MultiTASC: A Multi-Tenancy-Aware Scheduler for Cascaded DNN Inference at the Consumer Edge

Sokratis Nikolaidis†, Stylianos I. Venieris‡, and Iakovos S. Venieris†
†National Technical University of Athens, Athens, Greece, ‡Samsung AI Center, Cambridge, UK
Email: sokratisnikolaidis@mail.ntua.gr, s.venieris@samsung.com, venieris@cs.ece.ntua.gr

***Abstract*—Cascade systems comprise a two-model sequence, with a lightweight model processing all samples and a heavier, higher-accuracy model conditionally refining harder samples to improve accuracy. By placing the light model on the device side and the heavy model on a server, model cascades constitute a widely used distributed inference approach. With the rapid expansion of intelligent indoor environments, such as smart homes, the new setting of Multi-Device Cascade is emerging where multiple and diverse devices are to simultaneously use a shared heavy model on the same server, typically located within or close to the consumer environment. This work presents MultiTASC, a multi-tenancy-aware scheduler that adaptively controls the forwarding decision functions of the devices in order to maximize the system throughput, while sustaining high accuracy and low latency. By explicitly considering device heterogeneity, our scheduler improves the latency service-level objective (SLO) satisfaction rate by 20-25 percentage points (pp) over state-of-the-art cascade methods in highly heterogeneous setups, while serving over 40 devices, showcasing its scalability.**

Reference: S. Nikolaidis, S. I. Venieris and I. S. Venieris, "MultiTASC: A Multi-Tenancy-Aware Scheduler for Cascaded DNN Inference at the Consumer Edge," 2023 IEEE Symposium on Computers and Communications (ISCC), 2023
Link: https://ieeexplore.ieee.org/document/10217872
Keywords: Vision and Learning, Other

I. Introduction

In recent years, there has been significant progress in the field of on-device execution of deep learning (DL) inference tasks [1]. At the same time, with the rapid expansion of indoor intelligent environments [2], such as smart homes and offices, DL is poised to enable new use-cases by expanding to a greater variety of smart devices, such as IoT cameras and AI speakers. Nevertheless, due to their form-factor and energy-efficiency constraints, most of these devices lie on the low end of the computational spectrum. In contrast to modern high-end smartphones, which host powerful processors and accelerators (GPUs, NPUs) [3], low-end devices are not able to deploy state-of-the-art deep neural networks (DNNs), resorting to lightweight, but lower-accuracy models.

Given that offloading data to the cloud for inference can incur significant costs in terms of bandwidth, latency and privacy, an alternative scheme is emerging that places the server inside or closer to the consumer environment in the form of a dedicated AI hub that assists the surrounding devices [2]. In this context, a prominent deployment approach are the cascade architectures [4]–[9]. Cascade architectures make use of the fact that not all samples are of the same difficulty and choose to process only the more challenging cases with a powerful model deployed on the server, while letting easier samples, which usually form the majority of the data, to be processed on-device with a light model. A lot of research has been conducted on such architectures, mainly focusing on the forwarding decision criterion and selection of model pairs, progressing the potency of the scheme.

Despite the progress, the majority of these works have solely focused on the setting where the server is used by a *single* device at any given time. Such an assumption no longer holds in upcoming intelligent environments where multiple devices execute DL inference tasks simultaneously under the support of the same AI hub [10]. This gives rise to the new setting of *Multi-Device Cascade*, where multiple devices use the same model on a shared edge-based server. Such a system needs to be scalable in terms of number of devices, balancing fast response time and high accuracy across them. In this context, status-quo approaches, which treat each model cascade independently, would either lead to brute-forcing inference requests through the server's resources, resulting in the system being overwhelmed, or force all devices to fallback to on-device execution, negating any accuracy benefits. Therefore, there is an emerging need for novel methods that explicitly target the challenges of a Multi-Device Cascade.

In this work, we propose MultiTASC, a multi-tenancy-aware scheduler that allows the Multi-Device Cascade architecture to adapt to dynamic conditions by reconfiguring on-the-fly the forwarding criterion of the cascades, thus controlling the server's inference request rate at run time. To sustain smooth operation under device heterogeneity, we further introduce a heterogeneity-aware prioritization scheme that selectively adapts each device's operation depending on its capabilities. Overall, MultiTASC sustains high responsiveness, throughput and accuracy while the number of assisted devices scales in both homogeneous and heterogeneous device ecosystems. The key contributions of this paper are the following:

- A system model of the Multi-Device Cascade architecture. By expanding the cascade architecture to accommodate multiple devices, our parametrization exposes the tunable parameters and enables system designers to systematically investigate its trade-offs.
- A multi-tenancy-aware scheduler optimized for the Multi-Device Cascade architecture. The proposed scheduler aims to maximize throughput and accuracy while satisfying a latency constraint. This is accomplished through the adaptive manipulation of the forwarding decision functions of the devices that control the inference request flow to the server. By introducing a new metric, Capacity, our scheduler estimates the maximum amount of samples

that can be processed on the server within a given latency constraint and utilizes it to dynamically reconfigure the forwarding decision functions on the assisted devices.

