Meta-learning the Best Caching Expert

Tuning caching policies with expert advice

Maik de Vries[1]

Supervisors: Georgios Iosifidis[1]**, Naram Mhaisen**[1]

[1]**EEMCS, Delft University of Technology, The Netherlands**

June 21, 2024

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering

Name of the student: Maik de Vries
Final project course: CSE3000 Research Project
Thesis committee: Georgios Iosifidis, Naram Mhaisen, Neil Yorke-Smith

Meta-learning the Best Caching Expert

Maik de Vries

Abstract—In recent years, the novel framing of the caching problem as an Online Convex Optimisation (OCO) problem has led to the introduction of several online caching policies. These policies are proven optimal with regard to regret for any arbitrary request pattern, including that of adversarial origin. Nevertheless, their performance is primarily affected by the tuning of their so-called hyperparameters (e.g. learning rate), which is often done under the presumption of adversarial conditions. Consequently, as not all request patterns encountered in practice are adversarial, this suggests potential for further improvements. Unfortunately, the tuning of these hyperparameters to achieve optimal performance across non-adversarial request sequences remains dubious and poses a new challenge. This paper proposes an online meta-learner that combines several instances of the OGA caching policy, each distinct in their learning rate, framing the problem as an expert advice decision problem. The proposed meta-learner dynamically shifts between caching experts and achieves a regret upper bound similar to that of the best-performing caching expert. The penalty introduced for learning the best expert is limited to a logarithmic dependence on the number of experts, which improves upon previous works. Numerical evaluations composed of synthetic request traces demonstrate consistent improvements when the caching experts' relative performance varies over time.

1 Introduction

1.1 Motivation and Background

The field of computer science spans a range of disciplines, many of which face a common *caching problem*. A *cache* has as its main goal to provide fast access to a size-limited subset of some complete library; with applications thereof varying from virtual memory [1] to the world-wide-web [2], file systems [3], and operating systems [4] to name a few. The problem arises when it comes to deciding what is to be kept and what is to be evicted from such a cache.

Over the years, various *cache replacement algorithms* have been proposed [5], each with its strengths and weaknesses. Their performance is commonly quantified in terms of their *cache hit ratio*: the fraction of requests which can be served directly by the cache, resulting in a *cache hit*; whereas a request for content not present in the cache leads to a *cache miss*. These cache replacement algorithms, more commonly known as *caching policies*, aim to maximise their cache hit ratio.

The main limitation of most of these caching policies lies in the need to know certain properties of the request pattern upfront, such as whether it can be assumed to be stationary or non-stationary (i.e. time-varying). The most suitable policy for a given situation is then determined based on these assumptions. For instance, the *Least Frequently Used* (LFU) and *Least Recently Used* (LRU) caching policies are characterised by caching the requested content, with LFU evicting the least frequently used file, while LRU frees the file least recently accessed. Although both policies enjoy widespread usage, prior research demonstrates their sub-optimal performance when challenged with unpredictable request sequences [6]–[10]. Additionally, consider the case of content delivery networks (CDNs), which often span the entire globe, facing different and evolving request patterns as a result. This further illustrates the need for dynamic caching policies that base their decisions on prior observations rather than statistical expectations.

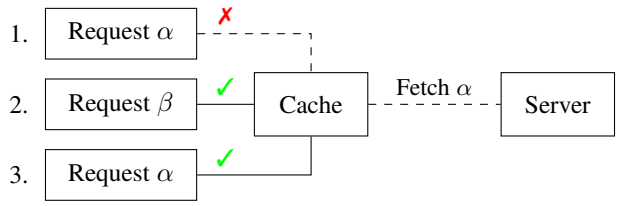


Figure 1 A caching policy determines what files to store in the cache to maximise the cache hit ratio: (1) a request is made for α, which results in a cache miss and α is fetched from the origin server; (2) a request is made for β, which results in a cache hit; (3) a second request is made for α, which now results in a cache hit.

In recent years, the age-old problem has been approached anew from the context of *online learning* [11]–[13], bringing forth several online caching policies [14]–[16]. This class of algorithms continuously adapt and learns the optimal decision policy based on the success of their previous decisions; and in doing so, avoids the aforementioned necessity for upfront knowledge. Hence, they are proven to guarantee an optimal caching policy for any arbitrary request pattern, including that of *adversarial* origin [15], [17]. Furthermore, their scalability and lacking requirement to be trained, in contrast to *offline learning*, come as additional advantages. Nevertheless, their performance is primarily affected by the tuning of their so-called *hyperparameters* (e.g. learning rate), which is often done under the presumption of adversarial conditions. Consequently, as not all request patterns encountered in practice are adversarial, this suggests potential for further improvements. Unfortunately, the tuning of these hyperparameters to achieve optimal performance across non-adversarial request sequences remains dubious and poses a new challenge.

This work aims to address the identified challenge of learning rate tuning and provide a solution to the question: *is there an effective way to learn the optimal tuning of such hyperparameters to derive the best caching policy?*

1.2 Methodology and Contributions

Consistent with the above-mentioned online caching policies, the approach proposed in the remainder of this paper follows the mathematical framework of *Online Convex Optimisation* (OCO) [18]–[20]. This subjects some learner algorithm to the process of selecting a decision vector y_t from a convex set $\mathcal{Y}$ at each time slot t; all without knowledge of its performance determined by some time-varying utility function $f_t(y)$, which is revealed only after y_t has been selected. The learner algorithm has as its objective to minimise the growth of *regret* over some horizon T:

$$R_T = \sum_{t=1}^{T} f_t(y^*) - \sum_{t=1}^{T} f_t(y_t)$$

Here, y^* is the benchmark algorithm with hindsight (i.e. complete knowledge of future functions $f_t, \forall t$), defined as:

$$y^* = \arg\max_{y \in \mathcal{Y}} \sum_{t=1}^{T} f_t(y)$$

In its application of the OCO framework to the caching problem, the role of "learner algorithm" is fulfilled by some caching policy; and its associated cache configuration takes on the role of "decision vector", which is decided upon before the reveal of the request dependent value of $f_t(y)$[1]. Furthermore, an online caching policy desires *sublinear* regret $R_T = o(T)$, so as to ensure negligible average regret $\frac{R_T}{T} \approx 0$ as $T \to \infty$. In contrast to the lower bound of $\Omega(T)$ for both the LFU and LRU caching policies [15], previous research has established sublinear regret to be the absolute lower bound for OCO-based policies [15], [17]; coining such policies *no-regret caching policies*.

