Contention Grading and Adaptive Model Selection for Machine Vision in Embedded Systems

BASAR KUTUKCU, University of California, San Diego, USA
SABUR BAIDYA, University of Louisville, USA
ANAND RAGHUNATHAN, Purdue University, USA
SUJIT DEY, University of California, San Diego, USA

Real-time machine vision applications running on resource-constrained embedded systems face challenges for maintaining performance. An especially challenging scenario arises when multiple applications execute at the same time, creating contention for the computational resources of the system. This contention results in increase in inference delay of the machine vision applications, which can be unacceptable for time-critical tasks. To address this challenge, we propose an adaptive model selection framework that mitigates the impact of system contention and prevents unexpected increases in inference delay by trading off the application accuracy minimally. The framework has two parts, which are performed pre-deployment and at runtime. The pre-deployment part profiles the system for contention in a black-box manner and produces a model set that is specifically optimized for the contention levels observed in the system. The runtime part predicts the inference delays of each model considering the system contention and selects the best model according to the predictions for each frame. Compared to a fixed individual model with similar accuracy, our framework improves the performance by significantly reducing the inference delay violations against a specified threshold. We implement our framework on the Nvidia Jetson TX2 platform and show that our approach achieves greater than 20% reductions in delay violations over the individual baseline models.

CCS Concepts: • **Computer systems organization → Embedded systems**; • **Computing methodologies → Neural networks**.

Additional Key Words and Phrases: adaptive computing, resource contention, DNN model selection

ACM Reference Format:
Basar Kutukcu, Sabur Baidya, Anand Raghunathan, and Sujit Dey. 2022. Contention Grading and Adaptive Model Selection for Machine Vision in Embedded Systems. *ACM Trans. Embedd. Comput. Syst.* 1, 1, Article 1 (January 2022), 28 pages. https://doi.org/10.1145/3520134

1 INTRODUCTION

Modern machine vision systems involve complex deep learning based algorithms [14, 19, 25, 29] that need significant computing resources to run under reasonable time limits. However, this is very challenging when the algorithms need to be realized in a resource-constrained system. Moreover, in many cases, the computing system running the machine vision application shares resources with other coexisting computing loads. For example, connected and autonomous vehicles process camera data together with RADAR and/or LiDAR data on the same computing system for better

This article is an extended version of [17].

Authors' addresses: Basar Kutukcu, bktkc@ucsd.edu, University of California, San Diego, USA; Sabur Baidya, sabur.baidya@louisville.edu, University of Louisville, USA; Anand Raghunathan, raghunathan@purdue.edu, Purdue University, USA; Sujit Dey, dey@ucsd.edu, University of California, San Diego, USA.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2022 Association for Computing Machinery.
1539-9087/2022/1-ART1
https://doi.org/10.1145/3520134

fused perception in crowded areas or in the presence of obstacles [6, 15, 36]. In such scenarios, the machine vision workload can contend with the other workloads for the computing system resources, further increasing the application latency. While increasing the priority of certain tasks can improve their latency, it can cause a starvation for the other tasks running on the same system. Instead, an alternative approach is to handle contention by adapting the machine vision workload to best utilize the computing resources available in the presence of contention. The machine vision is thus realized by choosing an appropriate model from a set of neural network-based image classification models, fitting the available computing resources. In this work, we examine the effects of contention on the image classification application, and propose a contention-aware adaptive model selection framework that minimally compromises the accuracy of the image classification application while satisfying the latency requirements.

Several previous efforts have explored reduced complexity models that fit within the constrained capabilities of embedded systems [20, 44]. These approaches typically involve a tradeoff between compute/storage requirements and model accuracy. However, contention-impacted systems present a moving target, since the contention levels may vary over time, effectively presenting different resource levels that we would like to fully utilize in order to achieve the lowest impact on application accuracy. Thus, the major challenges in a real-time system with contention are (i) to accurately predict the level of contention in the system, and (ii) adapt the application accordingly maintaining the performance constraint of the application. As many modern systems may not allow applications to access low-level system information in real-time due to security concerns or complexity of the platform, herein, we infer the level of contention from its impact on the application performance. We propose an application-level data-driven predictive framework for contention-aware adaptive model selection that aims to minimize the cost of system resources and overhead of our framework, while maintaining system performance stability in presence of dynamic workload variations.

Now, as the model selection framework needs to load a number of pre-trained models in memory and dynamically select a model at runtime, the length of the model set can not only impact the memory overhead but also the model switching cost and stability. Hence, we also propose a contention grading mechanism that intelligently selects the appropriate deep learning models in the model-set used by the adaptive model selection framework.

The main contributions of this work are as follows:

- An application-level profiler for system contention and a methodology to automatically regenerate the profiled system contention in a controlled environment.
- An offline model set pruning methodology that selects the optimal models for a given system contention profile and specific user requirements.
- A runtime model selection mechanism for an image classification application that adapts based on the system contention to stay below a predefined delay threshold.

We implement the framework on the Nvidia Jetson TX2 platform and show the advantage of our adaptive model selection framework in dynamic contention scenarios. We empirically show that our model pruning methodology improves the runtime model selection performance resulting in better tradeoff between latency and accuracy, and also reduction in memory overhead. The framework also shows the advantage of model selection from a set of independently trained models compared to early exit techniques [33, 35].

2 RELATED WORK

The challenge of enabling deep learning models on resource constrained devices has been extensively researched in previous efforts. One of the main techniques is designing ground-up efficient models that require less resources than high accuracy models while sacrificing accuracy as little as

possible [13, 27, 34, 43]. In [13], a squeeze-and-expand architecture is created by using 1x1 and 3x3 filters together. In [43], a channel shuffling operation is created after 1x1 group convolutions. In [27], linear bottlenecks and inverted residuals are used together with depthwise convolutions. In [34], the DenseNet architecture [12] is modified to design an efficient deep neural network. Unlike our framework, these approaches create a single efficient model and do not consider dynamic changes in computing resources available due to contention in the system. In the presence of contention, their inference delays will still increase unexpectedly. Similarly, when there is no contention, they miss the opportunity to use the available resources for higher accuracy since the design is fixed. These efforts are orthogonal to our work, and our proposed approach and framework will apply to all neural network models, including these compute-efficient models as well.

Another way of creating efficient deep learning models is to use quantization to come up with efficient designs out of any neural network model. Quantization decreases the precision of neural network weights and activations to improve efficiency. There are several efforts addressing neural network quantization in the literature [8]. Quantization can be considered in two categories, namely quantization-aware training [3, 23, 30, 44] and post-training quantization [1, 21]. Quantization is applied during training in the quantization-aware training methods. In [3], the weights and activations are constrained to -1 and +1. In [44], quantization clusters are learned during training together with weights. In [23], the weights and activations are quantized to binary values to allow XNOR and bitcount implementation of expensive operations. In [30], quantization is designed for graph neural networks. Quantization is applied after training in post quantization methods. In [1], 3 post training techniques are defined and their combinations are used for 4-bit quantization. In [21], the weights in different layers of a neural network are scaled to decrease the error caused by quantization. These approaches do not consider contention either. Therefore, their inference delays are vulnerable to dynamic system contention as well.

Pruning is another method that is similar to quantization in terms of its objectives and design flow. The less important weights of neural networks are pruned to create efficient neural network models. Pruning methods can be categorized into two main methods, namely unstructured and structured pruning [2]. Unstructured pruning removes individual parameters or neurons [9]. Structured pruning removes the coarse-grained structures such as filters or channels [20]. The effect of contention on pruning methods is similar to the quantization. They do not consider contention and their inference delays can be affected by contention.

All the previous related work is focused on creating one fixed efficient design with minimal accuracy loss. However, multiple models can be used to find a balance point between efficiency and accuracy. Real-time model selection is previously investigated in literature for different scenarios. In [32], the authors measure the input image's complexity before classification and select the ideal model for the specific input image content. In [22], two models are employed, one big and one little. Each image is classified by the little model first. Then, if the classification is found unsuccessful, big model classifies the same image again. However, the unsuccessful attempts in this method increase the latency, and hence, are not suitable for real-time machine vision system. In [5], a CNN based multiplexer is trained to select the optimal deep learning model for the given input image. The multiplexer considers the input image's complexity. In [7], the optimal model in a model set is selected or the current model is adapted when a class skew is detected in the input. The model set is prepared by pruning models by considering class skew. The aforementioned methods do not take the contention of the underlying computing system into account for model selection. In [42], a model switching methodology is developed to improve the performance for cloud servers which do not have strict delay constraints like embedded systems. In this work fluctuating workloads are considered as contention in the server environment.

Using early exit models can be an alternative to model switching. Defining early exit points in a neural network creates incremental sub-models where a latency-accuracy tradeoff occurs between these early exit points. In [33], this methodology is applied on multiple neural networks and latency-accuracy tradeoff is demonstrated. In [41], a methodology is proposed to convert any CNN to a multistage model. The stages of this multistage model are selected at runtime. In [35], an early exit model is designed to be used during runtime. The input and the contention are considered to select an approximate branch of the predefined early exit model. The contention is determined by matching previous inference delays of approximate branches and a look-up table that consists of benchmarks of each approximate branch. The runtime part of our work is different in two aspects. First, instead of approximate branches, we use multiple models that are specifically tuned to provide different accuracy-compute tradeoffs, *e.g.*, selected from existing resource-efficient deep learning model designs. This results in a better efficiency-accuracy tradeoff. Second, our contention measurement is embedded in regression models instead of look-up tables. Also, our delay normalization mechanism allows us to use different models' inference delay to measure contention. This can be useful when contention and therefore model selection change rapidly. In Section 5.5, we provide results demonstrating the advantages of our approach over early exit based technique [35].

Slimmable neural networks [40] are dynamic neural networks like early exit models. Slimming operation scales the model width by changing the number of channels in each layer. Therefore, slimming creates sub-models as well where a latency-accuracy tradeoff occurs between switches. In [39], slimmable networks are improved to be able to use arbitrary widths instead of predefined widths. In [38], a width search strategy is proposed for the number of channels instead of using predefined or arbitrary widths. Even though slimmable networks are not experimented with switching when contention exists, it is possible to use them in that way. Therefore, we compared our methodology with slimmable neural networks in Section 5.6.

In our previous work [17], we examined runtime model selection in the presence of contention. However, this work was assuming the contention levels are known beforehand and expecting a model set that is already tailored for the contention levels. In this work, we extended our previous work by adding an application level profiler to determine the contention levels and adding a model set pruning methodology to find optimal model set for the existing contention levels from a given large number of models.

3 OVERVIEW OF OUR APPROACH

3.1 Machine Vision Application

In this paper, we consider machine vision applications, specifically focusing on the image classification block within them. There are two main metrics that define the performance of image classification applications - inference delay and accuracy. In any machine vision system, it is desirable to minimize the inference delay and maximize the accuracy of the image classification task.

The presence of multiple, concurrently executing tasks in a computing system causes contention for the specific system resources, eventually resulting in increased delay for the completion of the tasks. These contentions and their impact are seen more frequently in resource-constrained embedded systems. For demonstration of contention and its impact, we consider sensor fusion in autonomous vehicles.

