The Case for Hierarchical Deep Learning Inference at the Network Edge

Ghina Al-Atat
ghina.alatat@imdea.org
IMDEA Networks Institute
Madrid, Spain

Andrea Fresa
andrea.fresa@imdea.org
IMDEA Networks Institute
Madrid, Spain

Adarsh Prasad Behera
adarsh.behera@imdea.org
IMDEA Networks Institute
Madrid, Spain

Vishnu Narayanan Moothedath
vnmo@kth.se
KTH Royal Institute of Technology
Stockholm, Sweden

James Gross
jamesgr@kth.se
KTH Royal Institute of Technology
Stockholm, Sweden

Jaya Prakash Champati
jaya.champati@imdea.org
IMDEA Networks Institute
Madrid, Spain

ABSTRACT

Resource-constrained Edge Devices (EDs), e.g., IoT sensors and microcontroller units, are expected to make intelligent decisions using Deep Learning (DL) inference at the edge of the network. Toward this end, developing tinyML models is an area of active research – DL models with reduced computation and memory storage requirements – that can be embedded on these devices. However, tinyML models have lower inference accuracy. On a different front, DNN partitioning and inference offloading techniques were studied for distributed DL inference between EDs and Edge Servers (ESs). In this paper, we explore Hierarchical Inference (HI), a novel approach proposed in [19] for performing distributed DL inference at the edge. Under HI, for each data sample, an ED first uses a local algorithm (e.g., a tinyML model) for inference. Depending on the application, if the inference provided by the local algorithm is incorrect or further assistance is required from large DL models on edge or cloud, only then the ED offloads the data sample. At the outset, HI seems infeasible as the ED, in general, cannot know if the local inference is sufficient or not. Nevertheless, we present the feasibility of implementing HI for image classification applications. We demonstrate its benefits using quantitative analysis and show that HI provides a better trade-off between offloading cost, throughput, and inference accuracy compared to alternate approaches.

CCS CONCEPTS

• **Computing methodologies** → *Distributed artificial intelligence.*

KEYWORDS

Edge Computing, Deep Learning, Hierarchical Inference

ACM Reference Format:
Ghina Al-Atat, Andrea Fresa, Adarsh Prasad Behera, Vishnu Narayanan Moothedath, James Gross, and Jaya Prakash Champati. 2023. The Case for Hierarchical Deep Learning Inference at the Network Edge. In *1st International Workshop on Networked AI Systems (NetAISys '23), June 18, 2023, Helsinki, Finland*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3597062.3597278

NetAISys '23, June 18, 2023, Helsinki, Finland

ACM ISBN 979-8-4007-0212-9/23/06.
https://doi.org/10.1145/3597062.3597278.

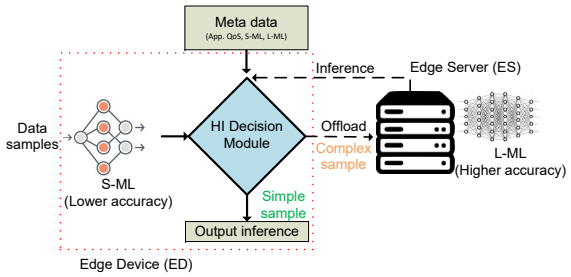


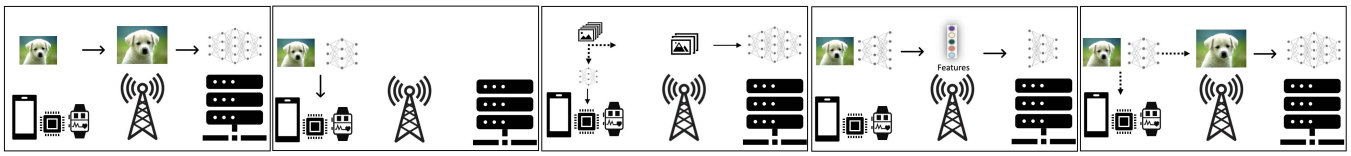
Figure 1: HI framework for DL inference at the network edge.

1 INTRODUCTION

Deep Learning (DL) models are compute and memory intensive and have been traditionally deployed in the cloud. However, recently, an increasing number of applications in Cyber-Physical Systems (CPS), remote healthcare, smart buildings, intelligent transport, etc., use DL inference at the network edge. This initiated a major research thrust in developing small-size ML models (S-ML) – ML models with reduced computation and memory storage requirements – and deploying them on resource-constrained Edge Devices (EDs) such as IoT sensors, wearable devices, mobile phones, drones, and robots. Such deployments are enabled by advancements in hardware and the rapid evolution of model compression techniques [3, 23].

