ApNet: Approximation-aware Real-Time Neural Network

Soroush Bateni and Cong Liu
The University of Texas at Dallas
{soroush, cong}@utdallas.edu

Abstract. Modern embedded cyber-physical systems are becoming entangled with the realm of deep neural networks (DNNs) towards increased autonomy. While applying DNNs can significantly improve the accuracy in making autonomous control decisions, a significant challenge is that DNNs are designed and developed on advanced hardware (e.g., GPU clusters), and will not easily meet strict timing requirements if deployed in a resource-constrained embedded computing environment. One interesting characteristic of DNNs is approximation, which can be used to satisfy real-time requirements by reducing DNNs’ execution costs with reasonably sacrificed accuracy. In this paper, we propose ApNet, a timing-predictable runtime system that is able to guarantee deadlines of DNN workloads via efficient approximation. Rather than straightforwardly approximating DNNs, ApNet develops a DNN layer-aware approximation approach that smartly explores the trade-off between the approximation degree and the resulting execution reduction on a per-layer basis. To further reduce approximation-induced accuracy loss at runtime, ApNet explores a rather interesting observation that resource sharing and approximation can mutually supplement one another, particularly in a multi-tasking environment. We have implemented and extensively evaluated ApNet on a mix of 8 different DNN configurations on an NVIDIA Jetson TX2. Experimental results show that ApNet can guarantee timing predictability (i.e., meeting all deadlines), while incurring a reasonable accuracy loss. Moreover, accuracy can be improved by up to 8% via a resource sharing increase of 3.5x on average for overlapping DNN layers.

I. INTRODUCTION

The recent widespread use of deep neural networks (DNNs) in embedded cyber-physical systems (CPS) has enabled accurate autonomous control decision making, such as DNN-driven autonomous driving [1]–[3]. However, applying DNNs in such environments also creates significant computing challenges. While DNNs are conventionally designed and developed on advanced hardware such as GPU clusters, they are now required to run on embedded hardware that imposes stringent size, weight, and power (SWaP) constraints. A major challenge is thus to meet timing requirements, which are natural to many CPS for making control decisions in real-time, while enjoying the power brought by DNN-driven approaches.

Recent research has found the hardware adopted in many embedded systems to be insufficient for running DNN workloads [4], [5]. An example is autonomous driving vehicles. Autonomous vehicles rely on multiple cameras and sensors, most of which require a DNN to process environmental information in order to produce control signals. The computational constraints of current purpose-built hardware are evident in this scenario [6]. Even with state-of-the-art computing hardware (e.g., Nvidia Drive PX2), autonomous vehicles can fail to detect obstacles in a real-time fashion partly due to the overwhelming DNN-induced computational demands [7], [8] (also see our measurements-based case studies in Sec. III).

One possible remedy to this real-time computing challenge is approximation. Approximation has been a recent hot topic in artificial intelligence and machine learning literature [9]–[13]. A set of efficient and practical (w.r.t. acceptable accuracy loss) DNN approximation methods have been proposed, which seek to compress and compact a DNN for a faster execution. To guarantee timing predictability in a DNN-driven embedded system, an intuitive idea is thus to appropriately apply approximation to DNN instances considering the corresponding deadlines. Nonetheless, the current literature does not address this specific problem. Existing solutions on exploring approximation and latency tradeoffs [14]–[16] seek to improve latency on a best-effort basis through approximation and mostly targets at general purpose workloads, thus not encompassing of the characteristics and timing requirements of the problem studied in this paper.

First and foremost, unlike many general purpose workloads, DNNs are built from multiple layers, each of which may exhibit rather divergent characteristics in terms of execution demands and approximation potential. Any layer-oblivious approximation strategy aiming at timing predictability may result in pessimistic accuracy loss, e.g., those treating a DNN as a single task entity. Our key observation is that runtime response of each layer of a DNN instance to approximation may be quite different. This motivates us to efficiently perform per-layer approximation considering specific characteristics of each layer to meet deadlines. Second, DNNs are usually run on general purpose graphics processing units (GPGPUs). In a multi-tasking DNN scenario, multiple DNN instances could potentially share the GPU to improve response time and resource utilization. Resource sharing enables concurrency, which may significantly improve resource utilization and thus latency. Approximated DNN layers have different resource utilization characteristics from their original configurations. Thus, approximation can potentially affect resource utilization and may lead to improved resource sharing benefits. On the other hand, improved resource sharing can result in shorter response time, which may relax timing requirements on a task
and thus further result in fewer approximations required to meet a deadline. This interesting observation on the mutually supplementary relation between resource sharing and approximation is critical to any resource-demanding multi-tasking environment, which has unfortunately never been explored in the literature.

**Contributions.** In this paper, we propose ApNet, a timing-predictable runtime system that guarantees hard deadlines of DNN workloads via efficient approximation. ApNet is fundamentally built upon theoretical analysis of a multi-layer DNN end-to-end scheduling framework. As we will motivate in Sec. IV-A, traditional sub-deadline assignment approaches fail to address the characteristics of the problem. We thus design a layer-aware sub-deadline assignment policy to include approximation potential, a measurement of effectiveness of approximation on a per-layer basis. Subsequently, we propose a formulation that can assign individual approximation requirements to layers based on their approximation potential. This formulation takes into consideration the real-world limitations of DNN approximation as well, as we shall see in Sec. IV-A.

Furthermore, we design and implement a runtime system based on our theoretical analysis for enhanced runtime multi-tasking performance, which exploits the mutually supplementary relation between approximation and resource sharing. One major drawback of naively enabling concurrency through resource sharing is the fact that different DNNs might hardly overlap since their cumulative resource usage exceeds the capacity limits of GPU. An interesting observation that motivated us to develop a runtime solution was that different approximated configurations of a DNN have different resource utilization profiles. In Sec. IV-B, we explore how ApNet can improve resource sharing and concurrency via more approximation. On the other hand, resource sharing enables higher resource utilization which can relax the approximation requirements on certain layers. Our runtime system explores this trade-off and is shown to significantly improve performance and reduce the overall degree of approximation to meet deadlines.

We have fully implemented and evaluated ApNet on a driving-specific embedded platform, the NVIDIA Jetson TX2, on popular DNN frameworks including AlexNet and CaffeNet and 6 compatible approximated configurations, under both single-tasking and multi-tasking environments. Extensive experimental results show that ApNet is able to guarantee deadlines both in single-tasking and multi-tasking scenarios. Furthermore, ApNet is able to improve utilization by up to 3.5x and decrease average turnaround time (ATT) by 60% on average, while improving accuracy by an average of 1% compared to a best effort approach.

**II. BACKGROUND AND SYSTEM MODEL**

**GPU background.** In this paper, we mainly target GPUs as the primary computing resource. This is because our target DNN-driven embedded systems such as an autonomous driving platform mainly rely on GPU to perform the majority of compute-intensive tasks including DNN workloads [17], [18]. GPU consists of thousands of simple computing cores capable of a few naive operations such as add. The critical limitation of current GPU implementations is that while GPU is technically a multi-core processor (multiple Streaming Multiprocessors (SMs)), this structure is not exposed to the operating system or the programmer. Thus, while GPU is capable of running multiple tasks at once, it is not configured to do so by default. Developers can get around this limitation by explicitly enabling concurrency, a method we use in this paper as well. The scheduling will nonetheless be done at hardware-level.

