsymbol{z}_t - \boldsymbol{y}_t \rangle\right].$$