Sensor Fusion and Contention in Autonomous Vehicles: Autonomous vehicles operate in a complex environment and therefore require a high level of perception that is achieved by using multiple sensors and their fusion. Autonomous cars have three main sensors - Camera, LiDAR, and

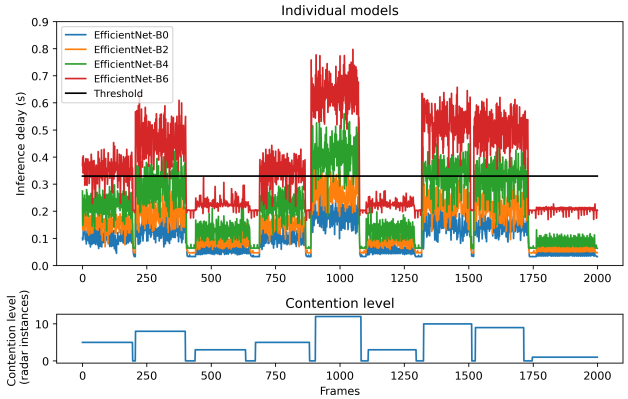

Fig. 1. Inference delays of different models under changing contention

RADAR. The fusion of these sensors can achieve better accuracy than using each sensor individually. However, this performance improvement comes with a cost since using more sensors requires more computation power. Further, the processing of each sensory modality results in contention, whose effects are especially significant in resource-constrained settings.

There are different fusion approaches as reviewed in [36]. This work categorizes fusion approaches into three levels, high-level [15], mid-level [18], and low-level [37]. All of these fusion approaches incur a processing cost in addition to sensors' individual processing costs. The computation for the fusion task is also affected by the system contention as well as contributing to it.

3.2 Delay Accuracy Trade-off

The trade-off between inference delay and accuracy is fundamental to image classification systems as more complex image classification algorithms result in higher accuracy but also require more time to compute those results. Since contention creates dynamic variations in the available compute resources, a machine vision system needs to optimize its performance in terms of delay and accuracy. Typical autonomous systems need to satisfy a delay constraint (*e.g.*, operate under a certain frame rate), while maximizing accuracy. In order to achieve this objective, we propose to use a set of N image classification models $\{M_i\}, i = 1, .., N$ with increasing complexity. Depending on the available resources in presence of contention, the system must choose the optimal model. For example, the inference delays of four different EfficientNet [31] models under varying contention are shown in Figure 1. The contention is created by multiple radar instances running on the same computing platform. The contention level graph at the bottom of Figure 1 shows varying contention levels. Each model's inference delay increases proportionally under increasing contention. Figure 1 shows that if we have an inference delay constraint, we can satisfy it by choosing an appropriate model for each contention level. For example, all of the models satisfy the delay threshold around frame 500 since the contention level is low. Therefore, the most complex model can be chosen at this contention level. However, the contention level is very high around frame 1000. EfficientNetB6 and EfficientNetB4 do not satisfy the delay threshold at this contention level. Therefore, the third most complex model, EfficientNetB2, should be chosen at this contention level. Choosing the simpler model satisfies the delay threshold, however it also results in accuracy loss.

3.3 Adaptive Model Selection At Runtime

For selection of an appropriate model, one needs to have a priori knowledge about the contention level in the system when the model will be executed, and select the best model that fits in the

available resources. Since it is not possible to know future contention levels precisely, one can estimate the contention level based on recent history, project the impact of contention on different image classification models, and select the best model for the next image frame.

The proposed framework predicts the future inference delays of the model set $\{M_i\}$ in the presence of contention. Then, we find a subset of models $\{M_j | D_j < T\}$, $j = 1, .., L$, $L \leq N$, where D_j is the predicted inference delay of model M_j, T is a latency threshold of the system and N is the total number of models in the model set during runtime. After that, we choose the appropriate model M_k such that the accuracy $A_k = \max\{A_j\}$.

3.4 Contention Grading and Defining Model Set

The adaptive model selection framework works at runtime and requires a set of models to be defined before runtime. Defining the model set is the other side of this problem and imposes an important challenge. The optimal model set differs based on the aim of the models, the system contention levels and the user requirements. There are usually a very large number of models available across the entire latency-accuracy tradeoff space. Having a large number of models to choose from can lead to the runtime framework incurring excessive overheads and/or switching models more frequently than needed. Thus, **it is important to define a model set that is minimal in size, while still offering sufficient options to adapt to the observed contention levels**. The optimality of a model set depends on satisfying the inference delay constraint while maximizing accuracy using minimum memory.

We find the optimal model set by measuring system contention and pruning a given model set based on the effects of contention in the system. In order to measure system contention, we profile the system using a specifically designed profiler. Then, we regenerate the system contention in a controlled environment to prune our model set. We have 3 pruning stages where we use independent notations in the following paragraphs. In the first stage, we remove Pareto-inferior models in the model set. Given a model set $\{M_i\}$, $i = 1, .., N$ where N is the number of models, for each model M_k, we find the subset of $\{M_i\}$ as $M_{kt} = \{M_j | D_j \leq D_k\}$ where D_j is the inference delay of model M_j. Then, let the A_{kt} be the accuracy set of the model set M_{kt}. If $max(A_{kt}) \neq A_k$, then we prune M_k from $\{M_i\}$. In the second stage, we remove models that have small gains compared to their adjacent models. These models have small gains with high cost where the cost includes inference delay and memory consumption. Given a model set $\{M_i\}$, $i = 1, .., N'$, where all models are on Pareto frontier and N' is the number of models, we find the slopes of each adjacent model pair as $S_j = \dfrac{A_{j+1} - A_j}{D_{j+1} - D_j}$ where A_j is the accuracy and D_j is the inference delay for M_j. Then given a lower (L_l) and higher (L_h) limits for the slopes, we prune the model M_j if $S_j > L_h$ or M_{j+1} if $S_j < L_l$. Whenever a pruning occurs, we recalculate all of the slopes and restart pruning. In the last stage, we remove the models that have no use in the contention levels of the system. In this step, the user requirements and the system contention levels are considered. Given a model set $\{M_i\}$, $i = 1, .., N''$ where N'' is the number of models, Contention levels $\{C_j\}$, $j = 1, .., P$ where P is the number of contention levels and the inference delay threshold T, we run the all models on each contention level to find inference delays $\{D_{ij}\}$. Then we find the most accurate model M_k that satisfies the delay threshold for each contention level j such that $A_k = max(A_i | D_{ij} < T)$. Then we add M_k to the valid model set and prune the rest of the models.

4 CONTENTION GRADING, MODEL SET GENERATION AND ADAPTIVE MODEL SELECTION: DETAILS

Our system consists of offline contention grading and model set pruning, followed by a runtime adaptive model selection. We discuss the details of each of these phases in this section.

4.1 Contention Grading

Contention grading and model set pruning comprises of three components. The first component is profiling system contention on the target system during runtime. The second component is mimicking the profiled system contention for detailed analysis of the model set. The last component is model set pruning, which outputs the optimal subset of the input model set.

4.1.1 Profiling System Contention.

Profiling the contention in a system can be a very complex task because of two main problems. The first one is - P1: the contention is created by overlapping execution of many different tasks. Therefore, the combination of these different tasks creates unpredictable contention levels. The second one is - P2: the difficulty of detecting the contention point. There are many modules (CPU, GPU, memory, bus etc.) in a computer system which can be requested by the tasks at the same time, resulting in a contention in these modules.

As a result of complex contention scenarios, instead of profiling the system contention, we decided to profile the impact of the contention to our application. In order to do that, we run a sample of our application along with all other tasks and measure the performance of our application. For example, since our application is a neural network based one, we run a sample neural network on the system and measure its inference delays to understand different levels of contention that are important to neural network based applications. We named this profiler as Contention Impact Profiler (CIP). CIP should be run on the system for long enough to observe all contention levels.

Using CIP solves the first problem (P1) because we observe the effect of contention and see the root of contention as a black box which can be a single task or multiple of them. This method also ignores the contention that has no effect on our application and therefore simplifies the contention profiling. Using CIP solves the second problem (P2) of contention profiling since it does not try to identify the contention point.

The CIP is needed for our framework for two aspects. The first one is the need of knowing the specific contention levels in a system and regenerating them in a controlled offline environment. These known and controlled contention levels are required and used in our model set pruning methodology. If we do not have these known and controlled contention levels, we cannot prune a model set for the target system with contention. The second one is the requirement for the generalization. The model set pruning is designed to work for any system and requirement. However, it requires system and contention specific information for each case. Measuring such information for each system would require a significant amount of effort and would decrease the value of our framework. The CIP covers this aspect by working as a black box in any system and collecting the required system and contention specific information by model set pruning stage.

4.1.2 Mimicking Contention.

Once we know the levels of contention, we mimic this contention in a controlled environment for the purpose of selecting a model set. We propose the use of Artificial Contention Units (ACUs) for this purpose. An ACU is a dummy workload used for the purpose of producing a specific level of contention. Different numbers of ACUs are used to generate different levels of contention. An ACU is composed of dummy instructions including vector addition, vector multiplication and FFT operations. These operations are big and diverse enough to create contention and small enough to give us a fine grained control over contention creation when multiple of them are used.

We use the same CIP that is used to profile the system contention, to profile contention synthetically created by ACUs. During this profiling, we increase the number of ACUs incrementally and measure the inference delays of the CIP. This creates an inference delay trace of the CIP

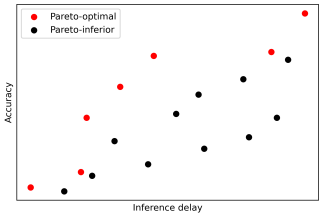

Fig. 2. Pareto-optimal and Pareto-inferior of a hypothetical model set

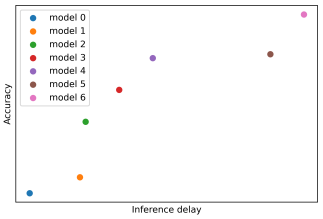

Fig. 3. Hypothetical model set before transition pruning

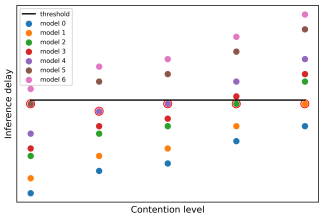

Fig. 4. Hypothetical model set before contention pruning

under increasing ACU contention in addition to the inference delay trace of the CIP under system contention as we obtained in 4.1.1. Note that both traces are measured on the same hardware by the same CIP.

At this point, we get back to the inference delay trace of the CIP under system contention and do some preprocessing. First, we take the moving average of the trace to remove noise. Then, we apply kernel density estimation to data to find the contention levels. In the end, we have the number and intensity of the contention levels in the system. At the last step, we match the system contention levels and ACU contention levels with same intensity. As a result, we have the set of ACU contention levels that can regenerate the system contention. Since we have a complete control over using ACUs, we systematically use them to measure the performance of all of our models and prune our model set.

4.1.3 Model Set Pruning.

The aim of contention grading is to prune the model set and propose an optimal subset. Profiling system contention and mimicking it are done to enable model set pruning. Subsequently, we prune the model set in three stages, which are described below.

Pareto Pruning

Pareto pruning is the first pruning stage. In this stage, we remove the models that are not on the accuracy-inference delay Pareto frontier of the model set. This is because the models that are not on the Pareto frontier should not be used in any application. If a model is not on the Pareto frontier, there is at least one other model in the model set that performs better with less cost. So, the models that are not on the Pareto frontier are either obsolete or poorly designed for the image classification task at hand. Pareto-optimal and Pareto-inferior models of a hypothetical model set are shown in Figure 2.

To construct the accuracy-inference delay, we use the average inference delay over all the contention levels of the system in this stage. This enables us to consider the overall response of models to system contention. If a model is on the Pareto frontier without contention, but the inference delay of the model increases more than other models in the presence of contention, then this model may lose its position on the Pareto frontier.

