Memory-efficient Online Caching Policies with Regret Guarantees

Xufeng Zhang, Sara Alouf, Giovanni Neglia
Inria, Université Côte d'Azur, France

***Abstract*—Online learning algorithms provide robust performance in caching problems but require substantial memory to store per-file historical data, limiting their scalability to large-catalog systems. To overcome this challenge, we propose a dimensionality reduction algorithm based on the Follow-the-Perturbed-Leader framework and the Johnson-Lindenstrauss lemma. Our method significantly reduces memory consumption while preserving sublinear regret, making it well-suited for caching under resource constraints. Experiments on both synthetic and real-world traces demonstrate its advantages over other memory-efficient approaches.**

***Index Terms*—caching, regret, Follow-the-Perturbed-Leader, Johnson-Lindenstrauss lemma.**

I. Introduction

Caching plays a crucial role in optimizing content delivery by temporarily storing files for faster retrieval [1]. In some applications, content catalogs expand to billions of files, posing significant challenges for cache management [2]. In Content Delivery Networks, this problem can be exacerbated as large files are often split into multiple chunks for efficient caching [3]. Addressing these challenges is key to improving caching efficiency.

The core of caching lies in determining which items should be stored, for how long, and when should they be replaced. For this type of problem, classic caching policies such as Least Frequently Used (LFU) and Least Recently Used (LRU) have been proposed and remain widely used today. LFU prioritizes file retention based on access frequency, while LRU operates by evicting the least recently accessed files [4]. For different caching policies/algorithms, we can compare their performance by the *regret*, which is the difference of the cost between them and the optimal static strategy cost. Considering an adversary, which models the worst-case request pattern, both LFU and LRU exhibit a regret of $\Omega(T)$, indicating that their performance can degrade significantly under request patterns lacking statistical regularity [5]. This suggests that these classical policies may be suboptimal in environments where robustness to arbitrary or adversarial request sequences is required, highlighting the need for caching policies that perform well under diverse and potentially unpredictable access patterns.

To achieve robust performance without relying on assumptions about request patterns, online caching algorithms based on Online Linear Optimization (OLO) have been proposed [6]. These algorithms aim to minimize regret, ensuring that over a sufficiently long time horizon, their time-average regret approaches zero, a property known as no-regret learning. A prominent example is the caching algorithm based on Online Gradient Ascent (OGA), which achieves a sublinear regret of $\mathcal{O}(\sqrt{T})$ [6]. Additionally, online learning approaches such as Follow-the-Leader (FTL) have been widely applied to caching problems. Variants such as Follow-the-Regularized-Leader (FTRL) and Follow-the-Perturbed-Leader (FTPL) enhance stability over FTL [7], [8]: FTRL adds a regularization term to smooth updates, while FTPL introduces a perturbation term to incorporate randomness.

Although online caching algorithms guarantee sublinear regret compared to classic policies, they often require significant memory. Specifically, traditional online caching algorithms must store per-file historical data, causing their memory requirements to scale with the catalog size, N. When N is large, these methods may become impractical due to memory constraints. In contrast, caching algorithms like LRU and TinyLFU [9] operate with minimal memory but suffer from linear regret. To the best of our knowledge, the only attempt to reduce the memory requirements of online caching algorithms is [10], which proposes an FTPL-based caching policy that processes only a subset of the request stream.

Our objective in this paper is to propose an online caching algorithm that achieves both sublinear regret and a reduced memory footprint. A classic approach to dimensionality reduction is based on a result by Johnson and Lindenstrauss [11], which shows that points in an Euclidean space can be projected into a lower-dimensional subspace while approximately preserving pairwise distances. This result is widely known in the literature as the Johnson–Lindenstrauss (JL) lemma, e.g., [12]. Our contributions in this paper are as follows:

- We propose an algorithm, FTPL-JL, that combines FTPL with the JL lemma for dimensionality reduction to reduce memory usage.

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-19-P3IA-0002, the European Network of Excellence dAIEDGE under Grant Agreement Nr. 101120726, and the EU HORIZON MSCA 2023 DN project FINALITY (G.A. 101168816).

ISBN 978-3-903176-72-0

- We prove that FTPL-JL achieves sublinear regret.
- We evaluate FTPL-JL experimentally on both synthetic and real-world traces and show that it outperforms the approach in [10], which also enjoys regret guarantees, while remaining competitive with other memory-efficient algorithms that do not provide regret guarantees like TinyLFU.

In the following, we introduce the problem we consider in Section II and elaborate on our algorithm in Section III. We analyze its theoretical guarantees in Section IV, and experimentally evaluate its performance against other caching algorithms in Section V. Finally, we conclude our paper in Section VI.

II. Problem Description and Background

We consider a single-cache problem where there are N possible files and the cache has a limited capacity C. Following the usual convention in the online learning literature, we consider a time horizon T. For caching policies, one may consider that a cost is incurred whenever a requested file is not present in the cache, or alternatively that a reward is accrued when the cache is storing the requested file. We now introduce the main definitions and notations that we use.

Request. Let g_t be the index of the file requested at time t, $g_t \in [N] \triangleq \{1, \dots, N\}$. It is helpful to represent the request by a vector $\boldsymbol{r}_t \in \{0,1\}^N$, such that all entries are 0 except for the g_t-th entry, which is 1, in other words $r_{t,g_t} = 1$.

Popularity. It is represented by the vector $\boldsymbol{n}_t$ that keeps track of the number of requests for each of the N files. Popularity in this case is updated as follows:

$$\boldsymbol{n}_t = \boldsymbol{n}_{t-1} + \boldsymbol{r}_t \ . \tag{1}$$

Popularity estimate. We consider that the popularity vector may not be known exactly, but an estimate of it is available. Let $\hat{\boldsymbol{n}} \in \mathbb{R}^N$ denote the estimated popularity.

Cache State. Let $\mathcal{C}_t \subset [N]$ be the index set of files cached at time t, then $|\mathcal{C}_t| \leq C$. At request time t, the caching algorithm decides whether the newly requested file should be cached based on the estimated popularity $\hat{\boldsymbol{n}}$. A feasible cache state can be expressed as:

$$\boldsymbol{S}_t = \{(g_\tau : \hat{n}_{\tau,g_\tau}) \mid g_\tau \in \mathcal{C}_t,\ 0 < \tau \leq t\}, \tag{2}$$

where $\hat{n}_{\tau,g_\tau}$ is g_τ's estimated popularity at time τ.

Decision. At each step t, the online caching algorithm decides which files should be in cache. Let $\boldsymbol{x}_t \in \{0,1\}^N$ be the decision, and $\mathcal{X}$ be the feasible decision set. We must have $||\boldsymbol{x}_t||_1 = C$ for any t.

Reward. For each new request, the system provides a reward when the requested file is in the cache. Formally, the one-step reward can be defined as the scalar product

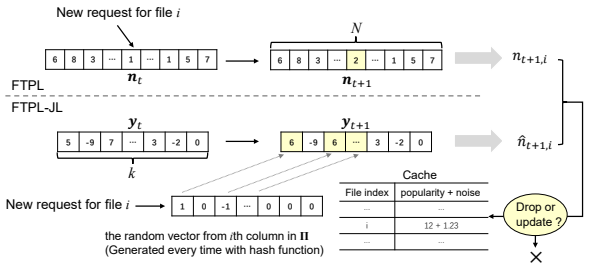

Fig. 1: Workflow of FTPL and FTPL-JL

of the decision $\boldsymbol{x}_t$ and the request $\boldsymbol{r}_t$, namely,

$$\text{reward}_t(\boldsymbol{x}_t) = \langle \boldsymbol{x}_t, \boldsymbol{r}_t \rangle \triangleq \sum_{i=1}^{N} x_{t,i} r_{t,i} = x_{t,g_t} \in \{0,1\}. \tag{3}$$

Regret. In the context of online algorithms, regret is a measure of the performance gap between the algorithm's decisions and the optimal static decision. Formally, for a sequence of decisions in T rounds, the regret is defined as the difference between the accumulated reward obtained by the optimal static policy and that of the online algorithm [8]. The regret can then be expressed as:

$$\begin{aligned} R_T &= \max_{\boldsymbol{x} \in \mathcal{X}} \sum_{t=1}^{T} \text{reward}_t(\boldsymbol{x}) - \sum_{t=1}^{T} \text{reward}_t(\boldsymbol{x}_t) \\ &\overset{(3),(1)}{=} \max_{\boldsymbol{x} \in \mathcal{X}} \langle \boldsymbol{x}, \boldsymbol{n}_T \rangle - \sum_{t=1}^{T} \langle \boldsymbol{x}_t, \boldsymbol{r}_t \rangle \ . \end{aligned} \tag{4}$$

The goal of the online algorithm is to minimize R_T in expectation, ensuring that its performance approaches that of the optimal static strategy as T increases. Here, the optimal static strategy refers to the best fixed decision, that would have minimized cumulative regret over the entire time horizon.

