Online Caching with no Regret
Optimistic Learning via Recommendations

Mhaisen, Naram; Iosifidis, George; Leith, Douglas

DOI
10.1109/TMC.2023.3317943

Publication date
2023

Document Version
Final published version

Published in
IEEE Transactions on Mobile Computing

Online Caching With no Regret: Optimistic Learning via Recommendations

Naram Mhaisen, George Iosifidis, and Douglas Leith

***Abstract*—The design of effective online caching policies is an increasingly important problem for content distribution networks, online social networks and edge computing services, among other areas. This paper proposes a new algorithmic toolbox for tackling this problem through the lens of *optimistic* online learning. We build upon the Follow-the-Regularized-Leader (FTRL) framework, which is developed further here to include predictions for the file requests, and we design online caching algorithms for bipartite networks with pre-reserved or dynamic storage subject to time-average budget constraints. The predictions are provided by a content recommendation system that influences the users viewing activity and hence can naturally reduce the caching network's uncertainty about future requests. We also extend the framework to learn and utilize the best request predictor in cases where many are available. We prove that the proposed optimistic learning caching policies can achieve *sub-zero* performance loss (regret) for perfect predictions, and maintain the sub-linear regret bound $O(\sqrt{T})$, which is the best achievable bound for policies that do not use predictions, even for arbitrary-bad predictions. The performance of the proposed algorithms is evaluated with detailed trace-driven numerical tests.**

***Index Terms*—Edge caching, network optimization, online learning, regret analysis.**

I. Introduction

A. Motivation and Background

THE quest for efficient data caching policies spans more than 50 years and remains today one of the most important research areas for wireless and wired communication systems [2]. Caching was first studied in computer systems where the aim was to decide which files to store in fast-accessible memory segments (*paging*) [3]. Its scope was later expanded due to the explosion of Internet web traffic [4] and the advent of content distribution networks (CDNs) [5], and was recently revisited as a technique to improve the operation of wireless networks through edge caches [6] and on-device caching [7]. A common challenge in these systems is to design an online policy that decides which files to store at a cache, without knowing the future file requests, so as to maximize the cache *hits* or some other cache-related performance metric.

Manuscript received 14 November 2022; revised 8 September 2023; accepted 8 September 2023. Date of publication 22 September 2023; date of current version 4 April 2024. This work was supported by European Commission under Grant 101017109 (DAEMON). An earlier version of this work was presented at the IFIP Networking Conference (IFIP Networking), 2022 [DOI: 10.23919/IFIP-Networking55013.2022.9829806]. Recommended for acceptance by Q. Zhao. *(Corresponding author: Naram Mhaisen.)*

Naram Mhaisen and George Iosifidis are with the Department of Software Technology, Delft University of Technology, 2628 Delft, The Netherlands (e-mail: n.mhaisen@tudelft.nl; g.iosifidis@tudelft.nl).

Douglas Leith is with the School of Computer Science and Statistics, Trinity College Dublin, D02 PN40 Dublin, The Ireland (e-mail: doug.leith@tcd.ie).

Digital Object Identifier 10.1109/TMC.2023.3317943

There is a range of online caching policies that tackle this problem under different assumptions on the request arrivals. Policies such as the LFU and LRU are widely-deployed, yet their performance deteriorates when the file popularity is non-stationary, i.e., the requests are drawn from a time-varying probability distribution [8], [9], [10]. This motivated modeling non-stationary request patterns [11], [12] and optimizing accordingly the caching decisions [13], [14]. Another line of work relies on learning techniques such as (multi-agent) reinforcement learning to estimate the request probabilities and make caching decisions accordingly [15], [16], [17]; but typically these solutions either do not offer optimality bounds, or do not scale due to having the library size in their bounds.

Caching was studied within the framework of online learning in [18] for a single-cache system; and in its more general form recently in [19] that proposed an online gradient descent (OGD) caching policy. Interesting follow-up works include sub-modular policies [20], online mirror-descent policies [21], and the characterization of their performance limits [22], [23]. The advantage of these online learning-based caching policies is that they are scalable, do not require training data, and their performance bounds are *robust* to any possible request pattern, even when the requests are generated by an adversary that aims to degrade the caching operation. On the other hand, forecasting models have been used to optimize caching e.g., evicting the file with the furthest predicted request [24]. While these policies have shown performance gains, their hit ratio can deteriorate if the forecasted requests cease to meet reality due to, e.g., a shift in the training data (users request). Follow-up works attempted to remedy this issue by designing mechanisms that identify performance deterioration and trigger re-training of the model [25]. Still, algorithms with explicit formal guarantees that are agnostic to the quality of the predictions are yet to be proposed.

Hence, an aspect that remains hitherto unexplored is whether predictions about future requests can improve the performance of such learning-based caching policies *without sacrificing their robustness*. This is important in modern caching systems where often the users receive content viewing recommendations from a recommendation system (*RecSys*). For instance, recommendations are a standard feature in streaming platforms such as YouTube and Netflix [26]; but also in social network platforms such as Facebook and Twitter, which moderate the users' viewing feeds [27]. Not surprisingly, the interplay between recommendations and caching attracted recent attention and prior works aimed to increase the caching hits or reduce routing costs, by either recommending already-cached files to users, or through the joint optimization of caching and recommendation decisions [28], [29], [30], [31], [32], [33]. These important works, however, consider static caching models and require knowing in advance the users' expected requests and their propensity to follow the recommendations, or make certain assumptions on the structure of the loss function, e.g., strongly-convex.

Changing vantage point, one can observe that since recommendations bias the users towards viewing certain contents, they can effectively serve as predictions of the forthcoming requests. This prediction information, if properly employed, can hugely improve the efficacy of *dynamic* caching policies. Nevertheless, the caching policy needs to adapt to the accuracy of recommendations (i.e., of the predictions) and the users' propensity to follow them – which is typically unknown and potentially time-varying. Otherwise, the caching performance might as well deteriorate by following these misleading *hints* about future requests. The goal of this work is to tackle exactly this challenging new problem and answer the question: *Can we leverage untrusted predictions in caching systems?* We answer this question in the affirmative by *proposing online learning-based caching policies that utilize predictions (of unknown quality) to boost performance, if those predictions are accurate, while still maintaining robust performance bounds otherwise*.

B. Methodology and Contributions

Our approach is based on the theory of Online Convex Optimization (OCO) that was introduced in [34] and has since been applied in several decision problems [35]. The basic premise of OCO is that a learner (here the caching system) selects in each slot t a decision vector $\boldsymbol{x}_t$ from a convex set $\mathcal{X}$, without knowing the t-slot convex performance function $f_t(\boldsymbol{x})$, that changes with time. The learner's goal is to minimize the growth rate of *regret* $R_T = \sum_{t=1}^{T} f_t(\boldsymbol{x}^\star) - f_t(\boldsymbol{x}_t)$, where $\boldsymbol{x}^\star = \arg\max_{\boldsymbol{x}\in\mathcal{X}} \sum_{t=1}^{T} f_t(x)$ is the benchmark solution designed with hindsight, i.e., with access to the entire sequence of future functions $\{f_t\}_{t=1}^{T}$. The online caching problem fits squarely in this setup, where $f_t(\boldsymbol{x})$ depends on the users' requests and is unknown when the caching is decided. And previous works [19], [20], [21], [22] have proved that OCO-based caching policies achieve $R_T = O(\sqrt{T})$, thus ensuring asymptotically zero average regret: $\lim_{T\to\infty} R_T/T = 0$.

Different from these important studies, we extend the learning model to include predictions that are available through the content recommendations. Improving the regret of learning policies via predictions is a relatively new area in machine learning research. For instance, [36] focuses on the *competitive-ratio* metric and developed algorithms that use untrusted predictions while maintaining worst-case performance bounds; while [37] applied similar ideas to the paging problem. However, it was shown in [38] that such competitive-ratio algorithms cannot ensure sublinear regret, which is the performance criterion we employ here, in line with all recent works [19], [20], [21], [22], [23].

For regret-minimization with predictions, [39] used predictions for the function gradient $\nabla f_t(\boldsymbol{x}_t)$ with guaranteed quality to reduce R_T from $O(\sqrt{T})$ to $O(\log T)$; and [40] enhanced this result by allowing some predictions to fail the quality condition. A different line of works uses *regularizing functions* which enable the learner to adapt to the predictions' quality [41], [42]. This idea is more promising for the caching problem where the recommendations might be inaccurate, or followed by the users for only arbitrary time windows; thus, we used it as starting point to develop our caching learning frameworks.

In specific, our approach relies on the Follow-The-Regularized-Leader (FTRL) algorithm [43] which we extend with predictions that offer *optimism* by reducing the uncertainty about the next-slot functions. We study different versions of the caching problem. First, we design a policy (OFTRL) for the bipartite caching model [6], which generalizes the standard single cache case [18], [37]. In fact, the bipartite model represents a wide range of caching systems including CDNs, Edge caching and Femtocaching scenarios, D2D caching networks, and so on. Theorem 1 proves that R_T is proportional to prediction errors ($\|\boldsymbol{c}_t - \widetilde{\boldsymbol{c}}_t\|^2$), diminishing to zero for perfect predictions; while still meeting the best achievable bound $O(\sqrt{T})$ for the regular OCO setup (i.e., without using predictions) [22] even if all predictions fail. We continue with the *elastic* caching problem, where the system resizes the used caches at each slot based, e.g., on volatile storage leasing costs [44], [45], [46]. The aim is to maximize the caching utility subject to a time-average budget constraint for the storage capacity costs. This places the problem in the realm of constrained-OCO [47], [48], [49], [50]. Using a new saddle point analysis with predictions, we prove Theorem 2 which reveals how the regret and the budget violation depend on the cache sizes and prediction errors, and how one can prioritize one metric over the other while achieving sublinear growth rates for both.

The above algorithms utilize the RecSys as the only source to predict the next time-slot cost function gradient $\widetilde{\boldsymbol{c}}_{t+1}$. In many cases, however, content providers might have access to *multiple* such sources. For example, a statistical user profiling model, a deep learning-based predictive model for content requests [51], or even another RecSys, see [52] and follow-up works. Those sources can be used to obtain multiple, and possibly contradicting, predictions. Our final contribution is, therefore, a *meta-learning* caching framework that utilizes predictions from multiple sources to achieve the same performance as a caching system that used the best such source to start with. We show that the regret in the case of multiple sources can be strictly negative depending on the request sequence and the existence of a high-accuracy predictor. At the same time, the meta-learning caching framework maintains sublinear regret when all predictors fail. Finally, we show that this framework can also be applied in cases with single RecSys, and discuss its pros and cons compared to the proposed regularization-based optimistic caching solutions.

In summary, the contributions of this work can be grouped as follows:

• Introduces an online learning framework for bipartite and elastic caching networks that leverages predictions to achieve a regret that is upper-bounded by *zero* for perfect recommendations and remains sub-linear $O(\sqrt{T})$ for arbitrary bad recommendations. The results are based on a new analysis technique that improves the bounds by a factor of $\sqrt{2}$ compared to the state-of-art optimistic-regret bounds [42], which we used in our recent work [1].

• Introduces a meta-learning framework that utilizes predictions from *multiple* sources and learns which of them to use, if any at all. This framework achieves regret similar to the best predictor-based system, and maintains sub-linear regret if all predictors fail.

• Evaluates the policies using various request models and real datasets [53], [54] and compares them with *(i)* the best in hindsight benchmark; and *(ii)* the online gradient descent policy, which is known to achieve the best possible regret bound (without using predictions) and outperforms other policies [19], [22].

The work presents *conceptual innovations*, i.e., using recommendations as an untrusted prediction source for caching decisions, and leveraging different online caching algorithms in an optimistic meta-learning algorithm; as well as *technical contributions* such as the tightened bound of optimistic proximal FTRL (Theorem 1) and the new optimistic proximal FTRL algorithm with budget constraints (Theorem 2). While we focus on data caching, the proposed algorithms can be directly applied to caching of services and code libraries in edge computing systems.

Paper Organization: The rest of this work is organized as follows. Section II introduces the system model and states formally the problem. Section III presents the optimistic online caching policy for the bipartite graph, and Section IV presents the respective policy and results for the case of elastic caching systems. Section V introduces the meta-learning framework with the inclusion of multiple predictors, and Section VI presents our numerical evaluation of the proposed algorithms using synthetic and real traces. We conclude in Section VII.

II. System Model and Problem Statement

A. Model Preliminaries

Notation: We use calligraphic capital letters, e.g., $\mathcal{X}$ to denote sets. Vectors are denoted with bold-face small letters,[1] e.g., $\boldsymbol{a}$, and we use the subscript t to highlight a vector's dependence on a specific time slot e.g., $\boldsymbol{a}_t$. Scalars are denoted with regular letters and can as well depend on the time, e.g., h_t. When a component of the vector is indexed, the subscript is repurposed to denote that component's (multi-)index, while the t moves to the superscript. i.e., for the d-dimensional vector $\boldsymbol{a}_t$ we write $\boldsymbol{a}_t = (a_1^t, a_2^t, \ldots, a_d^t)$. We denote with $\{\boldsymbol{a}_t\}_{t=1}^T$ the sequence of vectors or parameters from slot $t = 1$ up to slot T; whenever the horizon is not relevant we use $\{\boldsymbol{a}_t\}_t$. We also use the shorthand sum notation for scalars $b_{1:t} = \sum_{i=1}^T b_i$ and the element-wise sum of vectors $\boldsymbol{a}_{1:t} = \sum_{i=1}^T \boldsymbol{a}_i$. We denote with $[T]$ the integer set $1, 2, \ldots, T$. We make use of the indicator function $\boldsymbol{I}_{\mathcal{X}}(\boldsymbol{x})$, which evaluates to $\boldsymbol{I}_{\mathcal{X}}(\boldsymbol{x}) = 0$ if $x \in \mathcal{X}$ and ∞ otherwise. The notation is summarized in Table I.

[1]The only exception is the vector $\boldsymbol{F}_t$, which was done for better readability of Section V

TABLE I
Key Notation

Parameters	Physical Meaning
	Caching Network
$\mathcal{J}(J)$	Set (number) of caches
$\mathcal{I}(I)$	Set (number) of user locations
C_j	Capacity of cache j. $C_j \le C, \forall j \in \mathcal{J}$
ℓ_{ij}	Indicator for connectivity of location i to cache j
N	Number of files
q_{ni}^t	Request issued by user i for file n at slot t
w_{nij}	Utility for routing a unit of file n from cache j to i
$f_t(\cdot)$	The utility function at slot t
$\boldsymbol{c}_t$	Gradient of the utility function
$\tilde{\boldsymbol{c}}_t$	A prediction for $\boldsymbol{c}_t$
s_j^t	Price for unit storage in cache j
P	Number of predictors
	Decision Variables
y_{nj}^t	Portion of file n stored at cache j at slot t
z_{nij}^t	Portion of file n routed from cache j to i at slot t
$\boldsymbol{x}$	A shorthand for the concatenated variables (y, z)
	Learning Algorithms
$r_t(\cdot)$	A strongly convex regularizer function (for $t \ge 1$)
$v_t(\cdot)$	The regularized cost function (negative utility): $-f_t(\cdot) + r_t(\cdot)$
h_t	Prediction error $\Vert\tilde{c}_t - c_t\Vert$
σ_t	The change in the aggregated root of prediction error $\sqrt{h_{1:t}} - \sqrt{h_{1:t-1}}$
$\boldsymbol{F}_t$	Experts performance vector at slot t
$\boldsymbol{u}_t$	Weight vector used to combine experts' proposals

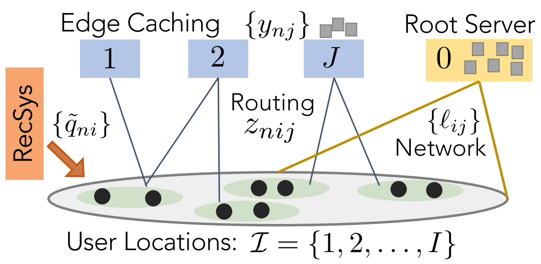

Fig. 1. System Model. A network of J caches serves file requests from a set of I users locations. Unserved requests are routed to the Root Server. Caching decisions are aided via the recommendations provided by the RecSys.

