Dynamic-OFA: Runtime DNN Architecture Switching for Performance Scaling on Heterogeneous Embedded Platforms

Wei Lou* Lei Xun* Amin Sabet Jia Bi Jonathon Hare
Geoff V. Merrett
University of Southampton, UK
{wl4u19, lx2u16, ms4r18, J.Bi, jsh2, g.merrett}@southampton.ac.uk

Abstract

*Mobile and embedded platforms are increasingly required to efficiently execute computationally demanding DNNs across heterogeneous processing elements. At runtime, the available hardware resources to DNNs can vary considerably due to other concurrently running applications. The performance requirements of the applications could also change under different scenarios. To achieve the desired performance, dynamic DNNs have been proposed in which the number of channels/layers can be scaled in real time to meet different requirements under varying resource constraints. However, the training process of such dynamic DNNs can be costly, since platform-aware models of different deployment scenarios must be retrained to become dynamic. This paper proposes Dynamic-OFA, a novel dynamic DNN approach for state-of-the-art platform-aware NAS models (*i.e. *Once-for-all network (OFA)). Dynamic-OFA pre-samples a family of sub-networks from a static OFA backbone model, and contains a runtime manager to choose different sub-networks under different runtime environments. As such, Dynamic-OFA does not need the traditional dynamic DNN training pipeline. Compared to the state-of-the-art, our experimental results using ImageNet on a Jetson Xavier NX show that the approach is up to 3.5x (CPU), 2.4x (GPU) faster for similar Top-1 accuracy, or 3.8% (CPU), 5.1% (GPU) higher accuracy at similar latency.*

1. Introduction

Modern heterogeneous embedded systems typically execute multiple applications concurrently. DNNs are increasingly being executed on embedded platforms due to the superior performance in many applications such as computer vision and natural language processing. Compared to cloud-based solutions, DNN inference on embedded platforms brings many advantages in latency, privacy and connectivity. However, DNN inference is both computationally and memory intensive, making it a challenge to deploy on embedded platforms.

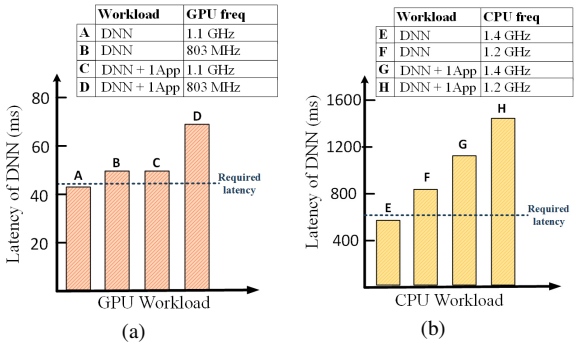


Figure 1: Experimental results illustrating how inference latency constraints characterised at design-time can be violated at runtime by changes in available hardware resources and executing tasks. (a) denotes the latency on GPU and (b) on CPU. The tables above indicate the combinations of workload and frequency.

Many approaches have been proposed to reduce the computational demands of DNN inference, such as platform-aware filter pruning [6, 22] and Neural Architecture Search (NAS) [2, 3]. These approaches produce a static DNN architecture with fixed parameters for the target application performance requirements based on the measurement on a fixed hardware resources. However, since available hardware resources dynamically change at runtime, performance requirements can be violated [19, 20]. Fig 1 illustrates these problems by using experimental results from a Jetson Xavier NX, where bar A represents an optimized DNN model executing on all GPU cores at 1.1GHz to deliver a 50ms target latency. However, under the same target latency, the optimization at design-time is invalid if the operating frequency changes or the DNN shares GPU cores with other applications at runtime, as shown by bars B, C

*Authors contributed equally.

