RT-mDL: Supporting Real-Time Mixed Deep Learning Tasks on Edge Platforms

Neiwen Ling[†], Kai Wang[†], Yuze He[†] and Guoliang Xing[†,*], Daqi Xie[§]
[†]The Chinese University of Hong Kong, Hong Kong SAR, China
[§]Edge Cloud Innovation Lab, Huawei Cloud, Shenzhen, China
Email: lingnw@link.cuhk.edu.hk, kai.wang@my.cityu.edu.hk, yzhh@link.cuhk.edu.hk,glxing@cuhk.edu.hk, xiedaqi1@huawei.com

ABSTRACT

Recent years have witnessed an emerging class of real-time applications, e.g., autonomous driving, in which resource-constrained edge platforms need to execute a set of real-time mixed Deep Learning (DL) tasks concurrently. Such an application paradigm poses major challenges due to the huge compute workload of deep neural network models, diverse performance requirements of different tasks, and the lack of real-time support from existing DL frameworks. In this paper, we present RT-mDL, a novel framework to support mixed real-time DL tasks on edge platform with heterogeneous CPU and GPU resource. RT-mDL aims to optimize the mixed DL task execution to meet their diverse real-time/accuracy requirements by exploiting unique compute characteristics of DL tasks. RT-mDL employs a novel storage-bounded model scaling method to generate a series of model variants, and systematically optimizes the DL task execution by joint model variants selection and task priority assignment. To improve the CPU/GPU utilization of mixed DL tasks, RT-mDL also includes a new priority-based scheduler which employs a GPU packing mechanism and executes the CPU/GPU tasks independently. Our implementation on an F1/10 autonomous driving testbed shows that, RT-mDL can enable multiple concurrent DL tasks to achieve satisfactory real-time performance in traffic light detection and sign recognition. Moreover, compared to state-of-the-art baselines, RT-mDL can reduce deadline missing rate by 40.12% while only sacrificing 1.7% model accuracy.

CCS CONCEPTS

• **Computer systems organization → Real-time system architecture**; • **Computing methodologies → Neural networks**.

KEYWORDS

Real-time Deep Learning, Real-time Scheduling, Edge Computing, Deep Learning System

[*]Corresponding author.

ACM Reference Format:
Neiwen Ling[†], Kai Wang[†], Yuze He[†] and Guoliang Xing[†,*], Daqi Xie[§]. 2021. RT-mDL: Supporting Real-Time Mixed Deep Learning Tasks on Edge Platforms . In *The 19th ACM Conference on Embedded Networked Sensor Systems (SenSys'21), November 15–17, 2021, Coimbra, Portugal.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3485730.3485938

1 INTRODUCTION

In recent years, Deep Learning (DL) has been increasingly adopted by real-time applications running on the network edge, including autonomous driving [30], smart roadside infrastructure [39], embedded computer vision [56], etc. Despite the limited compute resources, edge devices in these applications must support the execution of multiple DL tasks concurrently. For example, a smart lamppost [63] may run different deep neural networks (DNN) for license plate detection [55], pedestrian/vehicle tracking [12, 31], and even collision detection and warning for autonomous driving vehicles [24, 35]. These tasks vary substantially in terms of real-time requirement, level of model accuracy, and resource demand. We characterize this emerging application paradigm as *mixed real-time deep learning tasks*, in which an edge device must execute multiple DL tasks with highly diverse real-time/accuracy requirements. For example, a (semi-)autonomous vehicle must meet stringent performance requirements including tight deadline, low deadline missing rate and high accuracy in detecting imminent on-road collisions, while processing other tasks such as speech recognition for driver voice control in a *best-effort* manner.

Several techniques such as model compression [19, 29, 33, 59, 60] and neural architecture search [5, 9, 49, 50, 53] have been proposed to achieve timely execution of a single DL task on resource-constrained edge platforms. However, different from single DL task execution, mixed DL tasks will lead to resource contention, which may cause unpredictable *response time*. The common wisdom to address this issue in real-time literature is to assign *priorities* to different tasks. However, flexible, priority-based real-time scheduling is not well supported by current DL frameworks like PyTorch [37], TensorFlow [1] and DL accelerators like GPU, NPU [20] on mobile/edge platforms [54, 62]. Moreover, conventional real-time scheduling approaches treat each task as a "black box", which does not exploit the unique characteristics of DL tasks. For instance, an end-to-end DL task usually includes: 1) DNN model inference which not only can be executed more efficiently on GPU, but also has a highly adjustable delay via model compression, and 2) the pre- and post-processing such as data reading from sensors, result encoding etc., which usually run more efficiently on CPU. In this paper, we conduct a comprehensive experimental study that sheds light on the unique characteristics of mixed real-time DL tasks on edge platforms. First, most mainstream DNN models exhibit significant compressibility, while the trade-off between model accuracy and latency over different DNN models is highly diverse. Second, the real-time performance of mixed DL tasks on edge platforms is highly dependent on the task scheduling policy, while priority-based scheduling alone cannot effectively support mixed DL tasks. In particular, understanding the unique compute characteristics (e.g., substantial CPU workload and low GPU utilization) of DL task is critical to improve real-time performance of multiple DL tasks on edge platforms.

Motivated by these findings, we propose RT-mDL, a new real-time DL framework that supports running mixed DL tasks with diverse real-time/accuracy requirements on the edge platforms with heterogeneous CPU and GPU resource. First, to exploit highly diverse compressibility of DNN models, we propose a novel *storage-bounded model scaling* algorithm that generates a series of fine-grained candidate model variants with different compute workloads and accuracies under user-specified storage bound. We then design a new framework that aims to jointly optimize the model scaling and priority-based task scheduling to meet diverse real-time/accuracy requirements of mixed DL tasks. RT-mDL also includes a new priority-based DL task scheduler that uses independent CPU/GPU task queues to substantially improve the CPU/GPU temporal utilization, and a priority-based GPU packing mechanism to improve the GPU spatial utilization. We implement RT-mDL on an F1/10 autonomous driving testbed and three mainstream edge platforms. Our extensive experiments show that RT-mDL can enable multiple concurrent DL tasks to achieve satisfactory real-time performance. Moreover, compared to several state-of-the-art baselines with rule-based scheduling policy, RT-mDL can reduce deadline missing rate up to 40.12% while only sacrificing minor DNN model accuracy.

2 RELATED WORK

DNN Model Inference Acceleration. Many model compression techniques [19, 33, 59, 60] have been proposed to reduce the compute demand of DNN model inference at the price of accuracy degradation. For example, CNN models such as VGG can trade accuracy for compute workload by pruning or quantization. However, most previous works only consider the performance of compressing a single DNN model. Several studies (MCDNN [17], NestDNN [13]) on multi-level on-device DL exploit the trade-off between accuracy and latency of multiple DNN models. Without jointly considering the scheduling mechanism and DNN compressibility, these solutions cannot optimize the performance of concurrent DL tasks on edge platforms with heterogeneous CPU and GPU resource.

Real-time DL Task Scheduling. Several recent efforts are focused on real-time scheduling for DL tasks. S3DNN [65] is a supervised scheduling algorithm that improves the GPU utilization of DNN inference. The work in [6] proposes a deadline-based scheduler for DNN inference on integrated GPUs. DART [54] employs a pipeline-based scheduling architecture with data parallelism for real-time DNN inference requests. However, all these approaches focus on the real-time scheduling of DNN model inference only, while a typical DL task also includes pre-processing (e.g., data reading, decoding, feature extraction) and post-processing (e.g., rendering and encoding) that pose substantial CPU workload [32, 62]. Moreover, such classical rule-based real-time scheduling policies cannot be effectively applied to DL tasks on the current edge platforms. This is because mainstream DL frameworks like PyTorch, TensorFlow and DL accelerators like GPU, NPU [20] on mobile/edge platform [54, 62] lack real-time support such as flexible priority assignment.

Concurrent DL inference on Mobile/Edge Platform. Several recent studies are focused on concurrent DL inference tasks on mobile/edge platforms [4, 38, 52, 58, 62]. Yang *et al.* [58] propose to combine parallelism and pipelining with model sharing to improve the throughput of multiple DL tasks. While effective for multiple DL tasks that share the same DNN model, it is not applicable for mixed DL tasks. Heimdall [62] splits the models into fine-grained units for scheduling. AsyMo [52] builds an optimal execution plan offline by partitioning matrix multiplication (MM) and fairly scheduling tasks among threads. However, these solutions are not designed to meet diverse real-time/accuracy requirements of mixed DL tasks on edge platforms with heterogeneous CPU and GPU resources.

3 MOTIVATION

To understand the execution characteristics of mixed DL tasks, we profile the real-time performance of DL tasks on several GPU-accelerated edge platforms. The results provide key insights into the design of RT-mDL. In our profiling experiments, we select 6 DNN models AlexNet [26], VGG11/13/16/19 [44], ResNet18/34 [18], LeNet [28] that are trained on the CIFAR-10 dataset [25] (for image classification) and MNIST dataset [27] (for handwritten digital recognition), and conduct extensive profiling experiments on a typical desktop-class platform (Intel i9 CPU + NVIDIA RTX 2080 GPU), and two edge platforms (NVIDIA Jetson TX2 and AGX Xavier).

3.1 DL Model Compressibility

As shown in Fig. 1, a DNN model usually consists of various layers such as convolutional layer and fully-connected layer. It can be compressed by reducing the width of each layer (i.e., the filter/channel number of each layer), thereby trading accuracy for lower latency.

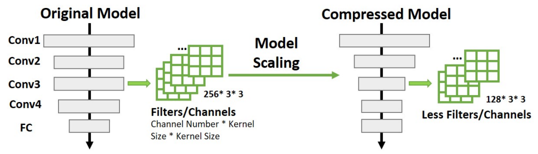


Figure 1: **Process of DNN Scaling.**

Fig. 2(a) shows the latency of the DNN models under different levels of accuracy, and Fig. 2(b) shows the accuracy for different model sizes. Only the results on Xavier are shown here due to space limitation, since the results on TX2 and Desktop are similar. First, it can be seen that almost all the DNN models exhibit a wide region of latency-accuracy trade-off. Taking VGG11 as an example, accuracy loss of one percent (from 81.64% to 80.34%) leads up to a 27.65% decrease of latency (from 10.23ms to 7.40ms) and a 38.54% decrease of storage (from 35.29MB to 21.69MB). Second, the model compressibility is highly diverse across different DNN models. Some DNN models show significantly better accuracy-latency trade-off

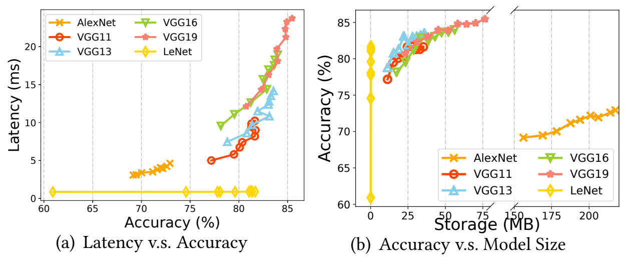


Figure 2: **Latency-accuracy trade-off for different DNN models on Xavier. (Results on TX2 and Desktop are similar to Xavier)**

than other DNN models. For example, with the same accuracy loss of 2%, the execution time of VGG19 on Xavier platform is reduced by 6.99ms, which is 10.4 times that of AlexNet (0.67ms). In summary, **there exists significant trade-off between latency and accuracy, which is also highly diverse across different DNN models.**

3.2 Real-Time Scheduling for Mixed DL Tasks

Real-world DL applications usually require periodically processing input data in real-time. For example, for a real-time object detection task, images may be captured from the camera at 20fps, which triggers an object detection job every 50ms, and each job is expected to complete before the generation of the next image (i.e., within 50 ms). We focus on *periodic tasks* where each instance of execution is referred to as a *job*. The expected completion time of a job is called *task deadline*.