**DNN background.** Our target real-time workloads are DNNs, which are constructed from the conceptual building block of a single neuron, as illustrated in Fig. 1(a). Multiple neurons construct a layer. A DNN at the very least consists of an input layer and an output layer, along with multiple hidden layers that are positioned in between. Fig. 1(b) depicts a sample DNN construct. Layers are ultimately represented by a thresholded weighted sum of the input set of that neuron. The output is passed on to the next layer, or is considered the ultimate answer. The weights in the weighted sum are "learned" in a training phase. In this paper, we do not consider the training phase since our target platform rarely requires training at runtime. The configuration of a DNN is a combination of the learned weights and the number and construct of the DNN layers.

**DNN approximation.** Much recent research has been done to enable approximation in DNN [9]–[12], [14], [15], which sacrifices an acceptable degree of accuracy loss for reduced execution cost. The resulting approximated DNN configurations have been proved to exhibit sufficient accuracy in many practical settings [9], [14]. The three DNN approximation techniques that are the most conventional in literature are compression, weight pruning, and matrix factorization. Compression reduces layer storage overhead by using compression techniques such as Huffman coding to store variables. Weight pruning involves removing variables that for example fall below a pre-set threshold, and thus reduce the computation overhead of that layer accordingly. Finally, matrix factorization divides the variable space into two matrices that are much smaller. This division is done through a decomposition technique that turns a matrix $C$ into $A$ and $B$ such that $AB \approx C$. This technique reduces storage and computation overhead at the same time. For example, Lowrank is a matrix factorization approximation, while CompressNet is a compression-based network configuration. Most modern configurations contain a
method of self-pruning at training time as well.

**Formal system model.** Since ApNet includes an analytical framework for guaranteeing deadlines, we present a formal system model. A formal presentation of a DNN task system is one consisting of $n$ DNN tasks $\tau = \tau_1, \ldots, \tau_n$. Each task $\tau_i$ consists of $z_i$ layers $\tau_{i1}, \ldots, \tau_{iz_i}$. Our focused workload in this paper is recurrent DNN-based processing on images continuously captured by cameras. Thus, each DNN task can be presented as a periodic task with a period of $T_i$ (e.g., a 33ms period corresponding to a 30 FPS camera). Let $\tau_{ik}$ denote the $j^{th}$ job of the $k^{th}$ layer of DNN task $\tau_i$, and $C^k_i$ denote the execution cost of $\tau_{ik}$.

Our primary focus of the processing unit is GPU, which has a first in first out (FIFO) non-preemptive scheduler. For such a system to be schedulable, it suffices that

$$\forall \tau_i \in \tau : \sum_{k=1}^{z_i} C^k_i \leq T_i,$$

in which $C_i$ is the total execution cost of $\tau_i$ (i.e., $C_i = \sum_{k=1}^{z_i} C^k_i$). $T_i$ can be viewed as a natural deadline for each task since the processing of a video frame needs to be completed before processing the next frame. However, we require a more stringent deadline ($D_i \leq T_i$). $D_i$ is defined as the relative deadline, i.e., the absolute deadline minus the release time of a job released by $\tau_i$. We will discuss this further in Sec. IV-A.

### III. Motivation

In this section, we lay out our measurement-based case studies and the obtained observations that have led us to design ApNet. Our testbed is an NVIDIA Jetson TX2 platform, which is very similar to NVIDIA’s driving platforms such as PX2 used in Tesla and Volvo automobiles [19].

**Deadline misses for running DNNs.** There have been studies before that have explored the area of DNNs in the context of real-time constraints [14]. Our previous research [20] was initiated based on a promise of better DNN responsiveness via smart scheduling. Fig. 2 depicts our observations regarding the state of the art DNN execution. In this set of experiments, we set up 5 major DNN configurations (AlexNet, CaffeNet, GoogleNet, ResNet, and VGGNet) on our TX2 platform and measure execution times. Fig. 2 depicts the cumulative distribution function for the execution times of each configuration. Fig. 2 also includes a tentative deadline, annotated by D (12ms) on the x-axis. This deadline is chosen based on the fact that modern cars need to execute multiple DNN instances because they include multiple cameras and perform control operations before a 33ms deadline, to allow for an acceptable response time to an outside event. A 12ms deadline is shown to be a generous deadline for a typical 4/8 camera autonomous vehicle [21], [22].

As is evident in the figure, all current configurations miss the deadline. Simple configurations such as AlexNet would barely miss the deadline. However, more advanced configurations with higher accuracies such as ResNet will miss the deadline by a wide margin.

**Per-layer approximation characteristics.** An area that is rather interesting to explore in order to improve latency performance and avoid deadline misses is approximation. Via approximation, embedded system hardware with stringent SWaP constraints such as Jetson TX2 might be able to process more computation in real time. The system should be able to trade accuracy and responsiveness given the right parameters. To understand the approximation characteristics of DNNs, we choose to run the popular DNN framework AlexNet and its two approximated configurations, DeepCompression and Lowrank which are shown to be efficiently practical [12], [23]. The measured results are shown in Table I. The first column depicts layer numbers. AlexNet and its derivatives have 24 layers. This table is summarized to show only 5 of these layers. The second to fourth column depict the average execution for each layer. Each DNN instance is ran for 1000 iterations to calculate the per-layer average execution time. Finally, the fifth column depicts the resource utilization for AlexNet(A), DeepCompression(D), and Lowrank(L). The resource utilization is calculated by measuring the SM usage. The last row includes total times for each configuration.

As is evident from Table I, different degrees of approximation may result in various improvements on runtime execution time. More importantly, such improvements are not uniform among layers. For example, applying lowrank, which is a 5% accuracy loss, will yield an 80% improvement for layer $\tau^1$, but will yield a 55% improvement for layer $\tau^{10}$. We note that DeepCompression provides improvements in storage and memory requirements, which may not be reflected in execution time. Our key observation is that various layers of a DNN model exhibit rather different approximation potential, i.e., the runtime response of each layer of a DNN model to approximation may be quite different. This motivates us to explore a layer-aware design that considers per-layer approximation potential for efficiently applying approximation to meet deadlines.

### TABLE I: Runtime and average resource utilization for example layers of three DNN workloads.

<table>
<thead>
<tr>
<th>Layer</th>
<th>AlexNet</th>
<th>DeepCompression</th>
<th>Lowrank</th>
<th>Utilization (A/D/L)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>294</td>
<td>289</td>
<td>37</td>
<td>99% / 99% / 97%</td>
</tr>
<tr>
<td>2</td>
<td>294</td>
<td>289</td>
<td>37</td>
<td>99% / 99% / 97%</td>
</tr>
<tr>
<td>10</td>
<td>119</td>
<td>100</td>
<td>53</td>
<td>99% / 99% / 86%</td>
</tr>
<tr>
<td>20</td>
<td>36</td>
<td>53</td>
<td>46</td>
<td>50% / 50% / 15%</td>
</tr>
<tr>
<td>23</td>
<td>1072</td>
<td>1087</td>
<td>50</td>
<td>20% / 20% / 2%</td>
</tr>
<tr>
<td>Total</td>
<td>13652</td>
<td>13757</td>
<td>10689</td>
<td></td>
</tr>
</tbody>
</table>
Relation between resource sharing and approximation. Recent research has shown that the under-utilization issue seen in GPGPU represent a major performance bottleneck [20], [24]. An efficient approach to improve resource utilization is through resource sharing, i.e., allowing the GPU to run multiple workloads concurrently.