II. Background & Related Work

On-Device DNN Inference. In recent years, the on-device deployment of DL models has rapidly gained ground [1]. Although on-device training remains a distant possibility, inference has been achieved on computationally constrained devices through the utilization of various methods, such as lightweight model design [11], quantization [12], pruning [13], knowledge distillation [14], and optimized scheduling [15]. Still, despite the maturity of these techniques, upcoming intelligent spaces, such as smart homes and offices, are often populated with small-form factor, resource-constrained devices (*e.g.* smart cameras, AI speakers), which lack the processing power to support high-accuracy, computationally intensive models. This fact has motivated the development of distributed collaborative inference approaches.

Distributed Collaborative Inference. Distributed inference systems employ a server to assist mobile and embedded devices in performing DNN inference tasks. This approach has given rise to two main schemes: *offloading* and *cascading*. Offloading [16]–[18] leverages the server's resources to alleviate part of the computational load on the device by splitting the DNN model into two parts; the first part is executed on the device, with the intermediate results forwarded to the server to proceed with the execution of the second part.

In the cascade scheme, samples are fed into a two-model sequence of DNNs, with progressively increasing complexity and accuracy. Once the input is processed by a model, its output is evaluated by a forwarding decision function, which determines whether the inference ends using the current result or proceeds to the next model. Several works have been conducted on this area. [4] proposes to analyze the difference between the best and the second best softmax result in order to determine which inputs can be processed only by the light model and which require the heavy model. [5] introduces a trainable forwarding criterion by training a neural head on the light model's feature extractor. [6] explores the idea of using more than two DNN models and compares different decision metrics. Finally, [7] and [8] propose solutions for deploying cascades under tight energy constraints.

Multi-Device Cascades. Previous works on cascades [4]–[8] have focused on an isolated setting, where a single device has exclusive access to a server. Nonetheless, with an increased rate of AI-enabled applications deployed in indoor environments, this assumption is unrealistic; instead, multiple devices will require support from the same server at any given time. Such a setting requires principled investigation, since a brute-force deployment, where devices run independently of each other, would lead to either an overloaded server, and in turn to long response times, or significant drop in accuracy by falling back to local execution. The Multi-Device Cascade setting remains unexplored and is the focus of this work.

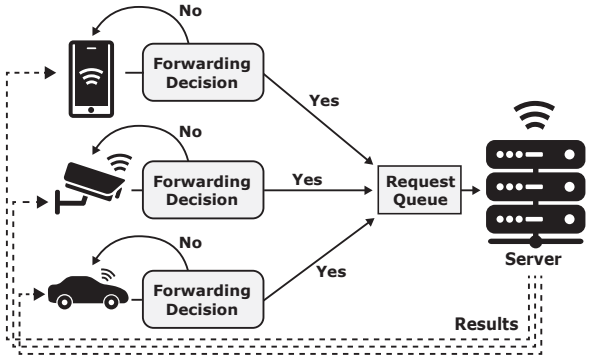

Fig. 1: System architecture of Multi-Device Cascade.

III. Multi-Device Cascade of Classifiers

Fig. 1 presents the system architecture of a Multi-Device Cascade, where IoT devices are running DL inference tasks. All devices perform the same task, *e.g.* object detection, but may host different models. The output predictions of each sample are passed on to a *Forwarding Decision* function that determines whether the DL model of the device is confident about its output. Depending on the decision, either the result remains as is, or the sample is forwarded to a server to be processed by a more accurate model. The forwarded samples from all devices are put into a *Request Queue* from where they are drawn to form the input of the server-hosted model; as such, the server-side model is *shared* among all devices. Finally, the results produced on the server are sent back to their corresponding devices.

