IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 11 February 2024, accepted 24 March 2024, date of publication 2 April 2024, date of current version 10 April 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3384233

RESEARCH ARTICLE

QoS-Aware Inference Acceleration Using Adaptive Depth Neural Networks

WOOCHUL KANG, (Member, IEEE)
Department of Embedded Systems Engineering, Incheon National University, Yeonsu-gu 22012, South Korea
e-mail: wchkang@inu.ac.kr

This work was supported in part by the National Research Foundation of Korea (NRF) Grant funded by the Ministry of Education under Grant NRF-2022R1F1A1074211; and in part by Incheon National University Research Grant, in 2021.

ABSTRACT While deep neural networks (DNNs) have brought revolutions to many intelligent services and systems, the deployment of high-performing models for real-world applications faces challenges posed by resource constraints and diverse operating environments. While existing methods such as model compression combined with inference accelerators have enhanced the efficiency of deep neural networks, they are not dynamically adaptable to dynamically changing resource conditions since they provide static accuracy-efficiency trade-offs. Further, since they are not aware of performance requirements, such as desired inference latency, they are not able to provide robust and effective performance under unpredictable workloads. This paper introduces a holistic solution to address this challenge, consisting of two key components: adaptive depth neural networks and the Quality of Service (QoS)-aware inference accelerator. The adaptive depth neural networks exhibit the ability to scale computation instantly with minimal impact on accuracy, utilizing a novel architectural pattern and training algorithm. Complementing this, the QoS-aware inference accelerator employs a feedback control loop, adapting network depth dynamically to meet desired inference latency. Experimental results demonstrate that the proposed adaptive depth networks outperform non-adaptive counterparts, achieving up to 38% dynamic acceleration via depth adaption, with a marginal accuracy loss of 1.5%. Furthermore, the QoS-aware inference accelerator successfully controls network depth at runtime, ensuring robust performance in unpredictable environments.

INDEX TERMS Deep learning, inference acceleration, deep neural networks, feedback control, real-time, energy efficiency, quality-of-service, QoS.

I. INTRODUCTION

In the past decade, tremendous progress has been made in deep learning towards providing robust and accurate inference capability for many real-world applications, such as robotics and self-driving cars. Since many real-world applications have resource constraints, several methods have been proposed to scale-down deep learning models, such as model compression [1], [2], knowledge distillation [3], [4], or compact architectures [5], [6], to name a few. However, in recent years, adaptive, real-time data-driven applications of deep neural networks have emerged, which need to perform time-sensitive tasks with efficiency under dynamically changing resource conditions. For example, when a self-driving car operates in densely populated urban areas, it requires more computation for object detection [7], [8] than when driving in rural areas. Despite varying workloads, the inference tasks need to be completed within a certain time bound for safety. While aforementioned methods for efficient deep neural networks have demonstrated significant gains in efficiency, their accuracy-efficiency is fixed during the development, and, hence, once deployed, further adaptation to dynamically varying environments and resource conditions is impossible. In an ideal situation, we would be able to train a single neural network and scale the network instantly at runtime with marginal loss of inference accuracy. Further, the runtime environment of the neural networks is aware of the time bound of the inference tasks and should be able to exploit the adaptability of the neural networks to meet the target inference latency and

The associate editor coordinating the review of this manuscript and approving it for publication was Peng-Yong Kong.

energy consumption under varying workloads and resource conditions.

In this paper, we present a holistic approach to providing predictable inference performance for deep learning applications. Our approach consists of two major components as shown in Figure 1. The first component is *adaptive depth neural networks* that can scale the computation of the networks instantly with marginal impact on inference accuracy. Another component is the *QoS-aware inference accelerator* that provide systematic trade-offs between accuracy and efficiency by adapting the depth of the networks. This work addresses challenges with these two components in providing predictable inference performance.

While several adaptive neural networks have been proposed that adapt widths [9], [10] [11], depths [12], [13] [14], resolution [15], [16], tokens [17], [18], or any combination of them, their performance loss with the model scaling is not negligible and they usually require significantly longer training time than non-adaptive counterpart networks. Further, while adaptive networks can reduce the amount of computation in theory, many of them show only marginal acceleration and energy savings on actual devices. In Section III, we introduce an architectural pattern and training algorithm for adaptive depth networks that address these challenges of prior adaptive networks. Our depth adaptation networks exploit the property of residual skip connections [19] to minimize the loss of inference accuracy, and, hence, can be applied to most modern residual networks, including both convolution networks and transformers. While our adaptive depth networks support many sub-networks of different depths, the training time is significantly shorter than previous adaptive networks.

In Section IV, we introduce an inference accelerator that exploits the proposed adaptive depth neural networks to meet the desired inference latency. Due to the complexity of neural networks and their runtime environments, it is a challenging task, if not impossible, to determine the proper network depth to meet the desired inference latency. To address this problem, the proposed QoS-aware inference accelerator exploits a feedback control loop, where the depth of the networks is adapted by constantly comparing the actual inference latency to the target inference latency. The network depth is determined to counteract the deviation from the desired inference latency. We build a system model that describes the relationship between the network depth and the latency of inference tasks, and apply formal control theory to design a controller for inference tasks.

To demonstrate the viability of our approach, representative deep neural networks, both from convolutional neural networks and vision transformers, are extended according to the proposed architectural pattern and trained for dynamic depth adaptation. Our experiment results in Section V-B show that our adaptive depths networks match or outperform non-adaptive counterpart networks and achieve actual inference acceleration through runtime adaptation of network depths. For example, the FLOPs of our adaptive depth ResNet-50 can be adapted dynamically by up to 38% with a marginal loss of up to 1.5%. We have implemented a prototype of the QoS-aware inference accelerator to further show that these adaptive depth networks can provide robust performance under unpredictable and varying environments. Our experiment results in Section V-C show that our inference accelerator can control the network depth dynamically at runtime and closely support the target inference latency despite unpredictable workloads.

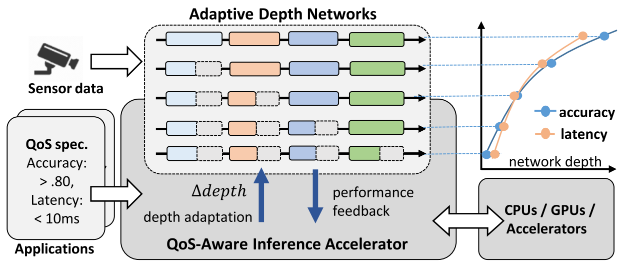


FIGURE 1. **QoS management architecture for adaptive, real-time data-driven deep learning applications. The QoS-aware inference accelerator constantly monitors the actual inference latency and adapts the depth of the neural network dynamically to support the target inference latency.**

II. OVERVIEW OF QOS-AWARE INFERENCE

A. SERVICE MODEL

This work targets soft real-time applications that need to perform deep learning inferences for continually incoming sensor data, e.g, video streams, sensor readings, etc., in a predictable manner. These applications may have varying workloads due to changes in the physical environment. Under such dynamic environments, these applications are supposed to support predictable inference performance in a resource-efficient manner.

Providing predictable target performance, or Quality-of-Service (QoS), for deep learning applications requires holistic support both from neural networks and inference accelerators. Figure 1 illustrates the proposed QoS-aware inference architecture where two major components are (1) adaptive depth neural networks and (2) the QoS-aware inference accelerator. An application requests the *QoS-aware inference accelerator* to execute an inference task with a desired performance goals, such as target inference latency. The QoS-aware inference accelerator executes the task with a neural network $\mathbf{M}$ that is specifically designed and trained to support various accuracy-efficiency trade-offs in a single network with minimal loss of inference accuracy. The QoS-aware inference accelerator constantly monitors the inference task and adjusts the depth of $\mathbf{M}$ by $\Delta depth$ to counteract the deviation of monitored performance from the desired QoS level.

To support the QoS goals, the inference accelerator might exploit another system-level resource manages, such as DVFS (dynamic voltage frequency scaling) governors, along with the adaptive depth networks. In this work, however, we focus on achieving QoS goals by adapting network depths alone, and reserve this issue as a future work.

