Towards inference delivery networks: distributing machine learning with optimality guarantees

Tareq Si Salem, Gabriele Castellano, Giovanni Neglia, Fabio Pianese, Andrea Araldo

Towards Inference Delivery Networks: Distributing Machine Learning with Optimality Guarantees

Tareq Si Salem*, Gabriele Castellano*†, Giovanni Neglia*, Fabio Pianese†, Andrea Araldo‡
*Inria, Université Côte d'Azur, France, {tareq.si-salem, gabriele.castellano, giovanni.neglia}@inria.fr,
†Nokia Bell Labs, France, {fabio.pianese, gabriele.castellano.ext}@nokia.com,
‡Télécom SudParis - Institut Polytechnique de Paris, France, andrea.araldo@telecom-sudparis.eu

***Abstract*—We present the novel idea of *inference delivery networks* (IDN), networks of computing nodes that coordinate to satisfy inference requests achieving the best trade-off between latency and accuracy. IDNs bridge the dichotomy between device and cloud execution by integrating inference delivery at the various tiers of the infrastructure continuum (access, edge, regional data center, cloud). We propose a distributed dynamic policy for ML model allocation in an IDN by which each node periodically updates its local set of inference models based on requests observed during the recent past plus limited information exchange with its neighbor nodes. Our policy offers strong performance guarantees in an adversarial setting and shows improvements over greedy heuristics with similar complexity in realistic scenarios.**

I. Introduction

Machine Learning (ML) models are often trained to perform inference, that is to elaborate predictions based on input data that need to be delivered to the final users. ML model training is a computationally and I/O intensive operation and its streamlining is the object of much research effort. Although inference does not involve complex iterative algorithms and is therefore generally assumed to be easy, it also presents fundamental challenges that are likely to become dominant as ML adoption increases [1]. In a future where AI systems are ubiquitously deployed and need to make timely and safe decisions in unpredictable environments, inferences will have to be served in real-time and the aggregate rate of predictions needed to support a pervasive ecosystem of sensing devices will become overwhelming.

Today two main deployment options for ML models are common: inferences can be served locally by the end devices (smartphones, IoT equipment, smart vehicles, etc.), where only simple models can run, or by a remote cloud infrastructure, where powerful "machine learning as a service" (MLaaS) solutions by the major cloud providers can serve inferences by sophisticated models at extremely high throughput.

However, there exist applications for which both options may be unsuitable: local models may have inadequate accuracy, while the cloud may fail to meet delay constraints. As an example, popular applications such as recommendation systems, voice assistants, and ad-targeting, need to serve predictions from ML models in less than 200 ms. Future wireless services, such as connected and autonomous cars, industrial robotics, mobile gaming, augmented/virtual reality, have even stricter latency requirements, often below 10 ms and in the order of 1 ms for what is known as the tactile Internet [2]. In enabling such strict latency requirements, the advent of Edge Computing plays a key role, as it deployes computational resources at the edge of the network (base stations, access points, ad-hoc servers). However, edge resources have limited capacity in comparison to the cloud and need to be wisely used. Therefore, integrating ML inference in the continuum between end devices and the cloud—passing through edge servers and regional micro data-centers—will require complex resource orchestration.

We believe that, to allocate resources properly, it will be crucial to study the trade-offs between accuracy, latency and resource-utilization, adapted to the requirements of the specific application. In fact, inference accuracy and, in general, resource efficiency increase toward the cloud, but so does communication latency. In this paper, we present the novel idea of *inference delivery networks* (IDN): networks of computing nodes that coordinate to satisfy inference requests achieving the best trade-off. An IDN may be deployed directly by the ML application provider, or by new IDN operators that offer their service to different ML applications, similarly to what happens for content delivery networks. The same inference task can be served by a set of heterogeneous models featuring diverse performance and resource requirements (e.g., different model architectures [3], multiple downsized version of the same pre-trained model [4], different configuration and execution setup). Therefore, we study the novel problem of how to deploy the available ML models on the available IDN nodes, where a deployment strategy consists in two coupled decisions: *(i)* where to place models for serving a certain task and *(ii)* how to select their size/complexity among the available alternatives.

In this paper, after defining a specific optimization problem and characterizing its complexity, we introduce INFIDA (INFerence Intelligent Distributed Allocation), a distributed dynamic allocation policy. Following this policy, each IDN node periodically updates its local allocation of inference models on the basis of the requests observed during the recent past and limited information exchange with its neighbors. The policy offers strong performance guarantees in an adversarial setting [5], that is a worst case scenario where the environment evolves in the most unfavorable way. Numerical experiments in realistic settings show that our policy outperforms heuristics with similar complexity. Our contributions are as follows:
(1) We present the novel idea of inference delivery networks.

(2) We frame the allocation of ML model in IDNs as an (NP-hard) optimization problem that captures the trade-off between latency and accuracy (Sec. III).
(3) We propose INFIDA, a distributed and dynamic allocation algorithm for IDNs (Sec. IV), and we show it provides strong guarantees in the adversarial setting (Sec. V).
(4) We evaluate INFIDA in a realistic simulation scenario and compare its performance with a greedy heuristic under different trade-off settings (Sec. VI).

II. Related Work

There is a vast literature on content placement [6] where objects, e.g., files or streams, can be stored (cached) into different nodes in order to reduce the operational cost of content delivery. The problem has been extended to the case of *service caching* (or placement), where an entire service can be offloaded onto nodes co-located with base-stations or mobile-micro clouds, engaging not only storage but also computational resources and energy [7], [8]. The problem considered in this paper differs from the above, as inference services can run under different configurations that lead to variable resource consumption and inference accuracy. In some sense, traditional services provide a binary answer to any user request: the request can be satisfied if and only if the service is deployed at the node. In ML inference, however, several models can provide an answer but the accuracy and the latency of the answer can be different [9].

A similar trade-off between resource usage and perceived quality typically emerges in the context of video caching. In [10]–[12] a same video can be cached into multiple network nodes at different qualities (or resolutions): the operator optimizes the user experience by jointly deciding the placement of videos and their quality. Video caching has however a far weaker dependency on latency, which is instead of paramount importance when placing interactive ML models, in particular for applications like augmented reality or autonomous driving. Moreover, in [10], [11] online placement is based on heuristic policies with no performance guarantees. The algorithm proposed in [12], instead, optimizes video placement in a snapshot of the system, assuming user demand is known in advance, while we propose a dynamic policy capable to adapt to varying requests in every time slot. We also prove strong performance guarantees, which are meant to hold in any condition thanks to our adversarial setting (Sec. V).

ML model allocation in an IDN can also be considered as a particular instance of *similarity caching* [13], a general model where items and requests can be thought as embedded in a metric space: edge nodes can store a set of items and requests, and the distance between a request and an item determines the quality of the matching between the two. Similarity caching was applied to a number of applications including contextual advertising, object recognition, and recommender systems (see [13] for specific references). To the best of our knowledge, the literature on similarity caching has restricted itself to *(i)* a single cache (with the exception of [14]–[16]), and *(ii)* homogeneous items with identical resource requirements. A consequence is that in our setting similarity caching policies would only allocate models based on their accuracy, ignoring the trade-offs imposed by their resource requirements.