FTPL is a well-known online learning approach that introduces random perturbations to balance exploration and exploitation. It makes decisions iteratively based on historical requests in order to reduce the cumulative cost. By introducing noise into the decision making process, it facilitates exploration and helps reduce excessive dependence on past information. The decision of FTPL can be defined as:

$$\forall t, \quad \boldsymbol{x}_t = \underset{\boldsymbol{x} \in \mathcal{X}}{\text{argmax}} \langle \boldsymbol{x}, \boldsymbol{n}_{t-1} + \eta \boldsymbol{\gamma} \rangle \tag{5}$$

where η is the learning rate and $\boldsymbol{\gamma}$ is a random noise vector.

Although FTPL has a sublinear regret guarantee, which makes it more robust, its memory footprint is $N \log_2 T + \mathcal{O}(C)$ as historical requests accumulate gradually. If we face a large N, this can lead to a lot of memory usage. Noisy Follow-the-Perturbed-Leader (NFPL) was proposed in [10] to address this issue. NFPL reduces memory usage by sampling requests, making it more efficient while maintaining competi-

tive performance and sublinear regret. The maximum memory required to store all N approximate counts is $N\log_2(pT)$, where p is the request sampling rate.

If only memory usage is a concern, an efficient caching algorithm like TinyLFU [9] can be considered, although it lacks regret guarantees. TinyLFU efficiently tracks file frequencies using a compact, approximate counting structure, typically implemented via a Counting Bloom Filter (CBF) whose memory footprint is $\omega\log_2(T)$ where ω is the size of CBF.

In the next section, we introduce FTPL-JL, a variant of FTPL that uses the JL lemma, to implement an online caching algorithm with regret guarantee using less memory.

III. The FTPL-JL Algorithm

FTPL-JL builds upon the JL lemma [11] to reduce the memory footprint. The JL lemma provides conditions under which points in $\mathbb{R}^N$ can be projected in the lower-dimensional space $\mathbb{R}^k$ ($k < N$) while keeping the distances between any pair of points within a multiplicative error ϵ. While algorithms like FTPL store and update the N-dimensional popularity vector $\boldsymbol{n}_t$ as in (1), FTPL-JL stores and updates a k-dimensional vector $\boldsymbol{y}_t$ which is the compressed representation of the popularity vector, as we illustrate in Fig. 1. As we show later, it is still possible to estimate the popularities from the compressed vector $\boldsymbol{y}_t$.

For completeness, we provide the statement of the JL lemma, following [12, Lem. 1.1].

Lemma 1. *(Johnson-Lindenstrauss Lemma) Given $0 < \epsilon < 1$ and any integer ℓ, let k be a positive integer such that $k \geq \mathcal{O}(\epsilon^{-2}\ln\ell)$. For every set A of ℓ points in $\mathbb{R}^N$ there exists $f : \mathbb{R}^N \to \mathbb{R}^k$ such that for all $\boldsymbol{u}, \boldsymbol{v} \in A$*

$$(1-\epsilon)\left\|\boldsymbol{u}-\boldsymbol{v}\right\|^2 \leq \left\|f(\boldsymbol{u})-f(\boldsymbol{v})\right\|^2 \leq (1+\epsilon)\left\|\boldsymbol{u}-\boldsymbol{v}\right\|^2.$$

Interestingly, linear operators of the form $h(\boldsymbol{x}) = \boldsymbol{\Pi x}$ satisfy the JL-condition with high probability for suitable choices of the random matrix $\boldsymbol{\Pi}$, as shown in [12, Thm. 1.1]. We provide a concise statement of this result.

Theorem 1. *Given $0 < \epsilon < 1$, $\beta > 0$ and any integer ℓ, let k be a positive integer such that:*

$$k \geq \frac{4+2\beta}{\epsilon^2/2-\epsilon^3/3}\ln\ell. \tag{6}$$

Let $\boldsymbol{\Pi}$ be a $k \times N$ random matrix whose entries are chosen as follows:

$$\forall i,\ \forall j,\ \Pi_{ij} = \begin{cases} +1 & \text{with probability } \frac{1}{6}, \\ 0 & \text{with probability } \frac{2}{3}, \\ -1 & \text{with probability } \frac{1}{6}. \end{cases} \tag{7}$$

Let $h : \mathbb{R}^N \to \mathbb{R}^k$ such that $h(\boldsymbol{x}) = \sqrt{\frac{3}{k}}\boldsymbol{\Pi x} \in \mathbb{R}^k$ for $\boldsymbol{x} \in \mathbb{R}^N$. For every set A of ℓ points in $\mathbb{R}^N$ and for all $\boldsymbol{u}, \boldsymbol{v} \in A$, with probability at least $1-\ell^{-\beta}$, the following holds:

$$(1-\epsilon)\left\|\boldsymbol{u}-\boldsymbol{v}\right\|^2 \leq \left\|h(\boldsymbol{u})-h(\boldsymbol{v})\right\|^2 \leq (1+\epsilon)\left\|\boldsymbol{u}-\boldsymbol{v}\right\|^2.$$

Thanks to Theorem 1, we keep track of the popularity of requested files through a compressed k-dimensional vector $\boldsymbol{y} = \boldsymbol{\Pi n}$ which is updated as follows (see Fig. 1):

$$\boldsymbol{y}_t = \boldsymbol{y}_{t-1} + \boldsymbol{\Pi r}_t = \boldsymbol{y}_{t-1} + \boldsymbol{\Pi}_{g_t}\ , \tag{8}$$

where $\boldsymbol{\Pi}_{g_t}$ is the g_t-th column of matrix $\boldsymbol{\Pi}$. In practice, storing the entire matrix $\boldsymbol{\Pi}$ is unnecessary, as its columns $\boldsymbol{\Pi}_i$ can be generated on demand using a (pseudo)random number generator initialized with a seed derived from the requested file index via a hash function. A similar approach can be applied to the generation of noise vector, thereby further reducing memory requirements. The total memory cost is then limited to storing the k-dimensional vector $\boldsymbol{y}_t$, which is a vector of integer numbers. Moreover, we observe that $\mathbb{E}[\boldsymbol{\Pi}_i] = \mathbb{E}[\boldsymbol{y}_t] = \boldsymbol{0}$, then the integer values in $\boldsymbol{y}_t$ are likely to be small and then require a small number of bits. In addition, we can also use $\boldsymbol{y}_t$ to obtain unbiased estimates of the popularity for all requested files, as explained below.

