Dynamically Reconfigurable Perception using Dataflow Parameterization of Channel Attention

Dynamically Reconfigurable Perception using Dataflow Parameterization of Channel Attention

Yujunrong Ma[1], Kshitij Nikhal[2], Jani Boutellier[3], Benjamin Riggan[4] and Shuvra S. Bhattacharyya[5]

***Abstract*— Despite the increasing performance of deep learning models for intelligent perception tasks, the size and complexity of state-of-the-art models have become increasingly large, which makes them difficult to deploy in resource-constrained edge computing environments. Moreover, the conditions and constraints under which these models operate may be impossible to fully anticipate at design time, and may undergo significant changes during inference. In this paper, we address these challenges by developing a new parameterized design approach for image-based perception that enables efficient and dynamic reconfiguration of convolutions using channel attention. Compared to switching among sets of multiple complete neural network models, the proposed reconfiguration approach is much more streamlined in terms of resource requirements, while providing a high level of adaptability to handle unpredictable and dynamically-varying operational scenarios. The efficiency and adaptability of the proposed approach are demonstrated on ResNet50 integrated with the Convolutional Block Attention Module (CBAM) concept.**

***Index Terms*— Dataflow, channel-attention, model pruning.**

I. Introduction

Deep Neural Network (DNN) models have dominated many machine learning applications in recent years. However, the increasing size of DNN models brings about major challenges for model deployment on resource-constrained devices. The convolution operator is one of the most general and powerful tools to extract features from multiple kinds of input. Increasingly large convolutional neural networks (CNNs) are employed to extract increasingly complex and high-level features in the input data [1], [2]. As networks become deeper, the kernels can be optimized to produce relatively larger activation responses when presented patterns corresponding to larger receptive fields (i.e., more effective pixels) and high frequency details.

Convolution layers are widely favored in the field of machine learning due to their remarkable suitability for analyzing data structured in grid-like or sequential patterns, as seen in images and audio. Despite the impressive expressive power of CNNs, for a modern DNN, such as ResNet50 [3], the convolutional layers account for over 80% of the total computation cost. The massive computation requirement hampers inference on edge devices due to limitations in memory and energy capacity. In this paper, we seek to address these problems by developing a new parameterized design approach for deep learning models that enables efficient and dynamic reconfiguration of convolutions using channel attention. Compared to switching among sets of multiple complete neural network models, the reconfiguration approach developed in this paper is much more streamlined in terms of resource requirements, while providing a high level of adaptability to handle unpredictable and dynamically-varying operational scenarios.

Channel attention is applied to CNNs to selectively emphasize specific channels in feature maps from the network's intermediate representations. The attention is designed to help the network focus on meaningful features during the learning process, enhancing its performance. In this paper, we use attention as an indicator to guide model pruning. In this context, a smaller channel attention score implies a poorer contribution to the final output from the given channel, and a greater priority for removing the channel during inference under resource-constrained operation.

Model pruning and hardware acceleration are two major methods to solve efficiency problems in DNN systems. Dataflow modeling approaches can help with both aspects [4], [5]. In this paper, we introduce a novel dataflow modeling approach for integration of channel attention into DNN design optimization. In particular, we apply dynamic dataflow methods to manage the intermediate outputs in pre-trained DNN models. This approach allows for rapid adjustment of different model variants—having different operational trade-offs—without the need for changing overall model architecture or retraining. Since the objective is to enable rapid adaptation to changing application requirements, architecture modification and retraining have prohibitive overhead (due to inefficient resource usage); our dataflow-based adaptation avoids such overhead.

The conditions and constraints under which DNN models operate may be impossible to fully anticipate at design time, and may undergo significant changes at run-time. This paper develops integrated methods involving channel attention and dataflow modeling to control the adaptation of DNN models in a manner that provides optimized performance subject to dynamic changes in operational conditions and constraints. The methods that we develop have the following advantages:

- efficient adaptation in response to dynamically-varying application requirements or to carry out a pre-defined schedule of different operating modes;

[1]Yujunrong Ma is with Department of Electrical and Computer Engineering, University of Maryland, USA. mayu1996@umd.edu

[2]Kshitij Nikhal is with Department of Electrical and Computer Engineering, University of Nebraska-Lincoln, USA. knikhal2@huskers.unl.edu

[3]Jani Boutellier is with School of Technology and Innovation, University of Vaasa, Finland. jani.boutellier@uwasa.fi

[4]Benjamin Riggan is with Department of Electrical and Computer Engineering, University of Nebraska-Lincoln, USA. briggan2@unl.edu

[5]Shuvra S. Bhattacharyya is with Department of Electrical and Computer Engineering, and Institute for Advanced Computer Studies, University of Maryland, College Park, USA. ssb@umd.edu

- utilizing variable rate dataflow methods to systematically integrate channel attention with DNN functionality, which enables efficient and reliable adaptation of model inference;
- integration of variable-rate dataflow modeling with the Open Neural Network Exchange (ONNX) [6], which is a widely-used open-source tool for representing and exchanging deep learning models across different machine learning frameworks and platforms.