Network: The caching network includes a set of edge caches $\mathcal{J} = \{1, 2, \ldots, J\}$ and a root cache indexed with 0, as shown in Fig. 1. The file requests emanate from a set of user locations $\mathcal{I} = \{1, 2, \ldots, I\}$. The connectivity between $\mathcal{I}$ and $\mathcal{J}$ is modeled with parameters $\boldsymbol{\ell} = (\ell_{ij} \in \{0, 1\} : i \in \mathcal{I}, j \in \mathcal{J})$, where $\ell_{ij} = 1$ if cache j can be reached from location i. We consider the general case where the caches have overlapping coverage; thus, each user can be (potentially) served by one or more edge caches. The root cache is within the range of all users in $\mathcal{I}$. This is a general non-capacitated bipartite model, see [55] for an overview of caching models; and extends the celebrated femtocaching model [6] since the link qualities (which are captured through the utility gains; see below) not only may vary with time, but can do so in an arbitrary (i.e., non-stationary) fashion. This latter feature is particularly important for the realistic modeling of volatile wireless edge caching systems [56], [57].

Requests: The system operation is time slotted, $t = 1, 2, \ldots, T$. Users submit requests for obtaining files from a library $\mathcal{N}$ of N files with unit size; we note that the analysis can be readily extended to files with different sizes. This will be made clear in Section II-B. Parameter $q_{ni}^t \in \{0,1\}$ indicates the submission of a request for file $n \in \mathcal{N}$ by a user at location $i \in \mathcal{I}$ in the beginning of slot t. At each slot we assume there is one request;[2] i.e., the caching decisions are updated after every request, as in LFU and LRU policies, [58], [59]. Hence, the request process comprises successive vectors $\boldsymbol{q}_t = (q_{ni}^t \in \{0,1\} : n \in \mathcal{N}, i \in \mathcal{I})$ from the set:

$$\mathcal{Q} = \left\{ \boldsymbol{q} \in \{0,1\}^{N \cdot I} \,\middle|\, \sum_{n\in\mathcal{N}} \sum_{i\in\mathcal{I}} q_{ni} = 1 \right\}.$$

We make no assumptions for the request pattern; it might follow a fixed or time-varying distribution that is unknown to the system; and can be even selected strategically by an *adversary* aiming to degrade the caching operation. If a policy's performance is satisfactory under this model, it is ensured to achieve (at least) the same performance for other request models.

Recommendations: There is a recommender system (*RecSys*) that suggests files to each user $i \in \mathcal{I}$, see [26] for the case of Netflix. User i requests one of the recommended files with a certain probability that captures the user's propensity to follow the recommendations. Unlike prior works that consider these probabilities fixed [28], [60], we model them as unknown and possibly time-varying. Namely, no assumption on their quality is guaranteed to remain valid.

A key point in our approach is that the content recommendations, if properly leveraged, can serve as *predictions* for the next-slot requests which are otherwise unknown. We denote with $\widetilde{\boldsymbol{q}}_t$ the prediction for the request $\boldsymbol{q}_t$ that the system will receive at the beginning of slot t, and we assume that $\widetilde{\boldsymbol{q}}_t$ is available at the end of slot $t-1$, i.e., when the RecSys provides its recommendations. Essentially, the recommender system serves as an indirect mechanism for predicting the next request, and this can happen in various ways. For example, the caching system can set $\widetilde{q}_{\hat{n}\hat{i}}^{t+1} = 1$ and $\widetilde{q}_{ni}^{t+1} = 0, \forall (n,i) \neq (\hat{n},\hat{i})$, where $(\hat{n},\hat{i})$ is the request with the highest predicted probability (top recommended file).[3] In Section V, we study the case where a set $\mathcal{P} = \{1, \ldots, P\}$ of P different predictors (other than the recommendation system) are available, each one offering a prediction $\boldsymbol{q}_t^{(p)}, p \in \mathcal{P}$ at every slot t.

Caching: Each cache $j \in \mathcal{J}$ stores up to $C_j << N$ files, while the root cache stores the entire library, i.e., $C_0 \geq N$. We also define $C = \max_{j\in\mathcal{J}} C_j$. Following the femtocaching model [6], we perform caching using the *Maximum Distance Separable* (MDS) codes, where files are split into a fixed number of F chunks, including redundancy chunks. A user can decode the file if it receives any F-sized subset of its chunks. For large values of F, the MDS model allows us to use continuous caching variables.[4] Hence, we define the variable $y_{nj}^t \in [0,1]$ which denotes the portion of F chunks of file $n \in \mathcal{N}$ stored at cache $j \in \mathcal{J}$, and we introduce the t-slot caching vector $\boldsymbol{y}_t = (y_{nj}^t : n \in \mathcal{N}, j \in \mathcal{J})$ that belongs to set:

$$\mathcal{Y} = \left\{ \boldsymbol{y} \in [0,1]^{N \cdot J} \,\middle|\, \sum_{n\in\mathcal{N}} y_{nj} \leq C_j,\, j \in \mathcal{J} \right\}.$$

We note that our model can be readily extended to files of different size, by replacing the capacity constraint in $\mathcal{Y}$ with

$$\sum_{n\in\mathcal{N}} v_n y_{nj} \leq C_j,\ j \in \mathcal{J}$$

For some general size vector $v \in \mathbb{R}_+^N$. Such a change will not affect the mathematical characteristics of the optimization problem (both sets correspond to linear constraints).

Routing: Since each user location $i \in \mathcal{I}$ may be connected to multiple caches, we need to introduce routing variables. Let z_{nij}^t denote the portion of request q_{ni}^t served by cache j. In the MDS caching model the requests can be simultaneously served from multiple caches and, naturally, we restrict[5] the amount of chunks not to exceed F. Hence, the t-slot routing vector $\boldsymbol{z}_t = (z_{nij}^t \in [0,1] : n \in \mathcal{N}, i \in \mathcal{I}, j \in \mathcal{J})$ is drawn from:

$$\mathcal{Z} = \left\{ \boldsymbol{z} \in [0,1]^{N \cdot J \cdot I} \,\middle|\, \sum_{j\in\mathcal{J}} z_{nij} \leq 1,\, n \in \mathcal{N}, i \in \mathcal{I} \right\}.$$

Requests that are not (fully) served by the edge caches $\mathcal{J}$ are served by the root server that provides the missing chunks. This decision needs not to be explicitly modeled as it is directly determined by the routing vector $\boldsymbol{z}_t$.

B. Problem Statement

Cache Utility & Predictions: We use parameters $w_{nij} \in [0, w]$ to model the system utility when delivering a chunk of file $n \in \mathcal{N}$ to location $i \in \mathcal{I}$ from cache $j \in \mathcal{J}$, instead of using the root server. This general utility model offers the ability to capture bandwidth savings or delay reductions, as well as other common objectives of edge-caching, in both wired and wireless networks. It is noteworthy that under this model, the caching benefits may vary for each cache and user location, and they can also change over time. Additionally, the problem of maximizing cache hits can be viewed as a specific instance within this context (when $w_{nij} = 1$), as discussed in [2]. To streamline presentation we

[2]The proposed policies will still deliver the same regret guarantees when requests are batched before an update. However, the Lipchitz constant will be scaled according to the batch size, and this will affect accordingly the constant factor of the guarantees.

[3]Note that our caching policy is orthogonal to the mechanism that maps the recommendations to predictions. Namely, our results will be stated in terms of the prediction error.

[4]Large files are composed of thousands of chunks, leading to a small chunk size (compared to the original file). This induces practically negligible errors in the utility function [55, Section 3.3]. In addition, even for exact discrete caching, relaxing the integrality constraints and solving the continuous version is an essential first step which is then followed by a randomized rounding technique (see, e.g., [61, Sec. 6]).

[5]This practical constraint is called the *inelastic* model and compounds the problem, cf. [22] for the simpler elastic model.

introduce vector $\boldsymbol{x}_t = (\boldsymbol{y}_t, \boldsymbol{z}_t) \in \mathbb{R}^m$, with $m = NIJ + NJ$, and define the system utility in slot t as:

$$f_t(\boldsymbol{x}_t) = \sum_{n\in\mathcal{N}} \sum_{i\in\mathcal{I}} \sum_{j\in\mathcal{J}} w_{nij} q_{ni}^t z_{nij}^t \qquad (1)$$

and we denote its gradient $\boldsymbol{c}_t = \nabla f_t(x_t)$. As it will become clear, our analysis holds also for non-linear concave functions $f_t(x)$; this generalization is useful in case, e.g., we wish to enforce fairness in the dispersion of caching gains across user locations [45].

The main challenge in online caching is the following: at the end of each slot t where we need to decide the cache configuration, the utility function f_{t+1} is not available. Indeed, this function depends on the next-slot request $\boldsymbol{q}_{t+1}$, which is made known only after $\boldsymbol{y}_{t+1}$ has been decided and fixed,[6] see [19], [22], [37]. Note that this is also the operation timing of the LRU/LFU policies [58], [59]. However, the recommendations provided to users can be used to form the vector of predicted requests $\widetilde{\boldsymbol{q}}_{t+1}$, which in turn can be used to create a prediction for the gradient of the next slot function $\widetilde{f}_{t+1}(\cdot)$. This predicted gradient, denoted $\widetilde{\boldsymbol{c}}_{t+1}$, suffices for designing the proposed optimistic algorithms.

Benchmark: In such learning problems, it is important to understand the objective that our algorithm aims to achieve. If we had access to an oracle for all requests over a horizon of T slots $\{\boldsymbol{q}_t\}_{t=1}^T$ (and the utility parameters) we could have devised the utility-maximizing static caching and routing policy $\boldsymbol{x}^\star = (\boldsymbol{y}^\star, \boldsymbol{z}^\star)$, by solving the following convex optimization problem:

$$\mathbb{P}_1: \quad \max_{\boldsymbol{x}}. \quad \sum_{t=1}^{T} f_t(\boldsymbol{x}) \qquad (2)$$

$$\text{s.t.} \quad z_{nij} \le y_{nj}\ell_{ij}, \quad i\in\mathcal{I}, j\in\mathcal{J}, n\in\mathcal{N}, \qquad (3)$$

$$\boldsymbol{z}\in\mathcal{Z}, \quad \boldsymbol{y}\in\mathcal{Y}, \qquad (4)$$

where (3) ensure that the routing decisions for each requested file use only caches that store enough chunks of that file. Let us define the convex set of constraints:

$$\mathcal{X} = \{\{\mathcal{Y}\times\mathcal{Z}\}\cap\{(3)\}\} \qquad (5)$$

that we will use henceforth to streamline presentation.

Clearly, this hypothetical solution $\boldsymbol{x}^\star$ can be designed only with *hindsight* and is the benchmark for evaluating our online learning policy π which outputs $\{\boldsymbol{x}_t\}_t$. Thus, in line with prior works, we use the metric of static regret:

$$R_T(\pi) = \sup_{\{f_t\}_{t=1}^T} \left[\sum_{t=1}^{T} f_t(\boldsymbol{x}^\star) - \sum_{t=1}^{T} f_t(\boldsymbol{x}_t) \right], \qquad (6)$$

which quantifies the performance gap of π from $\boldsymbol{x}^\star$, for any possible sequence of requests or, equivalently, functions $\{f_t\}_t$. Our goal is to find a policy that achieves sublinear regret, $R_T(\pi) = o(T)$, thus ensuring the average performance gap R_T/T will diminish as T grows. This policy, similar to other online policies, decides $\boldsymbol{x}_{t+1}$ at the end of each slot t using the previous utility functions $\{f_\tau\}_{\tau=1}^t$ and the next-slot prediction $\widetilde{f}_{t+1}$ devised from the RecSys.

Note that, in principle, the regret metric can be negative. This is especially true for optimistic policies. To see why, recall that $\boldsymbol{x}^\star$ is the best *fixed* caching configuration, whereas $\boldsymbol{x}_t$ is allowed to change for each t. Hence, it might happen, e.g., that $\boldsymbol{x}_t$ performs better than $\boldsymbol{x}^\star$ on some steps, $f_t(\boldsymbol{x}_t) \ge f_t(\boldsymbol{x}^\star)$, while performing similar to $\boldsymbol{x}^\star$ in the remaining ones $f_t(\boldsymbol{x}_t) \approx f_t(\boldsymbol{x}^\star)$. Of course, the occurrence of such an event depends on the request sequence and the policy that determines $\boldsymbol{x}_t$.

On the other hand, there are stricter benchmarks such as those that allow the hindsight policy to pick a different decision in each slot, $\boldsymbol{x}_t^\star \in \arg\max_{\boldsymbol{x}\in\mathcal{X}} f_t(\boldsymbol{x})$. These benchmarks give rise to the dynamic regret metric,[7] which compares the learning algorithm with $\sum_{t=1}^T f_t(\boldsymbol{x}_t^\star)$. However, it is well established that achieving sublinear dynamic regret requires additional assumptions on the variability of the function $f_t(\cdot)$ (see, e.g., the discussion in the introduction section of [62]). In addition, for the caching problem in particular, such a dynamic policy, (i.e., $\{\boldsymbol{x}_t^\star\}_{t=1}^T$), will alter the cache state *at every time-step*, inducing prohibitive switching cost. This is in contrast to the presented policies that *converge* to the best fixed solution. In the following sections, we show that optimism can greatly decrease the upper bound on R_T, increasing the chances of negative regret.

III. Optimistic Bipartite Caching

Unlike recent caching solutions that rely on Online Gradient Descent (OGD) [19] or on the Follow-the-Perturbed-Leader (FTPL) policy [22], our approach draws from the *Follow-The-Regularized-Leader* (FTRL) policy, cf. [63], appended with prediction-adaptive regularizers. A key element in our proposal is the *optimism* emanating from the availability of predictions, namely the content recommendations that are offered to users by the RecSys in each slot.

