AINNS: All-Inclusive Neural Network Scheduling via Accelerator Formalization

Jiaqi Zhang, *Student Member, IEEE*, Xiangru Chen, *Student Member, IEEE*, and Sandip Ray, *Senior Member, IEEE*

Abstract— Driven by the rapid development of accelerators and diverse efficiency requirements of the naturally heterogeneous neural network computation, recent years have seen increased heterogeneity in neural network accelerator systems in terms of network structures, accelerator dataflows and implementations. However, existing research fails to schedule and map the heterogeneous neural networks on heterogeneous accelerators efficiently. They rely on clumpy exhaustive search or complicated ad hoc mapping approaches due to the semantic gap between the networks and accelerators. This paper proposes a systematic method to transform various accelerators into standard parameterized containers of the neural network loops, which builds a direct connection between the computation and the underlying hardware resources. This enables us to match the neural networks with accelerators based on their essential characteristics (e.g., reuse opportunities and bandwidth requirements) without diving into the detailed architectures. To this end, we propose AINNS, an all-inclusive neural network scheduler, that automatically schedules and maps the NN computation on heterogeneous accelerators with just one universal algorithm. Our experimental results show the proposed AINNS not only performs well in the traditional neural network acceleration but also improves the system throughput and energy efficiency by 1.8x and 1.7x respectively in the most challenging heterogeneous acceleration system.

Index Terms— Neural network, accelerator formalization, heterogeneity, scheduling.

1 Introduction

The revival of neural networks (NN) has led to an explosion of deep NN models, which are trained and deployed for diverse tasks. Along with large-scale NNs, the past few years also witnessed a burst of efforts in the acceleration of these both computation- and memory-intensive workloads. The massive and ever-emerging NN models and accelerators aim to realize a myriad of intelligent agents in our daily life. Recently, we have equipped IoT devices with artificial intelligence and entered the era of Artificial Intelligence of Things (AIoT) [1].

In a world with everything intelligent and connected, it is ideal if we could uniformly and efficiently manage them. One salient challenge to this is the heterogeneity in both NN workloads and accelerators. As shown on the left side of Fig. 1, traditional frameworks view NNs as graphs of different kinds of layers and the underlying hardware accelerators also vary in dataflow and implementation. There are existing works [2][3] that efficiently address the challenges in heterogeneity in the neural networks by introducing certain intermediate representations. Several recent works [4][5] are also proposed to take the difference in the performance of the accelerators into consideration. However, there still lacks a systematic solution to the problem.

The key obstacle is the semantic gap among the workloads and accelerators, which is twofold. Since there is not a standard to regularize the accelerators, the scheduler does not understand the difference between different accelerators. Neither does it understand the correspondence between the NN computation and accelerators. This impedes an effortless universal algorithm to schedule and map the tasks among the heterogeneous accelerator cluster. Ad hoc dataflow customization in each accelerator is required and scheduling needs to rely on tremendous design space search [6]. Unfortunately, as more AI tasks are executed ubiquitously, there will be more dynamic and complicated acceleration requests. The static optimizations can no longer meet our demands. In this work, we address the need for an all-inclusive systematic approach to accelerating heterogeneous NNs on a cluster of heterogeneous accelerators, which works in real time.

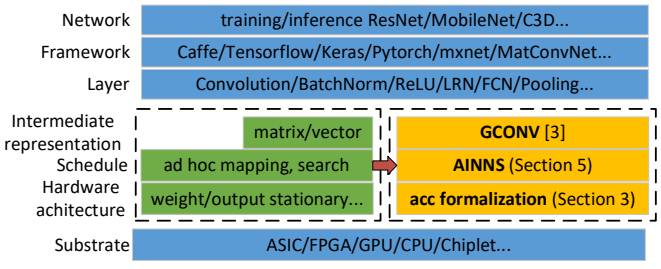

Fig. 1. Neural network acceleration stack before (left) and after (right) our work.