Single-Device Cascade. Let $x \in \mathcal{X}$ be the input of a DL inference task performed on an IoT device and $y \in \{1, ..., K\}$ the classification label produced by the model, where K is the number of classes. By using a decision function, $d(\cdot)$, on the output of the light device model, we can decide whether the result is satisfactory ($d(\cdot)=0$) or we should forward the sample to the heavier model for further processing ($d(\cdot)=1$). Denoting the classification function of the light model by $f_l : \mathcal{X} \rightarrow [0,1]^K$ that yields the softmax ouput vector of the model whose maximum value is the predicted class, and the classification of the heavy model by $f_h : \mathcal{X} \rightarrow [0,1]^K$, we formally define a collaborative cascade system as:

$$casc_{f_l,f_h,d}(x) = \begin{cases} f_l(x) & \text{if} \quad d(f_l(x)) = 0 \\ f_h(x) & \text{if} \quad d(f_l(x)) = 1 \end{cases} \tag{1}$$

Multi-Device Cascade. To capture Multi-Device Cascade architectures (Fig. 1), we extend the single-device cascade system modeling as follows. Let $\mathcal{D}$ be the set of devices that use the server as a collaborator. Then, the Multi-Device Cascade system is defined as:

$$casc_{f_l^i,f_h,d^i}(x^i) = \begin{cases} f_l^i(x^i) & \text{if} \quad d^i(f_l^i(x)) = 0 \;\; \forall i \in \{1, ..., |\mathcal{D}|\} \\ f_h(x^i) & \text{if} \quad d^i(f_l^i(x)) = 1 \end{cases}$$

where $x^i \in \mathcal{X}^i$ is a sample processed by the i-th device, f_l^i the classification function of the DL model deployed on the i-th device, f_h the shared heavy model on the server, and $d^i(f_l^i(x))$ the forwarding decision function of the i-th device.

Congestion Problem. When a single device is using the server as the collaborator, it has exclusive access to the server's computational resources and hence the response time is minimized. With the proliferation of IoT devices, such a server should be utilized by many devices at the same time, to amortize its cost and provide maximum utility. Nonetheless, depending on the conditions, if the arrival rate of incoming requests exceeds the attainable processing throughput of the server, the server will be overloaded and the requests will experience severe waiting times in the request queue.

Given the number of devices $|\mathcal{D}|$, we express the arrival rate of requests to the server as $AR_{\text{server}} = \sum_{i=1}^{|\mathcal{D}|} {}^{p^i_{\text{casc}}}/_{t^i_{\text{inf}}}$ where t^i_{inf} is the average inference latency of a sample on the i-th device and p^i_{casc} is the probability of a sample giving $d^i(f^i_l(x^i)) = 1$. Given the attainable throughput T_{server} of the server, we distinguish between three different states:

- $AR_{\text{server}} < T_{\text{server}}$: the server's processing rate is larger than the arrival rate, resulting in the server being underutilized. A larger number of difficult samples could be sent to the server to achieve higher accuracy.
- $AR_{\text{server}} = T_{\text{server}}$: equilibrium is attained. Requests are processed upon arrival without accumulating and the server's processing power is fully utilized.
- $AR_{\text{server}} > T_{\text{server}}$: the requests arrive faster than the server can process. If this state lasts, the request queue will be large, resulting in unwanted latency.

Since the probability p^i_{casc} of the forwarding decision function is not static, but changes during inference, such an architecture could benefit by dynamically adapting its state depending on the current conditions. Since t^i_{inf} and T_{server} are fixed based on the device and server-side processors, we opt to manipulate p^i_{casc} by changing the parameters of $d^i(f^i_l(x^i))$ in order to introduce adaptability to the system.

Problem Optimization. We formulate the aforementioned problem as a multi-objective optimization problem, aiming to maximize accuracy and throughput subject to a latency service-level objective (SLO). The next section describes our proposed scheduler that tackles this problem.

IV. Proposed solution

To combat the accumulation of requests or underutilization of server resources, we propose MultiTASC, a multi-tenancy-aware scheduler that dynamically adapts the forwarding decision functions on assisted devices in order to control the arrival rate of samples. Fig. 2 depicts its internal design. The scheduler monitors the state of the request queue, communicates with the assisted devices and tunes the flow of incoming requests based on the current conditions. To this end, we introduce four techniques: *i)* reconfigurable forwarding decision functions, *ii)* the Capacity metric on the server, *iii)* fractional update, and *iv)* device heterogeneity-aware prioritization for effectively communicating updates to the devices.