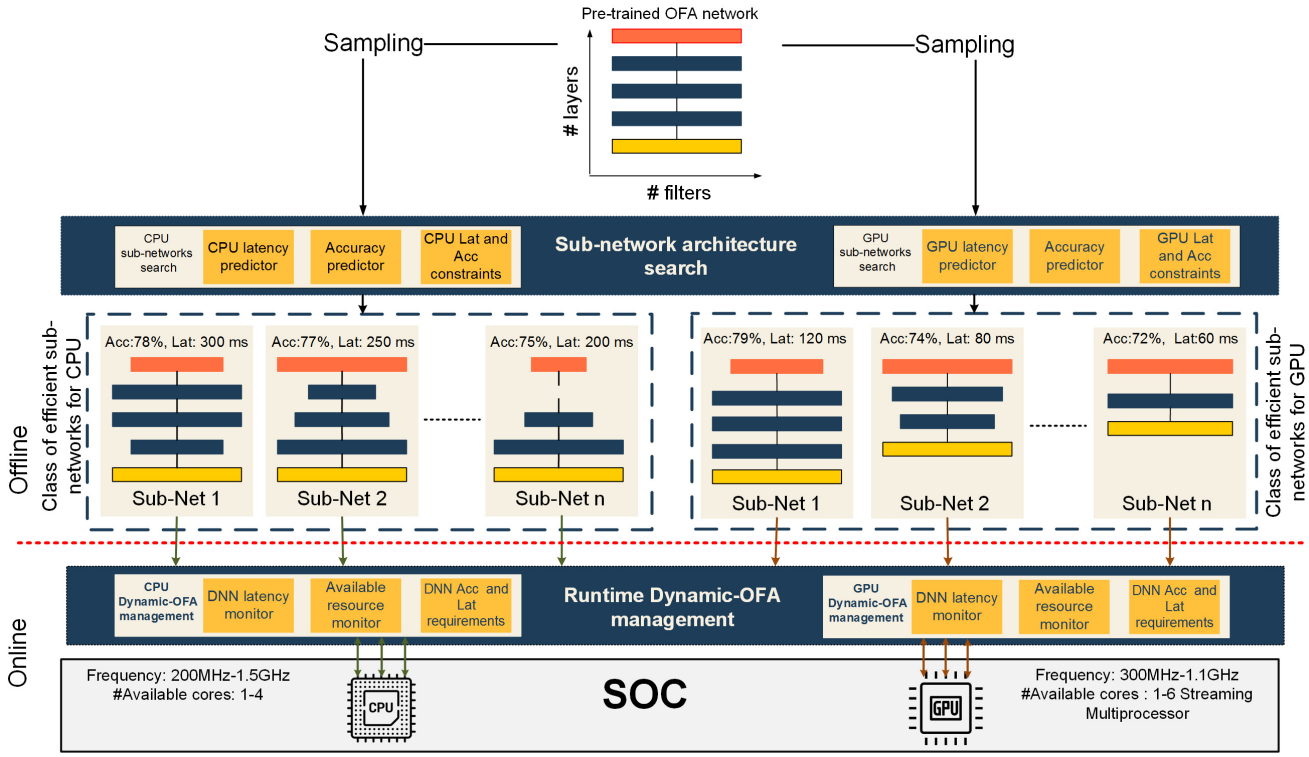


Figure 2: The workflow of Dynamic-OFA. Using pre-trained OFA networks that contain 2×10^{19} sub-network architectures as the backbone, sub-network architectures are sampled from OFA for both CPU and GPU at the offline stage. These architectures have different performance (*e.g.* latency, accuracy) and are stored in a look-up table to build a dynamic version of OFA without any additional training required. Then, at runtime, Dynamic-OFA selects and switches to optimal sub-network architectures to fit time-varying available hardware resources.

and D. The same applies in the CPU case, shown by bars E, F, G and H.

Since both software performance requirements and hardware resource availability can change dynamically at runtime [19, 20], various dynamic DNNs [21, 23–25] have been proposed to address this issue. These dynamic DNNs contain various sub-networks that each have a different accuracy and latency. Given different available hardware resources and application requirements, different sub-networks can be selected. However, two problems still remain in the previous approaches. First, previous dynamic DNNs involve an additional training pipeline which retrains the backbone network (*e.g.* MobileNet [8, 15], ResNet [5]) from scratch. The total training would be costly in real deployment scenarios, since platform-aware pruning/NAS need to be applied before dynamic DNN process, and all model variants of different deployment scenarios have to be retrained to become dynamic. Second, prior works rescale the network architecture to obtain sub-networks by either channel scaling [21, 23–25] or layer scaling [9]. However, the most efficient DNN architectures on CPUs typically have more layers but fewer channels, while the opposite is true for GPUs [3]. Therefore, previous scaling methods limit the application of dynamic DNN on heterogeneous System on Chip (SoC) platforms, since modern SoCs integrate multiple computing elements including CPU, GPU and NPU. As a result, different dynamic DNNs are needed for different heterogeneous computing elements.

This paper proposes Dynamic-OFA, a novel approach to provide dynamic DNN architectures for different software and hardware resource requirements. Dynamic-OFA works with state-of-the-art platform-aware NAS methods (*i.e.* Once-for-all network (OFA)) without requiring any additional model retraining. Moreover, to improve previous dynamic approaches, Dynamic-OFA scales the entire DNN architecture, including width, depth, filter sizes, and input resolutions to provide more efficient DNNs architectures for both CPU and GPU with one shared backbone.

Fig 2 shows the workflow of Dynamic-OFA with two main steps. In the first *offline* step, different sub-networks are extracted from the OFA [2] super-network. Accuracy and latency of extracted sub-networks are evaluated to find a family of efficient sub-networks on the Pareto-front for both CPU and GPU. In the second step, and during *runtime*,

Dynamic-OFA uses a runtime manager to switch between the optimal sub-network based on the runtime accuracy and latency requirements of the application, and available resources on the platform.

The contributions of this paper are:

1. A novel dynamic DNN approach that combines OFA with dynamic DNN. Compared to the state-of-the-art, our experimental results on Nvidia Jetson Xavier NX show that our approach is up to 3.5x (CPU), 2.4x (GPU) faster for similar ImageNet Top-1 accuracy, or 3.8% (CPU), 5.1% (GPU) higher accuracy at similar latency.

2. An improved search algorithm to efficiently search a family of sub-networks from the OFA super-network, by jointly considering inference accuracy and latency.

3. A runtime approach to dynamically switch between sub-networks to meet time-varying performance constraints and/or available hardware resources.