One such no-regret caching policy is the *Online Gradient Ascent* (OGA) algorithm, originally proposed in [15]. Although OGA enjoys the guarantee of sublinear regret, its strict upper bound is derived under the assumption of adversarial conditions (i.e. worst-case scenario); and subsequently, the optimal tuning of its learning rate is obtained in a similar fashion [15, Theorem 2]. This implies that there is potential room for improvement when the OGA caching policy encounters request patterns of non-adversarial origin.

This paper therefore suggests framing the optimisation problem as an *expert advice decision problem* [21]–[23], proposing the application of a *meta-learner* [24] over several OGA caching experts with varying learning rates; and in doing so, reduce the problem from careful tuning based on the underlying (unknown) system parameters, to learning the best performing caching expert.

The notion of tracking the best expert in this regard is well-established, as a sample from the plethora of previous research indicates [25]–[30]. Within the domain of caching, the novelty of caching policy experts has been the subject of studies in the field of (deep) reinforcement learning [31], [32]. Nevertheless, this approach is limited by the core assumption that the learnt request distribution remains stationary over time (i.e. suffers under time-varying conditions). Although expert-based meta-learners are widely studied in online learning, their application within the context of no-regret caching policies remains relatively unexplored.

As far as current understanding goes, the only works that study the combination of the expert and online caching problem are [33]–[37]. Notably, [33, Section 5] presents an optimistic online caching policy that utilises advice from multiple predictors through the use of an OGA meta-learner; which is extended into the realm of discrete caching in [34, Section 8]; whereas [35, Section 5] proposes leveraging recommendation systems as optimistic predictors under an OFTRL meta-learner. Unfortunately, as a result of using either the OGA or OFTRL algorithm in learning the optimal predictor, the regret upper bound of these meta-learners suffers an additional polynomial dependence on the number of predictors [33]–[35]. Furthermore, the application of a meta-learner across a variety of caching policies was studied in [36], and [37] proposed an IAWM meta-learner over several FTPL instances. Nonetheless, the regret upper bound of these meta-learners suffers an additional dependence on library size N [36, Section 4.4] [37, Section 3.2]. Hence, a more efficient meta-learner across no-regret caching experts remains hitherto unexplored.

As such, the main focus of this work is an *online meta-learner* [38] based on the *Exponentiated Gradient* (EG) [39] algorithm. To this end, the aforementioned OCO framework is extended with an additional layer on which the meta-learner operates (i.e. the meta-learner follows the OCO framework); at each time slot, the meta-learner is tasked with selecting a caching expert, incurring its cost and learning to minimise regret by updating caching experts' weights accordingly. Moreover, the use of the EG algorithm in this regard does not expand the regret upper bound as substantially as the above-mentioned meta-learners have; and instead introduces a *logarithmic* dependence relative to the number of caching experts [40, Algorithm 6.3].

In summary, the contributions made in this paper consist of:

- The introduction of an online meta-learner across distinct OGA caching experts as an application of the OCO framework to the expert advice decision problem. This approach achieves regret similar to the best caching expert, serving only a logarithmic dependence on the number of experts for learning the best-performing expert, and improves upon prior works [33]–[37].
- An empirical evaluation through representative synthetic request models; comparing its performance to the best in hindsight benchmark, as well as the individual caching experts.

Paper organisation

The remainder of this paper is organised as follows. Section 2 defines the system model and formally introduces the problem statement. Section 3 presents the online meta-learner algorithm, including an upper bound for its regret, and Section 4 provides a numerical evaluation composed of synthetic traces. Section 5 states the conclusions, and Section 6 reflects on any involved ethical aspects, as well as reproducibility.

1 Take note of the absent definition for utility function $f_t(y)$, illustrating the versatility of the OCO framework in modelling the online caching problem; thus permitting the optimisation of any arbitrary performance function.

2 System model and problem statement

2.1 System model

The model used to describe the system of interest follows that of [15]. For completeness, its definitions of terms will be provided first, as well as expanded upon next. In the remainder of this paper, let v_i^j denote the element at index j of some vector v_i.

The system consists of a library of unit-sized files $\mathcal{N} = \{1, 2, \ldots, N\}$ and a cache of size $C < N$. Furthermore, the system functions in a time-slotted fashion, with at each time slot $t = \{1, 2, \ldots, T\}$ a single request for a file $n \in \mathcal{N}$, denoted by the event $x_t^n = 1$.

The N-dimensional *request vector* x_t at time slot t is described by the feasible set of requests $\mathcal{X}$:

$$\mathcal{X} = \left\{ x \in \{0,1\}^N \;\middle|\; \sum_{n=1}^{N} x^n = 1 \right\}$$

Thus, for each slot t an N-dimensional request vector x_t contains a single element set to 1, describing a request for the corresponding file. Furthermore, no assumptions are made as to the properties of the distribution of x_t. The assumption for it to be unknown and arbitrary ensures caching policies deemed performant under this model, to perform similarly or better under different models.

The N-dimensional vector y_t contains the *cache configuration* at time slot t and is described by the feasible set of caching configurations $\mathcal{Y}$:

$$\mathcal{Y} = \left\{ y \in [0,1]^N \;\middle|\; \sum_{n=1}^{N} y^n \leq C \right\}$$

Hence, for each slot t an N-dimensional cache configuration vector y_t contains elements in the continuous range from 0 up to 1, and its sum equals at most the cache size C. These constraints allow for *partial caching* (also known as *coded caching*), which is desirable in certain applications (e.g. video files made up of independent chunks) [41]. Note that the aforementioned restriction on file size can be readily alleviated by substitution of the capacity constraint in $\mathcal{Y}$ with:

$$\sum_{n=1}^{N} s^n y^n \leq C$$

where s is an N-dimensional vector in $\mathbb{R}_+^N$, such that s^n describes the size of file $n \in \mathcal{N}$.