Research works on DL inference at the edge can be broadly classified into 1) tinyML, 2) DNN-partitioning, and 3) inference offloading. TinyML research [23] focuses on enabling extremely resource-limited IoT devices such as micro-controller units (MCUs) to perform on-device DL inference using custom-designed S-ML models. Doing inference at the ED using S-ML saves network bandwidth, and improves responsiveness and energy efficiency of the system [7, 27]. Recent advances in tinyML research enable sophisticated applications to perform tasks on images and audio that go far beyond the prototypical IoT applications such as monitoring environmental conditions (e.g., temperature, CO_2 levels, etc.), machine vibrations for predictive maintenance, or visual tasks such as detecting people or animals [21]. However, tinyML models have relatively poor inference accuracy due to their small size, which also limits their generalization capability and robustness to noise.

The authors in [14] proposed DNN partitioning that divides the inference execution of large-size DNNs between an ED and an Edge



(a) Full Offload **(b) TinyML** **(c) Inference Offloading** **(d) DNN-partitioning** **(e) Hierarchical Inference**

Figure 2: Different approaches for DL inference at the network edge

Server (ES). However, the benefits of this technique were only realized for computationally powerful EDs (such as smartphones) with mobile GPUs [4]. Finally, works on inference/computation offloading propose load-balancing techniques that divide the inference computational load between the ED and the ES [8, 20].

In contrast to the above works, we propose *Hierarchical Inference*, a novel framework for performing distributed DL inference at the edge. Consider the system where the ED is embedded with an S-ML model and enlists the help of ES(s) or cloud on which a state-of-the-art large-size ML model (L-ML) is available. HI differentiates data samples based on the inference provided by S-ML. A data sample is *simple data sample* if S-ML inference is sufficient, else it is a *complex data sample* requiring L-ML inference. The key idea of HI is that only complex data samples should be offloaded to the ES or cloud. Figure 1 shows the HI framework. The HI decision module takes as input the S-ML inference and uses the metadata about S-ML, L-ML, and the application QoS requirements to decide whether a sample is complex or simple and accordingly offload it or not.

Figure 2 illustrates different approaches for DL inference at the edge. In contrast to tinyML research, HI considers inference offloading. Further, HI examines the S-ML inference before making an offloading decision which is in stark contrast to existing inference offloading algorithms. Unlike DNN partitioning, HI is appealing for resource-constrained devices as one may design S-ML models for these devices and use existing L-ML models on ESs or the cloud.

Advantages of HI. On one hand, performing inference on EDs saves network bandwidth, improves responsiveness (reduces latency), and increases energy efficiency. On the other hand, doing inference on ESs results in high accuracy inference. HI reaps the benefits of performing inference on EDs without compromising on the accuracy that can be achieved at ESs. To see this, note that HI offloads only complex data samples which need further inference assistance and will (most likely) receive incorrect inference from S-ML at the ED, while simple data samples which are either redundant or receive correct local inference are not offloaded, thus saving bandwidth, reducing latency and energy for transmission and computation on L-ML. Although we use the term 'S-ML', it need not be an ML algorithm but can be any signal processing or statistical algorithm.

Note that, under HI, performing additional inference locally on all images incurs an extra cost. However, in scenarios where the cost of local computation is lower than that of transmission, this approach would eventually result in significant cost savings, especially since such scenarios are frequent.

Challenges of HI. There are a few challenges to implementing HI.

(1) Differentiating simple and complex data samples is the key to HI. However, in general, it is hard to do this differentiation as the ED does not know a priori if the inference output by S-ML is the ground truth or not.

(2) There are certain requirements for S-ML in order to enable HI. The S-ML should be small enough in size such that, 1) it should be viable for embedding on ED, 2) the computation energy required for S-ML inference should be less than the transmission energy required for transmitting a data sample, and 3) the S-ML should have reasonable accuracy such that for the application of interest the fraction of simple data samples should be higher than that of complex data samples.

Despite the above challenges, in this work, we demonstrate its feasibility by implementing it in two application scenarios: 1) CIFAR-10 image classification, and 2) dog breed image classification. For image classification, we use TFLite and design customized S-MLs for performing inference on images. For each use case, we also provide a quantitative analysis of the benefits of HI. Finally, we also compare the accuracy and delay performance of HI with existing techniques for DL inference: 1) tinyML (no offload), 2) DNN-partitioning, 3) Offloading for Minimizing Delay, and 4) Offloading for Maximizing Accuracy.

Remark: We also implement HI for rolling element fault diagnosis, an example use case of machine fault detection, where our S-ML is a simple threshold rule on the statistical average of the vibration data. It is presented in [1] due to space limitations.