Model scaling [10, 21, 50] is an effective approach to achieve the timely execution of a single DL task under resource constraints. However, mixed DL tasks will lead to the resource contention, and the resultant *blocking time*, defined as the waiting time for the resource occupied by other tasks, may cause unpredictable response time, i.e., the delay between job release and completion.

Under current DL frameworks, concurrently executed DL tasks will occupy the GPU at the same time, and the GPU driver (i.e., CUDA) will schedule the operations (i.e., matrix multiplication in model inference) in a round-robin manner. As a result, a more urgent task with shorter deadline can easily miss its deadline due to blocking of less urgent tasks with longer deadlines. The common wisdom to address this issue in real-time literature is to introduce *priorities* to different tasks. We now show the impact of priority-based scheduling policies on the real-time performance of mixed DL tasks. Since the native DL frameworks such as PyTorch and TensorFlow only provide a two-level priority assignment [54], we developed a naive priority-based scheduler by using an independent thread for each DL task, and adopting a priority queue to control their execution. We choose to run 4 mixed DL tasks simultaneously under all possible settings of task priorities (totally 24). The DL task set used here includes two AlexNet tasks and two VGG11 tasks. The deadlines of the tasks are set to achieve a 30fps (i.e., deadline is 33.3ms) processing of images on Xavier, which is in line with computer vision applications, and they are proportionally adjusted to maintain the same load for Desktop and TX2 platforms.

The best-case, medium and worst-case performance of each task are shown in Fig. 3. We observe that on TX2 the deadline missing rate of AlexNet-2 is 98.19% in the worst case, while reduced to

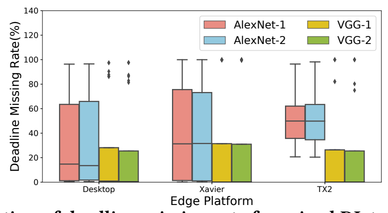


Figure 3: **Variation of deadline missing rate for mixed DL tasks under different task priority assignments.**

20.44% in the best case. Moreover, Fig. 3 shows that the worst deadline missing rates among all tasks under all possible priority assignments are 81.39% on Desktop, 99.33% on Xavier and 72.40% on TX2. In other words, none of the scheduling strategies can achieve low deadline missing rates for all tasks. This is because the tasks with low priorities can be frequently blocked by the tasks with higher priorities, and hence may not be executed in time. This result shows that **priority-based scheduling alone may not achieve satisfactory performance for mixed real-time DL tasks due to significant resource contention**.

3.3 DL Task CPU/GPU Utilization

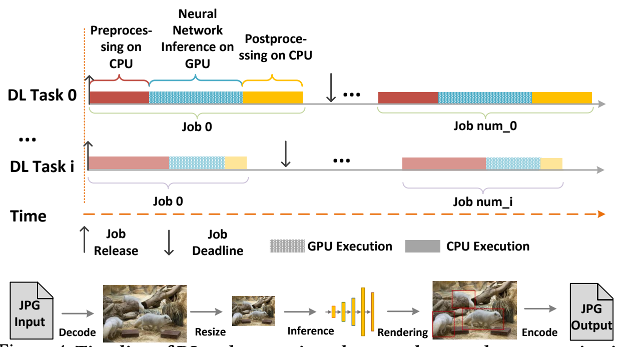


Figure 4: **Timeline of DL task execution: the neural network computation is executed on GPU, and the pre-processing and post-processing are executed on CPU.**

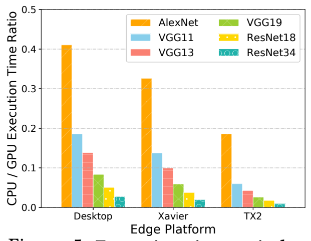


Figure 5: **Execution time ratio between CPU and GPU on different platforms.**

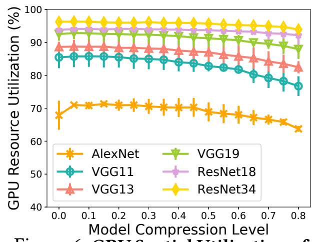


Figure 6: **GPU Spatial Utilization of DNN model under different levels of model compression (on Xavier).**

As shown in Fig. 4, the execution of an end-to-end DL task typically starts with the pre-processing, such as decoding and image resizing, then followed by the neural network computation, and finally the post-processing such as rendering and encoding. For the edge platforms with heterogeneous CPU and GPU resource, the common practice [32, 57, 62] is to execute the neural network computation on GPU while executing pre-processing and post-processing on CPU. Fig. 5 shows that the execution time for pre-processing and post-processing accounts for a considerable portion in task delay, e.g., about 30% of the DNN model execution time for

AlexNet on Xavier. This will lead to a temporal underutilization of CPU/GPU resource, because when a DL task is occupying CPU, the GPU is left idle even when there are other lower-priority DL tasks in the task queue. Moreover, Fig. 6 shows that GPU spatial utilization rate highly depends on the type of DNN model, while it also varies slightly with the model compression rates. More importantly, as is evident in the figure, some DNN models such as AlexNet and VGG11 cannot fully occupy the GPU, resulting in a substantial waste (14.54%-36.25%) of GPU resources. This is because a single DNN inference cannot fully utilize all the GPU computing cores (e.g., 512 cores on Xavier).

These results show that GPU is often not efficiently utilized, and CPU execution time should not be neglected when running mixed DL tasks on the edge. A key observation here is that, **an efficient scheduler should take account of such unique compute characteristics of DL tasks in order to improve CPU/GPU utilization (and hence achieve better real-time inference performance) for multiple DL tasks**.

4 OVERVIEW OF RT-MDL

4.1 Key Idea

This work considers the problem of supporting mixed real-time DL tasks in which an edge device executes multiple different DNNs with highly diverse real-time/accuracy requirements. Our key observation in Section 3 is that we need to carefully consider the task scheduling policies as well as the unique compute characteristics of DL tasks, including significant but diverse latency-accuracy tradeoffs of different DL models, non-negligible CPU execution time, and inefficient spatial and temporal GPU utilization. Motivated by these findings, the key idea of RT-mDL is to jointly optimize model scaling and priority-based task scheduling. Specifically, RT-mDL employs a novel storage-bounded model scaling algorithm to generate a series of model variants, and systematically optimizes the DL execution by finding the efficient combination of task priorities and scaling levels of mixed DL tasks. We propose a new formulation that aims to find the optimal execution strategy that minimizes the accuracy loss for all DL tasks under the given real-time requirements and storage bound. To exploit unique compute characteristics of DL tasks, RT-mDL also includes a new priority-based DL task scheduler which divides a DL task into CPU and GPU subtasks and schedule them using independent CPU/GPU task queues, which substantially improves the GPU and CPU temporal utilization. Moreover, to improve spatial utilization of GPU, the RT-mDL scheduler adopts a new GPU packing mechanism to enable parallel execution of DL inferences with the guarantee of priority.

4.2 Problem Definition

Specifically, we aim to optimize the *execution strategy* $\mathbf{s}$ which represents the model scaling strategy and priority assignment of all DL tasks. The execution strategy $\mathbf{s}$ for a task set with $\mathbf{n}$ DL tasks is parameterized by a vector $\langle k_1, ..., k_i, ..., k_n, h_1, ..., h_i, ..., h_n \rangle$, where task τ_i runs its k_i-th model variant (scaled model) $\tilde{N}_{i,k}$ and is scheduled with priority level h_i, $h_i \in \{1, 2, ..., n\}$. $k_i \in \{1, 2, ..., K_i\}$ where K_i is the maximum number of model variants for DL task τ_i. The optimization problem is to find a strategy $\mathbf{s}$ that minimizes the sum of normalized accuracy loss $\mathrm{LOSS}(\mathbf{s})$ (referred to as *accuracy loss ratio*) for all tasks, while meeting the real-time requirement of each task and an upper bound of storage for all the model variants, which is formulated below.

$$\begin{aligned} &\min_{\mathbf{s}} \ \mathrm{LOSS}(\mathbf{s}) = \textstyle\sum_i \frac{\mathrm{LOSS}_i(\mathbf{s})}{\mathrm{ACC}_i^{max}} \\ &s.t.\ \ \mathrm{MIS}_i(\mathbf{s}) \leq \zeta_i, \quad \textstyle\sum_i \sum_k Storage(\tilde{N}_{i,k}) \leq \overline{Storage} \end{aligned} \tag{1}$$

where for each DL task τ_i, $\mathrm{LOSS}_i(\mathbf{s})$ indicates the model accuracy loss under execution strategy $\mathbf{s}$, compared to the original model accuracy ACC_i^{max} without model scaling. We use *deadline missing rate* to quantify how well the task real-time requirement (i.e., task deadline) is met. The deadline missing rate $\mathrm{MIS}_i(\mathbf{s})$ here denotes the percentage of overtime jobs among all execution jobs of a DL task τ_i. Lastly, ζ_i indicates the real-time requirement of task τ_i, which is the upper bound of the deadline missing rate for task τ_i. For example, $\zeta_i = 5\%$ means that at least 95 percent of the jobs should be completed before its deadline. In other words, this DL task has a 95% probability of finishing in time.

We now use a real-world example to explain the above problem formulation. A (semi-)autonomous driving vehicle [61] is usually equipped with several on-board cameras and Lidars to detect the traffic lights/signs as well as microphones for driver voice control. Each task is periodically released according to the sampling rate of the sensor. The DL tasks for traffic light detection typically have tight deadlines (e.g., 20 ms for a frame rate of 50 fps) as well as extremely low probability of deadline missing (e.g., <1%) due to the criticality of the result. On the other hand, the speech recognition tasks for voice control can tolerate more relaxed deadlines and missing probabilities. A key challenge for supporting such a mixed set of real-time tasks concurrently is the compute resource contention on the vehicle's embedded platforms. Our problem formulation in Eq. (1) aims to maximize the overall accuracy of all the DL tasks while meeting their real-time requirements. To tackle the challenge of resource contention, our key idea is to find a solution that jointly optimizes the scaling levels of all DL tasks as well as the task priorities.