The remainder of this paper is organized as follows. In Section II, we survey literature that is related to channel pruning and dataflow modeling of DNN models. In Section III, we propose the method to prune ResNet + CBAM (Convolutional Block Attention Module) models based on channel attention. In Section IV, we conduct a series of experiments to analyze the performance of our proposed pruning methods, and demonstrate their effectiveness. We also discuss limitations of the proposed methods in Section IV. In Section V, we summarize our findings in our paper, and present useful directions for future work.

II. Related Work and Background

In this section, we summarize recent works about the development of channel attention techniques, how such techniques help with model pruning, and general methods to work with DNN models as dataflow graphs.

A. Channel Attention

The original aim of channel attention is to assign relevance scores to channels within convolutional layers, and facilitate the exploration of inter-channel relationships [7]. Early channel attention approaches were used in so-called Squeeze-and-Excitation (SE) Networks, which employed global average-pooled features to compute channel-wise attention [8].

Woo et al., in their development of the CBAM [9] concept, advanced channel attention methods by introducing both spatial and channel attention, thus enhancing the expressive power of attention mechanisms within CNNs. They also explored the combination of average pooling and max pooling to aggregate features. Building on these developments, Wang introduced Efficient Channel Attention (ECA) [10], which provides a local cross-channel interaction strategy without dimensionality reduction. ECA has a lower computational cost compared to other attention mechanisms.

Additionally, Misra et al. [11] explored triplet attention, which captures the importance of features across dimensions, for example channel attention and spatial attention, in a tensor. Qin et al. [12] reevaluated the extraction of global information, emphasizing compression as a pivotal perspective. Their analysis focused on the frequency domain, especially global average pooling. They demonstrated that global average pooling could be understood as a specific instance of the discrete cosine transform (DCT).

B. Pruning CNN Channels

Channel pruning is a vital technique to reduce the computational and memory requirements of a CNN without significantly sacrificing its performance. Liu introduced the concept of *network slimming* [13], which hinges on the application of L1-norm regularization to scaling factors in batch normalization layers, enabling the precise pruning of channels from deep, pre-trained CNNs.

Building upon this foundation, Liu et al. designed the Channel Pruning guided by Spatial and Channel Attention (CPSCA) approach [14], which attains better inference accuracy than other state-of-the-art pruning methods at similar pruning ratios. Their technique relies on channel scale, a statistical metric derived from element-wise averaging of the channel attention map to measure the channel importance.

Unlike other methods that rely solely on the CNN output for channel pruning, CNN interpretability theory involves guiding channel pruning through mask layers [15]. A mask layer refers to a CNN layer that applies a binary mask to the feature maps of a CNN. This mask selectively turns off certain channels, effectively allowing the network to focus on more relevant features for its tasks. A class-wise mask can be derived for each channel to configure the contribution of the channel in identifying each class in a multi-class classification system.

Additionally, Chen designed a channel-wise gate to dynamically estimate the conditional accuracy change due to a given pruning operation [16]. This estimation approach was used to incrementally prune channels during the training process. The dynamic channel pruning strategy developed by Chen facilitates the formation of a stable network structure during training so that a separate fine-tuning process is not needed after pruning.

C. Dataflow-based Modeling of DNNs

In this work, we apply dataflow as a methodology for modeling and optimization of DNN applications. For background on dataflow-based methods for signal and information processing (SIP) applications, see [17], [18]. In this context, a dataflow graph (DFG) provides a representation of a SIP application as a directed graph. Vertices in the graph, called *actors*, represent functional modules of arbitrary complexity, and edges represent queues that buffer data as it passes from the output of one actor to the input of another.

To model a DNN with a DFG [19], an important issue is the *granularity* of the dataflow representation — in particular, the complexity of the DNN modules that are represented by individual actors. A common approach is to encapsulate each DNN layer as a distinct actor (e.g., see [4], [20]).

To address the coupling of channels within multi-branch neural networks, Narshana et al. propose the Backwards Graph-based Saliency Computation (BGSC) algorithm [21]. This algorithm jointly considers coupled elements of Distributed Feature Channels (DFCs) to determine their saliencies. The saliencies in turn serve as crucial metrics for determining which parts of the network to prune, leading to modifications in the data flow graphs of the associated subnetworks.

Various methods have been developed that apply dataflow concepts to hardware acceleration of DNNs. For example,

a hardware design called sparse periodic systolic (SPS) dataflow prunes DNN models in a way that results in periodic pattern-based sparsity [22]. The approach is shown to lower latency and power dissipation of CNNs on FPGA devices. In the realm of sparse dataflow mapping for various types of DNN operators, such as convolutions and recurrent layers, Kwon et al. have developed a strategy that involves enhancing multipliers and adders with miniature switches and interconnecting them via a novel reconfigurable interconnect capable of supporting arbitrarily-sized neurons [23]. The Efficient Inference Engine (EIE) [24] is an accelerator that performs customized sparse matrix-vector multiplication and handles weight sharing with no loss of efficiency. EIE capitalizes on dynamic input vector sparsity, static weight sparsity, relative indexing, weight sharing, and very narrow weights (4 bits).