B. PERFORMANCE METRIC

In this work, we consider *inference latency* as a primary performance metric for QoS-aware inferences, since inference latency can be converted easily to different performance metrics. For instance, in real-time augmented reality wearables, the target frame rate, e.g, 25 frames/sec, can be obtained by setting the target inference latency to 40 milliseconds.

One difficulty with the inference latency is that it has high variance across different neural networks and underlying hardware/software environments. Therefore, we need more normalized performance metrics that can be used across various networks and runtime environments. We might consider *deadline miss ratio* that is often used to measure the percentage of requests that miss their target deadlines over a specified time period. However, many deep learning applications of edge devices do not make many inference requests, and, hence, deadline miss ratio does not provide statistically stable interpretation of the timeliness of inferences.

Therefore, we use *tardiness* as a primary metric to monitor the timeliness of inference requests:

$$tardiness = \frac{\text{measured inference latency}}{\text{desired inference latency}} \tag{1}$$

Unlike deadline miss ratio, *tardiness* enables stable interpretation even if only a small number of inference requests are made. If tardiness is greater than 1, it informs that the inferences take longer than their desired latency. Conversely, if the tardiness is less than 1, inferences complete earlier than their desired latency, which presents potential opportunity to save resources such as computation and energy.

III. ADAPTIVE DEPTH NETWORKS

In this section, we present an architectural pattern and training method that minimizes the degradation of inference accuracy of the original network while supporting instant depth adaptation at inference time.

A. NETWORK ARCHITECTURE

In modern networks, such as Resnets [19] and transformers [20], a residual block with a skip connection is a commonly found architectural pattern. Previous works [21] have shown that these residual blocks not only learn hierarchical features like traditional compositional networks, such as VGG networks [22], but also refine input features without changing their feature level. These prior results imply that if, by a chance, some residual blocks were trained to only perform feature refinement while preserving the level of input features, then skipping those blocks may not significantly affect network performance at test time. However, in residual networks with typical training methods, every residual block learns new level features and refine learned features, and, hence, randomly skipping residual blocks during inference results in significant performance degradation.

Therefore, we can hypothesize that if we intentionally train some residual blocks to focus more on refining the features while preserving the feature level, skipping those residual blocks may not affect the network performance significantly at test time. To this end, we first propose an architectural pattern for adaptive depth networks that divides every residual stage into two consecutive sub-paths, a *mandatory sub-path* and a *skippable sub-path*.

Figure 2 illustrates a residual stage with 2 sub-paths. In the s-th residual stage with L residual blocks, the first sub-path applies the first half blocks to produce the intermediate feature $\mathbf{h}^s_{L/2+1}$, or $\mathbf{h}^s_{base}$:

$$\underbrace{\mathbf{h}^s_1 + F_1(\mathbf{h}^s_1) + \ldots + F_{L/2}(\mathbf{h}^s_{L/2})}_{\mathbf{F}^s_{base}} = \mathbf{h}^s_{L/2+1} = \mathbf{h}^s_{base} \tag{2}$$

This first sub-path is mandatory for every sub-network. In contrast, the second sub-path can be skipped, to save computation cost, since it is supposed to only refine input features without changing the level of features. The second sub-path is applied to $\mathbf{h}^s_{L/2+1}$, or $\mathbf{h}^s_{base}$, and generates more refined features $\mathbf{h}^s_{super}$:

$$\mathbf{h}^s_{L/2+1} + \underbrace{F_{L/2+1}(\mathbf{h}^s_{L/2+1}) + \ldots + F_L(\mathbf{h}^s_L)}_{\mathbf{F}^s_{skippable}} = \mathbf{h}^s_{super} \tag{3}$$

This relation between $\mathbf{h}^s_{base}$ and $\mathbf{h}^s_{super}$ can be summarized as follows:

$$\mathbf{h}^s_{super} = \mathbf{h}^s_{base} + \mathbf{F}^s_{skippable}(\mathbf{h}^s_{base}) \tag{4}$$

This architectural pattern is repeated for every residual stage, and, hence, for an adaptive depth network with N_r residual stages, 2^{N_r} sub-networks of different depths can be chosen on the fly by selectively skipping $\mathbf{F}^s_{skippable}(\mathbf{h}^s_{base})$, where $s = 1, \ldots, N_r$.

For an actual implementation of adaptive depth network $\mathbf{M}$, we extend the forward pass of $\mathbf{M}$ as shown in Algorithm 1. The model $\mathbf{M}$ accepts an additional argument *skip* that indicates the residual stages in which their second sub-paths are skipped. For example, the smallest sub-network, or base-net, of $\mathbf{M}$ with 4 residual stages can be selected instantly by passing $skip = [T, T, T, T]$ along with input $\mathbf{x}$ to $\mathbf{M}$.

B. TRAINING OF ADAPTIVE DEPTH NETWORKS

In order to minimize the performance impact on skipping sub-paths, we train $\mathbf{h}^s_{base}$ and $\mathbf{h}^s_{super}$ to have similar feature distributions for input $\mathbf{X}$:

$$P(\mathbf{h}^s_{super}) \approx P(\mathbf{h}^s_{base}) \quad \text{for } x \in \mathbf{X}, s = 1, \ldots, N_r \tag{5}$$

If two feature representations $\mathbf{h}^s_{base}$ and $\mathbf{h}^s_{super}$ have similar distributions for the same input, skipping $\mathbf{F}^s_{skippable}(\mathbf{h}^s_{base})$ in Equation 4 incurs little internal covariate shift [23] to the following layers. During training, the property in Equation 5 can be enforced by minimizing Kullback-Leibler (KL) divergence between $\mathbf{h}^s_{super}$ and $\mathbf{h}^s_{base}$ over training input X:

$$\sum_{s=1}^{N_s} \underset{x \in \mathbf{X}}{D_{KL}}(\mathbf{h}^s_{super} \| \mathbf{h}^s_{base}) \tag{6}$$

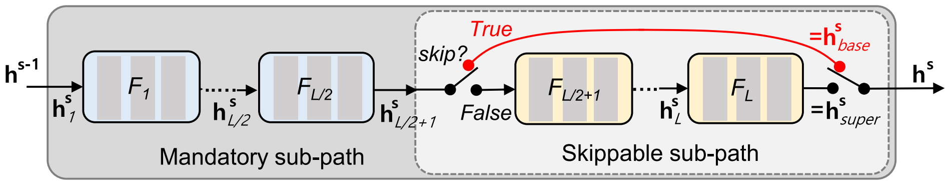


FIGURE 2. A residual stage with two consecutive sub-paths. While the first mandatory sub-path is trained to learn hierarchical features without any constraint, the second sub-path is trained to be skippable by being optimized to preserve the distribution of input features $\mathbf{h}^s_{base}$. Since $\mathbf{h}^s_{base}$ and $\mathbf{h}^s_{super}$ have similar distributions, skipping the skippable sub-path incurs marginal impact to the next layers. In the mandatory sub-path, we exploit another set of batch normalization (BN) operators when the second sub-path is skipped. Adaptive depth networks are built by repeating this architectural pattern throughout the network.

Algorithm 1 A Forward Pass of Adaptive Depth Network **M**. N_r Denotes the Number of Residual Stages of M

```
1:  Input $\mathbf{h}^0 = \mathbf{x}$: input data to M
2:  Input skip: boolean values for skipping sub-paths
3:  for s-th residual stage $R_s$ of M do                ▷ $(s = 1, \dots, N_r)$
4:      $\mathbf{h}^s_1 = \mathbf{h}^{s-1}$
5:      for l-th block $B_l$ of $R_s$ do                  ▷ $(l = 1, \dots, L)$
6:          if skip[s] == True and $l > \frac{L}{2}$ then
7:              break                           ▷ skip the last half blocks
8:          else
9:              $\mathbf{h}^s_{l+1} = B_l(\mathbf{h}^s_l)$        ▷ Execute l-th block of $R_s$
10:         end if
11:     end for
12:     if skip[s] == True then
13:         $\mathbf{h}^s = \mathbf{h}^s_{L/2+1}$                          ▷ $\mathbf{h}^s_{base}$
14:     else
15:         $\mathbf{h}^s = \mathbf{h}^s_{L+1}$                            ▷ $\mathbf{h}^s_{super}$
16:     end if
17: end for
```

KL divergence measures how two probability distributions are different for the same random variable.