Proposition 1. *Given the compressed vector $\boldsymbol{y}_t$, we can obtain an unbiased estimation of the file g_t popularity as follows:*

$$\hat{n}_{t,g_t} = \frac{3||\boldsymbol{y}_t||_2^2 - 3||\boldsymbol{y}_{t-1}||_2^2 - k}{2k}\ , \tag{9}$$

$$\textit{with} \qquad \mathbb{E}[\hat{n}_{t,g_t}] = n_{t,g_t}\ . \tag{10}$$

Proof. We know from [12, §4.1] that $\mathbb{E}\left[\frac{3}{k}||\boldsymbol{y}_t||_2^2\right] = ||\boldsymbol{n}_t||_2^2$. For file i requested at time t, we take the expectation of (9) and readily write

$$\begin{aligned}\mathbb{E}\left[\hat{n}_{t,i}\right] &= \frac{1}{2}\Big(||\boldsymbol{n}_t||_2^2 - ||\boldsymbol{n}_{t-1}||_2^2 - 1\Big) \\ &= \frac{1}{2}\Big((n_{t-1,i}+1)^2 - n_{t-1,i}^2 - 1\Big) = n_{t,i}\ .\end{aligned}$$

Thereby, our estimation of popularity is unbiased. □

The decision of FTPL-JL is (see (5) for comparison):

$$\forall t, \quad \boldsymbol{x}_t = \operatorname*{argmax}_{\boldsymbol{x}\in\mathcal{X}} \langle \boldsymbol{x}, \hat{\boldsymbol{n}}_{t-1} + \eta\boldsymbol{\gamma}\rangle\ . \tag{11}$$

Algorithm 1 introduces the details on how to reduce memory usage in the caching problem. Firstly, we need to generate a matrix of size $k \times N$ composed of random numbers according to a specific distribution (7) and generate a noise vector of length N (line 1). For our cache, it is initially allocated to the files have the largest noise (line 2). Upon receiving a request at time t (line 4), we first save the reduced-dimension vector of popularity $\boldsymbol{y}$ for later use (line 5) and then update it by adding to it the random vector $\boldsymbol{\Pi}_{g_t}$ (line 6). Then, we estimate the popularity of the requested file g_t by taking the difference between the L2 norms of the current and

Algorithm 1 Follow-the-Perturbed-Leader with Johnson-Lindenstrauss lemma (FTPL-JL)

Require: Cache capacity C, request g_t at time t, number of files N, lower dimension k, learning rate η, time horizon T, cache state $\boldsymbol{S}$, noise parameter λ. Define $\boldsymbol{S}[g_\tau] = \hat{n}_{\tau,g_\tau}$, and $\operatorname{argmin}_g \boldsymbol{S} = g_\mu$, where $\hat{n}_{\mu,g_\mu} \leq \boldsymbol{S}[g_\tau]$, $\forall \hat{n}_{\tau,g_\tau} \in \boldsymbol{S}$.

1: Generate $\boldsymbol{\Pi}$ (7), exponential noise vector $\boldsymbol{\gamma}$ with $\lambda = 1$, and learning rate $\eta = \sqrt{T/(C(1+\ln N))}$.
2: $\boldsymbol{y} = \boldsymbol{0}$, $\boldsymbol{S} = \{(g_i : \eta\gamma_i)\}$, where $\{g_i\}$ is top-C indices with largest $\eta\gamma_i$.
3: **for** $t = 1$ to T **do**
4: User requests a file g_t
5: $\boldsymbol{y}^- := \boldsymbol{y}$
6: $\boldsymbol{y} := \boldsymbol{y} + \boldsymbol{\Pi}_{g_t}$
7: $\hat{n}_{t,g_t} := \frac{1}{2k}\left(3||\boldsymbol{y}||_2^2 - 3||\boldsymbol{y}^-||_2^2 - k\right)$
8: **if** $|\boldsymbol{S}| < C$ **then**
9: $\boldsymbol{S} := \boldsymbol{S} \cup \{(g_t : \hat{n}_{t,g_t} + \eta\gamma_{g_t})\}$
10: **else if** g_t in $\boldsymbol{S}$ **then**
11: $\boldsymbol{S}[g_t] := \hat{n}_{t,g_t} + \eta\gamma_{g_t}$
12: **else if** $\hat{n}_{t,g_t} + \eta\gamma_{g_t} > \min\{\boldsymbol{S}[g_\nu], \forall g_\nu \in \boldsymbol{S}\}$ **then**
13: $\boldsymbol{S} := \boldsymbol{S} \setminus \{(g_\mu = \operatorname{argmin}_g \boldsymbol{S} : \boldsymbol{S}[g_\mu])\}$
14: $\boldsymbol{S} := \boldsymbol{S} \cup \{(g_t : \hat{n}_{t,g_t} + \eta\gamma_{g_t})\}$
15: **end if**
16: **end for**

previous reduced-dimension vectors (line 7). If there is space in cache, then we store in cache the requested file (lines 8–9). If the cache is full and the file is already cached, then we update its value with the latest estimated popularity (lines 10–11). Otherwise, we need to determine whether the file should be cached: if its estimated popularity (plus noise) is greater than the minimum popularity recorded in the cache, we replace the least popular file in cache with the requested file (lines 12–14). Files that are not in cache and whose estimated popularity (plus noise) is smaller than that of cached files should not be cached, and we do not update the cache content when such files are requested (line 15). After completing the above steps, the algorithm goes to the next time step. Observe that the estimate of a file's popularity is only updated when the file is requested. In other words, as long as a file is not requested, its estimated popularity remains constant. Recall from Prop. 1 that this estimate is unbiased.

To summarize, by using the JL lemma, the vectors we store have a size k instead of size N and we get unbiased estimates of the popularity of the requested files. In particular, over T requests, FTPL-JL requires $\Theta(\log_2(T))$ bits for each element of $\boldsymbol{y}$, resulting in a total memory requirement of $\Theta(k\log_2(T))$. Since the expected value of each element in $\boldsymbol{y}$ is zero, its actual memory usage may be much smaller in practice.

In the next section, we demonstrate that FTPL-JL preserves the regret guarantees of FTPL.

IV. Regret Guarantees Evaluation

In this section, we discuss the performance analysis of FTPL-JL and show that our algorithm can be used to make effective caching decisions. But first, we examine the following results that are helpful for our analysis.

Lemma 2. *For noise vector $\boldsymbol{\gamma}$ with each $\gamma_i \sim$ Exponential(λ), and a vector $\boldsymbol{x} \in \mathcal{X}$ with $\mathcal{X} = \{\boldsymbol{x} \in \{0,1\}^N | \sum_{i=1}^N x_i = C\}$, we have:*

$$\mathbb{E}\Big[\max_{\boldsymbol{x}\in\mathcal{X}} \langle \boldsymbol{x}, \boldsymbol{\gamma}\rangle\Big] \leq \tfrac{C}{\lambda}(1+\ln N)\ . \tag{12}$$

Proof. With the exponential noise $\boldsymbol{\gamma}$, we have [13]:

$$\begin{aligned}\mathbb{E}\left[\max\left(\gamma_1,\ldots,\gamma_N\right)\right] &= \frac{1}{N\lambda} + \frac{1}{(N-1)\lambda} + \cdots + \frac{1}{\lambda} \\ &\leq \tfrac{1}{\lambda}(1+\ln N). \end{aligned} \tag{13}$$

We know that $\sum_{i=1}^N x_i = C$, which means we have:

$$\mathbb{E}\left[\max_{\boldsymbol{x}\in\mathcal{X}} \langle \boldsymbol{x}, \boldsymbol{\gamma}\rangle\right] \leq C\mathbb{E}\left[\max\left(\gamma_1,\ldots,\gamma_N\right)\right] \leq \frac{C}{\lambda}(1+\ln N). \tag{14}$$

□

The next lemma specializes [14, Lemma 1.5] which provides expressions for the first and second derivatives of a stochastic smoothing of a convex function with a distribution that has an exponential form and a given scaling parameter. While these properties have been proved in other papers, we state them here for completeness.