A variety of methods have been developed to incorporate dynamic communication patterns between actors into dataflow modeling. Such work is relevant to our objective of supporting adaptive models whose dataflows can be modified dynamically to streamline their operation in relation to run-time changes in an application's operating requirements. For example, Variable Rate Dataflow (VRDF) is a form of dynamic dataflow that facilitates dynamic workload balancing in real-time applications [25]. Methods have been developed for efficient memory management of VRDF-based implementations [25].

VR-PRUNE [26] is dataflow modeling method together with a software framework that incorporates support for dynamic dataflow. The VR-PRUNE framework has been developed with a distinct emphasis on supporting high-performance processing on heterogeneous platforms. A distinguishing characteristic of VR-PRUNE modeling is its enforcement of certain kinds of symmetry in dynamic dataflow behavior. Such symmetry properties are satisfied in a wide variety of useful SIP applications. These properties lead to possibilities for more efficient implementation, and are useful to represent explicitly in the model when they are satisfied.

In this paper, we build upon VR-PRUNE modeling concepts to provide network adaptation capabilities based on channel attention. Moreover, we introduce ONNX compatibility in the VR-PRUNE software framework, which enables execution of ONNX DNN models that incorporate our novel methods for adaptive, attention-based network implementation.

III. Methods

In our approach to attention-guided model adaptation, we represent the given DNN as a dataflow graph G. Initially, each layer in the DNN is modeled as a distinct actor in G. However, at design time, we transform G by selecting a subset of layers Ω, and replacing the actor $A(L)$ associated with each layer $Lin\Omega$ by a subgraph consisting of layers $\delta(L,1), \delta(L,2), \ldots, \delta(L, lcount(L))$, where $lcount(L)$ gives the number of layers in the decomposition of L, and each layer $\delta(L,i)$ is again encapsulated by a distinct actor, which we denote by $A(\delta(L,i))$.

Intuitively, the transformation process described above can be viewed as a decomposition of each layer in Ω into a set of lower-complexity layers, which we refer to as *sublayers* of the original graph G. We denote the transformed version of G, with all layers in Ω replaced by their corresponding sublayers, as $T(G)$. In our approach to adaptive, attention guided DNN design, we further transform $T(G)$ by integrating dataflow structures that can selectively disable subsets of sublayers throughout $T(G)$ based on dynamically-varying input conditions or operating constraints that are monitored at run-time. This second phase of transformation results in a new, dynamic dataflow graph denoted $U(G)$, which is the final graph that is to be employed in the targeted deployment.

In the remainder of this section, we elaborate on details of the process of deriving $U(G)$ from G.

In this paper, we consider only multilayer perceptron (MLP) layers and convolutional layers as candidates for decomposition in the derivation of $T(G)$ from G. However, we envision that the approach can be extended to include other types of layers in the decomposition process. Indeed, extension to other types of layers in an interesting direction for future work.

The methods of this section are demonstrated by applying them to CBAM + ResNet50 as the network backbone (i.e., as the original DNN G). Experimental results from our demonstration using CBAM + ResNet50 are presented in Section IV. The methods of this section are general and can be applied to a wide variety of other DNNs; they are not limited to the system that we use in our experiments.

A. Restrictions on the Decomposition Process

In our design method, the decomposition of each layer in Ω is performed in a channel-wise fashion. As described previously, the decomposition process is restricted to two types of layers — MLP and convolution. We impose the following additional restrictions in the decomposition process.

1) The decomposition is performed such that the sublayers derived from a given layer $L \in \Omega$ are all layers of the same type as L.

2) The decomposition is performed such that $lcount(L)$ is the same for all $L \in \Omega$. That is, there is a positive-integer parameter γ such that $lcount(L) = \gamma$ for all $L \in \Omega$.

3) In the channel-wise decomposition of a given layer $L \in \Omega$, the number of input channels that is assigned to each sublayer is the same for all sublayers of the transformed network $T(G)$. We denote this constant number of channels per sublayer as χ. Thus, for all pairs (L,i), the number of channels in $\delta(L,i)$ is χ.

4) For convolutional layers that are decomposed, the number of output channels that is assigned to each sublayer is also the same for all sublayers of the transformed network $T(G)$. We denote this constant number of output channels per convolutional sublayer as β. Thus, for all convolutional pairs (L,i), the number of output channels in $\delta(L,i)$ is β.

5) For a given layer $L \in \Omega$, the partitioning of channels into sublayers is performed sequentially, based on the way in which the channels are ordered. That is, the first χ channels

are grouped into the same sublayer, channels indexed from $(\chi + 1)$ to 2χ (assuming that the indexing starts at 1) are grouped into another sublayer, and so on.

The five restrictions listed above are imposed in the developments of this paper to help simplify the process of constructing decomposed implementations. However, we assume that all five of the restrictions can be removed to significantly enlarge the design space of the decomposition process. Generalizing the methods of this paper to work when some or all of the above-listed restrictions are eliminated is an interesting direction for future work.

B. Decomposing MLP Layers

In the approach developed in this paper, the parameters Ω, γ, and χ are assumed to be specified by the designer. They could, for example, be derived through a generalized hyperparameter tuning process of the network. More direct methods for optimizing these parameters is another interesting direction for further study.