Our interesting observation that relates approximation to resource sharing is that different approximation techniques result in different GPU resource utilizations. The fifth column in Table I depicts the utilization of each layer according to the configuration. We note that this utilization is derived from GPU execution cycle and pipeline occupation. In a real-world implementation, many more parameters such as register occupancy and memory bandwidth saturation also need to be accounted for. As is depicted in Table I, the utilization changes between approximated configurations and the default AlexNet, and such change is also not uniform. Moreover, from a pure resource-utilization standpoint, layers of Lowrank have a higher potential for concurrency since they leave headroom for another task to share the resource at the same time. Intuitively, approximation has the potential of improving resource sharing under a multi-tasking scenario, since it would be more likely to concurrently execute multiple approximated layers with relatively lower utilization (compared to their original configurations) on GPU hardware. This observation motivates us to design a runtime system that explores the relation between resource sharing and approximation under multi-tasking scenarios.

IV. DESIGN

Given an unschedulable DNN system, the goal of ApNet is to compromise on accuracy to make the DNN system schedulable. We first present a theoretical analysis on the method of ApNet and how it would guarantee meeting deadlines given any deadline. Furthermore, we present a novel runtime solution that minimizes accuracy loss while still guaranteeing schedulability, particularly in a multi-tasking scenario, through exploring the interesting tradeoff space between resource sharing and approximated DNN execution.

A. Guaranteeing Schedulability via Approximation

As discussed in Sec. III, running even a single DNN task in an embedded system may easily miss the deadline, particularly if the adopted DNN model is complicated. To provide better latency guarantee, recent research has shown that DNNs have specific characteristics in that they can be approximated to various degrees on the fly [15], which improves latency performance at the cost of reasonably reduced accuracy. Moreover, as discussed in Sec. II, DNNs do not have a single variation of approximation as layers exhibit different approximation potential. The design of ApNet is motivated by observing these inherent characteristics. Any layer-oblivious design would not be practically efficient in neither guaranteeing deadlines nor minimizing accuracy loss due to approximation. We thus propose to apply approximation on a per-layer basis to reduce the resulting accuracy loss while meeting deadlines (as motivated in Sec. III).

1) A Layer-aware End-to-End Scheduling Approach Considering Per-Layer Approximation Potential: Our approach is fundamentally based upon the classical real-time end-to-end scheduling [25]. Specifically, for a task consisting of a chain of subtasks, the end-to-end deadline of the task is divided into sub-deadlines each of which is assigned to a subtask. Meeting all sub-deadlines of subtasks guarantees meeting the end-to-end deadline. For the DNN task system, each DNN task with an end-to-end deadline can be viewed as a chain of layers. Under this methodology, the most important question is how to assign sub-deadlines for layers. The sub-deadline assigned to each layer is the most critical factor impacting the needed approximation applied to the layer.

Unfortunately, traditional sub-deadline assignment policies may not work well in this problem context. For instance, proportional deadline (PD) assignment has been shown to be rather effective for the real-time sporadic pipeline task model [26]. Under the PD policy, given a relative end-to-end deadline \( D_i \) for the entire DNN task, PD generates sub-deadlines \((Vd_i)\) for a layer \( k \) such that:

\[
Vd^k_i = D \cdot \frac{C^k_i}{\sum_{j=1}^{z_i} C^j_i},
\]

in which \( z_i \) is the number of sub-layers and \( C^j_i \) is the WCET for layer \( j \). Obviously, PD only considers the execution time and deadline parameters, yet does not take into account the critical approximation characteristics of a layer. Before showing an example to show why this would be problematic, we need to formally represent each layers approximation potential:

**Definition 1: Approximation Potential \((\alpha)\).** Approximation Potential \((\alpha)\) is a measurement for a layer’s ability to respond to approximation. The approximation potential of layer \( k \) of task \( \tau_i \), denoted \( \alpha^k_i \), is defined as:

\[
\alpha^k_i = \min_{\text{all configurations}} \frac{\Delta C^k_i(\%)}{\text{Approx}(\%)}.
\]

in which

\[
\Delta C^k_i(\%) = \frac{C^k_i - C_{\text{baseline}}}{C_{\text{baseline}}}
\]

According to the above equation, each unit of \( \alpha^k_i \) for layer \( k \) indicates that one percentage of approximation can reduce the worst case execution time by \( \alpha^k_i \%). Specifically, for each configuration in the configuration pool, and for each layer \( k \), WCET is recorded. \( \Delta \text{WCET} \) is then calculated as the difference between this WCET and the WCET of the baseline (i.e., the configuration without any approximation) for that layer. \( \Delta \text{WCET} \) is finally divided by the WCET of the baseline to provide percentage based (normalized) execution time difference in the form of \( \Delta \text{WCET} \% \). \( \alpha^k_i \) for that configuration is then calculated by dividing this \( \Delta \text{WCET} \% \) by the reported accuracy loss for that configuration denoted by \( \text{Approx}(\%) \). Finally, the minimum value for layer \( k \) among all configurations is used as the value of \( \alpha^k_i \).

For example, an \( \alpha^k_i \) of 5 indicates that if this layer was approximated by 1%, you would expect the WCET to be adjusted by 5% of the original value \((C'_{\text{new}} = C_{\text{old}} - 0.05 \cdot C_{\text{old}})\).
TABLE II: The value of $\alpha$ under various input sizes under LowRank.

<table>
<thead>
<tr>
<th></th>
<th>256-I</th>
<th>256-K</th>
<th>512-I</th>
<th>512-K</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau^2$</td>
<td>16.55</td>
<td>16.90</td>
<td>17.06</td>
<td>16.96</td>
</tr>
<tr>
<td>$\tau^5$</td>
<td>12.29</td>
<td>10.13</td>
<td>11.89</td>
<td>11.75</td>
</tr>
<tr>
<td>$\tau^{12}$</td>
<td>16.025</td>
<td>14.23</td>
<td>15.7</td>
<td>14.95</td>
</tr>
<tr>
<td>$\tau^{18}$</td>
<td>10.80</td>
<td>13.7</td>
<td>10.80</td>
<td>10.804</td>
</tr>
<tr>
<td>$\tau^{23}$</td>
<td>9.38</td>
<td>9.27</td>
<td>9.33</td>
<td>9.28</td>
</tr>
</tbody>
</table>