A related set of works attempts to adapt inference to the capabilities of mobile hardware platforms through the principle of model splitting, a technique that distributes a ML model by partitioning its execution across multiple discrete computing units. Model splitting was applied to accommodate the hardware constraints in multi-processor mobile devices [17], to share a workload among mobile devices attached at the same network edge [18], and to partially offload inferences to a remote cloud infrastructure [19], possibly coupled with early exit strategies [20] and conditional hierarchical distributed deployment [21]. Model splitting is orthogonal to our concerns and could be accounted for in an enhanced IDN scheme.

There has been some work on ML model placement at the edge in the framework of what is called "AI on Edge" [22], but it considers a single intermediate tier between the edge device and the cloud, while we study general networks with nodes in the entire cloud-to-the-edge continuum. Our dynamic placement INFIDA algorithm could be applied also in this more particular setting, for example in the MODI platform [23]. The work closest to ours is [24], which proposes an online learning policy, with the premise of load balancing over a pool of edge devices while maximizing the overall accuracy. INFIDA has more general applicability.

Finally, VideoEdge [25] studies how to split the analytics pipeline across different computational clusters to maximize the average inference accuracy. Beside the focus on a specific application, the paper does not propose dynamic allocation placement algorithms.

III. Inference System Design

We consider a network of compute nodes, each capable of hosting some pre-trained ML models depending on its capabilities. Such ML models are used to serve inference requests for different classification or regression *tasks*.[1] Requests are generated by end devices and routed over given serving paths (e.g., from edge to cloud nodes). The goal of the system is to optimize the allocation of ML models across the network so that the aggregate serving cost is minimized. Our system model is detailed below.

A. Compute Nodes and Models

We represent the inference delivery network (IDN) as a weighted graph $G(\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of compute nodes, and $\mathcal{E}$ represents their interconnections. Each node $v \in \mathcal{V}$ is capable of serving inference tasks that are requested from anywhere in the network (e.g., from end-users, vehicles, IoT devices). We denote by $\mathcal{N}=\{1, 2, \ldots, N\}$ the set of tasks the system can serve (e.g., object detection, speech recognition, classification), and assume that each task $i \in \mathcal{N}$ can be served with different quality levels (e.g., different accuracy) and

[1] We are using the term task according to its meaning in the ML community, e.g., a task could be to recognize specific images.

TABLE I: Notation Summary.

$G(\mathcal{V},\mathcal{E})$	weighted graph, with nodes $\mathcal{V}$ and edges $\mathcal{E}$
$\mathcal{N}$ / $\mathcal{M}$	tasks / models catalog
s_m^v	size of model $m \in \mathcal{M}$ on node $v \in \mathcal{V}$
b^v	allocation budget constraint at node v
x_m^v	variable indicating model m is deployed on node v
ω_m^v	indicates model m is permanently deployed on node v
$\boldsymbol{p}$	routing path $\{p_1,\dots,p_J\}$ of connected nodes $p_j \in \mathcal{V}$
$\mathcal{R}$	set of request types $\rho=(i,\boldsymbol{p})$, $i \in \mathcal{N}$ and $\boldsymbol{p}$ is a path
$C^{p_j}_{\boldsymbol{p},m}$	cost of serving on p_j along path $\boldsymbol{p}$ using model m
r_ρ^t	number of time ρ is requested during time slot t
L_m^v	maximum capacity of model m on node v
$l_{\rho,m}^{t,v}$	potential capacity of m on node v for request ρ at t
γ_ρ^k	k-th smallest cost for request ρ along its path
λ_ρ^k	potential capacity of the model serving ρ with cost γ_ρ^k
z_ρ^k	effective capacity of the model serving ρ with cost γ_ρ^k
C / G	the overall system cost (9) / allocation gain (10)

different resources' requirements by a set of suitable models $\mathcal{M}_i=\{1,2,\dots,M_i\}$. We denote by $\mathcal{M}=\cup_{i\in\mathcal{N}}\mathcal{M}_i$ the catalog of all the available models. Note that each task is served by a separate set of models, i.e., $\mathcal{M}_i\cap\mathcal{M}_{i'}=\emptyset, \forall i,i' \in \mathcal{N}, i\neq i'$. Catalog $\mathcal{M}_i$ may encompass, for instance, independently trained models or shrunk versions of a high quality model generated through distillation [26], [27]. Finally, every model of the catalog may provide a different throughput (i.e., number of requests it can serve in a given time period), and therefore, support a different load (we formalize this in Sec. III-D).

For each compute node $v \in \mathcal{V}$, we denote by

$$x_m^v \in \{0,1\}, \text{ for } m \in \mathcal{M} \tag{1}$$

the decision variable that indicates if model $m \in \mathcal{M}$ is deployed on node v.[2] Therefore, $\boldsymbol{x}^v = [x_m^v]_{m\in\mathcal{M}}$ is the allocation vector on node v, and we use notation $\boldsymbol{x}$ to denote the global allocation decision.

We assume that the allocation of ML models at each node is constrained by a single resource dimension, potentially different at different nodes. A node could be, for instance, severely limited by the amount of available GPU memory, another by the maximum throughput in terms of instructions per second. The limiting resource determines the *allocation budget* $b^v \in \mathbb{R}_+$ at node $v \in \mathcal{V}$. We also denote with $s_m^v \in \mathbb{R}_+$ the size of model $m \in \mathcal{M}$ in terms of the limiting resource at node v. Therefore, budget constraints are expressed as

$$\sum_{m\in\mathcal{M}} x_m^v s_m^v \leq b^v, \forall v \in V. \tag{2}$$

To every task $i \in \mathcal{N}$, we associate a fixed set of *repository nodes* that always run at least a model capable of serving task i (e.g., high-performance models deployed in large data centers). Repositories ensure requests are satisfied even when the rest of the network is not hosting any additional model (there is no intermediate allocation).

We discern the repository models through constants $\omega_m^v \in \{0,1\}$, each indicating if model m is permanently deployed on node v. Then, we express the repository constraint as

$$x_m^v \geq \omega_m^v, \forall v \in \mathcal{V}, \forall m \in \mathcal{M}. \tag{3}$$

[2]Our formulation allows each node to host multiple copies of the same model to satisfy a larger number of request. This is captured by considering multiple distinct models with identical performance and requirements.

B. Inference Requests

We assume that every node has a predefined routing path towards a suitable repository node for each task $i \in \mathcal{N}$. The routing is therefore predefined, and our decisions only concern placement of models (i.e., variables x_m^v).