Suppose we are decomposing a layer J having type MLP. Let $w = (w[0], w[1], \ldots, w[n-1])$ denote the linear weight vector of the MLP, b denote the bias, and n denote the total number of channels ($n = \chi \times \gamma$). Then the computation for layer J can be expressed as

$$MLP(x) = b + \sum_{i=1}^{n} x[i]w[i], \tag{1}$$

where $x = (x[1], x[2], \ldots, x[n])$ is the input frame.

In our approach, this computation is decomposed as the sum of γ smaller MLPs, which correspond to the sublayers. For $j = 1, 2, \ldots, \gamma$, the computation for sublayer $\delta(J, j)$ can be expressed, using a minor abuse of notation, as

$$MLP_{J,j}(x) = \frac{b}{\gamma} + \sum_{i=(j-1)\chi+1}^{j\chi} x[i]w[i]. \tag{2}$$

In the decomposed form, the MLP for layer J is computed as

$$MLP(x) = \sum_{i=1}^{\gamma} MLP_{J,i}(x). \tag{3}$$

An illustration of a decomposed MLP layer with $\gamma = 3$ is given in Fig. 1.

As shown in Fig. 1, the regular input and MLP layer L are evenly divided into 3 groups, and the output of L is the summation of the 3 outputs from the groups.

C. Decomposing Convolutional Layers

The decomposition process for convolutional layers proceeds in a manner similar to the process described in Section III-B for MLP layers. Consider a convolutional layer M in G, and suppose that M operates on a an input X with dimensions $C \times H \times W$, and produces an output with dimensions $K \times H_{out} \times W_{out}$. Here, C and K give the number of channels in the input and the number of filters in the output, respectively, and H, H_{out}, W and W_{out} give the height and width of the input and output to M.

The original 2D kernel Ker is of size $K \times C \times p \times q$), and moves over the input data, performing element-wise multiplication. Here, p and q are additional kernel-specific parameters. To decompose the convolution operation, we construct the sublayers from kernels $Ker_1, Ker_2, \ldots, Ker_\gamma$, where each Ker_i is of size $\beta \times \chi \times p \times q$). A decomposed convolutional layer is illustrated in Fig. 2.

As shown in Fig. 2, the kernel is separated into 3 components in the channel dimension, where each component encapsulates $C/\gamma = C/3$. channels. The separated kernels share the same stride and other parameters, but the learned weights are different. The final output of the decomposed layer is a concatenation of the outputs from the three layers instead of the summation.

D. Decomposition Algorithm

Algorithm 1 sketches the process by which the given DNN model G is transformed into its decomposed form $T(G)$, and then further transformed by disabling selected sublayers. The selective disabling is performed to trade-off inference accuracy and speed based on guidance from the system designer. The guidance is provided throughout the input τ to the algorithm. The designer can experiment with different values of τ for a given deployment to select a setting that achieves the most favorable trade-off in relation to the constraints and requirements of the deployment. In Algorithm 1, $|\Omega|$ represents the number of elements in the set Ω.

In Algorithm 1, the transformation process is optimized using training and validation datasets D_t and D_v that are supplied as inputs.

The transformed graph U, which is produced by Algorithm 1, can be applied in an adaptive inference system by augmenting it with functionality to dynamically monitor attention scores and adjust the set of disabled sublayers based on recently-measured attention values. In such a deployment, U can be viewed as the initial model that is deployed, and forms the baseline for subsequent adaptations. In Section III-E, we discuss further techniques that can be used to employ our proposed network decomposition technique in adaptive deployments.

E. Dataflow Mapping

DNN models that are decomposed using the methods described in this section can be represented naturally using dataflow concepts. Such dataflow representations are useful for deriving efficient implementations of the decomposed models and enabling the dynamic disabling of selected sublayers in the models. The dynamic selection of disabled sublayers can be used to systematically trade off interference speed and accuracy based on time-varying operational requirements or resource availability.

To incorporate dataflow modeling with the decomposition methods developed in this paper, we apply the VR-PRUNE model of computation [26]. In VR-PRUNE, as in other forms