A 2% approximation would adjust WCET by 10% and so on. A question might arise about how to obtain $\alpha$ in a real-world application. To answer that, first and foremost, we note that neural networks are designed and trained first on an image set that is carefully selected for the target application of the DNN. Then, the DNN is initially tested on a test set, usually put aside from the original training data. If the DNN correctly recognizes the test set, it can be deployed. The value of $\alpha$ is thus calculated at the testing phase. However, at deployment stage, the DNN can incur multiple inputs with different characteristics. Therefore, an interesting question would be whether $\alpha$ is accurate regardless of future input. The intuitive answer to this question is that in most DNN configurations, the input size is determined as part of the training phase. Thus, either the input at the testing and deployment phase has to be resized prior to execution, or the DNN itself includes a preliminary cropping layer. Moreover, the primary factor in performance characteristics are weight and coefficient width (e.g. 8-bit integers per color channel) and layer dimensions and number of layers, all of which are determined at training time as well. Nonetheless, we were still curious as to how much variation can input data cause in a DNN in terms of performance and approximation characteristic. Table II depicts an example set for $\alpha$ using LowRank with different input sizes. The setup for this experiment will be discussed in Sec. V. We chose a set of images for each size (256x256 depicted as 256- and 512x512 depicted as 512-) from the test dataset of ILSVRC2012 (the same set LowRank has been trained on) and a set of images from the KITTI benchmark [27]. We calculated $\alpha$ according to Equ. 3. The percentage of approximation reported for LowRank is 5%. As is evident in the table, $\alpha$ remains almost constant across input sizes, but varies between layers. This is an expected behavior, since different inputs will still have the same width in calculation and the value of each pixel will not have a significant effect (e.g. all pixels are 8-bit integers per color channel anyway). Our future work will include a mechanism to include a finer-grained adjustment of approximation potential, which the complexity alone requires a separate report altogether.

Now to see why Equ. 2 would be problematic, imagine a two layer DNN configuration as depicted in Fig. 3a. The first layer of this DNN has a $C$ of 8, while the second layer has a $C$ of 12. Assume that layer 1 has an $\alpha$ of 1 and layer 2 has an $\alpha$ of 2. This arrangement of $\alpha$ is not unusual as the larger the layer execution time is, the higher the approximation potential usually is. According to Equ. 2, the required deadlines are calculated to be 6 and 15. According to PD and $\alpha$, layer 1 needs to be approximated by 25% to accommodate this deadline. Layer 2 would need a 12.5% approximation. However, this is rather counter-intuitive to each layer’s characteristic. Even though layer 2 has a higher approximation potential, and thus can gain more reduction in WCET, it has a lower approximation requirement compared to layer 1. This shortcoming arises because Equ. 2 does not take into account a layer’s approximation characteristic for sub-deadline assignment.

Motivated by this, ApNet develops a sub-deadline assignment policy that considers the approximation characteristics on a per-layer basis. To account for $\alpha$, we use a variation of PD called normalized PD (NPD) as an inspiration. The layer’s execution time is weighted against $\frac{1}{\alpha}$ so that the larger the $\alpha$, the tighter the sub-deadline assignment would be for this layer.

$$Vd^k_i = D_i \sum_{j=1}^{2^k_i} C^k_i \cdot \frac{1}{\alpha^k_i} - D_i$$

in which $D_i$ is the relative deadline of task $\tau_i$. In an interference-free situation, $D_i$ will be calculated as the difference between the absolute deadline and the original release time. However, in the case that interference exists from other tasks, as we shall discuss in Sec. IV-B, the deadline will be calculated as absolute deadline minus the actual release time (when a job starts execution) to account for interference. According to Eq. 4, the higher the approximation potential of a layer, the smaller the sub-deadline assigned to this layer is expected to be. For the same example of Fig. 3 with $\alpha$ values of 1 and 2, the relative deadlines would come out to 8.57 for layer 1 and 6.43 for layer 2. As the WCET for layer 1 is 8, the deadlines for layers 1 and 2 can be set to 8 and 15 respectively, as is shown in Fig. 3c.

2) Applying approximation to Meet Sub-deadlines: While Equ. 4 can help with assigning sub-deadlines, it cannot guarantee schedulability as $C^k_i > Vd^k_i$ may hold for certain layers. Thus, to meet each sub-deadline, approximation might be required. As discussed in Sec. II, different layers can have various degrees of approximation, which we define using $X^k$:

**Definition 2:** $X^k_i$. For a given layer $\tau^k_i$ of DNN task $\tau_i$, the required approximation to meet its sub-deadline $Vd^k_i$ is denoted by $X^k_i$.

$X^k_i = 0\%$ implies no approximation while $X^k_i = 100\%$ indicates that the layer needs to be skipped. Note that each layer may have a fixed number of approximation configurations in practice. We will discuss shortly how to identify $X^k_i$ in a practical setting.

If sub-deadlines are given using Equ. 4, it is relatively straightforward to derive $X^k_i$ using $\alpha^k_i$. This can be done by using $X^k_i \cdot \alpha^k_i$ as an adjustment knob for the execution cost $C^k_i$. The following employs the adjusted execution cost with an approximation value of $X^k_i$:

$$Vd^k_i = C^k_i - C^k_i \cdot X^k_i \alpha^k_i \implies X^k_i = \frac{C^k_i - Vd^k_i}{C^k_i \alpha^k_i},$$

with $X^k_i$ being the unknown to be solved. What is notable here
Fig. 3: A motivational example illustrating the influence of approximation potential on sub-deadline assignment.

is that Equ. 4 serves an opposite purpose to Equ. 5 in that the former generates sub-deadlines, while the latter requires sub-deadlines as input.

To clarify, we bring up our initial assumption: the DNN task system will miss a deadline. Therefore, the deadlines generated by Equ. 4 will not be sufficient on their own since for some $\tau_i$, $\sum C^k_i > \sum Vd^k_i$. The next step requires putting each generated sub-deadline into Equ. 5 and deriving $X^k_i$, the required approximation degree for layer $\tau^k_i$.

For the example of Fig. 3, using Equ. 5, the required approximations are depicted on the layers themselves. For example, for the combination of Equ. 4 and Equ. 5 depicted in Fig. 3(c), the first and second layer will yield a 0% and 20% approximation, respectively. We note that compared to Fig. 3(b) which did not assign sub-deadlines by considering approximation potential, just by increasing the second layer’s approximation by 7.5%, we avoid a 25% approximation for the first layer entirely.

3) Formal Analysis: So far, the assumption has been that the approximation degree of any layer $\tau^k_i$ (i.e., the value of $X^k_i$) can be arbitrary. This assumption is not true in real-world. DNNs are usually constructed from meticulously crafted layer configurations that are refined over the past decade. Even state of the art approximation techniques are applied with the intention of minimizing accuracy loss while maximizing execution cost reduction. Thus, for an effective system that uses established DNN configurations, there needs to be considerations so that Equ. 5 can point to valid approximation configurations. One such consideration can be summarized as a cap on approximation. Depending on the type of layer and the existing approximation techniques, there could be a certain accuracy loss for a layer before the results would become unacceptable. In such a scenario, the naive solution would be to skip that layer entirely. For example, imagine a cap of 15% exists for layer 2. If the required approximation is 20%, then the layer must be skipped in order to meet the deadline. This is not desirable since skipping a layer would almost always make the output result unusable. Thus, Equ. 5 needs to be updated to account for this cap, denoted by $cap^k_i$ for layer $\tau^k_i$.

$$ Vd^k_i = C^k_i - C^k_i \cdot \min\{X^k_i, cap^k_i\} \cdot \alpha^k_i. $$

(6)

For example, consider a set of real-world DNN configurations as depicted in Table IV (which is explained in detail as part of the evaluation setup in Sec. V). Column 3 shows the aggregate accuracy loss from each configuration. This means that if every layer from this configuration was to be chosen, the accuracy loss compared to the baseline would equal to the given value. The $cap^k_i$ in the case of Table IV, will be calculated according to BNN which represents the lowest accuracy configuration in the pool, since choosing that configuration will cause the maximum aggregate loss without skipping a layer entirely. The $cap^k_i$ will thus be determined by the lowest accuracy configuration in the configuration pool.