2. Related work

Neural Architecture Search (NAS) Many handcrafted efficient DNN models like MobileNet [8, 15] and ShuffleNet [13] achieve state-of-the-art performance. However, these models need to be further compressed to fit constraints of the target device using platform-aware compression techniques [6,22]. Since compression needs to be conducted for each new constraint, the time required for large-scale deployment is expensive. Furthermore, designing handcrafted DNN models needs expert knowledge and can be a challenging and time-consuming task. NAS automates architecture design, directly searching for the most efficient DNN architectures for target constraints. However, given new platform constraints, DNN models need to be researched and retrained; therefore, the required time is still prohibitive for large-scale deployments. For example, Tan *et al*. [16] introduced MnasNet, a multi-objective NAS approach that uses reinforcement learning to find the architecture for maximizing accuracy and latency on target hardware platforms. MnasNet cost 40,000 GPU hours (Nvidia V100 GPU) to find a single DNN architecture, and this raises a significant issue since the number of combinations of hardware platforms and software performance targets are significant. Cai *et al*. [3] proposed ProxylessNAS, which uses weight-sharing and differentiable architecture search; however, the search cost for each model is still 200 GPU hours which is also significant for large scale deployments.

Cai *et al*. [2] proposed the Once-for-all network (OFA), a NAS approach that supports large-scale deployments. Only one DNN model (a super-network) needs to be trained, and enables 2×10^{19} sub-networks to be used for different software performance requirements on different hardware platforms. While OFA offers a number of significant advantages, the available hardware resources in modern SoCs, containing heterogeneous computing elements that use energy-efficient features such as DVFS and task mapping, vary dynamically. OFA is not a dynamic DNN by default, since its search process is still too time-consuming (*e.g*. hours on Nvidia Jetson Xavier NX). This prevents *real-time* architecture switching at runtime, and motivates us to propose a general approach for building dynamic DNNs for OFA super-networks.

Dynamic DNNs Dynamic DNNs can switch the DNN architecture to fit time-varying available hardware resources and software performance requirements. A variety of approaches have been proposed to change the width of the backbone networks, like 'Slimmable' [25], 'Universally Slimmable' [24] and 'AutoSlim' [23]. These models can run different active channels and achieve instant and adaptive accuracy-latency trade-offs. MutualNet [21] shows improved performance by adding input resolutions with width as switchable dimensions. For different constraints, different sub-networks with varying widths and resolutions can be chosen and built as a query table. Approaches such as MSDNet [9] can change the depths of the backbone networks. However, the most efficient models for different hardware resources are usually different in architecture. For example GPUs prefer shallow and wide DNN architectures with early pooling, while CPUs prefer deep and narrow DNN architectures with late pooling [3]. The switchable dimensions for previous approaches are either width or depth; therefore, they cannot provide the most efficient architecture for both CPU and GPU on the modern SoCs. The Dynamic-OFA approach presented in this paper switches in four dimensions (width, depth, filter sizes, and input resolutions) to provide efficient architectures for both GPUs and CPUs by using a share backbone model.

3. Dynamic-OFA

As illustrated in Fig 2, Dynamic-OFA use the following process: a pre-trained OFA model [2] is used as the backbone (super-network), as introduced in Section 3.1. As an offline process, sub-networks with different latency and accuracy are sampled from the pre-trained super-network at design-time (Section 3.2). The sampling process is conducted separately for CPU and GPU because the most efficient sub-network architectures are different for heterogeneous computing resources. The batch-normalization (batch-norm) parameters of each sub-network are recalculated and stored (Section 3.3). At runtime, the sampled sub-network architecture can be switched to meet latency and accuracy requirements on time-varying available hardware resources (Section 3.4). No additional training is needed, and since all sub-networks share the same OFA model as

the backbone, only a single model needs to be stored on the device.

3.1. Backbone Network: Once-For-All (OFA)

The OFA network [2] supports 2×10^{19} sub-networks with different sizes by a single super-network, covering four dimensions of the DNN architecture: depth, width, kernel, and input resolutions. OFA uses progressive shrinking to train the super-network model. Progressive shrinking optimizes super-network parameters such that each supported sub-network maintains almost the same level of accuracy as independently training a network with the same architecture configuration. After training the super-network model, OFA uses an evolutionary search algorithm [14] to find sub-network architectures in a super-network model with different accuracy and latency trade-offs. OFA only requires a single set of parameters to store all sub-networks, since all sub-networks share the same parameters. OFA is able to select sub-network architectures for different hardware platforms, and then calibrates the batch-norm parameters for those selected sub-networks. The process of searching for sub-networks and calibrating batch-norm parameters take minutes on a GPU or hours on a CPU[1]. The actual search time at runtime depends on how tight the constraints are, and how the search problem shares hardware resources with other applications, and hence OFA cannot be used to adapt to resource changes at runtime.

Our approach identifies a family of efficient sub-networks on the Pareto-front for each computation element in a heterogeneous platform, and pre-calculates batch-norm parameters for those sub-networks offline. This allows switching the network architecture at runtime. Furthermore, since Dynamic-OFA conducts all the sub-network search at design-time on a server, the search cost is dramatically reduced to that of the single search cost, multiplied by the number of sub-networks.