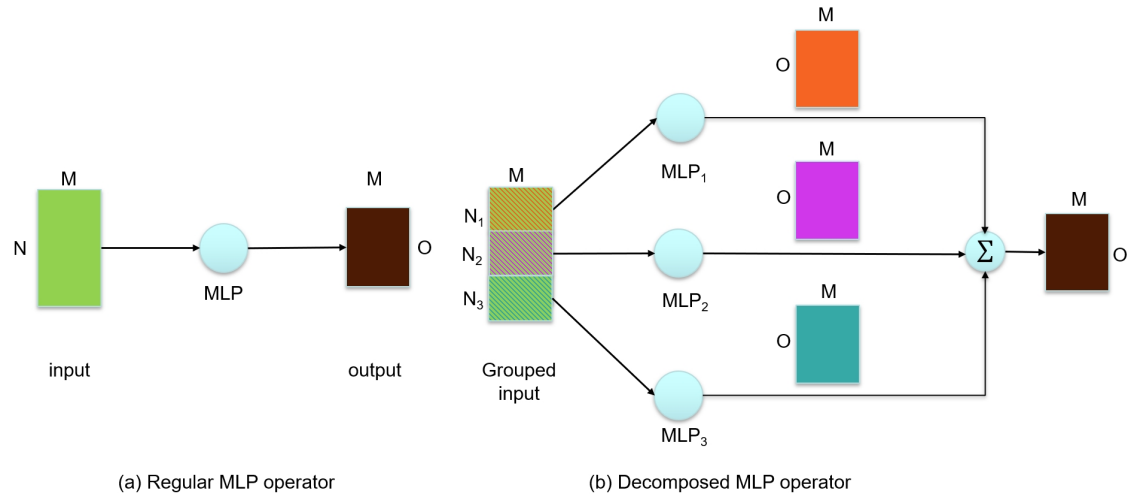

Fig. 1. Illustration of a decomposed MLP operator.

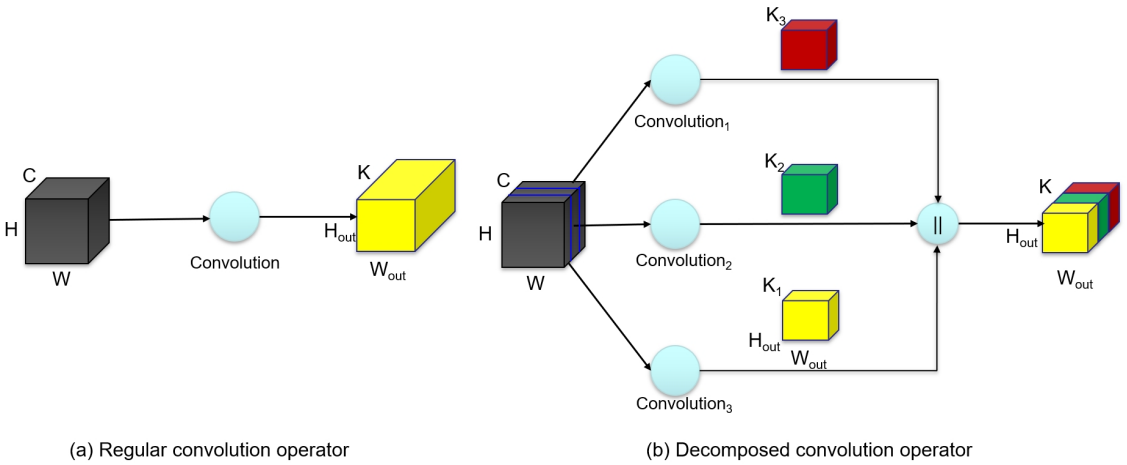

Fig. 2. Illustration of a decomposed convolutional layer.

of dataflow, a signal and information processing application is modeled as a directed graph $\Gamma = (B, E)$, where the set of vertices B represents the set of functional modules (actors) in the application, and the edges in E represent first-in, first-out (FIFO) communication channels between actors.

Ports in dataflow graphs refer to interfaces between actors and their incident edges. Ports in VR-PRUNE can be categorized into three different types: each port is either a (input or output) *control port*, *static regular port* (*SRP*) or *dynamic regular port* (*DRP*). An SRP ρ_s has a fixed token consumption/production rate $atr(\rho_s)$, which is set at design time. The token rate associated with a dataflow port ρ and a given execution f of the enclosing actor α is : (1) the number of data values consumed by α through ρ during f if ρ is an input port or (2) the number of data values produced by α through ρ during f if ρ is an output port.

A DRP ρ_d in VR-PRUNE has a variable token rate. As a design rule of VR-PRUNE, this token rate must satisfy the following constraint

$$lrl(\rho_d) \leq atr(\rho_d) \leq url(\rho_d), \tag{4}$$

where the terms lrl, atr, and url stand for lower rate limit, active token rate, and upper rate limit, respectively; these three quantities are always non-negative-integer valued. The values lrl and url are values that are fixed at design time, whereas atr may vary at run-time within the constraint imposed by Equation 4.

A control port ρ_c in VR-PRUNE must have a fixed token rate of 1.

VR-PRUNE actors can be classified into three types: static processing actors (SPAs), dynamic actors (DAs), and dynamic processing actors (DPAs). The ports of an SPA can only be of the type SRP, and therefore an SPA operates as a synchronous dataflow (SDF) [27] actor.

The dynamic attributes of VR-PRUNE manifest in the DAs and DPAs of a VR-PRUNE graph. Unlike SPAs, DAs and DPAs have DRPs. In the dataflow-based DNN design approach developed in this paper, DAs and DPAs are employed

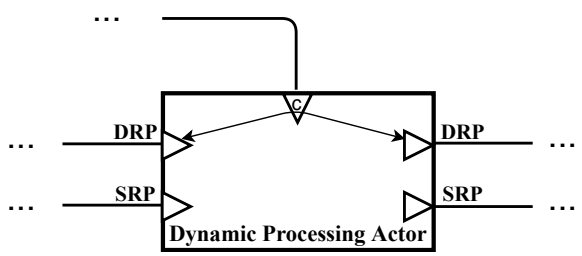

Fig. 3. An illustrative example of a dynamic processing actor.

to provide dynamic enabling and disabling of sublayers based on dynamically-computed attention scores.