At the end of this stage, we have a model subset where each model presents a unique tradeoff between accuracy and inference delay.

Transition Pruning

Pareto pruning creates a model subset where each model shows a non-zero improvement in one metric with respect to models adjacent to it in the Pareto frontier. However, some of those models might not be beneficial in practice. This is because Pareto optimality considers any gain without

evaluating the actual magnitude of the gain. There can be a very small gain with very high cost or a very high gain with very small cost, leading to models that do not get used in practice.

In our case, the gain is accuracy and the cost is inference delay or memory consumption. We use inference delay as our cost for our pruning calculations. However, since the inference delay and memory consumption are usually correlated (since both depend on the number of parameters in the model), our pruning in this stage improves memory consumption as well.

A hypothetical model set is shown in Figure 3. All models are Pareto optimal in this model set. However, two minimally useful transitions can be noticed. These transitions are - model 1 to model 2, and model 4 to model 5. First, let's consider the transition model 1 to model 2. There is a very high accuracy improvement from model 1 to model 2. However, their inference delays are almost the same. Therefore, the model 1 can be dropped from the model set. The second transition has a similar problem but in the opposite direction. There is a very small accuracy improvement from model 4 to model 5. However, model 5 requires significantly larger time to achieve this accuracy. Therefore, model 5 is not useful in this model set. Removing these models from the model set does not only simplify the model set but also improves the performance of adaptive model selection by eliminating the use of these models, and hence the associated overheads, at runtime.

When the given example is examined, a certain pattern can be noticed when there is an inefficient transition. This pattern is the slope of the transition. If the slope is too low or too high, it is an inefficient transition. We therefore use the slope to identify inefficient transitions and eliminate them. The lower and higher threshold of the slope can change depending on the problem and user requirements. Therefore, these values are hyperparameters in our methodology.

Contention Pruning

In this final stage of pruning, the model set is pruned considering specific contention levels and user requirements. Our pre-deployment profiler (CIP) gives the specific contention levels that can be observed in the system. The number of different contention levels is important because it also limits the number of models in the model set. Given a contention level, there can be only one best model, because only one model has the maximum accuracy among the models that satisfy the inference delay threshold (user requirement) at a specific contention level. Note that the vice versa is not true - one model can be the best model for multiple contention levels. As a result, we can say that **the total number of models must be less than or equal to the total number of contention levels** and we consider this rule in the contention pruning stage.

Before explaining this stage of pruning, note that Pareto pruning creates a model set where accuracies and inference costs vary monotonically. Therefore, if a model has a higher inference delay than another model, it is also more accurate than the other model. This property is used to define the more accurate model by looking at the inference delay in contention pruning stage.

Before contention pruning, we need to run all of our models under the contention levels that we found in the previous steps. We use ACUs to mimic the system contention, run each model under each contention level and save the average inference delays. The output of this part can be observed in Figure 4 where a hypothetical model set is used for explanation purposes. In the figure, there is an inference delay threshold which is shown as a black line. This is the user requirement which basically defines the maximum acceptable inference delay. Therefore, we want our models to perform under this threshold while being as much as accurate. The x-axis of the figure shows the contention level. The increasing contention levels and the models' responses are shown from left to right. In each contention level, we find the model that is just below the inference delay threshold and mark it as valid model. In the end, any model that is not in the valid list is pruned. The valid model in each contention level is shown with a red circle around it.

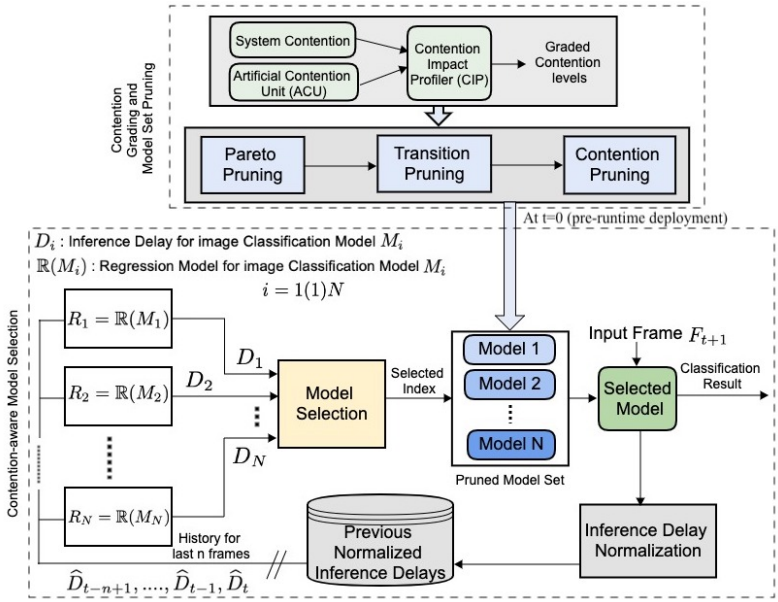

Fig. 5. Overview of the proposed framework

When we examine the pruned models, we can see that they are not suitable for the user requirement and contention profile of the system. For example, model 6 requires too much time and is not suitable even in smallest contention level. The model 0, on the other hand, is always below the inference delay threshold. However, the contention never increases up to a point where using model 0 is the optimal choice. Another pruned model is the model 3. Model 3 is below the threshold in some contention levels and above it in some other contention levels. So, it is expected that model 3 should be optimal at one contention level. However, as we can see from this example, the contention does not have to increase gradually at every level. A system may experience a jump in contention which eliminates the need for middle level models. We also notice that model 4 is the optimal choice for two contention levels. This can happen when some contention levels are close to each other.

In the end, 3 models are pruned in our hypothetical example. This leaves our model set with 4 models (model 1, 2, 4, and 5) which is smaller than the number of measured contention levels (5).

Figure 5 shows our proposed end-to-end framework, the top part of which shows the components involved in contention grading and model-set pruning. As the figure indicates, this phase is done before the runtime model selection starts at $t = 0$ when the pruned model set is forwarded to the predictive model selection framework.

4.2 Runtime Model Selection

The overview of the proposed predictive model selection framework is shown in the lower part of the Figure 5. The framework employs a set of image classification models provided by the contention grading and model set pruning component. The framework chooses the optimal model for the next frame's classification while considering the current contention on the system. The optimal model is determined by using historical information and a set of linear regression models. The historical information comes from the previous frames' normalized inference delays. There is one regression model for each image classification model used in the framework. The regression models are trained before runtime using their corresponding image classification models on randomly changing

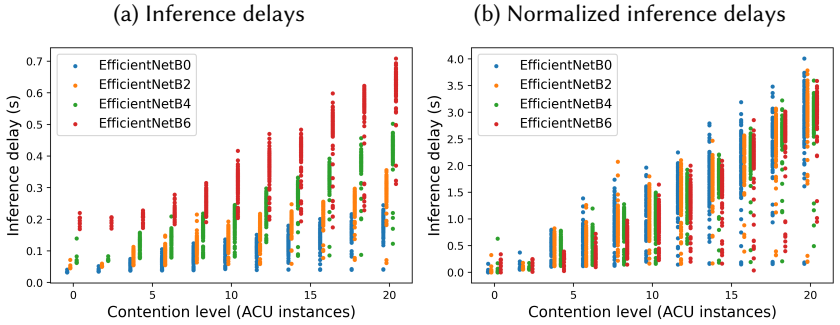

Fig. 6. Inference delays under increasing contention and normalization

contention level. All regression models take the same input, the previous normalized inference delays, and output the predicted inference delay for their corresponding image classification model. Then, the framework chooses the most appropriate model based on the delay threshold constraint and maximum accuracy as mentioned earlier.

Delay Normalization

Figure 6a shows the inference delays for EfficientNet-B0, B2, B4 and B6 [31] under increasing contention. It shows that the inference delay values depend on two things - the system level contention, and the image classification model type. Since our framework uses historical inference delay values to represent the impact of contention, we remove the model type dependency by normalization shown in Equation 1. In this equation, x is one inference delay of a model and X is a vector that consists of all inference delays of the same model. If we consider EfficientNetB6 in Figure 6a, x is one red dot and X is the vector of all red dots.

$$x_{normalized} = \frac{x - min(X)}{sqrt(var(X))} \tag{1}$$

The result of normalization is shown in Figure 6b. The minimum and variance values for each model is saved before runtime and used to normalize the inference delays of the models during runtime.

Prediction and Selection

The training data is created by running each model under randomly changing contention levels. As input, the normalized data is split into chunks of n consecutive normalized inference delays. All of the regression models take the same input as they will be predicting in parallel using the same input. As prediction output, non-normalized delays are used. Each regression model has different output corresponding to its image classification model. Hence, each regression model takes same input, n previous normalized inference delays, and predicts its corresponding image classification model's inference time for the next frame. After this step, the framework has a predicted inference delay for each image classification model. The image classification models are already ranked in terms of accuracy on static datasets before runtime. Therefore, the framework chooses the model which has the highest rank and also a predicted inference delay under the predefined threshold.

5 EXPERIMENTAL RESULTS

5.1 Experimental Setup

We implemented the proposed framework on the Nvidia Jetson TX2 platform. We used Tensorflow to train deep learning models that are used in Section 5.5. We used built-in image classification models of Tensorflow for the rest of the experiments. These built-in models are EfficientNets [31], ResNetV2 [10], InceptionV3 [29], DenseNets [12], MobileNetV1 [11], MobileNetV2 [27], and

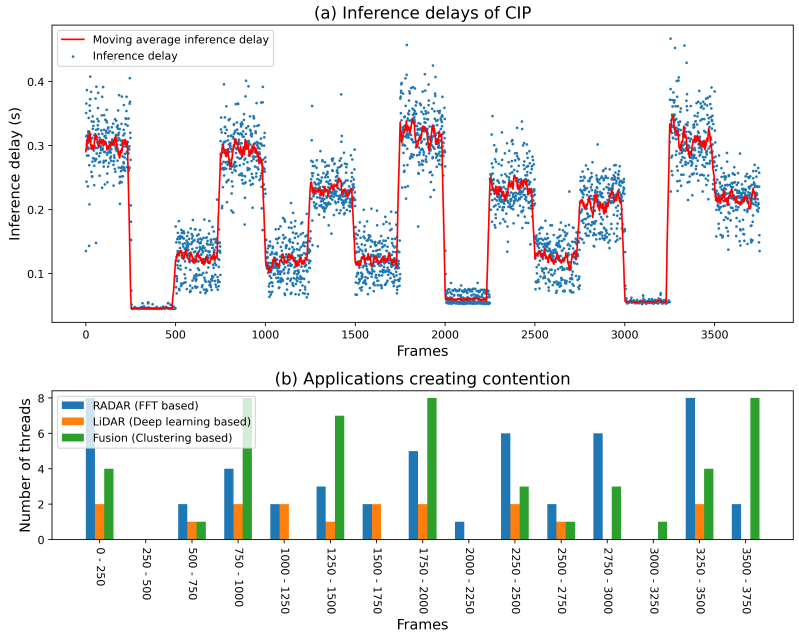

Fig. 7. Inference delays of CIP under system contention

NasNets [45]. These built-in models come with pre-trained weights on the ImageNet dataset [26]. Since the validation set of ImageNet is available for hyper-parameter tuning, we decided to use the ImageNetV2 dataset [24] for our test set. Therefore, all of the reported test results in this section are using ImageNetV2 dataset. Also, we resized images using the bi-linear method without cropping before inference for all of the models. We standardized the test set and resizing-cropping technique to make a fair comparison. Therefore, the reported accuracies may be different from the original ImageNet validation set accuracies that are reported in the original papers of the models. Since we focus on the relative accuracies, this is not an issue for our experiments. When we trained custom models, we used the original ImageNet dataset.

