Efficient and Optimal No-Regret Caching under Partial Observation

Younes Ben Mazziane[1], Francescomaria Faticanti[2], Sara Alouf[3], and Giovanni Neglia[3]

[1]Avignon University, Avignon, France. Email: younes.ben-mazziane@univ-avignon.fr
[2]ENS Lyon, Lyon, France. Email: francescomaria.faticanti@ens-lyon.fr
[3]Inria, Université Côte d'Azur, Sophia Antipolis, France. Emails: {sara.alouf, giovanni.neglia}@inria.fr

***Abstract*—Online learning algorithms have been successfully used to design caching policies with sublinear regret in the total number of requests, with no statistical assumption about the request sequence. Most existing algorithms involve computationally expensive operations and require knowledge of all past requests. However, this may not be feasible in practical scenarios such as *femtocaching*, where a base station (BS) jointly decides the content of many edge caches and visibility of all requests at the BS requires constant communication between these caches and the BS. To capture this constraint, we study a single cache problem under a more restrictive setting, that we refer to as the Bernoulli Partial Observability (BPO) model, in which the caching policy only observes a request with probability p, reflecting the fraction of requests forwarded from the edge caches to the BS in the *femtocaching* example. We propose a policy, based on the classic online learning algorithm Follow-the-Perturbed-Leader (FPL), that achieves an asymptotically optimal regret bound of $\mathcal{O}(\sqrt{CT/p})$ under BPO in $\mathcal{O}(1)$ amortized time complexity as T goes to infinity, where C is the cache size and T is the number of requests. Moreover, we show that our policy extends to bipartite caching albeit with a sublinear α-regret for $\alpha = 1-1/e$ and a higher computational cost. The experimental evaluation compares the proposed solution with classic caching policies and validates the proposed approach using both synthetic and real-world request traces.**

***Index Terms*—Caching, Online learning, Follow-the-Perturbed-Leader.**

I. Introduction

Caching techniques are extensively employed in computer systems, serving various purposes such as accelerating CPU performance [1] and enhancing user experiences in content delivery networks (CDNs) [2]. The primary objective of a caching system is to carefully choose files for storage in the cache to maximize the proportion of file requests that can be fulfilled locally. This approach effectively minimizes the dependence on remote server retrievals, which can be costly in terms of delay and network traffic. The presence of caching systems facilitates more efficient data delivery and leads to enhanced overall system performance, especially with the widespread adoption of traffic-intensive applications such as virtual and augmented reality [3], or edge video analytics [4].

Caching policies have been thoroughly investigated under varied assumptions concerning the statistical regularity of file request processes [5], [6]. However, real-world request sequences tend to deviate from these theoretical models, especially when aggregated over small geographic areas [7]. This deviation has inspired the exploration of online learning algorithms, beginning with the work of Paschos et al. [8], who proposed the Online Gradient Descent (`OGD`) algorithm [9] for caching. Online learning algorithms exhibit robustness to varying request process patterns, as they operate under the assumption that requests may be generated by an adversary.

In this context, the main metric of interest is the *regret*, which is the difference between the cost incurred by a given online caching algorithm (e.g., the expected number of cache misses) and the cost of the optimal static cache allocation with hindsight, i.e., with knowledge of the future requests over a fixed number of requests. The primary objective is then to design *no-regret* algorithms, i.e., online policies whose regret grows sublinearly with the total number of requests [8].

Several no-regret caching policies have been proposed in the literature, drawing on well-known online learning algorithms such as Online Gradient Descent (`OGD`) [8], Online Mirror Descent (`OMD`) [10], Follow-the-Regularized-Leader (`FRL`) [11], and Follow-the-Perturbed-Leader (`FPL`) [12]. However, most no-regret policies rely on computationally expensive operations and assume access to the complete history of past requests. In practice, many scenarios require caching policies to operate under *partial observations* of the request process.

For instance, in *femtocaching* systems [13], a central base station manages the content of multiple caches located at smaller cell base stations, known as *helpers*. While helpers can independently select their cache content, performing computationally intensive tasks, such as those required by no-regret algorithms, often exceeds their processing capabilities. Consequently, such operations may need to be offloaded to the base station, which has substantially greater computational resources. However, under standard conditions, the base station only has visibility into cache misses at the helpers. Gaining complete knowledge of past file requests would necessitate continuous communication between the helpers and the base station, thereby introducing significant overhead. In another instance, partial observations can result from request routing mechanisms in CDNs. For example, requests are often routed—via DNS redirection or similar techniques—to caches believed to store the requested items, thereby restricting the caching policy's visibility into the complete request process.

To the best of our knowledge, only [14] and [15] have explored caching policies under a specific partial observation regime, where the cache is only aware of requests for items it stores. The former focuses exclusively on scenarios where requests follow a stationary stochastic process. The latter avoids making statistical assumptions about the request process but proposes a policy with an amortized time complexity that is

quadratic in the catalog size and a regret bound that scales linearly with the same parameter.

A. Contributions

This paper investigates caching under a partial observability framework, which we refer to as the Bernoulli Partial Observability (BPO) regime, where the caching policy observes each request—whether for cached or noncached files—with probability p. In the context of the two motivating examples discussed earlier, the probability p corresponds to the fraction of requests forwarded from the helpers to the base station in the *femtocaching* scenario, and to the fraction of CDN requests routed to a specific cache in the CDN setting.

Our main contribution is to prove that simple modifications of the FPL *caching policy yield regret that scales optimally in the cache size, the catalog size, and the trace length, and as* $\mathcal{O}\left(1/\sqrt{p}\right)$ *with respect to the observation probability* p *under BPO. Moreover, this new policy, which we call Noisy-Follow-the-Perturbed-Leader (*NFPL*), has parameters that enable a trade-off among the expected regret, the variability of the regret, and the amortized time complexity. In particular, with a specific configuration of these parameters,* NFPL *is the first no-regret caching policy with an amortized time complexity of* $\mathcal{O}(1)$ *as the total number of requests approaches infinity. We also show that our policy extends to bipartite caching, but at the cost of sublinear* α*-regret guarantees with* $\alpha = 1 - 1/e$ *instead of standard regret guarantees, and higher computational complexity.*

The regret bound of NFPL is presented in Theorem 2. To prove this result, we study an online optimization framework where noisy estimates of linear loss functions are revealed sequentially, e.g., noise is due to partial observability of the loss functions. The goal is to dynamically adjust decisions to minimize the cumulative sum of the actual loss functions. We establish sufficient conditions under which NFPL in this more restrictive setting achieves sublinear regret. In particular, these conditions are met by NFPL under BPO, confirming its no-regret property in that context.

The NFPL policy, which is described in Alg. 1, closely resembles existing FPL caching approaches in the literature [12], [14], [15]. Specifically, it leverages request counts and randomly generated vectors to guide its decisions. This allows NFPL to cache relevant files that a greedy policy, such as Least-Frequently-Used (LFU), might overlook. However, NFPL introduces new features to balance regret and time complexity. It utilizes a batching approach, where cache updates occur only after collecting a batch of requests, similarly to some previously proposed no-regret policies [10], [16]. Furthermore, NFPL abstains from updating the cache state with a probability q, a tunable parameter. It also allows for controlled temporal correlations of its random vectors. In particular, we show that a specific coupling—inspired by the FPL-variant Follow-The-Lazy-Leader [17]—of these noise vectors leads to an efficient implementation that infrequently changes the cache state. Indeed, we bound the probability of such changes, and demonstrate in Theorem 3 that NFPL is the first no-regret caching policy with $\mathcal{O}(1)$ amortized time complexity when the total number of requests grows infinitely.

We complement the theoretical findings on the regret and time complexity of NFPL with simulations on two synthetic traces with Zipf-distributed popularities—one has stationary content popularity patterns and the other is adversarial—and a real-world trace from Akamai [18]. Simulations evaluate the performance of NFPL under BPO in terms of execution time and average and variance of the miss ratio. Notably, the NFPL caching policy performs exceptionally well on the adversarial trace, significantly outperforming alternative approaches. These numerical results underscore NFPL's robustness, even in the presence of non-stationary request sequences.

This paper extends our previous work [19] by significantly improving the dependency of the regret bound on p from $1/p$ to $1/\sqrt{p}$ and extending NFPL to bipartite caching. We also show that a careful choice of correlations between the noise vectors in NFPL leads to $\mathcal{O}(1)$ amortized time complexity while maintaining the regret guarantees. Furthermore, the numerical results on the adversarial trace with skewed file popularity, highlighting the robustness of NFPL, are new.

B. Road map

The rest of the paper is organized as follows: Section II formally describes the caching problem tackled in this paper, and Section III provides background details and discusses related work. The NFPL algorithm and its regret and time complexity guarantees are presented in Section IV. The extension to bipartite caching is discussed in Section V. A numerical evaluation of our approach is shown in Section VI. Finally, Section VII concludes the paper. Detailed proofs and additional numerical results are available in the supplementary material.

II. Problem Formulation

In what follows, the notation $[n]$ designates the set $\{1, \ldots, n\}$ for any positive integer n. As commonly used, $\mathbb{1}(F)$ stands for the indicator function that F is true. We use bold notation to represent vectors and matrices.

We consider a server storing a set $\mathcal{I}$ of N files and a single cache memory, with limited computational capabilities, that can store up to $C < N$ files from $\mathcal{I}$. A sequence of requests for items in $\mathcal{I}$ of length T, denoted $\mathbf{f} = (f_s)_{s\in[T]}$, arrives at the cache. If f_t is available in the cache at step t, the request is a *hit*, and the cache serves the request. Otherwise, it is a *miss*, and the cache forwards the request to the server.

A caching policy $\mathcal{A}$ decides the content of the cache at time t, denoted $S_{\mathcal{A}}(t)$, based on past requests $f_1, \ldots, f_{t-1}$. Algorithm $\mathcal{A}$ samples $S_{\mathcal{A}}(t)$ from a probability distribution $\mathcal{P}_{\mathcal{A}}(t)$ over the space $\{S \subset \mathcal{I} : |S| = C\}$, which we denote as $\binom{\mathcal{I}}{C}$. We adopt the *oblivious* adversarial model, wherein the request sequence can be arbitrary but must be independent of the random choices $(S_{\mathcal{A}}(t))_t$. In this model, the request sequence of fixed length T can be sampled from an arbitrary probability distribution, thus covering typical stationary and non-stationary stochastic models. A common performance metric for an algorithm $\mathcal{A}$ under adversarial models is its regret, denoted as $\mathcal{R}_T(\mathcal{A})$, and defined as follows,

$$\mathcal{R}_T(\mathcal{A}) \triangleq \sup_{\mathbf{f}\in\mathcal{I}^T} \left(\mathbb{E}_{\boldsymbol{\mathcal{P}}_{\mathcal{A}}}\left[M_{\mathcal{A}}(\mathbf{f})\right] - M^*(\mathbf{f})\right), \tag{1}$$

TABLE I: Table of notation.

Caching problem	
$\mathcal{I}$	set of files
$N = \lvert\mathcal{I}\rvert$	catalog size
C	cache capacity
T	total number of requests
$\mathbf{f} = (f_t)_{t\in[T]}$	sequence of requests
$S_{\mathcal{A}}(t)$	files stored by the policy $\mathcal{A}$
δ_t	observability of the request by $\mathcal{A}$
p	success probability of $(\delta_t)_{t\in[T]}$
$\mathcal{R}_T(\mathcal{A})$	regret of policy $\mathcal{A}$
NFPL	
β_t	sampling the request at step t
q	success probability of $(\beta_t)_{t\in[T]}$
B	batch size in NFPL
$\boldsymbol{\gamma}_t = (\gamma_{t,f})_{f\in\mathcal{I}}$	randomly generated noise vectors
η	parameter of $(\boldsymbol{\gamma}_t)_{t\in[T]}$
Online linear learning	
$\mathcal{X}$	decision set
$\mathcal{B}$	costs set
N	dimension of vectors in $\mathcal{X}$ and $\mathcal{B}$
T	time horizon
$\mathbf{r}_t$	cost vector at step t
$\mathbf{x}_t$	decision vector
$\langle \mathbf{r}_t, \mathbf{x}_t\rangle$	cost paid
$\mathbf{r}_{1:t}$	sum of $\mathbf{r}_s$ for all values of s from 1 to t
$M(\mathbf{r})$	value of $\mathbf{x}$ in $\mathcal{X}$ that minimizes $\langle \mathbf{r}, \mathbf{x}\rangle$
$\mathcal{R}_T(\mathcal{A})$	regret of algorithm $\mathcal{A}$
$\hat{\mathbf{r}}_t$	noisy cost vectors estimates
$\hat{\mathcal{B}}$	state space of $(\hat{\mathbf{r}}_t)_{t\in[T]}$

where $\boldsymbol{\mathcal{P}}_{\mathcal{A}} = (\mathcal{P}_{\mathcal{A}}(t))_{t\in[T]}$, $M_{\mathcal{A}}(\mathbf{f}) \triangleq \sum_{t=1}^{T} \mathbb{1}\left(f_t \notin S_{\mathcal{A}}(t)\right)$ is the number of misses of $\mathcal{A}$, and $M^*(\mathbf{f}) = \min_{S\in\binom{\mathcal{I}}{C}} \sum_{t=1}^{T} \mathbb{1}\left(f_t \notin S\right)$ is the number of misses of the optimal static caching policy with knowledge of $\mathbf{f}$. The purpose is to design a no-regret policy—one whose regret grows sublinearly in T.

While many no-regret caching policies have been proposed recently, most require the caching policy $\mathcal{A}$ to be aware of all past requests for the cache. As mentioned in the introduction, in the *femtocaching* example, the base station (BS), which stores all the files in $\mathcal{I}$, manages the content of multiple caches. Under standard conditions, the BS only has visibility into cache misses. Thus, we define three partial observability regimes of the requests, namely *Bernoulli Partial Observability* (BPO), Hit Partial Observability (HPO), and Miss Partial Observability (MPO), where the caching policy $\mathcal{A}$ has access to a *subset of requests*.

BPO. The caching policy observes each request with probability p. We model the availability of the request at step t via a Bernoulli random variable δ_t. If $\delta_t = 1$, $\mathcal{A}$ is aware of the request at that particular step. We assume that $(\delta_t)_{t\in[T]}$ are independent and identically distributed (i.i.d.) with success probability p.