Let us begin by defining the proximal regularizers:[8]

$$r_0(\boldsymbol{x}) = \boldsymbol{I}_{\mathcal{X}}(\boldsymbol{x}), \quad r_t(\boldsymbol{x}) = \frac{\sigma_t}{2}\|\boldsymbol{x}-\boldsymbol{x}_t\|^2, \ t\ge 1 \qquad (7)$$

where $\|\cdot\|$ is the euclidean norm, and recall that $\boldsymbol{I}_{\mathcal{X}}(\boldsymbol{x}) = 0$ if $x\in\mathcal{X}$ and ∞ otherwise. We apply properly selected regularizing parameters, which change the strong convexity of r_t according to the predictions' quality until t, and also ensure $r_t(\boldsymbol{x}) \ge 0, \forall t$, namely:

$$\sigma_1 = \sigma\sqrt{h_1}, \quad \sigma_t = \sigma\left(\sqrt{h_{1:t}} - \sqrt{h_{1:t-1}}\right), \quad t\ge 2$$

$$\text{with} \quad h_t = \|\boldsymbol{c}_t - \widetilde{\boldsymbol{c}}_t\|^2, \qquad (8)$$

where $\sigma \ge 0$, $\boldsymbol{c}_t = \nabla f_t(\boldsymbol{x}_t)$, and we used the shorthand sum notation $h_{1:t} = \sum_{i=1}^t h_i$ for the aggregate prediction errors during the first t slots. The basic step of the algorithm is:

$$\boldsymbol{x}_{t+1} = \arg\min_{\boldsymbol{x}\in\mathbb{R}^m} \left\{ r_{0:t}(\boldsymbol{x}) - (\boldsymbol{c}_{1:t} + \widetilde{\boldsymbol{c}}_{t+1})^\top \boldsymbol{x} \right\}, \qquad (9)$$

which calculates the decision vector using past utility observations $\boldsymbol{c}_{1:t}$, the aggregate regularizer $r_{0:t}(\boldsymbol{x})$ and the prediction

[6]In our case, since the routing is directly shaped by the caching, this restriction affects also z_{t+1}.

[7]We refer the reader to [35, Ch. 10] for variations on the regret metric

[8]A proximal regularizer induces a proximal mapping for the objective function; see [64, Ch. 6.1] for the formal definition.

Algorithm 1: Optimistic Bipartite Caching (π_{obc}).

```
1 Input: $\{\ell_{ij}\}_{(i,j)}$; $\{C_j\}_j$; $\mathcal{N}$; $\boldsymbol{x}_1 \in \mathcal{X}$; $\sigma = \sqrt{2}/D_{\mathcal{X}}$.
2 Output: $\boldsymbol{x}_t = (\boldsymbol{y}_t, \boldsymbol{z}_t)$, $\forall t$.
3 for $t = 1, 2, \ldots$ do
4     Route request $q_t$ according to configuration $\boldsymbol{x}_t$
5     Observe system utility $f_t(\boldsymbol{x}_t)$
6     Observe the new prediction $\widetilde{c}_{t+1}$
7     Update the regularizer $r_{0:t}(\boldsymbol{x})$ using (7)-(8)
8     Calculate the new policy $\boldsymbol{x}_{t+1}$ using (9)
  end
```

$\widetilde{\boldsymbol{c}}_{t+1}$. The update employs the negative gradients as it concerns a maximization problem. Henceforth, we refer to (9) as the *optimistic* FTRL (OFTRL) update. To provide some intuition on (9), ignore for a moment the regularization term. Our decision vector for $t+1$ is then simply the minimization of a linear cost (or, equivalently, the maximization of linear utility) that consists of the total cost witnessed until time t, plus the cost prediction for $t+1$. This is indeed a reasonable objective, provided that the prediction is accurate. However, the solution to any linear program is on the extremes of the decision set. Hence, the adversary can exploit this and make the learner jump between extreme points while placing maximum costs at those (e.g., via wrong predictions). Here comes the role of the regularization term, which stabilizes our decisions since the solution to the quadratic program is no longer on the extreme points. In fact, we want such regularization to be proportional to the witnessed accuracies of the predictions (hence the choices in (8)). This way, if the predictions are accurate, the linear program provides good decisions. Otherwise, the regularization induces a stable quadratic program whose solution cannot be arbitrarily harmed.

Policy π_{obc} is outlined in Algorithm 1. In each iteration, OBC solves a convex optimization problem, (9), involving a projection on the feasible set $\mathcal{X}$ (via $r_0(\boldsymbol{x})$). For the latter, one can rely on fast-projection algorithms specialized for caching, e.g., see [19]; while it is possible to obtain a closed-form solution for the OFTRL update for linear functions. We quantify next the performance of Algorithm 1.

Theorem 1: Algorithm 1 ensures the regret bound.

$$R_T \leq 2\sqrt{1+JC}\sqrt{\sum_{t=1}^{T} \|\boldsymbol{c}_t - \widetilde{\boldsymbol{c}}_t\|^2}.$$

For the proof, we modify the "strong FTRL lemma" [63, Lemma 5] by adding predictions for next-slot utility function. The following lemma bounds the regret in terms of the difference between two values of a strongly convex function evaluated at $\boldsymbol{x}_t$ and at $\boldsymbol{x}_{t+1}$, for each t.

Lemma 1: (Optimistic Strong FTRL Lemma) Let $v_t(\boldsymbol{x}) = -\boldsymbol{c}_t^\top \boldsymbol{x}_t + r_t(\boldsymbol{x}_t)$, and $v_{0:t}(\boldsymbol{x}) = -\boldsymbol{c}_{1:t}^\top \boldsymbol{x}_t + r_{0:t}(\boldsymbol{x}_t)$. Let $\boldsymbol{x}_{t+1}$ be selected according to (9). Then:

$$R_T \leq r_{0:T}(\boldsymbol{x}^\star) + \sum_{t=1}^{T} v_{0:t}(\boldsymbol{x}_t) - v_{0:t}(\boldsymbol{x}_{t+1}) - r_t(\boldsymbol{x}_t)$$
$$+ \widetilde{\boldsymbol{c}}_{T+1}^\top (\boldsymbol{x}_{T+1} - \boldsymbol{x}^\star) \tag{10}$$

Proof of Lemma 1:

$$\sum_{t=1}^{T} v_t(\boldsymbol{x}_t) - \left(v_{0:T}(\boldsymbol{x}^\star) - \widetilde{\boldsymbol{c}}_{T+1}^\top \boldsymbol{x}^\star\right)$$
$$= \sum_{t=1}^{T} \left(v_{0:t}(\boldsymbol{x}_t) - v_{0:t-1}(\boldsymbol{x}_t)\right) - \left(v_{0:T}(\boldsymbol{x}^\star) - \widetilde{\boldsymbol{c}}_{T+1}^\top \boldsymbol{x}^\star\right)$$
$$\leq \sum_{t=1}^{T} v_{0:t}(\boldsymbol{x}_t) + \sum_{t=1}^{T} \left(-v_{0:t-1}(\boldsymbol{x}_t)\right)$$
$$- \left(v_{0:T}(\boldsymbol{x}_{T+1}) - \widetilde{\boldsymbol{c}}_{T+1}^\top \boldsymbol{x}_{T+1}\right) \quad \text{(by def. of } \boldsymbol{x}_{T+1}\text{)}$$
$$\overset{(a)}{=} \sum_{t=1}^{T} v_{0:t}(\boldsymbol{x}_t) - v_0(\boldsymbol{x}_1) + \sum_{t=1}^{T-1} \left(-v_{0:t}(\boldsymbol{x}_{t+1})\right)$$
$$- v_{0:T}(\boldsymbol{x}_{T+1}) + \widetilde{\boldsymbol{c}}_{T+1}^\top \boldsymbol{x}_{T+1}$$
$$\overset{(b)}{\leq} \sum_{t=1}^{T} \left(v_{0:t}(\boldsymbol{x}_t) - v_{0:t}(\boldsymbol{x}_{t+1})\right) + \widetilde{\boldsymbol{c}}_{T+1}^\top \boldsymbol{x}_{T+1} \tag{11}$$

where equality (a) follows by reindexing the second sum (i.e., changing the sum index from t to $t+1$), and inequality (b) by dropping the non-positive term $-v_0(\boldsymbol{x}_1) = -r_0(\boldsymbol{x}_1)$ and appending the term $-v_{0:T}(\boldsymbol{x}_{T+1})$ to the second sum. Thus, we have that:

$$\sum_{t=1}^{T} v_t(\boldsymbol{x}_t) - v_{0:T}(\boldsymbol{x}^\star) + \widetilde{\boldsymbol{c}}_{T+1}^\top \boldsymbol{x}^\star$$
$$\leq \sum_{t=1}^{T} \left(v_{0:t}(\boldsymbol{x}_t) - v_{0:t}(\boldsymbol{x}_{t+1})\right) + \widetilde{\boldsymbol{c}}_{T+1}^\top \boldsymbol{x}_{T+1}. \tag{12}$$

Expanding the definition of $v_t(\boldsymbol{x}_t)$ and rearranging give the regret inequality:

$$\boldsymbol{c}_{0:T}^\top \boldsymbol{x}^\star - \sum_{t=1}^{T} \boldsymbol{c}_t^\top \boldsymbol{x}_t \leq r_{0:T}(\boldsymbol{x}^\star) + \sum_{t=1}^{T} \left(v_{0:t}(\boldsymbol{x}_t) - v_{0:t}(\boldsymbol{x}_{t+1})\right)$$
$$- r_t(\boldsymbol{x}_t) + \widetilde{\boldsymbol{c}}_{T+1}^\top (\boldsymbol{x}_{T+1} - \boldsymbol{x}^\star).$$

Noticing that the LHS is essentially the regret defined in (6) for $f_t(\boldsymbol{x}) = \langle \boldsymbol{c}_t, \boldsymbol{x}\rangle$ completes the proof.[9] □

With the weak assumption that the caching system will be notified upon the serving of the last request, we can set $\widetilde{\boldsymbol{c}}_{T+1} = 0$ and hence cancel the last term in the above inequality. Otherwise, it will be an additional constant factor in the regret bound.[10] Next, we will make use of the following results to bound each $v_{0:t}(\boldsymbol{x}_t) - v_{0:t}(\boldsymbol{x}_{t+1})$ term:

Lemma 2 [63, Lemma 7]: Let $\phi_1 : \mathbb{R}^n \to \mathbb{R}$ be a convex function such that $\boldsymbol{x}_1 = \arg\min_{\boldsymbol{x}} \phi_1$. Let ψ be a convex function such that $\phi_2(\boldsymbol{x}) = \phi_1(\boldsymbol{x}) + \psi(\boldsymbol{x})$ is strongly convex w.r.t

[9]This inequality holds for all $\boldsymbol{c}_t$, including $\sup_{\boldsymbol{c}} \langle \boldsymbol{c}, \boldsymbol{x}\rangle$.

[10]It is possible to slightly change the semantics of the algorithm to avoid this rare case by making the adversary first commit and hide the cost function, and then the learner pick the action, see [65, Thm 7.29].

norm $\|\cdot\|$. Then, for any $\boldsymbol{b} \in \partial\psi(\boldsymbol{x}_1)$ and $\boldsymbol{x}'$, we have that $\phi_2(\boldsymbol{x}_1) - \phi_2(\boldsymbol{x}') \leq \frac{1}{2}\|\boldsymbol{b}\|_\star^2$.

Now we are ready to prove Theorem 1:

Proof of Theorem 1: We start by applying Lemma 2 to the result in (10). Namely, we select:

$$\phi_1(\boldsymbol{x}) = v_{0:t}(\boldsymbol{x}) + \boldsymbol{c}_t^\top \boldsymbol{x}_t - \widetilde{\boldsymbol{c}}_t^\top \boldsymbol{x}_t, \text{ and}$$
$$\phi_2(\boldsymbol{x}) = \phi_1(\boldsymbol{x}) - \boldsymbol{c}_t^\top \boldsymbol{x}_t + \widetilde{\boldsymbol{c}}_t^\top \boldsymbol{x}_t.$$

This way, we have that $\boldsymbol{x}_t = \arg\min \phi_1(\boldsymbol{x}_t)$, $\phi_2(\boldsymbol{x}) = v_{0:t}(\boldsymbol{x})$, $\psi(\boldsymbol{x}) = (-\boldsymbol{c}_t + \widetilde{\boldsymbol{c}}_t)^\top \boldsymbol{x}$, and $(-\boldsymbol{c}_t + \widetilde{\boldsymbol{c}}_t) \in \partial\psi(\boldsymbol{x})$.

Then, dropping the non-positive terms $-r_t(\cdot)$ in (10), setting $\widetilde{\boldsymbol{c}}_{T+1} = 0$, and defining the norm $\|\cdot\|_{(t)} = \sqrt{\sigma_{1:t}}\|\cdot\|$ so that the regularizer $r_{1:t}(\boldsymbol{x})$ is 1-strongly-convex w.r.t. $\|\cdot\|_{(t)}$, we get:

$$R_T \leq r_{1:T}(\boldsymbol{x}^\star) + \frac{1}{2}\sum_{t=1}^{T} \|\boldsymbol{c}_t - \widetilde{\boldsymbol{c}}_t\|_{(t),\star}^2, \quad \forall \boldsymbol{x}^\star \in \mathcal{X}. \tag{13}$$

Now, we have that $r_t \leq \frac{\sigma_t}{2} D_\mathcal{X}^2$, where $D_\mathcal{X}$ is the euclidean diameter of the set $\mathcal{X}$ i.e., $\forall \boldsymbol{x}, \boldsymbol{x}_t \in \mathcal{X}$:

$$\begin{aligned}\|\boldsymbol{x} - \boldsymbol{x}_t\|^2 &= \sum_{n,j}(y_{nj} - y_{nj}^t)^2 + \sum_{n,i,j}(z_{nij} - z_{nij}^t)^2 \\ &\overset{(a)}{\leq} \sum_{n,j}|y_{nj} - y_{nj}^t| + \sum_{n,i,j}|z_{nij} - z_{nij}^t| \\ &\overset{(b)}{\leq} 2(JC+1) \triangleq D_\mathcal{X}^2\end{aligned}$$

where (a) holds as $y_{nj}, z_{nij} \in [0,1], \forall n,i,j$; (b) holds by the triangle inequality and definitions of $\mathcal{Y}$, $C \triangleq \max_j C_j$, and the fact that the routing variables differ at one coordinate only. To see why, recall that we serve one request per time slot and we set the routing variable $\boldsymbol{z}_t$ after observing that request $\boldsymbol{q}_t$. Hence, we can modify the routing variables and set $z_{nij}^\star = z_{nij}^t = 0 \; \forall n \neq n', i \neq i', j \neq j'$, where $q_{n'i'} = 1 \wedge \ell_{i'j'=1}$. Due to the structure of $f_t(\cdot)$, the utility of these modified $z_{nij}^\star$ and z_{nij}^t will not change (compared to their utility before modification). In words, knowing the requested file n', and the location from which it is requested i', there will be no utility from routing a non-requested file $n \neq n'$, or routing from a non-connected cache $j \neq j'$. Thus, we can zero these variables and get a smaller value for $D_\mathcal{X}$, without affecting the utility values.