In Algorithm 2, our training method, called *skip-aware self-distillation*, enforces the property in Equation 5 by including the Equation 6 as a regularization term in the loss function. In our algorithm, two parameter-sharing sub-networks, or *super-net* and *base-net*, are used to train **M**. *Super-net* and *base-net*, respectively, are the largest and the smallest sub-networks of **M**. For example, base-net can be selected from **M** by skipping skippable sub-path of every residual stage in Algorithm 1. In contrast, super-net executes every residual stage without skipping any sub-paths.

In Algorithm 2, each iteration with a batch has two steps. In the first step (lines 3 - 5), forward/backward passes of the super-net are performed to get gradients for training the super-net. In the second step (lines 6 - 8), the gradients for the base-net is obtained using the self-distillation from the super-net. During this self-distillation step, the term in Equation 6 is included in the loss function $loss_{base}$ to enforce explicitly the property in Equation 5.

In [24], we provide more detailed ablation analysis and formal rationale for why this skip-aware self-distillation reduces prediction errors while minimizing the impact of skipping sub-paths.

Algorithm 2 Training Adaptive Depth Network **M** With *skip-aware self distillation*

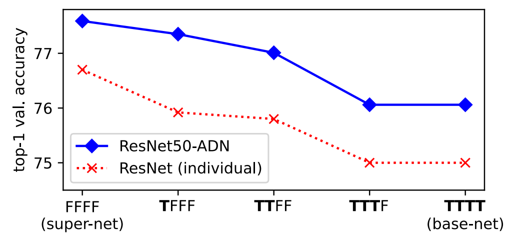
```
1:  for $i = 1$ to $n_{iters}$ do
2:      Sample a mini-batch of data x and label y
3:      $\hat{\mathbf{y}}_{super} = \mathbf{M}(\mathbf{x}, skip = [F, F, F, F])$        ▷ super-net
4:      $loss_{super} = criterion(\mathbf{y}, \hat{\mathbf{y}}_{super})$
5:      $loss_{super}.backward()$
6:      $\hat{\mathbf{y}}_{base} = \mathbf{M}(\mathbf{x}, skip = [T, T, T, T])$          ▷ base-net
7:      $loss_{base}$ = Equation (6) $+ D_{KL}(\hat{\mathbf{y}}_{super} \| \hat{\mathbf{y}}_{base})$
8:      $loss_{base}.backward()$
9:      Update parameters of M
10: end for
```

C. EFFECTS OF SCALING NETWORK DEPTH

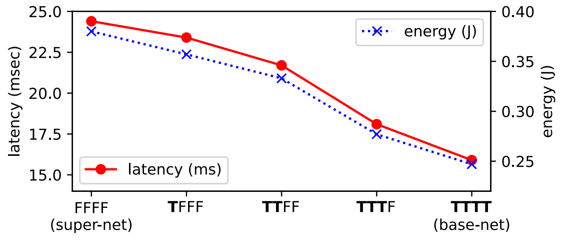
Scaling the depth of a neural network has an impact on many aspects of inference performance, including accuracy, latency, energy consumption, etc. In this section, we show the effect of scaling the depth of a network using *ResNet50-ADN* as an example. *ResNet50-ADN* is a variant of ResNet50 extended according to the architectural pattern in Figure 2 and is trained using Algorithm 2. The details of training setting and more results are discussed in Section V.

Figure 3-(a) shows the validation accuracy on ImageNet of ResNet50-ADN that is trained according to Algorithm 2. The depth of ResNet50-ADN is controlled by varying the number of residual stages in which their second sub-paths are skipped. In ResNet50-ADN, FLOPs can be varied from 4.12 GFLOPs to 2.58 GFLOPs by adapting the depth from the super-net to the base-net. The result shows that the accuracy of our ResNet50-ADN degrades gracefully as the depth is scaled down. While only the super-net and the base-net are trained explicitly in Algorithm 2, all sub-networks of ResNet50-ADN outperform individually trained ResNets of equivalent depths.

The result in Figure 3-(b) with a Nvidia Jetson Orin Nano device demonstrates that the inference latency and energy consumption of ResNet50-ADN is scaled linearly as the network depth is scaled. This linear relation between the network depth and latency (and energy) is essential for inference accelerators to provide actual acceleration for



(a) ImageNet validation accuracy of ResNet50-ADN is compared to the counterpart individual ResNets with the same depths. Each point of ResNet50-ADN represents the sub-network of different depths.



(b) Inference energy and latency of sub-networks of ResNet50-ADN. Measured on Nvidia Jetson Orin Nano with batch size of 1.

FIGURE 3. The effect of scaling the depth of ResNet50-ADN. The boolean values in the x-axis represent the sub-networks of ResNet50-ADN in which the sub-paths are skipped in the corresponding residual stages.

real-time deep learning applications. In the following section, we discuss our inference accelerator exploiting this property of adaptive depth networks to support predictable inference.

IV. QOS-AWARE INFERENCE ACCELERATOR

Inference accelerators for neural networks are responsible for efficiently executing trained models on target devices. In this section, we introduce an inference accelerator that supports the desired performance, or QoS, by exploiting the adaptability of our adaptive depth neural networks. We apply classic feedback control approach as an underpinning theory to support the desired performance under dynamically changing environments.

A. MODELING OF INFERENCE LATENCY

For systematic control of network depths, we need a mathematical model that captures the dynamics of inference tasks. We use *system identification* [25] for the modeling of deep learning inference tasks. System identification is a black box modeling approach in which detailed knowledge of the target system is not required, but only the statistical relationship between the control inputs (e.g., network depth) and the system output (e.g., inference latency) is captured through experiments [25].

As discussed in Section II-B, we use the *tardiness* as a primary performance metric to quantify the timeliness of inference tasks. The tardiness of an inference task can be modeled using linear difference equations as follows:

$$tardiness(k) = \sum_{i=1}^{n} \alpha_i tardiness(k-i) + \sum_{j=1}^{m} \beta_j depth(k-j) \quad (7)$$

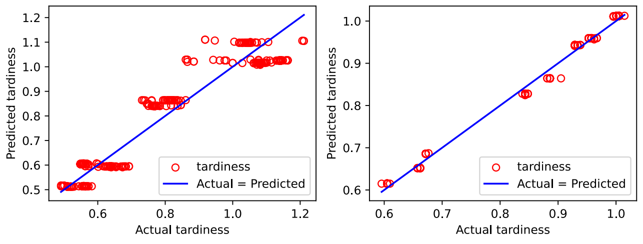


FIGURE 4. Comparison of predicted and measured tardiness for ResNet50-ADN. The longer sampling time reduces the variability of actual tardiness.

In Equation 7, $tardiness(k)$ and $depth(k)$ represent the tardiness of the inference task and the depth of the neural network, respectively, in the kth sampling time. This system model in the time domain shows that the tardiness of an inference task in time k is determined by the tardiness in previous n cycles and the network depths in previous m cycles. In general, more powerful models are obtained by having higher values for n and m since more past histories are considered for the modeling. However, a higher order model requires more complex controller design and runtime overheads. Throughout experiments with various neural networks, we found that the first-order model, or $n = m = 1$, is sufficient:

$$tardiness(k) = \alpha tardiness(k-1) + \beta depth(k-1) \quad (8)$$

In the system identification, the parameters α and β in Equation 8 are estimated by collecting data through experiments. For example, during the experiment, the tardiness of inference tasks is monitored in regular intervals while varying the network depth. Due to the stochastic nature of the target system, the sampling time needs to be determined carefully. For example, the latency of inference can be affected by other unpredictable factors such as resource contention from other tasks. While longer sampling interval can smooth such stochastic behavior, it means that the system reacts more slowly to the change of network depths.

TABLE 1. Model parameters and the model accuracy in R^2.

Sampling interval	α	β	R^2
Every 1 inferences	-0.044	1.634	0.925
Every 10 inferences	-0.125	1.166	0.986

Table 1 shows the model parameters that we obtained from ResNet50-ADN using 2 different sampling intervals. In Table 1, we also need to note that the system exhibits larger β than α. This implies that the inference tardiness is more affected by the depth of the network than past inference tardiness.