Fig. 3 provides an illustration of a DPA. The control input port sets the token rates of a pair of DRPs (one on the input side, and the other on the output side). When the actor executes, the value of a single token coming from the control port c sets the token rates of both DRPs. In our decomposed-DNN modeling approach, we apply DPAs to decompose layers that may be dynamically reconfigured (by changing the set of sublayers that is active), and we use SPAs to describe the other functional modules in the decomposed system. The set of dynamically-enabled sublayers that is active at any given time is determined by configuration actors, which can receive input from dynamically-determined attention scores.

To enable the importing of DNN models that have been written in TensorFlow [28] or PyTorch [29] to VR-PRUNE, we have extended the VR-PRUNE runtime framework (VPRF) with ONNX [6] support. In particular, our new version of the VPRF framework includes a VR-PRUNE graph generator built on the `libonnx` project [30]. The VR-PRUNE graph generator parses an ONNX model specification and automatically generates a VR-PRUNE graph from the DNN model derived from the parsing results. The VPRF application designer can specify — as input to the VR-PRUNE graph generator — which sections of the graph are to be dynamically reconfigurable. As part of its synthesis process, the VR-PRUNE graph generator inserts dynamic actors and control actors to achieve the specified functionality for dynamic reconfiguration.

Fig. 4 illustrates the interplay between a decomposed layer and the configuration actor that is responsible for enabling and disabling the associated sublayers. This example highlights the role of configuration actors in determining the token rates for DRPs, thereby controlling the dynamic enabling/disabling of selected sublayers, which is central to the adaptability of our transformed network architecture.

IV. Experiments

Our experiments are conducted using a VR-PRUNE-based prototype implementation of the ResNet50 + CBAM model [9] with decomposition into sublayers and attention-guided sublayer disabling as developed in Section III. We use $lcount = 4$ for the decomposition process. We conduct experiments on the ImageNet-1K dataset [31]. Processing includes a resize of 256×256 followed by single-crop evaluation with a size of 224×224 at validation/test time. We use the original ResNet50 + CBAM model as a baseline model in our experiments, and evaluate the performance of decomposed and partially-disabled models compared to this baseline. The processing platforms employed in our experiments are summarized in Table I.

As additional baseline configurations, we consider the disabling of entire blocks (among a total of 4) of the ResNet-50 backbone model. We consider instances of this baseline disabling method involving 1, 2 and 3 disabled blocks, respectively. Before evaluating each of these baseline-disabling instances, we fine-tune the modified model on the training data for 50 additional epochs. This baseline approach to disabling and network reconfiguration is selected as a straightforward alternative for manipulating network structure to trade off the accuracy and computational requirements of inference.

Table II shows experimental results involving the baseline in which entire blocks in ResNet-50 are disabled. The abbreviations `@1` and `@5` stand for rank-1 and rank-5 accuracy, respectively. Table III shows the performance of the proposed decomposition and selective-disabling approach with different numbers of disabled sublayers per decomposed layer L_i. In both tables, the original ResNet-50 + CBAM model is represented in the first row of data. From the data in the two tables, we see that disabling sublayers provides significant improvements in execution time, but not as much improvement as disabling entire blocks. However, the loss in accuracy for disabling sublayers is significantly less than the accuracy that is given up when disabling blocks. Thus, we can see that the proposed decomposition method leads to novel operational trade-offs compared to the block-based disabling baseline.

Limitations. Although we have shown the potential of VR-PRUNE to dynamically configure machine learning models and to save on computations, the implementation of fine-grained channel pruning requires further investigation. The main reasons for this are: (1) The limitation of the backbone model itself; for ResNet50 + CBAM, there are altogether 117 convolution operators in the graph, and the cumulative execution time of these is 34% of the total. Thus, the overall computational efficiency provided by disabling sublayers is limited to impact at most one-third of the overall computation. (2) The decomposition process generates extra actors for a dataflow model. For example, in our prototype implementation, it generates 96 additional Constant actors, 48 Convolution actors, and 4 additional Concat nodes, where Concat actors are implemented in ONNX [6] as inefficient memory-to-memory-copies.

V. Conclusion

This paper has developed new methods for pruning deep neural networks (DNNs) to trade-off some amount of accuracy loss for reduced computational requirements, and also to enable dynamic control of accuracy/computational-requirements trade-offs in response to time-varying oper-

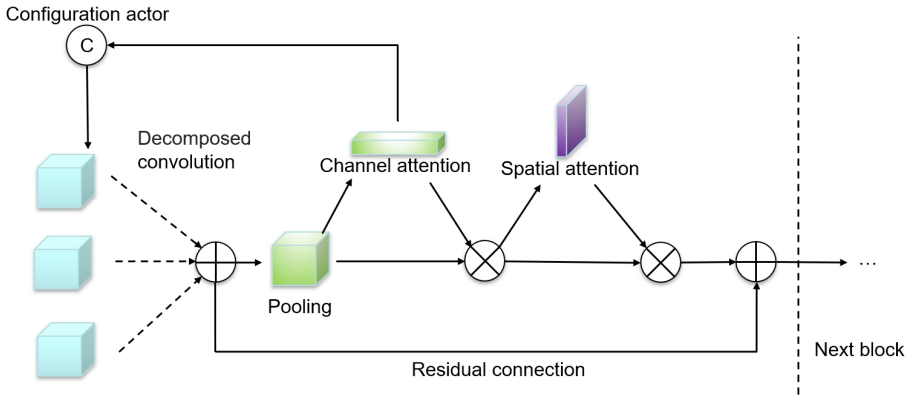

Fig. 4. An illustration of how the dynamic control of sublayers is implemented using VR-PRUNE.