4.3 System Architecture

To meet diverse real-time/accuracy requirements of mixed DL tasks as formulated in Eq. (1), we propose a new system RT-mDL that integrates model scaling and real-time scheduling to provide flexible execution strategies for mixed real-time DL task execution on edge platform with heterogeneous CPU and GPU resource. A bird-eye view of RT-mDL is shown in Fig. 7. RT-mDL has three major components, i.e., storage-bounded multi-level model scaling, priority-based DL task scheduler and execution strategy optimizer, which collaborate to optimize the execution strategy of DL tasks.

To support flexible model scaling for mixed DL tasks, RT-mDL includes a component called *storage-bounded multi-level model scaling*, which compresses the DNN models of DL tasks in depth/width dimension to generate a series of model variants with different levels of accuracy and latency. To ensure that all model variants can be stored on the edge platform, we propose a new weight sharing mechanism in multi-level model scaling to limit the total used storage of generated model variants up to a user-specified bound, e.g., 100MB. We note that some models cannot be efficiently compressed [22, 43]. However, they still benefit from RT-mDL when being executed together with compressible models due to optimized resource usage.

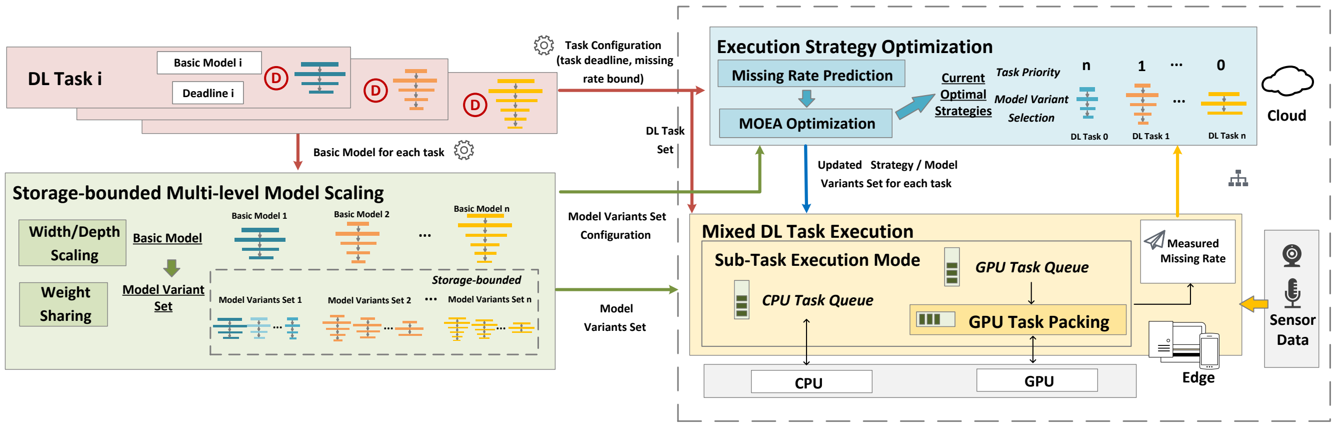


Figure 7: **System Architecture of RT-mDL: Storage-bounded multi-level model scaling generates model variants for each DL task, priority-based DL task scheduler supports flexible real-time scheduling of mixed DL task, execution strategy optimizer searches for the optimal execution strategy of model variant selection and priority assignment.**

The optimization of mixed DL task execution is an iterative process, which aims to find the best execution strategy which consists of model variant selection and priority assignment. Specifically, we adopt a new Multi-Objective Evolutionary Algorithm (MOEA) to solve the optimization problem defined in Eq. (1). A key challenge here is to estimate the response time (i.e., execution time from task release to completion) of each DL task so that we can predict the deadline missing rate. Existing approaches for response time estimation are too pessimistic [47], since they focus on the worst case execution time of each task. Hence, we adopt a proxy model to estimate deadline missing rate from runtime measurements on the edge platform. The estimated real-time performance from the proxy model will be used to find the better execution strategy during the iterative optimization process. Our experimental results show that our optimizer can be implemented efficiently on mainstream edge platforms, such as NVIDIA Jetson TX2 and AGX Xavier[1]. If the edge platform has extremely limited resource, the execution strategy optimizer can also be deployed on the cloud. In such a case, only the execution strategies and deadline missing rate measurements need to be transmitted between the edge and the cloud, which incurs insignificant communication overhead.

DL tasks are executed according to the found execution strategy, which specifies the chosen model variant and execution priority for each DL task. To support flexible real-time scheduling of mixed DL tasks, we design a priority-based DL task scheduler. It constructs independent priority-based CPU/GPU task queues, which achieves efficient utilization of heterogeneous CPU and GPU resource. We also design a priority-based GPU task packing algorithm to improve the spatial utilization of concurrent GPU sub-task execution, through enabling parallel execution of DNN inferences under the guarantee of priority. The integration of CPU/GPU sub-task model and priority-based GPU task packing can effectively improve the GPU utilization. In temporal dimension, when the CPU is occupied by a DL task, GPU interferences from other DL tasks can be scheduled on GPU. In spatial dimension, GPU cores can be utilized by multiple DNN inferences at the same time.

[1]In the remainder of this paper, we assume the optimizer is implemented on the edge device, unless otherwise indicated.

5 DESIGN OF RT-MDL

5.1 Storage-bounded Multi-level Model Scaling

To achieve flexible accuracy-latency trade-offs among multiple DL tasks (Section 5.2), in this section, we design a *Storage-bounded Multi-level Model Scaling* algorithm, where we carefully scale the architecture of the basic DNN model of each DL task to create a series of DNN model variants with different workloads. In order to ensure that all model variants can be stored on the resource-constrained edge platform for optimization at run-time, we also propose a fine-grained weight sharing mechanism to bound the total used storage of generated model variants.

In our design, to achieve fine-grained latency-accuracy trade-off, we adopt the mainstream trade-off approach, model architecture scaling, to generate multi-level model variants, although other approaches such as quantization, knowledge distillation are also applicable. The problem of generating model variants with storage bound is formulated in Eq. (2), in which the objective is to minimize the accuracy loss of the model variants under storage constraint while their compute workloads decrease in an equal gradient, which leads to different levels of execution latency.

$$\begin{array}{l} \forall\, i,k \quad \min\ \mathcal{L}(\mathcal{W}_{i,k}, \tilde{N}_{i,k}) \\ s.t.\ \ FLOPS(\tilde{N}_{i,k-1}) - FLOPS(\tilde{N}_{i,k}) \geq \frac{\varepsilon}{K_i} \quad , \\ \sum_i Storage(\bigcup_{k=0}^{K} \mathcal{W}_{i,k}) \leq \overline{Storage} \end{array} \tag{2}$$

$\mathcal{L}(\cdot)$ is the loss function of the k-th model variant $\tilde{N}_{i,k}$ with model parameters $\mathcal{W}_{i,k}$ for DL task τ_i. The compute workload of a model variant is represented by $FLOPS(\cdot)$ and the scaling gradient ε indicates the maximum scaling granularity. $Storage(\cdot)$ denotes the storage space occupied by the generated model variants, including the basic DL model $\tilde{N}_{i,0}$.

Width and Depth Scaling. We generate multiple candidate model variants for each DL task via the model scaling approach. The final model variants selected for each task will be determined in the weight sharing process described later in this section. Multi-level model scaling can be conducted either by width scaling or depth scaling. Width scaling adjusts the width of DNN models, and we use filter pruning technique here to reduce the channels of each DNN layer. We iteratively prune a subset of filters from the basic DNN

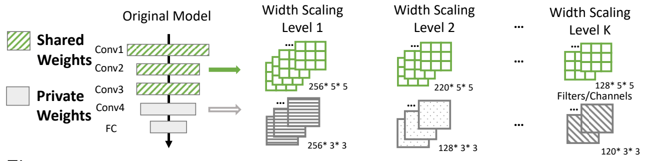


Figure 8: **Model scaling with partial wight sharing. The same type of weights are represented by the same pattern.**

model $\tilde{N}_{i,0}$ for DL task τ_i, and hence generate a series of candidate model variants $\{\tilde{N}_{i,1}, \tilde{N}_{i,2}, ..., \tilde{N}_{i,K_{max}}\}$ with a decreasing gradient $\frac{\varepsilon}{K_{max}}$ of compute workload (i.e., FLOPS). For depth scaling, we first identify the repeating units of the DNN model architecture, where one unit can be a single layer or combined multiple layers such as the *block* in MobileNetV3 or the *inception* in GoogLeNet. Then, similar to width scaling, we iteratively remove the units from the basic model and hence generate a series of candidate model variants with different scaled levels.

Multi-level model scaling can also be achieved by scaling DNN architecture in both width and depth dimensions simultaneously. However, such an approach will lead to an explosion of the candidate model variants number to $O(K_{max}^2)$ for each DL task. Hence, in this work, we only consider the model scaling in one dimension. According to the recent studies [51], smaller model size does not always correspond to a reduced inference time due to the possibility of increased memory access. This can be avoided by filtering out abnormal model variants based on a one-time profiling of generated model variants on the target edge platform.

Weight Sharing and Retraining. These model variants incur significant storage overhead on the edge platform, since most DL models contain millions of neurons and the model weights are in the order of 100MB. For example, it costs a total of 605.6MB to store all the model variants of VGG19 with 9-level scaling (5% FLOPS gradient between adjacent model variants). The total storage cost can grow quickly when the variants of multiple DNN models need to be stored locally. To address the storage capacity limitation while maintaining the descent gradient of the selected model variants, we design a fine-grained weight-sharing mechanism in the following.

As shown in Fig. 8, under weight sharing mechanism, model weights $\mathcal{W}_{i,k}$ of the k-th model variant consist of its private weights $W_{i,k}$ and shared weights W_i^*, which are shared among all the model variants of DL task τ_i. Shared weights and private weights are separated based on a layer granularity as shown in Eq. (3).

$$W_i^* = \sum_{j=0}^{J_{share}} W^j, \; W_{i,k} = \sum_{j=J_{share}}^{J_{all}} W_k^j \tag{3}$$

J_{share} is the number of layers that share weights among all model variants, and K is the number of extracted model variants from all the K_{max} generated model variants (extracted in the equal interval). The total storage usage of task τ_i can be rewritten as in Eq. (4), which reduces the storage usage of shared layers among model variants.

$$Storage(\bigcup_{k=0}^{K} \mathcal{W}_{i,k}) = Storage(W_i^*) + \sum_{k=0}^{K} Storage(W_{i,k}) \tag{4}$$

Given the storage bound $Storage_{bound}$, we search for J_{share} and K to satisfy Eq. (2). Considering that a larger K value corresponds to a finer grained model variant selection, we start from K_{max} and calculate the storage of all sharing mechanisms. If there exist solutions that meet the condition, we adopt the sharing scheme closest to the storage bound. If not, we increase the extraction interval by one and repeat. If there exist two types of models, the complexity of the whole search process is $O(K_{max} * J_{all}^1 * J_{all}^2)$, where J_{all} is the maximum number of shared layers for each model type. As an example, the total search time for VGG11 and AlexNet under the storage bound of 2500MB is about 176s.