At time slot t, the N-dimensional vector $w_t \in \mathbb{R}^N$ describes the weight of each file $n \in \mathcal{N}$ in the computation of the *utility function* $f(w_t, x_t, y_t)$. In other words, if file n were to be requested and resulted in a full cache hit, the achieved utility would be equivalent to w_t^n. The definition of the utility function $f(w_t, x_t, y_t)$ is given next:

$$f(w_t, x_t, y_t) \equiv f_t(y_t) = \sum_{n=1}^{N} w_t^n x_t^n y_t^n \tag{1}$$

Therefore, the system accumulates utility at time slot t when the requested file x_t^n has been (partially) cached in cache configuration y_t.

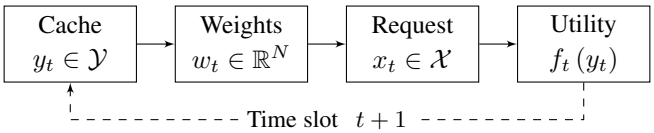


Figure 2 A cache configuration y_t is chosen; an adversary reveals weights w_t; an adversary reveals request x_t; the achieved utility $f_t(y_t)$ is computed; and the next time slot $t+1$ is processed.

2.2 Problem statement

In framing the caching problem as an *Online Convex Optimisation* (OCO) problem, it is important to highlight the following assumed system conditions. Given the arbitrary nature of the underlying distribution of x_t, it may additionally be understood as each x_t being chosen by an *adversary* to degrade the caching policy's performance; as well as having complete control over the selection of weights vector w_t. Furthermore, note that the cache configuration y_t is decided upon before the adversary introduces both w_t and x_t, and consequently, without any upfront knowledge of $f_t(y_t)$. Refer to Figure 2 for a visual outline of the OCO-based adaptation of the caching problem.

Recall that a *caching policy* seeks to maximise the number of cache *hits* and thus, minimise the number of cache *misses*. This is reflected in the definition of the utility function f_t in (1). Caching policies which therefore cache files that accumulate the most utility given an arbitrary request sequence, outperform policies which fail to cache these particular files. Within the context of *online learning*, caching policies are dynamic and continuously adapt. They learn the optimal decision policy based on previous requests, its past cache configurations, and the success of its decisions based on them. As a result, they similarly learn to cache the files that have been accumulating the most utility.

Similar to [15], evaluating the performance of some caching policy σ is done by comparing it to the *best static cache configuration in hindsight* benchmark y^*. This benchmark has complete knowledge of the request sequence $x_1, \ldots, x_T$, and is thus able to determine the highest utility-earning static cache configuration y^*:

$$y^* \in \arg\max_{y \in \mathcal{Y}} \sum_{t=1}^{T} f_t(y)$$

The performance of caching policy σ can then be quantified as a comparison, and is termed the *static regret of policy* σ:

$$R_T(\sigma) = \max_{P(x_1,\ldots,x_T)} \mathbb{E}\left[\sum_{t=1}^{T} f_t(y^*) - \sum_{t=1}^{T} f_t(y_t(\sigma)) \right]$$

An important consequence of defining the performance of caching policy σ as such is that the detrimental effect an adversary can have on the performance of σ will also translate to similar losses in that of y^*.

The objective is then to minimise the growth of regret over the infinite horizon $T \to \infty$, which requires a caching policy σ to achieve sublinear regret: $R_T(\sigma) = o(T)$. This ensures the average performance gap $\frac{R_T(\sigma)}{T}$ diminishes as T scales. In other words, caching policy σ adapts and learns the optimal cache configuration without any upfront knowledge of the distribution of x_t. Theoretically, depending on the encountered request pattern and chosen caching policy, the total regret could even be subzero.

This is due to the static nature of benchmark y^*; which, compared to the dynamic cache configuration y_t, might suffer reduced performance $f_t(y^*) \leq f_t(y_t)$ for some time slots, yet achieve similar performance $f_t(y^*) \approx f_t(y_t)$ during the remaining slots.

Recall that the *Online Gradient Ascent* (OGA) algorithm guarantees an optimal cache configuration under adversarial conditions, as well as a static regret upper bound of $\mathcal{O}\left(\sqrt{CT}\right)$ [15, Theorem 2]. Nonetheless, the assumed adversarial setting does not reflect all plausible request patterns; and tuning its optimality in this regard remains ambiguous, as will become clear shortly.

The *update rule* is central in the determination of cache configuration y_{t+1}, and its definition for the OGA algorithm is given in (2). Similar to other online caching policies, the selection of y_{t+1} is solely based on the current cache configuration y_t, the request x_t, and their associated utility $f_t(y_t)$. There is therefore no necessity to track and store the full history of all cache configurations nor all requests up to the current time slot t. Hence, after recording the obtained utility at time slot t, OGA's next cache configuration y_{t+1} is computed as:

$$y_{t+1} = \Pi_{\mathcal{Y}}\left(y_t + \eta \nabla f_t\right) \tag{2}$$

where $\Pi_{\mathcal{Y}}$ is the Euclidean projection onto $\mathcal{Y}$, defined as a constrained quadratic program, minimising the Euclidean squared distance:

$$\Pi_{\mathcal{Y}}(z) \triangleq \underset{y \geq 0}{\arg\min} \sum_{n=1}^{N} \left(z^n - y^n\right)^2$$
$$\text{s.t.} \quad \sum_{n=1}^{N} y^n \leq C \quad \text{and} \quad y^n \leq 1, \ \forall n \in \mathcal{N}$$

and η is the learning rate, with the gradient ∇f_t at y_t defined as the N-dimensional vector with elements:

$$\frac{\partial f_t}{\partial y_t^n} = w_t^n x_t^n, \ n = 1, \ldots, N$$

So at each time slot t, the cache configuration y_{t+1} moves a factor η into the direction of current request x_t, using Euclidean projection to project onto the set of feasible cache configurations $\mathcal{Y}$ if necessary.