Using this diameter bound, and the fact that the dual norm of $\|\boldsymbol{x}\|_{(t)}$ is $\|\boldsymbol{x}\|_{(t),\star} = \|\boldsymbol{x}\|/\sqrt{\sigma_{1:t}}$, inequality (13) can be written as:

$$R_T \leq \frac{\sigma_{1:T}}{2} D_\mathcal{X}^2 + \frac{1}{2}\sum_{t=1}^{T} \frac{h_t}{\sigma_{1:t}}. \tag{14}$$

Note that the sum $\sigma_{1:t}$ telescopes and evaluates to $\sigma\sqrt{h_{1:t}}$. Using this observation and substituting it in (14), and combining it with [66, Lem. 3.5] to bound the second term as follows $\sum_t^T h_t/\sqrt{h_{1:t}} \leq 2\sqrt{h_{1:T}}$, we eventually get:

$$R_T \leq \frac{\sigma}{2}\sqrt{h_{1:T}} D_\mathcal{X}^2 + \frac{1}{\sigma}\sqrt{h_{1:T}} \overset{(a)}{\leq} \sqrt{2} D_\mathcal{X}\sqrt{h_{1:T}},$$

where (a) is obtained by setting $\sigma = \sqrt{2}/D_\mathcal{X}$. Finally, substituting the actual diameter value, namely $D_\mathcal{X} = \sqrt{2(JC+1)}$, completes the proof. □

Discussion: Theorem (1) shows that the regret does not depend on the library size N and is also modulated by the quality of the predictions; accurate predictions tighten the bound, and in the case of perfect predictions, i.e., when users follow the recommendations, we get negative regret $R_T \leq 0, \forall T$, which is much stronger than the sub-linear growth rates in other works [19], [67]. In fact, even when predictions fail for a constant number of time slots $L \in [T]$, the regret will be constant of the same order $R_T \leq O(L)$, which is still a significant improvement over any time-dependent bound.

On the other hand, for worst-case prediction, we can still use the bound $\|\boldsymbol{c}_t - \widetilde{\boldsymbol{c}}_t\|^2 \leq 2\,w^2$, and get:

$$R_T \leq 2\sqrt{2}w\sqrt{JC+1}\sqrt{T} = O(\sqrt{T})$$

i.e., the regret is at most a constant factor worse than the regret of those policies that do not incorporate predictions.[11] Thus, OBC offers an efficient and safe approach for incorporating predictions in cases where we are uncertain about their accuracy, e.g., either due to the quality of the RecSys or the behavior of users.

Another key point is that the *utility parameters might vary with time* as well. Indeed, replacing $\boldsymbol{w}_t = (w_{nij}^t \leq w, n \in \mathcal{N}, i \in \mathcal{I}, j \in \mathcal{J})$ in $f_t(\boldsymbol{x}_t)$ does not affect the analysis nor the bound. This is important when the caching system employs a wireless network where the link capacities vary, or when the caching utility changes; and we note that this utility can be even *file-specific*. Parameters $\boldsymbol{w}_t$ can be also unknown when $\boldsymbol{x}_t$ is decided, exactly as it is with $\boldsymbol{q}_t$, and they can be predicted using e.g., channel measurements. Essentially the proposed model drops several restricted assumptions of prior works regarding, not only the knowledge of request rates/densities, but also about the system state, link quality, and user utilities. Finally, we observe that in case the algorithm is used to optimize the operation of an edge computing system, these parameters can capture the potentially time-varying utility of each computation, for each service (n) and each pair of user - cache.

On a technical note, Lemma 1 presents a novel theoretical result and enables the improvement of the best known proximal OFTRL bound of [42] by a constant factor of $\sqrt{2}$. This also opens the door for leveraging the modular analysis tools developed in [63] in the optimistic OCO framework for different special cases of the utility functions. This technical result is of independent interest.

Lastly, in the case of more than one request per time slot (e.g., B requests), the euclidean norm would instead be $D_\mathcal{X}^2 = 2(JC+B)$. Therefore, the same results hold in terms of the regret being always sub-linear and commensurate with the prediction accuracy. However, the worst case bounds will of course be scaled by a factor of $\sqrt{B}$ since now we would use $\|\boldsymbol{c}_t - \widetilde{\boldsymbol{c}}_t\|^2 \leq 2B\,w^2$.

[11]The factor is $\sqrt{2}$ compared to the "any-time" version of the bound that does not use predictions, and 2 compared to those that assume a known T.

Algorithm 2: Optimistic Elastic Caching (π_{oec}).

```
1 Input: ${\ell_{ij}}_{(i,j)}$, ${C_j}_j$, $\mathcal{N}$, $\lambda_1=0$, $x_1 \in \mathcal{X}_e$.
2 Output: $x_t = (y_t, z_t)$, $\forall t$.
3 for $t = 1, 2, \ldots$ do
4     Route request $q_t$ according to configuration $x_t$
5     Observe system utility $f_t(\boldsymbol{x}_t)$ and cost $g_t(\boldsymbol{x}_t)$
6     Update the budget parameter $\lambda_{t+1}$ using (19)
7     Update the regularizers $r_{0:t}(x)$ using (7)-(8), and
      $a_t = at^{-\beta}$
8     Observe prediction $\widetilde{c}_{t+1}$ and price $s_{t+1}$
9     Calculate the new policy $x_{t+1}$ using (20)
  end
```

IV. Optimistic Caching in Elastic Networks

We extend our analysis to *elastic* caching networks where the caches can be resized dynamically. Such architectures are important for two reasons. First, there is a growing number of small-size content providers that implement their services by leasing storage on demand from infrastructure providers [68]; and second, CDNs often resize their caches responding to the time-varying user needs and operating expenditures [69].

We introduce the t-slot price vector $\boldsymbol{s}_t = (s_j^t \leq s, j \in \mathcal{J})$, where s_j^t is the leasing price per unit of storage at cache j in slot t, and s its maximum value. In the general case, these prices may change arbitrarily over time, e.g., because the provider has a dynamic pricing scheme or the electricity cost changes [44], [45]; hence the caching system has access only to $\boldsymbol{s}_t$ at each slot t. We denote with B_T the budget the system intends to spend during a period of T slots for leasing cache capacity. The objective is to maximize the caching gains while satisfying:

$$\sum_{t=1}^{T} g_t(\boldsymbol{x}_t) = \sum_{t=1}^{T}\sum_{j\in\mathcal{J}}\sum_{n\in\mathcal{N}} s_j^t y_{nj}^t - B_T \leq 0. \tag{15}$$

In particular, the new benchmark problem in this case is:

$$\mathbb{P}_2: \quad \max_{x\in\mathcal{X}} \sum_{t=1}^{T} f_t(\boldsymbol{x}), \qquad \text{s.t.} \sum_{t=1}^{T} \mathrm{g}_t(\boldsymbol{x}) \leq 0, \tag{16}$$

which differs from $\mathbb{P}_1$ due to the leasing constraint.

Indeed, in this case the regret is defined as:

$$R_T^{(e)}(\pi) = \sup_{\{f_t\}_{t=1}^T} \left[\sum_{t=1}^{T} f_t(\boldsymbol{x}^\star) - \sum_{t=1}^{T} f_t(\boldsymbol{x}_t)\right], \tag{17}$$

where

$$\boldsymbol{x}^\star \in \mathcal{X}_e \triangleq \{\boldsymbol{x} \in \mathcal{X} \mid (3), g_t(\boldsymbol{x}) \leq 0, \forall t\},$$

i.e., $\boldsymbol{x}^\star$ is a feasible point of $\mathbb{P}_1$ with the newly introduced additional restriction to satisfy the budget constraint $g_t(\boldsymbol{x}) \leq 0$ in every slot. In the definition of $\mathcal{X}$, C now denotes the maximum leasable space. Learning problems with time-varying constraints are hard, see impossibility result in [70], and hence require such additional restrictions on the selected benchmarks. We refer the reader to [47] for a related discussion, and to [48], [49] for different benchmarks. Finally, apart from $R_T^{(e)}$, we need also to ensure a sublinear growth rate for the budget violation:

$$V_T^{(e)} = \sum_{t=1}^{T} [g_t(\boldsymbol{x}_t)]_+ .$$

To tackle this new problem we follow a saddle point analysis, which is new in the context of OFTRL.

Namely, we first define a Lagrangian-type function by relaxing the budget constraint and introducing the dual variable $\lambda \geq 0$:

$$\mathcal{L}_t(\boldsymbol{x}, \lambda) = \frac{\sigma_t}{2}\|\boldsymbol{x} - \boldsymbol{x}_t\|^2 - f_t(\boldsymbol{x}) + \lambda g_t(\boldsymbol{x}) - \frac{\lambda^2}{a_t}. \tag{18}$$

The last term is a non-proximal regularizer for the dual variable; and we use $a_t = at^{-\beta}$, where parameter $\beta \in [0, 1)$ can be used to prioritize either $R_T^{(e)}$ or $V_T^{(e)}$. The main ingredients of policy π_{oec} are the saddle-point iterations:

$$\lambda_{t+1} = \arg\max_{\lambda\geq 0} \left\{ -\frac{\lambda^2}{a_{t+1}} + \lambda \sum_{i=1}^{t} g_i(\boldsymbol{x}_i) \right\}, \tag{19}$$

$$\boldsymbol{x}_{t+1} = \arg\min_{\boldsymbol{x}\in\mathbb{R}^m} \left\{ r_{0:t}(\boldsymbol{x}) + \left(\sum_{i=1}^{t+1} \lambda_i \boldsymbol{s}_i - \boldsymbol{c}_{1:t} - \widetilde{\boldsymbol{c}}_{t+1} \right)^\top x \right\}, \tag{20}$$

and its implementation is outlined in Algorithm 2. Note that we use the same regularizer as in Section III for the primal variables $\boldsymbol{x}_t$, while λ_t modulates the caching decisions by serving as a *shadow price* for the average budget expenditure.

The performance of Algorithm OEC is characterized in the next theorem.

Theorem 2: Algorithm 2 ensures the bounds.

$$R_T^{(e)} \leq \sqrt{2} D_{\mathcal{X}} \sqrt{\sum_{t=1}^{T} \|\boldsymbol{c}_t - \widetilde{\boldsymbol{c}}_t\|^2} + \frac{aM}{2} T^{1-\beta},$$

$$V_T^{(e)} \leq \sqrt{\frac{2\sqrt{2}D_{\mathcal{X}}T^\beta}{a}\sqrt{\sum_{t=1}^{T}\|\boldsymbol{c}_t - \widetilde{\boldsymbol{c}}_t\|^2} + MT - \frac{2R_T^{(e)}T^\beta}{a}},$$

where we have used the grouped constants $M = \frac{(sJC)^2}{1-\beta}$.

Proof: Observe that the update in (20) is similar to (9) but applied to the Lagrangian in (18) instead of just the utility, and the known prices when $\boldsymbol{x}_{t+1}$ is decided represent perfect prediction for $g_t(\boldsymbol{x})$. Using Theorem 1 with $\boldsymbol{c}_t - \lambda_t \boldsymbol{s}_t$ instead of $\boldsymbol{c}_t$, and $\widetilde{\boldsymbol{c}}_t - \lambda_t \boldsymbol{s}_t$ instead of $\widetilde{\boldsymbol{c}}_t$, we can write:

$$\sum_{t=1}^{T} \left(f_t(\boldsymbol{x}^\star) - f_t(\boldsymbol{x}_t) + \lambda_t g_t(\boldsymbol{x}_t) - \lambda_t g_t(\boldsymbol{x}^\star)\right) \leq \sqrt{2} D_{\mathcal{X}} \sqrt{h_{1:T}},$$

and rearrange to obtain:

$$R_T^{(e)} \leq \sqrt{2} D_{\mathcal{X}} \sqrt{h_{1:T}} + \sum_{t=1}^{T} \lambda_t g_t(\boldsymbol{x}^\star) - \sum_{t=1}^{T} \lambda_t g_t(\boldsymbol{x}_t). \tag{21}$$

For the dual update (19), we can use the non-proximal-FTRL bound [63, Theorem 1] to write:

$$-\sum_{t=1}^{T}\lambda_t g_t(\boldsymbol{x}_t)+\lambda\sum_{t=1}^{T}g_t(\boldsymbol{x}_t)\leq\frac{\lambda^2}{a_T}+\frac{1}{2}\sum_{t=1}^{T}a_t g_t^2(\boldsymbol{x}_t). \tag{22}$$

Since $g_t(\boldsymbol{x}^\star)\leq 0,\forall t$ and combining (21), (22) we get:

$$R_T^{(e)}\leq\sqrt{2}D_{\mathcal{X}}\sqrt{h_{1:T}}-\lambda\sum_{t=1}^{T}g_t(\boldsymbol{x}_t)+\frac{\lambda^2}{a_T}+\frac{1}{2}\sum_{t=1}^{T}a_t g_t^2(\boldsymbol{x}_t) \tag{23}$$

Setting $\lambda=0$, using the identity:

$$\sum_{t=1}^{T}at^{-\beta}\leq\frac{aT^{1-\beta}}{1-\beta}$$

and the bound $g_t(\boldsymbol{x}_t)\leq sJC$, we arrive at the $R_T^{(e)}$ bound.

For the violations, we use the following property in (23):

$$\frac{a_T}{2}\left[\sum_{t=1}^{T}g_t(\boldsymbol{x}_t)\right]_+^2=\sup_{\lambda\geq 0}\left[\sum_{t=1}^{T}g_t(\boldsymbol{x}_t)\lambda-\frac{\lambda^2}{2a_T}\right],$$

Rearranging, we get:

$$\frac{a_T}{2}\left(V_T^{(e)}\right)^2\leq\sqrt{2}D_{\mathcal{X}}\sqrt{h_{1:T}}+\frac{a(sJC)^2}{2-2\beta}T^{1-\beta}-R_T^{(e)}.$$

Finally, taking the square root yields the $V_T^{(e)}$ bound. □

Discussion: The worst-case bounds in Theorem 2 arise when the predictions are failing. In that case, we have $\|\boldsymbol{c}_t-\widetilde{\boldsymbol{c}}_t\|^2\leq 2\,w^2$ and use the bound $-R_T^{(e)}=O(T)$ for the last term of $V_T^{(e)}$, to obtain $R_T^{(e)}=O(T^\kappa)$, with $\kappa=\max\{1/2,1-\beta\}$ while $V_T^{(e)}=O(T^\phi)$, with $\phi=\frac{1+\beta}{2}$. Hence, for $\beta=1/2$ we achieve the desired sublinear rates $R_T^{(e)}=O(\sqrt{T}),V_T^{(e)}=O(T^{3/4})$. However, when the RecSys manages to predict accurately the user preferences, the performance of π_{oec} improves substantially as the first terms in each bound are eliminated. Thus, for bounded T, we practically halve the regret and violation bounds.

It is also interesting to observe the tension between $V_T^{(e)}$ and $R_T^{(e)}$, which is evident from the $V_T^{(e)}$ bound and the condition $-R_T^{(e)}=O(T)$. The latter refers to the upper bound of the *negative* regret, thus when it is consistently satisfied (i.e., for all T), we obtain an even better result: π_{oec} *outperforms* the benchmark. Another likely case is when $-R_T^{(e)}=O(\sqrt{T})$, i.e., the policy does not outperform the benchmark at a rate larger than $\sqrt{T}$. Then, Theorem 2 yields $R_T^{(e)}=O(T^\kappa)$ with $\kappa=\max\{1/2,1-\beta\}$ while $V_T^{(e)}=O(T^\phi)$ with $\phi=\max\{1/2,1/4+\beta/2\}$. Hence, for $\beta=1/2$ the rates are reduced to $R_T^{(e)}=O(\sqrt{T}),V_T^{(e)}=O(\sqrt{T})$.

Finally, it is worth observing that π_{oec} can be readily extended to handle additional budget constraints such as time-average routing costs or average delays. And one can also generalize the approach to consider a budget-replenishment process where in each slot t the budget increases by an amount of b_t units. This is made possible due to the generality of the conditions (model perturbations can be non-stationary and correlated) under which the regret and violation bounds hold.