HPO. The caching policy observes all the requests for non-cached files and observes each of the requests for cached ones with probability p. Formally, the request at step t is observed if and only if $\delta_t^H = 1$, where $\delta_t^H = \mathbb{1}\left(f_t \notin S_{\mathcal{A}}(t)\right)$ (miss), and it is equal to δ_t otherwise (hit), where δ_t is the Bernoulli random variable defined for BPO. This setting reflects *femtocaching*, where only a fraction p of hit requests are forwarded from the edge caches to the BS.

MPO. The converse of HPO: all hits are revealed, and each miss is observed with probability p.

As an example, if $\mathcal{I} = \{a, b\}$, $C = 1$, $\mathbf{f} = (a, b, a, b)$, the Bernoulli mask in BPO is $(0, 1, 1, 0)$, the initial state is $\{a\}$, and the algorithm is LRU, then the Bernoulli masks under HPO and MPO are $(0, 1, 1, 1)$ and $(1, 1, 1, 0)$, respectively.

Why BPO is the hardest regime. As $\delta_t^{\mathrm{H}} \geq \delta_t$ for every t, the size of the subset of requests available to the algorithm under HPO is larger. Note that, by deliberately sampling requests for cached files with probability p under HPO (via δ_t), the observed request process becomes identical to that under BPO, since each request—hit or miss—is then observed independently with probability p. Thus, designing a no-regret caching policy under BPO enables achieving sublinear regret under MPO and HPO. The same considerations hold for MPO. Therefore, we focus on achieving sublinear regret under BPO.

In addition to the sublinear regret property, it is important to take into account the computational cost of the caching policy. To this end, we use as a performance metric the amortized time complexity of $\mathcal{A}$ over T rounds, that is, the average running time of $\mathcal{A}$ per request.

III. Background and Related Work

A. Online linear learning and FPL

In online learning, at each round t the agent selects a decision vector $\mathbf{x}_t \in \mathcal{X} \subset \mathbb{R}^N$ and then experiences a loss. In the *linear* setting, the loss is $\langle \mathbf{x}_t, \mathbf{r}_t\rangle$, where $\mathbf{r}_t$ is a vector from $\mathcal{B} \subset \mathbb{R}^N$ and $\langle \mathbf{r}, \mathbf{x}\rangle \triangleq \sum_{i=1}^{N} r_i x_i$ denotes the scalar product of the two vectors $\mathbf{r}$ and $\mathbf{x}$.

In this context, the metric of performance for an algorithm $\mathcal{A}$ selecting a decision vector $\mathbf{x}_t = \mathcal{A}(\mathbf{r}_1, \ldots, \mathbf{r}_{t-1})$ is the regret defined as follows,

$$\mathcal{R}_T(\mathcal{A}) = \sup_{\{\mathbf{r}_1,\ldots,\mathbf{r}_T\}} \left\{ \mathbb{E}\left[\sum_{t=1}^{T} \langle \mathbf{r}_t, \mathbf{x}_t\rangle\right] - \min_{\mathbf{x}\in\mathcal{X}} \langle \mathbf{x}, \sum_{t=1}^{T} \mathbf{r}_t\rangle \right\}. \tag{2}$$

The objective is to design an algorithm with sublinear regret, $\mathcal{R}_T(\mathcal{A}) = o(T)$. These algorithms are commonly known as *no-regret* algorithms since their time-average cost approaches the optimal static policy cost as T grows.

An intuitive solution to minimize the regret, known as Follow-The-Leader (FTL) [20], is to greedily select the state that minimizes the past cumulative cost, i.e., $x_{t+1} = M(\mathbf{r}_{1:t})$, where $M(\mathbf{r})$ denotes an arbitrary element of $\arg\min_{\mathbf{x}\in\mathcal{X}} \langle \mathbf{r}, \mathbf{x}\rangle$ for any vector $\mathbf{r}$, and $\mathbf{r}_{1:t} \triangleq \sum_{s=1}^{t} \mathbf{r}_s$ represents the aggregate sum of a given sequence of vectors $(\mathbf{r}_1, \ldots, \mathbf{r}_t)$. Although FTL proves optimal when $(\mathbf{r}_t)_t$ are sampled from a stationary distribution, it unfortunately yields linear regret in adversarial settings [20].

Follow-the-Perturbed Leader (FPL) improves the performance of FTL by incorporating, at each time step t, a noise vector $\boldsymbol{\gamma}_t$ of size N, whose components are i.i.d. random variables. Formally,

$$\mathbf{x}_t(\mathtt{FPL}) = M(\mathbf{r}_{1:t-1} + \boldsymbol{\gamma}_t). \tag{3}$$

Let $D \triangleq \sup_{\mathbf{x},\mathbf{y}\in\mathcal{X}} \|\mathbf{x} - \mathbf{y}\|_1$ be the diameter of the decision set $\mathcal{X}$, A be a bound on the norm 1 of vectors in the cost

set $\mathcal{B}$, and R be a bound on $\langle \mathbf{x}, \mathbf{r} \rangle$ for any $(\mathbf{x}, \mathbf{r}) \in \mathcal{X} \times \mathcal{B}$. [17, Thm. 1.1] shows that FPL is a no-regret algorithm for the broad class of online linear learning problems when the components of $\boldsymbol{\gamma}_t$ are uniform random variables.

Theorem 1: [17, Thm. 1.1] Let $\boldsymbol{\gamma}_t$ be sampled from a multivariate uniform distribution with independent components over $[0, \eta]^N$, with $\eta = \sqrt{RAT/D}$, then:

$$\mathcal{R}_T(\texttt{FPL}) \leq 2\sqrt{RADT}. \tag{4}$$

This no-regret property of FPL extends to other probability distributions for $\boldsymbol{\gamma}_t$, including exponential and Gaussian distributions [17], [22], [23]. These extensions may yield different regret bounds depending on the characteristics of the sets $\mathcal{X}$ and $\mathcal{B}$. Interestingly, FPL can be adapted to maintain its no-regret property in the *multi-armed bandit* problem [24], where decisions are represented by one-hot vectors and only the cost $\langle \mathbf{r}_t, \mathbf{x}_t \rangle$ is observed, rather than the cost vector $\mathbf{r}_t$.

In contrast to the online linear setting, where the decision set $\mathcal{X}$ could be arbitrary but the loss functions $\langle \mathbf{r}_t, \cdot \rangle$ are linear, another well-studied framework considers the case where $\mathcal{X}$ is convex while allowing for more general convex loss functions. This setting falls within the domain of online convex optimization, where algorithms such as OGD, OMD, and FRL are known for their no-regret guarantees [25].

B. Online learning for caching

The caching problem naturally fits into the online linear learning framework: the set $\mathcal{X}$ represents the possible cache allocations and the set $\mathcal{B}$ represents the possible requests. For example, one may define $x_{t,i} = 1$ if $i \in S_{\mathcal{A}}(t)$ and $x_{t,i} = 0$ otherwise, and $r_{t,i} = 1$ if $f_t = i$ and $r_{t,i} = 0$ otherwise.

This formulation was considered in [12], where the authors used FPL with Gaussian noise, following the spirit of [22], [24], but rather than directly adapting existing FPL results to the caching problem, they introduced a different proof technique. This approach allowed them to improve the dependence of the regret bound on N and C (see [12, Thm. 3]).

To leverage online convex optimization algorithms such as OGD and OMD, [8], [10] adopt an alternative modeling in which $x_{t,i}$ represents the fraction of the stored file, allowing for a convex decision set $\mathcal{X}$. In both works, the regret bounds and the algorithmic implementations had to be carefully adapted to account for the specific structure of caching.

[15] is the only work that considered the partial observability of adversarial requests. Specifically, the authors augmented FPL with a *Geometric Resampling* procedure [26], to design a no-regret policy, that we call FPLGR, under MPO with $p \in [0, 1]$.

Table II summarizes the requests' observability regime, regret bound, and time complexity, for the above-mentioned caching policies, namely, FPL [12], [21], OGD [8], OMD [10], FPLGR [15], and our proposed policy, NFPL.

NFPL. Our policy operates under MPO, when $p > 0$. It leverages the fraction of observed information ($p \in (0, 1]$) to achieve a time complexity of $\mathcal{O}(1)$ as $T \to \infty$, which is significantly better than the time complexity of existing caching policies, even when $p = 1$. Furthermore, the regret of NFPL is optimal because [12, Thm. 2] proves a lower bound of $\Omega(\sqrt{CT})$ on the regret of any caching policy.

FPL and FPLGR. The FPL caching policy achieves the best previously known time complexity of $\mathcal{O}(\ln C)$ when $p = 1$ [12], [21]. FPLGR is an extension of FPL to maintain the no-regret property under MPO. However, this comes at a significant cost: the regret bound scales linearly with N, and the time complexity increases to $\mathcal{O}(N^2 \ln N)$.

OGD and OMD. While OGD has an optimal regret bound, it is computationally expensive. OMD was proposed to mitigate this computational burden. In the fractional caching setting, performing cache updates after every time step yields update complexities of $\mathcal{O}(N)$ for OGD and $\mathcal{O}(C)$ for OMD. Conversely, if updates are performed only after collecting a batch of requests, the complexities become $\mathcal{O}(N^2)$ for OGD and $\mathcal{O}(N)$ for OMD. However, for the discrete caching problem addressed in this paper under BPO with $p = 1$, both OGD and OMD require an additional routine of complexity $\mathcal{O}(N)$ to convert a fractional cache state into a discrete one [10]. As a result, the overall time complexity for both algorithms is $\mathcal{O}(N)$. Recently, [27] proposed a no-regret online gradient-based caching policy with a time complexity of $\mathcal{O}(\ln(N))$, achieved under a relaxed cache capacity constraint satisfied in expectation, i.e., $\mathbb{E}\left[|S_{\mathcal{A}}(t)|\right] = C$.

More generally, recent works have shown that online learning techniques can enhance edge caching performance. For example, [16], [28], [29] propose optimistic no-regret algorithms that leverage neural network predictions about future requests, [30] proposes a randomized caching algorithm with a low *dynamic* regret, and [31] uses online learning to estimate fetching costs.

IV. The NFPL Caching Policy

No-regret caching policies typically rely on knowing the complete history of past requests. For instance, the FPL caching policy from [12] uses this knowledge to compute exact request counts for each file. However, this approach is incompatible with the BPO regime, formally defined in Section II, where only approximate request counts are available. To address this limitation, we propose an FPL-based caching policy, called NFPL, with sublinear regret under the BPO regime. The rest of this section is organized as follows: Section IV-A describes the dynamics of each NFPL algorithm, Section IV-B establishes the regret guarantees, Section IV-C analyzes the time complexity, and Section IV-D examines the trade-off between regret and computational efficiency.

A. Policy description

The general behavior of an NFPL policy is outlined in Algorithm 1. Three specific instances are presented later, which differ only in the definition of the **UpdateNoiseVectors** function.

At step $t = 0$, an NFPL algorithm generates a noise vector $\boldsymbol{\gamma}_0 = (\gamma_{0,f})_{f \in \mathcal{I}}$ from the multivariate uniform distribution with independent components, constrained within the range $[0, \eta]^N$, where $\eta = \sqrt{\frac{pq + (pq)^2(B-1)T}{2C}}$. Algorithms

TABLE II: Comparison of no-regret caching policies.

Algorithm	Regime	Regret Bound	Time Complexity
FPL [12], [21]	BPO, $p = 1$	$\mathcal{O}((\ln(N)^{1/4})\sqrt{CT})$ [12, Thm. 3]	$\mathcal{O}(\ln(C))$
OGD [8]	BPO, $p = 1$	$\mathcal{O}(\sqrt{CT})$ [8, Thm. 1]	$\mathcal{O}(N)$
OMD [10]	BPO, $p = 1$	$\mathcal{O}(C\sqrt{\ln(N)T})$ [10, Cor. 2]	$\mathcal{O}(N)$
FPLGR [15]	MPO, $p \in [0, 1]$	$\mathcal{O}(NC\sqrt{T})$ [15, Thm. 1]	$\mathcal{O}(N^2 \ln(N))$
NFPL [this paper]	BPO, $p \in (0, 1]$	$\mathcal{O}(\sqrt{CT/p})$ [Thm. 2]	$\mathcal{O}(1)$ as $T \to +\infty$ [Thm. 3]

Algorithm 1: NFPL

Input: Observed requests $\{f_t : t \in [T], \delta_t = 1\}$, cache capacity C, batch size B, probabilities p and q.
Output: C-sized set of stored content at each time step.
$\hat{\mathbf{n}}(0) = (\hat{n}_1(0), \ldots, \hat{n}_N(0))) \leftarrow \mathbf{0}$
$\eta \leftarrow \sqrt{\frac{pq+(pq)^2(B-1)T}{2C}}$
$\boldsymbol{\gamma}_0 \sim$ **Unif** $([0, \eta])^N$
$S(0) \leftarrow \arg\max_{\substack{S\subset[N]:\\|S|=C}} \sum_{j\in S} \hat{n}_j(0) + \gamma_{0,j}$
flag $\leftarrow 0$
for $t = 1$ to T **do**
  $\beta_t \sim$ **Bernoulli**(q)
  if $\delta_t = 1$ and $\beta_t = 1$ **then**
    $\hat{n}_{f_t}(t) \leftarrow \hat{n}_{f_t}(t-1) + 1$
    flag $\leftarrow 1$
  end if
  if $t\%B = 0$ and flag $= 1$ **then**
    $\boldsymbol{\gamma}_t \leftarrow$ **UpdateNoiseVectors**$(\boldsymbol{\gamma}_0, \eta, \hat{\mathbf{n}}(t))$
    $S(t) \leftarrow \arg\max_{\substack{S\subset[N]:\\|S|=C}} \sum_{j\in S} \hat{n}_j(t) + \gamma_{t,j}$
    flag $\leftarrow 0$
  else
    $S(t) \leftarrow S(t-1)$
  end if
end for

from this family maintain approximate counts of past requests for each file, denoted $\hat{\mathbf{n}}(t) = (\hat{n}_f(t))_{f\in\mathcal{I}}$, and expressed as $\hat{n}_f(t) = \sum_{t=1}^{T} \delta_t\beta_t \cdot \mathbb{1}(f_t = f)$, where δ_t and β_t are Bernoulli random variables with success probabilities denoted p and q, respectively. The variable δ_t indicates whether the request at step t is observed, while β_t is an algorithm-generated variable that reduces the frequency of cache updates. The role of the parameter q will be clarified later, in the discussion on time complexity and the tradeoff between regret and runtime in Sections IV-C and IV-D. Moreover, the cached content is only updated when the time step t is a multiple of B and if there has been at least one request in the B most recent time steps. At these time steps, which we denote by $\mathcal{T}$, a new noise vector $\boldsymbol{\gamma}_t$, whose probability distribution is identical to that of $\boldsymbol{\gamma}_0$, is used to determine the cache content. Specifically, the C files with the highest perturbed approximate counts: $\hat{n}_f(t) + \gamma_{t,f}$, for $f \in \mathcal{I}$, are stored in the cache. We consider the following three instances of NFPL:

1) **S-NFPL.** The noise vector remains static over time, i.e. $\boldsymbol{\gamma}_t = \boldsymbol{\gamma}_0,\ \forall t \in \mathcal{T}$.
2) **D-NFPL.** The noise vector is drawm from the multivariate uniform distribution on $[0, \eta]^N$ with independent components, independently of all previous noise vectors ($(\boldsymbol{\gamma}_t)_{t\in\mathcal{T}}$ are i.i.d.).
3) **L-NFPL.** The noise vectors are correlated and depend on $\boldsymbol{\gamma}_0$ and the approximate counts $\hat{\mathbf{n}}(t)$ as follows,

$$\boldsymbol{\gamma}_t = \boldsymbol{\gamma}_0 + \eta \left\lceil \frac{\hat{\mathbf{n}}(t) - \boldsymbol{\gamma}_0}{\eta} \right\rceil - \hat{\mathbf{n}}(t). \tag{5}$$

We observe that for all three instances, for any $t \in \mathcal{T}$, the noise vector $\boldsymbol{\gamma}_t$ follows a multivariate uniform distribution over $[0, \eta]^N$ with independent components (see Section B in the supplementary material). The instances differ only for the temporal correlations of these vectors at different time instants.