Another consideration would be whether or not it is possible to calculate the exact accuracy loss for the final output of our method. With the current state of DNN research, it is exponentially expensive to calculate the per layer accuracy loss in all DNN mutations. To understand this complexity, imagine a 24 layer base configuration with 6 compatible networks. Therefore, each layer can be replaced with 6 other layers from a different configuration, thus bringing the per-layer choice for a possible solution to 7. With 7 choices for each layer, the total number of mutations will be $7^{24}$. Each configuration will be inherently unique in terms of accuracy loss due to the complex nature of DNNs. Therefore, each one of the $7^{24}$ mutations will need to be individually tested to calculate the top-5 and top-1 error rate (an operation that takes days, if not weeks for each configuration) and compare it to the baseline to generate an accuracy loss. The effect on accuracy of switching a layer to another layer from a different configuration can then be exactly inferred by comparing the accuracy loss of one mutation to the other. To avoid this astronomical complexity, we thus develop a scoring method to estimate the per-layer impact on accuracy, which is depicted in column 4 of Table IV. This score is given to each layer based on the aggressiveness of the approximation method used in the configuration. For example, in the example of Table IV, BNN replaces each layer with a binarized version, which results in an aggressive accuracy loss. A low score of $−1$ indicates that using a layer from this configuration should be avoided at all cost. The scoring method is thus relative and acts as a weight for our method to prefer some configurations to others based on their total accuracy loss. We have extensively evaluated this scoring method in Sec. V, and the overall result is a hard guarantee on deadlines, but a best effort approach to accuracy loss. We next prove this deadline guarantee and later show the effectiveness of our method in Sec. V.

Property 1: A DNN is considered weakly schedulable if there exists a composition of $X_i = \{X^k_i| k \in z_i\}$ for each layer $\tau^k_i$ such that $\sum_{k \in z_i} C^k_i - C^k_i \cdot \min\{X^k_i , cap^k_i\} \cdot \alpha^k_i \leq D_i$.

However, this state is not desirable since it will result in the maximum accuracy loss. We would like to minimize accuracy loss as much as possible. For the example system shown in Fig. 1, the $X^k_i$ values would be derived as shown in Fig. 3d, giving layer 1 an approximation degree of 5% and layer 2 an approximation degree of 15%.

Finally, we prove the fact that Equ. 4 and Equ. 6 can generate a schedulable system under any deadline. Property 1 shows this intuitively. However, if Property 1 does not hold, we need to assume that $X^k_i$ is an arbitrary number that can go up to 100% for each layer. At that point, by assumption, the value of $C^k_i$ and $C^k_i \cdot X^k_i \alpha^k_i$ become equal. In other words, the
layer is skipped altogether, which is sacrificing 100% accuracy for that layer. We now prove the schedulability of the system with and without Property 1.

**Lemma 1:** Equ. 4 and Equ. 6 can generate a schedule for any DNN task $\tau_i$ with any deadline $D_i$.

**Proof:** A DNN task $\tau_i^k$ is unschedulable if $\sum C_i^k > D_i$. Imagine a sub-deadline assignment has been made via Equ. 4, which is $V d_i^1, ..., V d_i^z$. If the system misses the deadline, it means that there exists at least one task such that:

$$\exists \tau_i^k \in \tau_i : C_i^k > V d_i^k.$$  

This entails that for $k$, there exists a constant $H$ such that $H_i^k = C_i^k - V d_i^k$. Thus, in order to ensure that the system is schedulable, it is imperative that for each $k$ that the above equation holds true, we have:

$$0 = C_i^k - C_i^k \cdot X_i^k \alpha_i^k - V d_i^k.$$  

Replacing $C_i^k - V d_i^k$ with $H_i^k$ will yield

$$0 = H_i^k - C_i^k \cdot X_i^k \alpha_i^k \Rightarrow H_i^k = C_i^k \cdot X_i^k \alpha_i^k.$$  

Thus, as long as $H = C_i^k \cdot X_i^k \alpha_i^k$ for each $k$, and positive $H$, the system is schedulable. Since the maximum value for $H$ is $C_i^k$, for a system that can have an $X_i^k$ as high as $100\% (1)$, this equality can always hold true. On the other hand, for a weakly schedulable system, we have

$$\sum_{k \in z_i} C_i^k - C_i^k \cdot \min\{X_i^k, cap_i^k\} \cdot \alpha_i^k \leq D_i.$$  

Replacing $H$ in this equation will yield:

$$\sum_{k \in z_i} C_i^k - H_i^k \leq D_i.$$  

Thus, a weakly schedulable system guarantees that an $H_i^k$ always exists without requiring any further assumptions.

**Practical Considerations.** There are two last hurdles for our theoretical analysis to be fully encompassing of a real-world implementation of a real-time system containing DNNs. First and foremost, the three approximation methods discussed in Sec. II (i.e. compression, pruning and factorization) are drop-in methods. This means that for a given DNN configuration, such as CaffeNet, each layer can (depending on the type of layer) be easily swapped with an approximated version of the same layer. Thus, a layer-based scheduler can be flexible in adjusting approximation at runtime to accommodate for secondary measures such as resource-sharing. However, approximation in DNNs can be more complex than that. For example, AlexNet itself is a slightly more accurate version of CaffeNet. Moreover, while CaffeNet and AlexNet have 24 layers ($n = 24$), newer configurations such as ResNet [28] can have more than 150 layers. It is not possible to swap a layer of CaffeNet with a layer of ResNet because they are fundamentally incompatible. Nonetheless, ResNet does in fact improve upon accuracy substantially when compared to CaffeNet, but has the drawback of a considerably more overhead. This complexity, which we call cross-layer approximation, compatibility does not matter in our theoretical analysis since $z_i$ could be assigned to be the largest value possible. The $C_i^k$ for layers that are non-existent in some configurations is then zero. This can be effectively dealt with at implementation level. While applying Equ. 4, we need to make sure that choosing a configuration does not break compatibility with other layers so far.

Second, for our formulas, we did not consider the fact that a stable system has to use already established approximation configurations. This is due to the fact that approximation in DNNs are tricky and require weeks of tweaking and evaluation. Moreover, calculating accuracy is done through a train-test cycle on an established image database (e.g. ImageNet [29]) that would take a long time. A system that dynamically switches configurations per layer will thus be impossible to assess. Imagine ApNet decides to use a layer from Lowrank for $\tau_1$ and a layer from DeepCompression for $\tau_2$. This new configuration needs to be meticulously tested for accuracy measurements. Each time such a decision is made, this has to be redone. Thus, calculating accuracies for every possible runtime configuration would be an astronomical task. We use a combination of two methods to remedy this. First and foremost, while the value of $X_i^k$ is continuous, it will be converted to the next highest available discrete configuration at runtime, using a table similar to Table III. We also use the aforementioned scoring method to calculate an accuracy score based on the chosen configurations per-layer. A system will be weakly schedulable in this case if $X_i^k = \{\max_{\epsilon \in T_i^k} \{X_i^k, \epsilon}\}$, in which $\epsilon$ is the contents of Table III ($T_i^k$) for a given layer. This means that with the most approximation for each layer, the system has to be schedulable. We will elaborate on this in Sec. V.