V. Caching With Multiple Predictors

In this section, we consider the case where we have additional predictors, apart from the RecSys, predicting the next-slot utility. Thus, we have a set of predictions $\{\widetilde{\boldsymbol{c}}_t^{(p)},p\in\mathcal{P}\}$ at each slot t. To handle this setup and benefit from this abundance of predictions, we take a different approach and instead of using prediction-adaptive regularizers, as in the previous sections, we model the predictions as experts using the classical paradigm of learning through experts cf. [35]. Based on this approach, we design a novel tailored optimistic *meta-learning* policy to accrue the best possible caching gains.

In particular, we associate an expert to each predictor, and we will abuse notation denoting them both with $p\in\mathcal{P}$. We refer to these predictors-linked experts as the *optimistic experts*. Every optimistic expert p proposes its caching action $\{\boldsymbol{y}_t^{(p)}\}_t$ at each slot t, by solving the following problem:[12]

$$\boldsymbol{y}_t^{(p)}=\underset{\boldsymbol{y}\in\mathcal{Y}}{\text{argmax}}\ \ \widetilde{\boldsymbol{c}}_t^{(p)\top}\,\boldsymbol{y}. \tag{24}$$

Note that (24) is indeed a certainty-equivalent[13] linear program. We denote with $R_T^{(p)}$ the regret of each expert w.r.t the optimal-in-hindsight caching configuration for the entire time period of T slots, i.e.:

$$\boldsymbol{y}^\star=\underset{y\in\mathcal{Y}}{\text{argmax}}\,\boldsymbol{c}_{1:T}^\top\,\boldsymbol{y}.$$

Besides the optimistic experts, we consider an expert that *does not use predictions*. This special expert proposes an FTRL-based caching policy, and we refer to it as the *pessimistic expert* and associate it with the special index $p=0$. The pessimistic expert proposes caching actions $\{\boldsymbol{y}_t^{(0)}\}_t$ according to (9), but setting $\widetilde{\boldsymbol{c}}_t=0$ for the regularization parameter σ_t in (8). Its regret is denoted with $R_T^{(0)}$. Our full experts set is the union of the group of optimistic experts with the pessimistic expert $\mathcal{P}^+=\{0\cup\mathcal{P}\}$.

We aim to learn a caching policy whose regret is upper-bounded by the regret of the best expert and does not exceed the $O(\sqrt{T})$ regret of the pessimistic expert. Such a methodology of modeling policies as experts has been studied in the past [71]. However, this is the first work that implements optimistic learning through an experts model. In addition, here we also make the next step and propose to include a prediction for *the performance of each predictor*.

In particular, the caching decision is the convex combination of experts' proposals according to the *weights* $\boldsymbol{u}_t=(u_t^{(p)},p\in\mathcal{P}^+)$ selected from the simplex:

$$\Delta_{\mathcal{P}^+}=\left\{\boldsymbol{u}\in[0,1]^{P+1}\middle|\sum_{p\in\mathcal{P}^+}u_t^{(p)}=1\right\},$$

[12]To streamline the presentation, our analysis focuses on one cache, hence using only y_t decisions. However, this method can be readily extended to caching networks as discussed later.

[13]A certainty-equivalent program is one that considers a predicted utility vector as true and optimizes the decisions accordingly.

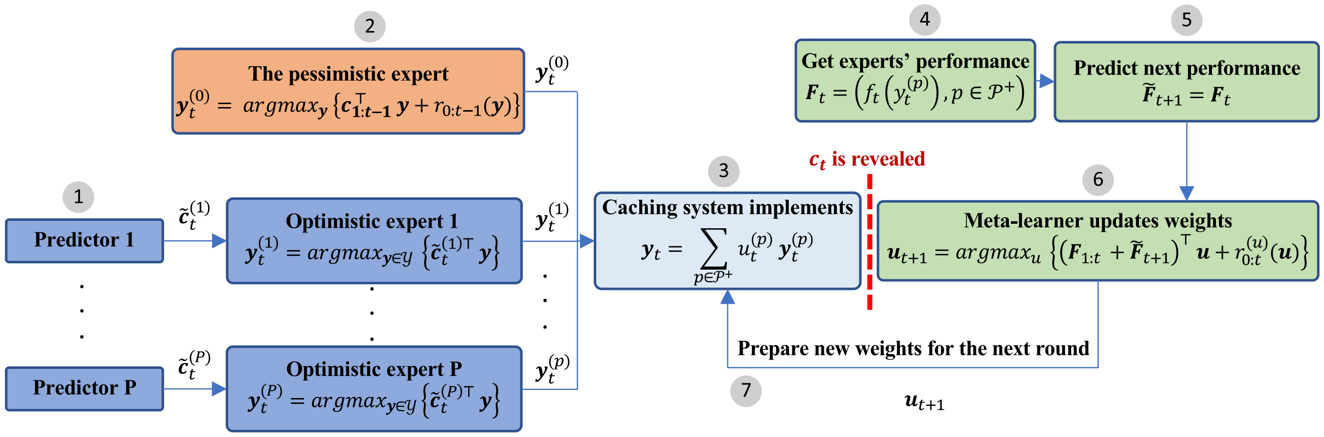
2
The pessimistic expert
$y_t^{(0)} = argmax_y \{c_{1:t-1}^\top y + r_{0:t-1}(y)\}$
$y_t^{(0)}$
1
Predictor 1
$\tilde{c}_t^{(1)}$
Optimistic expert 1
$y_t^{(1)} = argmax_{y\in\mathcal{Y}} \{\tilde{c}_t^{(1)\top} y\}$
$y_t^{(1)}$
Predictor P
$\tilde{c}_t^{(P)}$
Optimistic expert P
$y_t^{(P)} = argmax_{y\in\mathcal{Y}} \{\tilde{c}_t^{(P)\top} y\}$
$y_t^{(P)}$
3
Caching system implements
$y_t = \sum_{p\in\mathcal{P}^+} u_t^{(p)} y_t^{(p)}$
c_t is revealed
4
Get experts' performance
$F_t = (f_t(y_t^{(p)}), p\in\mathcal{P}^+)$
5
Predict next performance
$\tilde{F}_{t+1} = F_t$
6
Meta-learner updates weights
$u_{t+1} = argmax_u \{(F_{1:t} + \tilde{F}_{t+1})^\top u + r_{0:t}^{(u)}(u)\}$
Prepare new weights for the next round
7
u_{t+1}

Fig. 2. Decision step for the meta-learning policy π_{xc}. The policy is a combination of experts' proposals (predictors) according to their priority weights that were learned based on observations that are collected until slot t.

Algorithm 3: Experts Caching (π_{xc}).

```
1  Input: $C$; $y_1 \in \mathcal{Y}$; $\sigma = \sqrt{2}/D_{\mathcal{Y}}$.
2  Output: $y_t$, $\forall t$.
3  for $t = 1, 2, \ldots$ do
4      Observe utility predictions $\{\tilde{c}_t^{(p)}, p \in \mathcal{P}\}$
5      Calculate optimistic proposals $\{y_t^{(p)}\}_{p\in\mathcal{P}}$ with
        (24)
6      Update $r_{0:t-1}(x)$ using (7)-(8) with $\tilde{c}_t = 0$
7      Calculate pessimistic proposal $y_t^{(0)}$ with (9)
8      Serve request $q_t$ with meta-policy $y_t$ from (25)
9      Observe the utilities of experts' proposals
        $F_t = \left(f_t(y_t^{(p)}), p \in \mathcal{P}^+\right)$
10     Set experts' performance prediction $\tilde{F}_{t+1} = F_t$
11     Calculate the new weights $u_{t+1}$ using (26)
   end
```

namely:

$$y_t = \sum_{p\in\mathcal{P}^+} u_t^{(p)}\, y_t^{(p)}. \tag{25}$$

Thus, y_{t+1} remains a feasible caching vector, despite being produced by mixing all the experts' proposals. The mixing weights, $\{u_t\}_t$, are updated through a new OFTRL step, which is similar to the updates of π_{obc} but we use the superscript (u) for the involved parameters to make clear the distinction. In particular, the update is:

$$u_{t+1} = \arg\min_{u\in\Delta_{\mathcal{P}^+}} \left\{ r_{0:t}^{(u)}(u) - (F_{1:t} + \tilde{F}_{t+1})^\top u \right\}, \tag{26}$$

where $F_t = (f_t(y_t^{(p)}), p \in \mathcal{P}^+)$ is the t-slot performance vector for the experts. The regularizers and the respective parameters, in this case, are decided by the following formulas:

$$r_0^{(u)}(u) = I_{\Delta_{\mathcal{P}^+}}(u), \quad r_t^{(u)}(u) = \frac{\sigma_t^{(u)}}{2}\|u - u_t\|^2, \; t \geq 1,$$

$$\sigma_1^{(u)} = \sigma^{(u)}\sqrt{h_1^{(u)}}, \sigma_t^{(u)} = \sigma^{(u)}\left(\sqrt{h_{1:t}^{(u)}} - \sqrt{h_{1:t-1}^{(u)}}\right), \; t \geq 2,$$

$$\text{with} \quad h_t^{(u)} = \|F_t - \tilde{F}_t\|^2. \tag{27}$$

For the predictions of the *experts' performance* $\tilde{F}_t$, at each time step, we will use experts' performance of the previous time step:[14]

$$\tilde{F}_t = \left(\tilde{f}_t^{(p)},\ p \in \mathcal{P}^+\right) \triangleq \left(f_{t-1}(y_{t-1}^{(p)}),\ p \in \mathcal{P}^+\right). \tag{28}$$

This type of meta-optimism comes for free since, in practice, we expect the predictors to have consistent accuracy across contiguous slots; either accurately due to a recently trained model, or poorly due to e.g., distributional shift (see [72] and references therein) hence we can use the previous function. As for the pessimistic expert, its caching decisions do not vary much in consecutive slots (due to using strongly convex regularizers in updating the decisions).

The execution of the policy is summarized in Algorithm 3; and we also include the step-by-step visualization in Fig. 2. We see the sequence of steps where: (1) The Predictors output $\{\tilde{c}_t^{(p)}, p \in \mathcal{P}\}$; (2) The optimistic experts optimize for the predictions, whereas the pessimistic expert performs an FTRL step; (3) The experts' proposals are combined via the meta-learner weights; (4) The performance of experts' proposals is calculated through the revealed request; (5) The meta-learner optimistically sets the next-slot experts' performance to be the same as the current one; (6) & (7) The meta-learner performs an OFTRL step to calculate and set the weights for the next-slot. And the process repeats for the next slot.

The following theorem bounds the regret of the proposed meta-learning policy, which is defined as:

$$R_T^{(xc)} = \sum_{t=1}^{T} c_t{}^\top (y^\star - y_t).$$

[14]We note that the motivation for a large corpus of optimistic learning works stems from the fact that in many cases the cost functions are changing slowly [41], [42]. While the motivation in this work has been different until this section (availability of RecSys), the optimism in the meta-learner has the same scope as those initial works.

Recall that vector $\boldsymbol{c}_t$ indicates which file is requested and the utility associated with the request, and vectors $\boldsymbol{y}^\star$ and $\boldsymbol{y}_t$ are the optimal caching decisions/ the caching at time slot t, respectively.

Theorem 3: Algorithm 3 ensures the regret bound:

$$R_T^{(xc)} \leq 2\sqrt{\sum_{t=1}^{T} \|\boldsymbol{F}_t - \boldsymbol{F}_{t-1}\|^2} + \min_{p\in\mathcal{P}+}\left\{R_T^{(p)}\right\}$$
$$\leq 2w\sqrt{(P+1)T} + \min_{p\in\mathcal{P}+}\left\{R_T^{(p)}\right\}.$$

Proof: We first relate the regret of the combined caching decisions to that of the experts. Then, we can re-use the result of Theorem 1:

$$R_T^{(xc)} = \sum_{t=1}^{T}\left(\boldsymbol{c}_t{}^\top \boldsymbol{y}^\star - \boldsymbol{c}_t{}^\top \sum_{p\in\mathcal{P}+} u_t^{(p)}\,\boldsymbol{y}_t^{(p)}\right)$$
$$= \sum_{t=1}^{T} \boldsymbol{c}_t{}^\top \boldsymbol{y}^\star - \boldsymbol{F}_t{}^\top \boldsymbol{u}_t$$
$$= \sum_{t=1}^{T} \boldsymbol{c}_t{}^\top \boldsymbol{y}^\star - \boldsymbol{F}_t{}^\top \boldsymbol{u}^\star + \boldsymbol{F}_t{}^\top \boldsymbol{u}^\star - \boldsymbol{F}_t{}^\top \boldsymbol{u}_t$$
$$= R_T^{(u)} + \min_{p\in\mathcal{P}+}\left\{R_T^{(p)}\right\} \quad (29)$$

where $R_T^{(u)}$ is the regret for the *weights* $\boldsymbol{u}$: $R_T^{(u)} = \sum_{t=1}^{T} \boldsymbol{F}_t{}^\top \boldsymbol{u}^\star - \boldsymbol{F}_t{}^\top \boldsymbol{u}_t$. Note that (29) holds because $\boldsymbol{u}^\star = \text{argmax}_{\boldsymbol{u}}\, \boldsymbol{F}_{1:t}^\top \boldsymbol{u} = \boldsymbol{e}^{\boldsymbol{k}}, k = \text{argmax}_p f_t(\boldsymbol{y}_t^{(p)})$ and $\boldsymbol{e}^k$ is standard basis vector. Thus, we have:

$$\boldsymbol{F}_{1:t}^\top\, \boldsymbol{u}^\star = \max_{p\in\mathcal{P}+}\left\{\sum_{t=1}^{T} f_t\left(\boldsymbol{y}_t^{(p)}\right)\right\}. \quad (30)$$

We use the result of Theorem 1 to bound $R_T^{(u)}$:

$$R_T^{(u)} \leq \sqrt{2} D_{\Delta_{\mathcal{P}+}} \sqrt{\sum_{t=1}^{T} \|\boldsymbol{F}_t - \widetilde{\boldsymbol{F}}_t\|^2} \leq 2w\sqrt{(P+1)T}, \quad (31)$$

where the last inequality follows from the simplex diameter ($D_{\Delta_{\mathcal{P}+}} \leq \sqrt{2}$) and the fact that:

$$\|\boldsymbol{F}_t - \widetilde{\boldsymbol{F}}_t\|^2 \leq \sum_{p\in\mathcal{P}+} \|\left(f_t(\boldsymbol{y}_t^{(p)}) - f_{t-1}(\boldsymbol{y}_{t-1}^{(p)})\right)\|^2 \leq |\mathcal{P}^+|w^2$$

i.e., we predicted a miss or a hit, whichever happened at $t-1$, but the opposite happens at t. Substituting in (29) gives the bound.□

Discussion: A key observation in Theorem 3 is that its bound contains the regret of the best optimistic expert. This yields very improved bounds for $R_T^{(xc)}$ whenever there is an optimistic expert that achieves negative regret. For example, if an expert can achieve $R_T = \Theta(-T)$, e.g., because it has a very accurate RecSys (its recommendations are most-often followed), then the overall regret is $R_T^{(xc)} = O(\sqrt{T}) - \Theta(-T)$, which becomes strictly negative for large T. Moreover, the $O(\sqrt{T})$ term shrinks with more consistent performance of the experts, a condition that is rather expected in practical systems, thus getting us even faster to the regret of the best expert. Nonetheless, since the pessimistic expert exists in the group of experts, the min term is upper bounded by $O(\sqrt{T})$ and $R_T^{(xc)}$ will maintain $O(\sqrt{T})$ regardless of the performance of the optimistic experts.