To reduce the accuracy loss caused by model scaling, a retraining process is applied through global training for shared weights W_i^* and local training for private weights $W_{i,k}$. For each loop of global training, we fix the private model weights of each model variant, and jointly train the shared weights by updating the weights with the loss accumulation of all the model variants. In local training, we fix the shared wights and train the private weights of each model independently.

5.2 Joint Model Variant Selection and Task Scheduling

As discussed in Section 4.3, to optimize the mixed DL task execution on edge platforms, we adopt Multi-Objective Evolutionary Algorithm (MOEA) to search for an optimal execution strategy for selecting DNN model variants and scheduling DL tasks in an iterative process of optimization.

There are several challenges to find the strategy $\mathbf{s}$ that satisfies the task real-time requirements and minimizes the total accuracy loss ratios $\text{LOSS}^{\text{nor}}(\mathbf{s})$, as defined in Eq. (1). First, the deadline missing rate $\text{MIS}_i(\mathbf{s})$ for a given strategy $\mathbf{s}$ does not have closed-form expressions. It can be calculated based on the response time of the task execution. However, existing approaches for response time analysis are too pessimistic, since they aim to bound the worst case execution time of each task, which is often significantly longer than its actual execution time. Moreover, the problem defined in Eq. (1) has a large solution space since each execution strategy consists of decisions on both model variant selection and task priorities. For K model variants per task and n tasks, the solution space is $O(n!K^n)$. The effective profiling time for each strategy is in the order of minutes, which is prohibitively expensive to search exhaustively.

We tackle these challenges by adopting multi-objective optimization and performance approximation techniques. We employ a proxy model $\mathbf{f}(\cdot) = [f_1(\cdot), ..., f_n(\cdot)]^T$ to predict the deadline missing rate of each task for a given execution strategy, and transform the problem in Eq. (1) into a multi-objective optimization problem in Eq. (5), where each constraint in Eq. (1) corresponds to an optimization objective $f_i(\mathbf{s})$, i.e., $f_i(\mathbf{s}) \approx \text{MIS}_i(\mathbf{s})$.

$$\min_{\mathbf{s}} \quad [f_1(\mathbf{s}), ..., f_n(\mathbf{s}), \text{LOSS}^{\text{nor}}(\mathbf{s})] \tag{5}$$

Afterwards, we optimize the execution strategy s in an iterative optimization process with Multi-Objective Evolutionary Algorithm (MOEA), a widely adopted effective optimization algorithm [2, 23]. For each iteration, we will use MOEA to search for the optimal strategies, and the real-time performance of new strategies generated during MOEA process are predicted by the proxy model. The searched current optimal strategies will be used for actual real-time performance measurement, and the measurement results will then be used by the proxy model for further training. Such an iterative optimization approach allows RT-mDL to measure the real-time performance of multiple different execution strategies and then progressively optimize the selection of execution strategies.

Proxy Model for Missing Rate Prediction. We train a function $\mathbf{f}(\cdot) = [f_1(\cdot), ..., f_n(\cdot)]^T$ where each $f_i(\cdot)$ is a separate proxy model

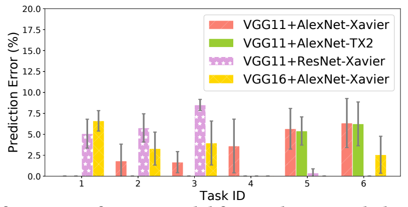


Figure 9: **Performance of proxy model for predicting task deadline missing rates.**

that predicts the deadline missing rate of task τ_i for a given strategy **s**. The proxy model is based on Random Forests (RF) approach [23], although other methods such as Gaussian Process Regression (GPR) are also applicable. Given a set S of strategies whose actual performance are measured on the edge platform, the training process can be regarded as minimizing the square error (MSE) with respect to set S.

$$\min \sum_{\mathbf{s}\in S} \left\| [f_1(\mathbf{s}), ..., f_n(\mathbf{s})]^T - [\mathrm{MIS}_1(\mathbf{s}), ..., \mathrm{MIS}_n(\mathbf{s})]^T \right\|_2 \quad (6)$$

We demonstrate the effectiveness of the proxy model $\mathbf{f}(\cdot)$ for predicting deadline missing rates of DL tasks under three DL task sets: VGG11 + AlexNet, VGG11 + ResNet, VGG16 + AlexNet on TX2 and Xavier platforms. We measure the error between the predicted and the actual deadline missing rates for the strategies whose actual performance are measured on the edge platform during the MOEA optimization process. Fig. 9 shows that for all DL task sets, the prediction error fluctuates below 10%, and the proxy model achieves considerable prediction performance.

MOEA-based Strategy Optimization. The advantage of applying MOEA-based strategy optimization is that infeasible or low-performance strategies can be efficiently eliminated during the search process in MOEA. For example, we add the following two rules for eliminating candidate strategies and reducing search space. First, for each new strategy **s**, the estimated deadline missing rate $f_i(\mathbf{s})$ for task τ_i will be set to zero if it satisfies the real-time requirement in Eq. (1). This will make strategy **s** outperform more new strategies during optimization, forcing the found strategies to excel in other dimensions, e.g., deadline missing rates of other tasks. Second, when two strategies are compared, we neglect the dimension of accuracy loss in Eq. (5). This will also make strategy **s** outperform more candidate strategies in terms of missing rate and for the same reason the found strategies are more likely to satisfy real-time requirements. Besides the rules introduced above, we also adopt specific cross-over and mutation methods for model variant selection and priority assignment, and an accuracy-based sorting algorithm to select the output strategies, which are omitted due to space limitation. As a result, the MOEA algorithm can efficiently produce high-performance strategies with high probability.

5.3 Priority-based DL Task Scheduler

In this section, we will introduce our priority-based DL task scheduler for edge platforms. Our design allows the existing DL frameworks (e.g., PyTorch, TensorFlow) and DL accelerators (e.g., GPU, NPU) to support mixed real-time DL task execution on mobile/edge platforms. Our profiling results in Section 3 show that supporting efficient mixed real-time DL tasks on edge platform should not only employ priority assignment but also improve CPU/GPU utilization by considering the unique compute characteristics of DL tasks. Motivated by these findings, we design a new priority-based scheduler that includes independent CPU/GPU task queues and GPU-packing mechanism, which has three major key advantages: i) Support flexible priority assignment; ii) Separate the CPU/GPU sub-tasks, which enables more efficient temporal utilization of CPU/GPU resource; iii) Enable GPU packing of multiple DNN inferences, which increases the GPU spatial utilization.

DL Task Model. As discussed in Section 3.3, a DL task consists of CPU execution part for pre-/post-processing like decoding and GPU execution part for DNN model inference. Therefore, each DL task τ_i ($i \in \{1, 2, ..., n\}$) comprises three sequential sub-tasks $C_{i,1}, G_i, C_{i,2}$ where $C_{i,q}$ denotes the execution time of the q-th CPU sub-task (pre-processing for $C_{i,1}$ and post-processing for $C_{i,2}$), while G_i is the execution time of GPU sub-task. As we introduced in Section 3.2, real-world DL applications usually require periodically processing input data in real-time. Therefore, in our DL task model, each DL task τ_i is a periodic task with period T_i and a user-defined relative deadline D_i, where D_i can equal to T_i. Each task τ_i is thus defined by an array $(C_{i,1}, G_i, C_{i,2}, D_i, T_i)$.

Priority-based CPU-GPU Sub-Task Scheduling. When a DL task starts execution, our scheduler will periodically release each job of this DL task according to its task period, as shown in Fig. 4. Job here refers to a unit of DL task execution, hence it has the same priority as its corresponding task. If a DL job is not finished in time, which means its response time exceeds the task deadline, our scheduler will not terminate the delayed DL job but wait for it to complete before releasing the next job.

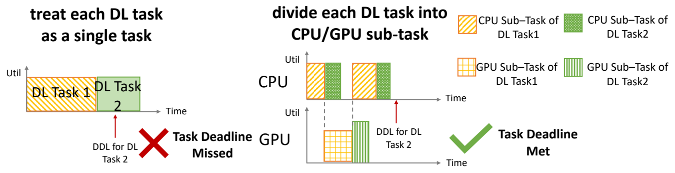


Figure 10: **Improve the efficiency of priority-based scheduling for mixed DL tasks by dividing a DL task into CPU and GPU subtasks and schedule them using independent CPU/GPU task queues.**

To achieve efficient GPU/CPU resource utilization, as shown in Fig. 10, our scheduler divides the DL task into the CPU sub-task and the GPU sub-task by Mutex, and puts these sub-tasks into independent priority-based CPU and GPU task queues. Each sub-task will inherit the priority of the original task, and our scheduler will release CPU/GPU resource to the sub-task at the head of the priority queue. In this way, the CPU sub-task from low-priority task can be executed when the high-priority task finishes its CPU sub-task and starts its GPU sub-task, thus achieving a more efficient temproal utilization of CPU/GPU resource.

To schedule CPU sub-tasks, we create an independent CPU *thread* for each DL task, and the scheduler will release the CPU resource to each thread according to the task priority. We choose not to run DL task in the CPU *process* manner, because the process creation will put the GPU sub-tasks (DNN model inference) from different DL tasks into different GPU *contexts*[2]. Different GPU contexts cannot be executed on the mobile GPU in parallel, which will lead to inefficient use of the GPU. The scheduler also supports multicore CPU execution through a global scheduling mechanism.

[2]A GPU CUDA context is analogous to a CPU process [36]

Specifically, a global CPU task queue is created for all DL tasks even when there exist multiple CPU cores. Once a CPU core becomes idle, the scheduler will release resources to the sub-task at the head of the queue.

To schedule GPU sub-tasks, we first create a similar priority queue using Mutex. We note that the GPU tasks here also include some CPU operations, such as GPU kernel launching. Therefore, the resource required by the GPU sub-task can also be controlled by our scheduler.