B. Regret guarantees

Any NFPL caching policy enjoys $\mathcal{O}(\sqrt{T})$ regret:

Theorem 2: If $\boldsymbol{\gamma}_t \sim$ **Unif** $([0, \eta])^N$ for all $t \in \mathcal{T}$, an NFPL caching policy achieves sublinear regret:

$$\mathcal{R}_T(\texttt{NFPL}) \leq 2\sqrt{\frac{2C(1 + pq(B-1))T)}{pq}} + \Theta\left(\frac{1}{\sqrt{T}}\right).$$

The result applies to all three instances introduced above.

Difficulty in proving Theorem 2. When $pq = 1$, the exact request count for each file is available, and one can show that NFPL is a special case of FPL with uniform noise, applied to an online linear learning formulation of the caching problem. This correspondence allows us to invoke [17, Theorem 1.1(a)] and improve the regret bound compared to the FPL-based caching policy of [12] (see Table II).

However, when $pq < 1$, the update rule of NFPL differs from that of FPL-based caching policies because it relies on approximate request counts $\hat{\mathbf{n}}$, which prevents a direct application of existing online learning results for FPL. Moreover, it is not straightforward to relate the distribution of the cache content selected by NFPL, $S_{\texttt{NFPL}}(t)$, to that of FPL, $S_{\texttt{FPL}}(t)$.

The difficulty of relating these two distributions arises already in the simple case of two files and cache capacity $C = 1$. Here, the cache can either hold file 1 or file 2. For FPL, the probability of caching file 1 is equal to $\Pr(\gamma_{t,1} - \gamma_{t,2} > n_2 - n_1)$, which can be derived in closed form because $\gamma_{t,1}$ and $\gamma_{t,2}$ are independent and uniformly distributed. However, in NFPL, the corresponding probability is $\Pr(\gamma_{t,1} - \gamma_{t,2} - \hat{n}_2(t) + \hat{n}_1(t) > 0)$, which involves a sum of uniform and binomial random variables. This sum has no closed-form distribution, which makes it difficult to relate the probability in NFPL to that in FPL, especially for general values of N and C.

Sketch of the proof of Theorem 2: The key idea is to consider a variant of the online linear learning framework

from Section III-A, where the agent does not observe the cost vectors $\mathbf{r}_t$ directly but only noisy estimates $\hat{\mathbf{r}}_t$. This variation captures the partial observability inherent in our caching problem. We first show that, under suitable assumptions on these estimates, FPL retains sublinear regret guarantees when the cost vectors $\mathbf{r}_t$ are replaced by their estimates $\hat{\mathbf{r}}_t$. To establish the regret bound for NFPL, we then introduce an auxiliary online learning problem in which the update rule of FPL coincides with that of NFPL in the original problem. We then prove that $\mathcal{R}_T(\texttt{NFPL}) \leq \hat{\mathcal{R}}_T(\texttt{FPL})$, where $\hat{\mathcal{R}}_T(\mathcal{A})$ denotes the regret of algorithm $\mathcal{A}$ in the auxiliary problem.

This reasoning yields a sublinear regret bound for NFPL that scales as $1/p$. Such a bound would hold even if the sequence of observed requests were fully adversarial. In our setting, however, the adversary only controls the sequence of requests, while the observation process is stochastic. By exploiting this structure, we can further tighten the bound to achieve $1/\sqrt{p}$ scaling.

The detailed proof is presented in the supplementary material (Section A). We observe that our analysis of NFPL extends beyond caching and applies to a broader class of online linear learning problems. ∎

NFPL in the general framework of online linear learning. A fundamental step in the proof of Theorem 2 is to show first that, for any online linear learning problem (with caching as a particular case), substituting the exact cost vectors by their respective noisy estimates within the FPL algorithm leads to sublinear regret under specific assumptions on the estimates. A simple and notable case of these assumptions holds when $\hat{\mathbf{r}}_t = b_t \cdot \mathbf{r}_t$, with $(b_t)_t$ being i.i.d. Bernoulli random variables. In this particular case, NFPL can be viewed as a sublinear regret algorithm that limits how often the adversary's feedback is revealed. This feature of NFPL in this setting makes it a *selective sampling* algorithm [32], also referred to as a *label-efficient forecaster* [33]. Such algorithms are motivated by practical scenarios where it is costly to observe the cost vector chosen by the adversary, prompting the need to reduce these observations while maintaining regret guarantees. However, to the best of our knowledge, most label-efficient algorithms in the literature are variations of the multiplicative weights algorithm, typically applied to the expert problem, which is a particular case of the online linear learning framework covered by NFPL.

Tightness of the regret bound. NFPL achieves $\mathcal{O}\left(\sqrt{CT/p}\right)$ regret, where the factor $1/\sqrt{p}$ reflects the performance loss caused by the partial observability of the requests. This dependency of the regret bound on p is the best that any online algorithm for the expert problem can achieve when only a fraction p of past labels is observed [34, Thm. 6.4]. We stress that a naive reduction of caching to an expert problem introduces $\binom{N}{C}$ experts—one per feasible cache configuration—yielding both an impractical update cost and a regret term that scales at least as $\log\binom{N}{C}$. In contrast, NFPL regret bound is independent of N and supports $\mathcal{O}(1)$ amortized time complexity for large T (see Theorem 3). Furthermore, when $p = 1$ and $B = 1$, the lower bound of [12, Thm. 2] is $\Theta(\sqrt{CT})$ proving that *NFPL is asymptotically optimal in C, N and T.*

Advantages of the parameters B and q. The parameter q enables memory reduction, since NFPL only increments the counters $\hat{\mathbf{n}}$ at step t when the Bernoulli random variable β_t is equal to 1—δ_t also needs to be equal to 1—which occurs with probability q. As a result, the maximum entry in the absolute value of the vector $\hat{\mathbf{n}}(T) + \boldsymbol{\gamma}_T$ is $\mathcal{O}(pqT)$, in expectation, compared to $\mathcal{O}(T)$ in the standard FPL. Therefore, NFPL saves memory by storing a vector with smaller entries. Finally, the parameters B and q allow to reduce the frequency of cache updates, which can lead to reduced communication cost between the main memory and the cache.

C. Amortized time complexity

The three variants of NFPL are characterized by distinct joint distributions of the noise vectors $(\boldsymbol{\gamma}_t)_{t\in[T]}$. Theorem 3 shows for each variant the effect of the parameters B and q on the amortized time complexity.

Theorem 3: The amortized time complexity of the three variants of NFPL is as follows:

- S-NFPL: $\mathcal{O}\left(1 + pq \ln C\right)$.
- D-NFPL: $\mathcal{O}\Big(1 + N \ln N/B\Big)$.
- L-NFPL: $\mathcal{O}\left(1 + pq \ln(C)\sqrt{C}/\sqrt{BT}\right)$.

Sketch of the proof: The most computationally intensive task in NFPL variants is identifying the top C files based on their perturbed counts, $\hat{\mathbf{n}} + \boldsymbol{\gamma}$, denoted as $\mathbf{m}$. In D-NFPL, the noise vectors $(\boldsymbol{\gamma}_t)_{t\in[T]}$ are consistently regenerated, necessitating a sorting operation to determine the top C files, which accounts for the $\mathcal{O}(N \ln N)$ term in its amortized time complexity. Conversely, in S-NFPL and L-NFPL, only one component of the vector $\mathbf{m}(t) = \hat{\mathbf{n}}(t) + \boldsymbol{\gamma}_t$ is updated between consecutive time steps, allowing the use of a *heap* data structure to track the top C files efficiently. The time complexity of an insertion/deletion in a heap is $\mathcal{O}(\ln C)$, explaining the $\ln C$ term in their amortized time complexity. In L-NFPL, it is even possible for $\mathbf{m}(t) = \mathbf{m}(t+1)$, and we show that the probability of this event occurring is on the order of $\mathcal{O}(\sqrt{C/(BT)})$, which justifies the amortized time complexity of L-NFPL. The detailed proof is in the supplementary material (Section B). ∎

Theorems 2 and 3 show that L-NFPL is the first $\mathcal{O}(\sqrt{T})$-regret caching policy with $\mathcal{O}(1)$ amortized time complexity as T goes to infinity, considerably improving over the best previously achieved amortized time complexity of $\mathcal{O}(\ln C)$.

D. Trade-off between the regret and the time complexity

We discuss in this section how batching, i.e., updating the cache after B requests, and sampling the requests with rate q, balance the regret. Finally, we conclude by comparing the three NFPL variants in terms of the variance of the miss ratio, which is not reflected in the regret metric expressed in (1), where only the expectation is considered.

Batching. Previous works proposed batching to reduce the computational cost of no-regret caching policies by roughly a factor of $1/B$ and resulting in a regret bound of $\mathcal{O}(\sqrt{BT})$ [10], [28]. Our paper is the first to extend a similar result for an FPL-based caching policy when the noise vector is regenerated in an i.i.d. fashion (D-NFPL), leading to a regret bound of $\mathcal{O}(\sqrt{BT})$ and an amortized time complexity of $\mathcal{O}(N \ln N/B)$. However, batching has a marginal effect on the amortized time complexity of S-NFPL.

Sampling. Having just a fraction pq of past requests being used by the decision-making process in S-NFPL allows the amortized time complexity to be lowered to $\mathcal{O}(1 + pq \ln C)$, instead of $\mathcal{O}(\ln C)$, but results in a regret bound that grows by a factor of $1/\sqrt{pq}$. A similar trade-off occurs in L-NFPL, where the regret bound is $\mathcal{O}(\sqrt{T/pq})$, and the amortized time complexity is $\mathcal{O}(1 + pq/\sqrt{T})$, ignoring the dependence on C and B. Consequently, for a $\mathcal{O}(\sqrt{T})$ no-regret caching policy that employs batching to match L-NFPL's computational cost, B would need to be set to $\mathcal{O}(\sqrt{T}/pq)$, leading to a regret bound of $\mathcal{O}(T^{3/4}/\sqrt{pq})$, significantly worse than L-NFPL's $\mathcal{O}(\sqrt{T/pq})$ regret for large values of T.

Variance. Although our theoretical results indicate that L-NFPL is the superior choice among the variants, our numerical simulations reveal that D-NFPL exhibits lower variability around the average cost, suggesting that it is more likely to perform consistently close to the average. To understand this aspect, we stress that FPL-based caching policies differ from LFU by randomly assigning a score (γ) to files that reflect their importance, independently of the request count, and then combining this score with the request count to decide if the file is stored. This allows the policy to store not only frequently requested files but also to explore other potentially relevant files, such as those that experience bursts of requests within a specific window of time but do not consistently rank among the top C files. In S-NFPL and L-NFPL, the score (γ) is determined once and remains fixed, whereas in D-NFPL, the score is continuously refined, enabling broader exploration. For instance, files initially given a high score in γ might later prove irrelevant, causing a drop in the performance of the FPL caching policy in comparison to LFU. However, by consistently updating the scores, D-NFPL ensures that relevant files are more likely to be selected, even if they are occasionally missed.

Remark 1 (Practical considerations): NFPL maintains a per-item counter, so its space scales linearly with N, which is prohibitive for large catalogs. A practical workaround is to use approximate counting (e.g., sketch-based data structures) to balance estimation accuracy and memory footprint [35]. A second issue is the configuration of η that depends on the horizon T, which is unknown because the request process is revealed in real time. A practical solution is to introduce a window parameter W and set η as if the time horizon is W. One may then either run NFPL on a sliding-window variant that only accounts for requests within the last W requests or perform exponential forgetting by multiplying all counters by a fixed factor after each window.

V. NFPL for Bipartite Caching

Problem formulation. *Bipartite caching* [13] is a generalization of the single cache model introduced in Section II. In this setting, we have a set $\mathcal{J}$ of J caches, a set $\mathcal{U}$ of U users, and a bipartite graph $\mathcal{G} = (\mathcal{U} \cup \mathcal{J}, \mathcal{E})$ where an edge $(u, j) \in \mathcal{E}$ means that user u can reach cache j. It is in general assumed that a user can reach at maximum d nearby caches with $d \ll J$. An algorithm $\mathcal{A}$ selects the content of each cache j, denoted $\mathcal{S}_{\mathcal{A},j}(t)$, a subset of $\mathcal{I}$ of size C_j. At each time step t, a user $u_t \in \mathcal{U}$ requests a file $f_t \in \mathcal{I}$ from all caches j in $\mathcal{M}_t$, defined as $\mathcal{M}_t \triangleq \{j \in \mathcal{J} : \ (u_t, j) \in \mathcal{E}\}$. The request is then a miss if none of the caches in $\mathcal{M}_t$ stores f_t, i.e., $f_t \notin \cup_{j \in \mathcal{M}_t} \mathcal{S}_{\mathcal{A},j}(t)$. The objective is to design an algorithm $\mathcal{A}$ with a low number of misses.

