LeadCache: Regret-Optimal Caching in Networks

Anonymous Author(s)
Affiliation
Address
email

Abstract

We consider a set-valued online prediction problem in the context of network caching. Assume that users are connected to a number of caches via a bipartite network. At any time slot, each user requests some file chosen from a large catalog. A user's request is met if the requested file is cached in at least one of the caches connected to the user. The objective is to predict and optimally store the files on the caches to maximize the total number of cache-hits. We propose LeadCache - an online caching policy based on the Follow-the-Perturbed-Leader paradigm. We show that the policy is regret-optimal up to a factor of $\tilde{O}(n^{3/8})$, where n is the number of users. We implement the policy by designing a new linear-time Pipage rounding algorithm. With an additional Strong-Law-type assumption, we show that the total number of file fetches under LeadCache remains almost surely finite. Additionally, we derive a tight regret lower bound using results from graph coloring. Our conclusion is that the proposed learning-based caching policy decisively outperforms the classical policies both theoretically and empirically.

1 Introduction

We consider an online structured learning problem, called Bipartite Caching, which lies at the heart of many large-scale internet applications, including Content Distribution Networks (CDN) and Cloud Computing. Formally, a set $\mathcal{I}$ of n users is connected to a set $\mathcal{J}$ of m caches via a bipartite network $G(\mathcal{I} \uplus \mathcal{J}, E)$. Each cache is connected to at most d users, and each user is connected to at most Δ caches (see Figure 1 (b)). There is a catalog consisting of N unique files, and each of the m caches can host at most C files at a time where $C < N$. The system evolves on a slotted timescale. Each of the n users may request any file from the catalog at each time slot. The file request sequence could be dictated by an adversary. Given the storage capacity constraints, an online caching policy determines which files are to be cached on which caches at each slot before the file requests for that slot arrive. The objective is to maximize the total number of *hits* by the unknown incoming requests by coordinating the caching decisions among multiple caches in an online fashion. Technically, the Bipartite Caching problem is a strict generalization of the online ***k*-sets** problem, which has been studied extensively in the learning literature [Koolen et al., 2010, Cohen and Hazan, 2015]. However, unlike the ***k*-sets** problem, which predicts a single subset at a time, in this problem, we are interested in sequentially predicting multiple subsets, each corresponding to one of the caches. The interaction among the caches through the non-linear reward function makes this problem challenging.

The Bipartite Caching problem can be thought of as a simplified abstraction of the more general Network Caching problem, which is central to the commercial CDNs, such as Akamai [Nygren et al., 2010], Amazon Web Services (AWS), and Microsoft Azure [Paschos et al., 2020]. Here one is provided with an arbitrary graph $\mathcal{G}(V, E)$, a set of users $\mathcal{I} \subseteq V$, and a set of caches $\mathcal{J} \subseteq V$. A user can retrieve a file from a cache only if the cache hosts the requested file. If the ith user retrieves the requested file from the jth cache, the user receives a reward of $r_{ij} \geq 0$ for that slot. If the requested file is not hosted in any of the caches reachable to the user, the user receives zero rewards for that slot.

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

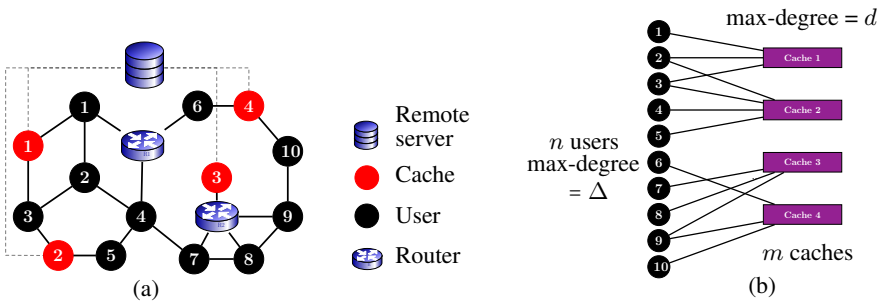

Figure 1: Reduction of the Network Caching problem (a) to the Bipartite Caching problem (b). In this schematic, we assumed that a cache, located within two hops, is reachable to a user.

The goal of a network caching policy is to dynamically place files on the caches so that cumulative rewards obtained by all users is maximized. Clearly, the Network Caching problem reduces to the Bipartite Caching problem when the rewards are restricted to the set $\{0,1\}$. It turns out that the algorithms presented in this paper can be extended to the general Network Caching problem as well.

1.1 Problem Formulation

Let us denote the file requested by the ith user by the one-hot encoded N-dimensional vector $\boldsymbol{x}_t^i$. In other words, $\boldsymbol{x}_{tf}^i = 1$ if the ith user requests the file $f \in [N]$ at slot t, and is zero otherwise. Since a user may request at most one file per slot, we have: $\sum_{f=1}^N \boldsymbol{x}_{tf}^i \leq 1, \quad \forall i \in \mathcal{I}, \forall t$. An online caching policy decides the file placements on the caches at slot t based on the information available so far. Unlike the classical caching algorithms, such as LRU, LFU, FIFO, Marker, which fetches a file immediately upon a cache-miss, we do not enforce this constraint, as this leads to linear regret [Bhattacharjee et al., 2020]. The set of files placed on the jth cache at time t is represented by the N-dimensional incidence vector $\boldsymbol{y}_t^j \in \{0,1\}^N$. In other words, $\boldsymbol{y}_{tf}^j = 1$ if the jth cache hosts the file $f \in [N]$ at time t, and is zero otherwise. Due to the limited cache capacity, the following constraint must be satisfied at each time slot t: $\sum_{f=1}^N \boldsymbol{y}_{tf}^j \leq C, \quad \forall j \in \mathcal{J}$.

The set of all admissible caching configurations, denoted by $\mathcal{Y} \subseteq \{0,1\}^{Nm}$, is dictated by the capacity constraint. Hence, in principle, the policy is allowed to replace all elements of the caches at every slot, incurring a potentially huge fetching/downloading cost over an interval. However, in Section 3, we show that the rate of the file fetchings to the caches under the proposed `LeadCache` policy goes to zero almost surely under very mild assumptions on the request sequence.