TABLE I

PLATFORMS USED FOR EXPERIMENTS.

Tag	CPU	GPU	Operating System
Plat1	Intel Core i7-12700H 14-Core Processor: 4.70 GHz	NVIDIA RTX A2000	Ubuntu Linux 22.04
Plat2	AMD Ryzen 5 3600 6-Core Processor: 3.60 GHz	NVIDIA RTX 2060 Super	Ubuntu Linux 18.04
RB5	Qualcomm Kryo 585	Qualcomm Adreno 650	Ubuntu Linux 18.04

TABLE II

PRUNING OF RESNET50 + CBAM AT THE BLOCK LEVEL.

	Plat1	Plat2	RB5	Accuracy
Res50 + CBAM	1.49s	1.72s	3.07s	@1 87.745 @5 97.672
Disable 1 block	1.32s	1.57s	2.68s	@1 69.476 @5 86.948
Disable 2 blocks	1.01s	1.25s	2.16s	@1 51.698 @5 70.716
Disable 3 blocks	0.70s	0.98s	1.73s	@1 12.246 @5 21.972

TABLE III

CHANNEL PRUNING.

	Plat1	Plat2	RB5	Accuracy
Res50 + CBAM	1.42s	1.72s	3.07s	@1 87.745 @5 97.672
Disable 1 sublayer	1.31s	1.67s	2.95s	@1 72.080 @5 91.907
Disable 2 sublayers	1.23s	1.58s	2.73s	@1 55.185 @5 72.627
Disable 3 sublayers	1.14s	1.52s	2.55s	@1 13.684 @5 23.284

ational constraints and conditions. The proposed methods are based on decomposing selected DNN layers into collections of lower-complexity layers (sublayers), and applying attention techniques to select subsets of sublayers to disable at design time or runtime. The VR-PRUNE framework for dataflow modeling is applied and extended to prototype the proposed methods and provide structured methods to achieve runtime sublayer-disabling functionality efficiently and reliably. The paper has described several interesting directions for future work in terms of generalizing the proposed methods. We envision that these generalizations have the potential to significantly enrich the design space of model decomposition, and enhance the novel trade-offs that are provided by decomposed implementations.

ACKNOWLEDGEMENTS

This research was supported in part by U.S. Army Cooperative Agreement W911NF2120076, and by the Academy of Finland projects 327912 REPEAT and 345683 SPHERE-DNA, as well as by the Business Finland Veturi project DAZE.

REFERENCES

[1] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[2] L. Alzubaidi, J. Zhang, A. J. Humaidi, A. Al-Dujaili, Y. Duan, O. Al-Shamma, J. Santamaría, M. A. Fadhel, M. Al-Amidie, and L. Farhan, "Review of deep learning: Concepts, cnn architectures, challenges, applications, future directions," *Journal of big Data*, vol. 8, pp. 1–74, 2021.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[4] R. Xie, H. Huttunen, S. Lin, S. S. Bhattacharyya, and J. Takala, "Resource-constrained implementation and optimization of a deep neural network for vehicle classification," in *Proceedings of the European Signal Processing Conference*, Budapest, Hungary, August 2016, pp. 1862–1866.

[5] Y.-H. Chen, J. Emer, and V. Sze, "Using dataflow to optimize energy efficiency of deep neural network accelerators," *IEEE Micro*, vol. 37, no. 3, pp. 12–21, 2017.

[6] ONNX developers, "Open neural network exchange," https://github.com/onnx/onnx, 2023, version: 1.15.0.

[7] Y. Zhang, K. Li, K. Li, L. Wang, B. Zhong, and Y. Fu, "Image super-resolution using very deep residual channel attention networks," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 286–301.

[8] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7132–7141.

[9] S. Woo, J. Park, J.-Y. Lee, and I. S. Kweon, "Cbam: Convolutional block attention module," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 3–19.

[10] Q. Wang, B. Wu, P. Zhu, P. Li, W. Zuo, and Q. Hu, "Eca-net: Efficient channel attention for deep convolutional neural networks," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 11 534–11 542.