A. Reconfigurable Forwarding Decision Function

Research effort has been invested into quantifying the prediction confidence of DNNs, leading to several approaches [6], [19]. In this work, we adopt the Best-versus-Second-Best (BvSB) metric [20], using the difference between the two highest values of the softmax outputs of the model as $\text{BvSB}\Big|_{f(x)} = P_1 - P_2$, where P_1 and P_2 are the maximum and second maximum values, respectively, in the output softmax vector of classifier $f(x)$. Other metrics, such as top-1 softmax or entropy, can be used with minimal modifications, potentially leading to different latency-accuracy trade-offs.

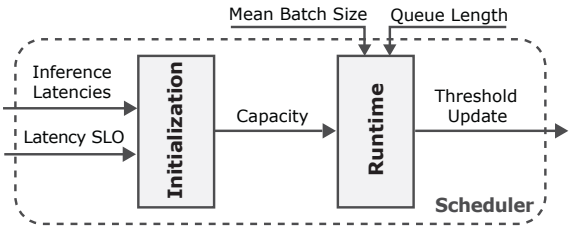

Fig. 2: MultiTASC's internal architecture.

The vast majority of existing cascade systems opts to select a specific threshold at design time, which then remains fixed upon deployment. In this work, to accommodate the adaptability needs of our target system, we adopt an alternative scheme where the decision function can be *dynamically reconfigured*. The decision function $d^i(\cdot)$ is thus defined as:

$$d^i(f^i_l(x)) = \begin{cases} 0 & \text{if} \quad \text{BvSB}\Big|_{f^i_l(x)} \geq c^{i,t} \\ 1 & \text{if} \quad \text{BvSB}\Big|_{f^i_l(x)} < c^{i,t} \end{cases} \tag{2}$$

where $c^{i,t}$ is the decision threshold of device i at time t. The per-device decision thresholds are exposed to our server-residing scheduler, which adapts them at run time.

B. The Capacity Metric

To fully take advantage of the server's computational resources and boost throughput, it is important to use batching, *i.e.* processing multiple samples at the same time. To avoid the latency that would arise from waiting for the request queue to reach a specific batch size, we employ dynamic batching [21]. With dynamic batching, we use the maximum batch size that is feasible with the current request queue length. Available batch sizes are $\mathcal{B}=\{1, 2, 4, 8, 16, 32, 64\}$. Due to diminishing returns, in some cases we use a lower maximum batch size, *e.g.* with EfficientNetB3 a batch size of 16 provides higher throughput and lower latency than a batch size of 32 and above.

To calculate the amount of samples that can be processed by the server within a given latency constraint, we introduce the Capacity metric. Based on the inference latency of each batch size and a given latency SLO, the scheduler calculates the maximum amount of samples that can be classified without latency violations. We call that amount Capacity.

To obtain the value of the server's Capacity, we cast the problem as an unbounded variation of the Knapsack problem, where the batch size throughput is analogous to the value/weight ratio. Since we know that the larger the batch size the higher the throughput (up to the point where the server's processing power is saturated), we introduce a greedy

TABLE I: Evaluated DNN Models

Model	Location	Device	Clock Rate	Accuracy	Latency	FLOPs	#Params
MobileNetV2	Low-end	Sony Xperia C5	1.69 GHz	71.85%	31 ms	0.6 B	3.5 M
EfficientNetLite0	Mid-tier	Samsung A71	2.20 GHz	75.02%	43 ms	0.8 B	4.7 M
EfficientNetB0	High-end	Samsung S20 FE	2.73 GHz	77.04%	33 ms	0.8 B	5.3 M
InceptionV3	Server	Tesla T4 GPU	585 MHz	78.29%	15 ms	11.4 B	23.8 M
EfficientNetB3	Server	Tesla T4 GPU	585 MHz	81.49%	25 ms	3.7 B	12.2 M

* See Table 1 in [1] for the detailed resource characteristics of the target mobile phones.

algorithm that adds the largest batch size as many times as possible. Let $\mathcal{B}$ be the pool of batch sizes, then:

$$C = \max_{n} \sum_{j=1}^{|\mathcal{B}|} b_j n_j, \quad \text{s.t.} \quad \sum_{j=1}^{|\mathcal{B}|} L_{\text{inf}}^{b_j} \, n_j \leq L^{\text{SLO}}, \; n_j \geq 0 \quad (3)$$

where C is the Capacity metric, b_j is the j-th batch size, n_j is the amount of times the batch size is used, $L_{\text{inf}}^{b_j}$ is the inference latency of the j-th batch size and L^{SLO} is the latency SLO.