The ith user gets a *cache-hit* at slot t if and only if *anyone* of the caches connected to the ith user hosts the requested file at slot t. In the case of a cache-hit, the user obtains a unit reward. On the other hand, in the case of a *cache-miss*, the user does not obtain any reward for that slot. Hence, for a given request vector $\boldsymbol{x}_t = \left(\boldsymbol{x}_t^i, i \in \mathcal{I}\right)$ from all users and the cache configuration vector $\boldsymbol{y}_t = \left(\boldsymbol{y}_t^j, j \in \mathcal{J}\right)$ of all caches, the total reward $q(\boldsymbol{x}_t, \boldsymbol{y}_t)$ obtained by the users at time t is

$$q(\boldsymbol{x}_t, \boldsymbol{y}_t) \equiv \sum_{i \in \mathcal{I}} \boldsymbol{x}_t^i \cdot \min\left\{\mathbf{1}_{N\times 1}, \Big(\sum_{j \in \partial^+(i)} \boldsymbol{y}_t^j\Big)\right\}, \tag{1}$$

where $\boldsymbol{a} \cdot \boldsymbol{b}$ denotes the inner-product of the vectors $\boldsymbol{a}$ and $\boldsymbol{b}$, $\mathbf{1}_{N\times 1}$ denotes the N-dimensional all-one column vector, the set $\partial^+(i)$ denotes the set of all caches connected to the ith user, and the "$\min$" operator is applied component wise. The total reward $Q(T)$ accrued in a time-horizon of length T is obtained by summing-up the slot-wise rewards, *i.e.,* $Q(T) = \sum_{t=1}^T q(\boldsymbol{x}_t, \boldsymbol{y}_t)$. Following the standard practice in the online learning literature, we measure the performance of any online policy π using the notion of static regret $R^\pi(T)$, defined as the maximum difference in the cumulative rewards obtained by the optimal fixed caching-configuration in the hindsight and that of the online policy π, *i.e.,*

$$R^\pi(T) \overset{\text{(def.)}}{=} \sup_{\{\boldsymbol{x}_t\}_{t=1}^T} \left(\sum_{t=1}^T q(\boldsymbol{x}_t, \boldsymbol{y}^*) - \sum_{t=1}^T q(\boldsymbol{x}_t, \boldsymbol{y}_t^\pi)\right), \tag{2}$$

where $\boldsymbol{y}^*$ is the best static cache-configuration in the *hindsight* for the file request sequence $\{\boldsymbol{x}_t\}_{t=1}^T$, *i.e.,* $\boldsymbol{y}^* = \arg\max_{\boldsymbol{y} \in \mathcal{Y}} \sum_{t=1}^T q(\boldsymbol{x}_t, \boldsymbol{y})$. We assume that the file request sequence is generated by an

oblivious adversary, *i.e.,* the entire request sequence $\{\boldsymbol{x}_t\}_{t\geq 1}$ is assumed to be fixed a priori. This is a mild assumption, because it is known from Cesa-Bianchi and Lugosi [2006] and Appendix B. of Suggala and Netrapalli [2020] that any algorithm that works for oblivious adversary also works for non-oblivious adversary, whose current actions may depend on past predictions of the algorithm. Note that the problem is *non-convex*, as we seek for integral cache allocations.

2 Background and Related Works

A canonical online learning model is Online Linear Optimization (OLO), which can be thought of as a repeated game played between a policy (also known as the forecaster) and an adversary [Cesa-Bianchi and Lugosi, 2006]. In this model, At every time slot t, the policy chooses an action vector $\boldsymbol{y}_t$ from a feasible action set $\mathcal{Y} \subseteq \mathbb{R}^d$. After that, the adversary reveals a reward vector $\boldsymbol{x}_t$ from a set $\mathcal{X} \subseteq \mathbb{R}^d$. The adversary is assumed to be *oblivious* in the sense that the reward vectors are determined before the game starts. At slot t, the policy receives a scalar reward given by the inner-product $q(\boldsymbol{x}_t, \boldsymbol{y}_t) := \langle \boldsymbol{x}_t, \boldsymbol{y}_t \rangle$. A classic objective is to design a policy with minimal regret (viz. Eqn. (2)). Follow the Perturbed Leader (FTPL), is a well-known online policy for the OLO problem [Hannan, 1957]. At time slot t, the policy adds a random noise vector $\boldsymbol{\gamma}_t$ to the cumulative reward vector $\boldsymbol{X}_t = \sum_{\tau=1}^{t-1} \boldsymbol{x}_t$, and selects the action $\boldsymbol{y}_t = \arg\max_{\boldsymbol{y}\in\mathcal{Y}} \langle \boldsymbol{X}_t + \boldsymbol{\gamma}_t, \boldsymbol{y} \rangle$. The regret bound of the FTPL policy depends on the structure of the problem (*i.e.,* the sets $\mathcal{X}$ and $\mathcal{Y}$) and the distribution of the additive noise vector $\boldsymbol{\gamma}_t$ (*e.g.,* Gumbel [Cesa-Bianchi and Lugosi, 2006], Exponential [Kalai and Vempala, 2005], Gaussian [Cohen and Hazan, 2015, Devroye et al., 2015]). See Abernethy et al. [2016] for a unifying treatment of the FTPL policies through the lens of stochastic smoothing. Moreover, in addition to static regret, stronger forms of regret, such as strongly adaptive regret, have also been investigated in the literature. There now exist efficient Black-Box reductions that transform any standard low-regret algorithm to a strongly adaptive one [Daniely et al., 2015, Jun et al., 2017]. Hence, in this paper, we only consider the standard regret minimization problem.