A routing path $\boldsymbol{p}$ of length $|\boldsymbol{p}| = J$ is a sequence $\{p_1,p_2,\dots,p_J\}$ of nodes $p_j \in \mathcal{V}$ such that edge $(p_j,p_{j+1}) \in \mathcal{E}$ for every $j \in \{1,2,\dots,J-1\}$. As in [28], we assume that paths are simple, i.e., they do not contain repeated nodes. A request is therefore determined by the pair $(i,\boldsymbol{p})$, where i is the task requested and $\boldsymbol{p}$ is the routing path to be traversed. We denote by $\mathcal{R}$ the set of all possible requests, and by $\mathcal{R}_i$ all possible requests for tasks i. When a request is propagated from node p_1 toward the associated repository node p_J, any intermediate node along the path that hosts a suitable model can serve it.

C. Cost Model

When serving request $\rho=(i,\boldsymbol{p}) \in \mathcal{R}$ on node p_j using model m, the system experiences an *inference cost* that depends on the quality of the model (i.e., on inference inaccuracy), its inference time, and the characteristics of node p_j. Additionally, the system experiences a *network cost*, due to using the path between p_1 and p_k. In general, we can write the total cost of serving a request as

$$C^{p_j}_{\boldsymbol{p},m} = f((p_1,\dots,p_j),m). \tag{4}$$

While our theoretical results hold under this very general cost model, in what follows—for the sake of concreteness—we refer to the following simpler model:

$$C^{p_j}_{\boldsymbol{p},m} = \sum_{j'=1}^{j-1} w_{p_{j'},p_{j'+1}} + d_m^{p_j} + \alpha(1-a_m) \tag{5}$$

where $w_{v,v'} \in \mathbb{R}_+$ is the (round-trip) latency of edge $(v,v') \in \mathcal{E}$, while $d_m^{p_j}$ and $(1-a_m)$ are respectively the inference delay and prediction inaccuracy of model m on node p_j. Parameter α weights the importance of accuracy w.r.t. latency.

Note that seeking cost minimization along a serving path usually leads to a trade-off: while the network cost always increases with j, in a typical network the service cost $d_m^{p_j}+\alpha(1-a_m)$ tends to decrease, as farther nodes (e.g., data centers) are better equipped and can run more accurate models.

D. Request Load and Serving Capacity

Let us assume that time is split in slots of equal duration. We consider a time horizon equal to T slots. During a slot t the system receives a batch of requests $\boldsymbol{r}_t = [r_\rho^t]_{\rho\in\mathcal{R}}$, where $r_\rho^t \in \mathbb{N}$ denotes the number of requests of type $\rho \in \mathcal{R}$.

Model $m \in \mathcal{M}$ has maximum capacity $L_m^v \in \mathbb{N}$ when deployed at node $v \in \mathcal{V}$, i.e., it can serve at most L_m^v requests during one time slot $t \in [T]$. We do not make specific assumptions on the time required to serve a request.

We denote by $l^{t,v}_{\rho,m}$ the *potential available capacity*, defined as the maximum number of type-ρ requests node v can serve at time t through model m, if m is deployed locally. The *effective available capacity* is then equal to $l^{t,v}_{\rho,m}x^v_m$. In general, $l^{t,v}_{\rho,m} \leq \min\{L^v_m, r^t_\rho\}$ because *(i)* the model can be busy processing other requests for the same task, and *(ii)* it cannot serve beyond what is requested at time t.

The vector of potential available capacities at time $t \in [T]$ is denoted by

$$\boldsymbol{l}_t = [l^{t,v}_{\rho,m}]_{(\rho,m,v)\in\bigcup_{i\in\mathcal{N}}\mathcal{R}_i\times\mathcal{M}_i\times\mathcal{V}}. \quad (6)$$

Note that computing $l^{t,v}_{\rho,m}$ requires to know the request arrival order, the scheduling discipline at node v, and the distribution of requests for other tasks across the different nodes. Our analysis in Sec. V considers a "pessimistic" scenario where both requests and available capacities are selected by an adversary. This approach relieves us from the need to model system detailed operations, while our proposed algorithm (Sec. IV) benefits from strong guarantees in the adversarial setting. In what follows, we can then consider that the vector $\boldsymbol{l}_t$ is exogeneously determined.

To make sure that any request can always be served on the associated repository node, we assume that the repository allocation is never saturated for any possible request load, i.e., for every $t \in [T]$ and task $i \in \mathcal{N}$ we have

$$\sum_{\rho\in\mathcal{R}_i} r^t_\rho \leq \sum_{m\in\mathcal{M}_i} \omega^v_m l^{t,v}_{\rho,m}, \text{ for all } v \in \mathcal{V}\colon \sum_{M\in\mathcal{M}_i} \omega^v_m \geq 1. \quad (7)$$

Without loss of generality, we assume that a repository node for a given task stores a single model with unlimited capacity (it represents a pool of high quality models).

E. Serving Model

Let $K_\rho = |\boldsymbol{p}||\mathcal{M}_i|$ denote the maximum number of models that request $\rho{=}(i,\boldsymbol{p}){\in}\mathcal{R}$ may encounter along its serving path $\boldsymbol{p}$. We order the corresponding costs $\{C^{p_j}_{\boldsymbol{p},m}, \forall m \in \mathcal{M}_i, \forall p_j \in \boldsymbol{p}\}$ in increasing order and we denote by $\kappa_\rho(v,m)$ the order of model m allocated at node v. If $v \notin \boldsymbol{p}$ we have $\kappa_\rho(v,m) = \infty$.

If $\kappa_\rho(v,m) = k$, then model m at node v has the k-th smallest cost to serve request ρ. We denote the model service cost, its potential available capacity, and its effective capacity as γ^k_ρ, $\lambda^k_\rho(\boldsymbol{l}_t)$, and $z^k_\rho(\boldsymbol{l}_t,\boldsymbol{x})$, respectively:

$$\gamma^k_\rho = C^v_{\boldsymbol{p},m}, \quad \lambda^k_\rho(\boldsymbol{l}_t) = l^{t,v}_{\rho,m}, \quad z^k_\rho(\boldsymbol{l}_t,\boldsymbol{x}) = x^v_m l^{t,v}_{\rho,m}. \quad (8)$$

We assume the IDN serves requests as follows. Each request is forwarded along its serving path and served when it encounters a model with the smallest serving cost among those that are not yet saturated.

Since models do not necessarily provide increasing costs along the path, this serving strategy requires that a node that runs a model $m \in \mathcal{M}_i$ and receives a request for task i, knows if there are better alternatives for serving task i upstream or not. In the first case, it will forward the request along the path, otherwise it will serve it locally. We argue that, in a real system, this partial knowledge can be achieved with a limited number of control messages. In fact, if node $p_h \in \boldsymbol{p}$ hosts the model with the k-th cost for request $(i,\boldsymbol{p})$, it only needs information about those models that *(i)* are located upstream on the serving path (i.e., on nodes $p_l \in \boldsymbol{p}$ with $l > h$), and *(ii)* provide a cost smaller than γ^k_ρ. Since the cost increases with the network latency (see (5)), the number of models satisfying these criteria is small in practice.[3] A node needs to propagate downstream a control message with the information about the requests it can serve and the corresponding costs. Nodes forwarding the control message progressively remove the information about the tasks they can serve with a smaller cost, until the control message payload is empty and the message can be dropped. These messages should be sent whenever the availability to serve additional requests changes.