C. Fractional Update

Changing the thresholds of all devices at once could lead to unwanted, sudden oscillations of the system. To achieve a smoother operation of the system and give enough time for each adaptation step to affect the execution, MultiTASC introduces the *fractional update* technique. With fractional update, MultiTASC updates the thresholds of only a certain percentage, denoted by P, of the total number of devices for each update. Thresholds are updated as dictated by Eq. (4). We set the invocation rate of the scheduler to be once every 2 seconds, allowing for update results to affect the system.

$$TC = \begin{cases} -M & \text{if} \quad \bar{b} > \alpha \cdot C \quad \text{and} \quad QL > \alpha \cdot C \\ +M & \text{if} \quad \bar{b} \leq \beta \cdot C \quad \text{and} \quad QL \leq \beta \cdot C \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

where M is the margin by which the threshold of the chosen devices changes, $\bar{b}$ the average of the last L batch sizes used for inference on the server, C the Capacity metric and QL the request queue length. Capacity is weighted using the parameters α and β. If requests accumulate beyond a certain limit despite the dynamic threshold updates, all thresholds are set to 0 until the server is decongested.

D. Device Heterogeneity-Aware Prioritization

MultiTASC explicitly considers device heterogeneity in order to maximize both the average accuracy and the system throughput. Our heterogeneity-aware prioritization strategy updates the threshold values using Eq. (4), but expands the pipeline by selecting which type of devices receive the update. The key insight of our strategy is that when it comes to decreasing the thresholds, MultiTASC prioritizes high-end and mid-tier devices since they host larger models and can maintain higher accuracy even with low thresholds. In contrast, when increasing thresholds, our scheduler prioritizes low-end devices, since they are the ones benefiting the most from higher thresholds due to hosting lighter models.

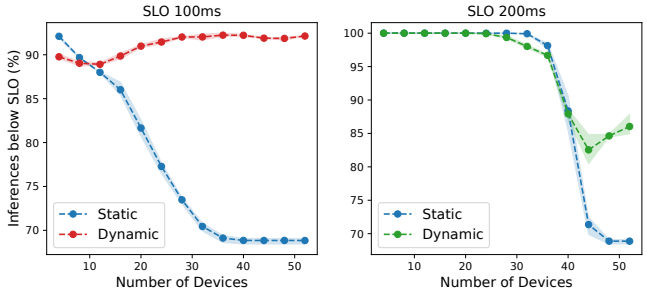

Fig. 3: SLO satisfaction rate for EfficientNetLite0-InceptionV3 pair.

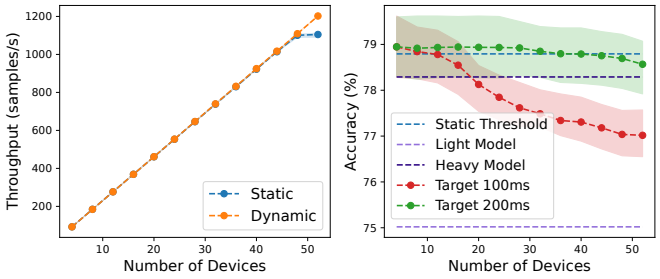

Fig. 4: Throughput and accuracy for EfficientNetLite0-InceptionV3 pair.

V. Evaluation

A. Experimental Setup

To evaluate the performance of MultiTASC, we built a prototype on top of TensorFlow 2.9.1 targeting an edge server and three tiers of mobile devices. The edge server hosts an NVIDIA Tesla T4 GPU, Intel(R) Xeon(R) 2.30GHz CPU and 12GB of RAM. For the clients, we target three smartphones of increasing processing capabilities, namely: Sony Xperia C5 Ultra, Samsung A71 and Samsung S20 FE, representing low-, mid- and high-end clients, respectively. To assess our system across various settings, we conduct simulation-based experiments, varying the target latency SLO and the number of client devices. We measure the average inference time across 200 runs on the target devices for the evaluated models. We follow the same approach for the server-side models across different batch sizes. All on-device measurements were performed using TensorFlow Lite and targeting the CPU of the respective mobile device. For the device-server communication, we employ the AMQP protocol, following the widely used practice for communication between IoT devices.

Models & Datasets. We target the task of 1k-class image classification. Concretely, we use the ImageNet dataset and its 50k-images validation set in our experiments. Table I shows the evaluated models. We obtain the ImageNet-pretrained models as provided by TensorFlow Hub. On the client side, to emulate the common approach where models are selected based on the capabilities of each device, we deploy MobileNetV2, EfficientNetLite0 and EfficientNetB0 on low-, mid- and high-end devices, respectively. On the server side, we chose InceptionV3 and EfficientNetB3 as representative models that are computationally heavy, but provide high accuracy.