2 RELATED WORKS

Since the advent of AlexNet [16] a decade ago, there has been an explosion in research on bigger DL models with an increasing number of layers and nodes per layer resulting in models with billions of parameters. However, large DL models are often over-parameterized and there has been significant research on model compression techniques that reduce the model size by trading accuracy for efficiency, e.g., see [9, 10]. Efficient model compression techniques have resulted in mobile-size DL models including SqueezeNets [13], MobileNets [24], and EfficientNet [25]. The efforts for designing small-size DL models were further fueled by the need to perform DL inference on edge devices, which span moderately powerful mobile devices to extremely resource-limited MCUs. In the following, we discuss related work on DL inference at the edge.

TinyML. The tinyML research [23] focuses on embedded/on-device ML inference using small-size ML models on extremely resource-constrained EDs such as IoT sensors including MCUs. The motivation for tinyML stems from the following drawbacks of sending data over a network [21]: 1) unreliable network connection, 2) communication latency, and 3) significant transmission energy consumption, among others. A suite of tinyML models is now available for MCUs enabling a wide variety of complex inference tasks such as image classification, visual wake word, and keyword spotting [6]. However, tinyML models have poor inference accuracy (relative to large-size DL models) due to their small size, which limits their

generalization capability and robustness to noise. Further, once a tinyML model is trained (on an edge server/cloud) and deployed on an IoT sensor or an MCU, improving its accuracy using active learning on each data sample may not be possible due to the resource constraints of the device. Therefore, to improve inference accuracy the EDs need to offload the data samples by enlisting the help of an ES or a cloud, where L-ML models are deployed.

DNN Partitioning. For the scenarios where the ED is a powerful mobile device, such as a smartphone, the authors in [14] proposed DNN partitioning, where the front layers of the DNN are deployed on mobile devices while the deep layers are deployed on ESs to optimize the communication time or the energy consumption of the mobile device. The premise of this technique is that the data that needs to be transmitted between some intermediate layers of a DNN is much smaller than the initial layers. Following this idea, significant research work has been done that includes DNN partitioning for more general DNN structures under different network settings [11, 17] and using heterogeneous EDs [12], among others. However, as we will show later, for DNN partitioning to be beneficial, the processing times of the DNN layers on the mobile device should be small relative to the communication time of the data generated between layers. Thus, works on DNN partitioning (e.g., see [4]) suggest mobile GPUs making this technique infeasible for extremely resource-constrained IoT sensors and MCUs.

Inference Offloading. Inference/computation offloading is a load-balancing technique for partitioning the set of data samples between the ED and the ES for computing the inference. It is worth noting that computation offloading between EDs and ESs was extensively studied in the literature, e.g., see [18]. However, the majority of the works studied offloading generic computation jobs, and the aspect of accuracy, which is relevant to offloading data samples, has only been studied recently, e.g., see [8, 20]. These works compute offloading decisions using one or more of the following quantities: job execution times, communication times, energy consumption, and average test accuracy of the ML models. Note, however, that the average test accuracy may not be the right indicator for the correct or incorrect inferences provided by an S-ML across different data samples and thus it may result in adversarial cases where most of the data samples that are scheduled on S-ML may receive incorrect inference. In contrast to inference offloading HI first examines the S-ML output to determine if the data sample is simple or complex and only offloads if it is a complex data sample. We note that early exiting within the layers of DNNs, proposed in [26], received considerable attention recently. This technique could be used in conjunction with the above DL inference approaches, including HI, to further reduce the latency in reference.

In our previous work [19], we proposed the idea of HI and used an online learning framework to compute offloading decisions for MobileNet as the S-ML model. In contrast, in this paper, we provide a general definition for HI, consider multiple use case scenarios, design S-ML algorithms for MCUs, and provide a quantitative comparison with existing approaches on DL inference at the edge.

3 HI FOR CIFAR-10 IMAGE CLASSIFICATION

We choose Image classification as our use case as it is a fundamental task inherent to a variety of applications including object detection, image analysis, video analytics, and remote sensing. Significant efforts have been devoted to develop advanced classification approaches and techniques aiming to improve classification accuracy.

We choose *CIFAR*-10 image dataset consisting of 50000 training samples, 10000 test samples and 10 classes. We design and train an S-ML that can be embedded on a resource-constrained ED such as MCU with eflash memory size of order 1 MB and SRAM of order few hundred KB. The details of the ML models are given below.