According to the presented serving strategy, the requests load is split among the currently available models giving priority to those that provide the smallest serving costs up to their saturation. In particular, the model with the k-th smallest cost will serve some requests of type ρ only if the less costly models have not been able to satisfy all of them (i.e., if $\sum_{k'=1}^{k-1} z^{k'}_\rho(\boldsymbol{l}_t,\boldsymbol{x}) < r^t_\rho$). If this is the case, it will serve with cost γ^k_ρ at most $z^k_\rho(\boldsymbol{l}_t,\boldsymbol{x})$ requests (its effective available capacity) out of the $r^t_\rho - \sum_{k'=1}^{k-1} z^{k'}_\rho(\boldsymbol{l}_t,\boldsymbol{x})$ requests still to be satisfied. The aggregate cost incurred by the system at time slot t is then given by

$$C(\boldsymbol{r}_t,\boldsymbol{l}_t,\boldsymbol{x}) = \sum_{\rho\in\mathcal{R}}\sum_{k=1}^{K_\rho}\gamma^k_\rho \min\left\{r^t_\rho - \sum_{k'=1}^{k-1} z^{k'}_\rho(\boldsymbol{l}_t,\boldsymbol{x}),\, z^k_\rho(\boldsymbol{l}_t,\boldsymbol{x})\right\} \cdot \mathbb{1}_{\left\{\sum_{k'=1}^{k-1} z^{k'}_\rho(\boldsymbol{l}_t,\boldsymbol{x}) < r^t_\rho\right\}}. \quad (9)$$

F. Allocation Gain and Static Optimal Allocations

We are interested in model allocations that minimize the aggregate cost (9), or, equivalently, that maximize the *allocation gain* defined as

$$G(\boldsymbol{r}_t,\boldsymbol{l}_t,\boldsymbol{x}) = C(\boldsymbol{r}_t,\boldsymbol{l}_t,\boldsymbol{\omega}) - C(\boldsymbol{r}_t,\boldsymbol{l}_t,\boldsymbol{x}). \quad (10)$$

The first term $C(\boldsymbol{r}_t,\boldsymbol{l}_t,\boldsymbol{\omega})$ on the right hand side is the service cost when only repositories are present in the network. Since intermediate nodes can help serving the requests at a reduced cost, $C(\boldsymbol{r}_t,\boldsymbol{l}_t,\boldsymbol{\omega})$ is an upper bound on the aggregate serving cost, and the allocation gain captures the cost reduction achieved by model allocation $\boldsymbol{x}$.

The static model allocation problem can then be formulated as finding the model allocation $\boldsymbol{x}^*$ that maximizes the sum of the allocation gains over the time horizon $[1,T]$, i.e., $\boldsymbol{x}^* = \arg\max\sum_t G(\boldsymbol{r}_t,\boldsymbol{l}_t,\boldsymbol{x})$ subject to the budget constraints (2). This is a submodular maximization problem under multiple knapsack constraints, which is NP-hard. It is known that submodular maximization problems can not be approximated with a ratio better than $(1{-}1/e)$ even under simpler cardinality constraints [29]. Under the multi-knapsack constraint, it is possible to solve the offline problem achieving a $(1{-}1/e{-}\epsilon)$-approximation through a recent algorithm proposed in [30].

[3]In realistic settings (Sec. VI), we experienced that each deployed model has at most 6 better alternatives on upstream nodes (worst case with $\alpha{=}1$).

Algorithm 1 INFIDA distributed allocation on node v

1: **for** $t = 1, 2, \ldots, T$ **do**
2: Compute $\boldsymbol{g}^v \in \partial_{\boldsymbol{y}_t^v} G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y}_t)$ through (12)
3: $\hat{\boldsymbol{y}}_t^v \leftarrow \nabla\Phi(\boldsymbol{y}_t^v)$ ▷ Map state to dual space
4: $\hat{\boldsymbol{h}}_{t+1}^v \leftarrow \hat{\boldsymbol{y}}_t^v + \eta_t \boldsymbol{g}_t^v$ ▷ Take gradient step in the dual space
5: $\boldsymbol{h}_{t+1}^v \leftarrow (\nabla\Phi)^{-1}(\hat{\boldsymbol{h}}_{t+1}^v)$ ▷ Map dual state back to the primal space
6: $\boldsymbol{y}_{t+1}^v \leftarrow \mathcal{P}_{\mathcal{Y}^v}^{\Phi}(\boldsymbol{h}_{t+1}^v)$ ▷ Proj. new state onto the feasible region
7: $\boldsymbol{x}_{t+1}^v \leftarrow \mathrm{DepRound}(\boldsymbol{y}_{t+1}^v)$ ▷ Sample a discrete allocation

We conclude by providing a useful alternative formulation of the allocation gain (the proof is in [31]):

Lemma III.1. *The allocation gain* (10) *has the following equivalent expression:*

$$G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x}) = \sum_{\rho\in\mathcal{R}} \sum_{k=1}^{K_\rho - 1} \left(\gamma_\rho^{k+1} - \gamma_\rho^k\right) \cdot \min\left\{ r_{t,\rho} - \beta_\rho^{t,k}, \sum_{k'=1}^{k} z_\rho^{k'}(\boldsymbol{l}_t, \boldsymbol{x}) - \beta_\rho^{t,k} \right\}, \tag{11}$$

where $\beta_\rho^{t,k} := \min\{r_{t,\rho}, \sum_{k'=1}^{k} z_\rho^{k'}(\boldsymbol{l}_t, \boldsymbol{\omega})\}$.

IV. INFIDA Algorithm

In this section, we propose INFIDA, an online algorithm that can operate in a distributed fashion without requiring global knowledge of the allocation state and requests arrival. In Sec. V, we show that INFIDA converges to an allocation within a $(1 - 1/e)$-approximation from the optimum.

A. Algorithm Overview

On every node $v \in \mathcal{V}$, INFIDA updates the allocation $\boldsymbol{x}^v \in \mathcal{X}^v = \{0,1\}^{|\mathcal{M}|}$ operating on a correspondent fractional state $\boldsymbol{y}^v \in \mathcal{Y}^v = [0,1]^{|\mathcal{M}|}$. Each variable y_m^v can be interpreted as the probability of hosting model m on node v, i.e., $y_m^v = \mathbb{P}[x_m^v = 1] = \mathbb{E}[x_m^v]$.

Note that $G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y})$ is a concave function of variable $\boldsymbol{y} \in \mathcal{Y} = \bigtimes_{v\in\mathcal{V}} \mathcal{Y}^v$. Indeed, (11) shows that $G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y})$ is a linear combination, with positive coefficients, of concave functions (the minimum of affine functions in $\boldsymbol{y}$).