A large number of papers on caching proceeds by assuming some stochastic model for the file request sequence, *e.g.,* Independent Reference Model (IRM) and Shot Noise Model (SNM) Traverso et al. [2013]. For adversarial request sequences, classic algorithms such as MIN, LRU, LFU, FIFO aim to minimize the competitive ratio [Van Roy, 2007, Lee et al., 1999, Dan and Towsley, 1990]. However, since the competitive ratio metric is multiplicative in nature, there can be a large performance gap in the hit-ratio of a competitively optimal policy and the optimal offline policy. To provide a robust *additive* performance guarantee, the caching problem has been recently investigated through the lens of regret minimization. Recall that, in the online ***k*-sets** problem, which arises in the context of online ad display and personalized news recommendation, the goal of the learner is to predict a subset of cardinality k to maximize the hits [Cohen and Hazan, 2015]. Clearly, the single cache problem is identical to the ***k*-sets** problem. Daniely and Mansour [2019] considered the problem of minimizing the regret plus the switching cost for a single cache. The authors proposed a variant of the celebrated exponential weight algorithm [Littlestone and Warmuth, 1994, Freund and Schapire, 1997] that ensures the minimum competitive ratio and a small but sub-optimal regret. Moreover, in the adversarial setting, the switching cost of their proposed policy grows as the square-root the length of the time-horizon. The Bipartite Caching model was first proposed in a pioneering paper by Shanmugam et al. [2013], where they considered a stochastic version of the problem with known popularity distribution of the files. Paschos et al. [2019] proposed an Online Gradient Ascent (OGA)-based Bipartite Caching policy which allows caching a fraction of the Maximum Distance Separable (MDS)-coded files. Closely related to this paper is the recent work by Bhattacharjee et al. [2020], where the authors designed a regret-optimal single-cache policy and a Bipartite Caching policy for fountain-coded files. However, the fundamental problem of designing a regret-optimal uncoded caching policy for the Bipartite Caching problem was left open due to its difficulty.

Why standard approaches fail: A straightforward way to formulate the Bipartite Caching problem is to pose it as an instance of the `Experts` problem [Cesa-Bianchi and Lugosi, 2006] with each of the possible $\binom{N}{k}^m$ cache configurations as experts. However, this approach is computationally infeasible due to the huge number of experts. Another possible way to learn a non-linear reward function is to linearize it via the first-order Taylor's series approximation [Orabona, 2019]. However, the non-differentiability of the reward function (1) and the integrality constraint of the allocation vectors precludes this approach in this problem. In view of the above challenges, we present our main technical contributions below.

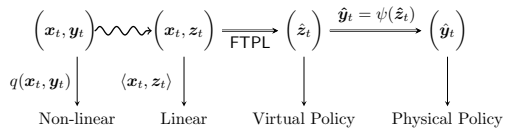

Figure 2: Reduction pipeline illustrating the translation of the Bipartite Caching problem with a non-linear reward function to an online learning problem with a linear reward function.

3 Main Results

1. FTPL for a non-linear non-convex problem: We propose `LeadCache`, a network caching policy based on the Follow the Perturbed Leader paradigm. The *non-linearity* of the reward function and the non-convexity of the feasible set (due to the integrality of cache allocations) pose a major challenge in using the generic FTPL framework [Abernethy et al., 2016]. To circumvent this difficulty, we switch to a *virtual action* domain $\mathcal{Z}$ where the reward function is linear. We use an anytime version of the FTPL policy for designing a virtual policy for the linearized virtual learning problem. Finally, we translate the virtual policy back to the original action domain $\mathcal{Y}$ with the help of a mapping $\psi : \mathcal{Z} \to \mathcal{Y}$, which is obtained by solving a combinatorial optimization problem. The overall reduction pipeline is summarized in Figure 2. Our technique is general, leads to a *dimension-free* regret bound, and may be used in other problems with a similar structure.

2. New Rounding Technique: The problem P that translates the virtual actions to the physical caching actions in the above scheme turns out to be an NP-hard Integer Linear Program. As our second independent contribution, we design a linear-time Pipage rounding technique for this problem Ageev and Sviridenko [2004]. Incidentally, our rounding technique substantially improves upon a computationally complex rounding algorithm proposed by Shanmugam et al. [2013] in the context of caching with i.i.d. requests. It is worth mentioning that improving upon the rounding algorithm by Shanmugam et al. [2013] has been recognized as one of the most outstanding challenges in large-scale network caching Paschos et al. [2018]. Our rounding algorithm runs in linear time and yields the same approximation guarantee as that of Shanmugam et al. [2013].

3. Bounding the Switching Cost: As our third contribution, we show that if the file requests are generated by a stochastic process satisfying an additional mild Strong Law-type property then the caching configuration under the `LeadCache` policy *almost surely* converges to the corresponding optimal configuration *in finite time*. As new file fetches to the cache from the remote server consumes bandwidth, this result implies that the proposed policy offers the best of both worlds - (1) sub-linear regret for adversarial requests, and (2) zero rate download for “stochastically regular" requests.

4. New Regret Lower Bound: As our final contribution, we derive a tight minimax regret lower bound for the Bipartite Caching problem. The proof of the new lower bound critically uses a fundamental result in graph coloring, known as Brook’s theorem (Diestel [2005]). Our lower bound improves upon a recent result by Bhattacharjee et al. [2020] and establishes a $\tilde{O}(n^{3/8})$ approximation guarantee for the regret of the `LeadCache` policy.

4 The `LeadCache` Policy

In this Section, we describe `LeadCache`- an efficient online network caching policy guaranteeing near-optimal regret. Since the reward function (1) is non-linear in the action variable $\boldsymbol{y}$, to use the FTPL policy, we *linearize* the problem by switching to a *virtual* action domain $\mathcal{Z}$ as detailed below.

The Virtual Caching Problem: Consider an associated Online Linear Optimization (OLO) problem, called Virtual Caching. In this problem, at each slot t, a virtual action $\boldsymbol{z}_t \equiv \left(\boldsymbol{z}_t^i, i \in \mathcal{I}\right)$ is taken in response to the file requests $\{\boldsymbol{x}_\tau\}_1^{t-1}$ received so far. The ith component of the virtual action, denoted by $\boldsymbol{z}_t^i \in \{0,1\}^N$, roughly indicates the availability of the files in the caches connected to the ith user. The set of all admissible virtual actions, denoted by $\mathcal{Z} \subseteq \{0,1\}^{N \times n}$, is given below in Eqn.