S-ML: A five-layer CNN is constructed for image classification using TensorFlow's Keras API with a convolutional layer, a max-pooling layer, a flatten layer, and two fully-connected dense layers. The model is trained on CIFAR-10 dataset and quantized to TFLite mode, achieving an accuracy of 62.58% and a size of 0.45 MB, suitable for MCUs.

L-ML: Existing DNN models have reached up to 99.5% accuracy on CIFAR-10 [5]. For our work, we choose EfficeintNet [22], which achieves an accuracy of 95%. Exploring the idea of HI, the data sample will first undergo inference by the S-ML. The ED will decide to either accept the inference or consider it incorrect and offload the data sample to the L-ML on the ES. *Simple data samples* in this context are the images that the S-ML is able to classify correctly, otherwise, they are considered as *complex data samples*.

In order to facilitate the ED to make the offloading decision, we use two key ideas proposed in [19]. The first idea is related to how to quantify the confidence of the S-ML's inference. For image classification, a DNN outputs a probability mass function (pmf) over the classes. One can obtain such pmf for other ML classification algorithms (such as linear classifier) by normalizing the output with the sum of the values output for each class. We use the maximum probability value, denoted by p, from the pmf as the confidence of S-ML. This is justified because it is a standard approach to declare the class corresponding to the maximum probability p as the true class. Based on the value of p, the ED will accept the inference or offload it to the ES. In particular, the ED's offloading decision for data sample i will be based on comparing p_i with a threshold $\theta \in [0, 1)$. It will be to *offload* if $p_i < \theta$ and *do not offload*, otherwise.

The second idea adopted from [19] is the cost model. If the S-ML's inference is rejected and the data sample i is offloaded to the ES, a fixed cost $0 \leq \beta < 1$ will be received for offloading. In practice, β may correspond to transmission energy consumption, data transfer cost , or remote inference delay. We choose to introduce HI using abstract cost β for offloading to enhance flexibility in the decision-making process and make it adaptable to diverse scenarios and requirements. Note that β may vary across different setups, allowing our algorithm to make more informed decisions that balance various costs and benefits, such as network conditions, resource availability, battery life, and user's personal preferences. Furthermore, denote by η_i the cost incurred at the ES for the incorrect L-ML inference for data sample i. It is equal to 1 if L-ML's inference is incorrect and is equal to 0, otherwise. If data sample i's local inference is accepted, then the incurred cost, denoted by γ_i, is 0 if the inference is correct, otherwise, it is 1. Thus, the cost for data sample i using HI is given by β and η_i in case p_i is less than θ. Otherwise, it is γ_i. Figure 3 shows the number of correctly and incorrectly classified images of the CIFAR-10 dataset using our S-ML model as a function of p. The figure clearly illustrates that for p greater than 0.6, the number of correctly classified images becomes higher than the

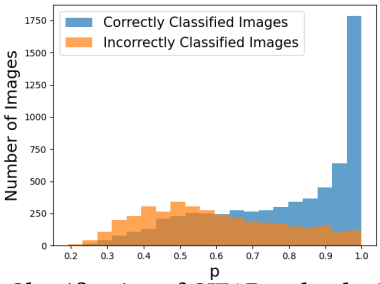


Figure 3: Classification of CIFAR-10 by the S-ML model.

incorrectly classified ones. Thus, it is an apt threshold candidate to decide whether to accept the local inference or not. Brute-force search shows that the optimal probability threshold θ^* that yields the minimum cost for CIFAR-10 dataset using $\beta = 0.5$ is 0.607.

HI Approach: All the images will be subjected to inference using the S-ML embedded on the ED. The ED offloads only the images for which p is less than the optimal threshold $\theta^* = 0.607$.

Results and Discussion. Under the full-offload approach, which involves deploying EfficientNet on the ES and offloading all the 10000 images, the ES achieves an accuracy of 95%. Only 500 out of 10000 images were falsely classified. However, it comes at a cost of $10000\,\beta + 500$. Conversely, the complete local approach, which involves accepting all of the local inferences produced by the S-ML can reduce the cost to 3742, which is a significant reduction particularly when β is close to one. However, it entails a significant trade-off in terms of inference accuracy 62.58%, which is insufficient to ensure reliability in the majority of use cases.

Using the HI approach with $\theta^* = 0.607$ ($\beta = 0.5$), only 35.5% of the images are offloaded (3550 images) of which 71 images were misclassified by the L-ML. This yields a cost of $3550\beta+71$. Moreover, out of the 6450 accepted local inferences, 1577 were misclassified (which contribute an extra 1577 to the cost). This means that the total cost of HI is $3550\beta + 1648$. Since 1648 images are misclassified under HI, it has 83.52% accuracy. Table 1 summarizes the comparison between HI and the previously mentioned approaches(Full-offload and no offload). Note that HI results in a cost reduction of $\left(\frac{6450\,\beta-1148}{10000\,\beta+500} \times 100\right)\%$ compared to full offload. For different values of β, the cost reduction is in the range $14 - 49\%$. Thus, HI can provide the best of both worlds, namely significant cost reduction while still achieving a dependable level of accuracy, which gracefully degrades with offloading cost β.