Within a time slot t, node v collects measurements from messages that have been routed through it (Sec. IV-B). At the end of every time slot, the node *(i)* computes its new fractional state $\boldsymbol{y}^v$, and *(ii)* updates its local allocation $\boldsymbol{x}^v$ accordingly. INFIDA is summarized in Algorithm 1 and detailed below.

State computation. The fractional state $\boldsymbol{y}^v$ is updated through an iterative procedure aiming to maximize $G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y})$. This could be the standard gradient ascent method, which updates the fractional state at each node as $\boldsymbol{y}_{t+1}^v = \boldsymbol{y}_t^v + \eta_t \boldsymbol{g}_t^v$, where $\eta_t \in \mathbb{R}_+$ is the step size and $\boldsymbol{g}_t^v$ is a subgradient of $G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y})$ with respect to $\boldsymbol{y}^v$. In our work, we use a generalized version of the gradient method called Online Mirror Ascent (OMA) [32, Ch. 4]. OMA uses a function Φ (mirror map) to map $\boldsymbol{y}$ to a dual space before applying the gradient ascent method (lines 3-5 of Algorithm 1). OMA reduces to the classic gradient ascent method if Φ is the squared Euclidean norm. Instead, we use the weighted negative entropy map $\Phi(\boldsymbol{y}^v) = \sum_{m\in\mathcal{M}} s_m^v y_m^v \log(y_m^v)$, which is known to achieve better convergence rate when optimizing functions with similar properties to G [32, Sect. 4.3].

To compute a feasible fractional state $\boldsymbol{y}^v$, we then perform a projection to the set $\mathcal{Y}^v$ of fractional allocations that satisfy the capacity constraint (2) on node v (line 6 of Algorithm 1). We adapt the projection algorithm from [33] to obtain a negative entropy projection $\mathcal{P}_{\mathcal{Y}^v}^{\Phi}$. Our adaptation is described in [31].

Allocation update. Once the fractional state $\boldsymbol{y}^v$ has been updated, the final step of INFIDA is to determine a new random discrete allocation $\boldsymbol{x}^v$ and update the local models accordingly. The sampled allocation $\boldsymbol{x}^v$ should *(i)* comply with the budget constraint (2) on node v and *(ii)* be consistent with the fractional state, i.e., $\mathbb{E}[x_m^v] = y_m^v \ \forall m \in \mathcal{M}$. In particular, we use the DepRound [34] subroutine (line 7 of Algorithm 1).

In the remainder of this section we detail how each node computes its contribution to the global subgradient, and the rounding strategy used to determine the discrete allocation.

B. Subgradient Computation

At the end of every time slot t, each node $v \in \mathcal{V}$ computes $\boldsymbol{g}_t^v$, a subgradient of $G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y}_t)$ with respect to $\boldsymbol{y}^v$, i.e., $\boldsymbol{g}_t^v \in \partial_{\boldsymbol{y}^v} G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y}_t)$, using the following expression (see [31]).

$$\begin{aligned} g_{t,m}^v &= \sum_{\rho\in\mathcal{R}} \sum_{k=\kappa_\rho(v,m)}^{K_\rho - 1} \lambda_\rho^k(\boldsymbol{l}_t) \left(\gamma_\rho^{k+1} - \gamma_\rho^k\right) \cdot \mathbb{1}_{\left\{\sum_{k'=1}^{k} z_\rho^{k'}(\boldsymbol{l}_t, \boldsymbol{y}_t) < r_\rho^t\right\}} \\ &= \sum_{\rho\in\mathcal{R}} \sum_{k=\kappa_\rho(v,m)}^{K_\rho^*(\boldsymbol{y}_t) - 1} \lambda_\rho^k(\boldsymbol{l}_t) \left(\gamma_\rho^{k+1} - \gamma_\rho^k\right), \ \forall m \in \mathcal{M}, \end{aligned} \tag{12}$$

where $K_\rho^*(\boldsymbol{y}_t)$ is the order of the last model that would be needed to serve all request batch r_ρ^t in the fractional state $\boldsymbol{y}_t$, i.e., $K_\rho^*(\boldsymbol{y}_t) := \min\left\{k \in [K_\rho - 1] : \sum_{k'=1}^{k} z_\rho^{k'}(\boldsymbol{l}_t, \boldsymbol{y}_t) \geq r_\rho^t\right\}$.

This subgradient can be computed at each node using only information from the control messages collected at the end of the time slot t. Consider a request of type $\rho = (i, \boldsymbol{p})$. The first node of the path, p_1, generates a control message with the total number r_ρ^t of type-ρ requests received during the slot, and sends the message along the path $\boldsymbol{p}$. Each node traversed, p_1 included, adds its cost and effective capacity, in the fractional state, for each model $m \in \mathcal{M}_i$. The message needs to travel upstream until it the model with the $K_\rho^*(\boldsymbol{y}_t)$-th smallest cost is detected, let us say at node p_h. Node p_h can then generate a new control message with the same payload and forward it in the reverse direction back to node p_1. Each node along the path has then all information needed to compute its own subgradient following (12). It is possible to significantly reduce the size of the control messages using opportune partial aggregation, which we detail in [31].

C. State Rounding

Once the new fractional state $\boldsymbol{y}_{t+1}$ is computed, each node v independently draws a random set of models to store locally in such a way that $\mathbb{E}[\boldsymbol{x}_{t+1}^v] = \boldsymbol{y}_{t+1}^v$. This sampling guarantees that the final allocation $\boldsymbol{x}_{t+1}^v$ satisfies constraint (2) in expectation. A naive approach is to draw each variable $x_m^{v,t+1}$ independently, but it leads to a large variance of the

total size of the models selected, potentially exceeding by far the allocation budget at node v.

To construct a suitable allocation we adopt the DepRound procedure from [34]. The procedure modifies the fractional state $\boldsymbol{y}^v_{t+1}$ iteratively: at each iteration, DepRound operates on two fractional variables $y^{v,t+1}_m, y^{v,t+1}_{m'}$ so that at least one of them becomes integral and the aggregate size of the corresponding models $s^v_m y^{v,t+1}_m + s^v_{m'} y^{v,t+1}_{m'}$ does not change. This operation is iterated until all the variables are rounded except (at most) one. This is done in $\mathcal{O}(|\mathcal{M}|)$ steps.

Note that, to satisfy $\mathbb{E}[\boldsymbol{x}^v_{t+1}] = \boldsymbol{y}^v_{t+1}$, the residual fractional variable, say it $y^{v,t+1}_{\bar{m}}$, needs to be rounded. At this point $x^{v,t+1}_{\bar{m}}$ can be randomly drawn. Now the final allocation can exceed the budget bound b^v by at most $s_{\bar{m}}$. In practice, we can cope with the dangling variable adopting several heuristics, e.g., deploy the model that provides the best marginal gain among those that fit the constraint, or admit elastic budgets.