Inspired by previous work [3] that generalized NN computation into standard operations, we propose to further formalize the NN accelerators as shown in Fig. 2. Compared with other models [7][8] that manually specify the dataflows, our formalization model focuses on the intrinsic data reuse functions of the accelerators. It eliminates not only the semantic gap between different accelerators but

• *Jiaqi Zhang, Xiangru Chen, and Sandip Ray are with the Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL 32611. E-mail: jiaqizhang, cxr1994816@ufl.edu, and sandip@ece.ufl.edu.*

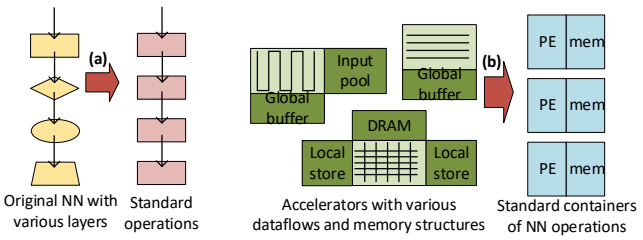

Fig. 2. **(a)** Neural network formalization (GCONV [3]). **(b)** Accelerator formalization.

also the one between NN workloads and the accelerators. The formalization works like a virtualization layer above the hardware layer as shown on the right side of Fig. 1. With the formalization of the NNs and the accelerators, the specific disparity of NN computation and accelerator implementations turns transparent to the scheduler, so it can rely on basic NN mapping and scheduling principles that are invariable despite of the heterogeneity. This allows the scheduler to focus more on the essential characteristics of the workloads and the hardware with less effort. Specifically, we can view various NN computation as standard nested loops of four parameters in several dimensions as in [3] and accelerators with diverse dataflows and implementations as parameterized containers of these NN loops.

Based on this, we implement an all-inclusive neural network scheduler (AINNS). The proposed AINNS is all-inclusive in two aspects: (1) It can cover workloads and accelerators with all levels of heterogeneity; (2) it integrates all the functions required in scheduling, including the local mapping and task dispatching that takes the computation partitioning and accelerator sharing into consideration.

In summary, this paper makes the following contributions:

(1) To motivate our work, we propose a taxology to define the level of heterogeneity of neural network workloads and accelerators.

(2) Our main contribution lies in the proposal of a systematic method to eliminate the semantic gap between the heterogeneous neural networks and accelerators by formalization.

(3) Our formalization method brings opportunities to schedule the neural network acceleration systems of any level of heterogeneity with one universal solution. To this end, we propose AINNS, an all-inclusive neural network scheduler.

(4) The experiments show that AINNS is both effective and efficient in scheduling benchmarks with all levels of heterogeneity. Notably in the case studies, AINNS reduces the energy consumption by 43% with the same usable user's satisfaction in the datacenter and improves the throughput by 1.8x while guaranteeing the real-time requirement in a smart plug-in hybrid electric vehicle.

The rest of the paper is organized as follows. Section 2 provides background on heterogeneity in NN acceleration and NN computation formalization. Section 3 introduces our proposed formalization model for the NN accelerators. Section 4 elaborates the basic principles for universal NN mapping and dispatching based on the formalization model. Section 5 proposes an AINNS architecture, which

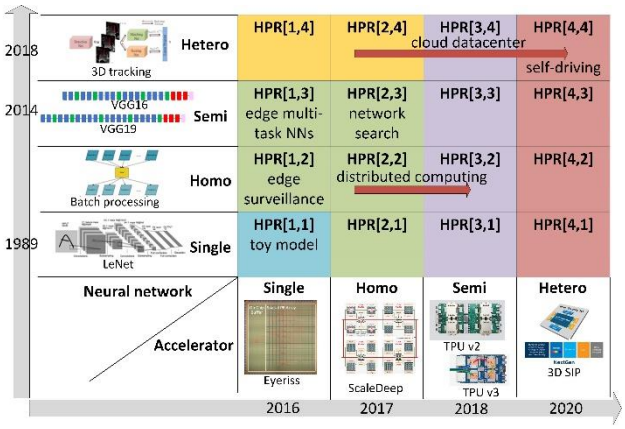

Fig. 3. Neural network-accelerator heterogeneity plane. Blue: early works; green: scaled-up systems; orange: compilers; purple: recent works; red: only AINNS.

is evaluated in Section 6. Sections 7 and 8 discusses the related works and concludes the paper respectively.

2 Background and Motivation

2.1 Heterogeneity in Both NNs and Accelerators

Based on real-world applications, the NN-accelerator plane can be divided into heterogeneity plane regions (HPR) [1,1] to [4,4], as shown in Fig. 3, where both NNs and accelerators manifest four levels of heterogeneity: (1) **Single:** only one accelerator/NN is deployed; (2) **homogeneous:** an accelerator or NN is duplicated and the system has homogeneous accelerator array or workloads; (3) **semi-heterogeneous:** the accelerators or the NNs adopt similar structure but different configurations, e.g., TPU v2 and TPU v3 or ResNe34 and ResNet50; (4) **heterogeneous:** all the accelerators or NNs in the system can be different in their structures, dataflows or configurations. In AIoT, we expect the following trends.