The accuracy of the models in Table 1 can be measured using R^2:

$$R^2 = 1 - \frac{variance(tardiness - \widehat{tardiness})}{variance(tardiness)}, \quad (9)$$

where $\widehat{tardiness}$ is the prediction from the system model in Equation 8. In general, a model is considered to be valid if $R^2 \geq 0.8$. The results show that the system model is more accurate when the tardiness is measured by averaging 10 inferences. Figure 4 also shows that the actual tardiness has high variability when the tardiness is measured on every inference.

B. FEEDBACK CONTROL OF INFERENCE LATENCY

Feedback control works by constantly monitoring actual behavior of a system and providing corrective action to drive the system to do the desired behavior [26]. Due to this self-correcting nature of the feedback control, only a simple statistical model such as Equation 8 is suffice and no further detailed knowledge is required for the complex interaction between the neural networks and the runtime environment. We apply the procedure for designing feedback controllers in classical control theory to our feedback control of inference tardiness.

Figure 5 shows the proposed feedback control loop for the QoS-aware inference accelerator, in which the latency of an inference task is compared to the target latency constantly. If the actual inference latency differs from the target latency, or the tardiness is not close to 1, the depth of the network is adapted to counteract the deviation.

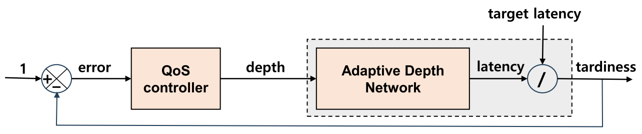


FIGURE 5. Feedback control loop to support target QoS (latency) using dynamic adaptation of network depth.

The input to the controller is the tardiness error, $e(k) = 1.0 - tardiness(k)$, and the output of the controller is the network depth, $depth(k)$. The designing of a feedback controller requires formal control theory to ensure that the controller has desired properties, such as settling time (k_s) and maximum overshoot(M_p), when tracking the desired tardiness. We choose to exploit a PI (Proportional Integral) controller that is known to be robust against noisy behavior that is typical in computing systems [27]. The PI controller has the following form:

$$depth(k) = depth(k-1) + (K_P + K_I)e(k) - K_P e(k-1), \tag{10}$$

where K_P and K_I are gains, respectively, for the proportional and the integral parts of the controller.

For convenient analysis, the control theory provides a way to express signals and systems in the z-domain. The system model in Equation 8 is described as a transfer function $G(z)$ as follows:

$$G(z) = \frac{\beta}{z - \alpha} \tag{11}$$

And the transfer function $K(z)$ for the PI control law in Equation 10 is described as follows:

$$K(z) = \frac{depth(z)}{e(z)} = K_P + \frac{K_I z}{z-1} \tag{12}$$

Finally, the closed loop transfer function $F(z)$, representing the overall system in Figure 5, is constructed by combining the system model and the controller as follows:

$$F(z) = \frac{K(z)G(z)}{1 + K(z)G(z)} \tag{13}$$

With the closed loop transfer function $F(z)$, the controller gains K_P and K_I can be obtained via several methods, such as Root Locus and pole placement, to support the desired controller properties.

TABLE 2. Different controller gains (K_P and K_I) have trade-offs between the responsiveness (small K_S) and stability (small M_P).

Settling time (k_s)	Max. overshoot (M_p)	K_P	K_I
5	15%	-0.24	0.87
10	10%	-0.42	0.31
15	5%	-0.52	0.12

Table 2 shows the controller gains and their properties, obtained through the pole placement technique. Note that there exists a trade-off between the settling time and the maximum overshoot. For example, while short settling time, e.g, k_s=5, guarantees faster response to tardy (or hasty) inference tasks, its high overshoot, e.g., $M_p = 15\%$, may result in high variance in inference latency.

C. NETWORK DEPTHS AS CONTROL INPUTS

During the design of the feedback controller, we assume that $depth(k)$ can be any continuous values. However, in our adaptive depth networks, the network depth has minimum/maximum bounds and only a few discrete steps, e.g., 5 depths in ResNet50-ADN, are available. This problem can be addressed by exploiting PWM (pulse width modulation) technique used in power control. For example, an arbitrary depth of a network can be achieved by switching between two sub-networks of different depths at a certain ratio. For example, if n and $N-n$ inferences out of total N inferences are performed by the base-net and the super-net, respectively, then, it has the effect of doing inferences with the network of following depth:

$$depth = \sum_{i=1}^{N} \frac{F_{super} \times (N-n) + F_{base} \times n}{F_{super} \times N} \tag{14}$$

where F_{super} and F_{base} denote the FLOPs of the super-net and the base-net, respectively. This effective network depth is a continuous value that is less than or equal to 1.

V. EVALUATION

The objectives of the evaluation are 1) to investigate the effectiveness and generality of the proposed adaptive depth networks, and (2) to test if the proposed QoS-aware inference accelerator can support the desired performance goals under dynamic environments by exploiting the adaptive depth networks.

A. TRAINING AND TESTING SETTINGS

We choose 3 representative residual networks as base models and extend them to adaptive depth networks according to the proposed architectural pattern in Section III-A: ResNet is a CNN model and ViT and Switn-T are vision transformers. For depth adaptation, every residual stage of these models is divided evenly into two sub-paths and the second sub-path is trained to be skippable. Since ViT-b has 12 encoder blocks without explicit residual stages, we divide them into 4 stages, each with 3 encoder blocks, and split each stage into 2 sub-paths with 2 and 1 encoder blocks, respectively. With this setting of skippable sub-paths, our adaptive depth networks can support 2^4 parameter-sharing sub-networks in a single network. These adaptive depth networks are denoted with the suffix '-ADN' and their sub-networks are denoted by a list of boolean values indicating residual stages in which the second the sub-path is skipped. For example, the base-net of ResNet50-ADN is denoted by ResNet50-ADN(TTTT) or ResNet50-ADN(base-net).

For the training of our adaptive depth networks, we use ILSVRC 2012 ImageNet dataset that has 1000 classes. The dataset has 1.28M training images and 50K validation images. For a fair comparison, we follow training settings in the original papers. Specifically, we adhere to the hyperparameter configurations recommended in the PyTorch reference training scripts [28], which offer comprehensive details on learning rates, batch sizes, training schedules, and other hyperparameters for various network architectures.

For testing of trained models on actual devices, we use Nvidia Jetson Orin Nano [29] that is an ARM-based embedded board with 1024 CUDA cores and 32 Tensor cores. For the real-time measurement of energy consumption, we use Yokogawa WT310 power meter that is connected to the device through the USBTMC communication interface.

B. PERFORMANCE OF ADAPTIVE DEPTH NETWORKS

1) IMAGENET CLASSIFICATION PERFORMANCE

The results in Table 3 shows the performance on the ImageNet classification task. It shows that our adaptive depth networks outperform counterpart non-adaptive baseline networks, both in CNNs and vision transformers. For example, the super-nets of our adaptive depth networks, or ResNet50-ADN (FFFF), Vit-b/16-ADN (FFFF), and Swin-T-ADN (FFFF), all achieve 1.2%, 0.2%, and 0.3% higher inference accuracy, respectively, than non-adaptive baselines. This result might seem strange at first since our networks support mulitple sub-networks in a single network. We conjecture that this performance improvement results from the regularization effect of our training method that has similar effect of stochastic depths [32]. For more detailed ablation analysis, readers are referred to [24].