V. Theoretical Guarantees

We provide the optimality guarantees of our INFIDA algorithm in terms of the ψ-regret [35]. In our scenario, the ψ-regret is defined as the gain loss in comparison to the best static allocation in hindsight $\boldsymbol{x}^* \in \arg\max_{\boldsymbol{x}\in\mathcal{X}} \sum_{t=1}^T G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x})$, discounted by a factor $\psi \leq 1$. Formally,

$$\psi\text{-Regret}_{T,\mathcal{X}} = \sup_{\{\boldsymbol{r}_t,\boldsymbol{l}_t\}_{t=1}^T \in (\mathcal{B}\times\mathcal{L})^T} \left\{ \psi \sum_{t=1}^T G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x}^*) - \mathbb{E}\left[\sum_{t=1}^T G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x}_t)\right] \right\},$$

where allocations $\boldsymbol{x}_t$ are computed using INFIDA and the expectation is over the randomized choices of `DepRound`, while $\mathcal{B}$ and $\mathcal{L}$ are respectively the set of all possible request batches and potential available capacities, i.e., $\boldsymbol{r}_t \in \mathcal{B}$ and $\boldsymbol{l}_t \in \mathcal{L}, \forall t \in [T]$. Note that, by taking the supremum over all the potential available capacity and request sequences, we measure regret in an adversarial setting, i.e., against an adversary that selects, for every $t \in [T]$, vectors $\boldsymbol{r}_t$ and $\boldsymbol{l}_t$ to jeopardize the performance of our algorithm.

The adversarial analysis is a modeling technique to characterize system performance under highly volatile external parameters (e.g., the sequence of requests $\boldsymbol{r}_t$) or difficult to model system interactions (e.g., the available capacities $\boldsymbol{l}_t$). This technique has been recently successfully used to model caching problems (e.g., in [33], [36]). Our main result is the following (the full proof is in [31]):

Theorem V.1. INFIDA *has a sublinear* $(1-1/e)$*-regret in the sequence of requests and potential capacities, i.e., there exists a constant* A *such that:*

$$(1-1/e)\text{-Regret}_{T,\mathcal{X}} \leq A\sqrt{T}, \tag{13}$$

where $A \propto RL_{\max}\Delta_C$. R *and* $L_{\max}$ *are upper bounds, respectively, on the request batch size at any time slot and on the model capacities, while* Δ_C *is the largest cost difference between serving at a repository node and at a source node.*

Proof. (sketch) We first prove that the expected gain of the randomly sampled allocations $\boldsymbol{x}_t$ is a $(1-1/e)$-approximation of the fractional gain. Then, we use online learning results [32] to bound the regret of Online Mirror Ascent schemes operating on a convex decision space and against concave gain functions picked by an adversary. The two results are combined to obtain an upper bound on the $(1-1/e)$-regret. □

We fully characterize the regret constant A in [31].

We observe that the regret bound depends crucially on the maximum batch size R, maximum model capacity $L_{\max}$ and maximum serving cost difference Δ_C. When considering the cost model in Eq. (5), Δ_C is given by the total latency of the longest path plus the largest inference delay difference between the repository and the most constrained nodes. This result is intuitive: when these values are bigger, the adversary has a larger room to select values that can harm the performance of the system.

As a direct consequence of Theorem V.1, the expected time-average $(1-1/e)$-regret of INFIDA can get arbitrarily close to zero for large time horizon. Hence, INFIDA achieves a time-average expected gain that is a $(1-1/e-\epsilon)$-approximation of the optimal static gain, for arbitrarily small ϵ. This observation also suggests that our algorithm can be used to solve the NP-hard static allocation problem with the best approximation bound achievable for this kind of problems [29]. This result is formally stated by the following corollary:

Corollary V.1.1. *(offline solution) Let* $\bar{\boldsymbol{y}}$ *be the average fractional allocation* $\bar{\boldsymbol{y}} = \frac{1}{T}\sum_{t=1}^T \boldsymbol{y}_t$ *of* INFIDA*, and* $\bar{\boldsymbol{x}}$ *the random state sampled from* $\bar{\boldsymbol{y}}$ *using* `DepRound`. $\forall \epsilon > 0$ *and over a sufficiently large time horizon* T*,* $\bar{\boldsymbol{x}}$ *satisfies*

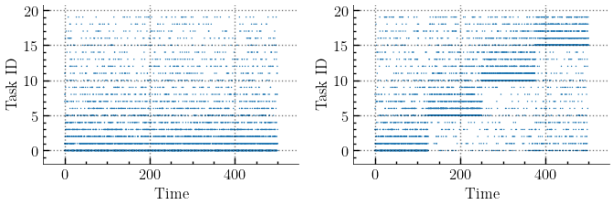
$$\mathbb{E}\left[\tfrac{1}{T}\sum_{t=1}^T G(\boldsymbol{r}_t, \boldsymbol{l}_t, \bar{\boldsymbol{x}})\right] \geq \left(1 - \tfrac{1}{e} - \epsilon\right) \tfrac{1}{T}\sum_{t=1}^T G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x}^*).$$

VI. Experimental Results

We evaluate INFIDA simulating a realistic scenario based on the typical structure of ISP networks. We compare our solution with a greedy policy (described below), as the greedy heuristic is known to achieve good performance in practice for submodular optimization [35].

Topology. We simulate a hierarchical topology similar to [37] that spans between edge and cloud, with different capacities at each tier. We consider 5 tiers: base stations (tier 4), each featuring a single low-budget server; central offices (tiers 3, 2), with tens of servers at each node; ISP data center (tier 1), featuring hundreds of servers; a remote cloud (tier 0), with practically unlimited capacity. We assume a hierarchical geographic distribution similar to LTE, with RTTs ranging from few milliseconds between tiers 2, 3, and 4, to tens of milliseconds between tiers 0, 1, and 2. The average RTT from base stations to cloud is ≈50 ms. We scale our experiments to a scenario with 60 base stations (86 nodes in total).

Processing Units. We simulate the performance of two different processing units. We assume tier 0 is equipped with high end GPUs (Titan RTX), while other tiers are equipped with



(a) Fixed Popularity Profile (b) Sliding Popularity Profile

Fig. 1: Popularity profiles of inference tasks. In (b), popularity changes at fixed time intervals through a cyclic shift. At any moment the requests are i.i.d. and sampled from a Zipf distribution with exponent 1.2.

mid-range GPUs (GeForce GTX 980). However, we limit the available GPU memory on 75% of the base stations to 1 GB.

Catalog and requests. We consider a catalog with 20 different object detection tasks, and 10 alternative models per task. We simulate performance based on state-of-the-art pre-trained models and their pruned versions [38], [39], profiled for each simulated processing unit (Table 2 in [31]). The popularity of requested tasks changes over time through a cyclic shift (Fig. 1). In our simulations, requests are first received from the base stations, and we submit to the system up to 15000 requests per second (rps).