GPU Task Packing. In order to improve the GPU spatial utilization for model inference, the scheduler also includes a priority-guided GPU packing mechanism, which enables parallel execution of DL tasks under the guarantee of priority. The reason we consider packing the GPU sub-tasks instead of CPU sub-tasks is twofold. First, GPU has more parallel execution units (Arithmetic Logic Unit) than CPU. Second, our profiling results in Section 3.3 show that model inference cannot fully utilize all the execution units on GPU. To achieve GPU task packing, the scheduler will employ two GPU *streams* [3] (i.e., *High-priority Stream* (HS) and *Low-priority Stream* (LS)) for DNN inference (GPU sub-task) in each DL task. The two-level priority stream is supported by most GPU-accelerated edge platforms (e.g., NVIDIA Jetson TX2 and Jetson AGX Xavier). After the GPU sub-task at the head of priority queue receives the starting signal, it will be pushed into the stream of high priority. The scheduler then search for another sub-task that can be executed in parallel with the current GPU sub-task and put it into the low-priority stream for execution. In this way, GPU spatial utilization can be improved efficiently. The specific packing rule in our scheduler is: (i) when HS is empty, the task with the highest priority in GPU task queue is submitted to HS only if it has higher priority than the current task processed in LS; (ii) when LS is empty, the packing algorithm finds task $\tau_{i'}(G)$ via looking ahead in the GPU task queue until the packing condition in Eq. (7) holds for task $\tau_{i'}(G)$ and the other task $\tau_i(G)$ in HS. The packing condition is formulated as follows,

$$Priority_{i'} < Priority_i, Utilization_{i'} + Utilization_i \leq \theta \tag{7}$$

where $Priority_i$ is the configured priority level and $Utilization_i$ is the spatial-dimension GPU utilization rate of the GPU sub-task $\tau_i(G)$, and θ is a controllable threshold for the maximum allowed GPU spatial utilization rate. Our experiments show that the GPU packing algorithm is effective in increasing GPU spatial utilization. In particular, it can reduce the average per-task response time on GPU by 34.06% (15.39ms v.s. 23.34ms) when running 6 DL tasks on Xavier. We omit the details due to space limit.

6 EVALUATION

We first present an end-to-end evaluation on an F1/10 [8] autonomous driving testbed which demonstrates that RT-mDL achieves significantly better performance than traditional execution strategies for 4 mixed real-time DL tasks. We then evaluate the performance of different components of RT-mDL in Section 6.3 - 6.5.

6.1 Implementation and Experiment Setup

We use NVIDIA Jetson TX2, Nano and AGX Xavier as edge platforms in our experiments (hardware configurations are shown in Table 1). The Desktop is used for model variants generation, and Laptop is used for strategy optimization in the case study. The details of DL tasks evaluated in our experiments are shown in Table 2.

[3]GPU tasks can be executed in parallel under the same level of streams

We choose PyTorch as the DL framework for model scaling, and use LibTorch (C++ frontend of PyTorch) with CUDA library for DL task inference on the edge. Each DL task has two versions of implementation: Python3 version is used for model variant generation and a C++11 version is used for actual execution on the target edge platform. This is because Python lacks support for thread scheduling due to its Global Interpreter Lock (GIL). Hence, we generate model variants and train models via Python, and the generated models are converted to Torch Script format to work with C++ code (Libtorch). The models can also be converted to ONNX format for C++ frontend under other DL frameworks like TensorFlow, MXNet and MindSpore. CPU affinity is fixed for DL task execution, and the working frequency of CPU and GPU on the edge is fixed to the maximum value. The GPU utilization of each DL task is measured by accessing the system log file. For baselines with rule-based scheduling policies we used in the evaluation is achieved through our priority-based scheduler.

Table 1: **Platforms used in evaluation experiments.**

Platform	GPU	CPU	Memory	Storage
Xavier	512-core Volta	8-core ARMv8.2	16GB	32GB
TX2	256-core Pascal	2-core ARM Denver + 4-core ARM A57	8GB	32GB
Nano	128-core Maxwell	4-core ARM A57	4GB	microSD
Desktop	RTX2080	8-core Intel i9-9900K	32GB	5TB
Laptop	Intel Iris Plus	4-core Intel	16GB	500GB

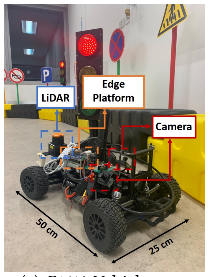
Table 2: **DL tasks used in evaluation experiments.**

DL Task Type	Dataset	DNN Model	Pre-/Post-Processing
Image Classification	CIFAR10 [25]	AlexNet, ResNet18/34, VGG11/13/16/19	Data Fetch, Image Resizing
Sign Recognition	GTSRB [46]	VGG11/19, ResNet18/34	
Object Detection	Self-collected traffic light dataset	tiny-YOLO [40]	Image Reading from Camera, Bounding Box Regression
Sound Classification	UrbanSound [41]	SBCNN [42]	Down-Sampling, MFCC Feature Extraction
Emotion Recognition	Ravdess [34]	LSTM [16]	

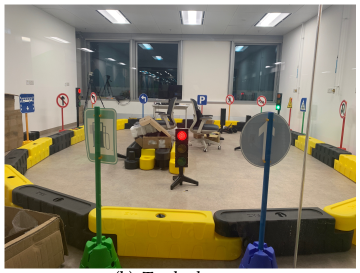
6.2 End-to-end System Evaluation

We implement RT-mDL and evaluate the end-to-end system performance in an autonomous driving testbed. As shown in Fig. 11(a), the F1/10 Autonomous Vehicle [8] is equipped with two cameras, a LiDAR and a small heterogeneous embedded platform (NVIDIA TX2 with Orbitty Carrier board). The main function of this system is to detect traffic lights on both sides of the road while classifying the traffic signs at the same time. There are 4 real-time DL tasks running on this platform: Task1-2 for traffic light detection (with tiny-YOLO-v3 model on live data), and Task3-4 for traffic sign recognition (with VGG11 model on GTSRB dataset preloaded on the vehicle). Fig. 11(b) and Fig. 11(c) show the actual scene and layout of the experiment. The autonomous vehicle runs a lane detection algorithm to follow along the lane fence at the speed of 1 m/s, and detects high and low traffic lights through two cameras separately. Some images captured by the vehicle are shown in Fig. 12. The tiny-YOLO model is pre-trained on the dataset collected from four different runway scenes (with different placement of traffic lights and traffic signs). The execution time of different segments in these DL tasks are shown in Table 3.

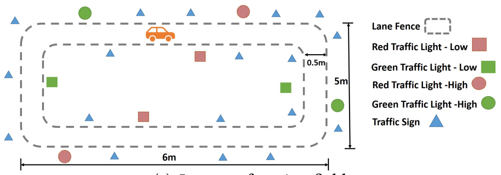
Performance of Execution Optimization. We first evaluate the performance of RT-mDL by comparing RT-mDL with two baselines: UM-NMC and DM-MC. The first baseline UM-NMC selects DL



(a) F1/10 Vehicle



(b) Testbed setup



(c) Layout of testing field

Figure 11: **F1/10 autonomous driving testbed.**

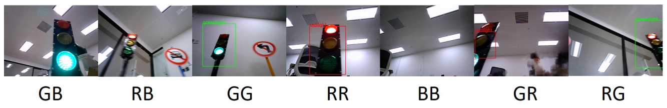


Figure 12: **Examples of detection results ('G'- Green, 'R' - Red, 'B' - Blank). The first letter of the image label (e.g., 'R' of 'RB') represents the ground truth of the image, while the second letter represents the detection result.**

models without scaling and executes them under the Utilization Monotonic scheduling rule. DM-MC chooses model variants with model scaling for execution under Deadline Monotonic scheduling rule. Although the task workload is reduced under this execution mechanism, the detection performance will be influenced by the model accuracy as shown in Eq. (8). A recorded video clip of real-time detection results from these three methods is available at the link[4].

We use RT-mDL to find the optimal execution strategy under this application scenario that meets the task deadline missing bound of 10%. The depth of tiny-YOLO model and the width of VGG11 model are scaled to generate model variants. The retraining process after scaling is conducted on their pre-trained dataset. We deploy the generated model variants at once on the vehicle. The execution strategy is optimized via the iterative optimization process introduced in Section 4.3. The results summarized in Table 3 show that all the DL tasks meet the deadline missing rate bound, while only suffering minor accuracy degradation.

Table 3: **Task Parameter and Results of RT-mDL.**

DL task	GPU sub-task (ms)	CPU sub-task1 (ms)	CPU sub-task2 (ms)	Sampling rate (fps)	DMR (%)
YOLO-Obj_Det-1	19.62	7.99	7.06	12	6.54
YOLO-Obj_Det-2	19.62	7.99	7.06	12	4.92
VGG-Sign_Rec-1	13.16	1.40	0.26	20	9.57
VGG-Sign_Rec-2	13.16	1.40	0.26	20	5.36

Table 4 shows Missed Alarm Rate (MAR) and False Alarm Rate (FAR) for each frame and probability for detecting red traffic light (Det_Prob). MAR and FAR in the table are calculated for single frame, which is given by the accuracy of executed model variants and the deadline missing rate as shown in Eq. (8).

[4]https://aiot.ie.cuhk.edu.hk/Project_EdgeAI.html

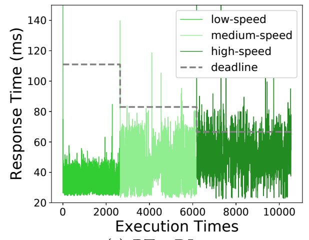
Table 4: **Traffic Light Detection('HL'-high traffic light, 'LL' - low traffic light, 'NMC' – no model scaling, 'MC' – model scaling, 'UM' – Utilization Monotonic, 'DM' – Deadline Monotonic).**

Approach	MAR (%)	FAR (%)	Det_Prob 0.5m(%)	Det_Prob 0.75m(%)
RT-mDL (HL)	**9.30**	**0**	**98.70**	**99.35**
UM-NMC (HL)	27.17	0	79.47	79.13
DM-MC (HL)	27.90	0.76	78.20	77.59
RT-mDL (LL)	**9.09**	**0.27**	**98.78**	**99.40**
UM-NMC (LL)	14.56	0.42	95.62	96.92
DM-MC (LL)	33.73	0.91	67.24	64.04

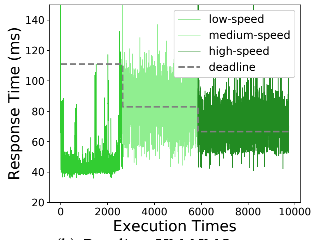
$$FAR = \frac{FP}{FP+TN} = \frac{FP(\tilde{N}_k)\times(1-MIS)}{FP+TN}$$
$$MAR = \frac{FN}{TP+FN} = \frac{FN(\tilde{N}_k)+TP(\tilde{N}_k)\times(1-MIS)}{TP+FN} \tag{8}$$

False Negative (FN) here means that no traffic light is detected or the detection result indicates green light when the traffic light is red. False Positive (FP) here means that the detection result is red light when there is no traffic light or traffic light is green. Given the per-frame MAR that represents the probability of unsuccessful detection, we can calculate the probability of detecting a red traffic light under different fusion criteria. For example, under the majority voting rule, if more than half of the live images during a distance of 0.5m are detected as red light, the detection result is deemed as red light. We evaluate two scenarios with distance thresholds of 0.5m and 0.75m. We note that they are usually set according to the driving speed, field of view of car cameras, and response time of vehicle breaking system. Table 4 shows that the vehicle with RT-mDL reduces the missing alarm rate for each frame by 17.87% and 18.3% respectively, compared with UM-NMC and DM-MC baselines for high traffic light detection. This result indicates that the vehicle has a lower probability for missing the red traffic light. Our approach achieves near 100% detection probability, and increases the *Det_Prob_0.5m* by 19.23% and 20.5% compared with other two execution mechanisms, suggesting that the autonomous vehicle with our framework has a substantially higher probability of stopping at a red traffic light.