Offline algorithms. In the single cache model, the optimal static caching policy, given knowledge of $\mathbf{f}$, stores the C files with the largest request counts in $\mathbf{f}$. In bipartite caching, finding the optimal policy is NP-hard when $d \geq 2$, and an efficient randomized $(1-(1-1/d)^d)$-approximation algorithm was proposed [13].

Online algorithms. Because the offline version of bipartite caching is NP-hard, researchers have either proposed heuristics obtained by adapting well-known single-cache policies [36], [37], or proposed randomized online algorithms with performance guarantees within $(1-1/e)$ factor from the optimal, in expectation, for both stochastic and adversarial requests [12], [28], [38]–[41]. For adversarial requests, the α-regret metric, denoted $\mathcal{R}_{T,\alpha}(\mathcal{A})$, has been considered. It is defined as,

$$\mathcal{R}_{T,\alpha}(\mathcal{A}) \triangleq \sup_{\mathbf{f},\boldsymbol{\mathcal{M}}} \left(\alpha H^*(\mathbf{f}, \boldsymbol{\mathcal{M}}) - \mathbb{E}_{\boldsymbol{\mathcal{P}}_{\mathcal{A}}} \left[H_{\mathcal{A}}(\mathbf{f}, \boldsymbol{\mathcal{M}})\right]\right), \quad (6)$$

where $H^*(\mathbf{f}, \boldsymbol{\mathcal{M}})$ is the number of hits of the optimal static caching policy with knowledge of $(\mathbf{f}, \boldsymbol{\mathcal{M}})$, and $H_{\mathcal{A}}(\mathbf{f}, \boldsymbol{\mathcal{M}})$ is the number of hits of $\mathcal{A}$, and $\alpha \leq 1$. The proposed algorithms achieve sublinear $(1 - 1/e)$-regret.

BPO model in bipartite caching. The random variable δ_t of the BPO regime in this case indicates the observability of both the requested file f_t and the nearby caches $\mathcal{M}_t$. In the *femtocaching* scenario, the base station storing all the files in $\mathcal{I}$ manages the content of the J caches, and under standard conditions, the base station has visibility only into cache misses. The probability p of the BPO regime captures then the communication rate between the J caches and the base station about requests served directly by them.

NFPL for fractional bipartite caching under BPO. NFPL with $B = 1$ can be seen as an adaptation of FPL to BPO: the decision vector is updated according to FPL whenever $\delta_t = 1$ and it remains unchanged otherwise. Similarly, any learning algorithm $\mathcal{A}$ for a problem with loss functions $(l_t(\cdot))_t$, can be adapted to the more restrictive setting where l_t is observed with probability p (i.e., the learner observes $\hat{l}_t = \delta_t l_t$). The adaptation of $\mathcal{A}$ to this restrictive setting, denoted $\hat{\mathcal{A}}$, updates according to $\mathcal{A}$ only when $\delta_t = 1$ and otherwise repeats the previous decision. Lemma 1 in the supplementary material (Section A) shows that, whenever $\mathcal{A}$ has a regret bound $E/(\eta T) + \eta F$, where E, F are problem–dependent constants

and η is a tunable parameter, $\hat{\mathcal{A}}$ has sublinear regret scaling as $1/\sqrt{p}$. In a relaxation of bipartite caching, the objective is to learn fractions $x_{f,j,t} \in [0,1]$ of each file f, in each cache j, at time t, subject to $\sum_f x_{f,j,t} = C_j$. This relaxation, called *fractional bipartite caching*, is an online convex optimization problem [12], [40], and thus existing algorithms for convex problems with a regret bound of the form $E/(\eta T) + \eta F$, can be adapted as above to BPO while maintaining sublinear regret guarantees. This is the case for FPL [25, Alg. 16] and OGD [40]. However, applying FPL to problems with non-linear loss functions can be costly, since computing an expectation at every iteration is required.

NFPL for bipartite caching under BPO. In bipartite caching the decision vectors are binary, $x_{f,j,t} \in \{0,1\}$. A common approach is to apply a no-regret algorithm to the fractional relaxation, then round the fractional decision to an integral one at each step via efficient rounding schemes [10], [28], [39]. The rounding step loses at most a factor $(1-1/e)$ in the expected hit count relative to the fractional solution. Hence, coupling any no-regret fractional algorithm adapted to BPO (e.g., $\hat{\text{FPL}}$ or $\hat{\text{OGD}}$) with such rounding produces a policy whose α-regret is sub-linear for $\alpha = 1 - 1/e$.

Heuristics for extending NFPL to bipartite caching under BPO. The NFPL variants described above provide sub-linear regret, but they are computationally demanding and operate each cache independently [12], [40], so nearby caches may end up storing identical files. A common way to coordinate caching decisions is to adapt classical single cache heuristics to the bipartite setting [36], [37]. For example, LRU-One [36]: when a request reaches the d caches in $\mathcal{M}_t$, exactly one cache—chosen uniformly at random, by proximity to the user, or by another simple rule—updates its state with the standard LRU operation, while the others ignore the request. The same idea can be applied to NFPL: designate a single cache in $\mathcal{M}_t$ to perform the NFPL update. The resulting NFPL-One heuristic has a low $\mathcal{O}(d)$ per-request complexity as $T \to \infty$, while reducing the rate of unnecessary duplication, so that neighboring caches tend to hold complementary, rather than overlapping, sets of files. Some numerical results are reported in the supplementary material (Section E).

VI. Numerical Evaluation

We conducted simulations of the NFPL algorithms and other existing policies, using both synthetic and real-world traces. Section VI-A introduces the traces, while Section VI-B discusses the caching baselines. In Section VI-C, we evaluate the effectiveness of our proposed algorithms, in terms of the average miss ratio, the variance around this quantity, and the computational cost. Finally, we show in Section VI-D the effect of sampling on the performance of the NFPL algorithm.

A. Traces

Zipf trace. We generate the requests from a catalog of $N = 10^4$ files, with $I = [N]$, following an i.i.d. Zipf distribution with exponent $\alpha = 1$, i.e., $(f_t)_{t=1}^T$ are i.i.d. random variables and $\Pr(f_t = i) \sim 1/i$. The Zipf distribution is a popular model for the request process in caching [42].

Zipf-RR trace. In this scenario, the catalog size is $N = 10^4$. The total number of requests for each file, $(n_i(T))_{i \in \mathcal{I}}$, follows the multinomial distribution, with the number of trials being T, and the probability associated with each file is given by the Zipf distribution with exponent $\alpha = 1.0$. Next, the files are numbered based on their total number of requests, with file 1 having the most requests and file N the fewest. The order in which these requests occur follows a specific pattern. Initially, the requests begin with file N and proceed in descending order to file 1. This cycle, from N down to 1, repeats until all requests for file N are exhausted. Once file N has no more requests, the sequence shifts to files $N-1$ down to 1, repeating this cycle until the requests for file $N-1$ are depleted. This process continues in the same manner for the remaining files. Eventually, when only file 1 remains, repeated requests for file 1 are made until all of its requests are depleted.

Akamai trace. The request trace, sourced from Akamai CDN as documented in [18], encompasses several days of file requests, amounting to a total of 2×10^7 requests for a catalog comprising $N = 10^3$ files.

B. Caching policies

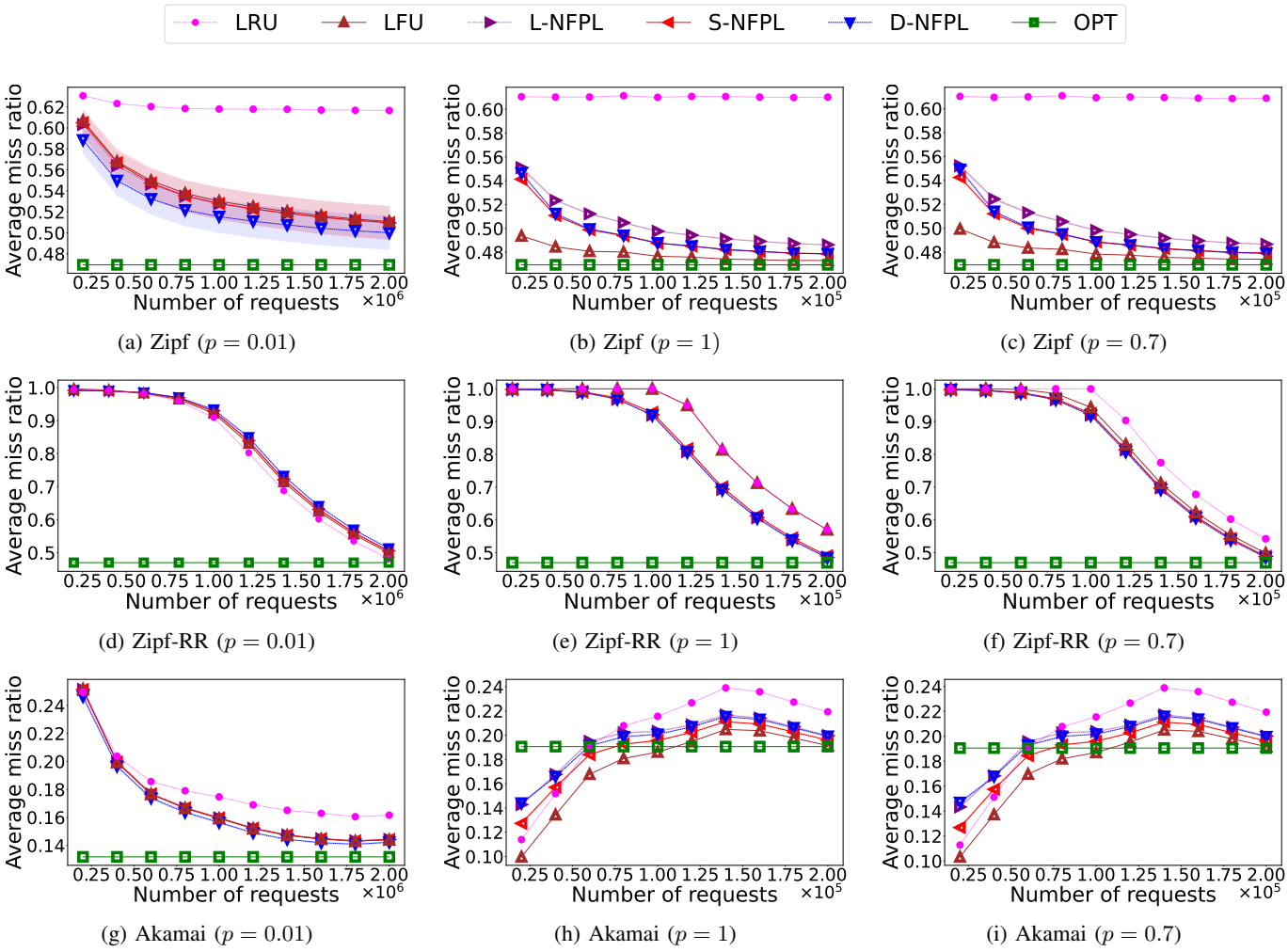
We compare our NFPL policies, with the optimal static cache allocation with hindsight (OPT), and two classic caching policies: Least-Frequently-Used (LFU) and Least-Recently-Used (LRU). Upon a miss, LFU and LRU evict from the cache the least popular file and the least recently requested file, respectively. All these caching policies operate under the assumption of the partial observability regime, where each request is observed with a probability p. Moreover, the performance metric is the *average miss ratio* computed as follows

$$\frac{1}{T} \sum_{t=1}^{T} \mathbb{1}(f_t \notin \mathcal{S}_{\mathcal{A}}(t)). \tag{7}$$

The average miss ratio is averaged over 50 runs, considering different observed requests in the partial observability regime and the noise vectors $\{\boldsymbol{\gamma}_t\}_t$ in the NFPL policies. To account for the variability across the runs, we report the 95% confidence intervals.

C. NFPL vs. classical policies

In this section, we present the results for the previously discussed traces and policies, focusing on how the average miss ratio evolves as the number of requests increases. The parameter p of the BPO regime takes values in $\{0.01, 0.7, 1.0\}$. The cache capacity C is equal to 100 for all the caching policies. Moreover, the total number of requests considered in each trace is $T = 2 \times 10^5$ when $p \in \{0.7, 1\}$ and $T = 2 \times 10^6$ when $p = 0.01$. The NFPL policies are configured with $q = 1$. The parameter B of L-NFPL and S-NFPL is set to 1, whereas its value in D-NFPL is $B = 10$ when $p = 0.01$ and $B = 100$ when $p \in \{0.7, 1\}$. The parameter η is set to $p\sqrt{BT/2C}$ for all the NFPL variants. The results are presented in Figure 1.



(a) Zipf ($p = 0.01$) (b) Zipf ($p = 1$) (c) Zipf ($p = 0.7$)

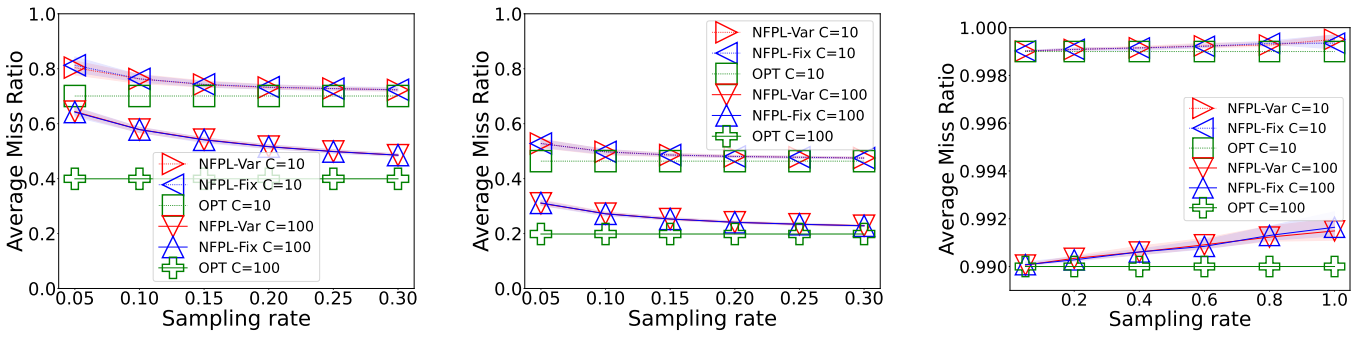
(d) Zipf-RR ($p = 0.01$) (e) Zipf-RR ($p = 1$) (f) Zipf-RR ($p = 0.7$)

(g) Akamai ($p = 0.01$) (h) Akamai ($p = 1$) (i) Akamai ($p = 0.7$)

Fig. 1: Average miss ratio for different values of p across caching policies and traces.