3.2. Optimal Sub-network Architecture Search

OFA's search is solely based on the latency constraint, as it selects sub-networks with the highest accuracy that meets a latency constraint. It uses a combination of a random search and evolutionary search: (1) a random search algorithm finds sub-networks that meet the latency constraints; (2) the evolutionary search performs refinement to mutate those random selected sub-network configurations, and then keeps the one which has the highest accuracy. Although this approach works for a single sub-network architecture search, it is not efficient for searching a family of sub-networks. To build dynamic-DNNs, multiple sub-network architectures with different accuracy-latency trade-offs are required to be dynamically switched at runtime. If we only use latency as the constraint (which is the original OFA search method), the random search algorithm often outputs the sub-networks with the same accuracy for different latency constraints. For example, a 75% accuracy, 30 ms model is searched under 30 ms, 40 ms and 50 ms constraints, but ideally we want models with higher accuracy when the latency is relaxed. The OFA algorithm only requires the latency of searched sub-networks to be lower than constraint but not for accuracy, and this prevents us from getting the Pareto trade-off curve. In our random search algorithm 1, by adding accuracy as a hard constraint, it forces the search algorithm to find sub-networks under certain latency constraints and have better accuracy (*e.g.* 75% accuracy under 30 ms, 76 % accuracy under 40 ms, 77% accuracy under 50 ms etc).

In Algorithm 1, accuracy and latency predictors are used to reduce the time cost of the search process. We use the original accuracy predictor that comes with the pre-trained OFA model; it is a three-layer neural network trained using 5000 network architectures and their accuracy data. However, since we want the dynamic DNN to have 1% accuracy steps between sub-networks, the accuracy of all searched models are re-measured on the ImageNet 50K validation set since the original OFA accuracy predictor is not accu-

Algorithm 1 Random Search Algorithm of Dynamic-OFA.

Input: maximum accuracy constraint (Acc_{max}), accuracy suitable range (Acc_r), latency (Lat), maximum latency constraint (Lat_{max}), initial latency (Lat_{init}), extracted sub-networks ($Parents$), latency constraint increment (Lat_{add}), n iterations of searching (itr_n), maximum iterations number (itr_{max})

Output: qualified network architecture ($Arch$), predicted accuracy (Acc_p), predicted latency (Lat_p)

function RANDOM_SEARCH
  $Lat \leftarrow Lat_{init}$
  $i = 0$
  while $i < itr_{max}$ **do**
    if $i\%itr_n = 0$ **and** $Lat < Lat_{max}$ **then**
      $Lat \leftarrow Lat + Lat_{add}$
    end if
    $Arch = Random\_Arch()$
    $Acc_p = Predict\_Accuracy(Arch)$
    $Lat_p = Predict\_Latency(Arch)$
    if $Acc_p > Acc_{max} - Acc_r$ **and** $Lat_p < Lat$ **then**
      if $Arch$ **not in** $Parents$ **then**
        return $Arch, Acc_p, Lat_p$
      end if
    end if
    $i \leftarrow i + 1$
  end while
  end function

[1]The platform is Nvidia Jetson Xavier NX. The time includes measure the accuracy of selected sub-network on ImageNet 50k validation set, since the accuracy prediction is not accurate enough, more details in section 3.2. Batch-norm is calculated on 2000 images.

rate enough (normally 2-4% higher than our measurement). We adapted the latency predictor in OFA for Nvidia Jetson Xavier GPU and CPU as a heterogeneous platform. A lookup table is used to record the latency for different DNN computing operations such as convolution, pooling, input and output. The latency of all operations is profiled at design-time by timing the execution time of all operations at all size variants. Moreover, the separate lookup table is built for different hardware platforms and different computing cores on these platforms (*e.g.* CPU, GPU). The latency of a sub-network can be predicted by summing up the latency of sub-network operations. The latency are also re-measured on devices for more accurate data.

To search for sub-networks, the latency constraint is initialized with a user-defined value Lat_{init}, shown in algorithm 1 line 2. An accuracy constraint is also fixed at a user-defined range $[Acc_{max} - Acc_r, Acc_{max}]$. Then, *RANDOM_SEARCH* starts searching for sub-networks by randomly generating sub-network architectures $Random\_Arch()$ (line 8), *i.e.* different depths, width, kernel sizes and input resolutions. $Predict\_Latency()$ and $Predict\_Accuracy()$ predict the latency and accuracy of each sub-network in lines 9 and 10. If none of the found architectures provide a latency of less than Lat and an accuracy in the range of $[Acc_{max} - Acc_r, Acc_{max}]$ for n iterations of searching (itr_n), the latency constraint is relaxed by increasing Lat_{add} (line 6). However, this relaxation cannot exceed a maximum threshold Lat_{max}. The discovered network architecture is stored in $Parents$ (line 13), which is subsequently used for mutation operations of the evolutionary search algorithm to search for the best sub-network.

We use the state-of-the-art pre-trained OFA [2] super-network model as the backbone network for image classification. This model uses MobileNet v3 [7] as the fundamental network architecture and is trained with the progressive shrinking approach [2] to enable 2×10^{19} sub-networks. A random search, Algorithm 1, is used within the evolutionary search algorithm[2] [14] to search for potential efficient sub-networks. It is notable that for given accuracy and latency constraints, numerous sub-network architectures can be found in the search process for each computation element, *e.g.* CPU, GPU in a heterogeneous platform. However, most architectures are sub-optimal because they provide lower accuracy or higher latency. The optimal sub-network architectures are on the Pareto curve of accuracy-latency scatter plot, and they are selected for building the Dynamic-OFA. Fig 3 shows the accuracy-latency scatter plot of extracted sub-networks for the GPU of Nvidia Jetson Xavier NX platform. Among obtained sub-networks, only those located on the Pareto-front (red line) are considered as a family of optimal sub-networks for use by Dynamic-OFA.