The suggested optimal tuning of learning rate η in [15, Theorem 2] is derived under the aforementioned assumption of adversarial influence. Consequently, the guarantee of sublinear regret is established as its upper bound, expressed in the following inequality:

$$R_T(OGA) \leq \frac{\operatorname{diam}(\mathcal{Y})^2}{2\eta} + \frac{\eta T L^2}{2} \tag{3}$$

where $\operatorname{diam}(\mathcal{Y})$ describes the maximum Euclidean distance between any two elements in set $\mathcal{Y}$, and L denotes the upper bound of the gradient ∇f_t:

$$\|\nabla f_t\|_\infty \leq \|w_t\|_\infty \leq L$$

The optimal learning rate then follows from solving (3) for η:

$$\eta = \frac{\operatorname{diam}(\mathcal{Y})}{L\sqrt{T}}$$

Hence, the proposed tuning of η is derived under the premise of an adversary. Although this additionally ensures sublinear static regret given any arbitrary request pattern, it cannot be deemed the most optimal (i.e. no one-size-fits-all solution). Unfortunately, tuning the learning rate to this end remains dubious, as no assumptions can be made for the distribution of x_t, other than it being of an adversarial nature.

These observations then drive for further exploration as to whether the careful tuning itself can be cast as an OCO problem.

3 META-LEARNER

It has been established that the *Online Gradient Ascent* (OGA) algorithm guarantees an optimal cache configuration under adversarial conditions [15]. Furthermore, tuning its learning rate for optimal performance given non-adversarial request sequences remains tedious. Therefore, this work aims to introduce an *online meta-learner*; framing the optimal tuning of said learning rate as an additional *Online Convex Optimisation* (OCO) problem in the form of an *expert advice decision problem*. As such, the *Exponentiated Gradient* (EG) [39] algorithm is utilised as a meta-learner across a multitude of OGA *caching experts*, distinct in their learning rates. Unlike the previous works of [33]–[37], the novel proposal of the EG meta-learner contains a logarithmic dependence on the number of experts in its regret upper bound (8).

An important requirement of an expert-based meta-learner is its ability to learn from each underlying expert's performance. Experts who are more performant should be regarded as more significant than experts who are less performant. This distinction enables the meta-leaner to learn which expert is best in a given scenario. Additionally, it must not be the case that once such an expert has been identified, it is assumed to remain the best-performing expert. Therefore, a change in an expert's performance should cause the meta-learner to change its associated weight. Furthermore, no single expert should ever become the sole dictator (i.e. total bias towards a single expert); and were this requirement to be omitted, no meta-learner would be able to guarantee the previous requirement, as a change in expert performance would not lead to a change in the expert's weight. The proposed EG meta-learner abides by these requirements, which will become clear through the description of its implementation provided shortly.

3.1 Caching Experts

Let the following terms and definitions expand upon the system model described in Section 2. The set of distinct caching experts $\mathcal{K} = \{1, \ldots, K\}$ is introduced, with none of the experts limited to any particular caching policy σ. Let $y_{k,t}$ denote the cache configuration $y \in \mathcal{Y}$ of expert $k \in \mathcal{K}$ at corresponding time slot t. The utility achieved by cache configuration $y_{k,t}$ is then equivalent to $f_t(y_{k,t})$, and let the *regret of expert k* be defined as $R_T\left(\sigma^k\right)$. Furthermore, each entry u_t^k in the K-dimensional vector u_t describes the utility achieved by the corresponding cache configuration $y_{k,t}$.

The K-dimensional vector m_t denotes the expert probability weights at time slot t and is described by the feasible set of expert probability weights $\mathcal{M}$:

$$\mathcal{M} = \left\{ m \in [0,1]^K \;\middle|\; \sum_{k=1}^{K} m^k = 1 \right\}$$

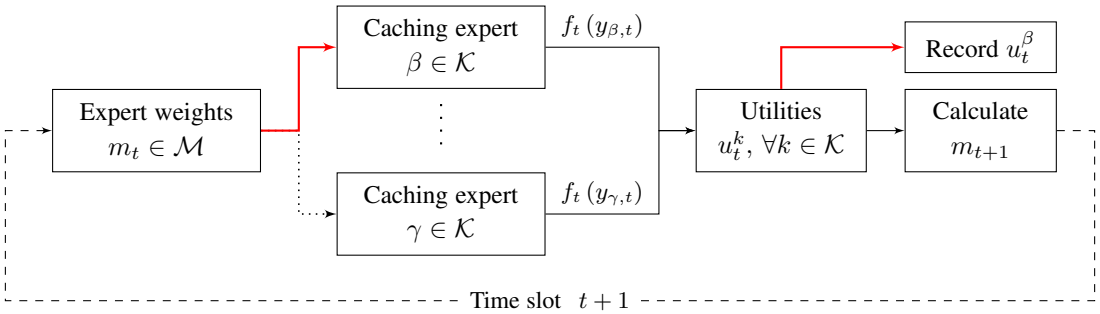


FIGURE 3 A caching expert β is randomly selected based on expert probability vector m_t; all caching experts process request x_t and output their obtained utilities $f_t\left(y_{k,t}\right)$; the meta-learner records the utility of the selected expert u_t^β; the expert probability vector m_{t+1} is computed; and the next time slot $t+1$ is processed.

Thus, each entry m_t^k denotes some probability weight $p\left(k\right)$ associated with expert $k \in \mathcal{K}$ in the probability vector m_t.

The aim is then to meta-learn a caching policy σ^* with a regret upper bound close to the achieved regret of the best caching expert:

$$R_T\left(\sigma^*\right) \leq \alpha + \min_{k \in \mathcal{K}} \left\{ R_T\left(\sigma^k\right) \right\} \tag{4}$$

where α denotes the penalty of having to learn the best caching expert and is ultimately determined based on the chosen meta-learner algorithm.

3.2 Exponentiated Gradient

The EG algorithm is tasked with updating the expert probability vector m_t relative to the achieved utility $f_t\left(y_{k,t}\right)$ of each expert k, and its update rule is given in (5). Recall that online learning problems base their updates solely on data available at the current time slot t, in contrast to keeping track of the full history of all prior time slots up to t. Therefore, the expert probability weights m_{t+1} are computed as:

$$m_{t+1}^k = \frac{m_t^k \exp\left(\delta \nabla u_t^k\right)}{\sum_{i=1}^K m_t^i \exp\left(\delta \nabla u_t^i\right)}, \; k = 1, \ldots, K \tag{5}$$

where the learning rate δ is defined as:

$$\delta = \sqrt{\frac{2 \ln K}{\|w_t\|_\infty^2 \, T}} \tag{6}$$

and the utility gradient ∇u_t^k of each expert $k \in \mathcal{K}$ is described by the K-dimensional utility vector u_t: $\nabla u_t \equiv u_t$. The result is the normalised vector m_{t+1}, and as such analogous to expert probabilities.