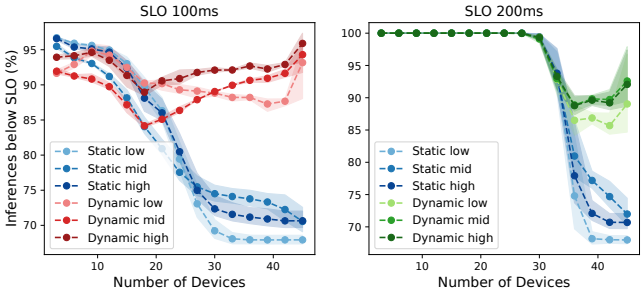

Fig. 5: SLO satisfaction rate for InceptionV3 on the server.

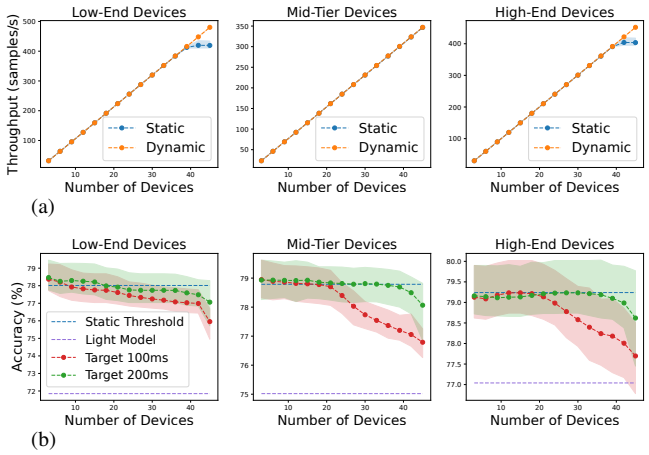

Fig. 6: (a) Throughput and (b) accuracy for InceptionV3 on the server.

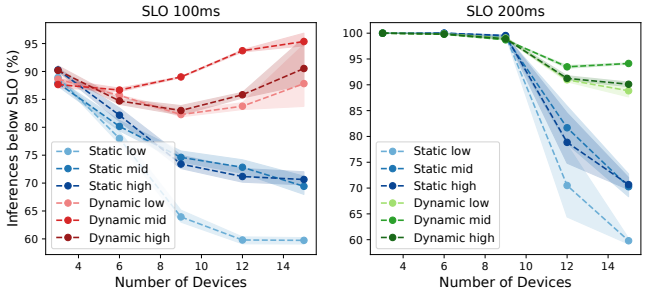

Fig. 7: SLO satisfaction rate for EfficientNetB3.

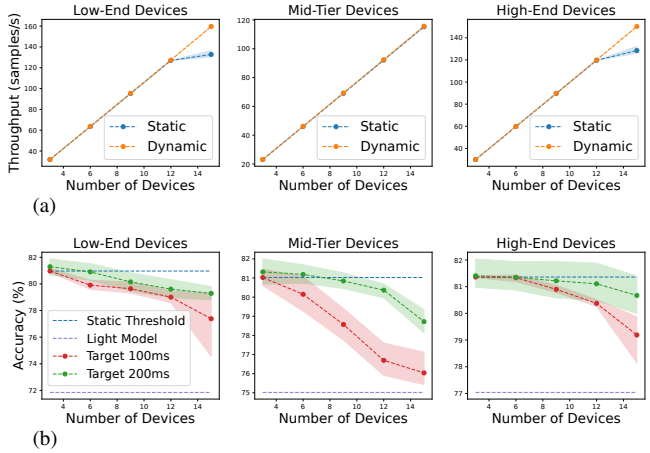

Fig. 8: (a) Throughput and (b) accuracy for EfficientNetB3 on the server.

Evaluation Settings. We focus on two types of device ecosystems: *i)* homogeneous, which comprises devices of equal processing capabilities that host the *same* local model; and *ii)* heterogeneous, which comprises devices of diverse processing capabilities, with each device hosting a model sized to its tier. In the homogeneous scenario, all devices were mid-tier, *i.e.* A71 phones, and ran EfficientNetLite0 with an average on-device inference latency of 43 ms. In the heterogeneous scenario, all three tiers of devices were deployed in equal percentage. In both cases, the dataset of each device consisted of 5,000 randomly selected samples from the last 40,000 images of ImageNet's validation set. Three different seeds were used and the average is reported. The metrics used for the evaluation are: the system throughput, the average accuracy across devices, latency SLO satisfaction rate for 100- and 200-ms SLOs, and scalability in terms of number of devices.