(4). The slot-wise reward $r(\boldsymbol{x}_t, \boldsymbol{z}_t)$ accrued by the virtual action $\boldsymbol{z}_t$ for the file request vector $\boldsymbol{x}_t$ at the tth slot is given by their inner product, *i.e.,*

$$r(\boldsymbol{x}_t, \boldsymbol{z}_t) := \langle \boldsymbol{x}_t, \boldsymbol{z}_t \rangle = \sum_{i \in \mathcal{I}} \boldsymbol{x}_t^i \cdot \boldsymbol{z}_t^i. \tag{3}$$

Virtual Actions: The set $\mathcal{Z}$ of all admissible virtual actions is defined as the set of all binary vectors $\boldsymbol{z} \in \{0,1\}^{N \times n}$ such that the following condition holds componentwise for some admissible physical cache configuration vector $\boldsymbol{y} \in \mathcal{Y}$:

$$\boldsymbol{z}^i \leq \min\Big\{\mathbf{1}_{N\times 1}, \Big(\sum_{j \in \partial^+(i)} \boldsymbol{y}^j\Big)\Big\}, \;\; 1 \leq i \leq n. \tag{4}$$

In other words, the set $\mathcal{Z}$ be can be explicitly characterized as the set of all binary vectors $\boldsymbol{z} \in \{0,1\}^{N \times n}$ satisfying the following constraints for some feasible $\boldsymbol{y}$:

$$z_f^i \;\; \leq \;\; \sum_{j \in \partial^+(i)} y_f^j, \;\; \forall i \in \mathcal{I}, f \in [N] \tag{5}$$

$$\sum_{f=1}^{N} y_f^j \;\; \leq \;\; C, \;\; \forall j \in \mathcal{J}, \tag{6}$$

$$y_f^j, z_f^i \;\; \in \;\; \{0,1\}, \;\; \forall i \in \mathcal{I}, \forall j \in \mathcal{J}, f \in [N]. \tag{7}$$

Let $\psi : \mathcal{Z} \to \mathcal{Y}$ be a mapping that maps any admissible virtual action $\boldsymbol{z} \in \mathcal{Z}$ to a corresponding physical caching action $\boldsymbol{y}$ satisfying the condition (4). Hence, the binary variable $z_{tf}^i = 1$ *only if* the file f is hosted in one of the caches connected to the ith user at time t in the physical configuration $\boldsymbol{y}_t = \psi(\boldsymbol{z}_t)$. The mapping ψ may be used to map any virtual caching policy $\pi^{\text{virtual}} = \{\boldsymbol{z}_t\}_{t \geq 1}$, to a physical caching policy $\pi^{\text{phy}} \equiv \psi(\pi^{\text{virtual}}) = \{\boldsymbol{y}_t\}_{t \geq 1}$ through the correspondence $\boldsymbol{y}_t = \psi(\boldsymbol{z}_t), \forall t \geq 1$. The following lemma relates the regret of these two online policies:

Lemma 1. *For any virtual caching policy $\pi^{virtual}$, define a physical caching policy $\pi^{phy} = \psi(\pi^{virtual})$. Then the regret of π^{phy} is bounded above by that of the policy $\pi^{virtual}$,* i.e., $R_T^{\pi^{phy}} \leq R_T^{\pi^{virtual}}, \forall T$.

See Section 9.1 in the supplementary material for the proof of the above lemma. Lemma 1 implies that a low-regret virtual caching policy may be used to obtain a low-regret physical caching policy using the mapping $\psi(\cdot)$. The key advantage of the virtual caching problem is that it is a standard OLO problem. In our proposed `LeadCache` policy, we use an anytime version of the FTPL policy for solving virtual caching problem. The overall `LeadCache` policy is described below:

Algorithm 1 The `LeadCache` Policy

1: $\boldsymbol{X}(0) \leftarrow \mathbf{0}$
2: Sample $\boldsymbol{\gamma} \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0, 1_{Nn\times 1})$
3: **for** $t = 1$ to T **do**
4: $\quad \boldsymbol{X}(t) \leftarrow \boldsymbol{X}(t-1) + \boldsymbol{x}_t$
5: $\quad \eta_t \leftarrow \frac{n^{3/4}}{(2d(\log \frac{N}{C}+1))^{1/4}} \sqrt{\frac{t}{Cm}}$
6: $\quad \boldsymbol{\Theta}(t) \leftarrow \boldsymbol{X}(t) + \eta_t \boldsymbol{\gamma}$
7: $\quad \boldsymbol{z}_t \leftarrow \max_{\boldsymbol{z} \in \mathcal{Z}} \langle \boldsymbol{\Theta}(t), \boldsymbol{z} \rangle$.
8: $\quad \boldsymbol{y}_t \leftarrow \psi(\boldsymbol{z}_t)$.
9: **end for**

In Algorithm 1, the flattened $Nn \times 1$ dimensional vector $\boldsymbol{X}(t)$ denotes the cumulative count of the file requests (for each (user, file) tuple), and the vector $\boldsymbol{\Theta}(t) = \big(\boldsymbol{\theta}^i(t), 1 \leq i \leq n\big)$, denotes the perturbed cumulative file request counts obtained upon adding a scaled i.i.d. Gaussian noise vector to $\boldsymbol{X}(t)$. Note that, in step 6, we can also sample a fresh Gaussian vector $\boldsymbol{\gamma}_t$ at every time, leading to a high probability regret bound [Devroye et al., 2015]. The following result gives an upper bound on regret:

Theorem 1. *The expected regret for the `LeadCache` policy is upper bounded by:*

$$\mathbb{E}\big(R_T^{\texttt{LeadCache}}\big) \leq \kappa n^{3/4} d^{1/4} \sqrt{mCT},$$

where $\kappa = O\big(\textsf{poly-log}(N/C)\big)$, and the expectation is taken over the randomness of the policy.

Note that, in sharp contrast with the generic bound of Suggala and Netrapalli [2020], our regret-bound is almost dimension-free, as it depends only logarithmically with the ambient dimension N.