First, the NN workloads are increasingly heterogeneous. Obviously, the power of a single fixed neural network (HPR[x,1] and [x,2]) is limited. Recent works [9] proposed multi-task neural networks where different numbers of layers are involved to perform different tasks. Here, the NNs share similar functional layers but have different network architectures (HPR[x,3]). The trendy automated machine learning that searches for the optimal NN based on permutation of a group of basic blocks [10] also falls into this region. Furthermore, it is common that an intelligent system relies on cooperation of several heterogeneous NN models (HPR[x,4]). For example, the robots leverage several CNNs and MLPs to recognize the shape, color and texture of the object and another CNN to predict the grasp gesture [11]. In addition, large-scale data centers are flooded by AI workloads including various NN tasks, e.g., computer vision, recommendation systems, and neural machine translation, etc. [12].

Second, the NN accelerators are also increasingly heterogeneous. Given the outstanding efficiency of customized accelerators, they are a competent solution in both mobile ends [13] and the data centers of IT giants [14]. In this big-data era, except for certain static low-power edge nodes,

```
for g.i in range(N[dim][g]):
  for op.i in range(N[dim][op]):
    for opc.i in range(N[dim][opc]):
      for ks.i in range(N[dim][ks]):
        O[g.i][op.i][opc.i] += I[g.i][opc.i+ks.i] × K[g.i][op.i][ks.i]
```

Fig. 4. 1-D GCONV model which can be scaled up to multiple dimensions. N[dim][param]: the value of the parameter in the dimension. O: output; I: input, K: kernel parameter.

the throughput of a single standalone NN chip (HPR[1,x]) is far below demand. Therefore, accelerators are usually deployed in arrays. The real-world accelerator arrays are rarely homogeneous (HPR[2,x]). For example, the rapid upgrading has left considerable out-of-date yet still functioning devices. This leads to accelerators with similar architecture but different configurations (HPR[3,x]) as in Google's cloud [15] where both TPU v2 and v3 are available. Besides, to cater to NN workloads with different characteristics, accelerators with a wide range of architectures and dataflows are deployed [6]. In traditional intelligent applications, these accelerators are responsible for independent pre-defined tasks. However, in AIoT, it is unavoidable that they need to work as a heterogeneous cluster (HPR[4,x]) to provide service to dynamic tasks [16].

Another trend in AIoT is that all levels of heterogeneity can exist in the same system. First, a hierarchical management is expected in an AIoT system. More heterogeneity can be observed by the scheduler in a higher level. For example, a simple gesture recognition camera is less heterogeneous than a smart classroom system in terms of both the NN tasks and accelerator architectures [1]. Second, considering there are tasks arriving on the fly and users with high mobility [17], the workloads in the queue and available accelerators in the array are dynamic, featuring changing heterogeneity over time and location. The connected self-driving car is one of the most typical examples [18].

This increasing heterogeneity in both NN workloads and accelerators poses great challenges to the schedulers. Fig. 3 marks the HPRs covered by previous works (the details will be discussed in Section 7). Unfortunately, none of them can efficiently perform scheduling in all the regions, especially the most challenging all-heterogeneous case. Therefore, an approach to designing effective all-inclusive neural network schedulers is in demand.

2.2 Formalized NN Computation Model

As shown in Fig. 1, AINNS relies on a model that standardizes the NN computation. Among the previous works, we choose the GCONV model [3], which generalizes various NN computation into general convolution operations, for several reasons: (1) GCONV can model all kinds of NN layers, including convolution and non-convolution in both inference and training; (2) GCONV operation preserves the convolution computation pattern and reuse opportunities, which can be exploited by the accelerators for improved efficiency; (3) GCONV model is scalable and semantically symmetric in all the dimensions, so that we can easily build a dimension-independent connection between the computation and our proposed accelerator model, as will be elaborated in Section 4.