**An Optimal yet Inpractical MILP Solution.** The end-to-end scheduling solution presented in Sec. IV-A for meeting DNN tasks’ deadlines through per-layer approximation provides an efficient runtime solution for any practical settings. But clearly it does not provide any optimality in terms of minimizing the required overall approximation.

Such an optimal solution of the problem can be described as a mixed integer linear programming (MILP) problem. Imagine that a table such as the examples in Table III exists for each layer. For a layer $\tau_i^k$, suppose there are $Q_k$ entries. The goal is to solve:

|TABLE III: Accuracy-Utilization table for layer 1 and 2 |
|---|---|---|
|Configuration| Accuracy Score ($X$)| Configuration| Utilization|
|CaffeNet| 1| 1| 90%|
|Lowrank| 0.5| 2| 75%|
|Configuration| Accuracy Score ($X$)| Configuration| Utilization|
|CaffeNet| 1| 1| 90%|
|Lowrank| 0.5| 2| 70%|

(a) $T_i^1$ (Layer 1)

(b) $T_i^2$ (Layer 2)
Moreover, a tighter deadline for a multi-DNN scenario is not a problem since the approximation values could be adjusted. Consider the example presented in Fig. 4(a). The system has two DNNs depicted with different colors, both expected to finish by the absolute deadline of 5. Moreover, each DNN has 3 layers, each with their respective resource utilization depicted on the y-axis. As we have presented in our previous study [20], the resource utilization progressively gets smaller as the layer depth increases. In this case, the first DNN can run with no approximation at all, while the second DNN requires some approximation. This transition happens naturally since Equ. 4 can work with relative deadlines. The relative deadline of the first DNN is 5, which it has ample time to finish. However, due to the non-preemptive nature of GPU, the effective release time of the second DNN is at time 3, leading to a relative deadline of 2.

However, as is evident in this figure, the GPU resource is wasted quite a lot. The natural remedy for this is to enable concurrent execution, and turn the GPU to a shared resource. However, there is a hurdle to this scenario. The resource utilization of layer 3 of the first DNN task in Fig. 4 does not leave enough resources for the first layer of the second DNN instance to concurrently execute. This brings an interesting challenge. While there is no more approximation required for meeting the deadline, there is a potential benefit for better resource sharing to perform even more approximation. As seen in Table. III, the utilization column can prove useful since the scheduler might decide to go to a higher approximation if the utilization requirements demand it for better concurrency. This addition does not violate our theoretical analysis since a loss of accuracy is always assumed to provide faster execution. We use a monitor module to constantly monitor the GPU for a set of utilization variables. Alg.1 depicts this monitor module alongside our scheduling procedure. Line 2 monitors GPU resources of SM Usage and Memory Usage. If the utilization falls below a threshold (T), the scheduler will try to release another DNN task early (lines 5-11) for concurrency benefits. If the next DNN instance in the GPU FIFO queue does not fit the free resources on GPU, the scheduler will look at higher approximation values for the current DNN (lines 12-14). The result of this procedure is depicted in Fig. 4(b) and Fig. 4(c) respectively. There is the benefit of concurrent execution in Fig. 4(b) as the overall response time is reduced to 4 by increasing the first layer’s approximation to 18%. However, by a 10% approximation for layer 2 of DNN instance 1, the response time can be improved further to 3, as per Fig. 4(c). This highlights the observation that more aggressive approximation may improve resource sharing, which in this case has improved the overall system response time to 3 from the original 5.

As motivated in Sec. III, on the other hand, improved resource sharing may lead to better response time performance, which benefits a task’s subsequent layers’ or other tasks’ execution such that the needed approximation can be reduced. As seen in Fig. 4(b) & (c), the second layer and third layer of the second DNN instance are adjusted to an 8% and 4% approximation. This is due the fact that the increase in the first layer’s approximation has relaxed the timing requirement on subsequent layers. Thus, these layers can have better accuracy. As we will discuss in Sec. V, this might result in better overall accuracy for the DNN.

We present an algorithm that can utilize Equ. 1 at runtime to relax approximation requirements whenever such a scenario arises. Alg. 2 depicts our design. If approximation requirement for the current layer changes (line 3), first, the new adjusted execution time is calculated and stored in variable diff (line

![Fig. 4: Motivational example. Approximation requirements for the second DNN are labeled on top of each layer.](image-url)
Algorithm 1 Scheduling Algorithm

Require: \( \tau_i \) of all remaining layers of DNN

Require: \( T \) of all remaining layers of DNN

Require: \( T_1 \) and \( T_2 \), approximation tables

1: function MONITOR
2: \[ \text{if } \text{GPU.MEM} - \text{USAGE}_{\text{MEM}} < T_{\text{MEM}} \text{ OR} \]
3: \[ \text{GPU.USAGE}_{\text{SM}} < T_{\text{SM}} \text{ then return true} \]
4: function SCHEDULE(InputList, Instances)
5: if Monitor then
6: for \( \tau_i \) in \( T \) do
7: \( o \leftarrow \) counter for \( \tau_i \in T_2 \) \( \triangleright o \) initialized to 0
8: for \( [X^{\tau_i}_j, \leq 100] \) then
9: if \( U(X^{\tau_i}_j) < 100 \) then
10: \( \text{QUEUE} \leftarrow \tau_i \cdot \text{pop}() \)
11: \( \text{QUEUE} \leftarrow \tau^2_i \cdot \text{pop}() \)
12: \( o \leftarrow o + 1 \)
13: if NONE found in \( \tau_i \) then
14: Move \( X^{\tau_i}_j \) to next row of \( T \) if exists
15: go 6

Algorithm 2 TradeOFF Algorithm

Require: \( \tau_i \) of all remaining layers of DNN

Require: \( T \) of approximation requirement for current layer \( k \)

1: function TradeOFF
2: \[ \text{while true do} \]
3: \[ \text{if } X^{k}_i \text{ changes then} \]
4: \( \text{diff} \leftarrow \text{difference in adjusted exec.} \)
5: \( \text{for } \tau^i_{j,k} \geq 1 \text{ do} \)
6: Recalculate \( X^i_j \) based on \( D_i = D_{i+1} - \text{diff} \)

4). Then, the relative deadline is updated by \text{diff} and the subsequent values of \( X_i^j \) are updated to keep the value of \( \sum C_i \) constant.

Of course, two notes should be taken with this approach. First and foremost, an overly aggressive concurrency will result in a longer combined execution. This is easily avoidable by using only an intra-SM multitasking [30] approach, meaning that no DNN can share an SM with another DNN, and concurrency is set up in such a way that the unused resources of the current DNN execution is devoted to another DNN, via an implementation trick of avoiding multi-kernel execution on a single SM. Secondly, our algorithm is runtime-based due to our assumption that the release time of DNN tasks could be arbitrary.