Zipf. D-NFPL outperforms all the other policies when the sampling probability is low ($p = 0.01$). In this scenario, it takes a long time for the request count of each item to become a reliable predictor of future demands, which makes LFU less effective. Additionally, while L-NFPL and S-NFPL do not rely solely on the observed popularity of files, they use a single realization of the noise vector $\boldsymbol{\gamma}$ to explore other potentially relevant files. This results in a lower rate of exploration compared to D-NFPL, where the noise vector is continually regenerated. In the other regimes where $p = 0.7$ and $p = 1.0$, LFU rapidly discerns the most popular files and subsequently converges to OPT, because files popularity does not change over time. However, due to the noise $\boldsymbol{\gamma}_t$, the NFPL policies require a longer duration to determine the files to be stored accurately, especially L-NFPL, because of its lazy update rule. Nevertheless, all the NFPL policies outperform LRU, whose miss ratio fails to converge to OPT.

Zipf-RR. When $p \in \{0.7, 1.0\}$, the NFPL variants achieve a lower average miss ratio than LRU and LFU. This advantage is particularly noticeable when $p = 1$, where all the NFPL variants yield a similar average miss ratio of 0.48, while LRU and LFU also produce similar results of approximately 0.57. Hence, the NFPL variants achieve a relative improvement of roughly 16% over the LRU/LFU group. To justify the superiority of the NFPL policies in this scenario, we divide the trace into two segments. The first segment, where the file ranked $C+1$ is still being requested, and the second one, where requests for this file stop. During the first segment, the specific order of the requests is such that the next request is neither of the C most popular files so far nor the C recently requested items so far, resulting in most of the requests being missed by both LFU and LRU. In contrast, in the second segment, LFU and LRU achieve perfect performance, with every request being a hit since only the top C files are requested. Despite the ideal performance in the second segment, the large number of misses for the popular files in the first segment prevents LRU and LFU from converging to the optimal solution. In contrast, the NFPL variants can handle this challenging trace and successfully converge to the optimum, as predicted by our theoretical results. On the other hand, when $p = 0.01$, the adversarial aspect of the observed trace fades, positioning LRU as the leading policy, outperforming all others. Nonetheless, the NFPL policies are guaranteed to reach the miss ratio of OPT for a large enough number of requests.



(a) Zipf (b) Akamai (c) Round-robin

Fig. 2: Average miss ratio vs. sampling probability.

Akamai: In the real-world trace, when $p = 0.01$, the D−NFPL policy outperforms all other caching policies. Indeed, as discussed in the Zipf trace, when the sampling probability is low, the request count becomes an unreliable predictor of future requests rendering LFU inefficient. Meanwhile, the significant loss of information damages LRU as well. In contrast, for the regimes where $p \in \{0.7, 1\}$, all policies demonstrate comparable performance, with LFU emerging as the most effective option. In particular, compared to the Zipf trace, the performance gap between LRU and OPT is significantly narrower. A possible explanation lies in the anticipated fluctuations in file popularity and the temporal correlations of requests, which can be advantageous to LRU. Additionally, both D−NFPL and L−NFPL perform worse than S−NFPL.

In the supplementary material, we provide additional numerical results in the supplementary material (Section D). We report the variability of the average miss ratio and running time for the experiments discussed in this section. We also compare NFPL with LFU and LRU on a stochastic non-stationary request process.

D. Sampling in NFPL

We evaluate the impact of different sampling methods within S−NFPL on the average miss ratio in the full observation setting, i.e., $p = 1$. Specifically, we compare two sampling approaches: the first, which we call NFPL−Var, independently samples each request with probability q, i.e., $(\beta_t)_{t=1}^T$ are i.i.d. Bernoulli random variables with parameter q. The second approach, which we refer to as NFPL−Fix, is slightly different. NFPL−Fix has a parameter b such that for every batch of B requests, b of them are selected uniformly at random, i.e., $(\beta_t)_{t=1}^T$ are Bernoulli random variables with parameter b/B and $\sum_{t=kB+1}^{(k+1)B} \beta_t = b$, for any k.

We compare the performance of NFPL−Fix, NFPL−Var, and OPT on all the considered traces but replacing Zipf-RR with Round-robin, a trace where all items have the same number of requests. The total number of requests in the Zipf, Akamai, and Round-robin is 5×10^6, 2×10^7, and 10^6, respectively. Two cache sizes are considered in each trace: $C \in \{10, 200\}$ for the Zipf trace and $C \in \{10, 100\}$ for the Akamai and Round-robin trace. Figure 2 illustrates the average miss ratio for all the aforementioned caching policies when varying sampling probabilities, i.e., q for NFPL−Var and b/B for NFPL−Fix.

Across the various traces we analyzed, the performance difference between NFPL−Fix and NFPL−Var is consistently minimal for all the sampling rates. This indicates that the selection of the sampling method may exert only a marginal impact on the performance of NFPL.

For the Zipf and Akamai traces, the performance of both NFPL−Fix and NFPL−Var tends towards that of OPT with increasing sampling rates. This is attributable to the relatively stationary nature of these traces, where the count of past requests serves as a good predictor for future requests; thus, more precise estimates bolster performance. In contrast, the round-robin trace benefits from noisier estimates, as it is preferable to overlook past requests in this scenario. As a result, the performance of NFPL−Fix and NFPL−Var deteriorates with a rising sampling rate.

VII. Conclusion

In this paper, we studied the single cache problem under adversarial requests in the partial observability regime, where each request is observed with a specific probability. We proposed the NFPL caching policy, the first to achieve optimal regret bound with $\mathcal{O}(1)$ amortized time complexity in this more restricted setting. We also proposed extensions of NFPL to bipartite caching. Through experiments on both synthetic and real-world traces, we highlighted the practical importance of our theoretical results. In future work, we plan to derive a tight lower bound on the regret of any caching policy under partial observability of the requests.

Acknowledgments

This research was supported in part by the French Government through the "Plan de Relance" and "Programme d'Investissements d'Avenir".

References

[1] J. Tse and A. J. Smith, "CPU cache prefetching: Timing evaluation of hardware implementations," *IEEE Transactions on Computers*, vol. 47, no. 5, pp. 509–526, 1998.

[2] T. Bektas *et al.*, "Designing cost-effective content distribution networks," *Computers & Operations Research*, vol. 34, no. 8, pp. 2436–2449, 2007.

[3] D. Chatzopoulos *et al.*, "Mobile augmented reality survey: From where we are to where we go," *IEEE Access*, vol. 5, pp. 6917–6950, 2017.

[4] G. Ananthanarayanan *et al.*, "Real-time video analytics: The killer app for edge computing," *Computer*, vol. 50, no. 10, pp. 58–67, 2017.

[5] R. Fagin, "Asymptotic miss ratios over independent references," *Journal of Computer and System Sciences*, vol. 14, no. 2, pp. 222–250, 1977.

[6] S. Traverso *et al.*, "Temporal locality in today's content caching: Why it matters and how to model it," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 5, pp. 5–12, 2013.

[7] M. Leconte *et al.*, "Placing dynamic content in caches with small population," in *IEEE INFOCOM 2016*, pp. 1–9.

[8] G. S. Paschos *et al.*, "Learning to cache with no regrets," in *IEEE INFOCOM 2019*, pp. 235–243.

[9] M. Zinkevich, "Online convex programming and generalized infinitesimal gradient ascent," in *ICML 2003*, pp. 928–936, 2003.

[10] T. Si Salem *et al.*, "No-regret caching via online mirror descent," *ACM TOMPECS*, vol. 8, no. 4, pp. 1–32, 2023.

[11] N. Mhaisen *et al.*, "Online caching with optimistic learning," in *2022 IFIP Networking Conference*, pp. 1–9, 2022.

[12] R. Bhattacharjee *et al.*, "Fundamental limits on the regret of online network-caching," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 4, no. 2, pp. 1–31, 2020.

[13] K. Shanmugam *et al.*, "FemtoCaching: Wireless content delivery through distributed caching helpers," *IEEE Transactions on Information Theory*, vol. 59, no. 12, pp. 8402–8413, 2013.

[14] A. Bura *et al.*, "Learning to cache and caching to learn: Regret analysis of caching algorithms," *IEEE/ACM Transactions on Networking*, vol. 30, no. 1, pp. 18–31, 2022.

[15] Q. Liu and Y. Zhang, "Learning to caching under the partial-feedback regime," in *IEEE CNSM*, pp. 154–162, 2022.

[16] F. Faticanti and G. Neglia, "Optimistic online caching for batched requests," *Computer Networks*, vol. 244, 2024.

[17] A. Kalai and S. Vempala, "Efficient algorithms for online decision problems," *JCSS*, vol. 71, no. 3, 2005.

[18] G. Neglia *et al.*, "Access-time-aware cache algorithms," *ACM TOMPECS*, vol. 2, no. 4, pp. 1–29, 2017.

[19] Y. Ben Mazziane *et al.*, "No-regret caching with noisy request estimates," in *IEEE VCC*, pp. 341–346, 2023.

[20] S. De Rooij *et al.*, "Follow the leader if you can, hedge if you must," *JMLR*, vol. 15, no. 1, pp. 1281–1316, 2014.

[21] S. Mukhopadhyay and A. Sinha, "Online caching with optimal switching regret," in *IEEE ISIT*, pp. 1546–1551, 2021.

[22] J. Abernethy *et al.*, "Online linear optimization via smoothing," in *Conference on learning theory*, pp. 807–823, PMLR, 2014.

[23] A. Cohen and T. Hazan, "Following the perturbed leader for online structured learning," in *ICML 2015*, pp. 1034–1042.

[24] J. D. Abernethy *et al.*, "Fighting bandits with a new kind of smoothness," in *NIPS*, pp. 2197–2205, 2015.

[25] E. Hazan, "Introduction to online convex optimization," *Foundations and Trends® in Optimization*, vol. 2, no. 3-4, pp. 157–325, 2016.

[26] G. Neu and G. Bartók, "Importance weighting without importance weights: An efficient algorithm for combinatorial semi-bandits," *JMLR*, vol. 17, pp. 1–21, 2016.

[27] D. Carra and G. Neglia, "An online gradient-based caching policy with logarithmic complexity and regret guarantees," *arXiv preprint arXiv:2405.01263*, 2024.

[28] N. Mhaisen *et al.*, "Optimistic no-regret algorithms for discrete caching," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 6, no. 3, pp. 1–28, 2021.

[29] N. Mhaisen *et al.*, "Online caching with no regret: Optimistic learning via recommendations," *IEEE Trans. Mob. Comput.*, vol. 23, no. 5, pp. 5949–5965, 2024.

[30] S. Fan *et al.*, "Dynamic regret of randomized online service caching in edge computing," in *IEEE INFOCOM 2023*, pp. 1–10.

[31] G. Quan *et al.*, "Minimizing edge caching service costs through regret-optimal online learning," *IEEE/ACM Transactions on Networking*, vol. 32, no. 5, pp. 4349–4364, 2024.

[32] R. M. Castro *et al.*, "Adaptive selective sampling for online prediction with experts," in *NeurIPS*, 2024.

[33] N. Cesa-Bianchi *et al.*, "Minimizing regret with label efficient prediction," *IEEE Transactions on Information Theory*, vol. 51, no. 6, pp. 2152–2162, 2005.

[34] N. Cesa-Bianchi and G. Lugosi, *Prediction, Learning, and Games*. Cambridge University Press, 2006.

[35] G. Einziger *et al.*, "TinyLFU: A highly efficient cache admission policy," *ACM Transactions on Storage (ToS)*, vol. 13, no. 4, pp. 1–31, 2017.

[36] A. Giovanidis and A. Avranas, "Spatial multi-LRU caching for wireless networks with coverage overlaps," *SIGMETRICS Perform. Eval. Rev.*, pp. 403–405, 2016.

[37] E. Leonardi and G. Neglia, "Implicit coordination of caches in small cell networks under unknown popularity profiles," *IEEE JSAC*, vol. 36, no. 6, pp. 1276–1285, 2018.

[38] S. Ioannidis and E. Yeh, "Adaptive caching networks with optimality guarantees," *SIGMETRICS Perform. Eval. Rev.*, pp. 113–124, 2016.

[39] Y. Li *et al.*, "Online caching networks with adversarial guarantees," *Proc. ACM Meas. Anal. Comput. Syst.*, 2021.

[40] G. S. Paschos *et al.*, "Online convex optimization for caching networks," *IEEE/ACM Transactions on Networking*, vol. 28, pp. 625–638, 2020.

[41] D. Paria and A. Sinha, "`LeadCache`: Regret-optimal caching in networks," in *NeurIPS*, pp. 4435–4447, 2021.

[42] L. Breslau *et al.*, "Web caching and Zipf-like distributions: Evidence and implications," in *IEEE INFOCOM 1999*, pp. 126–134.

VIII. Supplementary Material

A. Online learning with noisy loss functions

1) Linear loss functions: We consider an instance of the online linear learning framework in Section 3-A of the main paper, where at each step t, the agent only observes a random vector $\hat{\mathbf{r}}_t$, which serves as an estimation of the exact cost vector $\mathbf{r}_t$. Note that we will be using here notation from Section 3-A of the main paper. We further assume that these random vectors satisfy Assumption 1.

Assumption 1: The vectors $(\hat{\mathbf{r}}_t)_{t=1}^{T'}$ are bounded and independent such that: $\exists p \in \mathbb{R}^+ \setminus \{0\} : \mathbb{E}[\hat{\mathbf{r}}_t] = p \cdot \mathbf{r}_t$.

We call NFPL, the algorithm that substitutes the exact cost vector by its noisy estimation within the FPL algorithm, i.e.,

$$\hat{\mathbf{x}}_{t+1} = M(\hat{\mathbf{r}}_{1:t} + \boldsymbol{\gamma}'_{t+1}), \tag{8}$$

where $M(\mathbf{u}) \in \arg\min_{\mathbf{x}\in\mathcal{X}} \langle \mathbf{u}, \mathbf{x}\rangle$, $\forall \mathbf{u}$.

Define the random variables $\hat{A}_t = \|\hat{\mathbf{r}}_t\|_1$ and $\hat{R}_t = \sup_{\mathbf{x}\in\mathcal{X}} |\langle \hat{\mathbf{r}}_t, \mathbf{x}\rangle|$.