Lemma 3. *Consider the potential function $\Phi(\boldsymbol{n}) = \max_{\boldsymbol{x}\in\mathcal{X}} \langle \boldsymbol{x}, \boldsymbol{n}\rangle$ and its exponential smoothing:*

$$\tilde{\Phi}(\boldsymbol{n}) \triangleq \mathbb{E}_{\boldsymbol{\gamma}}\left[\max_{\boldsymbol{x}\in\mathcal{X}} \langle \boldsymbol{x}, \boldsymbol{n} + \eta\boldsymbol{\gamma}\rangle\right], \tag{15}$$

where $\gamma_i \sim$ Exponential(λ). $\Phi(\boldsymbol{n})$ is a closed proper convex function and $\tilde{\Phi}(\boldsymbol{n})$ is convex in $\boldsymbol{n}$. We have:

$$\nabla\tilde{\Phi}(\boldsymbol{n}) = \frac{\lambda}{\eta}\mathbb{E}_{\boldsymbol{\gamma}}\left[\Phi(\boldsymbol{n} + \eta\boldsymbol{\gamma})\mathbf{1}\right], \tag{16}$$

$$\nabla^2\tilde{\Phi}(\boldsymbol{n}) = \frac{\lambda}{\eta}\mathbb{E}_{\boldsymbol{\gamma}}\left[\nabla\Phi(\boldsymbol{n} + \eta\boldsymbol{\gamma})\mathbf{1}^T\right]. \tag{17}$$

We now state our main result on the analysis of the regret bound of FTPL-JL.

Theorem 2. *Taking into account the estimated popularity vector $\hat{\boldsymbol{n}}$, whose elements are obtained in* (9)*, the expected regret of FTPL-JL over random noise $\boldsymbol{\gamma}$ and estimated popularity $\hat{\boldsymbol{n}}$ satisfies the following upper-bound:*

$$\mathbb{E}_{\hat{\boldsymbol{n}},\boldsymbol{\gamma}}\left[R_T\right] \leq \sqrt{2C(1+\ln N)T}\ . \tag{18}$$

We obtain this result by following the steps in [15] and using Lemmas 2 and 3 (see Appendix A).

Consistently with other works [10], [16], our regret analysis considers in (18) the expectation over all random values. The bound expressed in Theorem 2 is insensitive to the value of k as if one could reduce the popularity vector to an arbitrarily low dimension. This, of course, must be avoided, as too small values of k would force the projection accuracy parameter ϵ to be prohibitively large (see 6). Although our popularity estimation depends on k (see (9)), by taking in (18) the expectation of the regret in particular over the popularity estimation, the reliance on k vanishes because the unbiased property of our estimates always holds regardless of k; see Prop. 1.

For these reasons, we consider in the following analysis the regret conditioned on a fixed realization of the projection matrix $\mathbf{\Pi}$. That is, we only take the expectation over the noise in the system, allowing for a more precise characterization of the regret behavior under a given dimensionality reduction matrix.

Theorem 3. *With a given matrix $\mathbf{\Pi}$, which yields an estimated popularity $\hat{\boldsymbol{n}}_t$ at each time t, the regret bound of FTPL-JL is as follows:*

$$\mathbb{E}_{\boldsymbol{\gamma}|\hat{\boldsymbol{n}}}[R_t] < \frac{\eta C}{\lambda}(1+\ln N) + \frac{\lambda}{2\eta}\sum_{t=1}^{T}(1+\|\boldsymbol{n}_{t-1}-\hat{\boldsymbol{n}}_{t-1}\|_1)^2 \tag{19}$$

We obtain the above result by the steps similar to [17, Thm. 3] (see Appendix B).

We observe from Theorem 3 that if our popularity estimate is close enough to the true value (meaning if the estimate is consistent) and if we set $\eta = \lambda\sqrt{T/(2C(1+\ln N))}$, then the bound (19) is essentially the same as in Theorem 2. We know from (9) that the estimation accuracy depends on the following difference:

$$\begin{aligned}\hat{n}_{t,i} - n_{t,i} \;&\propto\; \tfrac{3}{k}\|\boldsymbol{y}_t\|_2^2 - \tfrac{3}{k}\|\boldsymbol{y}_{t-1}\|_2^2 - \|\boldsymbol{n}_t\|_2^2 + \|\boldsymbol{n}_{t-1}\|_2^2 \\ &= \left(\tfrac{3}{k}\|\mathbf{\Pi}\boldsymbol{r}_t\|_2^2 + \tfrac{6}{k}\langle\mathbf{\Pi}\boldsymbol{n}_{t-1}, \mathbf{\Pi}\boldsymbol{r}_t\rangle\right) \\ &\quad - \left(\|\boldsymbol{r}_t\|_2^2 + 2\langle\boldsymbol{n}_{t-1}, \boldsymbol{r}_t\rangle\right) \;. \end{aligned} \tag{20}$$

According to the JL lemma, the first line of (20) is within a multiplicative error of the equivalent expression prior to the projection. Thereby, the difference between $\hat{n}_{t,i}$ and $n_{t,i}$ depends on ϵ and so does the bound in (19). This indicates that, in practice, we need to trade memory saving for regret bound, as a lower k leads to a larger ϵ and consequently a larger regret bound.

V. Experiments

In this section, we compare FTPL-JL with other caching policies, using both synthetic and real-world traces. Specifically, we divide the experiments and comparisons into three parts. In the first part, we compare the average reward of FTPL-JL and other memory-efficient caching policies like NFPL and TinyLFU

TABLE I: Trace description

Trace ($T = 10,000$)		Number of files requested
Zipf ($\alpha = 1.0$)	[19]	2813
Round-robin $5 \times [1, 2000]$	[10]	2000
Akamai	[20]	995
Glimpse	[21]	2668
Movie	[22]	488
Malware	[23]	5744

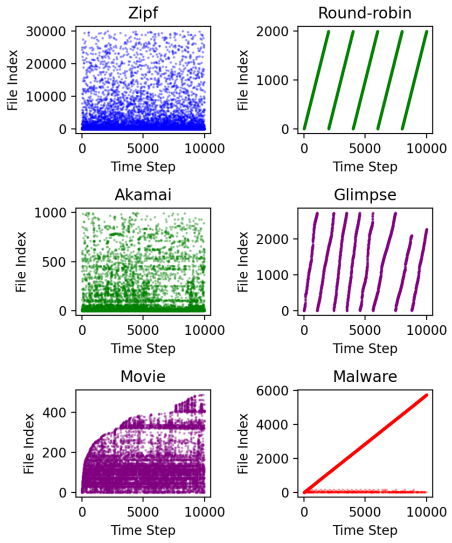

Fig. 2: Requests of the dataset

across different traces. Then we analyze the impact of memory on performance by plotting memory-cost curves. Finally, in the third part, we compare FTPL-JL with an even larger set of caching policies including LRU and SIEVE [18].

Traces: We employ multiple traces to evaluate system performance. For the Zipf trace, we set the parameter $\alpha = 1.0$ and possible catalog size $N = 10^4$ [19]. For the Round-robin trace, we generate a trace that iterates cyclically in the range $[1, 2000]$ [10]. We include diverse real-world traces, including Akamai trace (requests for objects in Akamai CDN [20]), Glimpse trace (requests for pages in the Bible containing a given word [21]), Movie trace (timestamped movie ratings [22]), and Malware trace (IoT nodes contacted over time by malware, [23]). These traces enable a comprehensive evaluation under realistic workload conditions.

From Figure 2, we observe that requests in the Zipf trace are concentrated on a subset of files, while other files are only occasionally requested. In the Round-robin trace, files are requested in a cyclic manner. Among the real-world traces, the Akamai trace exhibits characteristics similar to Zipf but with a smaller number of unique files. The Glimpse trace follows a pattern similar to Round-robin. The Movie trace reveals the introduction of new files (movies) over time, with varying levels of popularity. In the Malware trace, some IP addresses appear recurrently and often in close-by time instants,

TABLE II: Parameter setting for different algorithms

Memory m	FTPL-JL k	NFPL p	TinyLFU ω
$(T/20)\log_2(T)$	$T/20$	$T^{-0.95}$	$T/20$
$(T/5)\log_2(T)$	$T/5$	$T^{-0.8}$	$T/5$

while most are observed only once. For all traces, we assume that requests originate from a catalog of size $N = 10,000$ files. Table I provides the number of unique files requested in each trace.