V. EVALUATION

We have fully implemented and extensively evaluated ApNet on a driving-specific embedded hardware, i.e. the NVIDIA Jetson TX2 platform. Jetson TX2 is built around an NVIDIA Parker System-on-a-Chip (SoC). This SoC contains a 6-core big.LITTLE (2 big, 4 LITTLE) CPU and a 2 SM Pascal architecture GPU. This SoC is connected to 8GB of RAM that is shared between the CPU and GPU.

Due to the complicated nature of DNN structures, it is impossible to test every single mutation of many DNN configurations. In this paper, we thus evaluate ApNet using already established approximations of CaffeNet and AlexNet that have been shown to be practical and efficient [9], [10], [12], [31], [32]. Table IV depicts these configurations. The third column shows whether or not the approximated configurations are compatible with AlexNet and CaffeNet. In other words, a single layer from a compatible configuration can be directly dropped in place of a layer in CaffeNet or AlexNet.

Table IV also includes the accuracy loss corresponding to each approximation method. Calculating this accuracy loss is beyond the scope of our research and are extracted from the respective papers.

We note that each layers of AlexNet and CaffeNet are replaced with the approximated versions from Table IV. If ApNet chooses to run SqueezeNet, it has to restart execution.

Table IV also includes the accuracy loss corresponding to each approximation method. Calculating this accuracy loss is beyond the scope of our research and are extracted from the respective papers. This column reports the Top-5 error reported on the ImageNet database. Lastly, due to the complicated nature of DNN approximation, there is no way for us to report accuracy loss for every single approximated configuration (as per our discussion for Practical Considerations in Sec. IV-A).

For the purposes of this paper, we use a scoring mechanism instead. The fourth column in Table IV depicts the respective accuracy scores given to each configuration for each layer. Every percentage of overall accuracy loss is presented by a 0.1 deduction to the per layer score. The maximum score of each layer is assigned to 1. For example, running the BNN exclusively will yield an accuracy score of -24 and running AlexNet will yield a score of 24. Any mutations of these configurations will generate a score accordingly. The higher the approximation score is, the more accurate the overall result of the DNN configuration will be. The maximum accuracy score in the case of ApNet is 24.

Next, we will evaluate ApNet both on AlexNet and CaffeNet. We note that each layers of AlexNet and CaffeNet are replaced with the approximated versions from Table IV where deemed appropriate by our runtime solution. Since we use already established compatible DNN configurations, any mutation by the runtime system will not yield a worse accuracy than the worst case configuration (in our case BNN).

A. Single DNN Scenarios

Our theoretical analysis in Sec. IV-A shows that our method can meet deadlines under any deadline setup. In this section, we put that proposition into real-world testing, as is depicted in Table V. Because Eq. 4 is deadline dependent, it is imperative to test our method under different deadline settings.

We test ApNet under a tight deadline, set to 12ms and a loose deadline set to 24ms. 12ms is a generous deadline assignment for an 8 camera autonomous vehicle running at 30 frames per second [21], [22]. 24ms is also tested herein since it will leave enough headroom of around 9ms for other system tasks to complete.

We evaluate ApNet under the tight deadline setting of 12ms first, as depicted in Table Vc for CaffeNet and Table Va for AlexNet. The first column in these tables depicts the...
TABLE V: Execution costs and sub-deadline assignments for CaffeNet and AlexNet according to the rest of the approximation configurations for a deadline of 12ms.

<table>
<thead>
<tr>
<th>Layers</th>
<th>α</th>
<th>X</th>
<th>Vd</th>
<th>WCET</th>
<th>Adj. WCET</th>
<th>Score</th>
<th>X</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>τ²⁺</td>
<td>0</td>
<td>0</td>
<td>1.25</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>τ²⁻</td>
<td>11.84</td>
<td>0.08</td>
<td>80.50</td>
<td>2287</td>
<td>80.50</td>
<td>0.07</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>τ⁻</td>
<td>0</td>
<td>0</td>
<td>127.12</td>
<td>305</td>
<td>305</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>τ⁻⁺</td>
<td>0</td>
<td>0</td>
<td>69.18</td>
<td>166</td>
<td>166</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>τ⁻⁻</td>
<td>9.65</td>
<td>0.09</td>
<td>6.25</td>
<td>145</td>
<td>6.25</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>τ⁻⁻⁻</td>
<td>11.59</td>
<td>0.08</td>
<td>295.05</td>
<td>8205</td>
<td>295.05</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>τ⁻⁻⁻⁻</td>
<td>0</td>
<td>0</td>
<td>49.18</td>
<td>118</td>
<td>118</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>τ⁻⁻⁻⁻⁻</td>
<td>0</td>
<td>0</td>
<td>82.52</td>
<td>198</td>
<td>198</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>τ⁻⁻⁻⁻⁻⁻</td>
<td>0</td>
<td>0</td>
<td>49.59</td>
<td>119</td>
<td>119</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>τ⁻⁻⁻⁻⁻⁻⁻</td>
<td>0</td>
<td>0</td>
<td>800.66</td>
<td>1921</td>
<td>1921</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>τ⁻⁻⁻⁻⁻⁻⁻⁻</td>
<td>0</td>
<td>0</td>
<td>36.67</td>
<td>88</td>
<td>88</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>τ⁻⁻⁻⁻⁻⁻⁻⁻⁻</td>
<td>12.40</td>
<td>0.07</td>
<td>53.00</td>
<td>1578</td>
<td>53.00</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>τ⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻</td>
<td>0</td>
<td>0</td>
<td>45.84</td>
<td>110</td>
<td>110</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>τ⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻</td>
<td>13.46</td>
<td>0.07</td>
<td>38.65</td>
<td>1249</td>
<td>38.65</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>τ⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻</td>
<td>0</td>
<td>0</td>
<td>45.01</td>
<td>108</td>
<td>108</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>τ⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻</td>
<td>8.43</td>
<td>0.11</td>
<td>5.042</td>
<td>102</td>
<td>5.042</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>τ⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻</td>
<td>3.85</td>
<td>0.2</td>
<td>8.97</td>
<td>83</td>
<td>8.97</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>τ⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻</td>
<td>0</td>
<td>0</td>
<td>1.25</td>
<td>3</td>
<td>1.25</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>τ⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻</td>
<td>3.85</td>
<td>0.2</td>
<td>8.97</td>
<td>83</td>
<td>8.97</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>τ⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻</td>
<td>0</td>
<td>0</td>
<td>1.25</td>
<td>3</td>
<td>1.25</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>τ⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻⁻</td>
<td>1.25</td>
<td>0.05</td>
<td>23.51</td>
<td>1045</td>
<td>23.51</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Total: 12000 28791 10751.2 11.5 12.5

α = 0, X is always 0 since no approximation could be beneficial. Moreover, the approximation is capped at 20% since our lowest configuration is BNN at 20% accuracy loss. The assignments for X and Vd materialize in the form of Adjusted Worst Case Execution Time (Column 5). This value is calculated from the first part of Equ. 5 and by using the values from column four. Finally, an approximation score is assigned to each layer. This score is assigned by consulting Table IV. For example, for an X = 0.09, 9% approximation is requested. The next highest configuration with that amount of approximation is XNor-Net with an accuracy loss of 10%, and the corresponding score for that layer will be 0 (the per-layer score of XNor-Net is 0). By using the same method, the score for X = 0.5 is 0.5 (Lowrank is chosen with a per-layer score of 0.5).