Proof outline: Our analysis of the `LeadCache` policy exploits the elegant stochastic smoothing framework developed in Abernethy et al. [2016, 2014], Cohen and Hazan [2015], Lee [2018]. However, the generic regret bound given by Theorem 1.7 of Abernethy et al. [2016] is quite loose as it depends polynomially on the dimension N of the problem. Furthermore, as noted in Bhattacharjee et al. [2020], the regret bounds in Theorem 1 and Lemma 3 of Cohen and Hazan [2015] are also loose by a factor of $\Theta(\sqrt{C})$. In this paper, we derive a tight regret bound by carefully analyzing the Hessian of the stochastically smoothed potential function and exploiting the structure of the problem. See Section 9.2 of the supplementary material for the complete proof of Theorem 1.

4.1 Fast approximate implementation

The only computationally intensive steps in Algorithm 1 are (1) solving the optimization problem in step 7 to determine the virtual caching action, and (2) translating the virtual action back to the physical caching action in step (8). Since the perturbed vector $\boldsymbol{\Theta}(t)$ is obtained by adding white Gaussian noise (with unbounded support in both positive and negative real axes) to the cumulative request vector $\boldsymbol{X}(t)$, some of its components could be negative. In order to maximize the objective (7), it is clear that if the coefficient $\theta_f^i(t)$ is negative for some (i, f) tuple, it is feasible and optimal to set the corresponding virtual action variable z_f^i to zero. Hence, steps (7) and (8) may be combined as:

$$\boldsymbol{y}(t) \leftarrow \arg\max_{\boldsymbol{y}\in\mathcal{Y}} \underbrace{\sum_{i\in\mathcal{I}, f\in[N]} (\theta_f^i(t))^+ \left(\min\left(1, \sum_{j\in\partial^+(i)} y_f^j\right)\right)}_{L(\boldsymbol{y})}, \tag{8}$$

where $x^+ \equiv \max(0, x)$. Incidentally, we find that the problem (8) is mathematically identical to the uncoded *Femtocaching* problem with known file request probabilities studied in Section III of Shanmugam et al. [2013]. In Lemma 1 of Shanmugam et al. [2013], the authors showed that the problem (8) is NP-hard. They also described a complex rounding method, which involves repeatedly computing matchings in a graph. In the following, we propose an efficient *linear time* rounding technique that rounds a relaxation of (7) and enjoys the same performance guarantee.

LP Relaxation: First, we introduce a new set of variables $z_f^i := \min(1, \sum_{j\in\partial^+(i)} y_f^j), \forall i, f$, and relax the integrality constraints to arrive at the following LP:

$$\max \sum_{i,f} (\theta_f^i(t))^+ z_f^i \tag{9}$$

Subject to,

$$z_f^i \leq \sum_{j\in\partial^+(i)} y_f^j, \quad \forall i \in \mathcal{I}, f \in [N] \tag{10}$$

$$\sum_{f=1}^{N} y_f^j \leq C, \quad \forall j \in \mathcal{J}, \tag{11}$$

$$0 \leq y_f^j \leq 1, \quad \forall j \in \mathcal{J}, f \in [N] \tag{12}$$

$$0 \leq z_f^i \leq 1, \quad \forall i \in \mathcal{I}, f \in [N]. \tag{13}$$

Denote the objective function of the problem (8) by $L(\boldsymbol{y})$ and its optimal value (over integers) by OPT. Let $\boldsymbol{y}^*$ be an optimal solution to the relaxed LP (9). Since the LP is a relaxation to (8), it automatically holds that $L(\boldsymbol{y}^*) \geq \mathsf{OPT}$. To round the cache allocation vector $\boldsymbol{y}^*$ to an integral one, we use the general framework of *Pipage rounding* introduced by Ageev and Sviridenko [2004].

Pipage Rounding: Our rounding technique is markedly simpler compared to Algorithm 1 of Shanmugam et al. [2013]. While we round one fractional allocation of a single cache at a time, the rounding procedure of Shanmugam et al. [2013] jointly rounds several allocations in multiple caches at the same time by computing matchings in a bipartite graph. Our simplification stems from the fact

that Shanmugam et al. [2013] directly use the Pipage rounding procedure designed for a different template problem, which includes additional spurious constraints on the maximum number of copies a file may have on different caches at any point in time. Shanmugam et al. [2013] relax these additional constraints by setting constraint violation budgets sufficiently high. However, due to the spurious constraints, the resulting rounding procedure becomes unnecessarily complicated. Moreover, despite its high complexity, their rounding procedure is not general, as it requires the coefficients $\boldsymbol{\theta}$ of the objective function (8) to have a product form structure, which does not necessarily hold. In Algorithm 2, we design a linear-time rounding technique without making any assumption on the coefficients.

Algorithm 2 Cache-wise Pipage rounding

1: $\boldsymbol{y} \leftarrow$ Solution of the LP (9).
2: **while** $\boldsymbol{y}$ is not integral **do**
3: Select a cache j with two fractional variables $y^j_{f_1}$ and $y^j_{f_2}$.
4: Set $\epsilon_1 \leftarrow \min(y^j_{f_1}, 1 - y^j_{f_2}), \epsilon_2 \leftarrow \min(1 - y^j_{f_1}, y^j_{f_2})$.
5: Define two new feasible cache-allocation vectors $\boldsymbol{\alpha}, \boldsymbol{\beta}$ as follows:

$$\alpha^j_{f_1} \leftarrow y^j_{f_1} - \epsilon_1, \alpha^j_{f_2} \leftarrow y^j_{f_2} + \epsilon_1, \quad \text{and} \quad \alpha^k_f \leftarrow y^k_f, \text{ otherwise}, \tag{14}$$

$$\beta^j_{f_1} \leftarrow y^j_{f_1} + \epsilon_2, \beta^j_{f_2} \leftarrow y^j_{f_2} - \epsilon_2, \quad \text{and} \quad \beta^k_f \leftarrow y^k_f, \text{ otherwise}. \tag{15}$$

6: Set $\boldsymbol{y} \leftarrow \arg\max_{\boldsymbol{x} \in \{\boldsymbol{\alpha}, \boldsymbol{\beta}\}} \phi(\boldsymbol{x})$.
7: **end while**
8: **return** $\boldsymbol{y}$.