Cache setting: Unless otherwise specified, we consider a cache that can store 100 files and a time horizon of $T = N = 10,000$. All caches are initially empty. The learning rates of FTPL-JL and NFPL are set according to theoretical guidelines to minimize the regret bound.

A. *FTPL-JL vs. other memory-saving algorithms*

In this section, we simulate FTPL-JL and compare it with other memory-saving algorithms. The performance of each algorithm is evaluated by comparing their average rewards, calculated as follows:

$$\text{avg-reward}_t = \sum_{i=1}^{t} \langle \boldsymbol{r}_i, \boldsymbol{x}_i \rangle / t. \tag{21}$$

We observe that the average reward corresponds to the cumulative average hit ratio up to time t.

We compare FTPL-JL with two other memory-efficient policies: NFPL and TinyLFU. The memory requirements of these policies are $k\log_2(T)$, $N\log_2(pT)$, and $\omega\log_2(T)$, respectively (see Section III for FTPL-JL and Section II for NFPL and TinyLFU). All three policies are configured under the same memory budget m. Table II specifies how their respective parameters k, p, and ω should be set for two different values of the memory m, corresponding to 5% and 20% of the memory that the standard FTPL would use.

The results are shown in Figure 3. We observe that on more stationary traces, such as Zipf and Akamai, TinyLFU achieves the best performance due to its ability to effectively leverage the stable distribution of requests. FTPL-JL performs worse but still maintains comparable performance and achieves a significant reward. As expected, larger memory leads to better performance. In contrast, NFPL, despite enjoying the same regret guarantees as FTPL-JL, performs poorly. This is because the sampling ratio p, required to match the same memory budget, is extremely small, causing NFPL to sample only a few requests over the entire trace. In the Movie trace, although new files appear over time, there remains a clear subset of highly popular files that are worth caching, leading to similar relative performance of the three caching policies.

The Malware trace exhibits the same ranking of the three policies but shows a different effect of memory. Notably, for TinyLFU and FTPL-JL, the reward increases as memory decreases. For TinyLFU, this phenomenon can be explained as follows: with very small

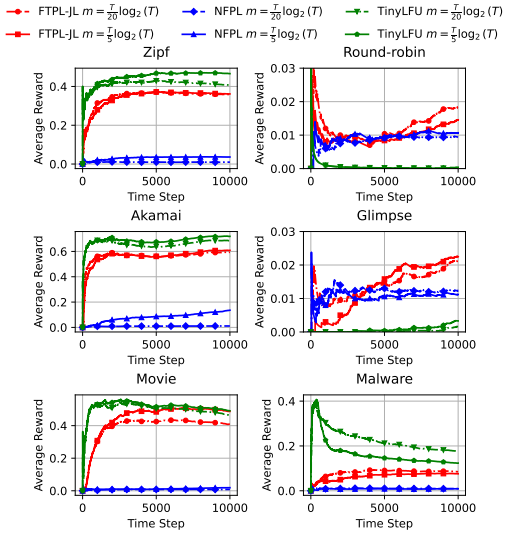

Fig. 3: Average reward

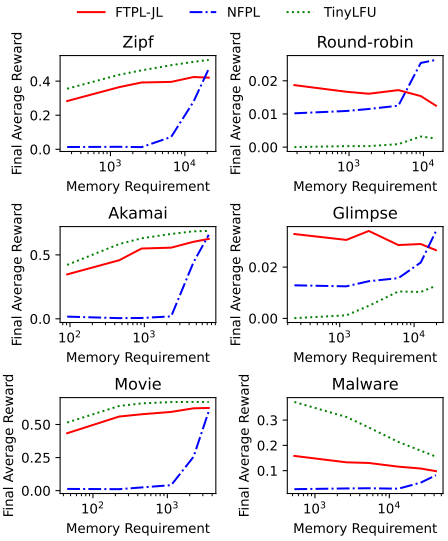

Fig. 4: Reward vs. memory Curves. The final average reward is the average reward over the whole horizon T.

memory, TinyLFU behaves similarly to LRU,[1] allowing it to exploit the temporal locality of this trace. The behavior of FTPL-JL with small memory is more complex, but we still observe that limited memory results in noisier popularity estimates, leading to a behavior that deviates further from LFU.

Results are markedly different for the Round-robin trace. In this case, NFPL achieves the best performance: due to its very low sampling rate, it stores a random set of 100 files, attaining a reward of 0.01. Over time, FTPL-JL reaches the same performance. Over a longer time horizon, the rewards of both FTPL-JL and NFPL

[1]Consider the extreme case when all files are hashed to the same counters. Whatever file is requested at time t, its popularity is estimated to be t. The file is then inserted and the least recently requested file (whose popularity is believed to be $t - C$) is evicted.

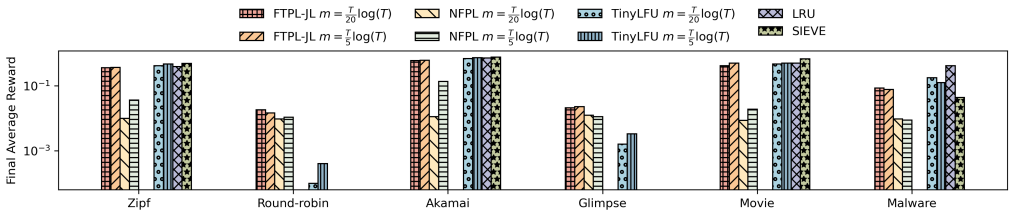

Fig. 5: Final average reward: comparison of various caching policies with two memory settings

would increase as they correctly identify the requested files. For a slightly longer time horizon, FTPL-JL would outperform NFPL. TinyLFU achieves a lower reward because, in the Round-robin trace, the most frequently requested files are less likely to be requested again in the near future. While LFU would obtain a null reward—always storing the 100 most requested files, none of which will be requested in the next timeslot—TinyLFU performs slightly better due to estimation errors introduced by its approximate counting structures.

The trace Glimpse leads to similar conclusion. As the request pattern is not exactly periodic and some files are requested more times than the others, TinyLFU and FTPL-JL perform both better than in the Round-robin trace and in particular FTPL-JL outperforms NFPL already after the first 6000 requests.

B. Memory and performance curve

The experimental results in Figure 4 illustrate the relationship between memory requirements and the performance of three algorithms across different traces.

For Zipf, Akamai and Movie traces, FTPL-JL consistently outperforms NFPL, and exhibits moderately worse performance than TinyLFU. Rewards tend to increase with memory for all three policies.

The other three traces illustrate different effects of memory on the performance of the policies. In general, increasing memory improves the performance of NFPL. TinyLFU exhibits the most distinct behaviors. As previously observed, TinyLFU tends to behave like LRU for very small memory and like LFU for large memory. Consequently, its reward decreases with increasing memory in Malware, which exhibits higher temporal locality. In Round-robin, both LRU and LFU are suboptimal. However, for intermediate memory values, TinyLFU manages to correctly identify the subset of requested files while introducing sufficient noise to avoid retaining only the most recently requested files. For Glimpse, which has a similar request pattern but different file popularities, more memory is always beneficial.

Across these three traces, FTPL-JL's performance is the least dependent on memory. Overall, there appears to be a slight trend toward increasing reward as memory decreases, which can be attributed to the benefits of greater exploration due to noisier estimates.