Approach	No Offload	Full Offload	Hierarchical Inference
Offloaded Images (%)	0(0%)	10000(100%)	3550(35.5%)
Misclassified Images (%)	3,742(37.42%)	500(5%)	1,577 **ON ED** + 71 **ON ES** (16.48%)
Accuracy (%)	62.58%	95%	83.52%
Cost (β)	3742	$10000\,\beta + 500$	$3550\,\beta + 1648$

Table 1: Image classification performance comparison for CIFAR-10 dataset.

4 HI FOR DOG BREED IMAGE CLASSIFICATION

In this section, we introduce another use case that comes in alignment with the concept of hierarchical inference naturally. Suppose that we have the same set-up (ED connected to an ES) and that we are only interested in classifying the dog breeds of the dogs present in CIFAR-10. On one hand, achieving high accuracy for this complicated task using tinyML models is still challenging due to the complex nature of dog breed classification and the limited resources of tinyML devices. On the other hand, fully offloading all of the images can result in high accuracy but is, however, very costly. Under HI, we propose to use an S-ML to classify dog images and non-dog images and only offload the dog images to ES for an L-ML to classify the dog breed. Note that, in this use case the dog images are complex as they require further assistance for classification at the ES. All non-dog images are simple.

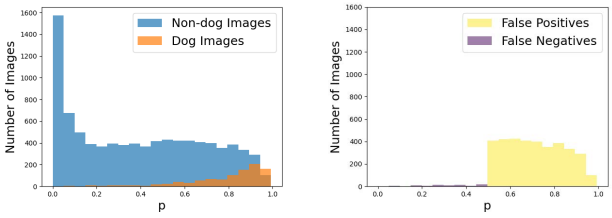
The approach used in this use case generalizes to any context where the data samples of interest would be sufficiently complex that almost none of them could be accurately inferred on the edge device. Then, the S-ML will not do any final inference for the data of interest. Instead, its job would be just to eliminate a large part of the irrelevant data and only delegate the data samples of interest.

S-ML: The ED requires a small binary ML model to recognize if an image features a dog or not. We construct and train a compact CNN with five layers for binary classification, with a ReLU-activated dense layer of 32 neurons and a sigmoid-activated final layer. The SML produces probability score p ranging from 0 to 1, indicating the likelihood of a positive class, i.e., dogs. The model is converted to a quantized TFLite format for resource-limited devices, resulting in a 0.23 MB model with 63.86% accuracy.

L-ML: We assume a perfect L-ML with 100% accuracy. The S-ML will classify dog images, which are then offloaded to the ES for breed determination by L-ML. The lack of true labels for CIFAR-10 necessitates this assumption for comparison with the ideal ES accuracy. In fact, it is not unrealistic, as there are multiple models achieving exceptional accuracy in dog breed classification for various datasets [2].

For data sample i, the S-ML outputs p_i, the probability that it features a dog. The ED will utilize this probability to decide whether to offload the sample or consider it irrelevant and disregard it. The ED's objective is to offload only the dog images, and since the S-ML is a binary classification model, $p = 0.5$ serves as a threshold for dog classification and thus the decision rule for data sample i, will be to *Offload* if $p_i \geq 0.5$ and *Do not offload* otherwise.
If the data sample i has to be offloaded to the ES, the cost received will depend on its L-ML's inference. A fixed cost $0 \leq \beta < 1$ will be received for offloading a sample from the dataset of interest (if it was true dog image). Alternatively, if the data sample offloaded was irrelevant (does not feature a dog), a cost of 1 will be incurred.

Figure 4a represents the actual non-dog (simple data samples) and dog images (complex data samples) in our dataset distributed as a function of p. The images having $p \geq 0.5$ will be classified as dog images (complex) by the S-ML. The figure clearly shows that most of the dog images were correctly identified by the S-ML, i.e., the precision of the trained S-ML is high. Specifically, only 88 dog images out of 1000 dogs are misclassified and are not offloaded. These are the false negatives. This means that for the task of classifying the dog breeds of CIFAR-10, 912 out of 1000 dogs will be identified by the S-ML and delegated to the L-ML to determine their correct breed. This yields a 91.2% accuracy. Also, the figure illustrates that the S-ML will incorrectly classify a substantial number of non-dog images, 3521 to be precise, as dog images, leading to their unnecessary offloading to the L-ML. These misclassifications are false positives. These false positives will only affect the total



(a) Classification of CIFAR-10: Dog and Non-dog images. **(b) False Negatives and False Positives of CIFAR-10 by the S-ML.**

Figure 4: Proposed S-ML output for dog breed classification.

cost incurred but not the accuracy of HI. Figure 4b presents the instances of false positives and false negatives as a function of p.