Greedy heuristic. The greedy heuristic is adapted from the cost-benefit greedy [35]. A node v uses counters $r^v_{m,\rho}$ to keep track of the number of times a request $\rho \in \mathcal{R}$ is forwarded upstream but could have been served locally at a lower cost compared to the repository, i.e., using any model $m \in \mathcal{M}$ that introduces a positive gain $g^v_{m,\rho}$. For every model m, an importance weight is computed as $w^v_m = \frac{1}{s^v_m}\frac{1}{|\mathcal{R}|}\sum_{\rho\in\mathcal{R}} g^v_{m,\rho}\min\{r^v_{m,\rho}, L^v_m\}$, where s^v_m is the size of model m and $\min\{r^v_{m,\rho}, L^v_m\}$ is the number of requests that could have been improved by m. The node selects the model m_* with the highest importance while respecting the resource budget constraint, then subtracts the quantity $\min\{r^v_{m,\rho}, L^v_m\}$ from $r^v_{m_*,\rho}$ and from and all the $r^v_{m',\rho}$: $g^v_{m',\rho} < g^v_{m_*,\rho}$, i.e., models that provide a gain lower than m_*. This procedure is repeated until the resource budget of the node is consumed.

Performance Metric. The performance of a policy $\mathcal{P}$ is evaluated in terms of the time averaged gain normalized to the number of requests per second (NTAG); The NTAG evaluated over T requests is defined as:

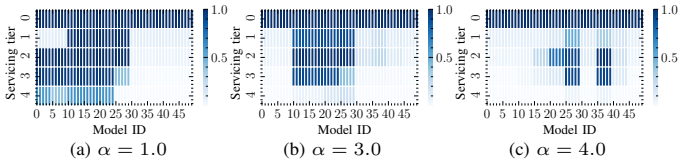
$$\mathrm{NTAG}(\mathcal{P}) = \sum_{t=1}^{T} \frac{G_{\mathcal{P}}(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x}_t)}{T\|\boldsymbol{r}_t\|_1}. \tag{14}$$

A. *Trade-off between Latency and Accuracy*

We first evaluate how INFIDA adapts to different trade-offs between end-to-end latency and inference accuracy, varying the α parameter in the request cost (5).

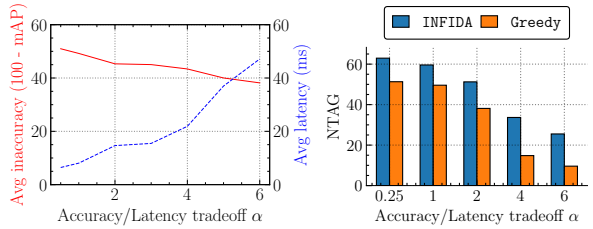
Fig. 2 shows the average allocation decision at each tier of the topology for different values of α. Models are ordered horizontally by increasing accuracy with 5 potential replicas for each model. Note that tier 0 acts as repository and then its allocation is fixed. For this set of experiments, we submit a load of 5000 rps.

For $\alpha = 1$ (Fig. 2c), INFIDA allocates a considerable amount of small models (which provide low accuracy) near the edge (i.e., tiers 2-4), as they can serve a large number of requests. By giving more importance to the accuracy ($\alpha = 3$) the system tends to avoid serving the requests at the very edge, also increasing the probability to deploy more accurate models (Fig. 2c). For $\alpha = 4$, also the number of models deployed on tiers 1-3 drastically decreases, as the system selects high quality models that quickly consume the resources of the nodes. The few allocations on tiers 1-3 also suggests that a significant amount of requests is served in the cloud, being latency less important.



(a) $\alpha = 1.0$ (b) $\alpha = 3.0$ (c) $\alpha = 4.0$

Fig. 2: Average allocation decisions y^v_m of INFIDA on the various tiers of the network topology, under Fixed Popularity profile. The model IDs are sorted for increasing accuracy. Values are averaged over all nodes in each tier.



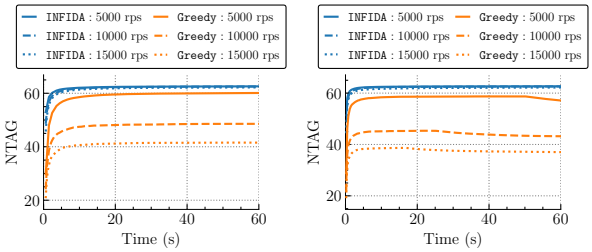
(a) (b)

Fig. 3: For different values of α: (a) average latency (dashed line) and inaccuracy (solid line) costs experienced with INFIDA; (b) NTAG of INFIDA and Greedy.

For better visualizing the trade-off, we describe below a set of results obtained in a topology with smaller nodes (up to 5 times), which allowed us to stress the system even with a moderate load of 5000 requests per second (Fig 3).

Fig. 3a shows the average experienced inaccuracy and latency for different values of α. When the accuracy is not important (i.e., $\alpha \approx 0$), INFIDA effectively achieves very low end-to-end latency (few milliseconds), by prioritizing the deployment of small and inaccurate models on the edge nodes. Noticeably, the kink in both curves at $\alpha \approx 4$ suggests that, when a higher accuracy is required, the system starts to prefer models on the cloud, leading to a sudden change in the trade-off and to a significant increase in latency.

In Fig. 3b we show the time averaged gain of INFIDA compared to the Greedy policy for different values of α. The gain is normalized to the number of requests per second. The plot shows that the gain decreases by increasing α. This is expected since the gain (10) is defined as the improvement w.r.t. the repository allocation (tier 0). Therefore, when the latency is not important, high accuracy models at tier 0 are preferred, and there is no much room for improvement (the optimal gain eventually tends to zero for $\alpha \to +\infty$). Noticeably, in this situation our algorithm performs much better than Greedy (the gain is up to 2.5 times higher). This suggests that INFIDA does



(a) *Fixed Popularity* profile (b) *Sliding Popularity* profile

Fig. 4: NTAG of INFIDA and Greedy for different request loads.

a better job in spotting the few remaining alternatives that can reduce the cost compared to the repository.

B. Scalability on Requests Load

Last, we show how the system performs under different requests loads. For this set of experiments, we set $\alpha = 0.5$. Fig. 4 shows that, in general, INFIDA provides a higher gain compared to the Greedy heuristic.

In particular, while Greedy performs similar to INFIDA for 5000 rps, it quickly deteriorates when the load increases, reducing its average gain by $\approx$33% for 15000 rps. It is also noteworthy that Greedy has visibly perturbed performance when the popularity of the tasks changes over time (Fig. 4b).

On the other hand, results show that INFIDA preserves its performance providing the same Normalized Time-Averaged Gain for all the analyzed request loads and under both Fixed and Sliding Popularity.