Table 3 also shows that, in both of our adaptive depth CNNs and vision transformers, inference accuracy does not degrade significantly as the depth of the networks gets shallow. For instance, the base-nets of our adaptive depth networks have only 1.45%, 2.2%, and 3.1% lower inference accuracy in

TABLE 3. Performance of adaptive depth networks. Accuracy is measured on ImageNet validation dataset. Accuracy of baseline networks are from the original papers. Latency and energy are measured in Jetson Orin Nano with the batch size of 1.

sub-net	FLOPs	Acc@1 (%)	latency (ms)	energy (J)
FFFF	4.12G	77.34	24.4	0.380
TFFF	3.90G	77.20	23.4	0.357
TTFF	3.46G	76.75	21.7	0.333
TTTF	2.80G	75.60	18.1	0.277
TTTT	2.58G	75.89	15.9	0.247
Baseline [19]	4.12G	76.1	24.4	0.380

(a) ResNet50-ADN

sub-net	FLOPs	Acc@1 (%)	latency (ms)	energy (J)
FFFF	17.58G	81.3	72.9	1.261
TFFF	16.12G	80.5	65.2	1.153
TTFF	14.67G	79.9	59.3	1.050
TTTF	13.21G	79.3	53.5	0.945
TTTT	11.76G	79.1	45.8	0.841
Baseline [30]	17.58G	81.1	72.9	1.261

(b) ViT-b/16-ADN

sub-net	FLOPs	Acc@1 (%)	latency (ms)	energy (J)
FFFF	4.49G	81.8	47.3	0.691
TFFF	4.12G	80.5	42.5	0.608
TTFF	3.75G	79.7	37.3	0.538
TTTT	2.69G	78.7	22.8	0.342
Baseline [31]	4.49G	81.5	47.3	0.691

(c) Swin-T-ADN

TABLE 4. Performance of sub-networks of slimmable ResNet50 [9]. While the FLOPs of slimmable ResNet50 can be scaled by up to 93% (from 4.12 GFLOPs to 0.29 GFLOPs), inference latency is accelerated only by up to 9% (from 24.5ms to 22.5ms).

sub-net	FLOPs	Acc@1 (%)	latency (ms)	energy (J)
x1.00 width	4.12G	76.0	24.5	0.380
x0.75 width	2.35G	74.9	23.0	0.308
x0.50 width	1.07G	72.1	22.9	0.271
x0.25 width	0.29G	65.0	22.5	0.237

ResNet50-ADN, ViT-b/16-ADN, and Switn-T-ADN, respectively. This marginal degradation of inference accuracy is important to exploit the inference acceleration through the adaption of network depth. In Figures 3-(a) and 6, the sub-networks of our adaptive depth networks are compared to counterpart individual networks with the same network depths, or FLOPs.

2) PERFORMANCE ON DEVICES

In the 3rd and 4th columns of Table 3, we show inference latency and energy consumption in Nvidia Jetson Orin Nano. The result shows that our adaptive depth networks have actual inference acceleration and energy saving by scaling down network depths. For instance, scaling down the depth from the Swin-T-ADN (FFFF) to Swin-T-ADN (TTTT) results in about 35% acceleration and energy saving. For practical purposes, actual inference acceleration and energy saving is more important than theoretical reduction of FLOPs. Figures 3-(b) and 7 demonstrate that both the inference latency and energy consumption of our adaptive depth networks are proportional to network depths. In contrast, in many previous adaptive networks, theoretical scaling of FLOPs is not translated to actual inference acceleration.

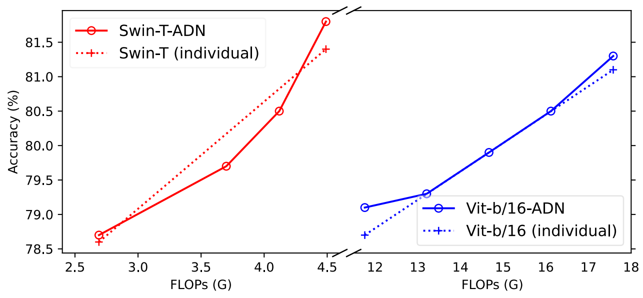


FIGURE 6. Validation accuracy of vision transformers on ImageNet. Each point corresponds to a sub-network with different depths.

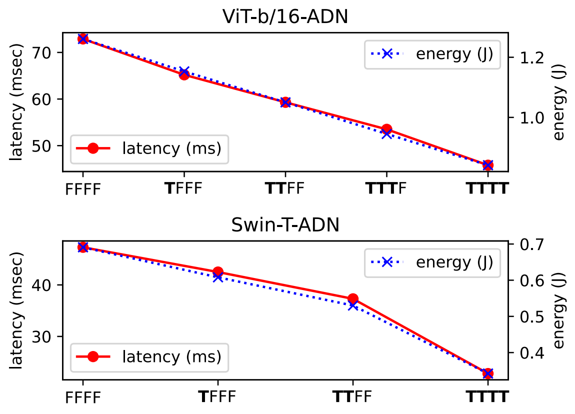


FIGURE 7. Inference energy and latency of sub-networks of our ViT-b/16-ADN and Swin-T-ADN. Measured on Nvidia Jetson Orin Nano with the batch size of 1.

For instance, Table 4 shows the performance of slimmable ResNet50 [9], a representative width adaptation network. It shows that adapting network width has marginal effect on actual inference acceleration. For example, while FLOPs of slimmable ResNet50 decreases by up to 93% from 4.12 GFLOPs to 0.29 GFLOPs, the latency decreases only by up to 9%.

3) LENGTH RATIOS BETWEEN SUB-PATHS

When constructing the proposed adaptive depth networks, one important hyperparameter to consider is the ratio between the lengths of mandatory and skippable sub-paths. In Table 5, we experiment with three distinct ratios of sub-path lengths for ResNet50-ADN to explore the impact of varying these ratios. Figure 8 shows the result. When employing deep mandatory sub-paths, fewer layers are bypassed by sub-networks, resulting in a reduced range for depth scaling. Conversely, when using shallow mandatory sub-paths, sub-networks can skip more layers, leading to an increased range for depth scaling. For instance, in ResNet50-ADN, the FLOPs can be scaled from 4.1 down to 3.0 GFLOPs with deep mandatory sub-paths, while shallow mandatory sub-paths allow scaling down to 1.7 GFLOPs. While employing shallow mandatory sub-paths increases the depth scaling ranges, low inference capability of shallow mandatory sub-paths affects all sub-networks and significantly degrades the performance. The result also shows that deep mandatory sub-paths longer

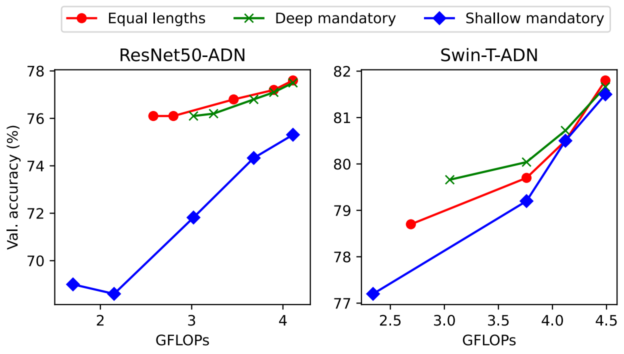


FIGURE 8. The performance of ResNet50-ADNs (left) and Swin-T-ADNs (right) that have different ratios between the mandatory and the skippable sub-paths. Shallow mandatory sub-paths allows more depth scaling at the cost of lower performance.

than half does not yield further improvement in classification performance; instead, it only reduces the range of depth scaling.

This experiment highlights a limitation of our approach. While other adaptive network methods, like width scaling, offer a broader range of FLOPs scaling, our proposed adaptive depth networks have a more restricted range. For instance, in Table 4, the slimmable ResNet50 achieves FLOPs scaling from 4.12 down to 0.29 GFLOPs by adjusting network widths. In contrast, our ResNet50-ADN can only scale FLOPs up to 1.7 GFLOPs when shallow mandatory sub-paths are employed. To achieve additional FLOPs scaling, we may explore synergies with other adaptation dimensions, such as adjusting network widths [10] and input resolutions [15], in combination with our depth adaptation method. We leave this as our future work.

C. EVALUATION OF QOS-AWARE INFERENCES

In this set of experiments, we investigate if the proposed adaptive depth networks combined with the QoS-aware inference accelerator can support effective QoS under unpredictable and dynamically changing environments.

We compare the performance of our approach to several baselines. For example, *ResNet50-ADN* is our adaptive depth ResNet50 whose depth is controlled by the QoS-aware inference accelerator. *ResNet50(FFFF)* and *ResNet50(TTTT)* are the super-nets and the base-nets of ResNet50-ADN, respectively, and their depths are fixed during the experiment. With '*2-ResNets*', we make 2 separate instances of the super-net and the base-net of ResNet50-ADN to support the target inference latency with our QoS controller. Since these 2 separate network instances do not share parameters, additional memory space (about 100 MBytes for ResNet50) is required. In order to demonstrate the generality of our approach, we perform the same experiments for Swin-T-ADN and its baselines.