Full offloading can achieve maximal accuracy but at a cost of 9000β for offloading irrelevant images. On the other hand, implementing HI can tremendously reduce the accrued cost while maintaining a reliable level of accuracy. Table 2 compares HI with the full-offload approach. Using HI leads to a reduction of offloaded images by 55.67% which reduces the cost by $(\frac{88\beta+5479}{10000\beta+9000} \times 100)\%$ while maintaining an accuracy of 91.2%. The relative cost reduction for different values of β, is in the range $50-60\%$.

From the above use case analysis, we conclude that HI offers a balanced solution that strikes a middle ground between the two extremes of no offload and full offload. On one end, fully offloading the samples yields maximum accuracy but comes with high costs. On the other end, performing all inferences locally reduces costs but requires significant compromises in accuracy. HI approach has the potential to offer the best of both worlds, allowing for a significant reduction in costs while still maintaining a reliable level of accuracy.

Approach	Full offload	HI
Number of Offloaded Images	10,000	4,433
Accuracy (%)	100%	91.2%
Cost	$1000\,\beta + 9000$	$912\,\beta + 3521$
Cost Reduction (%)	0%	$(\frac{88\beta+5479}{1000\beta+9000} \times 100)\%$

Table 2: HI binary classification for CIFAR-10 dataset performance comparison with Full offload.

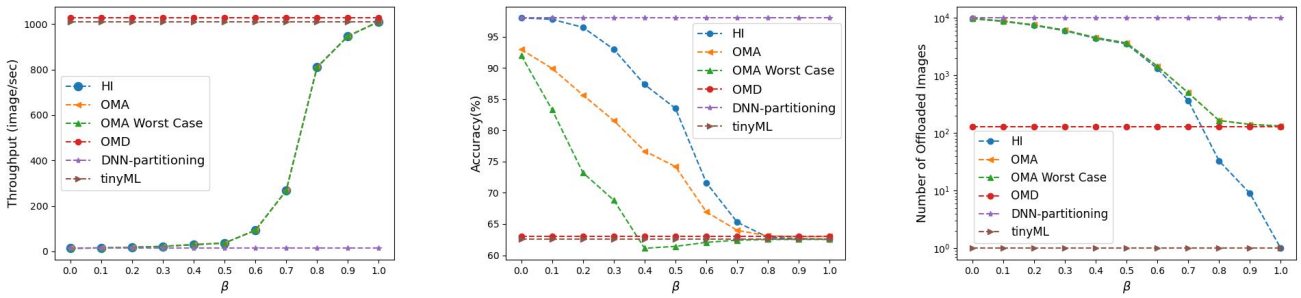
5 COMPARISON WITH EXISTING APPROACHES

In this section, we compare HI with tinyML, two algorithms related to inference offloading, and DNN partitioning. We use the CIFAR-10 dataset and use throughput, accuracy, and number of offloaded images as metrics. Below, we present the algorithms, describe the experimental setup, and discuss the results.

(1) *TinyML*: Embedded ML approach where all the inferences of the S-ML are accepted (no offload).
(2) *Offloading for Maximizing Accuracy (OMA)*: Computation offloading approach that partitions inference jobs between the ED and ES for maximizing accuracy given a time constraint. The time constraint we subject is equal to the makespan of the HI approach, where makespan is defined as the total time required to process all the jobs. Two cases are investigated: 1) Partitioning the images between ED and ES randomly, and 2) *OMA Worst Case*, where the offloaded inference jobs belong to the set of simple samples.
(3) *Offloading for Minimizing Delay (OMD)*:A work-conserving schedule that schedules the images one after another on S-ML and L-ML whenever either of them becomes idle. This schedule minimizes the makespan.
(4) *DNN-partitioning [15]*: Since the CIFAR-10 dataset has 32×32 images, DNN-partitioning approach results in full offload in our context. Further explanations and measurements are provided in the Appendix of [1].