Adaptation to Scenario Variations. We also evaluate the adaptation performance of RT-mDL. As we introduced in Section 4.3, RT-mDL can adapt to scenario variation by searching for the optimal solution for various task execution conditions before deployment and then adjust the strategy dynamically at runtime. In this autonomous driving application, we build various scenarios by running the F1/10 autonomous vehicle at different speeds, resulting in different execution time requirements (deadlines). Specifically, the vehicle runs at three speeds of 0.75m/s, 1m/s and 1.25m/s for 300 seconds (2500-4200 jobs), and the corresponding sampling rates are 9fps, 12fps and 15fps, respectively.



(a) RT-mDL



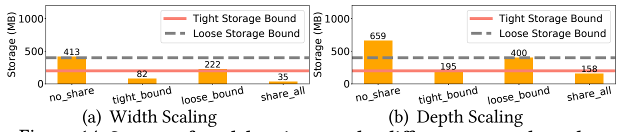
(b) Baseline: UM-NMC

Figure 13: **Performance of Scenario Adaptation.**

Fig. 13 shows the response time of Task2 (YOLO-based traffic light detection) over three speed levels with and without RT-mDL. Fig. 13 (a) shows that RT-mDL consistently maintains a deadline missing rate below the threshold 10% among all the speed levels (9.28% for high-speed). In contrary, the UM-NMC strategy, will cause massive deadline misses under high-speed level (60.09% for medium-speed, 71.66% for high-speed). This is because the higher speed corresponds to more severe resource contention, which leads to more deadline misses. We can see that a shorter response time does not always correspond to a lower deadline missing rate. Under the same UM-NMC strategy, the response time of high-speed (71.7ms on average) is shorter than the medium-speed (85.42ms on average), although its missing rate is higher. This result confirms our observation that fixed rule-based scheduling cannot achieve satisfactory real-time performance for mixed DL tasks. Hence, we cannot evaluate the real-time performance of the task only by the length of the response time. We also observe that there is a spike in the first job of DL task, this is because the DNN inference is lazy initialized in the current DL frameworks, which takes about 87 ms. Overall, the results in this section show that RT-mDL can adapt to significant scenario variation and maintain consistent low deadline missing rate under various driving speeds.

6.3 Performance of Model Scaling

We now evaluate the performance of our storage-bounded multi-level model scaling algorithm under tight and loose storage bounds. We generate model variants for VGG11 and LeNet via width scaling, while adopting depth scaling for VGG19 and ResNet34. To evaluate the storage saved by our method, we compare with two baselines: *no_share* and *share_all*. The results in this section show that our multi-level model scaling method can meet different storage requirements while retaining the same accuracy level.

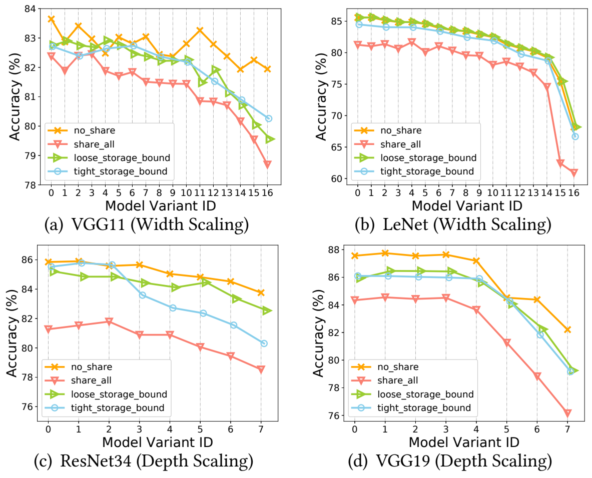


(a) Width Scaling (b) Depth Scaling

Figure 14: **Storage of model variants under different storage bound.**

Fig. 14 shows the overall storage of all model sets, we observe that model sets can meet storage bound under either tight bound or loose bound. For example, the overall storage of VGG11-LeNet model set is 82.05 MB and 221.73 MB under *tight_bound* (100MB) and *loose_bound* (250MB) respectively, both of which meet the storage bounds. Without weight sharing (*no_share*), it takes 412.77MB to store all the model variants of VGG11 and LeNet. The results are also consistent for depth scaling, on the model set VGG19 and ResNet34. Hence, we conclude that our method can meet the storage requirements on model variants.

Fig. 15 shows the accuracy of each model variant generated under different storage bounds. The accuracy of basic model is different of each method in Fig. 15, because the weights of basic model will be retrained together with other model variants. We observe that model variants generated by RT-mDL can achieve higher accuracy than those generated by *share_all* method, e.g., 3.67% per model variants for ResNet34. This is because weight sharing among all the model variants will ensure the shared weights fit well for all the model variants, regardless of the characteristics of each individual



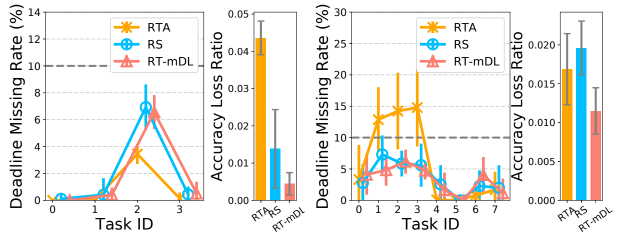
(a) VGG11 (Width Scaling) (b) LeNet (Width Scaling)
(c) ResNet34 (Depth Scaling) (d) VGG19 (Depth Scaling)

Figure 15: **Model variant accuracy with different storage bound.**

model variant. Without weight sharing, the accuracy of each model variant is a little higher, at the price of high overhead in storage usage, e.g., 464MB more storage usage for ResNet34-VGG19.

6.4 Effectiveness of Strategy Optimization

To evaluate the effectiveness of the proposed MOEA-based optimization approach in RT-mDL, we compare its performance with other two methods for searching feasible execution strategies. We design a baseline that optimizes the execution strategies based on Response Time Analysis (RTA) [7], which is widely adopted in real-time literature for bounding the response time of each task under specific scheduling algorithm [7, 11, 54, 57]. The difference from our approach is that real-time performance of DL task set under different execution strategies is evaluated by calculating the response time. To validate the effectiveness of multiple objective optimization, we design a baseline based on Random Search (RS). Under this approach, an execution strategy is randomly generated each time, and the performance of the execution strategy is evaluated on the target edge platform and then stored in the database. The search process terminates until the best strategy in the database meets the real-time performance requirements.



(a) 4 DL Tasks (b) 8 DL Tasks

Figure 16: **Performance of different optimizer.**

We use deadline missing rate and the sum of accuracy loss ratio for all tasks (we use accuracy loss ratio to denote it at the rest of section for concise) defined in Eq. (1) as the metrics to quantify the performance of RT-mDL and the two baselines. Each baseline is repeatedly evaluated for ten times under the same task configuration. The mean and variance of the experiment results are presented in Fig. 16. The 4-task set contains two ResNet18 tasks and two

VGG19 tasks, while the 8-task set contains four VGG13 tasks and four AlexNet tasks. The deadline of each task is set randomly, and the maximum deadline missing bound of each task is set to 10% (which is the default setting for the rest of Section 6 unless stated otherwise). Results show that RT-mDL performs better than the other two baselines in terms of deadline missing rate and accuracy loss ratio under the 8-task scenario. In contrast, when there are only 4 tasks, RTA approach achieves a lower deadline missing rate while sacrificing more accuracy. This is because RTA is pessimistic and hence leads to high accuracy loss ratio. This also causes the failure of RTA approach to find a feasible solution under the 8-task scenario. As shown in Fig. 16(b), Task 1-3 all exceed the 10% deadline missing bound (i.e., 12.86%, 14.22%, 14.78%, respectively). Compared with RTA, our approach can reduce deadline missing rate by 23.48% for total eight tasks with an accuracy loss ratio of 1.15%.

6.5 Joint Model Scaling and Scheduling

To reflect the performance gain of joint model scaling and scheduling, we compare our approach with other five methods that focus on either model scaling or scheduling policy, including DM-NMC, DM-MVS, DM-MC, SPO-NMC, and UM-NMC. We generate 8 variants for each model and the absolute accuracy loss for the smallest model variant is 3.46% (AlexNet), 2.15% (VGG11), 2.74% (VGG13), 4.44% (VGG16) and 5.06% (ResNet18), respectively. Baseline DM-MVS differs from RT-mDL only in scheduling policy, i.e., it is obtained by fixing the scheduling policy to DM, and therefore the solution space is changed to model variant selection. This baseline is similar to several state-of-art approaches (e.g., NestDNN[13], MCDNN[17]) as we introduced in Section 2. Baseline SPO-NMC differs from RT-mDL only in model variant selection, i.e., it is obtained by fixing the model variant to the basic model for each task and therefore the goal is to find the best task priority assignment. To show the generality of our joint model scaling and scheduling approach, in the following sections, we will evaluate the performance of RT-mDL under different task workloads, DNN model combinations, platforms and missing rate requirements.

Impact of Different Workloads. We demonstrate the effectiveness of RT-mDL under different workloads induced by various settings of task deadlines. Specifically, we consider loose and tight deadlines for six typical DL tasks, where the deadline for each task is set to be four times of the measured average execution time under the setting of tight deadlines (i.e., heavy workload), and six times for loose deadlines (i.e., light workload). As shown in Fig. 17(a), with light workload, RT-mDL reduces the deadline missing rate of task AlexNet-2 to 6.15% compared with SPO-NMC (8.94%), UM-NMC (16.29%), DM-NMC (10.90%) with minor accuracy loss ratio (1.0%). Compared with DM-MC and DM-MVS, RT-mDL can sacrifice 74.69% and 21.63% less accuracy loss ratio respectively, while achieving comparable performance of deadline missing rate. For tight deadlines as shown in Fig. 17(b), RT-mDL still can achieve 1.87% average deadline missing rate with minor accuracy loss ratio (1.7%), while SPO-NMC, UM-NMC and DM-NMC can only achieve 27.77%, 25.28% and 41.99% respectively. RT-mDL achieves 58.32% reduction in accuracy loss ratio compared with DM-MC with comparable deadline missing rate, while DM-MVS can only achieve 46.04% reduction compared with DM-MC. RT-mDL performs better than most other methods under both loose and tight deadlines.