1) SUPPORTING VARYING TARGET INFERENCE LATENCY

We first investigate the effectiveness of our adaptive depth networks by varying the target inference latency at runtime.

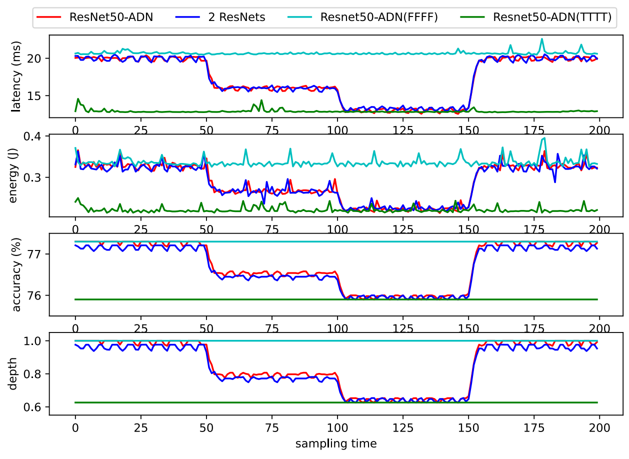
TABLE 5. **The configurations of ResNet50-ADN with different length ratios between the mandatory and the skippable sub-paths.**

		Equal lengths (default)		Deep mandatory sub-paths		Shallow mandatory sub-paths	
	# of total blocks	Mandatory blocks	Skippable blocks	Mandatory blocks	Skippable blocks	Mandatory blocks	Skippable blocks
stage 1	3	2	1	2	1	1	2
stage 2	4	2	2	3	1	1	3
stage 3	6	3	3	4	2	2	4
stage 4	3	2	1	2	1	1	2

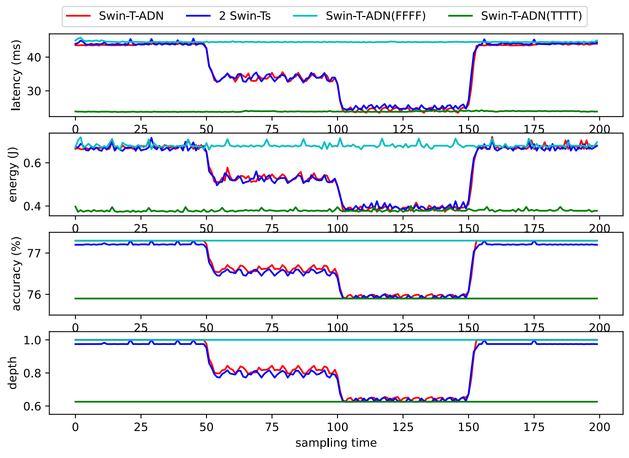
For ResNet50-ADN and its baselines, the target inference latency is varied from 20ms to 16ms, then to 13ms, and back to 20ms at the 50th, 100th, and 150th sampling point, respectively. Our ResNet50-ADN and baselines run continuously to process input images over 200 sampling periods. Similarly, the target inference latency of Swin-T-ADN and its baselines are varied from 44ms to 34ms, then to 24ms, and back to 44ms at the 50th, 100th, and 150th sampling point, respectively.

Figure 9 shows the transient behavior during 200 sampling periods and Figure 10 shows the average performance. The results show that both ResNet50-ADN and Swin-T-ADN support the target inference latency closely with high energy efficiency. In Figure 10, our ResNet50-ADN and Swin-T-ADN consume about 17% and 20% less energy, respectively, than their super-nets while closely supporting the varying target latency. The super-nets maintain the full network depths without skipping any network layers, and, hence, they maintain the original inference accuracy during runtime. However, their inference tardiness is greater than 1.0, implying that they miss the target inference latency. In contrast, the base-nets consume the least energy by maintaining the minimum depth of the networks. For example, the base-net of ResNet50-ADN consumes about 52% less energy than the super-net. However, the base-net of ResNet50-ADN reduces the network depth unnecessarily and this results in 1.3% loss of inference accuracy.

Our adaptive depth networks leverage both high accuracy super-nets and energy-efficient base-nets from a single network according to the control signal from the QoS controller that determines the ratio of two sub-networks. For example, in Figure 9-(a), ResNet50-ADN reduces network depth and inference accuracy only when the shorter inference latency is requested. For example, between the 100th and the 150th sampling times, ResNet50-ADN reduces the network depth by approximately 37% to support the reduced target inference latency of 13ms. However, once the target latency is raised to 20ms at the 150th sampling point, the network depth and resulting inference accuracy are promptly restored within 5 sampling periods.



(a) For ResNet50, the target latency is changed from 20 ms to 16 ms, then to 13 ms, and back to 20 ms.



(b) The target latency for Swin-T is changed sequentially from 44 ms to 34 ms, then to 24 ms, and finally back to 44 ms.

FIGURE 9. **Dynamic response of adaptive and non-adaptive networks while the target inference latency is varied at the 50th, 100th, and 150th sampling points. The depths of our adaptive depth neural networks are controlled continuously by the QoS-aware inference accelerator.**

2) ROBUSTNESS UNDER UNPREDICTABLE WORKLOADS

In this experiment, we investigate if our approach can support the target inference latency under unpredictable workloads. During the experiment of both ResNet50-ADN and Swin-T-ADN, an inference task runs continuously for 150 sampling periods to process input images. The target inference latency is set to 20ms and 41ms, respectively, for ResNet50-ADN and Swin-T-ADN. At the 50th sampling point, we activate a distrurbance task that multiplies two 400 $\times$ 400 matrices continuously for 50 sampling periods. Since both the inference task and the disturbance task share a single GPU device, they interfere each other, potentially resulting in delays in inference tasks.

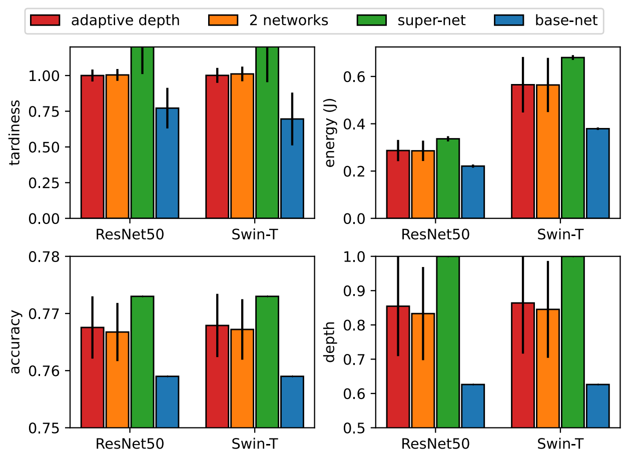
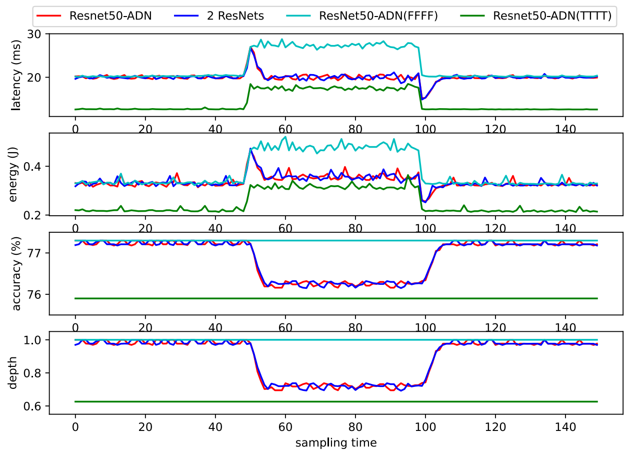
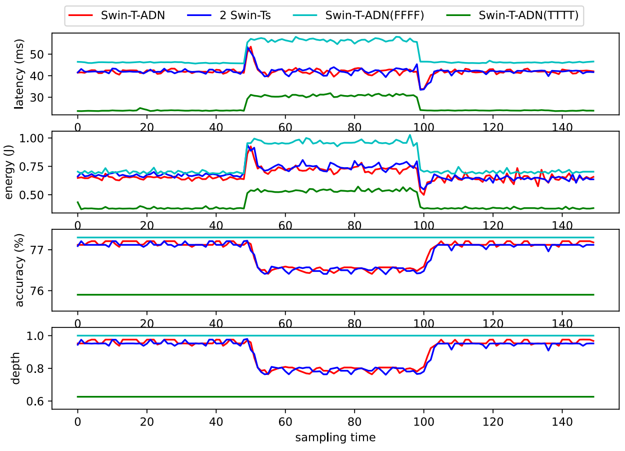


FIGURE 10. Average performance while the target inference is varied during 200 sampling periods. Tardiness 1 with tight error bounds implies that the desired latency is closely supported.