5.2 Contention Grading and Model Set Pruning Evaluation

We consider contention scenarios imposed by multiple autonomous vehicle applications including RADAR processing (FFT based), LiDAR processing (Deep neural network based) and sensor fusion (clustering based). We run the deep neural network based image classification application concurrently with these other applications on the target system. These coexisting applications create different contention patterns in the system, e.g., the radar processes and sensor fusion run on the CPU while the LiDAR processes run on the GPU, thus creating contentions with the parts of the image classification algorithm sharing those system resources.

5.2.1 Contention Grading Evaluation.

System Profiling with CIP:
We generate a random number of threads for each of our contending applications. Then we use the CIP to profile this system. In the core of the CIP, we used an application based on EfficientNetB2 image classification model and measured its inference delays to profile the impact of the contention on the system. The inference delays of the CIP and the number of threads of applications creating contention are presented in Figure 7. Note that, the information about applications that create contention is not used by the CIP. We only provide the number of threads of contention applications

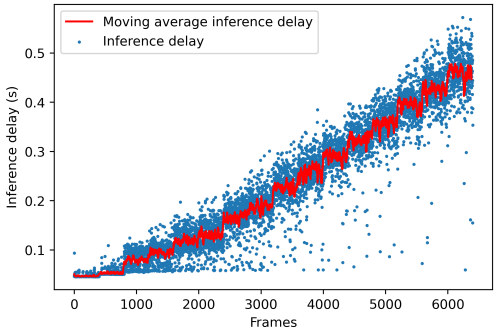

Fig. 8. Inference delays of CIP under increasing ACU contention

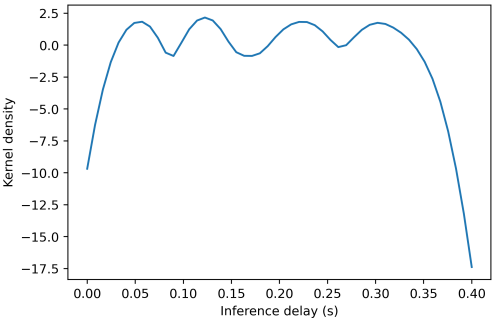

Fig. 9. Kernel density estimation of the system profile

for better understanding of CIP behavior. We generated 15 contention combinations by changing the number of threads of each contention application. 12 of them are unique combinations. In any system, same combination of threads can repeat over time or different combinations of threads can result in the same contention level. We can see examples of both of these scenarios in our system profile.

Mimicking Contention with ACU:

The same EfficientNetB2 based CIP is used to profile the incrementally increasing ACU contention. We increased the number of ACU threads by 2 threads at every 400 frames. The inference delays of the CIP are shown in Figure 8. This fine grained contention steps will be used to regenerate the system contention at model pruning step. However, in order to do that, we need to know the system contention levels that match to specific steps of ACU contention.

We use kernel density estimation with Gaussian kernel to find the contention levels in the system. We selected the Gaussian kernel because the distribution of inference delays show a similar pattern to Gaussian distribution at every specific contention level. When we apply kernel density estimation on the inference delay axis, we remove the position information of the inference delay samples. Therefore, we automatically combine repeating or similar contention levels in time, which can be caused by repeating same contention combination or completely different combinations with same effects. The kernel density estimation is shown in Figure 9. The peaks of this plot are the means of the estimated Gaussian kernels. Therefore, the peaks are the specific contention levels that exist in our system contention. The inference delay values of these peaks are matched to inference delay values at ACU profile to find required the number of ACU threads to regenerate each contention level. In this specific example, the number of ACU threads are found to be 2, 8, 16 and 20.

5.2.2 Model Set Pruning Evaluation.

Once we have the required number of ACUs to regenerate the system contention, we benchmark our input set of models under the artificial contention that is generated by ACUs. After this benchmarking step, the accuracy and average inference delay across all contention levels of each model is calculated. These values for our specific example are shown in Figure 10a. The abbreviations in the legend of the figure and their corresponding models are as following: eb0 to eb7 are for EfficientNet models, r50V2 and r101v2 are for ResNetV2 models, iv3 is for InceptionNetV3, d121 to d201 are for DenseNet models, mnet is for MobileNet, mnet2 is for MobileNetV2, nasm is for NasNetMobile, and nasl is for NasNetLarge. This plot clearly shows that some models have no advantage at all compared to others. For example whole families of ResNetv2 and DenseNet architectures are performing with less accuracy using more inference time compared to other

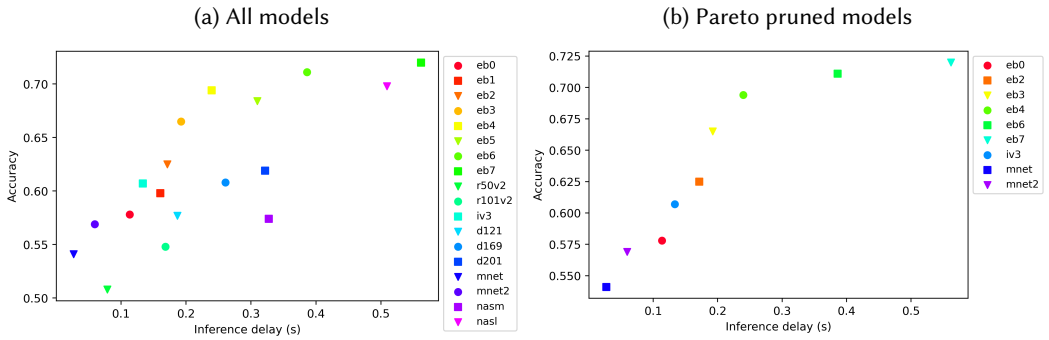

Fig. 10. Pareto pruning - Accuracy vs inference delay values of models

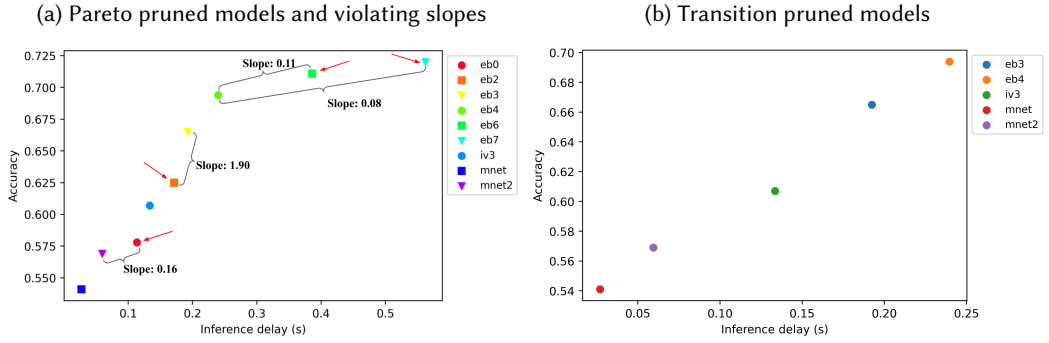

Fig. 11. Transition pruning - Accuracy vs inference delay values of models

models. Therefore, we calculate the accuracy-inference delay Pareto frontier of the models to remove these bad performing models from consideration for our task. The Pareto pruned model set is shown in Figure 10b.

Each model in Pareto pruned model set is guaranteed to give best accuracy at its inference delay or below. However, this theoretical result does not correspond to the equally good practical result when these models are used in an application on an embedded system. A model can still be on the Pareto frontier if it improves accuracy very slightly but requires a lot more time and memory for inference. Using such models harms the performance of our application as they do not provide significant advantage while still requiring the cost. Therefore, we remove these models from our model set as well. In order to remove these models, we check the slope of every consecutive models. If the slope is too big, we remove the model with smaller accuracy. If the slope is too small, we remove the model with higher accuracy. In our specific example, we define the slope thresholds as 0.25 and 1.5. The slopes that violate these thresholds are shown in Figure 11a. The pruned models are pointed by a red arrow. The resulting model set is shown in Figure 11b.

We can consider some of the prunings to understand how this stage can save memory. For example, eb4 (EfficientNetB4) requires 79.1 MB while eb6 (EfficientNetB6) requires 174.8 MB. Even if eb6 would satisfy the user requirements, it would require more than 2 times larger memory than eb4 without providing significant advantage.

In the final step of pruning, the contention levels and the user requirement are considered. The user requirement is the inference delay threshold. The application's inference delay for one frame should not exceed this inference delay threshold in any contention levels. The remaining models' inference delays are considered under the existing contention levels as in Figure 12. In this figure, every model is run for 250 frames for each of the contention levels. Note that, the previous steps of

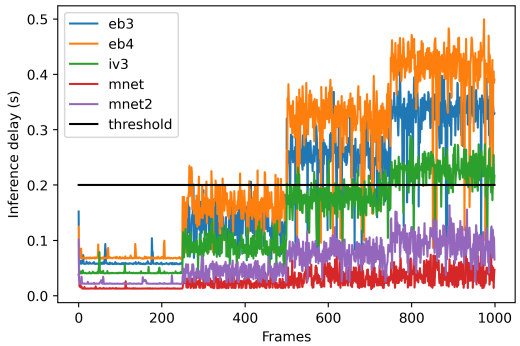

Fig. 12. Inference delays of the transition pruned models under existing contention levels

pruning guaranteed that a model with higher inference delay has also better accuracy with a decent margin. In this step, we select the best performing model that satisfies the inference delay threshold in each contention level. Note that, we do not need to select one unique model for each contention level. In first two contention levels, eb4 (EfficientNetB4) is the optimal model. In the third level, iv3 (InceptionNetv3) is the optimal model. eb3 (EfficientNetB3) satisfies the delay threshold at contention level 2 but violates it at contention level 3. Since it is not selected at contention level 2 and there is no intermediate contention level between levels 2 and 3, eb3 is pruned in this step. Similarly, mnet (MobileNetv1) satisfies the inference delay threshold at all of the contention levels. However, there is always at least one model that satisfies the delay threshold and performs better than mnet. Therefore, mnet is also pruned from the model set. In the contention level 4, the optimal model is mnet2 (MobileNetv2). As a final result, the optimal model set for this contention scenario consists of eb4 (EfficientNetB4), iv3 (InceptionNetv3) and mnet2 (MobileNetv2). Our contention grading and model pruning methods decreased the number of model from 18 to 3 for a contention scenario where 12 unique combinations of 3 real applications are running on the system.

5.3 Runtime Performance Evaluation

After we obtain a pruned model set, we run our contention-aware adaptive model selection framework. First we show the performance of our predictive model selection method and then also show how the prior contention grading stage positively contributed to the model selection performance.

5.3.1 Predictive Model Selection Performance.

We compare our predictive model selection with two reactive model selection approaches. The first one is called 1-step reactive model selection which checks the last frame's inference and then selects 1-step stronger model if the last frame's inference is below threshold. Otherwise, it selects the next (1-step) weaker model. The second reactive approach is called N-step reactive model selection. This approach similarly checks the last frame's inference delay and selects 1-step stronger model if the last frame's inference is below the threshold. However, if the last frame's inference is above the threshold, it conservatively selects the weakest model for the next frame to satisfy the delay threshold.