[2]The same evolutionary search algorithm that the OFA used.

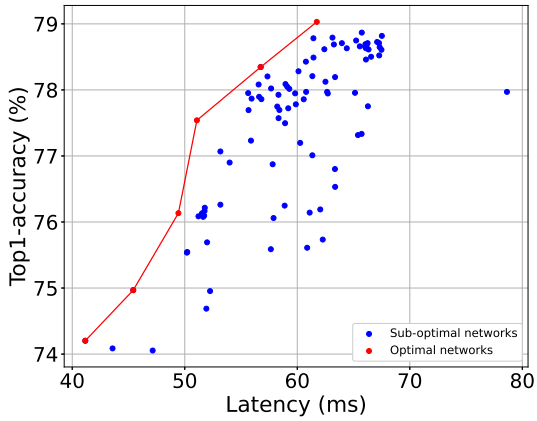


Figure 3: The accuracy and latency of potential sub-network architectures. The optimal architectures are on the Pareto curve.

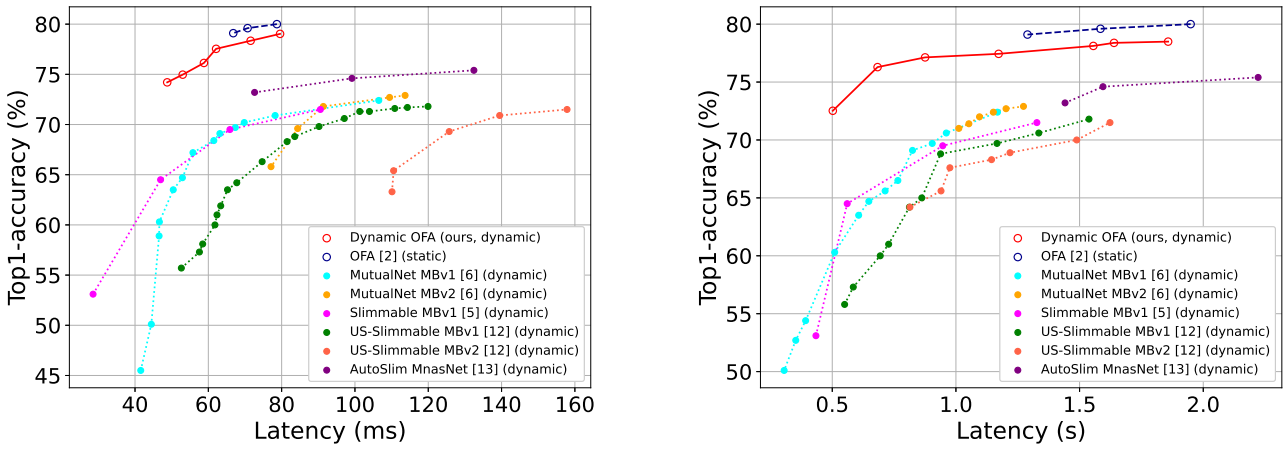
3.3. Batch-norm Calibration

Searching sub-networks at runtime is computationally expensive. Therefore, to make OFA dynamic at runtime, the search algorithm is run offline, and the optimal sub-network configurations are stored as a lookup table to achieve fast architecture switching at runtime. Moreover, the batch-norm parameters [10] differ for each sub-network architecture and need to be recalculated for different sub-networks. However, the calibration can take minutes on GPU and hours on CPU, which is unacceptable for real-time DNN architecture switching at runtime. Since we have selected a small number of optimal sub-network architectures, the batch-norm parameters can be pre-calculated at design-time. At runtime, sub-network architectures can be switched to meet different performance requirements on time-varying available hardware resources. The pre-stored batch-norm parameters (about 2KB for each sub-network) can be loaded, significantly reducing the architecture switching time.

3.4. Runtime Architecture Switching

At runtime, sub-network architectures of Dynamic-OFA can be switched to meet different performance requirements on time-varying available hardware resources. Because of the stored sub-network architectures and their batch-norm parameters, Dynamic-OFA supports real-time architecture switching for the dynamic computing environments. We consider two operating scenarios:

1. When a single dynamic-OFA runs on the device, a look-up table is used to directly find the sub-network 'level' to meet different user-defined accuracy and latency constraints. This is a similar approach to previ-



(a) GPU (b) Single-core CPU

Figure 4: Experimental results of Dynamic-OFA's accuracy-latency trade-offs on the a) GPU and b) CPU of the Nvidia Jetson Xavier NX. State of the art approaches (shown in different colours) are also plotted, including static [2] and dynamic DNNs [21,23–25]. Dynamic-OFA is 2.4x (GPU) and 3.5x (CPU) faster (at similar accuracy) or has 5.1% (GPU) and 3.8% (CPU) higher Top-1 ImageNet accuracy (at similar latency) than AutoSlim-MnasNet [23].