TABLE III: Performance in Glimpse with multi cache sizes

Cache size	FTPL-JL	NFPL	TinyLFU	LRU	SIEVE
1000	0.24	0.10	0.18	0.012	0.040
500	0.13	0.051	0.059	0.0	0.0
100	0.022	0.0097	0.0049	0.0	0.0
50	0.0098	0.0059	0.0018	0.0	0.0

For Round-robin and Glimpse FTPL-JL outperforms in almost all cases both TinyLFU and NFPL. Overall, FTPL-JL demonstrates consistently good performance across a variety of request patterns, thanks to its regret guarantees. While NFPL also enjoys sublinear regret guarantees, its regret scales inversely with the request sampling ratio [10, Corollary 2], resulting in very poor performance when memory is highly constrained.

C. Other policies

Figure 5 compares the performance of the policies considered so far, along with two additional ones: the widely used LRU and the recently proposed SIEVE [18]. SIEVE is based on FIFO (First-In-First-Out) and employs a queue and a pointer to track cache data with access bits. It is characterized by its straightforward mechanism of eviction based on whether an object has been accessed, providing high efficiency and scalability. Both LRU and SIEVE have memory requirements on the order of C. In Figure 5, the policies are divided into two groups: the four bars on the left correspond to caching policies with regret guarantees (FTPL-JL and NFPL), while the four bars on the right correspond to the other caching policies. While TinyLFU with the largest memory performs best on the Zipf, Akamai, and Movie traces, and LRU is the best policy on Malware, FTPL-JL maintains consistently good performance across all six traces and achieves the best results on Round-robin and Glimpse, where LRU and SIEVE perform particularly poorly. Compared to NFPL, which also has regret guarantees, FTPL-JL outperforms it under the same memory constraints in most situations.

We also analyze the performance of algorithms using the Glimpse trace with varying cache sizes. For FTPL-JL, NFPL, and TinyLFU, we fix the memory size $m = (T/5)\log_2(T)$ to simplify comparisons. The results in Table III show that TinyLFU works better with large caches, and FTPL-JL keeps the best average reward with different cache sizes. At the same time,

when the cache size is reduced from 1000 to 500, FTPL-JL maintains its performance more robustly compared to other algorithms, which experience a sharp decline. This highlights the potential of FTPL-JL in resource-constrained environments, such as limited memory or cache.

VI. Conclusions

In this paper, we proposed a novel dimensionality reduction algorithm for online caching, leveraging the Follow-the-Perturbed-Leader (FTPL) framework in conjunction with the Johnson-Lindenstrauss (JL) lemma. Our approach effectively mitigates memory consumption while preserving sublinear regret, thereby addressing a fundamental challenge in caching systems with large catalogues. By conducting experiments on both synthetic and real-world traces, we have demonstrated that our proposed algorithm significantly reduces memory usage without compromising performance, highlighting its practical applicability in large-scale systems with stringent resource constraints.

References

[1] Abubakr O. Al-Abbasi, Vaneet Aggarwal, and Moo-Ryong Ra. Multi-tier caching analysis in CDN-based over-the-top video streaming systems. *IEEE/ACM Transactions on Networking*, 27(2):835–847, 2019.

[2] Sara McAllister, Benjamin Berg, Julian Tutuncu-Macias, Juncheng Yang, Sathya Gunasekar, Jimmy Lu, Daniel S. Berger, Nathan Beckmann, and Gregory R. Ganger. Kangaroo: Theory and practice of caching billions of tiny objects on flash. *ACM Trans. Storage*, 18(3), August 2022.

[3] Kianoosh Mokhtarian and Hans-Arno Jacobsen. Caching in video CDNs: Building strong lines of defense. In *Proc. of 9th European Conference on Computer Systems*, pages 1–13, 2014.

[4] Zhe Li, Gwendal Simon, and Annie Gravey. Caching policies for in-network caching. In *ICCCN 2012*, pages 1–7, 2012.

[5] Shiji Zhou, Zhi Wang, Chenghao Hu, Yinan Mao, Haopeng Yan, Shanghang Zhang, Chuan Wu, and Wenwu Zhu. Caching in dynamic environments: A near-optimal online learning approach. *IEEE Transactions on Multimedia*, 25:792–804, 2021.

[6] Georgios S Paschos, Apostolos Destounis, Luigi Vigneri, and George Iosifidis. Learning to cache with no regrets. In *IEEE INFOCOM 2019*, pages 235–243, 2019.

[7] Shai Shalev-Shwartz. Online learning and online convex optimization. *Foundations and Trends® in Machine Learning*, 4(2):107–194, 2012.

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

[9] Gil Einziger, Roy Friedman, and Ben Manes. TinyLFU: A highly efficient cache admission policy. *ACM Trans. Storage*, 13(4):1–31, 2017.

[10] Younes Ben Mazziane, Francescomaria Faticanti, Giovanni Neglia, and Sara Alouf. No-regret caching with noisy request estimates. In *IEEE VCC 2023*, pages 341–346, 2023.

[11] William B. Johnson and Joram Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. *Contemporary Mathematics*, 26(189-206):1, 1984.

[12] Dimitris Achlioptas. Database-friendly random projections: Johnson-Lindenstrauss with binary coins. *Journal of computer and System Sciences*, 66(4):671–687, 2003.

[13] Pierpaolo Vivo. Large deviations of the maximum of independent and identically distributed random variables. *European Journal of Physics*, 36(5):055037, 2015.

[14] Jacob Abernethy, Chansoo Lee, and Ambuj Tewari. Perturbation techniques in online learning and optimization. *Perturbations, Optimization, and Statistics*, 233:17, 2016.

[15] Alon Cohen and Tamir Hazan. Following the perturbed leader for online structured learning. In *International Conference on Machine Learning*, pages 1034–1042. PMLR, 2015.

[16] Rupert Freeman, David Pennock, Chara Podimata, and Jennifer Wortman Vaughan. No-regret and incentive-compatible online learning. In *ICML 2020*, pages 3270–3279, 2020.

[17] Naram Mhaisen, Abhishek Sinha, Georgios Paschos, and George Iosifidis. Optimistic no-regret algorithms for discrete caching. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 6(3):1–28, 2022.

[18] Yazhuo Zhang, Juncheng Yang, Yao Yue, Ymir Vigfusson, and KV Rashmi. SIEVE is Simpler than LRU: an Efficient Turn-Key Eviction Algorithm for Web Caches. In *USENIX NSDI 24*, pages 1229–1246, 2024.

[19] Francescomaria Faticanti and Giovanni Neglia. Optimistic online caching for batched requests. *Computer Network*, 244:110341, 2024.

[20] Giovanni Neglia, Damiano Carra, Mingdong Feng, Vaishnav Janardhan, Pietro Michiardi, and Dimitra Tsigkari. Access-time-aware cache algorithms. *ACM Trans. on Modeling and Performance Evaluation of Computing Systems*, 2(4):1–29, 2017.

[21] Jongmoo Choi, Sam H Noh, Sang Lyul Min, and Yookun Cho. Towards application/file-level characterization of block references: A case for fine-grained buffer management. In *ACM SIGMETRICS 2000*, pages 286–295, 2000.

[22] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Trans. on Interactive Intelligent Systems*, 5(4):1–19, 2015.

[23] Sebastian Garcia, Agustin Parmisano, and Maria Jose Erquiaga. IoT-23: A labeled dataset with malicious and benign IoT network traffic, January 2020.

[24] Jacob Abernethy, Chansoo Lee, Abhinav Sinha, and Ambuj Tewari. Online linear optimization via smoothing. In *Conference on Learning Theory*, pages 807–823. PMLR, 2014.

Appendix

A. Proof of Theorem 2

Consider $\Phi(\boldsymbol{n}) \triangleq \max_{\boldsymbol{x}\in\mathcal{X}}\langle \boldsymbol{x}, \boldsymbol{n}\rangle$, which is called a potential function, and its associated exponential smoothing $\tilde{\Phi}(\boldsymbol{n})$ as defined in (15). Taking the expectation of the regret as defined in (4), we write

$$\mathbb{E}_{\hat{\boldsymbol{n}},\boldsymbol{\gamma}}\left[R_t\right] = \Phi(\boldsymbol{n}_T) - \sum_{t=1}^{T}\left\langle \mathbb{E}_{\hat{\boldsymbol{n}},\boldsymbol{\gamma}}\left[\boldsymbol{x}_t\right], \boldsymbol{r}_t\right\rangle . \quad (22)$$

To upper bound the expected regret, we derive an upper bound on $\Phi(\boldsymbol{n}_T)$ and a lower bound first on $\mathbb{E}_{\hat{\boldsymbol{n}},\boldsymbol{\gamma}}\left[\boldsymbol{x}_t\right]$ and then on the summation in (22).