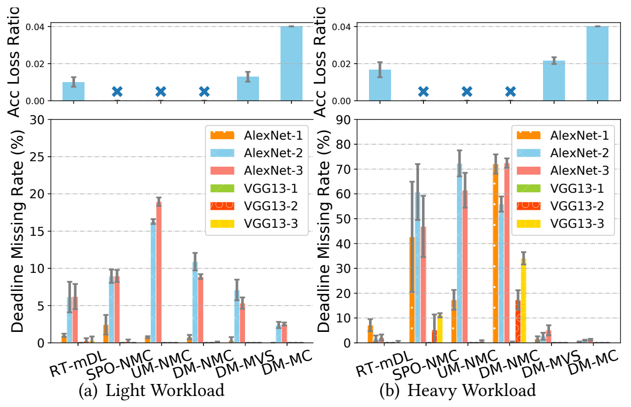


Figure 17: **Performance comparison under different task workloads ('NMC' – no model scaling, 'MC' – model scaling, 'MVS' – model variant selection, 'SPO' – scheduling policy optimization, 'UM' – Utilization Monotonic, 'DM' – Deadline Monotonic).**

DNN Model Combination. In this section, we assess the generality of RT-mDL for various model combinations and edge platforms. We choose two mixed task sets: one contains ResNet34-based sign recognition and SBCNN-based sound classification, and the other contains VGG19-based image classification and LSTM-based emotion recognition. LSTM-based DL task runs totally solely on the CPU cores, because it performs better on the CPU than on the GPU. Due to its low compressibility in the width/depth dimensions, it has only a single-level model variant. Results are shown in Fig. 18(a) and 18(b), where the histogram indicates the deadline missing rate and the line graph shows the accuracy loss ratio. Compared with DM-NMC, RT-mDL can reduce 49.50% deadline missing rate on average for SBCNN-ResNet34 task set and 10.48% for LSTM-VGG19 task set. Compared with DM-MC, RT-mDL has comparable average deadline missing rate but achieves 67.06%, 74.61% reduction in accuracy loss ratio for these two task sets respectively. The results show that RT-mDL achieves better performance over baselines across various DNN model combinations.

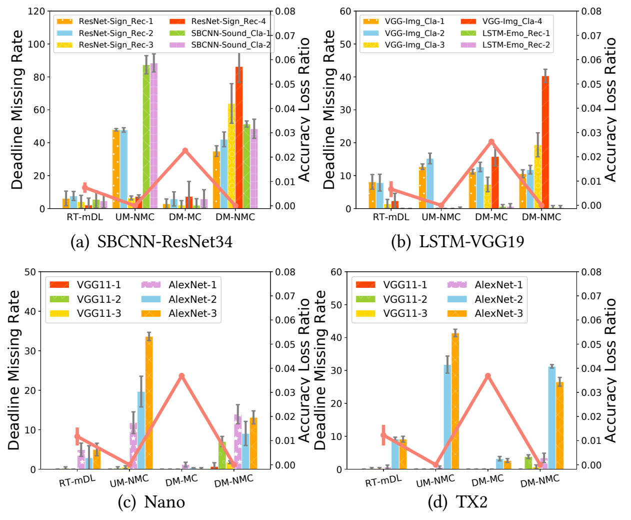


Figure 18: **Performance comparison under different DL task sets and edge platforms ('NMC' – no model scaling, 'MC' – model scaling, 'UM' – Utilization Monotonic, 'DM' – Deadline Monotonic).**

The experiments in previous sections are conducted on Xavier. In this section, we evaluate RT-mDL on low-power devices. As shown in Fig. 18(c) and Fig. 18(d), RT-mDL can reduce the deadline missing rate effectively on both edge platforms with minor accuracy loss ratio. Specifically, RT-mDL suffers a minor accuracy loss ratio 1.17% on Nano, while reducing deadline missing rate by 8.79%, 5.41% on average, compared with UM and DM scheduling algorithms respectively. On TX2, RT-mDL achieves 66.74% reduction in accuracy loss ratio compared with the maximum level model scaling (DM-MC) while achieving comparable average deadline missing rate (3.23% for RT-mDL, 0.97% for DM-MC). This result shows that RT-mDL can be effectively applied on different edge platforms.

Different Missing Rate Requirements. RT-mDL supports various settings of missing rate bounds, which is desirable for mixed-critical real-time applications [14, 48]. To evaluate RT-mDL for mixed-critical systems, we conduct experiments on task sets with five different settings of missing rate bounds. Fig. 19 shows the results, where all the five runs are under the same VGG-11-AlexNet task set and deadline setting. We set three levels of missing rate requirements: 5%, 10%, 20%, and group six DL tasks into three groups. Each task group is assigned a missing rate bound. For example, high_low_mid means that VGG11-1 and VGG11-2 are assigned missing rate bound of 20%, VGG11-3 and AlexNet-1 are assigned the missing rate bound of 5%, while the other two tasks are assigned 10%. *same_mr* represents that all tasks are equally important (i.e., with all 10% bounds). As shown in Fig. 19, RT-mDL performs satisfactorily across different settings of missing rate bounds. For example, the resultant missing rate is 16.45% under high missing rate bound, while it is 5.12%, 1.81% under medium and low missing rate bounds, respectively. This result shows that RT-mDL can adapt the assignment of task priority and selection of model variants efficiently for different mixed-critical systems.

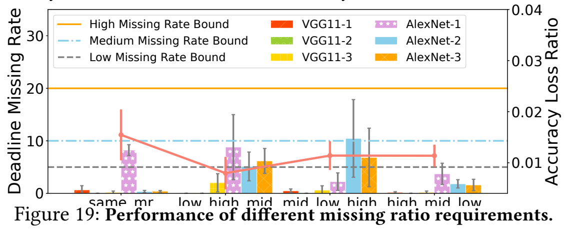


Figure 19: **Performance of different missing ratio requirements.**

6.6 System Overhead

The major system overhead of RT-mDL is caused by the priority-based DL task scheduler, since it needs to run constantly. The execution strategy optimizer of RT-mDL may also incur considerable overhead. However, it is only infrequently invoked in the iterative optimization process, which can also be offloaded to the cloud. Therefore, we focus on analyzing the CPU overhead of DL task scheduler in this section. We measure the overhead by setting the CPU affinity of our scheduler/optimizer to a single CPU core. Table 5 shows the incurred CPU overhead (measured as percentage of CPU usage) by our scheduler. The majority of scheduling overhead is due to the control of CPU/GPU sub-task execution. queuing and GPU task packing. As shown in Table 5, when the number of DL tasks increases, CPU overhead remains almost the same. Especially, due to the different processing capability of CPU cores as shown in Section 6.1, CPU overhead is different among platforms.

Table 5: **CPU Overhead of Task Scheduling**

Task Set Size	Desktop	Xavier	TX2
4 DL tasks	2.43%	6.35%	7.42%
8 DL tasks	3.78 %	7.66%	7.39%
12 DL tasks	3.27 %	7.36%	6.49%

7 DISCUSSION

Uncertain Workload. RT-mDL is designed to support periodic DL tasks, which is consistent with a wide range of applications that need to sample sensors at fixed frequencies. RT-mDL can also be extended to handle uncertain workload which may be caused by opportunistic data processing, e.g., frame filtering on low-end vision systems [15, 64]. In such a case, RT-mDL needs to monitor workload at runtime, and the vacated resources can be allocated to other tasks, e.g., via upgrading the model variants of other existing tasks. For example, important tasks with tight deadlines can be upgraded to more accurate but resource-demanding model variants.

Integration with Dynamic Scheduling. RT-mDL can be integrated with dynamic scheduling [3, 45] to assign task priorities on demand, which can allow the system to adapt to unpredictable workload. For example, we can replace the priority assignment in RT-mDL with the weight assignment for each task, and schedule tasks online based on the combination of online priority (which may be determined based on the absolute deadline) and offline optimized offset/weight. The rationale for such a design is that online priority only reflects the temporal urgency of each job, while the offline offset/weight can reflect the importance of the real-time requirements for each task.

Implications on DL framework and GPU/NPU architecture. RT-mDL can be extended to work with other DL frameworks such as TensorFlow, MXNet (discussed in Section 6.1), and AI accelerators such as NPU [20]. However, the architecture of NPU is highly vendor-dependent. Therefore, when porting RT-mDL to a new NPU architecture, the scheduling mechanism needs to be modified to take advantage of the native speedup mechanisms offered by the NPU. Lastly, the design of RT-mDL suggests that real-time performance of DL tasks would greatly benefit from fine-grained priority levels that are not yet to be supported by current DL frameworks and GPU-accelerated edge platforms.

8 CONCLUSION

In this paper, we present a new Deep Learning framework named RT-mDL, which supports running mixed real-time DL tasks on edge platforms through joint optimization of DNN model scaling and real-time scheduling. We evaluate the end-to-end performance of RT-mDL on an F1/10 autonomous driving testbed, as well as accuracy and real-time performance under different settings on task deadlines, DL task types and platforms. The experimental results indicate that RT-mDL can achieve significant performance improvement over state-of-art baselines.

ACKNOWLEDGEMENT

The work described in this article was partially supported by the Innovation and Technology Commission of Hong Kong under Grant No. GHP/126/19SZ, and National Natural Science Foundation of China under Grant No. 62032021.

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Taimoor Akhtar and Christine A Shoemaker. Multi objective optimization of computationally expensive multi-modal functions with rbf surrogates and multi-rule selection. *Journal of Global Optimization*, 64(1):17–32, 2016.

[3] Hakan Aydin, Rami Melhem, Daniel Mossé, and Pedro Mejia-Alvarez. Dynamic and aggressive scheduling techniques for power-aware real-time systems. In *Proceedings 22nd IEEE Real-Time Systems Symposium (RTSS 2001)(Cat. No. 01PR1420)*, pages 95–105. IEEE, 2001.

[4] Soroush Bateni and Cong Liu. Apnet: Approximation-aware real-time neural network. In *2018 IEEE Real-Time Systems Symposium (RTSS)*, pages 67–79. IEEE, 2018.

[5] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018.

[6] Nicola Capodieci, Roberto Cavicchioli, Marko Bertogna, and Aingara Paramakuru. Deadline-based scheduling for gpu with preemption support. In *2018 IEEE Real-Time Systems Symposium (RTSS)*, pages 119–130. IEEE, 2018.

[7] Daniel Casini, Alessandro Biondi, Geoffrey Nelissen, and Giorgio Buttazzo. Partitioned fixed-priority scheduling of parallel tasks without preemptions. In *2018 IEEE Real-Time Systems Symposium (RTSS)*, pages 421–433. IEEE, 2018.