VII. Conclusions

In this paper, we introduced the idea of inference delivery networks (IDNs), networks of computing nodes that coordinate to satisfy inference requests in the continuum between Edge and Cloud. IDN nodes can serve inference requests with different levels of accuracy and end-to-end delay, based on their geographic location and processing capabilities. We formalized the NP-hard problem of allocating ML models on IDN nodes, capturing the trade-off between latency and accuracy. We proposed INFIDA, a dynamic ML model allocation algorithm that operates in a distributed fashion and provides strong guarantees in an adversarial setting. We evaluated INFIDA simulating the realistic scenario of an ISP network, and compared its performance with a greedy heuristic. Our results show that INFIDA adapts to different latency/accuracy trade-offs and scales well with the number of requests, outperforming the greedy policy in all the analyzed settings.

Acknowledgement. This work has been carried out in the framework of a common lab agreement between Inria and Nokia Bell Labs. G. Neglia acknowledges support from Inria under the exploratory action MAMMALS.

References

[1] I. Stoica *et al.*, "A berkeley view of systems challenges for ai," *arXiv:1712.05855*, 2017.
[2] M. Simsek *et al.*, "5g-enabled tactile internet," *IEEE JSAC*, vol. 34, no. 3, pp. 460–473, 2016.
[3] A. G. Howard *et al.*, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv:1704.04861*, 2017.
[4] L. Deng *et al.*, "Model compression and hardware acceleration for neural networks: A comprehensive survey," *Proc. of the IEEE*, 2020.
[5] G. Goel *et al.*, "Beyond Online Balanced Descent: An Optimal Algorithm for Smoothed Online Optimization," *NIPS*, 2019.
[6] X. Qiu *et al.*, "Cost-Minimizing Dynamic Migration of Content Distribution Services into Hybrid Clouds," *IEEE Trans. Par.Distr.Sys.*, 2015.
[7] J. Xu *et al.*, "Joint service caching and task offloading for mobile edge computing in dense networks," in *IEEE INFOCOM*, 2018, pp. 207–215.
[8] S. Wang *et al.*, "Dynamic Service Placement for Mobile Micro-Clouds with Predicted Future Costs," *IEEE Trans. Par.Distr.Sys.*, 2017.
[9] A. Ben Ameur *et al.*, "On the Deployability of Augmented Reality Using Embedded Edge Devices," in *IEEE CCNC*, 2021.
[10] A. Araldo *et al.*, "Representation Selection Problem : Optimizing Video Delivery through Caching," in *IFIP Netw.*, 2016.
[11] M. Choi *et al.*, "Wireless Video Caching and Dynamic Streming Under Differentiated Quality Requirements," *IEEE JSAC*, vol. 36, 2018.
[12] Z. Qu *et al.*, "Cooperative caching for multiple bitrate videos in small cell edges," *IEEE Trans.Mob.Comp.*, vol. 19, no. 2, pp. 288–299, 2020.
[13] M. Garetto *et al.*, "Similarity Caching: Theory and Algorithms," in *IEEE INFOCOM*, 2020. [Online]: https://hal.inria.fr/hal-02411268
[14] P. Sermpezis *et al.*, "Soft cache hits: Improving performance through recommendation and delivery of related content," *IEEE JSAC*, 2018.
[15] J. Zhou *et al.*, "Adaptive offline and online similarity-based caching," *IEEE Networking Letters*, vol. 2, no. 4, pp. 175–179, 2020.
[16] M. Garetto *et al.*, "Content placement in networks of similarity caches," *arXiv:2102.04974*, 2021.
[17] N. D. Lane *et al.*, "Deepx: A software accelerator for low-power deep learning inference on mobile devices," in *ACM/IEEE IPSN*, 2016.
[18] N. Fernando *et al.*, "Computing with nearby mobile devices: A work sharing algorithm for mobile edge-clouds," *IEEE Trans. Cloud Computing*, vol. 7, no. 2, pp. 329–343, 2019.
[19] Y. Kang *et al.*, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGARCH Comp.Arch.News*, 2017.
[20] S. Teerapittayanon *et al.*, "Branchynet: Fast inference via early exiting from deep neural networks," in *Int.Conf.Pattern Recog. (ICPR)*, 2016.
[21] S. Teerapittayanon *et al.*, "Distributed deep neural networks over the cloud, the edge and end devices," in *IEEE ICDCS*, 2017.
[22] S. Deng *et al.*, "Edge intelligence: The confluence of edge computing and artificial intelligence," *IEEE IoT J.*, 2020.
[23] S. S. Ogden *et al.*, "MODI: Mobile deep inference made efficient by edge computing," in *USENIX HotEdge 2018*, 2018.
[24] Y. Jin *et al.*, "Provisioning Edge Inference as a Service via Online Learning," in *IEEE SECON*, 2020.
[25] C.-C. Hung *et al.*, "Videoedge: Processing camera streams using hierarchical clusters," in *IEEE/ACM Symposium on Edge Computing*, 2018.
[26] G. Hinton *et al.*, "Distilling the knowledge in a neural network," *preprint arXiv:1503.02531*, 2015.
[27] S. Ravi, "Custom On-Device ML Models with Learn2Compress," 2018.
[28] S. Ioannidis *et al.*, "Adaptive caching networks with optimality guarantees," *SIGMETRICS Perform. Eval. Rev*, vol. 44, pp. 113–124, 2016.
[29] G. L. Nemhauser *et al.*, "Best algorithms for approximating the maximum of a submodular set function," *Math. of Op. Res.*, 1978.
[30] Y. Fairstein *et al.*, "A (1-1/e-ε)-approximation for the monotone submodular multiple knapsack problem," in *Euro. Symp. on Algorithms*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.
[31] T. Si Salem *et al.*, "Towards inference delivery networks: Distributing machine learning with optimality guarantees," *arXiv:2105.02510*, 2021.
[32] S. Bubeck, "Convex optimization: Algorithms and complexity," *Found. Trends Mach. Learn.*, vol. 8, no. 3-4, pp. 231–357, Nov. 2015.
[33] T. Si Salem *et al.*, "No-Regret Caching via Online Mirror Descent," in *IEEE International Conference on Communications (ICC)*, 2021.
[34] J. Byrka *et al.*, "An improved approximation for k-median, and positive correlation in budgeted optimization," in *ACM-SIAM symposium on Discrete algorithms*, 2014.
[35] A. Krause *et al.*, "Submodular function maximization." *Tractability*, vol. 3, pp. 71–104, 2014.
[36] G. S. Paschos *et al.*, "Learning to cache with no regrets," in *IEEE INFOCOM*, 2019.
[37] A. Ceselli *et al.*, "Mobile edge cloud network design optimization," *IEEE/ACM Trans. on Net.*, vol. 25, no. 3, pp. 1818–1831, 2017.
[38] A. Bochkovskiy *et al.*, "Yolov4: Optimal speed and accuracy of object detection," *arXiv:2004.10934*, 2020.
[39] Y. Cai *et al.*, "Yolobile: Real-time object detection on mobile devices via compression-compilation co-design," *arXiv:2009.05697*, 2020.