Upper bound on $\Phi(\boldsymbol{n}_T)$. We first write an inequality with the potential function and its exponential smoothing. By Jensen's inequality, we have

$$\begin{aligned}\tilde{\Phi}(\boldsymbol{n}_T) &\overset{(15)}{=} \mathbb{E}_{\boldsymbol{\gamma}}\left[\max_{\boldsymbol{x}\in\mathcal{X}}\langle \boldsymbol{x}, \boldsymbol{n}_T + \eta\boldsymbol{\gamma}\rangle\right] \\ &\geq \max_{\boldsymbol{x}\in\mathcal{X}}\left\langle \boldsymbol{x}, \boldsymbol{n}_T + \eta\,\frac{\mathbf{1}}{\lambda}\right\rangle = \Phi\left(\boldsymbol{n}_T\right) + \frac{C\eta}{\lambda}.\end{aligned}$$

As a result, $\Phi\left(\boldsymbol{n}_T\right) < \Phi\left(\boldsymbol{n}_T\right) + \dfrac{C\eta}{\lambda} \leq \tilde{\Phi}(\boldsymbol{n}_T)$. (23)

Lower bound on $\mathbb{E}_{\hat{\boldsymbol{n}},\boldsymbol{\gamma}}\left[\boldsymbol{x}_t\right]$. We first derive $\mathbb{E}_{\boldsymbol{\gamma}|\hat{\boldsymbol{n}}}\left[\boldsymbol{x}_t\right]$:

$$\mathbb{E}_{\boldsymbol{\gamma}|\hat{\boldsymbol{n}}}\left[\boldsymbol{x}_t\right] \overset{(11)}{=} \mathbb{E}_{\boldsymbol{\gamma}|\hat{\boldsymbol{n}}}\left[\underset{\boldsymbol{x}\in\mathcal{X}}{\operatorname{argmax}}\left\langle \boldsymbol{x}, \hat{\boldsymbol{n}}_{t-1}+\eta\boldsymbol{\gamma}\right\rangle\right] \overset{(a)}{=} \nabla\tilde{\Phi}\left(\hat{\boldsymbol{n}}_{t-1}\right) \overset{(16)}{=} \frac{\lambda}{\eta}\mathbb{E}_{\boldsymbol{\gamma}|\hat{\boldsymbol{n}}}\left[\Phi(\hat{\boldsymbol{n}}_{t-1}+\eta\boldsymbol{\gamma})\mathbf{1}\right] \quad (24)$$

where (a) comes from [24, Eqs. (4)-(5)]. Next, we find a lower bound on $\mathbb{E}_{\hat{\boldsymbol{n}},\boldsymbol{\gamma}}\left[\boldsymbol{x}_t\right]$ as follows:

$$\begin{aligned}\mathbb{E}_{\hat{\boldsymbol{n}},\boldsymbol{\gamma}}\left[\boldsymbol{x}_t\right] &\overset{(24)}{=} \tfrac{\lambda}{\eta}\,\mathbb{E}_{\hat{\boldsymbol{n}},\boldsymbol{\gamma}}\left[\Phi(\hat{\boldsymbol{n}}_{t-1}+\eta\boldsymbol{\gamma})\mathbf{1}\right]\\ &= \tfrac{\lambda}{\eta}\,\mathbb{E}_{\boldsymbol{\gamma}}\left[\mathbb{E}_{\hat{\boldsymbol{n}}|\boldsymbol{\gamma}}\left[\Phi(\hat{\boldsymbol{n}}_{t-1}+\eta\boldsymbol{\gamma})\mathbf{1}\right]\right]\\ &\overset{(a)}{\geq} \tfrac{\lambda}{\eta}\,\mathbb{E}_{\boldsymbol{\gamma}}\left[\Phi\left(\mathbb{E}_{\hat{\boldsymbol{n}}|\boldsymbol{\gamma}}\left[\hat{\boldsymbol{n}}_{t-1}+\eta\boldsymbol{\gamma}\right]\right)\mathbf{1}\right]\\ &\overset{(10)}{=} (\lambda/\eta)\,\mathbb{E}_{\boldsymbol{\gamma}}\left[\Phi(\boldsymbol{n}_{t-1}+\eta\boldsymbol{\gamma})\mathbf{1}\right] \overset{(16)}{=} \nabla\tilde{\Phi}\left(\boldsymbol{n}_{t-1}\right),\end{aligned}$$

where (a) is Jensen's inequality (Φ is a convex function). As a result, the summation in (22) verifies:

$$\sum_{t=1}^{T}\left\langle \mathbb{E}_{\hat{\boldsymbol{n}},\boldsymbol{\gamma}}\left[\boldsymbol{x}_t\right], \boldsymbol{r}_t\right\rangle \geq \sum_{t=1}^{T}\left\langle \nabla\tilde{\Phi}\left(\boldsymbol{n}_{t-1}\right), \boldsymbol{r}_t\right\rangle . \quad (25)$$

Intermediate result on $\mathbb{E}_{\hat{\boldsymbol{n}},\boldsymbol{\gamma}}\left[R_t\right]$. By combining the inequalities (23) and (25), we use (22) to write:

$$\mathbb{E}_{\hat{\boldsymbol{n}},\boldsymbol{\gamma}}\left[R_t\right] < \tilde{\Phi}(\boldsymbol{n}_T) - \sum_{t=1}^{T}\left\langle \nabla\tilde{\Phi}\left(\boldsymbol{n}_{t-1}\right), \boldsymbol{r}_t\right\rangle . \quad (26)$$

To rewrite the right-hand side (r.h.s.) of this inequality, we use a Taylor's expansion of $\tilde{\Phi}$ with a second order remainder:

$$\begin{aligned}\tilde{\Phi}\left(\boldsymbol{n}_t\right) = \tilde{\Phi}\left(\boldsymbol{n}_{t-1}\right) &+ \left\langle \nabla\tilde{\Phi}\left(\boldsymbol{n}_{t-1}\right), \boldsymbol{r}_t\right\rangle\\ &+ \frac{1}{2}\left\langle \boldsymbol{r}_t, \nabla^2\tilde{\Phi}\left(\tilde{\boldsymbol{r}}_t\right)\boldsymbol{r}_t\right\rangle,\end{aligned} \quad (27)$$

where $\tilde{\boldsymbol{r}}_t$ is a point on the line segment connecting $\boldsymbol{n}_{t-1}$ and $\boldsymbol{n}_t$. We now sum (27) over the entire time horizon T which allows us to rewrite (26) as follows:

$$\mathbb{E}_{\hat{\boldsymbol{n}},\boldsymbol{\gamma}}\left[R_t\right] < \tilde{\Phi}\left(\boldsymbol{n}_0\right) + \frac{1}{2}\sum_{t=1}^{T}\left\langle \boldsymbol{r}_t, \nabla^2\tilde{\Phi}\left(\tilde{\boldsymbol{r}}_t\right)\boldsymbol{r}_t\right\rangle . \quad (28)$$

Last steps: bounds on r.h.s. of (28). Using the definition of exponential smoothing and $\boldsymbol{n}_0 = \mathbf{0}$, we write:

$$\tilde{\Phi}\left(\boldsymbol{n}_0\right) \overset{(15)}{=} \mathbb{E}_{\boldsymbol{\gamma}}\left[\max_{\boldsymbol{x}\in\mathcal{X}}\left\langle \boldsymbol{x}, \eta\boldsymbol{\gamma}\right\rangle\right] \overset{(12)}{\leq} \frac{\eta C}{\lambda}(1+\ln N) . \quad (29)$$