Lemma 1 is to show that NFPL, with no assumption about the temporal correlations between the noise vectors $(\boldsymbol{\gamma'}_t)_{t=1}^{T'}$, retains the $\mathcal{O}(\sqrt{T'})$ regret guarantees.

Lemma 1: Under Assumption 1:

$$\mathcal{R}_{T'}(\texttt{NFPL}) \leq \frac{1}{p}\left(\frac{1}{\eta}\sup_{\mathbf{r}_1,\dots\mathbf{r}_T}\sum_{t=1}^{T'}(\mathbb{E}\left[\hat{A}_t\hat{R}_t\right]) + \eta D\right).$$

Proof of Lemma 1: . We denote by H the cost of NFPL when the adversary selects the sequence $(\mathbf{r}_t)_{t=1}^{T'}$, i.e., $H = \sum_{t=1}^{T'} \langle \mathbf{r}_t, \hat{\mathbf{x}}_t\rangle$. We also define H^* as the corresponding static optimum, defined as $H^* = \langle \mathbf{r}_{1:T'}, M(\mathbf{r}_{1:T'})\rangle$.

A key idea of the proof is to consider a surrogate online linear learning problem with the same decision set as the original one, namely $\mathcal{X}$, and cost set $\hat{\mathcal{B}}$ containing all possible values of $(\hat{\mathbf{r}}_t)_{t=1}^{T'}$. Let G be $1/p$ times the cost of the FPL algorithm in this surrogate problem; formally, $G = 1/p \sum_{t=1}^{T'} \langle \hat{\mathbf{r}}_t, \hat{\mathbf{x}}_t\rangle$. Additionally, let G^* denote the corresponding static optimum, given by $G^* = 1/p \langle \hat{\mathbf{r}}_{1:T'}, M(\hat{\mathbf{r}}_{1:T'})\rangle$. We decompose $H - H^*$ in three terms:

$$H - H^* = (H - G) + (G - G^*) + (G^* - H^*). \tag{9}$$

In what follows, we prove that 1) $\mathbb{E}[G - G^*]$ is smaller than or equal to the right hand side in the lemma's statement, 2) $\mathbb{E}[H - G] = 0$, and 3) $\mathbb{E}[G^* - H^*] \leq 0$. Combining the three inequalities allow us to conclude the proof.

We note that a direct application of [17, Theorem 1.1 a)] allow us to conclude that

$$\mathbb{E}[G - G^*] \leq \frac{1}{p}\left(\frac{\hat{A}\hat{R}T}{\eta} + \eta D\right), \tag{10}$$

where $\hat{A} = \sup_{\hat{\mathbf{r}}\in\hat{\mathcal{B}}}\|\hat{\mathbf{r}}_t\|_1$ and $\hat{R} = \sup_{\hat{\mathbf{r}}\in\hat{\mathcal{B}},\mathbf{x}\in\mathcal{X}} |\langle \hat{\mathbf{r}}, \mathbf{x}\rangle|$. This is sufficient to conclude regret sublinearity by choosing $\eta \propto \sqrt{T}$, but with a suboptimal dependency on p. The inequality in Lemma 1 leads to a tighter $1/\sqrt{p}$ bound (see Corollary 1).

In order to obtain the tighter inequality in the lemma, we adapt the proof of [17, Theorem 1.1 a)] as follows. We observe that the distributions over $\hat{\boldsymbol{r}}_{1:t-1} + \boldsymbol{\gamma}'_t$ and $\hat{\boldsymbol{r}}_{1:t} + \boldsymbol{\gamma}'_t$ are similar. In particular, they are both distributions over cubes. If the cubes overlap on a fraction f of their volume, then

$$\mathbb{E}_{\boldsymbol{\gamma}'_t}\left[\langle M(\hat{\boldsymbol{r}}_{1:t-1} + \boldsymbol{\gamma}'_t), \hat{\boldsymbol{r}}_t\rangle\right] \leq \mathbb{E}_{\boldsymbol{\gamma}'_t}\left[\langle M(\hat{\boldsymbol{r}}_{1:t} + \boldsymbol{\gamma}'_t), \hat{\boldsymbol{r}}_t\rangle\right] + (1-f)\hat{R}_t.$$

This is because on the fraction that they overlap, the expectation is identical, and on the fraction that they do not overlap, one can only be $\hat{R}_t$ larger, by the definition of $\hat{R}_t$. [17, Lemma 3.2] allows us to conclude that

$$1 - f \leq \frac{1}{\eta}\|\hat{\boldsymbol{r}}_t\|_1 \leq \frac{1}{\eta}\hat{A}_t,$$

and then

$$\mathbb{E}_{\boldsymbol{\gamma}'_t}\left[\langle M(\hat{\boldsymbol{r}}_{1:t-1} + \boldsymbol{\gamma}'_t), \hat{\boldsymbol{r}}_t\rangle\right] \leq \mathbb{E}_{\boldsymbol{\gamma}'_t}\left[\langle M(\hat{\boldsymbol{r}}_{1:t} + \boldsymbol{\gamma}'_t), \hat{\boldsymbol{r}}_t\rangle\right] + \frac{1}{\eta}\hat{A}_t\hat{R}_t.$$

The rest of the proof follows as in [17, Theorem 1.1 a)] taking an additional expectation over the noisy cost vectors and leads to

$$\mathbb{E}[G - G^*] \leq \frac{1}{p}\left(\frac{1}{\eta}\sup_{\mathbf{r}_1,\dots\mathbf{r}_T}\sum_{t=1}^{T'}(\mathbb{E}\left[\hat{A}_t\hat{R}_t\right]) + \eta D\right). \tag{11}$$

We conclude that $\mathbb{E}[G - H] = 0$, by observing that $\hat{\mathbf{x}}_t$ depends only on $(\hat{\mathbf{r}}_s)_{s=1}^{t-1}$ and hence it is independent from $\hat{\mathbf{r}}_t$, thanks to Assumption 1. Therefore,

$$\mathbb{E}[G] = {}^1\!/_p \sum_{t=1}^{T'} \langle \mathbb{E}[\hat{\mathbf{r}}_t], \mathbb{E}[\hat{\mathbf{x}}_t] \rangle = \mathbb{E}[H] \tag{12}$$

Finally, $\mathbb{E}[G^* - H^*] \leq 0$ because

$$\mathbb{E}[G^*] = {}^1\!/_p \mathbb{E}[\langle \hat{\mathbf{r}}_{1:T'}, M(\hat{\mathbf{r}}_{1:T'}) \rangle] \tag{13}$$
$$\leq {}^1\!/_p \mathbb{E}[\langle \hat{\mathbf{r}}_{1:T'}, M(\mathbf{r}_{1:T'}) \rangle] \tag{14}$$
$$= \langle \mathbf{r}_{1:T'}, M(\mathbf{r}_{1:T'}) \rangle = H^* \tag{15}$$

■

In the particular case where $\hat{\mathbf{r}}_t = \delta_t \mathbf{r}_t$ with $(\delta_t)_t$ i.i.d. Bernoulli random variables with parameter p, Corollary 1 shows that the regret bound of NFPL is sublinear in T' and scales as $\frac{1}{\sqrt{p}}$.

Corollary 1: If $\hat{r}_t = \delta_t r_t$ and $\eta = \sqrt{\frac{pRAT'}{D}}$, then

$$\mathcal{R}_{T'}(\texttt{NFPL}) \leq 2\frac{1}{\sqrt{p}}\sqrt{RADT'}. \tag{16}$$

Proof of Corollary 1: In this case $\hat{R}_t \leq \delta_t R$ and $\hat{A}_t \leq \delta_t A$. Then $\mathbb{E}\left[\hat{A}_t \hat{R}_t\right] \leq \mathbb{E}\left[\delta_t^2\right] AR = pAR$ and the result follows from Lemma 1 and the choice of η. ■

2) General loss functions: In the analysis above, we considered a natural extension of FPL to the case where the loss functions are linear ($\langle \mathbf{r}_t, \cdot \rangle$), and the learner has access to noisy estimations of these loss functions ($\hat{\mathbf{r}}_t$) statisfying Assumption 1. Now we consider a different online learning problem with arbitrary loss functions $l_t(\cdot)$ where the learner has only access to $\delta_t l_t(\cdot)$, where $(\delta_t)_t$ are the i.i.d. Bernoulli random variables with parameter p. Note that in comparison to Assumption 1, the linear loss functions do not need to be linear but at the same time, they have a particular structure, namely $\delta_t l_t(\cdot)$.

Let $\mathcal{A}$ be a no-regret algorithm for the setting where the exact loss functions are observed by the learner. It is further assumed that this algorithm is parametrized by η, and the regret guarantees are of the form,

$$\mathcal{R}_{T'}(\mathcal{A}) \leq \frac{ET'}{\eta} + \eta F, \tag{17}$$

where F and E are constants related to the online learning problem.

We call $\hat{\mathcal{A}}$ the direct adaptation of $\mathcal{A}$ to the noisy loss functions setting, where the decision vector $\hat{\mathbf{x}}_t$ is updated according to $\mathcal{A}$ whenever $\delta_t = 1$, and otherwise it is equal to its previous value. Similarly to Lemma 1, Lemma 2 also provides no-regret guarantees for $\hat{\mathcal{A}}$. Interestingly, in this case we again obtain a $1/\sqrt{p}$ dependency on the observation probability, although this dependency follows from a different argument. Lemma 2 is particularly useful for analyzing the general bipartite caching problem in Section 5.

Lemma 2: If the regret of $\mathcal{A}$ satisfies (17) and $\eta = \sqrt{\frac{pET'}{F}}$, then

$$\mathcal{R}_{T'}(\hat{\mathcal{A}}) \leq 2\frac{1}{\sqrt{p}}\sqrt{EFT'}. \tag{18}$$

Proof of Lemma 2: We define the quantities introduced in Lemma 1 for general loss functions, namely, G, G^*, H, and H^*, as follows,

$$H = \sum_{t=1}^{T'} l_t(\hat{\mathbf{x}}_t),\ H^* = \min_{\mathbf{x}\in\mathbf{X}} \sum_{t=1}^{T'} l_t(\mathbf{x}) \tag{19}$$

$$G = \frac{1}{p}\sum_{t=1}^{T'} \delta_t l_t(\hat{\mathbf{x}}_t),\ G^* = \frac{1}{p}\min_{\mathbf{x}\in\mathbf{X}} \sum_{t=1}^{T'} \delta_t l_t(\mathbf{x}) \tag{20}$$

The objective is to bound $H - H^*$, by decomposing it in the sum of the three terms: $H - G$, $G - G^*$, and $G^* - H^*$.

It is easy to show that $\mathbb{E}[G] = \mathbb{E}[H]$ because $\hat{\mathbf{x}}_t$ is independent of δ_t. Let $\mathbf{x}^*$ be the minimizer of H, then

$$\mathbb{E}[G^*] \leq \frac{1}{p}\mathbb{E}\left[\sum_{t=1}^{T'} \delta_t l_t(\mathbf{x}^*)\right] = H^*. \tag{21}$$

Note that by definition, pG is the cost of $\mathcal{A}$ on the losses $(l_t)_{t\in\mathcal{T}}$ where $\mathcal{T} = \{t \in [T'] : \delta_t = 1\}$. Thus, conditioned on the realizations of $(\delta_t)_{t=1}^{T'}$, $p(G - G^*)$ is bounded by the regret bound for $\mathcal{A}$ for a sequence of length $|\mathcal{T}| = \sum_{t=1}^{T'} \delta_t$. Thanks to (17), we obtain that,

$$G - G^* \leq \frac{1}{p}\left(\frac{E|\mathcal{T}|}{\eta} + F\eta\right). \tag{22}$$

Taking expectation over the randomness of $(\delta_t)_{t=1}^{T'}$, we obtain that,

$$\mathbb{E}\left[G - G^*\right] \leq \frac{ET'}{\eta} + \frac{\eta F}{p}. \tag{23}$$

Finally, the best choice of η is $\sqrt{pET'/F}$, which yields the target result.

■

B. Proof of Thm. 2

The expected number of misses of any randomized caching policy $\mathcal{A}$ is given by $\sum_{t=1}^{T} \Pr\left(f_t \notin S_{\mathcal{A}}(t)\right)$. Note that for any NFPL instance, it can be shown that $\boldsymbol{\gamma}_t$ is uniformly distributed in $[0,\eta]^N$ for all t, making the probability distribution of the set of stored items in the cache the same for any NFPL instance. Therefore, the expected number of misses remains the same, independent of the temporal correlations between these noise vectors.

For both S-NFPL and D-NFPL, it is evident that $\boldsymbol{\gamma}_t$ is uniformly distributed in $[0,\eta]^N$ for all t. In the case of L-NFPL, the noise vector is inspired by the Follow-the-Lazy-Leader (FLL) algorithm from [17]. The noise vector in this case can be rewritten as:

$$\boldsymbol{\gamma}_t = \eta\left(\frac{\boldsymbol{\gamma}_0 - \hat{\mathbf{n}}(t)}{\eta} - \left\lfloor\frac{\boldsymbol{\gamma}_0 - \hat{\mathbf{n}}(t)}{\eta}\right\rfloor\right). \tag{24}$$

Note that the random variable $\mathbf{u} = \frac{\boldsymbol{\gamma}_0 - \hat{\mathbf{n}}(t)}{\eta}$ is uniform within a range of width 1 in each component. Therefore, $\mathbf{u} - \lfloor\mathbf{u}\rfloor$ is a uniform random vector in $[0,1]^N$, and finally, $\boldsymbol{\gamma}_t$ is uniform in $[0,\eta]^N$.

We first establish the regret bound for NFPL when $pq = 1$, before addressing the general scenario with arbitrary probabilities p and q.

Particular Case: $pq = 1$**.**

Here, NFPL corresponds to FPL from [17], tailored for the caching problem in the full observation regime. This connection becomes clear when we cast the caching problem within the online linear learning framework outlined in Section 3-A, identifying the decision set $\mathcal{X}$ and the cost set $\mathcal{B}$ as introduced in that section. Finally, applying [12, Thm. 1.1], we establish the regret bound for NFPL.

Since NFPL updates the cache state after every B requests, we model the caching problem as an online linear one, with a total number of rounds $T' = \lceil T/B\rceil$, Let $t \in [T']$.