As illustrated in the last row of Table V, ApNet is able to meet the overall deadline. The total sum value of Vd is always 12ms for the tight deadline. The WCET for CaffeNet and AlexNet are noticeably large, and more than twice the required deadline. After adjustments through ApNet, the respective WCET are below the deadline.

Under the loose deadline setting of 24ms, as seen in Table Vb and d, both CaffeNet and AlexNet will still miss their deadlines originally, but they should not require as much approximation to meet the demand. In these tables, we only depict a new value for X and the respective scores. As is evident in the tables, the accuracy score is improved in both cases. Noticeably, the accuracy score of CaffeNet under ApNet is improved by 1. An increase of accuracy score of 1 is 4% of the total score of 24 and is significant in the realm of DNNs.

B. Multitasking Scenarios

In this section, we evaluate ApNet under a multi-DNN scenario. We compare ApNet against three major approaches:

- A baseline solution that is either based on AlexNet or CaffeNet (denoted “Baseline”).
- A state-of-the-art best effort approach that explores approximation for better latency performance [10], which seeks to improve latency on a best effort basis on top of a carefully crafted configuration, SqueezeNet (denoted “Best Effort”).
Our goal is to understand under a multitasking scenario, the capability of ApNet in meeting deadlines, reducing accuracy loss (less approximation), improving resource sharing, and reducing incurred runtime overhead.

1) Meeting Deadlines: In this set of experiments, we apply the same setup and run ApNet and other approaches for 1000 iterations with 4 concurrent DNN processes. Fig. 5 depicts the cumulative distribution function of average normalized turnaround times. We run ApNet based on both CaffeNet (Fig. 5a) and AlexNet (Fig. 5b). As depicted in the figure, ApNet easily meets a deadline of 24ms in both cases. ApNet-SP overall does not perform well since it does not exploit the advantage of resource sharing and concurrency. Moreover, the best effort method is not capable of meeting the strict deadline. Notably, since AlexNet is slightly faster than CaffeNet, it runs slightly faster than our ApNet-SP, as depicted in Fig. 5b. Regardless, the full multi-tasking ApNet can guarantee deadlines with ample headroom.

2) Accuracy: The second aspect of our multi-tasking design was to reduce the amount of approximation (i.e., improve accuracy) through exploring the mutually supplementary relation between resource sharing and approximation. Fig. 6 compares ApNet against the other approaches in terms of average layer accuracy under a 1 DNN (1-P), 2 DNN (2-P), and 4 DNN (4-P) scenario. We use the same scores described in Table IV to calculate per-layer accuracy score and average the value among all layers. Since AlexNet and CaffeNet are our baselines, they have the absolute best average layer accuracy of 1.

As seen in the figure, Best Effort has a constant accuracy score of 0.9 since it is not reactive to multi-processing. The same situation applies to our single tasking solution, which does not change accuracy based on runtime variables and incurs a constant average accuracy score of 0.5. As is seen in the figure, ApNet can be almost as accurate as AlexNet and CaffeNet whenever there is only one DNN process running. Moreover, the accuracy drops when more concurrent processes are running. ApNet outperforms the best effort approach in terms of accuracy under a 2 DNN process scenario, but perform slightly worse under the 4 DNN process scenario. This is natural because the best effort approach, as well as the baseline, will miss all the deadlines under the 4 process scenario. ApNet provides an optimization that reasonably sacrifices on accuracy in order to guarantee deadlines. The maximum accuracy improvement over ApNet-SP is 80%.

3) Resource Utilization: As we have discussed in Sec. IV-B, ApNet is motivated by the observation that resource sharing and approximation have a mutually supplementary relation. Thus, in this set of experiments, we seek to understand whether ApNet is able to improve the system resource utilization. Fig. 7a depicts the per-layer utilization of ApNet compared to the baseline CaffeNet under a 2 process scenario. Fig. 7b depicts the per-layer utilization for AlexNet as baseline. As is evident in the figure, both CaffeNet and AlexNet will show a dip in resource utilization towards the end of the first DNN task and the beginning of the execution of the second DNN task. This is to be expected since final layers of DNNs are usually the layers that under-utilize the GPU, as we have previously reported [20].

As is evident in Fig. 7, ApNet is able to keep the utilization consistently high. Two observations are worth mentioning here. First, there will still be minor dips in resource utilization under ApNet, although minimal. This is due to a combination of overhead and an occasional lack of enough combined resource utilization for overlapping layers of consecutive DNN processes. The second notable observation is that a slightly large dip in ApNet also happens, but at a later point of time (at around 65% execution) compared to the baseline. This is due to the fact that the execution times for ApNet and the baselines are not the same, but are nonetheless scaled to 100% in Fig. 7 for clarity purposes. In actual execution, ApNet has a significantly shorter execution time, and thus, this dip in...
resource utilization will happen much earlier when compared to CaffeNet and AlexNet. The difference in resource utilization in these overlapping layers is 3.5x on average.

4) Overhead: Finally, since ApNet includes an online module, we also evaluate the incurred overhead at runtime, as per Table VI. The majority of overhead is due to resource-monitoring and DNN configuration switching. As seen in this table, under a 1 process scenario, ApNet will skip the majority of its online operations. As the system scales to a 2 process and a 4 process scenario, the overhead will be progressively larger. Nonetheless, the overhead remains reasonably small and does not affect the real-time behavior of ApNet (as ApNet guarantees meeting deadlines).

VI. RELATED WORK

DNNs are an essential component in any embedded system that requires sensor/image data processing for autonomous decision making [33], particularly in the case of autonomous vehicles [34]. DNN workloads are shown to be highly compute-intensive [20]. Best effort DNN approximations have been proposed before in order to achieve optimized latency performance in different application domains [13]–[16], [35]. However, none to the best of our knowledge can guarantee timing predictability (i.e., meeting hard deadlines or achieving any sort of analytically verifiable timing guarantees). Moreover, these solutions mostly target at the single DNN scenario, which cannot be efficiently applied in multi-tasking scenarios where resource sharing may play a critical role in performance optimization.

A recent set of work has explored the fitness of DNNs on embedded systems through various natural variables, such as latency [20], real-time guarantees [4] and concurrency [36]. However, research on the influence of approximation on these natural variables is lacking. In this paper, our proposed method explores the mutually beneficial relationship between guaranteed real-time predictability and approximation. Moreover, our runtime module can utilize the trade-off between approximation and resource utilization to improve both at the same time.

VII. CONCLUSION

We propose ApNet, a real-time DNN platform that can guarantee hard deadlines via layer-wise approximation. ApNet is composed of an end-to-end scheduling approach that considers layer’s approximation potential, and a runtime component that can expand this guaranteed predictability to multi-tasking scenarios and exploit GPU resource sharing to improve both utilization and DNN accuracy. Extensive evaluation shows that ApNet is timing-predictable and achieves acceptable DNN accuracy while inducing a low overhead.

REFERENCES