Baseline. As a baseline, we use a scheduler with statically selected thresholds that remain fixed at run time. To choose the static threshold, we use the first 10,000 images of ImageNet's validation set as our calibration set and evaluate all cascade model pairs in terms of accuracy and forwarding probability. As such, we tune the threshold so that approximately 30% of samples are forwarded to the heavy model, providing a balanced accuracy-latency trade-off. In cases where that threshold yielded an accuracy loss of more than 1 pp compared to the highest achievable cascade accuracy, we used the lowest threshold that satisfied the 1 pp limit. This baseline is equivalent to a set of state-of-the-art cascades [5], [6], [9].

B. Evaluation of MultiTASC's Performance

Here, we assess the performance of our scheduler across both device ecosystems. We optimized MultiTASC's parameters P, M, L, α and β using grid search and set them equal to 20%, 0.05, 5, 0.83 and 0.125, respectively.

Homogeneous Scenario. Fig. 3 shows the SLO satisfaction rate comparison for the pair EfficientNetLite0-InceptionV3. For a small amount of devices, MultiTASC keeps the satisfaction rate close to the baseline. As the baseline starts failing at 20 and 40 devices for SLOs of 100 and 200 ms, respectively, MultiTASC gradually reconfigures the thresholds of the forwarding decision functions and manages to keep satisfaction rate high, gaining 15-20 pp across the SLOs. In Fig. 4, we can see that after 45 devices the baseline's throughput is starting to plateau, due to the queue not being served quickly enough. The users of the devices would experience this plateau as excessive response time. In contrast, with MultiTASC, the aggregate throughput continues to increase linearly with a growing number of devices, indicating that the requests are not allowed to accumulate and more devices can be served.

With respect to accuracy, for a small number of devices, MultiTASC slightly increases accuracy. This happens because our approach recognizes that the server is being underutilized and increases the thresholds accordingly. As the number of devices becomes larger, accuracy is traded off to achieve lower latency and sustain higher throughput, but still stays within 2 pp of the baseline. Additionally, we observe that the accuracy achieved by the cascade is in most cases higher than

the accuracy of the heavy model, showcasing the benefits of classifier cascades for the specific model pair. Moreover, we observe the scalability of MultiTASC, serving up to 50 devices without significant loss of accuracy. Different model pairings gave similar results, providing high satisfaction rate, sustaining the throughput and keeping accuracy within acceptable limits. When comparing server-side models, InceptionV3 can serve a larger amount of devices compared to EfficientNetB3, due to its lower computational demands, but EfficientNetB3 achieves higher accuracy when it comes to a lower number of devices.

We further observe that when the on-device inference latency is below the SLO, the lowest limit for the SLO satisfaction rate is the percentage of samples that are not forwarded to the server. For the EfficientNetLite0-InceptionV3 pair, this is around 69% and can be observed when the baseline solution reaches 40 devices for the 100-ms SLO (Fig. 3)

Heterogeneous Scenario. Fig. 5-6 show the SLO satisfaction rate, throughput and accuracy comparison between MultiTASC and the baseline, when targeting a heterogeneous device environment with InceptionV3 as the server-side model. We report the performance results separately for each device tier. When calculating the throughput, different latency targets give similar results, so we present only one. Similarly to the homogeneous case, we observe that the satisfaction rates are maintained high across all tiers of devices, while the baseline leads to catastrophically degraded latency after 30 and 40 devices for SLOs of 100 and 200 ms, respectively (Fig. 5). Our approach achieves gains of approximately 20-25 pp in satisfaction rate across all device tiers. We also observe the throughput plateau, but only for the low- and high-end devices. This is attributed to the fact that the latency of EfficientNetLite0 on the mid-tier devices is $1.32\times$ slower than the other tiers, resulting in mid-tier devices continuing inference after the inference on low-end and high-end devices has finished. This allows the server to decongest since only $1/3$ of the devices remains. Accuracy is again within acceptable values, never dropping below 2 pp from the baseline.

Fig. 7-8 show the performance comparison when EfficientNetB3 is used as the server-side model. Our observations are similar to the previous case of InceptionV3, showcasing the generality of our approach. From this set of experiments, we infer that the highest-performing server-side model is in most cases the higher-throughput one, *e.g.* InceptionV3. Specifically, we observe that InceptionV3 can serve a much larger number of devices while keeping accuracy within acceptable levels compared to the complex, low-throughput EfficientNetB3. EfficientNetB3 achieves higher accuracy when the number of devices is 9 or lower for SLO of 100 ms or 14 and lower for SLO of 200 ms. In all cases, via adjusting the forwarding decision thresholds during execution, MultiTASC manages to keep response times low and system throughput high by gradually trading off accuracy.