Since Algorithm 3 can work with a single optimistic expert, which is the setup handled by π_{obc}, it is interesting to compare their bounds. In fact, neither of those algorithms is better in all possible scenarios (regarding request patterns; evolution of utility parameters, etc. see also Section VI) than the other; and their relative performance ranking (in terms of utility) depends on the specific problem instance. For example, under worst-case predictions, we have that:[15]

$$R_T^{(xc)} \leq 2\,w\sqrt{(P+1)T} + 2w\sqrt{CT}$$
$$\leq 2(\sqrt{2}+\sqrt{C})w\sqrt{T} \qquad (P=1), \quad (32)$$

which can be actually better than policy π_{obc}'s worst-case prediction bound[16] for practical values of the constants C.

On the other hand, in cases where inaccurate predictions occur for a certain fraction of the steps $\lceil\alpha T\rceil, 0<\alpha<1$ the min term in the $R_T^{(xc)}$ bound might evaluate to the pessimistic expert's regret since the optimistic experts can suffer linear regrets[17] $R_T^{(p)} \leq O(\alpha T)$. For π_{obc}, the regret will be of the form

$$R_T \leq 2\sqrt{C}\sqrt{\sum_{t\in[\lceil\alpha T\rceil]} \|\widetilde{\boldsymbol{c}}_t - \boldsymbol{c}_t\|}$$
$$\leq 2\sqrt{2\,C}w\sqrt{\lceil\alpha T\rceil}. \quad (33)$$

For example, for $\alpha \leq 1/2$, The upper bound in (33) is tighter than that in (32) for all C, w. Hence, π_{obc}'s regret can be smaller if in the described case. Overall, the choice between π_{xc} and π_{obc} in the case of a single predictor depends on the request sequence and the number of steps where predictions fail. Section VI demonstrates these cases using various scenarios.

Regarding the extension to caching networks (more than one cache), note that sets $\mathcal{Y}$ and $\mathcal{Z}$ are convex by definition. Also, the set defined by the connectivity constraints (3) is convex (linear constraint in the variable x). Recalling that the Cartesian product and the intersection of convex sets is convex, we conclude that the constraint set $\mathcal{X}$ defined in (5), from which the joint caching-routing variable $\boldsymbol{z}$ is selected, remains convex. Finally, as demonstrated earlier, the meta-learner takes the convex combination of the experts' proposals, and since each expert proposes a caching-routing configuration $\boldsymbol{z}_t^{(p)} \in \mathcal{X}$, the

[15]The pessimistic expert's regret is bounded by $2w\sqrt{CT}$ for the single cache setup (i.e., the diameter $D_{\mathcal{Y}} = \sqrt{2\,C}$).

[16]Note that π_{obc}'s worst-case bound is $2w\sqrt{2CT}$ for the single cache setup discussed here.

[17]This happens, e.g., when the pessimistic expert achieves a hit on the steps $\lceil\alpha T\rceil$.

TABLE II
ONLINE CACHING POLICIES WITH *ADVERSARIAL* GUARANTEES: A SUMMARY OF THE CONTRIBUTIONS AND COMPARISON WITH LITERATURE.

Algorithm	Model and Conditions	Guarantees ($R_T, V_T \leq$)		Adaptive Learning
		Best case	Worst case	
1 (π_{obc})	• Bipartite network • Coded files • Predictions	0	$O\left(\sqrt{T}\right)$	✓
2 (π_{ec})	• Bipartite • Coded files • Predictions • Budget constr.	$O\left(\frac{\kappa}{2}\sqrt{T}\right), O\left(\frac{\kappa}{2}T^{\frac{3}{4}}\right)$	$O\left(\kappa\sqrt{T}\right), O\left(\kappa T^{\frac{3}{4}}\right)$	✓
3 (π_{xc})	• Bipartite network • Coded files • $P \geq 1$ predictors	$A_P \triangleq \min_{p\in\mathcal{P}+}\left\{R_T^{(p)}\right\}$	$2w\sqrt{(P+1)T}+A_P$	✓
[21]	• Single cache • Coded & uncoded files	$O\left(\sqrt{T}\right)$		–
[19]	• Bipartite network • Coded files	$O\left(\sqrt{T}\right)$		–
[22]	• Bipartite network • Coded (single cache) & uncoded	$O\left(\sqrt{T}\right)$		–
[20]	• General graph network • Coded & uncoded files	$O\left(\sqrt{T}\right)$		–
[23]	• Bipartite network • Coded & uncoded files	$O\left(\sqrt{T}\right)$		–

meta caching-routing policy:

$$\boldsymbol{z}_{t+1} = \sum_{p\in\mathcal{P}+} u_{t+1}^{(p)}\, \boldsymbol{z}_{t+1}^{(p)}, \qquad \boldsymbol{u}_{t+1} \in \Delta_{\mathcal{P}+} \tag{34}$$

remains a valid one (i.e., $\boldsymbol{z}_t \in \mathcal{X}, \forall t$). Therefore, indeed, the ideas proposed in this section can be readily applied to bipartite caching networks.

Now that we have presented all of our algorithms, let us summarize in Table II their main features and revisit how they compare with the state-of-the-art results. For Alg. 1–2, the best case refers to the scenario where the request predictions are perfect $\widetilde{\boldsymbol{c}}_t = \boldsymbol{c}_t, \forall t$; and the worst case to the scenario where predictions are furthest from the truth $\widetilde{\boldsymbol{c}}_t = \arg\max_c \|\boldsymbol{c} - \boldsymbol{c}_t\|, \forall t$. The dependence of the constant factors of the regret bound, denoted with κ to facilitate presentation, was made explicit for Alg. 2 where we saw that these constants shrink with the predictions' accuracy; and the same holds for Alg. 1. For Alg. 3, the best case refers to the scenario where the experts' predictions are perfect $\widetilde{\boldsymbol{F}}_t = \boldsymbol{F}_t, \forall t$; while the worst case arises when $\widetilde{\boldsymbol{F}}_t = \arg\max_{\boldsymbol{F}} \|\boldsymbol{F} - \boldsymbol{F}_t\|, \forall t$. Algorithms that do not leverage predictions in regret analysis (all prior work in caching[18]) have the best and worst case columns merged. We also distinguish between adaptive and static learning rates. While some prior works do employ time-adaptive learning (dynamic steps), as e.g., in [19], none of them adapts the rates (or, equivalently the regularization) to the observed gradients $\{\boldsymbol{c}_t\}_t$ and/or prediction errors, as we propose here, but instead use the Lipschitz constant w, where $\|\boldsymbol{c}_t\| \leq w, \forall t$. This leads to looser bounds in most practical cases [63] and, of course, does not allow to benefit from the availability of predictions.

VI. PERFORMANCE EVALUATION

We evaluate π_{obc}, π_{oec} and π_{xc} under different request patterns and predictions modes; and we benchmark them against $x^\star$ and the OGD policy [19] that outperforms other state-of-the-art policies [58], [59]. Note that the OGD policy has been shown to match the theoretical lower bound on the regret [19, Thm. 1]. In other words, it achieves (up to constant factors) as small regret as theoretically possible under adversarial settings and *without* predictions. Hence, it serves as the main competitor in our experiments. We observe that when reasonable predictions are available, the proposed policies have an advantage, and under noisy predictions, they still reduce the regret at the same rate with OGD, as proven in the Theorems. First, we compare π_{obc} and π_{xc} against OGD [19] in the single cache case. We then study π_{obc} for the bipartite model and π_{oec} with the presence of budget constraints. We consider three requests scenarios, stationary Zipf requests (with parameter $\zeta = 1.1$) and two actual request traces: YouTube (YT) [53] and MovieLens (ML) [54]. For predictions, we assume that at each time step, the user follows the recommendation with probability ρ (unknown to the caching system), and we experiment with different ρ values. The full codebase for the proposed policies and experiments is available via GitHub [73].

Single Cache Scenarios: We set $w = 1$ to study the cache hit rate scenario, use a library size of $N = 10^4$ files, and cache capacity of $C = 100$ files. Fig. 3(a)–(c) depict the attained average utility, $\frac{1}{t}\sum_{i=1}^{t} f_i(\boldsymbol{x}_i)$, for each policy and the Best in Hindsight (BHS) cache configuration *until that slot*, i.e., we find the best in hindsight[19] for each t. Note that BHS always achieves utility 1 initially (first requests to fill the cache). Thus, we cut the y-axis for better presentation in Fig. 3(b), (c). It can be seen that the accurate predictions (i.e., when users follow the recommendations 70% of the time) push the performance of our online caching towards BHS. For example, in Fig. 3(b), the utility gap is at most 21% after $t = 6\,k$. At the same time, even when users do not follow the recommendation at all, we still maintain a diminishing regret; the gap in Fig. 3(c) goes from 81.1% at $t = 1\,k$, to 26.3% at $t = 10\,k$. In Fig. 4, we study the effect of increasing the cache size. Expectedly, the regret increases since the diameter of the decision set also increases. However, with higher prediction quality, this effect is minimized since the regret is proportional to a shrinking error term. We also plot $15\sqrt{T}/T$ which is proportional to the average theoretical regret bound (Recall that $R_T \propto \sqrt{CT}$), and remark that the average regret of the proposed algorithms decays at a similar rate.

[18]We note the exception of [37] which considers *ML advice* in the paging problem (single cache, uncoded), but its bounds are defined w.r.t. the cache size and quantified in terms of competitive ratio – a different metric than regret, see discussion in [38].

[19]Unlike [19] that finds $x^\star$ for $t = T$, we find a $x_t^\star$ for each t. Thus, the gap among any policy and BHS is the *evolving* average regret R_t/t.

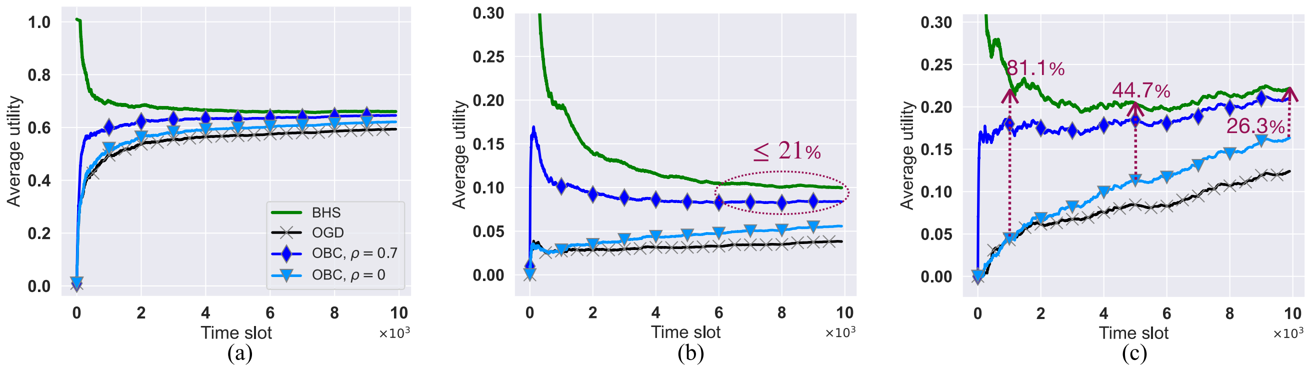

Fig. 3. Utility in the single cache model with one RecSys of different recommendation quality levels (i.e., ρ) in (a) Zipf requests with $\zeta = 1.1$, (b) YouTube request traces, (c) MovieLens request traces.

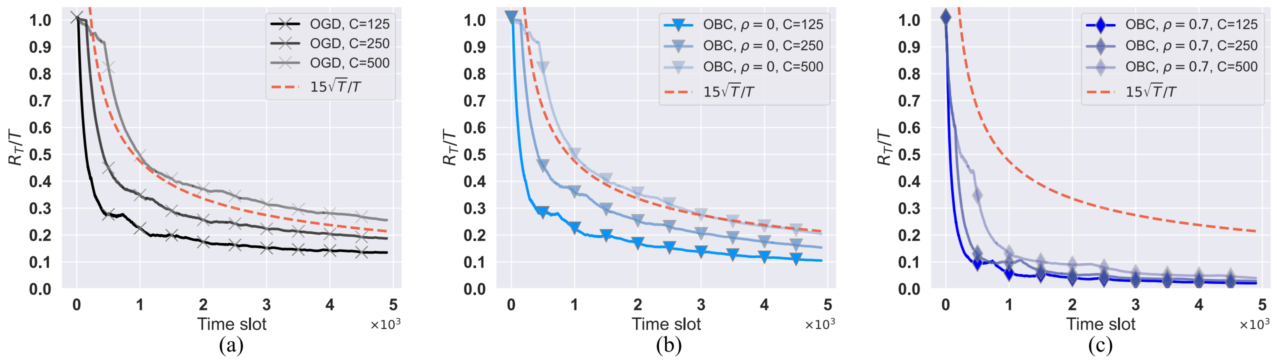

Fig. 4. Regret over time in the single cache model with different values of the cache capacity for (a) π_{ogd}, (b) π_{obc} with $\rho = 0$, (c) π_{obc} with $\rho = 0.7$.

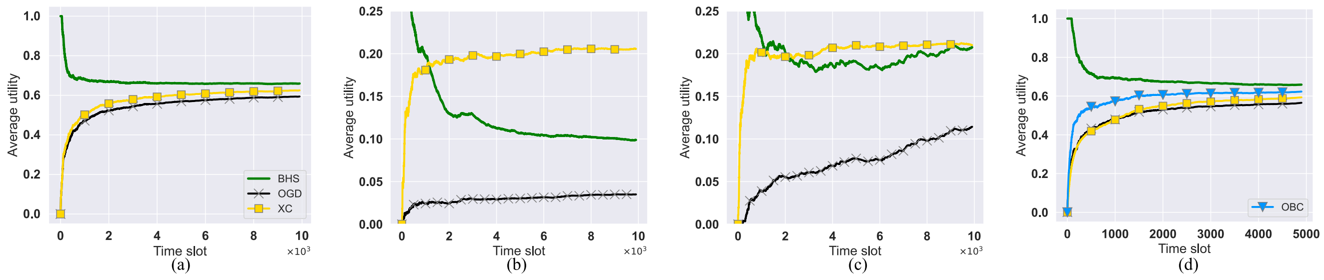

Fig. 5. Utility in the single cache model with two RecSys of recommendation qualities $\rho = 2\%$ and $\rho = 20\%$, each modeled as an expert within XC, in (a) Zipf requests with $\zeta = 1.1$, (b) YouTube request traces, (c) MovieLens request traces. (d) A comparison between π_{obc} and π_{xc} using one RecSys of an alternating recommendation quality.