TABLE 1
EXAMPLES OF FOUR LOOPS OF GCONV MODEL

Example	Dim	Param
kernel size in convolution/pooling layer	*H*/*W*	*ks*
output size in convolution/pooling layer	*H*/*W*	*opc*
output feature map in convolution layer	*C*	*op*
input feature map in convolution layer	*C*	*ks*
batch size in convolution layer	*B*	*opc*
batch size in batch normalization layer	*B*	*ks*
channel range in local response normalization layer	*C*	*ks*
output size in normalization/dropout layer	*H*/*W*	*g*

W: width, H: height, C: channel, B: batch. This table only lists the most representative examples to help understand the loops. GCONV can model a wide range of NN computation as in [3].

The key idea of GCONV is to view the operation of a certain layer on the input data in each dimension as general convolution. A 1-D GCONV operation is described as a nest of four loops as listed in Fig. 4, i.e., *g* for groups of inputs, *op* for groups of kernels, *ks* for kernel size and *opc* for output size. This 1-D GCONV can be scaled up to multiple dimensions to represent various computation in NNs, even for traditionally non-convolution layers. Several representative examples are given in Table 1.

There are parallel and convolution reuses in each dimension of GCONV. Specifically, the inputs and kernel parameters are reused by computation in loops *op* and *opc* respectively and the outputs are reused through reduction in *ks* (**parallel reuse**). In addition, the computation of neighboring outputs share the inputs overlapped (when $ks > s$ and $opc > 1$) in the convolution windows (**convolution reuse**). Most efficient NN accelerators are designed to maximize these reuses.

3 FORMALIZATION OF NN ACCELERATORS

As discussed in Section 1, AINNS can adopt a uniform scheduling algorithm only when the gap between the NNs and accelerators is eliminated. Therefore, formalization of the accelerators is also required besides the NN formalization introduced in Section 2.2. An accelerator is composed of the computation (i.e., PE array) and memory resources.

3.1 PE Array Model

The main computation component in the NN accelerator is a group of processing elements (PEs) performing the basic element-wise arithmetic (e.g., MAC) on the inputs in parallel by unrolling the NN nested loop. To reduce data movement, PEs are arranged into multiple dimensions and exquisite interconnections among them are introduced to exploit the data reuse opportunities. With little to no difference in the implementation of individual PEs, the interconnections distinguish different accelerators and play an important role in determining the performance and energy efficiency of the accelerators on a certain network. Therefore, to model the PE array is indeed to abstract the interconnections for data reuse.

Challenge: Traditionally, the accelerators are *dataflow-defined*, which means the interconnections between PEs

TABLE 2
DATA REUSE FUNCTIONS

Reuse Type	Reused Data	Unrolled Loop	Unroll Dimension	Function	Examples
parallel	input	*op*	spatial	broadcast	almost all
	kernel	*opc*	temporal	stationary	all
	output	*ks*	spatial	*reduction*	[21][22]
			temporal	stationary	all
convo-lution	input	*ks opc*	spatial	*diagonal*	[19][23]*
			temporal	stationary	all
			spat+temp	*shift*	[21][24]

The functions in italic are explicitly modeled in our formalization. *[23] *diagonally collects outputs.*

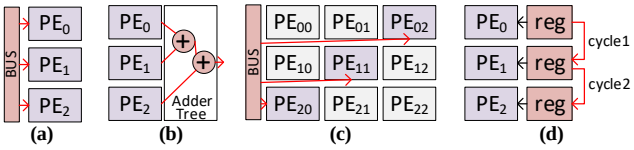

Fig. 5. Spatial reuse functions: **(a)** broadcast; **(b)** *reduction*; **(c)** *diagonal*; **(d)** *shift*.

only serve certain dataflows. They can be recognized as no-local-reuse, input-, weight-, output-stationary, a group-wise variation [19] or a combination of them [20] in classic taxonomy. We will see that this classification is not precise in the discussion of Table 2. Recent works [7][8] interpret the accelerator dataflows as a constraint on the loop blocking, i.e., how the loops in Fig. 4 are ordered and unrolled in different spatial (PE) and temporal (memory) dimensions of an accelerator. For example, the row-stationary dataflow of Eyeriss [19] is represented as S_C | Q_K | R_C_P ($ks_H\_ks_C$ | $opc_H\_opc_C$ | $ks_W\_ks_C\_opc_W$ in the GCONV model) in row (spatial) | column (spatial) | scratchpad (temporal) in [8]. This representation is precise but still not efficient in real-time all-inclusive scheduling. It requires manual specification of the loop order, which is burdensome and inflexible, especially when there are various NN computation suitable for different dataflows. Moreover, since this representation reveals little about the objective structure of the accelerators, it is hard for the scheduler to align heterogeneous accelerators or to implement a universal algorithm to schedule and map the NN tasks. The scheduler can only exhaustively search the design space pruned by these constraints.