VI. Conclusion

This paper presents MultiTASC, a novel scheduler that enables cascade collaborative systems when assisting multiple devices at the same time. By dynamically updating the thresholds of forwarding decision functions on devices during run time, MultiTASC, manages to avoid request accumulation on server-side which keeps response latency below SLO targets and throughput high across different device pool sizes. Moreover, it considers the device heterogeneity problem and is able to maintain performance while accommodating different tiers of devices. The flexibility and scalability of MultiTASC, will help push collaborative cascade systems to enable broader DL deployment in indoor intelligent environments.

References

[1] S. I. Venieris, I. Panopoulos, and I. S. Venieris, "OODIn: An Optimised On-Device Inference Framework for Heterogeneous Mobile Devices," in *SMARTCOMP*, 2021.

[2] S. Laskaridis, S. I. Venieris, A. Kouris, R. Li, and N. D. Lane, "The Future of Consumer Edge-AI Computing," *arXiv*, 2022.

[3] M. Almeida, S. Laskaridis, A. Mehrotra, L. Dudziak, I. Leontiadis, and N. D. Lane, "Smart at What Cost? Characterising Mobile Deep Neural Networks in the Wild," in *IMC*, 2021.

[4] E. Park, D. Kim, S. Kim, Y.-D. Kim, G. Kim, S. Yoon, and S. Yoo, "Big/little deep neural network for ultra low power inference," in *CODES+ISSS*, 2015.

[5] M. Li, Y. Li, Y. Tian, L. Jiang, and Q. Xu, "AppealNet: An Efficient and Highly-Accurate Edge/Cloud Collaborative Architecture for DNN Inference," in *DAC*, 2021.

[6] X. Wang, Y. Luo, D. Crankshaw, A. Tumanov, F. Yu, and J. E. Gonzalez, "IDK Cascades: Fast Deep Learning by Learning not to Overthink," in *UAI*, 2018.

[7] S. I. Mirzadeh and H. Ghasemzadeh, "Optimal Policy for Deployment of Machine Learning Models on Energy-Bounded Systems," in *IJCAI*, 2020.

[8] D. Stamoulis, T.-W. R. Chin, A. K. Prakash, H. Fang, S. Sajja, M. Bognar, and D. Marculescu, "Designing Adaptive Neural Networks for Energy-Constrained Image Classification," in *ICCAD*, 2018.

[9] A. Kouris, S. I. Venieris, and C.-S. Bouganis, "CascadeCNN: Pushing the Performance Limits of Quantisation in Convolutional Neural Networks," in *FPL*, 2018.

[10] T. Nakamura, S. Saito, K. Fujimoto, M. Kaneko, and A. Shiraga, "Spatial- and Time-Division Multiplexing in CNN Accelerator," *Parallel Computing*, 2022.

[11] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," in *CVPR*, 2018.

[12] S. Han, H. Mao, and W. J. Dally, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding," in *ICLR*, 2015.

[13] Y. He, X. Zhang, and J. Sun, "Channel Pruning for Accelerating Very Deep Neural Networks," in *ICCV*, 2017.

[14] G. Hinton, O. Vinyals, and J. Dean, "Distilling the Knowledge in a Neural Network," in *NeurIPS*, 2014.

[15] B. Cox, R. Birke, and L. Y. Chen, "Memory-aware and Context-aware Multi-DNN Inference on the Edge," *Pervasive and Mobile Computing*, 2022.

[16] M. Almeida, S. Laskaridis, S. I. Venieris, I. Leontiadis, and N. D. Lane, "DynO: Dynamic Onloading of Deep Neural Networks from Cloud to Device," *TECS*, 2022.

[17] J. Huang, C. Samplawski, D. Ganesan, B. Marlin, and H. Kwon, "CLIO: Enabling Automatic Compilation of Deep Learning Pipelines across IoT and Cloud," in *MobiCom*, 2020.

[18] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge," in *ASPLOS*, 2017.

[19] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On Calibration of Modern Neural Networks," in *ICML*, 2017.

[20] A. J. Joshi, F. Porikli, and N. Papanikolopoulos, "Multi-Class Active Learning for Image Classification," in *CVPR*, 2009.

[21] A. Ali, R. Pinciroli, F. Yan, and E. Smirni, "BATCH: Machine Learning Inference Serving on Serverless Platforms with Adaptive Batching," in *SC*, 2020.