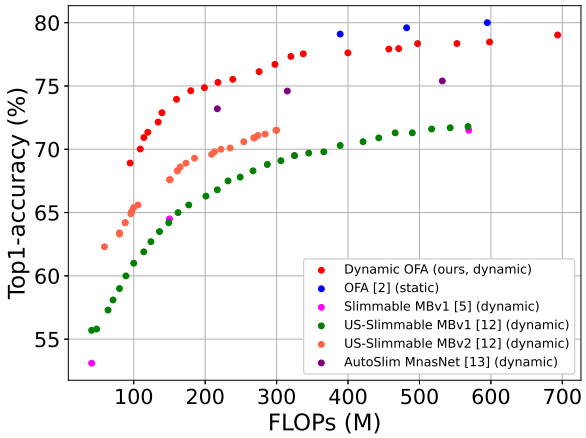


Figure 5: Experimental results of Dynamic-OFA's accuracy-FLOPs trade-offs. Compare with state-of-the-art approaches, Dynamic-OFA achieves up to 50% FLOPs reduction (at similar accuracy) and 2.95% higher Top-1 ImageNet accuracy (at similar FLOPs) than AutoSlim-MnasNet [23].

ous dynamic DNNs like Slimmable [23–25] and MutualNet [21].

2. When two workloads share the same GPU resources (*e.g.* one dynamic-OFA with another workload, or two dynamic-OFAs), a reactive control approach is used instead of a look-up table, due to the increased state space. The latency of the current Dynamic-OFA model is measured over a certain time interval using a sliding window. The RTM continually monitors the latency of all Dynamic-OFA workloads at runtime. When a latency constraint violation occurs, the RTM gradually changes sub-network levels while observing whether latency constraints are subsequently met. Although the design-time profiled look-up table is not used, the trade-off between sub-networks still holds.

The overhead of the RTM includes the monitoring of latency and the calculation of average latency, neither of which are computationally expensive in comparison to the task being monitored. Detailed overhead measurement will be shown in the next section.

4. Experimental evaluation

Dynamic-OFA is developed for both the CPU and GPU of the Nvidia Jetson Xavier NX platform, and the accuracy and latency are empirically measured. The results are compared with both state-of-the-art static OFA backbone [2] and dynamic DNNs [21, 23–25]. We have also executed other applications (*e.g.* online model training [1] or a 2nd Dynamic-OFA) alongside our Dynamic-OFA, to demonstrate runtime management of the sub-network architectures for accommodating both applications on the same GPU.

4.1. Experimental Setup

We deploy and evaluate Dynamic-OFA on the Nvidia Jetson Xavier NX. The platform has a 384-core GPU, a 6-

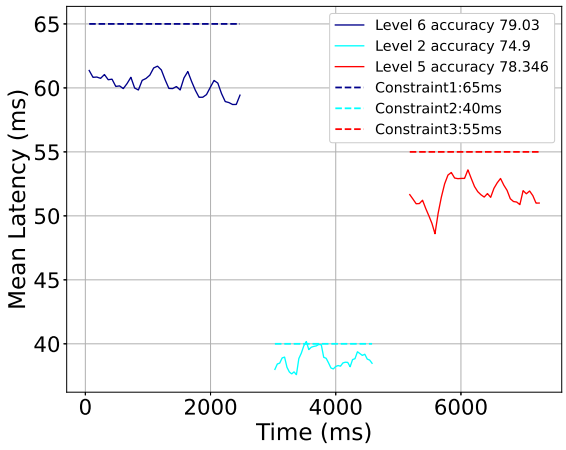


Figure 6: Runtime management of dynamic-OFA to meet the dynamic latency constraints. Latency constraints are denoted by dashed lines. Different sub-networks with different accuracies are denoted using different colours.

core CPU, and 8 GB of unified memory. The frequency is locked at the maximum[3]. A single-core CPU is used for all CPU experiments since the program is single-threaded. We use the open-source pre-trained OFA model (MobileNet v3, $w = 1.2$ [2]) as the backbone for Dynamic-OFA. We evaluate the accuracy of Dynamic-OFA on the ImageNet [4] 50K validation images for a more accurate measurement than OFA's accuracy predictor.

4.2. Top-1 Accuracy and Latency

Dynamic-OFA contains six optimal sub-networks for the GPU, which span a 20-100 ms latency range (Fig 4a). The same model also contains seven optimal sub-networks for the CPU, spanning a range from 500-3000 ms (Fig 4b). The search range for Top-1 accuracy is 70% to 85% for both CPU and GPU. In this paper, we denote sub-networks using "levels." Different levels have different latency and accuracy. For example, level 1 is the sub-network architecture that has the lowest latency and accuracy. On GPU, level 6 is the architecture that has the highest latency and accuracy, whereas, on CPU, level 7 has the highest latency and accuracy.