[8] F1TENTH Community. F1tenth. https://f1tenth.org/.

[9] Xiaoliang Dai, Peizhao Zhang, Bichen Wu, Hongxu Yin, Fei Sun, Yanghan Wang, Marat Dukhan, Yunqing Hu, Yiming Wu, Yangqing Jia, et al. Chamnet: Towards efficient network design through platform-aware model adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 11398–11407, 2019.

[10] Piotr Dollár, Mannat Singh, and Ross Girshick. Fast and accurate model scaling. *arXiv preprint arXiv:2103.06877*, 2021.

[11] Sandeep D'souza and Ragunathan Rajkumar. Cycletandem: Energy-saving scheduling for real-time systems with hardware accelerators. In *2018 IEEE Real-Time Systems Symposium (RTSS)*, pages 94–106. IEEE, 2018.

[12] Xianzhi Du, Mostafa El-Khamy, Jungwon Lee, and Larry Davis. Fused dnn: A deep neural network fusion approach to fast and robust pedestrian detection. In *2017 IEEE winter conference on applications of computer vision (WACV)*, pages 953–961. IEEE, 2017.

[13] Biyi Fang, Xiao Zeng, and Mi Zhang. Nestdnn: Resource-aware multi-tenant on-device deep learning for continuous mobile vision. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, pages 115–127. ACM, 2018.

[14] Tristan Fautrel, Laurent George, Frédéric Fauberteau, and Thierry Grandpierre. An hypervisor approach for mixed critical real-time uav applications. In *2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pages 985–991. IEEE, 2019.

[15] Yue Gao, Jun-Hai Yong, and Fuhua Frank Cheng. Video shot boundary detection using frame-skipping technique. *20∬-02-i8. http://www. cs. uky. edul-chenglPUBLIPaper BD*, 1, 2011.

[16] Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28(10):2222–2232, 2016.

[17] Seungyeop Han, Haichen Shen, Matthai Philipose, Sharad Agarwal, Alec Wolman, and Arvind Krishnamurthy. Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, pages 123–136, 2016.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[19] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 784–800, 2018.

[20] Brian Hickmann, Jieasheng Chen, Michael Rotzin, Andrew Yang, Maciej Urbanski, and Sasikanth Avancha. Intel nervana neural network processor-t (nnp-t) fused floating point many-term dot product. In *2020 IEEE 27th Symposium on Computer Arithmetic (ARITH)*, pages 133–136. IEEE, 2020.

[21] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *European conference on computer vision*, pages 646–661. Springer, 2016.

[22] Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991*, 2015.

[23] Md Shahriar Iqbal, Jianhai Su, Lars Kotthoff, and Pooyan Jamshidi. Flexibo: Cost-aware multi-objective optimization of deep neural networks. *arXiv preprint arXiv:2001.06588*, 2020.

[24] Dewant Katare and Mohamed El-Sharkawy. Embedded system enabled vehicle collision detection: an ann classifier. In *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 0284–0289. IEEE, 2019.

[25] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[26] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[27] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[28] Yann LeCun et al. Lenet-5, convolutional neural networks. *URL: http://yann. lecun. com/exdb/lenet*, 20(5):14, 2015.

[29] Hongjia Li, Ning Liu, Xiaolong Ma, Sheng Lin, Shaokai Ye, Tianyun Zhang, Xue Lin, Wenyao Xu, and Yanzhi Wang. Admm-based weight pruning for real-time deep learning acceleration on mobile devices. In *Proceedings of the 2019 on Great Lakes Symposium on VLSI*, pages 501–506. ACM, 2019.

[30] Peiliang Li, Xiaozhi Chen, and Shaojie Shen. Stereo r-cnn based 3d object detection for autonomous driving. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7644–7652, 2019.

[31] Peilun Li, Guozhen Li, Zhangxi Yan, Youzeng Li, Meiqi Lu, Pengfei Xu, Yang Gu, Bing Bai, Yifei Zhang, and DiDi Chuxing. Spatio-temporal consistency and hierarchical matching for multi-target multi-camera vehicle tracking. In *CVPR Workshops*, pages 222–230, 2019.

[32] Luyang Liu, Hongyu Li, and Marco Gruteser. Edge assisted real-time object detection for mobile augmented reality. In *The 25th Annual International Conference on Mobile Computing and Networking*, pages 1–16, 2019.

[33] Ning Liu, Xiaolong Ma, Zhiyuan Xu, Yanzhi Wang, Jian Tang, and Jieping Ye. Autocompress: An automatic dnn structured pruning framework for ultra-high compression rates. In *AAAI*, pages 4876–4883, 2020.

[34] Steven R Livingstone and Frank A Russo. The ryerson audio-visual database of emotional speech and song (ravdess): A dynamic, multimodal set of facial and vocal expressions in north american english. *PloS one*, 13(5):e0196391, 2018.

[35] S Divya Meena and Agilandeeswari Loganathan. Intelligent animal detection system using sparse multi discriminative-neural network (smd-nn) to mitigate animal-vehicle collision. *Environmental Science and Pollution Research*, 27(31):39619–39634, 2020.

[36] NVIDIA. Cuda toolkit documentation. https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#context.

[37] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[38] Roger Pujol, Hamid Tabani, Leonidas Kosmidis, Enrico Mezzetti, Jaume Abella Ferrer, and Francisco J Cazorla. Generating and exploiting deep learning variants to increase heterogeneous resource utilization in the nvidia xavier. In *31st Euromicro Conference on Real-Time Systems (ECRTS 2019)*, volume 23, 2019.

[39] Dipankar Raychaudhuri, Ivan Seskar, Gil Zussman, Thanasis Korakis, Dan Kilper, Tingjun Chen, Jakub Kolodziejski, Michael Sherman, Zoran Kostic, Xiaoxiong Gu, et al. Challenge: Cosmos: A city-scale programmable testbed for experimentation with advanced wireless. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, pages 1–13, 2020.

[40] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv*, 2018.

[41] J. Salamon, C. Jacoby, and J. P. Bello. A dataset and taxonomy for urban sound research. In *22nd ACM International Conference on Multimedia (ACM-MM'14)*, pages 1041–1044, Orlando, FL, USA, Nov. 2014.

[42] Justin Salamon and Juan Pablo Bello. Deep convolutional neural networks and data augmentation for environmental sound classification. *IEEE Signal Processing Letters*, 24(3):279–283, 2017.

[43] Weijing Shi and Raj Rajkumar. Point-gnn: Graph neural network for 3d object detection in a point cloud. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1711–1719, 2020.

[44] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[45] Marco Spuri and Giorgio C Buttazzo. Efficient aperiodic service under earliest deadline scheduling. In *RTSS*, pages 2–11, 1994.

[46] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks*, (0):–, 2012.

[47] Jinghao Sun, Nan Guan, Xu Jiang, Shuangshuang Chang, Zhishan Guo, Qingxu Deng, and Wang Yi. A capacity augmentation bound for real-time constrained-deadline parallel tasks under gedf. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2200–2211, 2018.

[48] Domitian Tamas-Selicean and Paul Pop. Design optimization of mixed-criticality real-time applications on cost-constrained partitioned architectures. In *2011 IEEE 32nd Real-Time Systems Symposium*, pages 24–33. IEEE, 2011.

[49] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2820–2828, 2019.

[50] Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019.

[51] Xiaohu Tang, Shihao Han, Li Lyna Zhang, Ting Cao, and Yunxin Liu. To bridge neural network design and real-world performance: A behaviour study for neural networks. In *MLSys*, April 2021.

[52] Manni Wang, Shaohua Ding, Ting Cao, Yunxin Liu, and Fengyuan Xu. Asymo: scalable and efficient deep-learning inference on asymmetric mobile cpus. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, pages 215–228, 2021.

[53] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10734–10742, 2019.

[54] Yecheng Xiang and Hyoseung Kim. Pipelined data-parallel cpu/gpu scheduling for multi-dnn real-time inference. In *2019 IEEE Real-Time Systems Symposium (RTSS)*, pages 392–405. IEEE, 2019.

[55] Lele Xie, Tasweer Ahmad, Lianwen Jin, Yuliang Liu, and Sheng Zhang. A new cnn-based method for multi-directional car license plate detection. *IEEE Transactions on Intelligent Transportation Systems*, 19(2):507–517, 2018.

[56] Xiufeng Xie and Kyu-Han Kim. Source compression with bounded dnn perception loss for iot edge computer vision. In *The 25th Annual International Conference on Mobile Computing and Networking*, pages 1–16, 2019.

[57] Ming Yang, Tanya Amert, Kecheng Yang, Nathan Otterness, James H Anderson, F Donelson Smith, and Shige Wang. Making openvx really" real time". In *2018 IEEE Real-Time Systems Symposium (RTSS)*, pages 80–93. IEEE, 2018.

[58] Ming Yang, Shige Wang, Joshua Bakita, Thanh Vu, F Donelson Smith, James H Anderson, and Jan-Michael Frahm. Re-thinking cnn frameworks for time-sensitive autonomous-driving applications: Addressing an industrial challenge. In *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 305–317. IEEE, 2019.

[59] Shuochao Yao, Yiran Zhao, Huajie Shao, ShengZhong Liu, Dongxin Liu, Lu Su, and Tarek Abdelzaher. Fastdeepiot: Towards understanding and optimizing neural network execution time on mobile and embedded devices. In *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*, pages 278–291, 2018.

[60] Shuochao Yao, Yiran Zhao, Aston Zhang, Lu Su, and Tarek Abdelzaher. Deepiot: Compressing deep neural network structures for sensing systems with a compressor-critic framework. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*, pages 1–14, 2017.

[61] Ibrar Yaqoob, Latif U Khan, SM Ahsan Kazmi, Muhammad Imran, Nadra Guizani, and Choong Seon Hong. Autonomous driving cars in smart cities: Recent advances, requirements, and challenges. *IEEE Network*, 34(1):174–181, 2019.

[62] Juheon Yi and Youngki Lee. Heimdall: mobile gpu coordination platform for augmented reality applications. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, pages 1–14, 2020.

[63] Fotios Zantalis, Grigorios Koulouras, Sotiris Karabetsos, and Dionisis Kandris. A review of machine learning and iot in smart transportation. *Future Internet*, 11(4):94, 2019.

[64] Zhaolin Zhang, Haoshan Shi, and Shuai Wan. Dynamic frame-skipping scheme for live video encoders. In *2010 International Conference on Multimedia Technology*, pages 1–3. IEEE, 2010.

[65] Husheng Zhou, Soroush Bateni, and Cong Liu. Sˆ 3dnn: Supervised streaming and scheduling for gpu-accelerated real-time dnn workloads. In *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 190–201. IEEE, 2018.