(a) For ResNet50, the target inference latency is set to 20 ms.



(b) For Swin-T, the target inference latency is set to 41ms.

FIGURE 11. Robustness against disturbance. For both ResNet50 and Swin-T, disturbance workloads are injected at the 50th sampling time and lasts for 50 sampling periods.

Figure 11 shows the transient behavior of ResNet50-ADN and Swin-T-ADN and their baselines. Figure 12 shows the average performance during the experiment. The results show that both ResNet50-ADN and Swin-T-ADN support the target inference latency even in the presense of the disturbance task. For example, in Figure 11-(a), the latency of ResNet50-ADN increases by 35% due to the sudden injection of interfering workloads. However, the latency is stabilized back to the target within 5 sampling periods because the QoS controller actively decreases the network depth by approximately 20%. In contrast, the inference latency of non-adaptive ResNet50 is increased suddenly by approximately 37%, resulting in deadline misses during the presence of the disturbance task. In real-world scenarios, missing these deadlines can result in serious consequences, such as tardy reactions to traffic situations in autonomous vehicles. In order to prevent deadline misses, we might consider using networks that have low average inference latency. However, due to the non-adaptive nature of such efficient networks, its low inference accuracy cannot be improved even if the disturbance task disappears.

VI. RELATED WORK

Adaptive neural networks have the ability of providing several sub-networks of various computational capabilities from a single neural network. There have been significant efforts to develop adaptive neural networks by exploiting the inherent redundancy of neural networks. Several dimension of neural networks have been considered for architectural adaptation, including network widths [9], [10], [11], [14], depths, [12], [13], [33], [34], [35], input resolutions [15], [17], and some combination of them. For instance, slimmable neural networks [9] can select a sub-network of 4 different widths from a single trained neural network. Recently, some adaptive depth networks have been proposed for transformer-based language models [11], [12]. However, most of these adaptive networks exploit iterative self-distillation to train sub-networks, requiring significantly longer training time than their counterpart non-adaptive networks. Further, theoretic reduction of computation via network adaption does not always lead to actual inference acceleration as demonstrated in Table 4. *Dynamic networks* [16] is another class of adaptive networks that can adapt network depths, width, or tokens of CNNs [15], [36], [37], [38], [39] and transformers [18], [40], [41], [42] in an input dependent manner. For example, dynamic depth networks [15], [36], [37], [38], [39] exploit auxiliary decision networks to determine if a layer can be skipped for a given input. However, the policies learned for the decision networks are opaque to users, and, hence, they cannot be used for predictable adaptation to meet the resource conditions.

Various hardware- and software-based inference accelerators have been developed to run trained neural networks efficiently on target devices [43], [44]. While hardware-based accelerators try to maximize the throughput of deep learning operations on specialized hardware [45], software-based accelerators mainly focus on optimizing

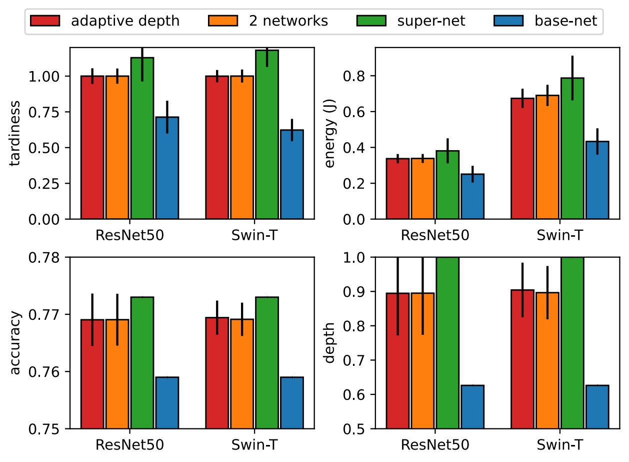


FIGURE 12. **Average performance for 150 sampling periods. Disturbance workloads are activated for 50 sampling periods. Tardiness 1 with tight error bounds implies that our adaptive depth networks closely support the desired inference latency.**

resource management, pipeline design, model restructuring, and quantization [46], [47], [48], [49], [50]. For example, DeepX [48] decomposes network layers and executes them by leveraging a mixture of heterogeneous processors. TensorRT [47] provides various runtime optimization such as quantizing floating-point numbers into integers, and combining several layers into a single layer for low latency. While most hardware/software inference accelerators provide significant gains in efficiency, they are not aware of performance requirements from applications and they cannot adapt when the operating environments change. To address this problems, there is a set of works that propose software-based accelerators to support predictable inference performance [51], [52], [53]. For example, DeepRT [51] support predictable inference latency and energy consumption by controlling the speed of CPU and GPU devices using a MIMO (multiple inputs/multiple outputs) controller. DMS [52] exploits a feedback controller that dynamically prunes filters of convolution networks at runtime to support target inference latency and energy consumption. However, in DMS, the model scaling through pruning filters results in significant loss of inference accuracy.

VII. CONCLUSION

In this paper, we introduce a holistic approach to providing predictable inference performance for deep learning applications even in the presence of external disturbances. Our approach comprises two major components: adaptive depth neural networks and the QoS-aware inference accelerator. The proposed adaptive depth neural networks, exploiting a novel architectural pattern and the training method for depth adaptation, can scale computation instantly with marginal impact on accuracy. We demonstrate that the proposed adaptive depth networks outperform non-adaptive counterpart networks and can provide actual inference acceleration by adapting network depth, achieving up to 38% acceleration with a marginal accuracy loss of 1.5%. On target devices, the proposed QoS-aware inference accelerator dynamically control the depth of these adaptive depth networks using a feedback control loop to support the desired inference latency. We present the performance of the QoS-aware inference accelerator using an experimental prototype and various adaptive depth networks. Evaluation results demonstrate that our approach can make effective use of constrained resources while supporting the desired inference latency for unpredictable workloads by dynamically switching between high-accuracy and high-efficiency sub-networks.

In our future work, we plan to enhance this research in several directions. Firstly, we will investigate methods to achieve additional FLOPs scaling while minimizing performance degradation. This exploration may involve leveraging other adaptation dimensions, such as network widths and input resolutions, in conjunction with our depth adaptation method. Secondly, we plan to apply our QoS-aware inference framework to practical deep learning tasks, particularly object detection for autonomous vehicles. Given the dynamic nature of autonomous vehicle environments, including varying computational requirements, we will explore how our QoS-aware inference acceleration can ensure robust performance under unpredictable environments.

REFERENCES

[1] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural network," in *Proc. Adv. Neural Inf. Process. Syst.*, Dec. 2015, pp. 1135–1143.

[2] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," in *Proc. Int. Conf. Learn. Represent.*, 2017.

[3] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015, *arXiv:1503.02531*.

[4] L. Beyer, X. Zhai, A. Royer, L. Markeeva, R. Anil, and A. Kolesnikov, "Knowledge distillation: A good teacher is patient and consistent," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2022, pp. 10915–10924.

[5] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*.

[6] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 6848–6856.

[7] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 779–788.

[8] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 1137–1149, Jun. 2017.

[9] J. Yu, L. Yang, N. Xu, J. Yang, and T. Huang, "Slimmable neural networks," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2018.

[10] D. Wang, C. Gong, M. Li, Q. Liu, and V. Chandra, "AlphaNet: Improved training of supernets with alpha-divergence," in *Proc. Int. Conf. Mach. Learn. (ICML)*, in Proceedings of Machine Learning Research, vol. 139, M. Meila and T. Zhang, Eds., Jul. 2021, pp. 10760–10771.