Design: The key to our deterministic rounding procedure is to consider the following surrogate objective function $\phi(\boldsymbol{y})$ instead of the original objective $L(\boldsymbol{y})$ in Eqn. (8):

$$\phi(\boldsymbol{y}) \equiv \sum_{i,f} (\theta^i_f(t))^+ \big(1 - \prod_{j \in \partial^+(i)} (1 - y^j_f)\big). \tag{16}$$

Following a standard algebraic argument (see eqn. (16) of Ageev and Sviridenko [2004]), we have:

$$L(\boldsymbol{y}) \overset{(a)}{\geq} \phi(\boldsymbol{y}) \geq \left(1 - (1 - \frac{1}{\Delta})^{\Delta}\right) L(\boldsymbol{y}), \quad \forall \boldsymbol{y} \in [0,1]^N, \tag{17}$$

where $\Delta \equiv \max_{i \in \mathcal{I}} |\partial^+(i)|$. Note that, the inequality (a) holds with equality for all binary vectors $\boldsymbol{y} \in \{0,1\}^{mN}$. Our Pipage rounding procedure, described in Algorithm 2, starts with an optimal solution of the LP (9). Then it iteratively perturbs two fractional variables (if any) in a single cache in such a way that the value of the surrogate objective function $\phi(\boldsymbol{y})$ never decreases (Steps (15)-(6)) while at least one of the two fractional variables is rounded to an integer. Step (4) ensures that the feasibility is maintained at every step of the perturbation. Upon termination (which happens within $O(mN)$ steps), the rounding procedure yields a feasible integral allocation vector $\hat{\boldsymbol{y}}$ with an objective value $L(\hat{\boldsymbol{y}})$, which is within a factor of $1 - (1 - \frac{1}{\Delta})^{\Delta}$ of the optimum objective, where $\Delta \equiv \max_{i \in \mathcal{I}} |\partial^+(i)|$. The following theorem formalizes this claim.

Theorem 2. *Algorithm 2 is an* $1 - (1 - \frac{1}{\Delta})^{\Delta}$ *approximation algorithm for solving the problem 8.*

See Section 10 of the supplementary material for the complete proof of Theorem 2.

5 Bounding the Number of Fetches

Repeatedly fetching files from the remote server to the local caches consumes bandwidth and increases the network congestion. Hence, it is imperative that the average number of new file fetches to the caches remains small over a long time interval. In the classical caching policies, such as LRU, FIFO, and LFU, a file is fetched if there is a cache-miss. Since the number of fetching is equal to the number of cache-misses for these policies, it is enough for them to minimize the cache-miss rates only. However, since the `LeadCache` policy decouples the fetching process from the cache-misses, in addition to low regret, we need to ensure that the number of file fetches remains small as well. In

this paper, we prove the surprising result that if the file request process satisfies a mild stochastic regularity property, the file fetches *stop almost surely after a finite time*. Note that the above result is of a different flavor from the long line of work that minimizes the switching regret Devroye et al. [2015], Kalai and Vempala [2005], Geulen et al. [2010]. These papers do not make any stochastic assumption but give a much weaker guarantee on the number of fetches.

A. Stochastic Regularity Assumption: Let $\{\boldsymbol{X}(t)\}_{t\geq 1}$ be the cumulative request-arrival process. We assume that, there exists a set of non-negative numbers $\{p_f^i\}_{i\in\mathcal{I}, f\in[N]}$ such that for any $\epsilon > 0$:

$$\sum_{t=1}^{\infty} \sum_{i\in\mathcal{I}, f\in[N]} \mathbb{P}\Big(\Big|\frac{\boldsymbol{X}_f^i(t)}{t} - p_f^i\Big| \geq \epsilon\Big) < \infty. \tag{18}$$

Using the first Borel-Cantelli Lemma, the regularity assumption **A** implies that the process $\{\boldsymbol{X}(t)\}_{t\geq 1}$ satisfies the strong-law: $\boldsymbol{X}_f^i(t)/t \to p_f^i$, a.s., $\forall i \in \mathcal{I}, f \in [N]$. However, the converse may not be true. Nevertheless, the assumption **A** is quite mild and holds, *e.g.*, when the file request sequence is generated by a renewal process having an inter-arrival distribution with a finite fourth moment (See Section 12 of the supplementary material for the proof). Following is our main result for this section:

Theorem 3. *Under the stochastic regularity assumption **A**, the file fetches to the caches stop after a finite time with probability* 1 *under the* `LeadCache` *policy.*

Please refer to Section 11 of the supplementary material for the proof of Theorem 3. Policies, such as LRU, LFU, FIFO, which immediately fetch every missed file, *do not* enjoy the above property.

6 A Minimax Lower Bound

We now establish a minimax lower bound on the regret achievable by any online policy for the Bipartite Caching problem. Before we state and prove our result, we first explain what makes this lower bound fundamentally new. Recall that our reward function (1) is non-linear. As such, the standard lower bounds for the OLO problems, such as Theorem 5.1 and 5.12 of Orabona [2019], Theorem 5 of Abernethy et al. [2008] are insufficient for our purpose. The first non-trivial regret lower bound for the Bipartite Caching problem was given by Bhattacharjee et al. [2020]. In the following, we strengthen this bound by utilizing results from graph coloring.

Theorem 4 (Regret Lower Bound)**.** *For a catalog of size $N \geq \max(2\frac{d^2Cm}{n}, 2mC)$ the regret of any online caching policy π is lower bounded as: $R_T^\pi \geq \max(\sqrt{\frac{mnCT}{2\pi}}, d\sqrt{\frac{mCT}{2\pi}}) - \Theta(\frac{1}{\sqrt{T}})$.*

Proof outline: We construct an i.i.d. file request sequence where all users request an identical file sampled uniformly at random. Then we tightly lower-bound the total reward accrued by the offline optimal policy. While Bhattacharjee et al. [2020] constructs an offline cache configuration with distinct files in each cache (*global exclusivity*), we construct an offline static caching configuration, where each user is guaranteed to find distinct files on each of its connected caches (*local exclusivity*). This construction effectively linearizes the reward function that can be analyzed by the probabilistic *Balls-into-Bins* framework. Using results from graph coloring theory, we find a small catalog size that guarantees the existence of such a construction. Our result improves the lower bound of Bhattacharjee et al. [2020] in the regime $d > \sqrt{n}$. See Section 13 of the supplementary material for the proof.