For multiple predictors, we use π_{xc}. In Fig. 5(a)–(c), we evaluate π_{xc} with 3 experts: an FTRL expert, and two other optimistic experts. The first optimistic expert is endowed with a predictor (e.g., a recommendation system) that gets followed with probability $\rho = 2\%$. For the other it is $\rho = 20\%$. As shown in the plots, π_{xc} achieves negative regret on the traces (it outperforms the BHS policy) and converges to the performance of the best expert (0.20 utility). This is because in more spread distributions and real request traces, predicting the next request provides a great advantage for policies that modify the cache online over the fixed BHS. In the stationary Zipf request pattern, the optimal cache is for the files with top probabilities, which are easily captured by BHS. Thus, BHS policy performs the best. In Fig. 5(d) we show the advantages of π_{obc} compared to π_{xc} *with two experts:* an FTRL expert and a recommendation-based expert. ρ alternates between 100% and 0% (i.e., requesting the file recommended at a time step t, and any other file at $t+1$, and so on). Here, π_{obc} outperforms π_{xc} since the alternating prediction accuracy induces frequent switching between the two experts in π_{xc}: the performance of the optimistic expert alternate between 0 and 1, while that of pessimistic expert is in the range $(0.55, 0.65)$. Hence, π_{xc} is inclined to place some weight on the prediction expert at one step, only to retract and suffer a greater loss at the following one had it stayed with the full weight on

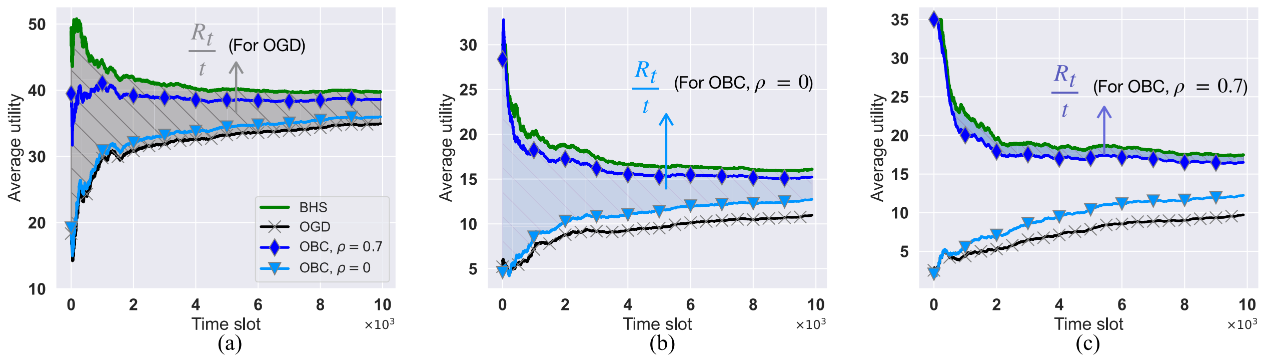

Fig. 6. Attained utility in the bipartite model under different recommendation quality levels in (a) Zipf requests with $\zeta = 1.1$, (b) YouTube request traces, (c) MovieLens request trace.

the FTRL expert. Due to the additional regret caused by such frequent switching, π_{obc}'s regret is 54.8% of π_{xc}'s. π_{obc} also achieves 38.2% of π_{ogd}'s regret. It is noteworthy that the optimal *dynamic* policy would achieve a utility of 1 across all time steps. This is because, unlike the BHS policy, it is allowed to change at every time step and can thus simply cache every file before it gets requested (recall that the optimal dynamic caching policy is defined as $\boldsymbol{x}_t^\star = \arg\max_{\boldsymbol{x}\in\mathcal{X}} f_t(\boldsymbol{x})$). However, as mentioned earlier when introducing the benchmark, it is unclear whether such a policy is desirable since it comes with a high switching cost. Furthermore, under the assumptions in this paper, no algorithm can find such a policy beforehand. *Bipartite Networks.* We consider next a bipartite graph with 3 caches and 4 user locations, where the first two locations are connected with caches 1 and 2, and the rest are connected to caches 2 and 3. The utility vector is $w_n = (1, 2, 100), \forall i, j$, thus an efficient policy places popular files on cache 3. This is the setup used in [19] that we adopt here to make a fair comparison. For the zipf scenario, we consider a library of $N = 1000$ files and $C = 100$. For the traces scenario, files with at least 10 requests are considered, forming a library of $N = 456$ files for the YouTube dataset, and we set $C = 50$, and $N = 1152$ for the ML dataset, and we increase $C = 100$. The location of each request is selected uniformly at random. Similar to the single-cache case, we plot the average utility of the online policies and the best static configuration *until each* t. Recall that the area between a policy and BHS is the average regret of that policy. To avoid clutter, we shade this area for OGD in the first sub-figure, as an example, and for OBC in the next two.

In Fig. 6(a), the effect of good predictions is evident as OBC maintains utility within 5.5% of BHS's utility after $t = 2.5\,k$. Even when the recommendations are not followed, OBC preserves the sublinear regret, achieving a gap of 30.4% and 8.5% for $t = 1\,k$ and $t = 10\,k$, respectively. Akin patterns appear in the traces scenarios. Namely the similarity between OBC with good predictions and BHS, and the improvement in OBC utility despite the recommendations quality. We also note lower utility across all policies due to the more spread requests. Note that under this bipartite model, the optimal dynamic policy, $\boldsymbol{x}_t^\star$, would achieve a utility of $\exp f_t(\boldsymbol{x}_t^\star) = {}^4/_4 + {}^{200}/_4 = 51$. This is because the requests appear uniformly randomly at one of the four users' locations, and the first two locations can always be served by cache 2 (utility 2), whereas the other two locations can be served by cache 3 (utility 100).

Next, we consider the case of budget constraint and evaluate π_{oec} for Zipf requests in Fig. 7(a), and the traces in Fig. 7(b), (c). The prices at each t are generated uniformly at random in the normalized range $[0, 1]$, and the available budget is generated randomly $b_t = \mathcal{N}(0.5, 0.05) \times 10$ i.e., enough for approximately 10 files. Such tight budgets magnify the role of dual variables and allow testing the constraint satisfaction. The benchmark $\boldsymbol{x}^\star$ is computed once for the *full time horizon*, and its utility is plotted for each t. In both scenarios, we note the constraint violation for all policies is similar, fluctuating during the first few slots and then stabilizing at zero. Hence, we plot it for one case.

Concluding, we find that π_{oec} can even outperform the benchmark. This is because the actual request patterns in the traces are not actually adversarial. Also, unlike the benchmark policy, π_{oec} is allowed to violate the budget at some time slots, provided that the constraints are eventually satisfied, which occurs either due to strict satisfaction or due to having an ample subsidy at some slots. For example, in the first scenario (Fig. 7(a)), the good predictions enable OEC to outperform $x^\star$ by 42.2% after observing all requests ($T = 5\,K$). OGD, and OEC with noisy predictions attain utility units improvement of 16.1%, 39.3%, respectively, over the BHS. We note in Fig. 7(b), (c) the delay in learning due to initially over-satisfied budget constrain. This is a transient effect as the dual variables approach their optimal value. Eventually, the bounds on regret are satisfied. We stress that the algorithms scale for very large libraries $\mathcal{N}$; the only bottleneck in the simulations is finding $x^\star$, which involves the horizon T; this is not required in real systems.

Computational Complexity: Note that the OFTRL update (e.g., (9)) is either a linear program in case $r_t(\cdot) = 0, \forall t$, or a quadratic program otherwise. In the former case, the solution is finding the top C most requested files. This can be done through a simple ordering operation whose worst-case complexity is linear in the dimension of the decision variable (e.g., $O(N\log(N)$ for each cache). If, however, we have regularization terms, then the resulting quadratic program can be solved in closed form in $\mathbb{R}$. Then, for the projection to the feasible set $\mathcal{X}$, we can use

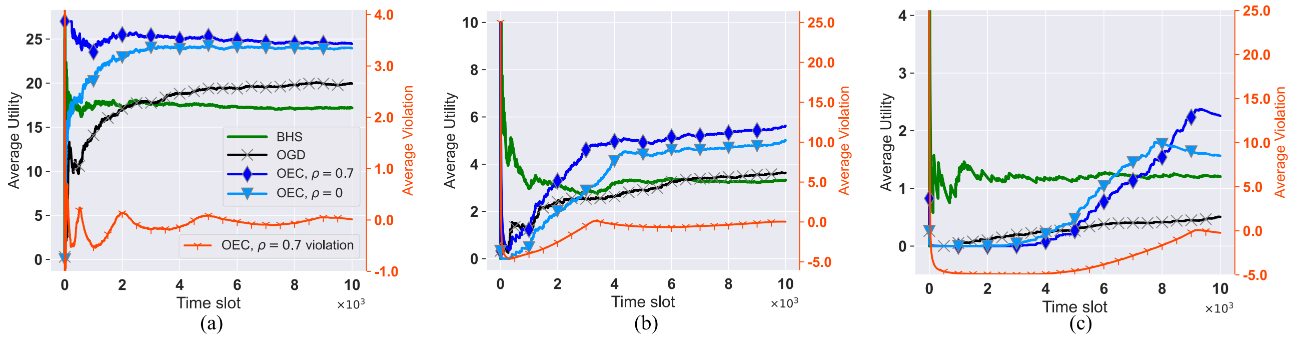

Fig. 7. Utility and budget utilization with (a) Zipf requests with $\zeta = 1.1$ and (b) YouTube traces (c) MovieLens traces.

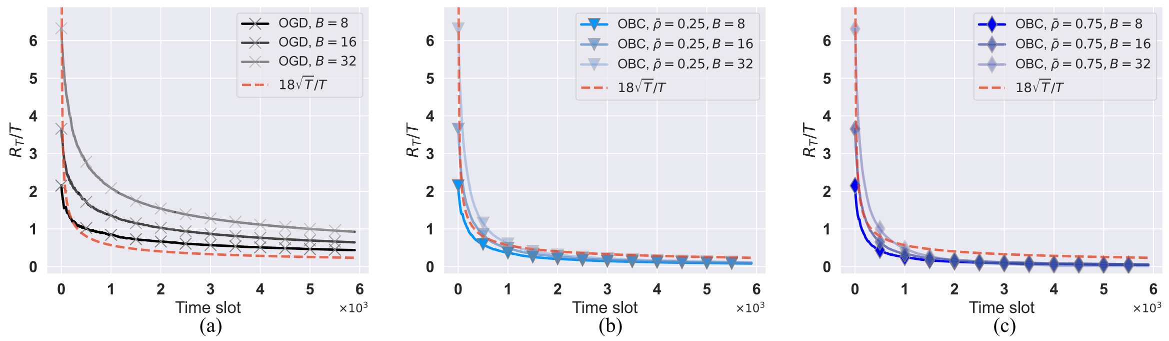

Fig. 8. Average regret over time for the single cache, generalized batched requests model for (a) π_{ogd}, (b) π_{obc} with $\bar{\rho} = 0.25$, and (c) π_{obc} with $\bar{\rho} = 0.75$.

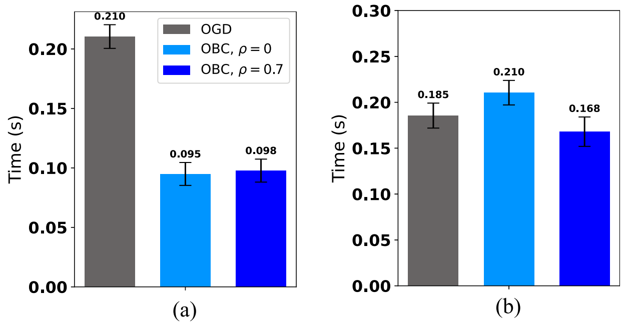

Fig. 9. Average time consumed per decision step in the bipartite network configuration and the ML Dataset ($N = 1152$, $C = 100$, $|\mathcal{I}| = 4$, $|\mathcal{J}| = 3$) with (a) pre-reserved storage (π_{obc}), and (b) elastic storage (π_{oec}).

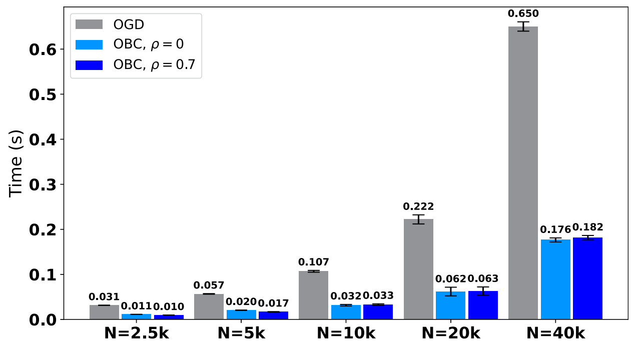

Fig. 10. Average time consumed per decision step in the single cache configuration and stationary requests with different library sizes N. The sample size is $T = 10\,k$.

specialized projection algorithms for the capped simplex (the capacity constraints for each cache) whose worst-case complexity is still polynomial in the dimension (i.e., $O(N^2)$) [74].

The above discussion refers to worst-case scenarios. In practice, we leverage the fact that we repeatedly solve similar optimization problems at each time step (the update step differs by adding one linear term and one quadratic term). Thus, the solution in a step can be used as an initial point for the following one. We note that the experimental running times are significantly less than the worst-case ones, as can be seen in Figs. 9 and 10. These simulations were performed using `CVXPY` 1.2 package with `Python` 3.10 running on an Apple M1 Pro Chip and 16 GB of RAM. The difference between OGD and the optimistic one is due to the difference in the update problem structure (lazy versus greedy projection). The chosen library sizes were selected considering that the simulations are done on a conventional PC, yet they do provide insight into the scalability, which is mostly positive as discussed earlier. Larger library sizes might necessitate custom hardware setup, or heuristic techniques such as dividing the library into sub-sets and running one of the proposed algorithms on each subset.

Batched Requests: As mentioned in the discussion of the system model, as well as that of Theorem 1, our assumption on the request model (one request per time slot) is for technical ease of analysis. The main feature of the proposed policies, which is having $R_T \propto O(\sqrt{\sum_{t=1}^{T} \|\boldsymbol{c}_t - \widetilde{\boldsymbol{c}}_t\|})$, remains valid when the request model is batched (i.e., processing B requests per time slot). However, the upper bound (not necessarily the regret) will be scaled accordingly since the diameter of the decision set will now increase. Namely, following the same steps in the proof of Theorem. 1 (after (13)), instead of $D_{\mathcal{X}} = 2(JC+1)$, we would have $D_{\mathcal{X}} = 2(JC+B)$. Fig. 8 shows experimentally how the batch size affects the regret. We also plot the line $^{18\sqrt{T}}/_{T}$ (recall that $R_T \propto G\sqrt{T}$, and in turn $G \propto B$). In this experiment, we introduce a new parameter, $\bar{\rho}$, which denotes the percentage of requests correctly predicted out of the total B. We note in these experiments that with better predictions, the effect is amortized since the term $\sqrt{\sum_{t=1}^{T} \|\boldsymbol{c}_t - \widetilde{\boldsymbol{c}}_t\|}$ remains small.

VII. Conclusion

The problem of online caching is timely with applications that extend beyond content delivery to edge computing and in fact to any dynamic placement problem with Knapsack-type constraints. This work proposes a new suite of caching policies that leverage predictions obtained from content recommendations, and possibly other forecasters, to minimize the caching regret w.r.t an ideal (yet unknown) benchmark cache configuration. As recommender systems permeate online content viewing platforms, such policies can play an essential role in optimizing caching efficacy. We identified and built upon this new connection between caching and recommender systems. The proposed algorithmic framework is scalable and robust to the quality of recommendations and the possible variations of network state and the request sequences, which can even be decided by an adversary. The achieved bounds improve upon the previously known caching regret performance, see [19], [20], [21], [22] and references therein. Finally, we believe this work opens new research directions both in terms of caching, e.g., pursuing the design of optimistic policies for uncoded caching; and in terms of resource scheduling in pertinent network and mobile computing problems using untrusted sources of optimism, i.e., predictors of unknown or varying accuracy.