The temporal plots for 6000 frames under a specific contention regime are shown in Figure 13. This contention regime is generated by randomly sampling real system contention applications that are shown in Figure 7. The inference delay and the selected model's index are given for three different model selection methods. The individual models' inference delays are also plotted for comparison purposes. The inference delays are averaged over 10 frames to smooth the plots. The delay plots for individual models in Figure 13a show that using a single model under varying contention is not optimal. The individual plots also suggest the best model under a specific contention regime, e.g.,

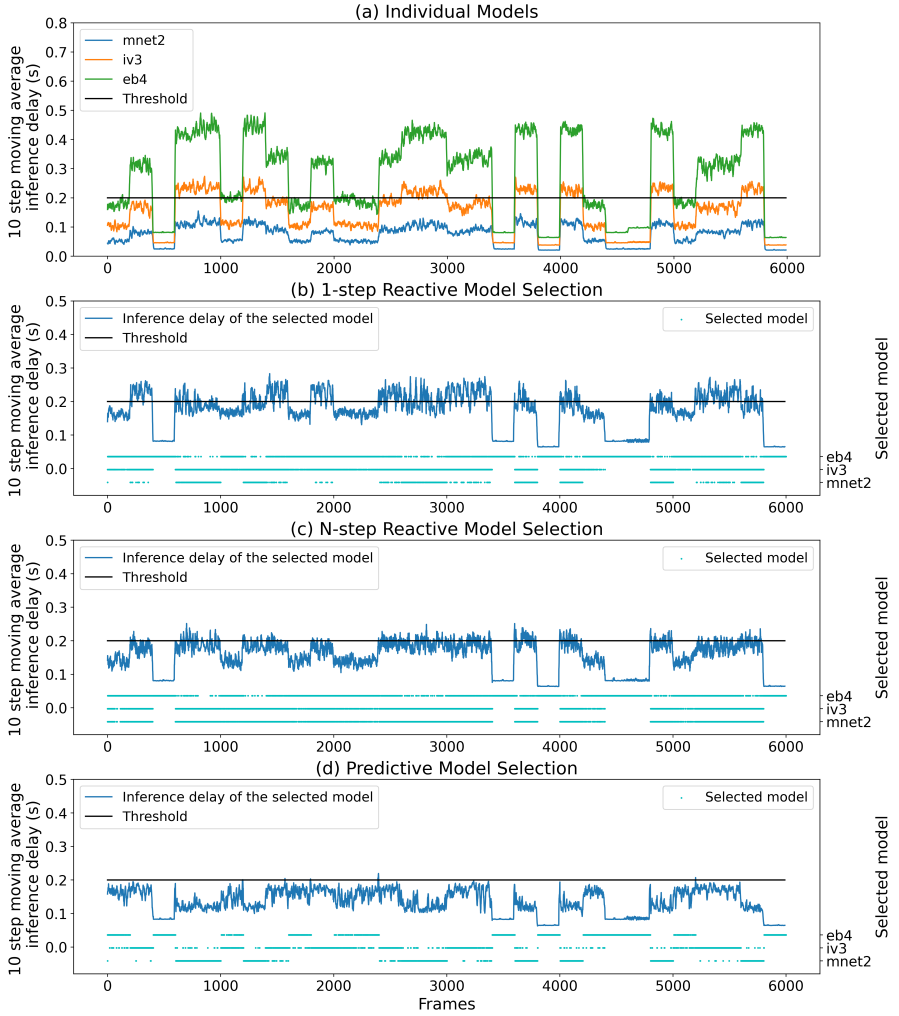

Fig. 13. Temporal comparison of individual models, reactive methods and the predictive method under varying contention

around the frames 900, 1900, 3900, the ideal models are mnet2, iv3, eb4 respectively. It can be seen that the predictive method can successfully select the optimal model most of the time in Figure 13d. It can also choose multiple models under the same contention region. One example of this can be seen just after frame 2000. In this region, predictive model selection selects eb4 and iv3 frequently. This happens when contention corresponds to the middle of two models, i.e contention is high for iv3 and is low for eb4. In this case, predictive model selection changes the optimal model selection between eb4 and iv3 frequently to satisfy inference delay threshold and maximize the accuracy. 1-step reactive model selection and N-step reactive model selection frequently fail to satisfy delay threshold as shown in Figure 13b and Figure 13c, respectively.

Table 1 shows the summary of data for Figure 13 in terms of average performance of different schemes. If a frame classification takes more time than predefined threshold, we consider it as delay violation. The delay violation is a way to measure wrong selections. The table shows that all of the model selection methods have an accuracy around the middle of individual models. However, the reactive methods have large delay violations as well. On the other hand, the predictive method has only 11.66% delay violation.

Model	Accuracy (%)	Delay Violations (%)
MobileNetV2 (mnet2)	57.50	0.05
InceptionNetV3 (iv3)	63.93	29.75
EfficientNet-B4 (eb4)	70.00	65.50
Average-(mnet2, iv3, eb4)	63.81	31.76
1-step Reactive Model Selection	65.93	34.61
N-step Reactive Model Selection	64.86	27.40
Predictive Model Selection	64.60	11.66

Table 1. Comparison of methods in a specific contention regime - Dataset: ImageNetV2

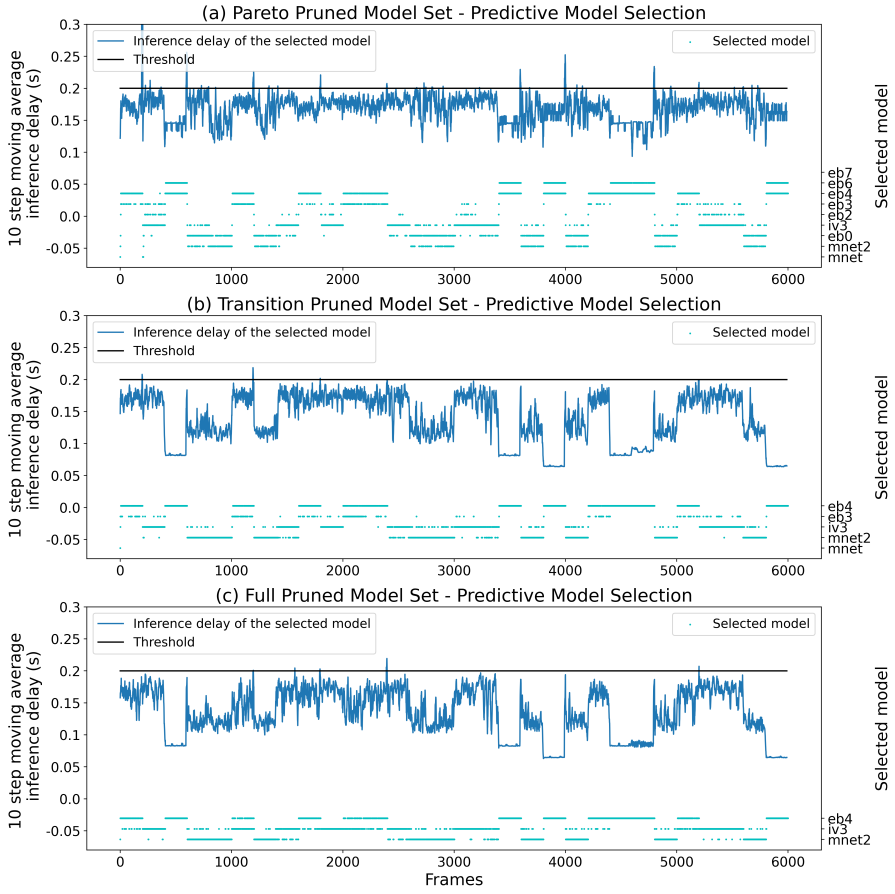

Fig. 14. Temporal comparison of predictive method with model sets after each pruning stage under varying contention

5.3.2 *The Effect of Contention Grading on Runtime.*

In this section, we consider the effect of contention grading on runtime with respect to accuracy and delay violation. There are 3 stages of pruning which are Pareto pruning, transition pruning and contention pruning. These are applied one after another in this order. Therefore, we will compare 3 model sets on runtime using the same predictive model selection method. The Pareto pruned model set has 9 models which are shown in Figure 10b, the transition pruned model set has 5

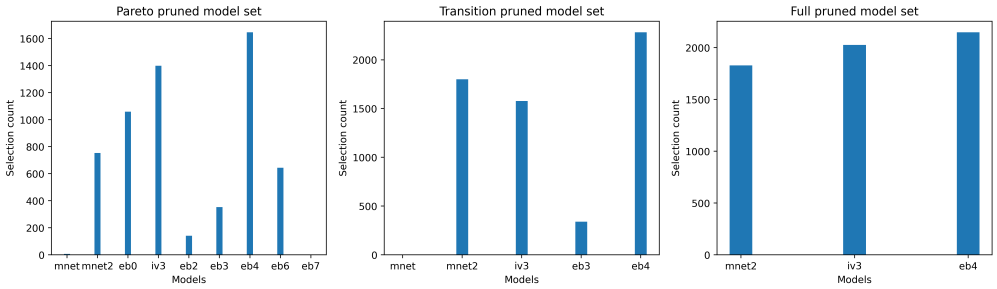

Fig. 15. Selection counts of models for each model set

Model	Accuracy (%)	Delay Violations (%)	Memory Consumption (MB)
Pareto-pruned	65.91	31.35	4290
Transition-pruned	64.91	12.46	3650
Final	64.60	11.66	3460

Table 2. Model set comparison - Dataset: ImageNetV2

models which are shown in Figure 11b, full pruned model set has 3 models which are MobileNetV2, InceptionNetV3, and EfficientNetB4.

The temporal comparison of three model sets using the predictive method is shown in Figure 14. When the number of models increase in a model set, the number of model switching also increases which harms the performance since there is only one optimal model in one contention level. When we examined the selected indexes of Pareto pruned and transition pruned model sets, we see that mnet and eb7 are almost never selected. Therefore, they occupy memory without providing any gain to system. We plotted the selection counts of models for each model set in Figure 15. This plot shows the most frequently used models in each model set. The most frequently used models are similar in most cases (mnet2, iv3, eb4). The only exception is eb0 in Pareto pruned model set where it is used more than mnet2. eb0 is pruned in transition pruning since it requires more than 1.5x memory of mnet2 while it does not give significant accuracy gain over mnet2. Our model set pruning stage finds these frequently selected models (mnet2, iv3, eb4) before runtime.

We examined the overall performance of these model sets in Table 2. The Pareto-pruned model set has a large delay violation percentage at 31.35%. Transition pruning achieves a decrease in delay violation significantly from 31.35% to 12.46% with 1.0% absolute accuracy loss. The final pruning achieves the smallest delay violation percentage at 11.66% with another 0.31% absolute accuracy loss. Furthermore, the final pruning achieves the smallest memory consumption. The provided memory measurements include base Tensorflow cost which is around 3GB. This is a one time cost and independent from the number of loaded models. Therefore, another comparison can be made without including this base cost. Then memory consumption values are 1240MB, 600MB, 410MB for Pareto-pruned, transition-pruned and final model sets, respectively. Considering these results, the final pruning achieves the best memory efficiency by occupying 0.33 of what Pareto pruned model set occupies and 0.68 of transition pruned model set occupies.