To bound the scalar product in (28), we recall that only one file g_t is requested at time t, allowing us to write:

$$\begin{aligned}\left\langle \boldsymbol{r}_t, \nabla^2\tilde{\Phi}\left(\tilde{\boldsymbol{r}}_t\right)\boldsymbol{r}_t\right\rangle &= \left(\nabla^2\tilde{\Phi}(\tilde{\boldsymbol{r}}_t)\right)_{g_t g_t}\\ &\overset{(17)}{=} \frac{\lambda}{\eta}\,\mathbb{E}_{\boldsymbol{\gamma}}\left[\left(\nabla\Phi(\tilde{\boldsymbol{r}}_t+\eta\boldsymbol{\gamma})\mathbf{1}^T\right)_{g_t g_t}\right]\\ &= \frac{\lambda}{\eta}\,\mathbb{E}_{\boldsymbol{\gamma}}\left[\left(\nabla\Phi(\tilde{\boldsymbol{r}}_t+\eta\boldsymbol{\gamma})\right)_{g_t}\right]\\ &\overset{(a)}{=} \frac{\lambda}{\eta}\,\mathbb{E}_{\boldsymbol{\gamma}}\left[\left(\underset{\boldsymbol{x}\in\mathcal{X}}{\operatorname{argmax}}\langle \boldsymbol{x}, \tilde{\boldsymbol{r}}_t+\eta\boldsymbol{\gamma}\rangle\right)_{g_t}\right] \overset{(b)}{\leq} \frac{\lambda}{\eta},\end{aligned} \quad (30)$$

where (a) comes from [24, Eqs. (4)-(5)], and (b) is from the fact that $x_{g_t} \in \{0,1\}$. Last, we use the inequalities (29) and (30) to rewrite the bound (28) as follows:

$$\mathbb{E}_{\hat{\boldsymbol{n}},\boldsymbol{\gamma}}\left[R_T\right] < \frac{\eta C}{\lambda}(1+\ln N) + \frac{\lambda T}{2\eta}.$$

Choosing now $\eta = \lambda\sqrt{T/(2C(1+\ln N))}$, we get 18 completing the proof.

B. Proof of Theorem 3

We first write the conditional expectation of R_T:

$$\mathbb{E}_{\boldsymbol{\gamma}|\hat{\boldsymbol{n}}}\left[R_t\right] \overset{(4)}{=} \Phi(\boldsymbol{n}_T) - \sum_{t=1}^{T}\left\langle \mathbb{E}_{\boldsymbol{\gamma}|\hat{\boldsymbol{n}}}\left[\boldsymbol{x}_t\right], \boldsymbol{r}_t\right\rangle . \quad (31)$$

Next, we rewrite $\mathbb{E}_{\boldsymbol{\gamma}|\hat{\boldsymbol{n}}}\left[\boldsymbol{x}_t\right]$ as follows:

$$\mathbb{E}_{\boldsymbol{\gamma}|\hat{\boldsymbol{n}}}\left[\boldsymbol{x}_t\right] \overset{(24)}{=} \frac{\lambda}{\eta}\,\mathbb{E}_{\boldsymbol{\gamma}|\hat{\boldsymbol{n}}}\left[\Phi(\hat{\boldsymbol{n}}_{t-1}+\eta\boldsymbol{\gamma})\mathbf{1}\right] \overset{(16)}{=} \nabla\tilde{\Phi}\left(\hat{\boldsymbol{n}}_{t-1}\right) . \quad (32)$$

Combining (23), (31) and (32) yields:

$$\mathbb{E}_{\boldsymbol{\gamma}|\hat{\boldsymbol{n}}}\left[R_t\right] < \tilde{\Phi}(\boldsymbol{n}_T) - \sum_{t=1}^{T}\left\langle \nabla\tilde{\Phi}\left(\hat{\boldsymbol{n}}_{t-1}\right), \boldsymbol{r}_t\right\rangle . \quad (33)$$

We again use a Taylor's expansion of $\tilde{\Phi}$ with a second order remainder, but this time around point $\hat{\boldsymbol{n}}_{t-1}$:

$$\begin{aligned}\tilde{\Phi}\left(\boldsymbol{n}_t\right) = \tilde{\Phi}\left(\hat{\boldsymbol{n}}_{t-1}\right) &+ \left\langle \nabla\tilde{\Phi}\left(\hat{\boldsymbol{n}}_{t-1}\right), \boldsymbol{n}_t - \hat{\boldsymbol{n}}_{t-1}\right\rangle\\ &+ \frac{1}{2}\left\langle \boldsymbol{n}_t - \hat{\boldsymbol{n}}_{t-1}, \nabla^2\tilde{\Phi}\left(\tilde{\boldsymbol{r}}_t^*\right)\left(\boldsymbol{n}_t - \hat{\boldsymbol{n}}_{t-1}\right)\right\rangle,\end{aligned} \quad (34)$$

where $\tilde{\boldsymbol{r}}_t^*$ is a point on the line segment connecting $\hat{\boldsymbol{n}}_{t-1}$ and $\boldsymbol{n}_t$. From the convexity of $\tilde{\Phi}$:

$$\tilde{\Phi}\left(\hat{\boldsymbol{n}}_{t-1}\right) \leq \tilde{\Phi}\left(\boldsymbol{n}_{t-1}\right) + \left\langle \nabla\tilde{\Phi}\left(\hat{\boldsymbol{n}}_{t-1}\right), \hat{\boldsymbol{n}}_{t-1} - \boldsymbol{n}_{t-1}\right\rangle . \quad (35)$$

By combining (34) and (35), we can write:

$$\begin{aligned}\tilde{\Phi}\left(\boldsymbol{n}_t\right) \leq \tilde{\Phi}\left(\boldsymbol{n}_{t-1}\right) &+ \left\langle \nabla\tilde{\Phi}\left(\hat{\boldsymbol{n}}_{t-1}\right), \boldsymbol{r}_t\right\rangle\\ &+ \frac{1}{2}\left\langle \boldsymbol{n}_t - \hat{\boldsymbol{n}}_{t-1}, \nabla^2\tilde{\Phi}\left(\tilde{\boldsymbol{r}}_t^*\right)\left(\boldsymbol{n}_t - \hat{\boldsymbol{n}}_{t-1}\right)\right\rangle.\end{aligned} \quad (36)$$

We now focus on the scalar product in the last line of (36). We can write:

$$\begin{aligned}\left\langle \boldsymbol{n}_t - \hat{\boldsymbol{n}}_{t-1}, \nabla^2\tilde{\Phi}(\tilde{\boldsymbol{r}}_t^*)(\boldsymbol{n}_t - \hat{\boldsymbol{n}}_{t-1})\right\rangle &\overset{(a)}{\leq} \frac{\lambda}{\eta}\left(\left\|\boldsymbol{n}_t - \hat{\boldsymbol{n}}_{t-1}\right\|_1^2\right)\\ &= \frac{\lambda}{\eta}\left(\left\|\boldsymbol{r}_t + \boldsymbol{n}_{t-1} - \hat{\boldsymbol{n}}_{t-1}\right\|_1^2\right)\\ &\overset{(b)}{\leq} \frac{\lambda}{\eta}\left(\left\|\boldsymbol{r}_t\right\|_1^2 + \left\|\boldsymbol{n}_{t-1} - \hat{\boldsymbol{n}}_{t-1}\right\|_1^2\right)\end{aligned} \quad (37)$$

where (a) is from [17, Thm. 3] and (b) is from the triangle inequality. Observe that $\|\boldsymbol{r}_t\|_1^2 = 1$.

By using the bounds (29) and (37), and summing (36) over time horizon T, we can rewrite the regret bound (33) to find 19 completing the proof.