In contrast to OGA's *additive* update rule, EG instead makes use of a *multiplicative* definition. This allows for faster convergence of the best expert in comparison to additive updates. However, a major pitfall of multiplicative updates is total convergence (i.e. all but one weight to zero) [42]. This would leave a single expert as the sole dictator and prevent the meta-learner from adapting once its performance starts to vary, a problem alluded to at the start of this section. The solution in circumventing this problem is twofold: the interpretation of expert weights as *probabilities*, and characteristics of the gradient ∇u_t detailed next.

Algorithm 1: Caching expert meta-learner (σ^*)

Input: T; $\mathcal{K}$; $\|w_t\|_\infty$
Output: $f_t\left(\sigma^*\right), \forall t$

Initialise m_t according to (7)
Calculate δ according to (6)
for $t = 1$ **to** T **do**
  Select $k_t \in \mathcal{K}$ at random based on probabilities m_t
  Observe the utilities $u_t^k, \forall k \in \mathcal{K}$
  Record utility of selected expert $u_t^{k_t}$
  Calculate expert probability vector m_{t+1} using (5)

The definition of gradient $\nabla u_t \equiv u_t$ ensures experts who achieve nonzero utility are rewarded by increasing their respective probability weights. This follows from the fact that utility cannot be negative, which guarantees that the exponent $e^{\delta \nabla u_t} \geq 1$. Hence, an expert's weight is solely determined by its relative performance: outperforming all other experts will result in a substantial increase, whereas similar performance will result in minimal changes. Furthermore, a nonperforming expert will quickly see its weight tumble due to other agents' increasing weights, suppressing it in normalisation. These properties ensure the meta-learner's ability to shift towards the best-performing caching expert, even when a single expert dominates probability vector m_t.

The main components of the online meta-learner algorithm σ^* have now been defined and discussed. The complete algorithm is detailed in Algorithm 1 and a step-by-step description is given next; see Figure 3 for a visual overview.

Before processing time slot $t = 1, 2, \ldots, T$, the K-dimensional vector m_t is initialised and describes the probability weight of each caching expert $k \in \mathcal{K}$ at time slot $t = 1$:

$$m_{t=1} = \left[\frac{1}{K}\right]^K \tag{7}$$

Therefore assigning each caching expert with uniform probability, and as a result, no particular caching expert is considered to be the best before the first time slot commences. Additionally, the static learning rate δ is computed based on the number of experts K, the upper bound on the file weights in vector w_t and the system horizon T. Then for each time slot t, an expert k_t is selected at random according to the expert probability vector

m_t. Next, the vector u_t is determined as request x_t is revealed and the meta-learner accumulates utility equal to its randomly picked expert: $u_t^{k_t}$. Finally, the expert probability weights m_{t+1} are calculated.

The meta-learner penalty α in the regret upper bound inequality of σ^* (4), inherits the upper bound proven in [40, Algorithm 6.3]; which similarly utilises the EG algorithm as meta-learner across experts. Considering σ^* does not include modifications to the EG algorithm, its proven regret upper bound comes for free. The regret upper bound of caching expert meta-learner σ^* over horizon T is thus:

$$R_T(\sigma^*) \leq \alpha + \min_{k \in \mathcal{K}} \left\{ R_T\left(\sigma^k\right) \right\}$$
$$\text{with} \quad \alpha \leq \frac{\sqrt{2}}{2} \|w_t\|_\infty \sqrt{T \ln K} \tag{8}$$

Therefore, the penalty of having to learn the best caching expert is logarithmic relative to the number of caching experts. This is an improvement over the previously established meta-learner upper bounds in [33]–[37].

It is important to highlight the inclusion of the regret achieved by the best caching expert in the upper bound of σ^*; therefore, ensuring the meta-learner will perform no worse than a logarithmic term comparatively. However, this does not encapsulate the scenario in which the best caching expert varies with time. In such cases, one can imagine meta-learner σ^* enjoys regret strictly lower than any of its underlying caching experts.

4 Evaluation

The proposed online meta-learner caching policy σ^* is evaluated as a single cache across three i.i.d. Zipfian distributed request models; with different values for parameter $\zeta \in \{0.6, 0.8, 1.0\}$, each representative of real-world applications [43]–[45]. Since OCO-based caching policies already enjoy proven optimality under adversarial conditions, none of the simulations considers such a case. Comparisons are made to the *Best Static Configuration in Hindsight* (BSCH) benchmark y^*, as well as the individual caching experts $k \in \mathcal{K}$ available to σ^*.

In particular, the case where $\mathcal{K}$ consists of various *Online Gradient Ascent* (OGA) [15] experts with distinct learning rates $\eta \in \{0.05, 0.1, 0.3, 1.0\}$ is studied. Furthermore, the library is set to contain $N = 2500$ files, with a cache size restricted to $C = 250$ files, and file weights set uniformly $w_t = [1]^N, \ \forall t$ to study the cache hit rate scenario. Finally, let the *average utility* be defined as $\frac{1}{t}\sum_{i=1}^{t} f_i(y_i)$.

Figure 4 depicts the average utility progression of each caching expert $k \in \mathcal{K}$ across the three above-mentioned Zipfian distributed request models; and Figure 5 plots the evolution of each corresponding expert probability weights vector m_t. These figures illustrate the success of meta learner σ^* in learning the best caching expert for any time slot t. On top of that, they further demonstrate their ability to adapt once expert performance varies, causing a shift in which expert is regarded as the best performing. However, the rate at which this happens heavily depends on the difference in achieved performance between any two experts; when the relative performance gap is minimal, the switchover in which expert is regarded best takes substantially longer; Figure 5c shows an example thereof, and Figure 4c confirms that the two caching experts closely match in their obtained utility during this time.