5.4 System Implementation Details

Our framework works with neural network based applications. Therefore, we decided to use Jetson TX2 which has a GPU for neural network loads. EfficientNetB0, a neural network that is extensively used in our experiments, runs in 34.9 ms on Jetson TX2 GPU and in 61.1 ms on Jetson TX2 CPU. Therefore, Jetson TX2 GPU gives a speedup of 1.75 over a mobile CPU. Moreover, our framework

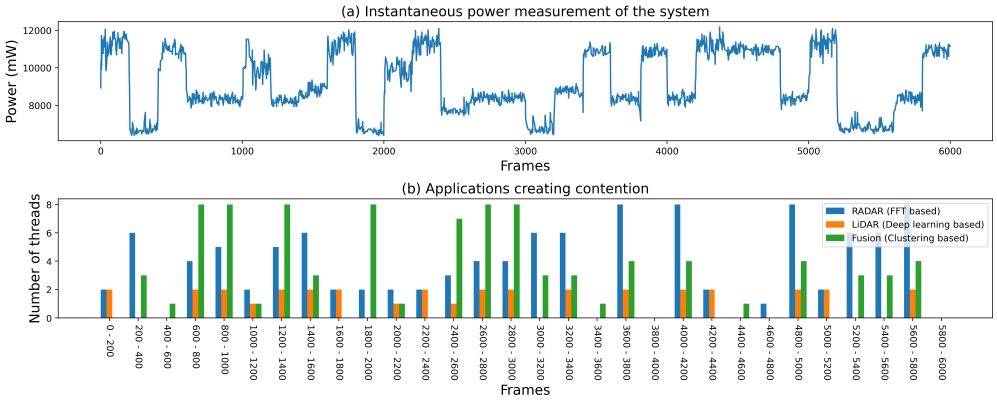

Fig. 16. System power measurement when model selection is running under varying contention

is designed for real time applications and Jetson TX2 is a good fit since it is an embedded system. Lastly, we are using Tensorflow for neural network applications and Jetson TX2 is running Linux with Tensorflow support.

The power consumption corresponding to Figure 14c is shown in Figure 16a. The details of applications that create contention are also given in Figure 16b. The RADAR and Fusion applications run on CPU, the LiDAR application runs on GPU. The power is measured by the built-in power sensor of Jetson TX2 which provides the average of the last 512 samples from continuously probed data when it is called. The peak power is 12170.0 mW.

The inference delays and the power consumption are not changing in parallel, which would be the expected behavior in single threaded applications. However, since our system is multi-threaded and uses both CPU and GPU, the behavior is different. This is because the power usages of GPU and CPU are different. The behavior can be understood by comparing the first three regions. In region 400-600, the contention is very small and limited to CPU and therefore the inference delay of our application is small. However, power consumption is high. This is because the GPU is used extensively all the time. On the other hand, in the region 200-400, the contention is high and heavily focused on CPU. Therefore, the inference delay of our application is high which is compensated by choosing a smaller model. This is because the CPU is being used heavily and becomes a bottleneck in the system. However, since the CPU does not consume as much power as the GPU, the power consumption (average of 512 samples from Jetson TX2 sensor) is low. The contention of region 0-200 is similar to region 400-600. However, there is one more GPU application in region 0-200. Therefore, the power consumption is bigger.

We also measured temperature values of GPU, CPU and the board. Once the system is used for a while and stabilizes, the temperature does not change much. GPU and CPU temperature stay between 38C and 40C, and the board temperature stays around 35C. Jetson TX2 has a fan and therefore active cooling results in stable temperature values for our workload.

The average running time cost for one frame of our predictive framework is 0.29 *ms* which is approximately 690 times smaller than average inference delay for one frame. Therefore, we can say that the time cost of the framework is insignificant. This only includes the selection logic which is a relatively light calculation. A matrix multiplication is used for linear regression models and an iteration is used for model selection. Overall, model selection takes too little time to trigger any measurement hardware and we do not see any unusual pattern in general power and temperature measurements. Therefore, the power consumption and temperature overhead of model selection is negligible.

Model M	Average of first inference delays right after switch to Model M (s)	Average of all inference delays when the Model M is used (s)	Difference in percentage
EfficientNetB0	0.03431	0.03324	3.21%
EfficientNetB2	0.04663	0.04683	-0.41%
EfficientNetB3	0.05318	0.05261	1.09%
EfficientNetB4	0.06495	0.06426	1.07%

Table 3. Inference delays of 4 models when model switching is used

All of our models are stored in RAM during runtime. To measure switching overhead, we loaded 4 models in RAM, ran 1 model for 100 frames, then switched to another model and kept this cycle for 10000 frames. The Table 3 shows the difference between first inference delays right after switch and mean of all inference delays for each model. The table shows that there is no significant difference between the first inference delay and the rest. Sometimes, the average of first inference delay is even faster than the average of the rest as in the case of EfficientNetB2. Note that these values are average. Therefore, in many switch cases the other models also are faster in their first inference delay compared to the rest. As a result, we can say that we do not observe any perceivable switching overhead.

5.5 Comparison with Early Exit Based Method

Early exit networks present an alternative approach to adapting neural network based applications to contention [35]. An early exit network consists of different exit points that are typically derived by adding classifier layers to different intermediate features in a neural network to generate the final class predictions. Different early exit branches are selected as a response to changing contention levels. Since this work is closely related to the runtime part of our work, we compare the contention-aware early exit methodology with our work in this section.

We designed an early exit model to adapt to changing contention levels based on the methodology presented in [35]. We used EfficientNetB0 architecture as the backbone of our early exit model. All of the EfficientNet architectures consist of 7 blocks. These blocks are scaled in terms of width and depth to create heavier models, while the number of blocks stays constant throughout all EfficientNet architectures. Therefore, we decided to use these blocks as our early exit paths. We created 5 early exit branches from the output of block 3 to the output of 7. For each exit, we built a classifier top that is similar to the original EfficientNet top. This classifier top includes a 1x1 convolution layer to set channel sizes of the features to some constant value (1280), a global average pooling to remove spatial size dependency and a fully connected layer to generate predictions. This top design makes the early exit branches input size agnostic. Moreover, we created 4 different input sizes as (128x128, 160x160, 192x192, 224x224) by following a similar practice to [35]. In the end, our early exit model supports 20 different combinations of early exit branches and input sizes.

We also designed 4 individual models to compare with the early exit model. Since the early exit model is trained from scratch, we also trained individual models from scratch under the same conditions to make fair accuracy comparisons. Therefore, we did not use pre-trained models as in the previous section. We used the same intermediate points as the early exit branches to design individual models. For example, the smallest individual model starts as an EfficientNetB0 model but stops at block 4 and ends with a classifier top. Similarly, the other models stop at blocks 5,6, and 7. As a result, our individual models are directly comparable with the corresponding early exit branches in terms of architecture.

The training is done on the ImageNet training dataset. The ImageNet validation dataset is used for monitoring improvement and early stopping. The ImageNetV2 test set is used for reporting

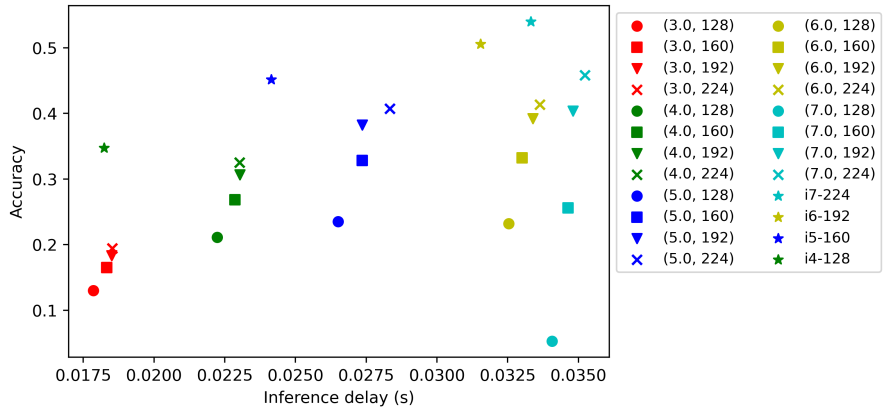

Fig. 17. Performance comparison of early exit branches -(branch no, input size) and corresponding individual models - i(model size)-input size

the test accuracies. The Adam optimizer [16] is used to train the parameters. Random cropping, random horizontal flipping and random contrast (factor 0.8-1.2) are used as data augmentation techniques.

5.5.1 *Multi Objective Optimization and Impact on Accuracy.*

The training of early exit model is a multi objective optimization. During forward propagation, the same data is fed to the network and each early exit branch makes a prediction. An error is calculated at each early exit branch. Hence, during back propagation, multiple gradients are propagated backwards. This results in multiple objective optimization of the shared parameters. For example, a convolution layer in block 3 needs to learn both low level features for early exit branch 7 and high level features for early exit branch 3. This results in longer training times and also inferior accuracy.

In [33], it is shown that the early exit method can have regularization effect since it makes it harder to train the neural network. However, this effect is only applicable when the data is too small or the network is too high capacity for the data. Also, there are other regularization techniques that are widely adopted in the neural network design such as dropout [28] or data augmentation [4].

The inference delays and accuracies of early exit branches and corresponding individual models are shown in Figure 17. The early exit branches in the legend are indicated as (branch number, image size). The individual models in the legend are indicated as i(model size)-image size. Note that branch number and model size are directly comparable as explained previously. This similarity is shown with colors in the plot. The individual models outperform the early exit branches in terms of inference delay and accuracy. Moreover, the early exit model has a time overhead due to control logic. For example, early exit branch (7.0, 224) and individual model i7-224 are completely same in terms of architecture and input size. However, individual model runs slightly faster than the early exit branch. The same difference can be observed for the other individual models and their corresponding early exit branches.

5.5.2 *Runtime Performance Comparison.*

We tested the early exit model and the individual model set on runtime with the contention profile in previous section. We used our regression model based selection methodology for early exit runtime branch selection and individual model set runtime model selection. The temporal comparison and the numerical results for the early exit, individual model set and the smallest

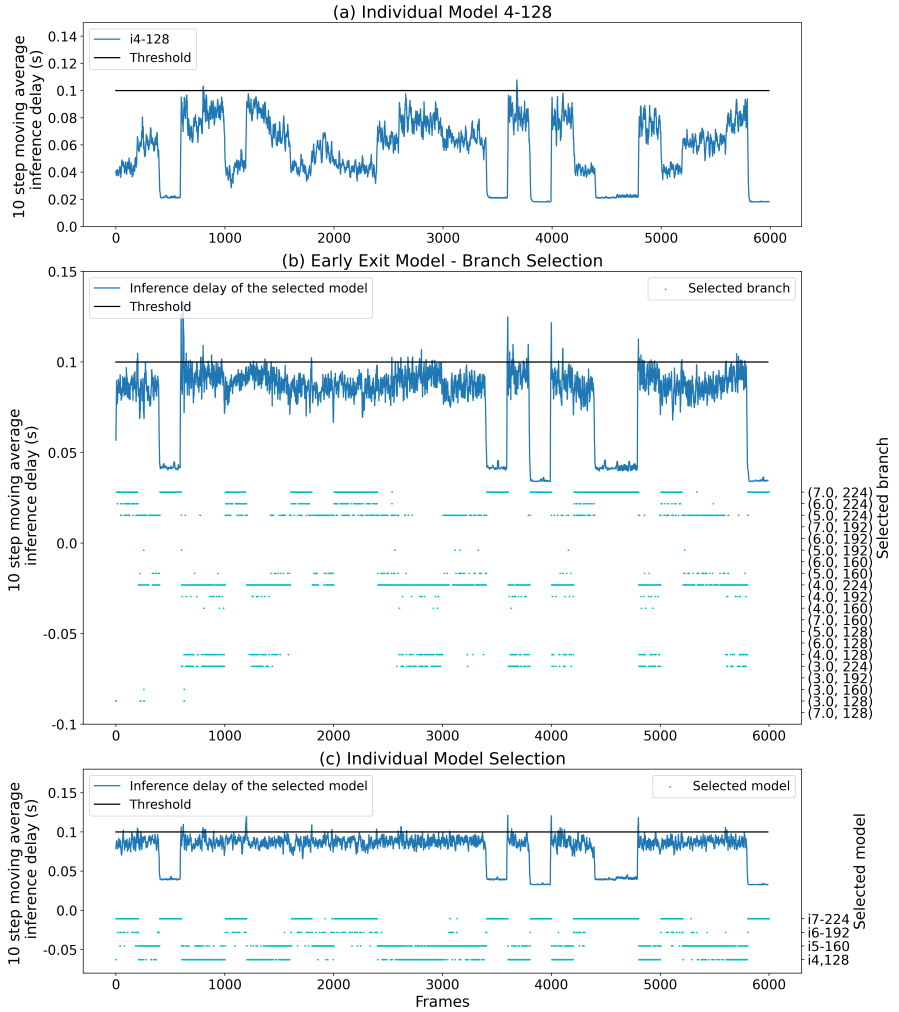

Fig. 18. Temporal comparison of the smallest individual model, early exit, individual model selection under varying contention

individual model are shown in Figure 18 and Table 4. The individual model set has almost absolute 9% higher accuracy than the early exit model. Moreover, the delay violation of the individual model set is slightly less than the early exit model.