Approximation guarantee for `LeadCache`: Theorem 1 and Theorem 4, taken together, imply that the proposed `LeadCache` policy achieves the optimal regret up to a factor of $\tilde{O}(\min((nd)^{1/4}, (n/d)^{3/4}))$. Clearly, the worst-case factor is obtained when $d = \sqrt{n}$. Hence, irrespective of the value of d, the `LeadCache` policy is regret-optimal up to a factor of $\tilde{O}(n^{3/8})$. Furthermore, when d is a constant, the approximation ratio can be further improved to $\tilde{O}(n^{1/4})$.

7 Experiments

Baseline policies: In this section, we compare the performance of the `LeadCache` policy with other baselines on a standard dataset. In the LRU policy, each cache independently considers the set of all requested files from its connected users. In the case of a *cache-miss*, the LRU policy fetches the requested file into the cache while evicting a file from the cache that was requested *least recently*. The

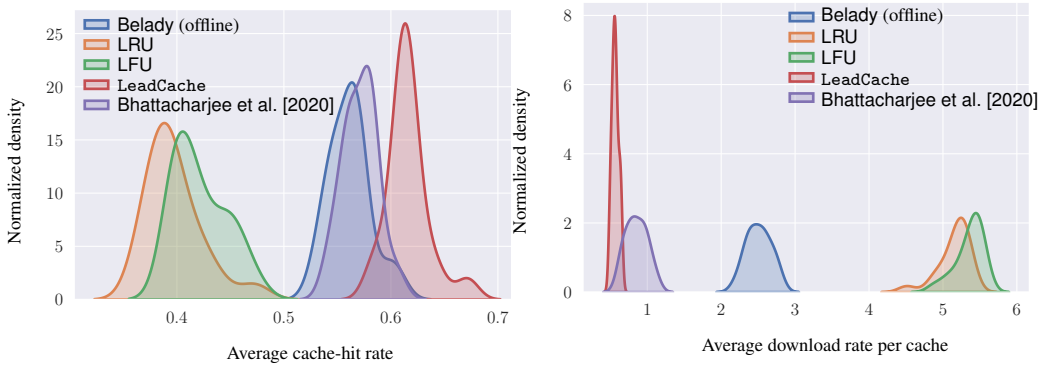

Figure 3: Empirical distributions of (a) Cache-Hit rates and (b) Fetch-rates of different caching policies.

LFU policy works similarly to the LRU policy with the exception that, in the case of a cache-miss, the policy evicts a file that was requested the *least number of times* among all files currently on the cache. Finally, for the purpose of benchmarking, we also experiment with Belady's offline MIN algorithm Aho et al. [1971], which is optimal for each *individual* cache when the entire file request sequence is known a priori. Note that Belady's algorithm is *not* optimal in the network caching setting as it does not consider the adjacency relations between the caches and the users. As a result, similar to LRU and LFU, multiple users may find the same files cached in more than one connected caches, which does not help in improving the hit-rate. To the best of our knowledge, the characterization of the optimal offline policy in the network caching setting remains an open problem. The heuristic multi-cache policy by Bhattacharjee et al. [2020] uses an FTPL strategy assuming the reward function to be linear.

Experimental Setup: In our experiments, we use a publicly available anonymized production trace from a large CDN provider available under a BSD 2-Clause License [Berger et al., 2018, Berger, 2018]. The trace data consists of three fields, namely, request number, file-id, and file size. In our experiments, we construct a Bipartite caching network with $n = 30$ users and $m = 10$ caches. Each cache is connected to $d = 8$ randomly chosen users. Thus, every user is connected to ≈ 2.67 caches on average. The storage capacity C of each cache is taken to be 10% of the catalog size. We divide the trace consisting of the first $\sim 375K$ requests into $k = 20$ consecutive sub-intervals. File requests are uniformly assigned to the users before running the experiments on each of the sub-intervals.

Results and Discussion: Figure 3 shows the performance distribution of different caching policies in terms of the average cache-hit rate per file and the average number of file-fetches per cache. The average values of the above metrics for different file request sequences are listed in Table 1 in the supplementary section. From the plots, it is clear that the `LeadCache` policy outperforms the baseline policies in terms of *both* cache-hit rate and fetch-rate. Recall that Belady's policy is an *offline* policy that knows the future file requests. Figure 7 in the supplementary section shows a joint plot of the cache-hit rates and download rates of different policies. From the plots, it is clear that the `LeadCache` policy excels by coordinating the caching decisions among different caches and quickly adapting with the file request sequence. Section 14 of the supplementary material gives additional plots for the popularity distribution and the temporal dynamics of the policies for a given file request sequence, where we observe a similar pattern.

8 Conclusion and Future Work

We proposed an efficient network caching policy and showed that it is competitive with the existing caching policies, both theoretically and empirically. We proved a tight lower bound for the achievable regret and proved that the number of file-fetches incurred by our policy is finite under reasonable assumptions. Although any existing CDN load-balancer can be used along with `LeadCache`, a major limitation of our work is that it does not jointly optimize the load balancing and cache allocations. Furthermore, it will be interesting to design caching policies with strong guarantees for both regret and the competitive ratio [Daniely and Mansour, 2019].

References

Wouter M Koolen, Manfred K Warmuth, Jyrki Kivinen, et al. Hedging structured concepts. In *COLT*, pages 93–105. Citeseer, 2010.

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

Erik Nygren, Ramesh K. Sitaraman, and Jennifer Sun. The akamai network: A platform for high-performance internet applications. *SIGOPS Oper. Syst. Rev.*, 44(3):2–19, August 2010. ISSN 0163-5980. doi: 10.1145/1842733.1842736. URL `https://doi.org/10.1145/1842733.1842736`.

Georgios Paschos, George Iosifidis, and Giuseppe Caire. Cache optimization models and algorithms. *Foundations and Trends® in Communications and Information Theory*, 16(3–4):156–345, 2020. ISSN 1567-2190. doi: 10.1561/0100000104. URL `http://dx.doi.org/10.1561/0100000104`.