TABLE 1
Performance relative to best-performing caching expert (%)

	Time slot			
ζ	0.25×10^5	0.5×10^5	1.0×10^5	2.0×10^5
0.6	3.49	0.09	−0.98	−0.55
0.8	3.06	0.46	−0.54	−0.36
1.0	3.31	0.94	−0.05	−0.25

Performance comparison between meta-learner σ^* and the best-performing caching expert at various time slots, expressed as a percentage (%) for each Zipfian distributed request model with parameter ζ.

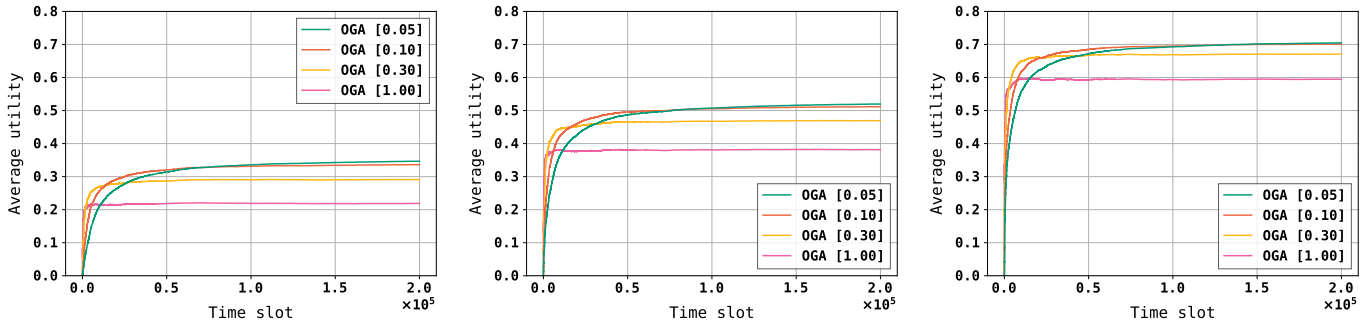
Figure 6 illustrates the average utility obtained by caching expert meta-learner σ^* in comparison to the BSCH benchmark, and the OGA caching experts $k \in \mathcal{K}$ (unlabelled grey entries, see Figure 4). Unsurprisingly, the BSCH benchmark outperforms σ^* across the three Zipfian distributed request models, which is due to their stationary nature. This permits the BSCH policy to simply cache the files most frequently requested. Table 1 summarises the performance of σ^* at various time slots, relative to the best caching expert. Although σ^* slightly underperforms once a single caching expert dominates (recall its learning penalty (8)), a consistent improvement is achieved when individual caching experts' performance fluctuates.

Figure 7 depicts the average regret $\frac{1}{t}\sum_{i=1}^{t} R_i$ of σ^*, as well as that of the theoretical regret upper bound $\sqrt{2CT}$ [15]. Based on these figures, it can be concluded that the regret of σ^* diminishes as T scales. Hence, caching expert meta-learner σ^* is a no-regret caching policy.

5 Conclusions

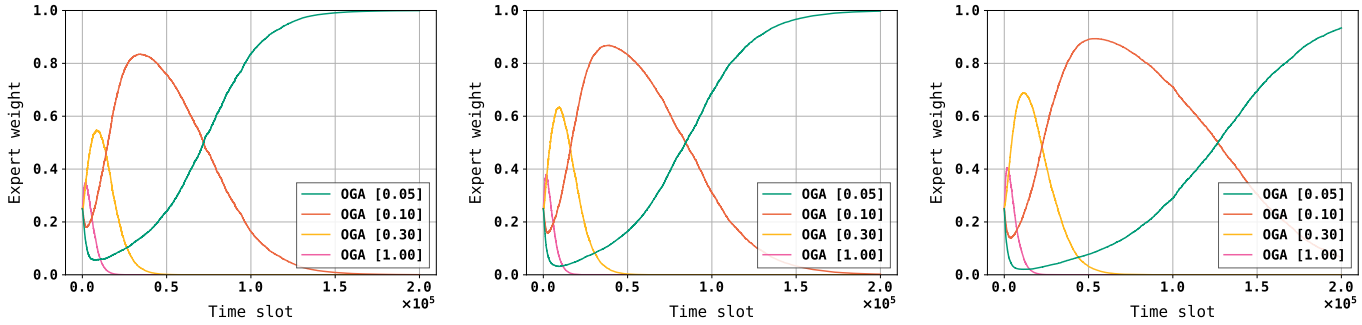
In current state-of-the-art caching research, the optimal tuning of online caching policies is often derived under adversarial assumptions; with rarely any efforts made in tuning given different conditions. This paper proposes a novel method of learning the optimal tuning of an online caching policy's learning rate, to derive the best caching policy. This is achieved by framing the optimisation problem as an expert advice decision problem and leveraging the use of an online meta-learner across several distinct caching policy experts. The obtained online meta-learner caching policy improves upon the previously known static regret bounds of related works [33]–[37], yet remains scalable and robust. Furthermore, it is worth noting any arbitrary online caching policy can be exploited in this regard; including those that utilise optimistic predictors.

Finally, it must be noted that further research should additionally confirm whether these results hold for time-variant request patterns; and future studies could explore potential improvements through the combination of the proposed meta-learner algorithm with adaptive online learning methods.



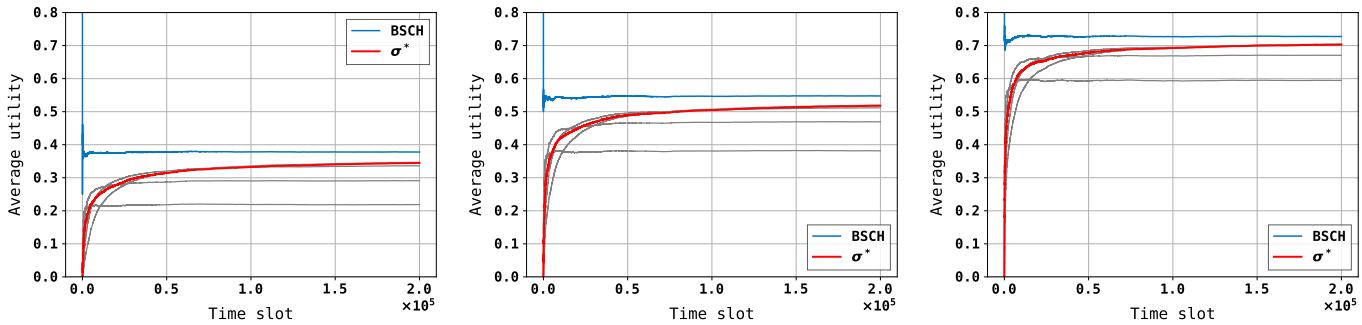
(A) $\zeta = 0.6$ (B) $\zeta = 0.8$ (C) $\zeta = 1.0$