There are 20 early exit and input size combinations in the early exit model. However, this granularity is not used completely even though a large variety of contention levels (12 unique contention combinations as in Figure 7) are experienced. This is because one early exit-input size combination can be the optimal choice for more than one contention level, as in the case of individual models. On the other hand, this individual model set of 4 models can achieve slightly less delay violation and much better accuracy than the early exit model. Therefore, we can say that the high level of granularity of early exit networks is not helpful even in the presence of frequently changing contention. The accuracy of early exit model is only comparable to the smallest individual model which has a significantly lower delay violation of 5.91%.

In the early exit architecture, convolution layer parameters are shared among early exit branches. Therefore, the architecture aims to achieve less parameters than the total parameters of multiple

Model	Accuracy (%)	Delay Violations (%)	Memory Consumption (MB)
i4-128	34.60	05.91	3070
Early exit model	37.03	21.90	3300
Individual models	45.87	20.81	3390

Table 4. Early exit model and individual model set comparison - Dataset: ImageNetV2

.

individual models. However, a large part of the parameters in convolutional neural networks are coming from the fully connected layers at the classifier. The recent and successful EfficientNet architectures can be example for this. EfficientNetB0 has 5,330,571 parameters in total of which 1,281,000 parameters are from the fully connected layer at the classifier. Therefore, whenever we add a branch, we add a fully connected layer and a large number of parameters. As a result, the size of the early exit model is 44 MB, whereas the total size of our individual models is 55 MB (7+9+18+21). The early exit model, of course, would have much less parameters compared to the total of 20 individual models which correspond to each early exit branch-input size combination. However, as we discussed earlier, we do not need that many models for effectively adapting to contention. Moreover, our contention grading and model set pruning framework allows us to decrease the total number of models by intelligently selecting optimal models for the system contention and user requirement.

One drawback of using early exit models at runtime is the switching cost of the branches. Whenever the output of the model is changed to a different early exit branch, an additional time is required for the execution graph. We examined these switch costs. Some switching combinations take more time than the other ones but we did not observe a certain pattern. The average switching cost is 5.58 ms. The inference delays of early exit branches are ranging from 18 ms to 35 ms under no contention. Therefore, the average switching cost can be up to 31% of the inference delay whenever a switching occurs.

In summary, we believe that the proposed approach, which comprises of selecting individual models for contention adaptation is more effective in terms of accuracy, inference latency and runtime overheads compared to early exit based methods.

5.6 Comparison with Slimmable Network Based Method

Even though the original work of slimmable networks [40] does not consider runtime in the presence of contention, it is possible to use them in this context. Slimmable networks use less parameters to create sub-networks in the same backbone architecture. Since slimming is similar to early exit in the sense that they are both dynamic neural network methods and utilize weight sharing, we implement a slimmable neural network and compare our method with it.

We designed a slimmable neural network with 4 switches (0.25x, 0.50x, 0.75x, 1.0x) where the backbone architecture is EfficientNetB0. We use the same individual models that are used in the previous section. Therefore, the architectures of our biggest individual model and 1.0x switch of our slimmable network are exactly the same. The same training configuration is used as specified in the previous section.

5.6.1 The Effect of Batch Size on Slimmable Networks.

Slimming operation is basically using less number of filter channels in convolution operations. Therefore, slimming reduces the FLOPs. However, this does not always translate to speedup. Every operation has an arithmetic intensity value which can be calculated by the ratio of number of FLOPS to number of byte accesses. Similarly, every processor has an ops to byte ratio that can be calculated by the ratio of math bandwidth to memory bandwidth. If the arithmetic intensity

of an operation is smaller than the ops to byte ratio of the processor, the operation is limited by the memory. Conversely, if the arithmetic intensity of an operation is larger than the ops to byte ratio of the processor, the operation is limited by math (arithmetic). Finally, if neither of the math and memory pipelines of the processor are saturated by the operation, the operation is limited by the latency. The latency limitation happens when parallelism of the operation is not enough to saturate the processor's capabilities. While it is possible to calculate the arithmetic intensity of each operation in a neural network, it is not practical to do so since we are using very deep neural networks. Moreover, theoretical arithmetic intensity calculation is only a first-order approximation. Therefore, we provided empirical results with different batch sizes in Figure 19 to show the saturation points of GPU pipelines where the slimming operation becomes useful. It is important to note that these results are specific to a given combination of neural network and GPU. Using a better GPU in all aspects or using a neural network with smaller width would result in requiring larger batch size to make slimming useful. Figure 19 shows that the minimum batch size of 8 is required to achieve a speedup at every slimming point. However, using batch inferences in embedded systems is not useful. Since embedded systems are usually used in real-time applications, waiting for new data for batches and then running batch inference may not be practical. Moreover, since embedded systems are already resource constrained systems, running inference with large batch size takes too much time and results in missing deadlines of many points in the batch data.

Even if we find a very specific scenario where inference with large batch size in an embedded system is required, our proposed method using individual models is still superior compared to the use of slimmable network switches in terms of accuracy. This is because training a slimmable network is a multi objective optimization like early exit since the weights of a slimmable network are shared among different switches. The comparison of individual models and slimmable networks for different batch sizes are shown in Figure 20. Slimmable network switches do not provide a tradeoff in batch size 1 and provide only a partial tradeoff in batch size 4. It starts to provide a tradeoff in batch size 8. However, the individual models provide a tradeoff in all batch sizes and have better accuracy than slimmable network switches. Note that i7-224 individual model and 1.0x slimmable network switch have the exact same architecture. The other individual models do not have the exact same architectures with slimmable switches but since they have similar inference delays for high batch sizes, their accuracies can be fairly compared.

All models in our experiments are implemented in graph execution instead of eager execution. The graph execution requires models to be statically compiled. As a result, it is much faster than eager execution. Since the slimmable networks are dynamic networks, implementing them in graph mode requires different approaches and additional logic. We noticed these implementation differences result in 1-4 ms deviation in inference delay. However, when the batch size is increased, this deviation becomes negligible.

5.6.2 Runtime Performance Comparison.

Even though our framework is designed for real-time systems and large batch sizes are not preferred in real-time systems, we compare our methodology with slimmable networks by increasing batch size for the sake of comparison. We used a batch size of 8 and decreased the intensity of contention compared to previous sections in order to keep the inference delays in a reasonable range. Therefore, the metrics in this section are not comparable to the ones in previous sections.

The temporal comparison and the numerical results for the slimmable model and individual model set are shown in Figure 21 and Table 5. The threshold is determined by the maximum delay of the heaviest model under no contention. This is a fair selection of threshold since the heaviest model is the same in both methods. Table 5 shows the accuracy achieved by our proposed individual model

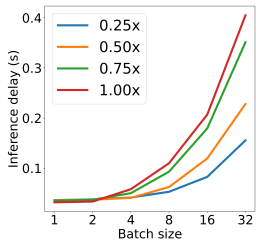

Fig. 19. The effect of batch size on inference delays of switches of a slimmable network

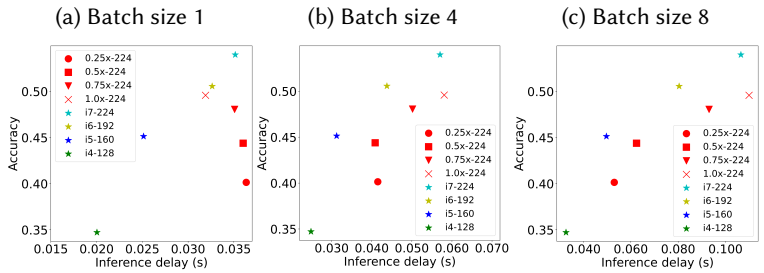

Fig. 20. Accuracy - inference delay plots of individual models and slimmable network switches under different batch sizes

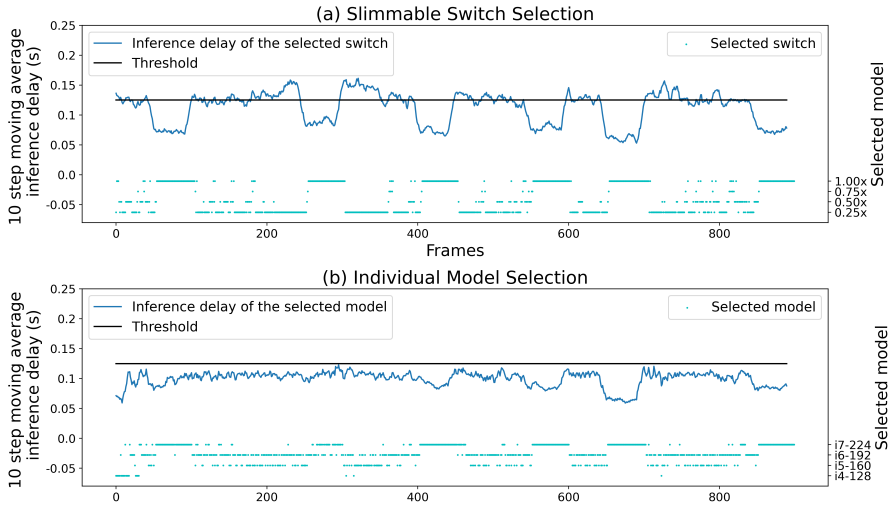

Fig. 21. Temporal comparison of slimmable model and individual model selection under varying contention

Model	Accuracy (%)	Delay Violations (%)
Slimmable model	44.12	42.44
Individual models	51.25	16.55

Table 5. Slimmable model and individual model set comparison - Dataset: ImageNetV2.

selection method is better than using the slimmable model as expected from previous analysis. The delay violation of the individual model set is also significantly better than slimmable model. These results are expected since the individual model set provides a better tradeoff than the slimmable model.

The slimmable networks are similar to the early exit networks in the sense that they share weights for different switches. Therefore, they use less memory compared to individual model set that has same number of models as the switches in the slimmable network. However, as we discussed in the previous sections, our model set pruning technique reduces the number of models for a given system and applications, and therefore achieves efficient memory consumption.