We use the S-ML and L-ML described in Section 3 for all the above approaches. Our S-ML can fit on any IoT device but for the current work, we deploy it on a Raspberry Pi 4*B*, which features a single CPU with 4 cores, a frequency of 1.5 GHz, and 5 GB of RAM. As for the L-ML, it is hosted on an ES featuring 2 CPUs with 16 cores each, a frequency of 2.4 GHz, an NVIDIA Tesla T4 GPU, and 256 GB of RAM. Both devices are connected to the same WLAN and are linked via 802.11 operating on a 5 GHz channel approved by ETSI for Multi-access Edge Computing (MEC) communication. The transmission of images from ED to ES is accomplished through TCP sockets. In order to estimate the communication time, we utilize iPerf to determine the communication bandwidth. We conduct 30 experiments, each lasting 60 seconds, to derive our estimates. The average communication bandwidth between Raspberry Pi and ES recorded was 10.45 MB/s (SD = 0.6 MB/s). The average inference time per one sample by the S-ML on the Raspberry Pi is 0.99 *msec*, which is significantly lower than the 74.34 *msec* required to offload the image from the ED to the ES, and to execute the inference on the L-ML on ES with GPU processing.

We compare HI with the various approaches in terms of throughput (relating to the makespan and thus delay), accuracy and the number of offloaded images (which relates to cost). Figure 5a (5b)



(a) Throughput of CIFAR-10 image classification for different approaches. **(b) Accuracy of CIFAR-10 image classification for different approaches.** **(c) Number of images offloaded for CIFAR-10 image classification for different approaches.**

Figure 5: Different approaches comparison for varying β.

shows the throughput (accuracy) of the four approaches vs HI for different values of β. Note that β has an effect only on the results of HI, but since the chosen time constraint of OMA (and OMA worst case) is equal to the makespan of HI, we see that OMA is also sensitive to the choice of β. As mentioned in Section 3, β models the transmission energy consumption or data transfer cost. So as β increases, accepting the local inference instead of offloading will be more favorable which in practice leads to the decrease in accuracy and increase in throughput we notice in Figures 5a and 5b. Figure 5a shows that tinyML has maximum throughput and OMD has throughput close to it. In addition, tinyML scores the lowest number of offloaded images (lowest cost) but Figure 5b clearly states that their accuracy is the lowest and not reliable for most of the DL inference tasks. Also, note that DNN-partitioning has maximum accuracy but it is the worst in terms of throughput and the number of offloaded images. HI and OMA have, by design, similar throughput but HI provides significant gains in terms of accuracy and lower number of offloaded images (thus cost) compared to OMA.

Compared to other approaches, HI offers a significant reduction in latency and cost while maintaining high accuracy. In fact, HI approach was found for example to reduce latency and the number of offload images by approximately 63.15% and 64.45% respectively compared to the full offload approach, while still maintaining 83.52% accuracy for $\beta = 0.5$. In summary, HI has demonstrated its ability to provide the best trade-off between accuracy, delay, and cost. As a result, it represents a promising solution for real-time applications on resource-constrained devices.

6 CONCLUSION

In conclusion, this paper explored a novel approach, Hierarchical Inference (HI), for performing distributed DL inference between edge devices and edge servers. HI addresses the challenges of resource-constrained EDs in making intelligent decisions using DL inference. Through application use cases and quantitative analysis, we demonstrated the feasibility and benefits of HI. We presented three different use cases for HI: machine fault detection, CIFAR-10 dataset image classification, and dog breed classification. We provide a quantitative comparison between HI and existing techniques for DL inference at the edge which include, tinyML, inference offloading, and DNN partitioning. Our results demonstrate that HI achieves significant cost savings while maintaining reliable accuracy. For instance, following HI's approach for CIFAR-10 image classification reduces latency and the number of offloaded images compared to the full-offload approach, on average for values of β ranging from 0 to 1, by 60.84% and 62.18% while maintaining an accuracy greater than 80%. With minimal compromise on accuracy, HI achieves low latency, bandwidth savings, and energy savings, making it a promising solution for edge AI systems.

REFERENCES

[1] Ghina Al-Atat, Andrea Fresa, Adarsh P. Behera, Vishnu N. Moothedath, James Gross, and Jaya P. Champati. 2023. The Case for Hierarchical Deep Learning Inference at the Network Edge. arXiv:2304.11763 [cs.DC]

[2] Ying Cui, Bixia Tang, Gangao Wu, Lun Li, Xin Zhang, Zhenglin Du, and Wenming Zhao. 2023. Classification of dog breeds using convolutional neural network models and support vector machine. *bioRxiv* (2023). https://doi.org/10.1101/2023.02.15.528581

[3] Lei Deng, Guoqi Li, Song Han, Luping Shi, and Yuan Xie. 2020. Model Compression and Hardware Acceleration for Neural Networks: A Comprehensive Survey. *Proc. IEEE* 108, 4 (2020), 485–532. https://doi.org/10.1109/JPROC.2020.2976475

[4] Chongwu Dong, Sheng Hu, Xi Chen, and Wushao Wen. 2021. Joint optimization with DNN partitioning and resource allocation in mobile edge computing. *IEEE Transactions on Network and Service Management* 18, 4 (2021), 3973–3986.