FIGURE 4 Average utility of each OGA caching expert $k \in \mathcal{K}$ across three Zipfian distributed request sequences with different values of ζ; (a) $\zeta = 0.6$, (b) $\zeta = 0.8$, (c) $\zeta = 1.0$; Parameters: $T = 2 \times 10^5$, $N = 2500$, $C = 250$.



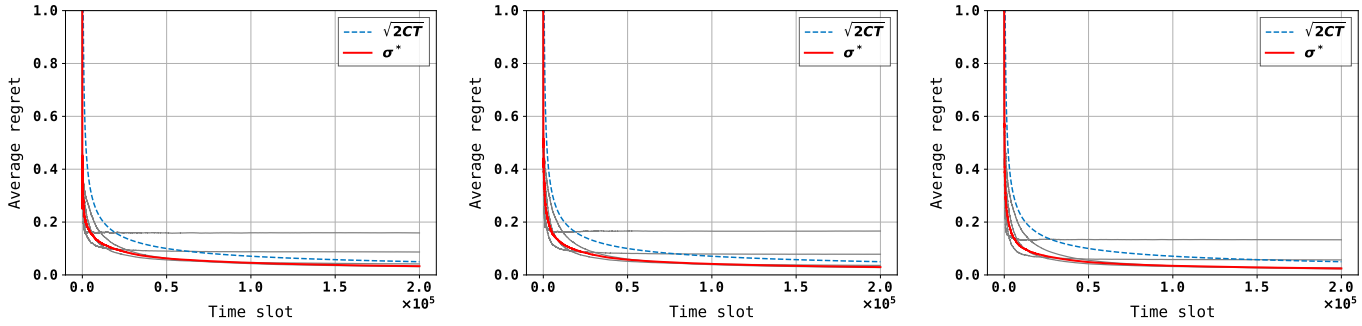
(A) $\zeta = 0.6$ (B) $\zeta = 0.8$ (C) $\zeta = 1.0$

FIGURE 5 Expert probability weights of each OGA caching expert $k \in \mathcal{K}$ across three Zipfian distributed request sequences with different values of ζ; (a) $\zeta = 0.6$, (b) $\zeta = 0.8$, (c) $\zeta = 1.0$; Parameters: $T = 2 \times 10^5$, $N = 2500$, $C = 250$.



(A) $\zeta = 0.6$ (B) $\zeta = 0.8$ (C) $\zeta = 1.0$

FIGURE 6 Average utility of meta learner σ^* compared to the BSCH benchmark and individual caching experts $k \in \mathcal{K}$ (unlabelled grey entries), across three Zipfian distributed request sequences with different values of ζ; (a) $\zeta = 0.6$, (b) $\zeta = 0.8$, (c) $\zeta = 1.0$; Parameters: $T = 2 \times 10^5$, $N = 2500$, $C = 250$.



(A) $\zeta = 0.6$ (B) $\zeta = 0.8$ (C) $\zeta = 1.0$

FIGURE 7 Average regret of meta learner σ^* compared to the theoretical upper bound and individual caching experts $k \in \mathcal{K}$ (unlabelled grey entries), across three Zipfian distributed request sequences with different values of ζ; (a) $\zeta = 0.6$, (b) $\zeta = 0.8$, (c) $\zeta = 1.0$; Parameters: $T = 2 \times 10^5$, $N = 2500$, $C = 250$.

6 Responsible research

A violation of any of the ethical aspects involved in research is deemed unacceptable. Hence, the establishment of this paper and any of the involved research has been conducted with the inclusion of these aspects in mind. Given the nature of this particular research, most ethical aspects are not at risk of being violated. However, research misconduct (falsification, manipulation or misinterpretation of data) nor plagiarism are at the basis of this work, and care has been taken for it to be avoided at all costs.

Reproducibility has been regarded as an important aspect throughout the entire research process. Therefore, uttermost care has been taken to include all required concepts, terms, definitions, algorithms, parameters, and other variables as part of this paper to allow the obtained results to be reproducible. Furthermore, all code as part of this research has been made publicly available through a GitHub repository[2] to ease this process of replication.

References

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

[2] C. C. Aggarwal, J. L. Wolf, and P. S. Yu, "Caching on the world wide web," *IEEE Trans. Knowl. Data Eng.*, vol. 11, no. 1, pp. 95–107, 1999.

[3] M. N. Nelson, B. B. Welch, and J. K. Ousterhout, "Caching in the sprite network file system," *ACM Trans. Comput. Syst.*, vol. 6, no. 1, pp. 134–154, 1988.

[4] M. J. Bach, *The Design of the UNIX Operating System*. Prentice-Hall, 1986.

[5] S. Podlipnig and L. Böszörményi, "A survey of web cache replacement strategies," *ACM Comput. Surv.*, vol. 35, no. 4, pp. 374–398, 2003.

[6] D. Lee, J. Choi, J. Kim, S. H. Noh, S. L. Min, Y. Cho, and C. Kim, "On the existence of a spectrum of policies that subsumes the least recently used (LRU) and least frequently used (LFU) policies," in *SIGMETRICS*. ACM, 1999, pp. 134–143.

[7] 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.

[8] M. Ahmed, S. Traverso, P. Giaccone, E. Leonardi, and S. Niccolini, "Analyzing the performance of LRU caches under non-stationary traffic patterns," *CoRR*, vol. abs/1301.4909, 2013.

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

[10] C. Fricker, P. Robert, and J. Roberts, "A versatile and accurate approximation for LRU cache performance," in *ITC*. IEEE, 2012, pp. 1–8.