References

[1] N. Mhaisen, G. Iosifidis, and D. Leith, “Online caching with optimistic learning,” 2022. [Online]. Available: https://arxiv.org/abs/2202.10590

[2] G. S. Paschos, G. Iosifidis, M. Tao, D. Towsley, and G. Caire, “The role of caching in future communication systems and networks,” *IEEE J. Select. Areas Commun.*, vol. 36, no. 6, pp. 1111–1125, Jun. 2018.

[3] L. A. Belady, “A study of replacement algorithms for a virtual-storage computer,” *IBM Syst. J.*, vol. 5, no. 2, pp. 78–101, 1966.

[4] C. Aggarwal, J. L. Wolf, and P. S. Yu, “Caching on the world wide web,” *Trans. Knowl. Data Eng.*, vol. 11, no. 1, pp. 94–107, 1999.

[5] J. Kangasharju, J. Roberts, and K. Ross, “Object replication strategies in content distribution networks,” *Comput. Commun.*, vol. 25, no. 4, pp. 376–383, 2002.

[6] K. Shanmugam, N. Golrezaei, A. G. Dimakis, A. F. Molisch, and G. Caire, “Femtocaching: Wireless content delivery through distributed caching helpers,” *IEEE Trans. Inform. Theory*, vol. 59, no. 12, pp. 8402–8413, Dec. 2013.

[7] N. Golrezaei, A. F. Molisch, A. G. Dimakis, and G. Caire, “Femtocaching and device-to-device collaboration: A. new architecture for wireless video distribution,” *IEEE Commun. Mag.*, vol. 51, no. 4, pp. 142–149, Apr. 2013.

[8] D. D. Sleator and R. E. Tarjan, “Amortized efficiency of list update and paging rules,” *Commun. ACM*, vol. 28, no. 2, pp. 202–208, 1985.

[9] P. R. Jelenković and X. Kang, “Characterizing the miss sequence of the LRU cache,” *SIGMETRICS Perform. Eval. Rev.*, vol. 36, no. 2, pp. 119–121, 2008.

[10] D. Lee et al., “On the existence of a spectrum of policies that subsumes the least recently used (LRU) and least frequently used (LFU) policies,” *SIGMETRICS Perform. Eval. Rev.*, vol. 27, no. 1, pp. 134–143, 1999.

[11] S. Traverso, M. Ahmed, M. Garetto, P. Giaccone, E. Leonardi, and S. Niccolini, “Temporal locality in today’s content caching: Why it matters and how to model it,” *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 5, pp. 5–12, 2013.

[12] F. Olmos, B. Kauffmann, A. Simonian, and Y. Carlinet, “Catalog dynamics: Impact of content publishing and perishing on the performance of a LRU cache,” in *Proc. 26th Int. Teletraffic Congr.*, 2014, pp. 1–9.

[13] M. Leconte, G. Paschos, L. Gkatzikis, M. Draief, S. Vassilaras, and S. Chouvardas, “Placing dynamic content in caches with small population,” in *Proc. IEEE 35th Annu. Int. Conf. Comput. Commun.*, 2016, pp. 1–9.

[14] S.-E. Elayoubi and J. Roberts, “Performance and cost effectiveness of caching in mobile access networks,” in *Proc. 2nd ACM Conf. Inf.-Centric Netw.*, 2015, pp. 79–88.

[15] S. O. Somuyiwa, A. György, and D. Gündüz, “A reinforcement-learning approach to proactive caching in wireless networks,” *IEEE J. Select. Areas Commun.*, vol. 36, no. 6, pp. 1331–1344, Jun. 2018.

[16] A. Sadeghi, F. Sheikholeslami, and G. B. Giannakis, “Optimal and scalable caching for 5G using reinforcement learning of space-time popularities,” *IEEE J. Select. Areas Commun.*, vol. 12, no. 1, pp. 180–190, Feb. 2018.

[17] Y. Sun, S. Chen, Z. Wang, and S. Mao, “A joint learning and game-theoretic approach to multi-dimensional resource management in fog radio access networks,” *IEEE Trans. Veh. Technol.*, vol. 72, no. 2, pp. 2550–2563, Feb. 2023.

[18] S. Geulen, B. Vöcking, and M. Winkler, “Regret minimization for online buffering problems using the weighted majority algorithm,” in *Proc. 23rd Conf. Learn. Theory*, 2010, pp. 132–143.

[19] G. S. Paschos, A. Destounis, L. Vigneri, and G. Iosifidis, “Learning to cache with no regrets,” in *Proc. IEEE Int. Conf. Comput. Commun.*, 2019, pp. 235–243.

[20] Y. Li, T. Si Salem, G. Neglia, and S. Ioannidis, “Online caching networks with adversarial guarantees,” in *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 5, no. 3, pp. 35:1–35:39, 2021.

[21] T. Si Salem, G. Neglia, and S. Ioannidis, “No-regret caching via online mirror descent,” in *Proc. Int. Conf. Commun.*, 2021, pp. 1–6.

[22] R. Bhattacharjee, S. Banerjee, and A. Sinha, “Fundamental limits on the regret of online network-caching,” in *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 4, no. 2, pp. 25:1–25:31, 2020.

[23] D. Paria and A. Sinha, “LeadCache: Regret-optimal caching in networks,” in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2021, pp. 4435–4447.

[24] Z. Song, D. S. Berger, K. Li, and W. Lloyd, “Learning relaxed belady for content distribution network caching,” in *Proc. 17th USENIX Symp. Netw. Syst. Des. Implementation*, 2020, pp. 529–544.

[25] G. Yan, J. Li, and D. Towsley, “Learning from optimal caching for content delivery,” in *Proc. 17th Int. Conf. Emerg. Netw. Exp. Technol.*, 2021, pp. 344–358.

[26] C. A. Gomez-Uribe and N. Hunt, “The Netflix recommender system: Algorithms, business value, and innovation,” *ACM Trans. Manage. Inf. Syst.*, vol. 6, no. 4, pp. 13:1–13:19, 2016.

[27] X. Amatriain, “Building industrial-scale real-world recommender systems,” in *Proc. 6th ACM Conf. Recommender Syst.*, 2012, pp. 7–8.

[28] L. E. Chatzieleftheriou, M. Karaliopoulos, and I. Koutsopoulos, “Jointly optimizing content caching and recommendations in small cell networks,” *IEEE Trans. Mobile Comput.*, vol. 18, no. 1, pp. 125–138, Jan. 2019.

[29] Y. Fu, Q. Yu, T. Q. S. Quek, and W. Wen, “Revenue maximization for content-oriented wireless caching networks (CWCNs) with repair and recommendation considerations,” *IEEE Trans. Wireless Commun.*, vol. 20, no. 1, pp. 284–298, Jan. 2021.

[30] M. Sheng, W. Teng, X. Chu, J. Li, K. Guo, and Z. Qiu, “Cooperative content replacement and recommendation in small cell networks,” *IEEE Trans. Wireless Commun.*, vol. 20, no. 3, pp. 2049–2063, Mar. 2021.

[31] T. Giannakas, P. Sermpezis, and T. Spyropoulos, “Network friendly recommendations: Optimizing for long viewing sessions,” *IEEE Trans. Mobile Comput.*, vol. 22, pp. 1633–1645, 2021.

[32] Y. Fu, Y. Zhang, A. Wong, and T. Q. Quek, “Revenue maximization: The interplay between personalized bundle recommendation and wireless content caching,” *IEEE Trans. Mobile Comput.*, vol. 22, no. 7, pp. 4253–4265, Jul. 2023.

[33] N. Garg, M. Sellathurai, V. Bhatia, B. N. Bharath, and T. Ratnarajah, "Online content popularity prediction and learning in wireless edge caching," *IEEE Trans. Commun.*, vol. 68, no. 2, pp. 1087–1100, Feb. 2020.

[34] M. Zinkevich, "Online convex programming and generalized infinitesimal gradient ascent," in *Proc. Int. Conf. Mach. Learn.*, 2003, pp. 928–936.

[35] E. Hazan, "Introduction to online convex optimization," 2019. [Online]. Available: https://arxiv.org/abs/1909.05207

[36] A. Antoniadis, C. Coester, M. Elias, A. Polak, and B. Simon, "Online metric algorithms with untrusted predictions," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 345–355.

[37] T. Lykouris and S. Vassilvtiskii, "Competitive caching with machine learned advice," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 3302–3311.

[38] L. Andrew et al., "A tale of two metrics: Simultaneous bounds on competitiveness and regret," in *Proc. 26th Annu. Conf. Learn. Theory*, 2013, pp. 741–763.

[39] O. Dekel, A. Flajolet, N. Haghtalab, and P. Jaillet, "Online learning with a hint," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 5299–5308.

[40] A. Bhaskara, A. Cutkosky, R. Kumar, and M. Purohit, "Online learning with imperfect hints," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 822–831.

[41] A. Rakhlin and K. Sridharan, "Optimization, learning, and games with predictable sequences," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2013, pp. 3066–3074.

[42] M. Mohri and S. Yang, "Accelerating online convex optimization via adaptive prediction," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2016, pp. 848–856.

[43] S. Shalev-Shwartz and Y. Singer, "A primal-dual perspective of online learning algorithms," *Mach. Learn.*, vol. 69, no. 2–3, pp. 115–142, 2007.

[44] A. Sadeghi, F. Sheikholeslami, A. G. Marques, and G. B. Giannakis, "Reinforcement learning for adaptive caching with dynamic storage pricing," *IEEE J. Select. Areas Commun.*, vol. 37, no. 10, pp. 2267–2281, Oct. 2019.

[45] J. Kwak, G. Paschos, and G. Iosifidis, "Dynamic cache rental and content caching in elastic wireless CDNs," in *Proc. 16th Int. Symp. Model. Optim. Mobile, Ad Hoc, Wireless Netw.*, 2018, pp. 1–8.

[46] E. Nygren, R. K. Sitaraman, and J. Sun, "The Akamai network: A. platform for high-performance internet applications," *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 3, pp. 2–19, 2010.

[47] T. Chen, Q. Ling, and G. B. Giannakis, "An online convex optimization approach to proactive network resource allocation," *IEEE Trans. Signal Process.*, vol. 65, no. 24, pp. 6350–6364, Dec. 2017.

[48] N. Liakopoulos, A. Destounis, G. Paschos, T. Spyropoulos, and P. Mertikopoulos, "Cautious regret minimization: Online optimization with long-term budget constraints," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 3944–3952.

[49] V. Valls, G. Iosifidis, D. Leith, and L. Tassiulas, "Online convex optimization with perturbed constraints: Optimal rates against stronger benchmarks," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2020, pp. 2885–2895.

[50] X. Yi, X. Li, L. Xie, and K. H. Johansson, "Distributed online convex optimization with time-varying coupled inequality constraints," *IEEE Trans. Signal Process.*, vol. 68, pp. 731–746, 2020.

[51] Z. Shen, K. Yang, Z. Xi, J. Zou, and W. Du, "DeepAPP: A deep reinforcement learning framework for mobile application usage prediction," *IEEE Trans. Mobile Comput.*, vol. 22, no. 2, pp. 824–840, Feb. 2023.

[52] J. B. Schafer, J. A. Konstan, and J. Riedl, "Meta-recommendation systems: User-controlled integration of diverse recommendations," in *Proc. ACM Int. Conf. Inf. Knowl. Manage.*, 2002, pp. 43–51.

[53] M. Zink, K. Suh, Y. Gu, and J. Kurose, "Characteristics of YouTube network traffic at a campus network - measurements, models, and implications," *Comput. Netw.*, vol. 53, no. 4, pp. 501–514, 2009.

[54] F. M. Harper and J. A. Konstan, "The MovieLens datasets: History and context," *ACM Trans. Interact. Intell. Syst.*, vol. 5, no. 4, pp. 19:1–19:19, 2015.

[55] G. Paschos, G. Iosifidis, and G. Caire, "Cache optimization models and algorithms," *FnT Commun. Inf. Theory*, vol. 16, no. 3/4, pp. 156–345, 2020.

[56] G. Paschos, E. Bastug, I. Land, G. Caire, and M. Debbah, "Wireless caching: Technical misconceptions and business barriers," *IEEE Commun. Mag.*, vol. 54, no. 8, pp. 16–22, Aug. 2016.

[57] X. Huang, S. Zhao, X. Gao, Z. Shao, H. Qian, and Y. Yang, "Online User-AP association with predictive scheduling in wireless caching networks," *IEEE Trans. Mobile Comput.*, 2020, pp. 1–7.

[58] A. Giovanidis and A. Avranas, "Spatial multi-LRU caching for wireless networks with coverage overlaps," *SIGMETRICS Perform. Eval. Rev.*, vol. 44, no. 1, pp. 403–405, 2016.

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

[60] D. Tsigkari and T. Spyropoulos, "User-centric optimization of caching and recommendations in edge cache networks," in *Proc. IEEE 21st Int. Symp. "World Wireless, Mobile Multimedia Netw"*, 2020, pp. 244–253.

[61] T. Si Salem, G. Neglia, and S. Ioannidis, "No-regret caching via online mirror descent," 2021. [Online]. Available: https://arxiv.org/abs/2101.12588

[62] L. Zhang, T. Yang, R. Jin, and Z.-H. Zhou, "Dynamic regret of strongly adaptive methods," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 5877–5886.

[63] H. B. McMahan, "A survey of algorithms and analysis for adaptive online learning," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 3117–3166, 2017.

[64] A. Beck, *First-Order Methods in Optimization*. Philadelphia, PA, USA: SIAM, 2017.

[65] F. Orabona, "A modern introduction to online learning," 2019. [Online]. Available: https://arxiv.org/abs/1912.13213

[66] P. Auer, N. Cesa-Bianchi, and C. Gentile, "Adaptive and self-confident on-line learning algorithms," *J. Comput. Syst. Sci.*, vol. 64, no. 1, pp. 48–75, 2002.

[67] G. S. Paschos, A. Destounis, and G. Iosifidis, "Online convex optimization for caching networks," *IEEE/ACM Trans. Netw.*, vol. 28, no. 2, pp. 625–638, Apr. 2020.

[68] "Amazon elastic CDN service - ElastiCache," Accessed: Nov. 10, 2022. [Online]. Available: https://aws.amazon.com/elasticache/

[69] Juniper Networks, "The elastic CDN solution" Solution Brief, Dec. 2014. [Online]. Available: https://www.juniper.net/assets/kr/kr/local/pdf/solutionbriefs/3510532-en.pdf

[70] S. Mannor, J. N. Tsitsiklis, and J. Y. Yu, "Online learning with sample path constraints," *J. Mach. Learn. Res.*, vol. 10, pp. 569–590, 2009.

[71] D. Anderson and D. J. Leith, "Learning the best expert efficiently," 2019. [Online]. Available: https://arxiv.org/abs/1911.04307

[72] Y. Zhang et al., "How to retrain recommender system? A sequential meta-learning method," in *Proc. 43rd Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2020, pp. 1479–1488.

[73] N. Mhaisen, "online-caching," Apr. 20, 2022. [Online]. Available: https://github.com/Naram-m/online-caching

[74] W. Wang and C. Lu, "Projection onto the capped simplex," 2015. [Online]. Available: https://arxiv.org/abs/1503.01002