Cache state. The set of cached items at any step t', that satisfies $tB \leq t' \leq (t+1)B - 1$, is represented by the binary vector $\mathbf{x}_t = [x_{t,i}]_{i\in[N]}$, which indicates the files missing in the cache; that is, $x_{t,i} = 1$ if and only if file i is not stored in the cache at time t, i.e., $i \notin S_{\mathcal{A}}(t')$. A feasible cache allocation is then represented by a vector in the set:

$$\mathcal{X} = \left\{\mathbf{x} \in \{0,1\}^N \,\middle|\, \sum_{i=1}^{N} x_i = N - C\right\}. \tag{25}$$

Requests. The request process is represented as a sequence of vectors $\mathbf{r}_t = (r_{t,i} \in \mathbb{N} : i \in [N])\ \forall t$, where $r_{t,f}$ is the number of requests received within the range $[(t-1)B + 1, tB]$, i.e., $r_{t,f} = \sum_{s=(t-1)B+1}^{tB} \mathbb{1}(f_s = f)$. Then, each of these vectors belongs to the set:

$$\mathcal{B} = \left\{\mathbf{r} \in \mathbb{N}^N \,\middle|\, \sum_{i=1}^{N} r_i = B\right\}. \tag{26}$$

Cost. At each time slot t, the cache pays a cost equal to the number of misses, i.e., to the number of requests for files not in the cache. The cost can be computed as follows:

$$\langle \mathbf{r}_t, \mathbf{x}_t\rangle = \sum_{i=1}^{N} r_{t,i}x_{t,i}. \tag{27}$$

The decision vector of FPL with uniform noise [17] in an online linear learning problem is as follows,

$$\mathbf{x}_{t+1}(\texttt{FPL}) \in \arg\min_{\mathbf{x}\in\mathcal{X}} \langle \mathbf{x}, \sum_{s=1}^{t} \mathbf{r}_s + \boldsymbol{\gamma}_{t+1} \rangle, \tag{28}$$

where $(\boldsymbol{\gamma}_t)_t$ are random vectors of N i.i.d. uniform random variables, each within the range $[0, \eta]$. Let D be the diameter of the set $\mathcal{X}$, i.e., $D = \sup_{\mathbf{x},\mathbf{y}\in\mathcal{X}} \|\mathbf{x} - \mathbf{y}\|_1$, A be a bound on the norm 1 of vectors in $\mathcal{B}$, and R a bound on $\langle \mathbf{x}, \mathbf{r} \rangle$ for any $(\mathbf{x}, \mathbf{r}) \in \mathcal{X} \times \mathcal{B}$.

The regret bound of FPL over T' rounds is upper bounded by $RAT'/\eta + D\eta$ [17, Thm. 1.1].

For the specific online learning problem with $\mathcal{X}$ in (25) and $\mathcal{B}$ in (26), the vector $\sum_{s=1}^{t} \mathbf{r}_s$ represents the number of requests for all files at step tB. Moreover, the constants associated to the sets $\mathcal{X}$ and $\mathcal{B}$ are given by: $R = A = B$ and $D = 2C$. Therefore, FPL in this setting stores the C files with the largest $n_f(tB) + \gamma_{tB,f}$ for $f \in \mathcal{I}$, which coincides with the update rule of NFPL, when $\eta = \sqrt{BT/2C}$, with T the total number of requests. As a result, applying [17, Thm. 1.1] in this setting yields the following regret bound for NFPL:

$$B^2 T' \sqrt{\frac{2C}{BT}} + 2C\sqrt{\frac{BT}{2C}} \tag{29}$$

$$= \sqrt{2CB}\left(\frac{BT'}{\sqrt{T}} + \sqrt{T}\right) \tag{30}$$

$$\leq \sqrt{2CB}\left(B\left(\frac{T}{B} + 1\right)\frac{1}{\sqrt{T}} + \sqrt{T}\right) \tag{31}$$

$$= \sqrt{2CB}\left(2\sqrt{T} + \frac{B}{\sqrt{T}}\right). \tag{32}$$

General Case: Arbitrary p and q.

Note that to apply [17, Thm. 1.1 and Lem. 1.2], FPL relies on the cumulative cost vector $\mathbf{r}_{1:t}$, which represents in the caching problem the exact request count for each file. Unfortunately, NFPL with $pq < 1$ leverages approximate counts represented via the vector $\hat{\mathbf{n}}$ and therefore we can not use the results from [17]. For this reason, we consider an instance of the online linear learning framework in Section 3-A, where at each step t, the agent only observes a random vector $\hat{\mathbf{r}}_t$.

To recover the update rule of the caching policy NFPL, consider the general NFPL applied for the online linear learning problem with $\mathcal{X}$ in (25), $\mathcal{B}$ in (26), and $\hat{\mathbf{r}}_t = (\hat{r}_{t,f})_{f\in\mathcal{I}}$ set to

$$\hat{\mathbf{r}}_t = \sum_{s=(t-1)B+1}^{tB} \delta_s \beta_s \mathbb{1}(f_s = f). \tag{33}$$

It follows that $\mathbb{E}[\hat{\mathbf{r}}_t] = pq\mathbf{r}_t$. Moreover, at each step t, $R_t = A_t = \sum_{s=(t-1)B+1}^{tB} \delta_s\beta_s$, and thus $\mathbb{E}[R_t A_t] = Bpq + B(B-1)(pq)^2$. Thanks to Lemma 1 and with $\eta = \sqrt{\frac{pq+(pq)^2(B-1)T}{2C}}$, we obtain the desired regret bound following similar steps as in (29)-(32). This finishes the proof of Theorem 2.

C. Proof of Thm. 3

Regarding the implementations of the three instances of NFPL: S-NFPL, D-NFPL, and L-NFPL, we examine for each first the case where $B = 1$ and $pq = 1$, then the general case. The most computationally demanding operation in the algorithm is identifying the top C files in terms of perturbed counts, namely, $\hat{\mathbf{n}} + \boldsymbol{\gamma}$.

D-NFPL. At each time step, the noise vectors $(\boldsymbol{\gamma}_t)_{t\in[T]}$ are regenerated. Consequently, all entries of $\hat{\mathbf{n}} + \boldsymbol{\gamma}$ are updated, necessitating a sorting operation at every time step to find the top C files. This leads to an amortized time complexity of $\mathcal{O}(N \log N)$. When $B > 1$, sorting occurs only when t is a multiple of B, reducing the time complexity by a factor of B.

S-NFPL. The noise vectors remain constant over time, so only the component corresponding to the requested item in $\hat{\mathbf{n}} + \boldsymbol{\gamma}$ is updated at each time step. This allows us to utilize a *heap* data structure to efficiently keep track of the top C files. Specifically, at each step t, if $f_t \in S_t$, the corresponding count in the heap is incremented by 1, while maintaining the heap structure with a time complexity of $\mathcal{O}(\ln C)$. If f_t is not in S_t, the file with the smallest perturbed count is retrieved from the heap in $\mathcal{O}(1)$ time. This file's count is then compared to $\hat{n}_{f_t}(t) + \gamma_{t,f_t}$. If the perturbed count of f_t exceeds that of the smallest file in the heap, the smallest file is removed, and f_t is inserted into the heap with time complexity of $\mathcal{O}(\ln C)$. This operation maintains the heap structure and ensures that the heap always contains the top C files based on their perturbed counts. It becomes clear now that the amortized time complexity of S-NFPL is $\mathcal{O}(\ln C)$. If $pq < 1$, updates are made to the heap data structure whenever a request is taken into account, i.e., $\delta_t\beta_t = 1$, which happens with probability pq, justifying the $\mathcal{O}(pq \ln C)$ amortized time

complexity. Whether or not B is greater than 1, the heap updates must occur whenever $\delta_t\beta_t = 1$, which explains the exclusion of B from the time complexity.

L-NFPL. Define the vector $\mathbf{m}_t$ as follows, $\mathbf{m}_t = \hat{\mathbf{n}}_t + \boldsymbol{\gamma}_t$. Given the expression of the noise vector in L-NFPL, $\gamma_{t,f} - \gamma_{t-1,f} > 0$ if and only if $f = f_t$. Therefore, the vector $\mathbf{m}$ has the property that at most one of its components is updated at each step, which allows to employ a heap to keep track of the top C files and to incur a time complexity of $\mathcal{O}(\ln C)$ per insertion/deletion, similarly to S-NFPL. However, in L-NFPL, it is possible to have $\mathbf{m}_t = \mathbf{m}_{t-1}$ when $\gamma_{t,f_t} = \gamma_{t-1,f_t} - 1$. Therefore, whenever $\gamma_{t,f_t} = \gamma_{t-1,f_t} - 1$, the counts of all items remain the same, alleviating the need for updating the heap data structure, and the time complexity in these steps is $\mathcal{O}(1)$. The amortized time complexity is then proportional to:

$$\frac{\ln C}{T} \sum_{t=1}^{T} \Pr\left(\gamma_{t,f_t} \neq \gamma_{t-1,f_t} - 1\right). \tag{34}$$

Next, we prove that $\Pr\left(\gamma_{t,f_t} \neq \gamma_{t-1,f_t} - 1\right) \leq \frac{pq}{\eta}$, justifying the amortized time complexity of L-NFPL since $\eta = \sqrt{BT/2C}$. Define Δ_t as $\left\lceil \frac{\hat{n}_{f_t}(t) - \gamma_{0,f_t}}{\eta} \right\rceil - \left\lceil \frac{\hat{n}_{f_t}(t-1) - \gamma_{0,f_t}}{\eta} \right\rceil$.

$$\Pr\left(\gamma_{t,f_t} - \gamma_{t-1,f_t} + 1 \neq 0\right) = \Pr\left(\eta\Delta_t \neq 0\right) \tag{35}$$

$$= \Pr\left(\Delta_t \geq 1\right), \tag{36}$$

where the last step follows because Δ_t is a positive integer.

To compute $\Pr\left(\Delta_t \geq 1\right)$, observe that if $\delta_t\beta_t = 0$, $\hat{n}_{f_t}(t) = \hat{n}_{f_t}(t-1)$ and hence $\Delta_t = 0$, which enables us to write:

$$\begin{aligned}\Pr\left(\Delta_t \geq 1\right) &= \Pr\left(\Delta_t \geq 1 | \delta_t\beta_t = 1\right) pq + \Pr\left(\Delta_t \geq 1 | \delta_t\beta_t = 0\right)(1 - pq) \\ &= \Pr\left(\Delta_t \geq 1 | \delta_t\beta_t = 1\right) pq. \end{aligned} \tag{37}$$

If $\eta \leq 1$, the statement $\Pr\left(\Delta_t \geq 1\right) \leq pq/\eta$ becomes evident thanks to (37). We assume that $\eta > 1$ and $\delta_t\beta_t = 1$. Define the following quantities:

$$z = \frac{\hat{n}_{f_t}(t) - \gamma_{0,f_t}}{\eta},\ k = \lceil z \rceil. \tag{38}$$

A direct consequence of $\delta_t\beta_t = 1$ is $\hat{n}_{f_t}(t) = \hat{n}_{f_t}(t-1) + 1$, we can then write:

$$\begin{aligned}&\Delta_t = k - \left\lceil z - \frac{1}{\eta} \right\rceil \leq z + 1 - z + \frac{1}{\eta} = 1 + \frac{1}{\eta} < 2 \\ &\implies \Delta_t \leq 1. \end{aligned} \tag{39}$$

Moreover,

$$\begin{aligned}\Delta_t = 1 &\iff k = \left\lceil z - \frac{1}{\eta} \right\rceil + 1 \\ &\iff k - 2 < z - \frac{1}{\eta} \leq k - 1 \\ &\iff -2 + \frac{1}{\eta} < z - k \leq -1 + \frac{1}{\eta}. \end{aligned} \tag{40}$$

Combining (37),(39), and (40), we obtain:

$$\Pr\left(\Delta_t \geq 1\right) = \Pr\left(\Delta_t \geq 1 | \delta_t\beta_t = 1\right) pq \tag{41}$$

$$= \Pr\left(\Delta_t = 1 | \delta_t\beta_t = 1\right) pq \tag{42}$$

$$= \Pr\left(-2 + \frac{1}{\eta} < z - k \leq -1 + \frac{1}{\eta}\right) pq \tag{43}$$

$$= \Pr\left(-1 < z - k \leq -1 + \frac{1}{\eta}\right) pq \tag{44}$$

$$\leq \frac{pq}{\eta}. \tag{45}$$

where we used the fact that $z - k$ is uniform over $[-1, 0]$ for any value of n_{f_t} to write (44) and (45). The correctness of the statement above is due to z being uniformly distributed over an interval of width equal to 1. Indeed, since γ_{0,f_t} is uniform over $[0, \eta]$, it follows that z is uniform over $[\frac{n_{f_t}}{\eta} - 1, \frac{n_{f_t}}{\eta}]$.

Similarly to S-NFPL, even if t is a multiple of $B > 1$, the update of the heap is necessary whenever $\delta_t\beta_t = 1$, explaining the absence of the factor $1/B$ from the amortized time complexity. This completes the proof.

TABLE III: Performance metrics for NFPL, LFU, and LRU policies with the Zipf trace ($p \in \{1, 0.7, 0.01\}$).

	$p = 1$			$p = 0.7$			$p = 0.01$		
Algorithm	Avg. miss	Variance	Time (s)	Avg. miss	Variance	Time (s)	Avg. miss	Variance	Time (s)
L-NFPL	0.49	1.27×10^{-4}	3.77×10^{-1}	0.49	1.9×10^{-4}	2.85×10^{-1}	0.51	1.56×10^{-2}	2.81×10^{-1}
S-NFPL	0.48	1.36×10^{-4}	4.26×10^{-1}	0.48	2.23×10^{-4}	3.11×10^{-1}	0.51	1.56×10^{-2}	2.54×10^{-1}
D-NFPL	0.48	6.17×10^{-5}	5.48	0.48	1.33×10^{-4}	1.2	**0.5**	$\mathbf{1.58 \times 10^{-4}}$	1.03
LFU	**0.47**	1.11×10^{-6}	$\mathbf{3.5 \times 10^{-1}}$	**0.47**	$\mathbf{7.32 \times 10^{-6}}$	$\mathbf{2.6 \times 10^{-1}}$	0.51	1.54×10^{-2}	$\mathbf{2.46 \times 10^{-1}}$
LRU	0.61	$\mathbf{0.49 \times 10^{-7}}$	9.82×10^{-1}	0.61	2.86×10^{-4}	7.2×10^{-1}	0.62	1.11×10^{-2}	2.78×10^{-1}

TABLE IV: Performance metrics for NFPL, LFU, and LRU policies with the Zipf-RR trace ($p \in \{1, 0.7, 0.01\}$).