In the end, we believe our proposed approach is more effective in terms of accuracy and inference latency than the slimmable neural network based method. Moreover, our approach does not have

limitations such as minimum batch size depending on the neural network architecture and the hardware as slimmable neural networks do.

6 CONCLUSION

In this paper, we proposed a two stage framework to enable contention-aware adaptive image classification model selection. Our framework takes a deep learning model set, a user requirement and the system with contention and creates a contention-aware application that runs on the system. In the first stage, we define Contention Impact Profiler (CIP) that can profile system contention effect to our application. Then we analyze the profile with kernel density estimation to find the system contention levels. We define Artificial Contention Units (ACU) to regenerate these contention level in a controlled environment. Then, we run 3-stage model pruning on the given model set to select optimal models for the system contention and the user requirement. In the second stage, we define a runtime framework to use previously found models to adapt to changing contention. Our runtime framework employs linear regression models to predict future inference of the models and selects the optimal model for the existing contention. The experimental results show that our predictive model selection outperforms the average of individual models in both accuracy and inference delay violation. Predictive model selection also outperforms the reactive model selection methods and early exit method. We demonstrated our technique using image classification while the contention is created by fusion, RADAR and LiDAR tasks to model an autonomous car environment. However, our framework can work with any neural network based primary application along with any contention applications. For example, the primary application can be object detection while contention can be created by data communication and data encryption. Alternatively, the primary application can be neural network based speech processing and video game graphics can create contention in a mobile system.

ACKNOWLEDGMENTS

This work is partially supported by DARPA under grant number 304259-00001.

REFERENCES

[1] Banner, R., Nahshan, Y., and Soudry, D. Post training 4-bit quantization of convolutional networks for rapid-deployment. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada* (2019), H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, Eds., pp. 7948–7956.

[2] Blalock, D. W., Ortiz, J. J. G., Frankle, J., and Guttag, J. V. What is the state of neural network pruning? In *Proceedings of Machine Learning and Systems 2020, MLSys 2020, Austin, TX, USA, March 2-4, 2020* (2020), I. S. Dhillon, D. S. Papailiopoulos, and V. Sze, Eds., mlsys.org.

[3] Courbariaux, M., and Bengio, Y. Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1. *CoRR abs/1602.02830* (2016).

[4] Cubuk, E. D., Zoph, B., Mané, D., Vasudevan, V., and Le, Q. V. Autoaugment: Learning augmentation strategies from data. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019* (2019), Computer Vision Foundation / IEEE, pp. 113–123.

[5] Eshratifar, A. E., and Pedram, M. Runtime deep model multiplexing for reduced latency and energy consumption inference. In *38th IEEE International Conference on Computer Design, ICCD 2020, Hartford, CT, USA, October 18-21, 2020* (2020), IEEE, pp. 263–270.

[6] Fayyad, J., Jaradat, M. A., Gruyer, D., and Najjaran, H. Deep learning sensor fusion for autonomous vehicle perception and localization: A review. *Sensors 20*, 15 (2020), 4220.

[7] Feng, B., Wan, K., Yang, S., and Ding, Y. SECS: efficient deep stream processing via class skew dichotomy. *CoRR abs/1809.06691* (2018).

[8] Gholami, A., Kim, S., Dong, Z., Yao, Z., Mahoney, M. W., and Keutzer, K. A survey of quantization methods for efficient neural network inference. *CoRR abs/2103.13630* (2021).

[9] Han, S., Pool, J., Tran, J., and Dally, W. J. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada* (2015), C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds., pp. 1135–1143.

[10] He, K., Zhang, X., Ren, S., and Sun, J. Identity mappings in deep residual networks. In *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part IV* (2016), B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds., vol. 9908 of *Lecture Notes in Computer Science*, Springer, pp. 630–645.

[11] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR abs/1704.04861* (2017).

[12] Huang, G., Liu, Z., van der Maaten, L., and Weinberger, K. Q. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017* (2017), IEEE Computer Society, pp. 2261–2269.

[13] Iandola, F. N., Moskewicz, M. W., Ashraf, K., Han, S., Dally, W. J., and Keutzer, K. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *CoRR abs/1602.07360* (2016).

[14] Iandola, F. N., Moskewicz, M. W., Karayev, S., Girshick, R. B., Darrell, T., and Keutzer, K. Densenet: Implementing efficient convnet descriptor pyramids. *CoRR abs/1404.1869* (2014).

[15] Jahromi, B. S., Tulabandhula, T., and Cetin, S. Real-time hybrid multi-sensor fusion framework for perception in autonomous vehicles. *Sensors 19*, 20 (2019), 4357.

[16] Kingma, D. P., and Ba, J. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (2015), Y. Bengio and Y. LeCun, Eds.

[17] Kutukcu, B., Baidya, S., Raghunathan, A., and Dey, S. Contention-aware adaptive model selection for machine vision in embedded systems. In *3rd IEEE International Conference on Artificial Intelligence Circuits and Systems, AICAS 2021, Washington, DC, USA, June 6-9, 2021* (2021), IEEE, pp. 1–4.

[18] Li, Y., Jha, D. K., Ray, A., and Wettergren, T. A. Feature level sensor fusion for target detection in dynamic environments. In *American Control Conference, ACC 2015, Chicago, IL, USA, July 1-3, 2015* (2015), IEEE, pp. 2433–2438.

[19] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S. E., Fu, C., and Berg, A. C. SSD: single shot multibox detector. In *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part I* (2016), B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds., vol. 9905 of *Lecture Notes in Computer Science*, Springer, pp. 21–37.

[20] Luo, J., Wu, J., and Lin, W. Thinet: A filter level pruning method for deep neural network compression. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017* (2017), IEEE Computer Society, pp. 5068–5076.

[21] Meller, E., Finkelstein, A., Almog, U., and Grobman, M. Same, same but different: Recovering neural network quantization error through weight factorization. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA* (2019), K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97 of *Proceedings of Machine Learning Research*, PMLR, pp. 4486–4495.

[22] Park, E., Kim, D., Kim, S., Kim, Y., Kim, G., Yoon, S., and Yoo, S. Big/little deep neural network for ultra low power inference. In *2015 International Conference on Hardware/Software Codesign and System Synthesis, CODES+ISSS 2015, Amsterdam, Netherlands, October 4-9, 2015* (2015), G. Nicolescu and A. Gerstlauer, Eds., IEEE, pp. 124–132.

[23] Rastegari, M., Ordonez, V., Redmon, J., and Farhadi, A. Xnor-net: Imagenet classification using binary convolutional neural networks. In *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part IV* (2016), B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds., vol. 9908 of *Lecture Notes in Computer Science*, Springer, pp. 525–542.

[24] Recht, B., Roelofs, R., Schmidt, L., and Shankar, V. Do imagenet classifiers generalize to imagenet? In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA* (2019), K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97 of *Proceedings of Machine Learning Research*, PMLR, pp. 5389–5400.

[25] Redmon, J., and Farhadi, A. Yolov3: An incremental improvement. *CoRR abs/1804.02767* (2018).

[26] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M. S., Berg, A. C., and Fei-Fei, L. Imagenet large scale visual recognition challenge. *Int. J. Comput. Vis. 115*, 3 (2015), 211–252.

[27] Sandler, M., Howard, A. G., Zhu, M., Zhmoginov, A., and Chen, L. Mobilenetv2: Inverted residuals and linear bottlenecks. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018* (2018), Computer Vision Foundation / IEEE Computer Society, pp. 4510–4520.

[28] Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res. 15*, 1 (2014), 1929–1958.

[29] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. Rethinking the inception architecture for computer

vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016* (2016), IEEE Computer Society, pp. 2818–2826.

[30] Tailor, S. A., Fernández-Marqués, J., and Lane, N. D. Degree-quant: Quantization-aware training for graph neural networks. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021* (2021), OpenReview.net.

[31] Tan, M., and Le, Q. V. Efficientnet: Rethinking model scaling for convolutional neural networks. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA* (2019), K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97 of *Proceedings of Machine Learning Research*, PMLR, pp. 6105–6114.

[32] Taylor, B., Marco, V. S., Wolff, W., Elkhatib, Y., and Wang, Z. Adaptive deep learning model selection on embedded systems. In *Proceedings of the 19th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems, LCTES 2018, Philadelphia, PA, USA, June 19-20, 2018* (2018), Z. Zhang and C. Dubach, Eds., ACM, pp. 31–43.

[33] Teerapittayanon, S., McDanel, B., and Kung, H. T. Branchynet: Fast inference via early exiting from deep neural networks. In *23rd International Conference on Pattern Recognition, ICPR 2016, Cancún, Mexico, December 4-8, 2016* (2016), IEEE, pp. 2464–2469.

[34] Wang, J., Bohn, T. A., and Ling, C. X. Pelee: A real-time object detection system on mobile devices. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada* (2018), S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., pp. 1967–1976.

[35] Xu, R., Koo, J., Kumar, R., Bai, P., Mitra, S., Maghanath, G., and Bagchi, S. Approxnet: Content and contention aware video analytics system for the edge. *CoRR abs/1909.02068* (2019).

[36] Yeong, D. J., Velasco-Hernández, G. A., Barry, J., and Walsh, J. Sensor and sensor fusion technology in autonomous vehicles: A review. *Sensors 21*, 6 (2021), 2140.

[37] Yoo, J. H., Kim, Y., Kim, J. S., and Choi, J. W. 3d-cvf: Generating joint camera and lidar features using cross-view spatial feature fusion for 3d object detection. In *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part XXVII* (2020), A. Vedaldi, H. Bischof, T. Brox, and J. Frahm, Eds., vol. 12372 of *Lecture Notes in Computer Science*, Springer, pp. 720–736.

[38] Yu, J., and Huang, T. S. Network slimming by slimmable networks: Towards one-shot architecture search for channel numbers. *CoRR abs/1903.11728* (2019).

[39] Yu, J., and Huang, T. S. Universally slimmable networks and improved training techniques. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019* (2019), IEEE, pp. 1803–1811.

[40] Yu, J., Yang, L., Xu, N., Yang, J., and Huang, T. S. Slimmable neural networks. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019* (2019), OpenReview.net.

[41] Yuan, Z., Wu, B., Sun, G., Liang, Z., Zhao, S., and Bi, W. S2DNAS: transforming static CNN model for dynamic inference via neural architecture search. In *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part II* (2020), A. Vedaldi, H. Bischof, T. Brox, and J. Frahm, Eds., vol. 12347 of *Lecture Notes in Computer Science*, Springer, pp. 175–192.

[42] Zhang, J., Elnikety, S., Zarar, S., Gupta, A., and Garg, S. Model-switching: Dealing with fluctuating workloads in machine-learning-as-a-service systems. In *12th USENIX Workshop on Hot Topics in Cloud Computing, HotCloud 2020, July 13-14, 2020* (2020), A. Phanishayee and R. Stutsman, Eds., USENIX Association.

[43] Zhang, X., Zhou, X., Lin, M., and Sun, J. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018* (2018), Computer Vision Foundation / IEEE Computer Society, pp. 6848–6856.

[44] Zhu, C., Han, S., Mao, H., and Dally, W. J. Trained ternary quantization. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings* (2017), OpenReview.net.

[45] Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. Learning transferable architectures for scalable image recognition. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018* (2018), Computer Vision Foundation / IEEE Computer Society, pp. 8697–8710.