[5] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *Proc. ICLR*.

[6] Colby Banbury et al. 2021. MLPerf Tiny Benchmark. In *Proc. Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*.

[7] Igor Fedorov, Ryan P. Adams, Matthew Mattina, and Paul N. Whatmough. 2019. SpArSe: Sparse architecture search for CNNs on resource-constrained microcontrollers. *Advances in Neural Information Processing Systems* 32 (2019).

[8] Andrea Fresa and Jaya P. Champati. 2022. An Offloading Algorithm for Maximizing Inference Accuracy on Edge Device in an Edge Intelligence System. In *Proc. ACM MSWiM*. 15–23.

[9] Song Han, Huizi Mao, and William J. Dally. 2016. Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding. In *Proc. ICLR*.

[10] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. 2018. AMC: AutoML for Model Compression and Acceleration on Mobile Devices. In *Proc. ECCV*. 815–832.

[11] Chuang Hu, Wei Bao, Dan Wang, and Fengming Liu. 2019. Dynamic Adaptive DNN Surgery for Inference Acceleration on the Edge. In *Proc. IEEE INFOCOM*. 1423–1431. https://doi.org/10.1109/INFOCOM.2019.8737614

[12] Chenghao Hu and Baochun Li. 2022. Distributed Inference with Deep Learning Models across Heterogeneous Edge Devices. In *Proc. IEEE INFOCOM*. 330–339. https://doi.org/10.1109/INFOCOM48880.2022.9796896

[13] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and < 0.5MB model size.

[14] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. 2017. Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge. In *Proc. ACM ASPLOS*. 615–629. https://doi.org/10.1145/3037697.3037698

[15] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. 2017. Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge. *SIGARCH Comput. Archit. News* 45, 1 (apr 2017), 615–629. https://doi.org/10.1145/3093337.3037698

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Proc. NIPS*. 1097–1105.

[17] En Li, Liekang Zeng, Zhi Zhou, and Xu Chen. 2020. Edge AI: On-Demand Accelerating Deep Neural Network Inference via Edge Computing. *IEEE Transactions on Wireless Communications* 19, 1 (2020), 447–457. https://doi.org/10.1109/TWC.2019.2946140

[18] Pavel Mach and Zdenek Becvar. 2017. Mobile Edge Computing: A Survey on Architecture and Computation Offloading. *IEEE Communications Surveys Tutorials* 19, 3 (2017), 1628–1656.

[19] Vishnu N. Moothedath, Jaya P. Champati, and James Gross. 2023. Online Algorithms for Hierarchical Inference in Deep Learning applications at the Edge. arXiv:2304.00891

[20] Ivana Nikoloska and Nikola Zlatanov. 2021. Data Selection Scheme for Energy Efficient Supervised Learning at IoT Nodes. *IEEE Communications Letters* 25, 3 (2021), 859–863. https://doi.org/10.1109/LCOMM.2020.3034992

[21] Emil Njor, Jan Madsen, and Xenofon Fafoutis. 2022. A Primer for tinyML Predictive Maintenance: Input and Model Optimisation. In *Proc. Artificial Intelligence Applications and Innovations*. 67–78.

[22] Julius Ruseckas. n.d.. EfficientNet on CIFAR10. https://juliusruseckas.github.io/ml/efficientnet-cifar10.html.

[23] Ramon Sanchez-Iborra and Antonio F. Skarmeta. 2020. TinyML-Enabled Frugal Smart Objects: Challenges and Opportunities. *IEEE Circuits and Systems Magazine* 20, 3 (2020), 4–18.

[24] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *Proc. IEEE CVPR*. 4510–4520.

[25] Mingxing Tan and Quoc V. Le. 2019. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In *Proc. ICML*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.), Vol. 97. PMLR, 6105–6114.

[26] Surat Teerapittayanon, Bradley McDanel, and H.T. Kung. 2016. BranchyNet: Fast inference via early exiting from deep neural networks. In *Proc. ICPR*. 2464–2469.

[27] Yundong Zhang, Naveen Suda, Liangzhen Lai, and Vikas Chandra. 2017. Hello Edge: Keyword Spotting on Microcontrollers. *CoRR* abs/1711.07128 (2017).