	$p = 1$			$p = 0.7$			$p = 0.01$		
Algorithm	Avg. miss	Variance	Time (s)	Avg. miss	Variance	Time (s)	Avg. miss	Variance	Time (s)
L-NFPL	**0.48**	1.7×10^{-4}	3.69×10^{-1}	0.49	2.32×10^{-4}	2.77×10^{-1}	0.51	6.32×10^{-4}	2.71×10^{-1}
S-NFPL	0.49	1.12×10^{-4}	4.23×10^{-1}	0.49	$\mathbf{1.26 \times 10^{-4}}$	3.13×10^{-1}	0.50	6.36×10^{-4}	2.47×10^{-1}
D-NFPL	**0.48**	1.09×10^{-4}	3.11	**0.48**	1.27×10^{-4}	1.22	0.51	7.98×10^{-4}	1.04
LFU	0.57	$\mathbf{1.07 \times 10^{-6}}$	$\mathbf{3.63 \times 10^{-1}}$	0.50	1.48×10^{-4}	$\mathbf{2.6 \times 10^{-1}}$	0.50	4.64×10^{-4}	$\mathbf{2.38 \times 10^{-1}}$
LRU	0.57	1.11×10^{-6}	9.15×10^{-1}	0.54	2.26×10^{-4}	6.64×10^{-1}	**0.48**	$\mathbf{1.69 \times 10^{-4}}$	2.67×10^{-1}

D. Additional Experimental Results

In this section, we discuss more detailed results on the comparison among the instances of NFPL, LFU, and LRU in terms of average miss ratio, variance of miss ratio, and execution time, for each trace and different sampling probabilities. We report the results in Tables III, IV and V for the Zipf, Zipf-RR and Akamai traces, respectively.

Zipf. We can observe in Table III that, among all the NFPL instances, L-NFPL achieves the lowest execution time (except for $p = 0.01$ where it is slightly outperformed by S-NFPL), and D-NFPL presents the lowest variance, in line with the discussion in Section 4-D of the main paper. In such a stationary scenario, LFU performs optimally with the smallest average miss ratio.

Zipf-RR. As already discussed in Section 6 of the main paper, with the Zipf-RR traces there is a significant difference concerning the Zipf, especially with $p = 1$ (Table IV), where the best average miss ratio is achieved by D-NFPL and L-NFPL. When $p = 1$, D-NFPL presents the smallest variance and largest execution time among the instances of NFPL. Such an execution time is due to the sorting performed by D-NFPL at each step. Similar to the Zipf, L-NFPL has the best execution time among all the NFPL policies except for the case when the sampling probability is low. In such a case, S-NFPL achieves the lowest execution time given that having a low sampling probability is almost equivalent to having a smaller number of requests.

Akamai. With the real trace of Akamai (Table V), all the compared solutions are very close in terms of average miss ratio. Surprisingly, in this scenario, we can observe an inverted trend concerning the two cases before: the average miss ratio decreases as the probability of sampling decreases as well. NFPL does not show the best average miss ratio since, with $N = 1000$ and $C = 100$, the algorithm has not reached the asymptotic regime yet.

We consider a non-stationary stochastic request model, denoted **Zipf–Swap**. The request process of length T is partitioned into s consecutive segments of roughly equal length (the last segment absorbs any remainder). In each segment, requests are drawn i.i.d. from a Zipf distribution with skew parameter α, but the rank of items changes across segments. Formally, let $\mathcal{I} = [N]$ and let $\mathbf{p}^{(1)} = (p_i^{(1)})_{i \in [N]}$ be the Zipf probability vector in the first segment, where items are ranked in decreasing popularity, i.e., $p_i \sim 1/i^{\alpha}$. For segment $k \in \{2, \ldots, s\}$ we define the probability vector of segment k, $\mathbf{p}^{(k)}$, as the cyclic rotation of $\mathbf{p}^{(1)}$ by $(k-1)\Delta$ ranks. For example, in the second segment, with $\Delta = 1$ the rank of item $i \in \{1, \ldots, N-1\}$ becomes $i+1$, and the rank of item N becomes 1. The parameters s and Δ respectively control the frequency and the magnitude of the popularity shifts. The special case $s = 1$ recovers the stationary Zipf process.

Using configuration as in Section 6-C of the main paper, we evaluate the average miss ratio over time of LRU, LFU, and L-NFPL on a **Zipf–Swap** trace with $N = 10^4$, $s = 200$, $\alpha = 1.0$, and $\Delta \in \{1, 5\}$. For observation probabilities $p \in \{0.7, 1\}$, we set the horizon to $T = 2 \times 10^5$, while for $p = 0.01$ we use $T = 2 \times 10^6$. Figure 3 reports the average miss ratio over time.

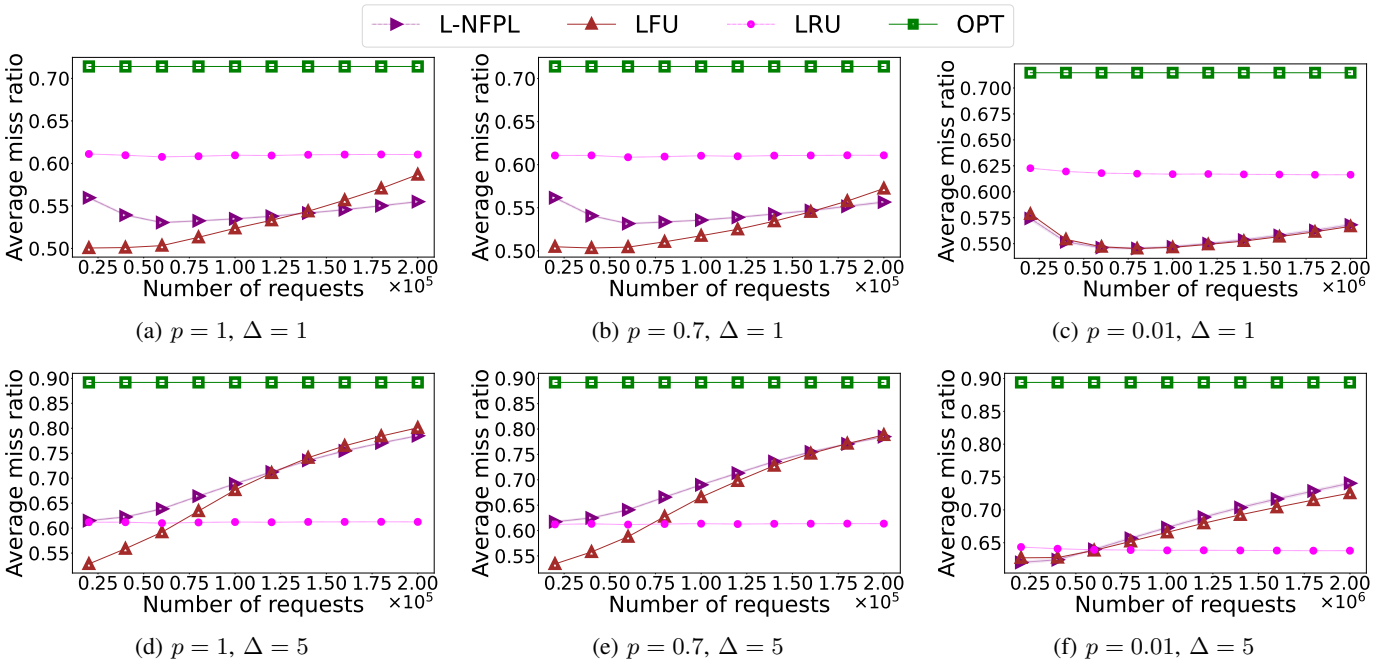
When $\Delta = 1$, L-NFPL has the best miss ratio, followed by LFU and then LRU. In this scenario, the popularity shift is small enough that popularity-based policies are performing well. Moreover, the noise vectors of L-NFPL and its lazy update rule help forget items that were popular in the past in comparison to LFU. On the other hand, when $\Delta = 5$, LRU has the best miss ratio, and LFU and L-NFPL have similar performance. In this case, the magnitude of the popularity shift is significant enough to favor reactive policies such as LRU.

E. NFPL for bipartite caching

Researchers have proposed heuristics obtained by adapting well-known single-cache policies [36], [37]. The naive adaptation treats every cache independently: cache j reacts to the subsequence of requests $\{f_t : j \in \mathcal{M}_t\}$. [36] showed how to extend LRU in two ways:

TABLE V: Performance metrics for NFPL, LFU, and LRU policies with the Akamai trace ($p \in \{1, 0.7, 0.01\}$).

Algorithm	$p = 1$			$p = 0.7$			$p = 0.01$		
	Avg. miss	Variance	Time (s)	Avg. miss	Variance	Time (s)	Avg. miss	Variance	Time (s)
L-NFPL	0.2	2.08×10^{-4}	3.77×10^{-1}	0.2	2.1×10^{-4}	2.82×10^{-1}	0.144	3.1×10^{-4}	4.98×10^{-1}
S-NFPL	**0.19**	1.7×10^{-4}	2.65×10^{-1}	0.2	1.86×10^{-4}	2.04×10^{-1}	0.144	3.1×10^{-4}	4.63×10^{-1}
D-NFPL	0.2	7.82×10^{-5}	2.61×10^{-1}	0.2	1.02×10^{-4}	1.99×10^{-1}	**0.142**	3.5×10^{-4}	5.58×10^{-1}
LFU	**0.19**	4.52×10^{-5}	$\mathbf{2.2 \times 10^{-1}}$	**0.19**	1.16×10^{-4}	$\mathbf{1.54 \times 10^{-1}}$	0.144	3×10^{-4}	$\mathbf{4.56 \times 10^{-1}}$
LRU	0.22	$\mathbf{3.35 \times 10^{-6}}$	1.17	0.22	$\mathbf{8.96 \times 10^{-5}}$	8.37×10^{-1}	0.16	$\mathbf{2.14 \times 10^{-4}}$	5.64×10^{-1}



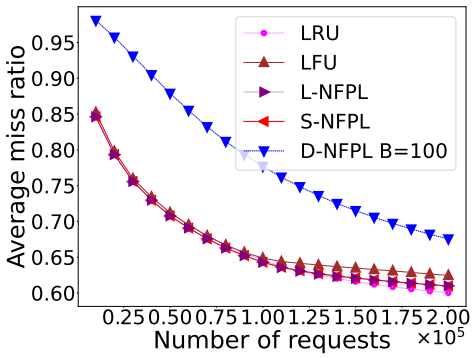
(a) $p = 1$, $\Delta = 1$ (b) $p = 0.7$, $\Delta = 1$ (c) $p = 0.01$, $\Delta = 1$

(d) $p = 1$, $\Delta = 5$ (e) $p = 0.7$, $\Delta = 5$ (f) $p = 0.01$, $\Delta = 5$

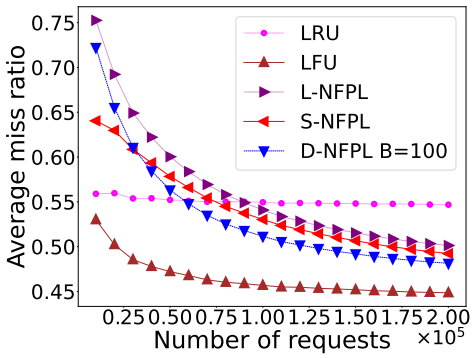
Fig. 3: Average miss ratio over time on the Zipf–Swap trace ($N = 10^4$, $s = 200$), averaged over 50 runs.

- LRU-One: On a hit, a cache that already stores f_t—chosen uniformly at random or by a prescribed rule—moves the key of f_t to the front of its LRU list. On a miss, a cache selected from $\mathcal{M}_t$ according to a similar criterion, downloads f_t, inserts it at the front, and evicts its least-recently-used item.
- LRU-All: On a miss, all caches download f_t, place it at the front, and evict their least-recently-used items. On a hit, all the caches that already store f_t in $\mathcal{M}_t$ move its key to the front, and all other caches in $\mathcal{M}_t$ remain unchanged.

We extend NFPL, LFU, and LRU to operate in a bipartite caching context (see Section 5) in the same way described for LRU. In the One instance, only one cache is updated following a miss or a hit. Conversely, in the All instance, all caches in $\mathcal{M}_t$ update their content. All caches update their state according to the specific single-cache policy. Figures 4 and 5 illustrate the behavior of the One and All instances, respectively, as the success probability p varies. We simulate a network of $J = 6$ caches with $(\mathcal{M}_t)_t$ i.i.d. random subsets of caches of size 2 ($d = 2$), where each cache has capacity $C = 100$, and the catalog size is set to 10^4 under a stationary Zipf distribution with $\alpha = 1.0$. Both figures reveal a similar trend, more prominently shown in the All instance, to the single-cache case. Indeed, in the case of partial observation with small probability p, the noise regeneration of NFPL policies leads to better performance compared to LFU. Additionally, we can notice that the All instances achieve a smaller average miss ratio compared to their One counterparts. For large values of the probability p, LFU outperforms the other policies because of the stationarity of the request process.

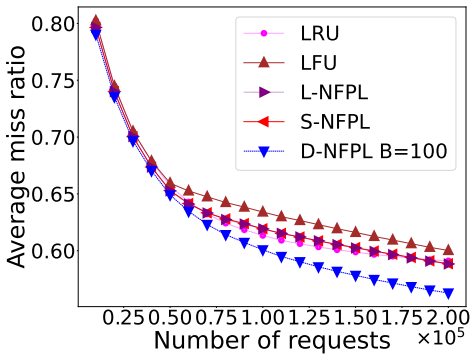


(a) Zipf ($p = 0.01$)

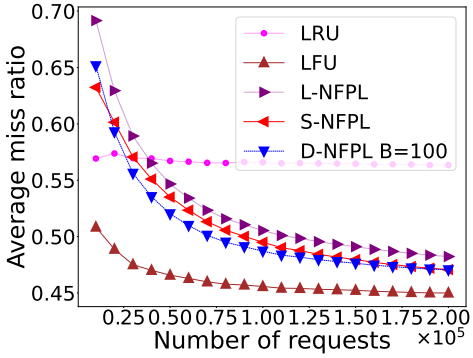


(b) Zipf ($p = 1$)

Fig. 4: `Multi-One` instances for caching algorithms on a network of caches with $J = 6$ and $d = 2$.



(a) Zipf ($p = 0.01$)



(b) Zipf ($p = 1$)

Fig. 5: `Multi-All` instances for caching algorithms on a network of caches with $J = 6$ and $d = 2$.