Proposed model: Our proposed model describes the accelerators in a *function-defined* way. Instead of binding the accelerator with a fixed dataflow, we focus on the intrinsic functions of the interconnections. We enumerate the possible primitives for spatial and temporal data reuses discussed in Section 2.2 in a systematic manner. Table 2 lists the data reuse functions and corresponding accelerators that adopt each function.

First, the temporal reuse of all the data (both parallel and convolution reuse) can be intrinsically realized by keeping the data **stationary** in memory without any special hardware function. For output parallel reuse that requires reduction, it can also be performed in a stationary manner, i.e., read-reduce-write using the reduction function within the PE. Therefore, the temporal data reuse functions are

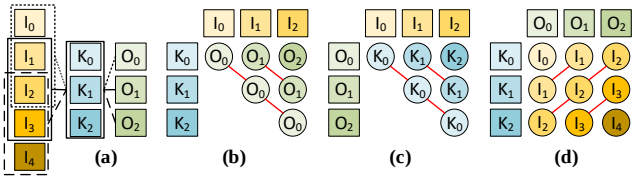

Fig. 6. Convolution reuse patterns for each type of data. **(a)** is an example of convolution. Each box on the inputs is the convolution window of each output. **(b)** to **(d)** indicate the convolution reuse (red line) of each type of data (round) when unrolling the other two (square).

not explicitly modeled. Note that the xx-stationary dataflow defination does not hold because it only defines the innermost temporal unrolling.

Another function not explicitly modeled is the spatial parallel reuse of inputs and kernel parameters. It is realized by **broadcast** (Fig. 5(a)) through the data bus, which is commonly used in modern accelerators. However, a ***reduction*** function (e.g., the adder tree in Fig. 5(b)) over a certain dimension of PEs needs to be clearly defined to dictate if the output parallel reuse can be exploited.

The convolution reuse is an interplay between the input, output, and kernel data. As shown in Fig. 6, the overlapping of convolutional windows provides reuse opportunities for the inputs, kernels, or outputs when the other two types of data are unrolled. Note that since not all the inputs are utilized for the computation of all the outputs (e.g., I_0, I_1, I_3, I_4), the unrolling of inputs (Fig. 6(b) and (c)) always results in ineffectual computation, which is extremely severe in the weight update phase during training [21]. Therefore, we focus on the unrolling of kernels and outputs with convolution reuse of inputs (Fig. 6(d)). To exploit the convolution reuse, both the kernels and outputs can be either spatially or temporally unrolled. When they are unrolled in different spatial dimensions, the inputs can be reused by ***diagonal*** broadcasting (Fig. 5(c)). And when one of them are unrolled temporally, the reuse of inputs should be implemented by ***shift*** function (Fig. 5(d)), where the inputs are shared by different PEs in different cycles via forwarding. Note that the systolic array [25] also adopts data forwarding interconnections between the PEs. However, they in essence exploit the parallel reuses (i.e., broadcast or *reduction*) with lower bandwidth demands.

With data reuse functions defined, the PE array is described by a vector

[PE size, reduction, diagonal, shift]

in each spatial dimension in our model, which clarifies PE array size besides the three reuse functions. In NN accelerators, the interconnections can be fixed or reconfigurable [6]. For example, although [21] provides an adder tree that connects all the PEs, *reduction* can be optionally performed, as long as the number of outputs does not exceed the bandwidth. Therefore, we explicitly indicate if the functions are not-available (N), allowed (A) or mandatory (M) in each PE dimension. Table 3 lists the PE array models of four representative accelerators.

3.2 Memory Model

The storage resources of the accelerators appear in various forms including the operand registers for each PE, the

TABLE 3
PE ARRAY MODEL EXAMPLES

Accelerator	Dimension	PE Array Model
Eyeriss [19]	1 (row)	[12, A, A, N]
	2 (column)	[14, N, A, N]
NLR [22]	1 (T_n)	[7, M, N, N]
	2 (T_m)	[64, N, N, N]
Eager Pruning [21]	1 (PE array)	[512, A, N, A]
	2 (subsystem)	[4, A, N, N]
TPU [25]	1 (row)	[256, N, N, N]
	2 (column)	[256, M, N, N]

Each dimension is [PE size, reduction, diagonal, shift].