[11] L. Hou, Z. Huang, L. Shang, X. Jiang, X. Chen, and Q. Liu, "DynaBERT: Dynamic BERT with adaptive width and depth," in *Proc. Conf. Neural Inf. Process. Syst. (NeurIPS)*, vol. 33, 2020, pp. 9782–9793.

[12] A. Fan, E. Grave, and A. Joulin, "Reducing transformer depth on demand with structured dropout," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2020.

[13] G. Huang, D. Chen, T. Li, F. Wu, L. van der Maaten, and K. Weinberger, "Multi-scale dense networks for resource efficient image classification," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2018.

[14] H. Li, H. Zhang, X. Qi, Y. Ruigang, and G. Huang, "Improved techniques for training adaptive deep networks," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 1891–1900.
[15] L. Yang, Y. Han, X. Chen, S. Song, J. Dai, and G. Huang, "Resolution adaptive networks for efficient inference," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 2366–2375.
[16] Y. Han, G. Huang, S. Song, L. Yang, H. Wang, and Y. Wang, "Dynamic neural networks: A survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 11, pp. 7436–7456, Nov. 2022.
[17] L. Beyer, P. Izmailov, A. Kolesnikov, M. Caron, S. Kornblith, X. Zhai, M. Minderer, M. Tschannen, I. Alabdulmohsin, and F. Pavetic, "FlexiViT: One model for all patch sizes," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2023, pp. 14496–14506.
[18] M. Fayyaz, S. A. Koohpayegani, F. R. Jafari, S. Sengupta, H. R. V. Joze, E. Sommerlade, H. Pirsiavash, and J. Gall, "Adaptive token sampling for efficient vision transformers," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, S. Avidan, G. Brostow, M. Cissé, G. M. Farinella, and T. Hassner, Eds., 2022, pp. 396–414.
[19] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.
[20] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. U. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. Conf. Neural Inf. Process. Syst. (NeurIPS)*, vol. 30, 2017.
[21] S. Jastrzebski, D. Arpit, N. Ballas, V. Verma, T. Che, and Y. Bengio, "Residual connections encourage iterative inference," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2018.
[22] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. 13th Int. Conf. Learn. Represent. (ICLR)*, 2015.
[23] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 448–456.
[24] W. Kang, "Adaptive depth networks with skippable sub-paths," 2023, *arXiv:2312.16392*.
[25] L. Ljung, "System identification," in *Signal Analysis and Prediction*. London, U.K.: Springer, 1998, pp. 163–173.
[26] P. K. Janert, *Feedback Control for Computer Systems: Introducing Control Theory to Enterprise Programmers*. O'Reilly Media, 2013.
[27] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, *Feedback Control of Computing Systems*. Hoboken, NJ, USA: Wiley, 2004.
[28] (2023). *Image Classification Reference Training Scripts in PyTorch*. [Online]. Available: https://github.com/pytorch/vision/tree/main/references/classification
[29] (2023). *NVIDIA Jetson Orin Nano*. [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/
[30] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16 × 16 words: Transformers for image recognition at scale," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2021.
[31] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin Transformer: Hierarchical vision transformer using shifted windows," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2021, pp. 9992–10002.
[32] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger, "Deep networks with stochastic depth," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*. Cham, Switzerland: Springer, 2016, pp. 646–661.
[33] H. Hu, D. Dey, M. Hebert, and J. A. Bagnell, "Learning anytime predictions in neural networks via adaptive loss balancing," in *Proc. AAAI Conf. Artif. Intell.*, 2019, vol. 33, no. 1, pp. 3812–3821.
[34] L. Zhang, J. Song, A. Gao, J. Chen, C. Bao, and K. Ma, "Be your own teacher: Improve the performance of convolutional neural networks via self distillation," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 3712–3721.
[35] C. Wan, H. Hoffmann, S. Lu, and M. Maire, "Orthogonalized SGD and nested architectures for anytime neural networks," in *Proc. Conf. Mach. Learn.*, 2020, pp. 9807–9817.
[36] Z. Wu, T. Nagarajan, A. Kumar, S. Rennie, L. S. Davis, K. Grauman, and R. Feris, "BlockDrop: Dynamic inference paths in residual networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 8817–8826.
[37] C. Li, G. Wang, B. Wang, X. Liang, Z. Li, and X. Chang, "Dynamic slimmable network," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 8603–8613.
[38] Q. Guo, Z. Yu, Y. Wu, D. Liang, H. Qin, and J. Yan, "Dynamic recursive neural network," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 5142–5151.
[39] Y. Li, L. Song, Y. Chen, Z. Li, X. Zhang, X. Wang, and J. Sun, "Learning dynamic routing for semantic segmentation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 8550–8559.
[40] L. Meng, H. Li, B.-C. Chen, S. Lan, Z. Wu, Y.-G. Jiang, and S.-N. Lim, "AdaViT: Adaptive vision transformers for efficient image recognition," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2022, pp. 12299–12308.
[41] H. Yin, A. Vahdat, J. M. Alvarez, A. Mallya, J. Kautz, and P. Molchanov, "A-ViT: Adaptive tokens for efficient vision transformer," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2022, pp. 10799–10808.
[42] B. Heo, S. Yun, D. Han, S. Chun, J. Choe, and S. J. Oh, "Rethinking spatial dimensions of vision transformers," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2021, pp. 11916–11925.
[43] P. Dhilleswararao, S. Boppu, M. S. Manikandan, and L. R. Cenkeramaddi, "Efficient hardware architectures for accelerating deep neural networks: Survey," *IEEE Access*, vol. 10, pp. 131788–131828, 2022.
[44] D. Xu, T. Li, Y. Li, X. Su, S. Tarkoma, T. Jiang, J. Crowcroft, and P. Hui, "Edge intelligence: Empowering intelligence to the edge of network," *Proc. IEEE*, vol. 109, no. 11, pp. 1778–1837, Nov. 2021.
[45] K. Seshadri, B. Akin, J. Laudon, R. Narayanaswami, and A. Yazdanbakhsh, "An evaluation of edge TPU accelerators for convolutional neural networks," 2021, *arXiv:2102.10423*.
[46] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and Huffman coding," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2016.
[47] (2023). *NVIDIA TensorRT*. [Online]. Available: https://developer.nvidia.com/tensorrt
[48] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, L. Jiao, L. Qendro, and F. Kawsar, "DeepX: A software accelerator for low-power deep learning inference on mobile devices," in *Proc. 15th ACM/IEEE Int. Conf. Inf. Process. Sensor Netw. (IPSN)*, Apr. 2016, pp. 1–12.
[49] (2023). *ONNX Runtime: Cross-Platform Accelerated Machine Learning*. [Online]. Available: https://onnxruntime.ai
[50] B. Fang, X. Zeng, and M. Zhang, "NestDNN: Resource-aware multi-tenant on-device deep learning for continuous mobile vision," in *Proc. 24th Annu. Int. Conf. Mobile Comput. Netw.* New York, NY, USA: Association for Computing Machinery, Oct. 2018, pp. 115–127, doi: 10.1145/3241539.3241559.
[51] W. Kang and J. Chung, "DeepRT: Predictable deep learning inference for cyber-physical systems," *Real-Time Syst.*, vol. 55, no. 1, pp. 106–135, Jan. 2019.
[52] W. Kang, D. Kim, and J. Park, "DMS: Dynamic model scaling for quality-aware deep learning inference in mobile and embedded devices," *IEEE Access*, vol. 7, pp. 168048–168059, 2019.
[53] W. Kang and J. Chung, "Power- and time-aware deep learning inference for mobile embedded devices," *IEEE Access*, vol. 7, pp. 3778–3789, 2019.

WOOCHUL KANG (Member, IEEE) received the Ph.D. degree in computer science from the University of Virginia, in 2009. He was a Senior Researcher with the Electronics and Telecommunications Research Institute, South Korea, from 2000 to 2004 and from 2009 to 2012, and a Postdoctoral Research Associate with the University of Illinois at Urbana–Champaign, USA, from 2012 to 2013. Since 2013, he has been a Professor with the Department of Embedded Systems Engineering, Incheon National University, Incheon, South Korea. His research interests include deep learning for edge devices, real-time systems, distributed middleware, and feedback control of computing systems.