Rajarshi Bhattacharjee, Subhankar Banerjee, and Abhishek Sinha. Fundamental limits on the regret of online network-caching. *Proc. ACM Meas. Anal. Comput. Syst.*, 4(2), June 2020. doi: 10.1145/3392143. URL `https://doi.org/10.1145/3392143`.

Nicolo Cesa-Bianchi and Gábor Lugosi. *Prediction, learning, and games*. Cambridge university press, 2006.

Arun Sai Suggala and Praneeth Netrapalli. Online non-convex learning: Following the perturbed leader is optimal. In *Algorithmic Learning Theory*, pages 845–861, 2020.

James Hannan. Approximation to bayes risk in repeated play. *Contributions to the Theory of Games*, 3:97–139, 1957.

Adam Kalai and Santosh Vempala. Efficient algorithms for online decision problems. *Journal of Computer and System Sciences*, 71(3):291–307, 2005.

Luc Devroye, Gábor Lugosi, and Gergely Neu. Random-walk perturbations for online combinatorial optimization. *IEEE Transactions on Information Theory*, 61(7):4099–4106, 2015.

Jacob Abernethy, Chansoo Lee, and Ambuj Tewari. Perturbation techniques in online learning and optimization. *Perturbations, Optimization, and Statistics*, page 233, 2016.

Amit Daniely, Alon Gonen, and Shai Shalev-Shwartz. Strongly adaptive online learning. In *International Conference on Machine Learning*, pages 1405–1411, 2015.

Kwang-Sung Jun, Francesco Orabona, Stephen Wright, and Rebecca Willett. Improved strongly adaptive online learning using coin betting. In *Artificial Intelligence and Statistics*, pages 943–951. PMLR, 2017.

Stefano Traverso, Mohamed Ahmed, Michele Garetto, Paolo Giaccone, Emilio Leonardi, and Saverio Niccolini. Temporal locality in today's content caching: why it matters and how to model it. *ACM SIGCOMM Computer Communication Review*, 43(5):5–12, 2013.

Benjamin Van Roy. A short proof of optimality for the min cache replacement algorithm. *Information processing letters*, 102(2-3):72–73, 2007.

Donghee Lee, Jongmoo Choi, Jong-Hun Kim, Sam H Noh, Sang Lyul Min, Yookun Cho, and Chong-Sang Kim. On the existence of a spectrum of policies that subsumes the least recently used (lru) and least frequently used (lfu) policies. In *SIGMETRICS*, volume 99, pages 1–4. Citeseer, 1999.

Asit Dan and Don Towsley. *An approximate analysis of the LRU and FIFO buffer replacement schemes*, volume 18. ACM, 1990.

Amit Daniely and Yishay Mansour. Competitive ratio vs regret minimization: achieving the best of both worlds. In *Algorithmic Learning Theory*, pages 333–368, 2019.

Nick Littlestone and Manfred K Warmuth. The weighted majority algorithm. *Information and computation*, 108(2):212–261, 1994.

Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

Karthikeyan Shanmugam, Negin Golrezaei, Alexandros G Dimakis, Andreas F Molisch, and Giuseppe Caire. Femtocaching: Wireless content delivery through distributed caching helpers. *IEEE Transactions on Information Theory*, 59(12):8402–8413, 2013.

Georgios S Paschos, Apostolos Destounis, Luigi Vigneri, and George Iosifidis. Learning to cache with no regrets. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pages 235–243. IEEE, 2019.

Francesco Orabona. A modern introduction to online learning. *arXiv preprint arXiv:1912.13213*, 2019.

Alexander A Ageev and Maxim I Sviridenko. Pipage rounding: A new method of constructing algorithms with proven performance guarantee. *Journal of Combinatorial Optimization*, 8(3): 307–328, 2004.

Georgios S Paschos, George Iosifidis, Meixia Tao, Don Towsley, and Giuseppe Caire. The role of caching in future communication systems and networks. *IEEE Journal on Selected Areas in Communications*, 36(6):1111–1125, 2018.

Reinhard Diestel. Graph theory 3rd ed. *Graduate texts in mathematics*, 173, 2005.

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

Chansoo Lee. *Analysis of Perturbation Techniques in Online Learning*. PhD thesis, 2018.

Sascha Geulen, Berthold Vöcking, and Melanie Winkler. Regret minimization for online buffering problems using the weighted majority algorithm. In *COLT*, pages 132–143. Citeseer, 2010.

Jacob Abernethy, Peter L Bartlett, Alexander Rakhlin, and Ambuj Tewari. Optimal strategies and minimax lower bounds for online convex games. 2008.

Alfred V Aho, Peter J Denning, and Jeffrey D Ullman. Principles of optimal page replacement. *Journal of the ACM (JACM)*, 18(1):80–93, 1971.

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

Daniel S Berger. CDN Traces. `https://github.com/dasebe/optimalwebcaching`, 2018.

Martin J Wainwright. *High-dimensional statistics: A non-asymptotic viewpoint*, volume 48. Cambridge University Press, 2019.

Pascal Massart. *Concentration inequalities and model selection*, volume 6. Springer, 2007.

Jacob Abernethy and Alexander Rakhlin. Optimal strategies and minimax lower bounds for online convex games. In *In Proceedings of the Nineteenth Annual Conference on Computational Learning Theory*, 2008.

Noga Alon and Joel H Spencer. *The probabilistic method*. John Wiley & Sons, 2004.

Godfrey Harold Hardy, John Edensor Littlewood, George Pólya, György Pólya, DE Littlewood, et al. *Inequalities*. Cambridge university press, 1952.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 8.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No] We will be releasing the full codebase upon the acceptance of the paper.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [N/A]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] Showed the empirical distributions.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [N/A]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] Included the full reference to the dataset used Berger et al. [2018], Berger [2018].
   (b) Did you mention the license of the assets? [Yes] In Section 7, we mentioned that the dataset is available under a BSD 2-Clause License.
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [Yes] Mentioned that we used an anonymized production trace. See Section 7.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]