Dynamic-OFA achieves up to 79% Top-1 accuracy on ImageNet; close to the 80% accuracy of the static OFA backbone [2] which is considered the state-of-the-art under mobile settings (<600M MACs). The reason our Dynamic-OFA has 1% lower accuracy is because OFA applies extra fine-tuning steps for its individual model, whereas our model cannot be easily fine-tuned since it contains 13 different sub-networks. Compared to state-of-the-art dynamic DNNs [23], Dynamic-OFA is up to 2.4x (GPU) and 3.5x (CPU) faster (at a similar accuracy) or has 5.1% (GPU) and 3.8% (CPU) higher Top-1 ImageNet accuracy (at a similar latency). Furthermore, since previous dynamic DNNs are designed using FLOPs rather than latency, they can access less hardware information during design-time. Hence we also compared our Dynamic-OFA with them on a accuracy-FLOPs trade-off curve (Fig 5). Dynamic-OFA achieves up to 50% FLOPs reduction (at similar accuracy) and 2.95% higher Top-1 ImageNet accuracy (at similar FLOPs) than prior art.

[3]Due to the intensive nature of DNN workloads, current DVFS governors will typically operate the SoC at the maximum frequency.

Model	Time
Static OFA [2]	minutes to hours
MutualNet-MBv2 [21]	17 ms
AutoSlim-MnasNet [23]	33 ms
Dynamic OFA	**73 ms**

Table 1: Comparison of average runtime DNN architecture switching time on GPU

At runtime, different sub-network architectures can be used to meet dynamic software performance requirements and available hardware resources. The runtime architecture switching time is shown in Table 1; Dynamic-OFA supports real-time DNN architecture switching, its switching time is much faster than the static OFA model [2] since we only do search in a small subset of sub-networks. Furthermore, loading the pre-calculated batch-norm takes only 2ms. Dynamic-OFA is only slightly slower than AutoSlim [23] and MutualNet [21] due to larger memory footprint, but it has a much better accuracy-latency/FLOPs trade-offs (Figs 4 and 5). Such a small difference is not a considerable issue in real applications, since new operating environments (*i.e.* software performance requirements, available hardware resources) normally last much longer (*e.g.* minutes to hours).

4.3. Dynamic Performance Requirements

The sub-network architectures of Dynamic-OFA can be switched to meet the dynamic latency constraints at runtime. When only Dynamic-OFA is running on the device, all hardware resources are available to it. A look-up table that contains all accuracy-latency trade-offs can be obtained at design-time. RTMs can choose different operating points at runtime. As shown in Fig 6, the Dynamic-OFA model is deployed on the GPU and runs at the level 6 architecture while the latency constraint is 65ms at the beginning. At around 2500 ms, the latency constraint gets increased to 40 ms. Therefore the sub-network architecture is switched to level 2 for speedup at the cost of trading 4.13% accuracy. Then, accuracy is recovered when the latency constraint is later reduced to 55ms.

The RTM, uses the sliding window to calculate the av-

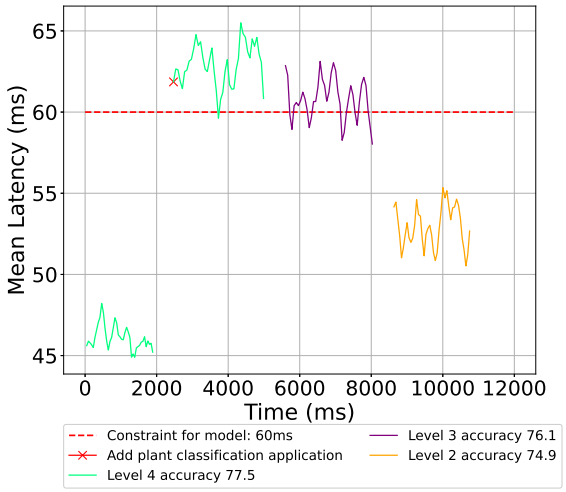


Figure 7: At $t = 0$, a single DNN is executing. After 2.5 s, a second application begins executing, reducing available GPU resource and impacting on the DNN inference latency. The runtime adapts, reducing the DNN accuracy until the performance constraint is met.

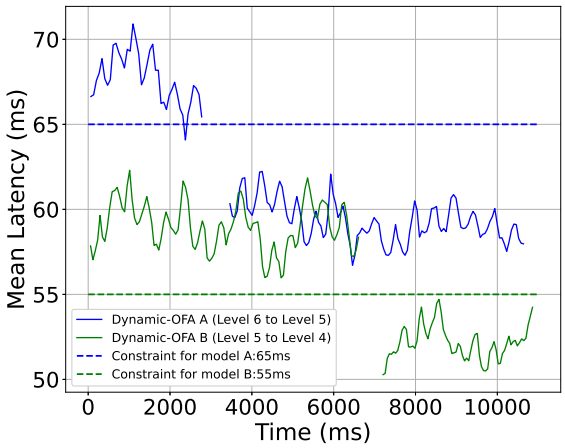


Figure 8: Runtime management of two concurrently executing Dynamic-OFA models, each with its own latency constraint (dashed lines).

erage latency every 10 images. Based on the latency of our model on the GPU, the reaction time of the RTM is approximately 1-2s (*i.e.* about 50 image classifications). The time overhead of the RTM is around 15 ms for each architecture switching, including the latency of monitoring 50 images and calculating the average time across the sliding windows.

4.4. Managing Concurrent Workloads

The sub-network architectures of Dynamic-OFA can be switched to meet software performance constraints while fewer computing resources are available. Fig 7 shows results where GPU computing resources are shared between Dynamic-OFA and a training task of a static DNN. The static DNN is an Nvidia open-source plant classification based on ResNet-18 [1]. The training tasks starts to run at 2500 ms (donated by 'X'), and Dynamic-OFA becomes slower since fewer GPU cores are available to it. The sub-network architecture is gradually switched from level 4 to level 2 to meet the latency constraint by trading 2.6% accuracy.