[11] D. Misra, T. Nalamada, A. U. Arasanipalai, and Q. Hou, "Rotate to attend: Convolutional triplet attention module," in *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, 2021, pp. 3139–3148.
[12] Z. Qin, P. Zhang, F. Wu, and X. Li, "Fcanet: Frequency channel attention networks," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 783–792.
[13] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2736–2744.
[14] M. Liu, W. Fang, X. Ma, W. Xu, N. Xiong, and Y. Ding, "Channel pruning guided by spatial and channel attention for dnns in intelligent edge computing," *Applied Soft Computing*, vol. 110, p. 107636, 2021.
[15] Y. Zhang, M. Lin, C.-W. Lin, J. Chen, Y. Wu, Y. Tian, and R. Ji, "Carrying out cnn channel pruning in a white box," *IEEE Transactions on Neural Networks and Learning Systems*, 2022.
[16] Z. Chen, T.-B. Xu, C. Du, C.-L. Liu, and H. He, "Dynamical channel pruning by conditional accuracy change for deep neural networks," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 2, pp. 799–813, 2020.
[17] E. A. Lee and T. M. Parks, "Dataflow process networks," *Proceedings of the IEEE*, vol. 83, no. 5, pp. 773–799, 1995.
[18] S. S. Bhattacharyya, E. Deprettere, R. Leupers, and J. Takala, Eds., *Handbook of Signal Processing Systems*, 3rd ed. Springer, 2019.
[19] A. Parashar, P. Raina, Y. S. Shao, Y.-H. Chen, V. A. Ying, A. Mukkara, R. Venkatesan, B. Khailany, S. W. Keckler, and J. Emer, "Timeloop: A systematic approach to dnn accelerator evaluation," in *2019 IEEE international symposium on performance analysis of systems and software (ISPASS)*. IEEE, 2019, pp. 304–315.
[20] J. Boutellier, J. Wu, H. Huttunen, and S. S. Bhattacharyya, "Prune: Dynamic and decidable dataflow for signal processing on heterogeneous platforms," *IEEE Transactions on Signal Processing*, vol. 66, no. 3, pp. 654–665, 2017.
[21] T. Narshana, C. Murti, and C. Bhattacharyya, "Dfpc: Data flow driven pruning of coupled channels without data." in *The Eleventh International Conference on Learning Representations*, 2022.
[22] J. H. Heo, A. Fayyazi, A. Esmaili, and M. Pedram, "Sparse periodic systolic dataflow for lowering latency and power dissipation of convolutional neural network accelerators," in *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, 2022, pp. 1–6.
[23] H. Kwon, A. Samajdar, and T. Krishna, "Maeri: Enabling flexible dataflow mapping over dnn accelerators via reconfigurable interconnects," *ACM SIGPLAN Notices*, vol. 53, no. 2, pp. 461–475, 2018.
[24] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: Efficient inference engine on compressed deep neural network," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 243–254, 2016.
[25] M. H. Wiggers, M. J. Bekooij, and G. J. Smit, "Buffer capacity computation for throughput constrained streaming applications with data-dependent inter-task communication," in *2008 IEEE Real-Time and Embedded Technology and Applications Symposium*. IEEE, 2008, pp. 183–194.
[26] J. Boutellier, Y. Ma, J. Wu, M. Khan, and S. S. Bhattacharyya, "Vr-prune: Decidable variable-rate dataflow for signal processing systems," *IEEE Transactions on Signal Processing*, vol. 70, pp. 1819–1833, 2022.
[27] E. A. Lee and D. G. Messerschmitt, "Static scheduling of synchronous data flow programs for digital signal processing," *IEEE Transactions on Computers*, vol. 100, no. 1, pp. 24–35, 1987.
[28] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous systems," https://www.tensorflow.org/, 2022, version 2.9.0.
[29] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch," https://pytorch.org/, 2022, version 1.11.0.
[30] Libonnx developers, "Libonnx: a lightweight, portable pure c99 onnx inference engine," https://github.com/xboot/libonnx, 2022.
[31] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. IEEE, 2009, pp. 248–255.

Algorithm 1 Proposed network decomposition algorithm.

Input:

G: dataflow graph model of the original DNN.
$\Omega = \{L_1, L_2, \ldots, L_p\}$, where $p = |\Omega|$.
$lcount$: The number of sublayers in each decomposed layer.
D_v: The validation dataset.
D_t: The training dataset.
τ: the number of channels to be disabled.

Output:

The decomposed graph with selected groups disabled.

Initialization: Convert $L_1, L_2, \ldots, L_p$ in G to decomposed layers to form $T(G)$.
$U \leftarrow T(G)$
$i = 1$
Exercise $T(G)$ on D_v and record the resulting attention scores $\{w(c)\}$ for all channels c in all sublayers within $T(G)$.
while $i \leq p$ **do**
    $j = 1$
    while $j \leq lcount$ **do**
        Set the sublayer attention score $W_{i,j}$ to be the average of $w(c)$ over all output channels c within sublayer $\delta(L_i, j)$.
        $j \leftarrow j + 1$
    end while
    $i \leftarrow i + 1$
end while
Rank the values $\{W_{i,j}\}$ into a sorted list V.
Set ω to be the τth smallest value in V.
$m = 1$
while $m \leq p$ **do**
    $n = 1$
    while $n \leq lcount$ **do**
        if $W_{m,n} \leq \omega$ **then**
            Disable the sublayer $\delta(L_m, n)$ in U and modify the connections of operators in U accordingly.
        end if
        $n \leftarrow n + 1$
    end while
    $m \leftarrow m + 1$
end while
Fine tune U using D_t.
Return U as the decomposed, partially-disabled graph.