[11] J. Hannan, "Approximation to bayes risk in repeated play," *Contributions to the Theory of Games*, vol. 3, no. 2, pp. 97–139, 1957.

[12] A. Blum, "On-line algorithms in machine learning," in *Online Algorithms*, ser. Lecture Notes in Computer Science, vol. 1442. Springer, 1996, pp. 306–325.

[13] N. Cesa-Bianchi and G. Lugosi, *Prediction, learning, and games*. Cambridge University Press, 2006.

[14] S. Geulen, B. Vöcking, and M. Winkler, "Regret minimization for online buffering problems using the weighted majority algorithm," in *COLT*. Omnipress, 2010, pp. 132–143.

[15] G. S. Paschos, A. Destounis, L. Vigneri, and G. Iosifidis, "Learning to cache with no regrets," in *INFOCOM*. IEEE, 2019, pp. 235–243.

[16] T. S. Salem, G. Neglia, and S. Ioannidis, "No-regret caching via online mirror descent," *ACM Trans. Model. Perform. Evaluation Comput. Syst.*, vol. 8, no. 4, pp. 11:1–11:32, 2023.

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

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

[19] S. Shalev-Shwartz, "Online learning and online convex optimization," *Found. Trends Mach. Learn.*, vol. 4, no. 2, pp. 107–194, 2012.

[20] E. Hazan, "Introduction to online convex optimization," *Found. Trends Optim.*, vol. 2, no. 3-4, pp. 157–325, 2016.

[21] N. Littlestone and M. K. Warmuth, "The weighted majority algorithm," *Inf. Comput.*, vol. 108, no. 2, pp. 212–261, 1994.

[22] Y. Freund and R. E. Schapire, "Game theory, on-line prediction and boosting," in *COLT*. ACM, 1996, pp. 325–332.

[23] N. Cesa-Bianchi, Y. Freund, D. Haussler, D. P. Helmbold, R. E. Schapire, and M. K. Warmuth, "How to use expert advice," *J. ACM*, vol. 44, no. 3, pp. 427–485, 1997.

[24] J. Schmidhuber, "Evolutionary principles in self-referential learning, or on learning how to learn: The meta-meta-. hook," Ph.D. dissertation, Technical University of Munich, Germany, 1987.

[25] V. G. Vovk, "A game of prediction with expert advice," *J. Comput. Syst. Sci.*, vol. 56, no. 2, pp. 153–173, 1998.

[26] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *J. Comput. Syst. Sci.*, vol. 55, no. 1, pp. 119–139, 1997.

[27] M. Herbster and M. K. Warmuth, "Tracking the best expert," *Mach. Learn.*, vol. 32, no. 2, pp. 151–178, 1998.

[28] D. P. Foster and R. Vohra, "Regret in the on-line decision problem," *Games and Economic Behavior*, vol. 29, no. 1, pp. 7–35, 1999.

[29] A. T. Kalai and S. S. Vempala, "Efficient algorithms for online decision problems," *J. Comput. Syst. Sci.*, vol. 71, no. 3, pp. 291–307, 2005.

[30] D. Anderson and D. J. Leith, "Learning the best expert efficiently," *CoRR*, vol. abs/1911.04307, 2019.

[31] W. Jiang, G. Feng, S. Qin, T. P. Yum, and G. Cao, "Multi-agent reinforcement learning for efficient content caching in mobile D2D networks," *IEEE Trans. Wirel. Commun.*, vol. 18, no. 3, pp. 1610–1622, 2019.

[32] F. Wang, F. Wang, J. Liu, R. Shea, and L. Sun, "Intelligent video caching at network edge: A multi-agent deep reinforcement learning approach," in *INFOCOM*. IEEE, 2020, pp. 2499–2508.

[33] N. Mhaisen, G. Iosifidis, and D. J. Leith, "Online caching with optimistic learning," in *IFIP Networking*. IEEE, 2022, pp. 1–9.

[34] N. Mhaisen, A. Sinha, G. S. Paschos, and G. Iosifidis, "Optimistic no-regret algorithms for discrete caching," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 6, no. 3, pp. 48:1–48:28, 2021.

[35] N. Mhaisen, G. Iosifidis, and D. J. Leith, "Online caching with no regret: Optimistic learning via recommendations," *IEEE Trans. Mob. Comput.*, vol. 23, no. 5, pp. 5949–5965, 2024.

[36] K. Lam, "Dynamic cache replacement policy selection using experts," Master's thesis, Delft University of Technology, Netherlands, 2022.

[37] M. Mäkelä, "Adaptive caching with follow the perturbed leader replacement policy," Bachelor's thesis, Delft University of Technology, Netherlands, 2022.

[38] C. Finn, A. Rajeswaran, S. M. Kakade, and S. Levine, "Online meta-learning," in *ICML*, ser. Proceedings of Machine Learning Research, vol. 97. PMLR, 2019, pp. 1920–1930.

[39] J. Kivinen and M. K. Warmuth, "Additive versus exponentiated gradient updates for linear prediction," in *STOC*. ACM, 1995, pp. 209–218.

[40] F. Orabona, "A modern introduction to online learning," *CoRR*, vol. abs/1912.13213, 2019.

[41] L. Wang, S. Bayhan, and J. Kangasharju, "Optimal chunking and partial caching in information-centric networks," *Comput. Commun.*, vol. 61, pp. 48–57, 2015.

[42] M. K. Warmuth, "The blessing and the curse of the multiplicative updates," in *Discovery Science*, ser. Lecture Notes in Computer Science, vol. 6332. Springer, 2010, p. 382.

[43] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and zipf-like distributions: Evidence and implications," in *INFOCOM*. IEEE Computer Society, 1999, pp. 126–134.

[44] P. Gill, M. F. Arlitt, Z. Li, and A. Mahanti, "Youtube traffic characterization: a view from the edge," in *Internet Measurement Conference*. ACM, 2007, pp. 15–28.

[45] G. S. Paschos, G. Iosifidis, and G. Caire, "Cache optimization models and algorithms," *Found. Trends Commun. Inf. Theory*, vol. 16, no. 3-4, pp. 156–343, 2019.

2 Caching Expert Meta-Learner: https://github.com/maikdevries/Ceml