Two Dynamic-OFA models can also coexist of when they share the same GPU. Fig 8 shows two Dynamic-OFA models deployed on the same GPU, and their latency constraint become violated. The sub-network architecture of two Dynamic-OFA models are switched collectively so that the latency constraint of both models can be met, while keeping accuracy as high as possible. The constraints for model A and model B are 65 ms and 55 ms, respectively. Model A runs at the highest level at the beginning, and model B runs at level 5. To match the latency constraints, model A switches to level 5, but model B is still slower than the constraint, so model B switches to level 4: leaving both models meeting their constraints.

5. Conclusions

This paper has proposed Dynamic-OFA, a novel dynamic DNN approach. Dynamic-OFA brings together two concepts: the OFA model and dynamic DNNs, which provide solid improvements over the previous state-of-the-art. Dynamic-OFA does not require any additional dynamic DNN model retraining and has the architecture flexibility for all heterogeneous computing elements with a share backbone. We empirically evaluated our approach against the start-of-the-art, our results show that our approach can provide better accuracy-latency trade-offs, up to 3.5x (CPU), 2.4x (GPU) faster for similar ImageNet Top-1 accuracy, or 3.8% (CPU), 5.1% (GPU) higher accuracy at similar latency.

Dynamic-OFA is a general approach for building dynamic DNNs, and the backbone network could be any super-networks trained by the OFA training pipeline. Our future work will investigate other applications such as network for IoT devices [12], transformers for natural language processing (NLP) tasks [18], generative adversarial networks (GANs) [11], 3D DNNs [17], etc.

6. Acknowledgements

This work was supported in part by the Engineering and Physical Sciences Research Council (EPSRC) under Grant EP/S030069/1. Experimental data can be found at: https://doi.org/10.5258/SOTON/D1804. Code is available open-source on https://github.com/UoS-EEC.

References

[1] Nvidia Jetson Example: Re-training on the PlantCLEF Dataset. [Online]. Available: https://github.com/dusty-nv/jetson-inference/blob/master/docs/pytorch-plants.md. 6, 8

[2] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. In *ICLR*, 2020. 1, 2, 3, 4, 5, 6, 7

[3] Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In *ICLR*, 2019. 1, 2, 3

[4] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *CVPR*, 2009. 7

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 2

[6] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. AMC: Automl for model compression and acceleration on mobile devices. In *ECCV*, 2018. 1, 3

[7] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for MobileNetv3. In *ICCV*, 2019. 5

[8] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 2, 3

[9] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens Van Der Maaten, and Kilian Q Weinberger. Multi-scale dense convolutional networks for efficient prediction. In *ICLR*, 2018. 2, 3

[10] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *CVPR*, 2015. 5

[11] Muyang Li, Ji Lin, Yaoyao Ding, Zhijian Liu, Jun-Yan Zhu, and Song Han. GAN compression: Efficient architectures for interactive conditional GANs. In *CVPR*, 2020. 8

[12] Ji Lin, Wei-Ming Chen, Yujun Lin, John Cohn, Chuang Gan, and Song Han. MCUNet: Tiny deep learning on IoT devices. In *NeurIPS*, 2020. 8

[13] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient CNN architecture design. In *ECCV*, 2018. 3

[14] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *AAAI*, 2019. 4, 5

[15] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018. 2, 3

[16] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *CVPR*, 2019. 3

[17] Haotian Tang, Zhijian Liu, Shengyu Zhao, Yujun Lin, Ji Lin, Hanrui Wang, and Song Han. Searching efficient 3D architectures with sparse point-voxel convolution. In *ECCV*, 2020. 8

[18] Hanrui Wang, Zhanghao Wu, Zhijian Liu, Han Cai, Ligeng Zhu, Chuang Gan, and Song Han. HAT: Hardware-aware transformers for efficient natural language processing. In *ACL*, 2020. 8

[19] Zirui Xu, Fuxun Yu, Chenchen Liu, and Xiang Chen. Reform: Static and dynamic resource-aware DNN reconfiguration framework for mobile device. In *DAC*, 2019. 1, 2

[20] Lei Xun, Long Tran-Thanh, Bashir M Al-Hashimi, and Geoff V Merrett. Optimising resource management for embedded machine learning. In *DATE*, 2020. 1, 2

[21] Taojiannan Yang, Sijie Zhu, Chen Chen, Shen Yan, Mi Zhang, and Andrew Willis. MutualNet: Adaptive convnet via mutual learning from network width and resolution. In *ECCV*, 2020. 2, 3, 6, 7

[22] Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. NetAdapt: Platform-aware neural network adaptation for mobile applications. In *ECCV*, 2018. 1, 3

[23] Jiahui Yu and Thomas Huang. Autoslim: Towards one-shot architecture search for channel numbers. *arXiv preprint arXiv:1903.11728*, 2019. 2, 3, 6, 7

[24] Jiahui Yu and Thomas S Huang. Universally slimmable networks and improved training techniques. In *ICCV*, 2019. 2, 3, 6

[25] Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks. In *ICLR*, 2019. 2, 3, 6