TABLE 4
MEMORY MODEL EXAMPLES

Accelerator	Data	Local Storage	Global Buffer
Eyeriss	K	[224, 1, F, F, 10]	[4096, 4, T, T, 130]
	I	[12, 1, F, F, 6]	[51200, 1, T, T, 30]
	O	[24, 1, F, F, 6]	[-2, 4, T, T, 90]
NLR	K	[1, 1, F, F, 1]	[786432, 7, T, T, 290]
	I	[1, 1, F, T, 1]	[-3, -3, T, T, 290]
	O	[1, 1, T, F, 1]	[393216, 64, T, T, 90]
Eager Pruning	K	[1, 1, F, F, 1]	[786432, 32, T, F, 130]
	I	[64, 512, T, F, 6]	[786432, 32, T, F, 130]
	O	[32, 32, T, F, 1]	[786432, 32, T, F, 190]
TPU	K	[1, 1, F, F, 1]	[2097152, 45, T, T, 90]
	I	[1, 1, T, F, 1]	[12582912, 256, T, T, 60]
	O	[1, 1, F, F, 1]	[-2, 256, T, T, 90]

Each level of memory has [capacity (B), bandwidth to higher-level memory (B/cycle), associativity in PE dim1, associativity in PE dim2, normalized communication cost to higher-level memory (/B)] for input, output or kernel. F: false, T: true.

register arrays shared by multiple PEs, the on-chip SRAMs, the off-chip DRAMs, etc. Regardless of the specific implementation, our formalization models a hierarchical and inclusive memory abstraction. There are three kinds of data, inputs, kernel parameters and outputs (partial results) in the memory system. Therefore, there are three sets of parameters for each level of memory. We model each level of memory for each kind of data as

[capacity, bandwidth, associativity bits, communication cost]

as shown in Table 4. For simplicity, we assume all the accelerators implement 8-bit data as in [25].

For the capacity and bandwidth, minus values represent sharing between different kinds of data. For instance, -2 in the global buffer of TPU means the outputs share the memory capacity with the inputs. The associativity bits indicate if all the PEs in a certain dimension share the memory capacity and bandwidth. Usually, the local storage is exclusive and global buffers are shared by all. One exception is Eager Pruning which divides the PEs into subsystems and the input pool and global buffers are only shared within each subsystem (PE dimension 1). The association bit can also be utilized to model the mandatory broadcasting by forcing the PEs to share only one data. For example, the input is broadcast to T_m PEs (PE dimension 2) in NLR. Note that the capacity and bandwidth pertain to each associated memory.

The last parameter is the communication energy cost between the memory and its higher level or the PEs for the local operand registers. In this work, we generate the energy consumption using the CACTI [26] simulation for RAMs and Synopsys Design Compiler for registers based on the capacity and bandwidth assuming the accelerators are implemented in the same process technology. The costs are normalized to that of a single register shifting. In general practice, the costs can be evaluated or estimated using off-the-shelf models according to the specific implementation and technology.

This model flexibly applies to different memory hierarchies. For instance, some accelerators feed the PEs directly from the global buffers [25] and others may adopt more complicated hierarchical on-chip memory system [21], needless to say the various off-chip storage. Although Table 4 only lists the formalized model for local storage and global buffer, we model the entire memory hierarchy of each accelerator in the experiments.

3.3 Inter-Accelerator Connection

In accelerator arrays, it is common to distribute the computation among accelerators to improve performance. Therefore, when scaling into multi-accelerator systems, we model the connection between the accelerators. In spite of the various connection technologies, e.g., buses, networks, or interposers, we focus on the communication costs and bandwidths between each pair of accelerators, which are modeled in a connection matrix.

4 PRINCIPLES FOR AINNS

At the system level, the total throughput and energy efficiency are determined by not only how the NN tasks are distributed among the accelerators (dispatching) but also how the NN computation is executed on a certain accelerator (mapping). Our accelerator formalization model proposed in Section 3 provides an interface to construct principles for NN dispatching and mapping that are invariable regardless of the heterogeneity. On one hand, the formalization model eliminates the disparity between accelerators with various architectures and dataflows, so that they can be compared explicitly. On the other hand, the accelerator formalization model is also consistent with the GCONV model. This enables a concise and universal connection between NNs and the underlying accelerators. Intuitively, we view the NN computation as entities with parameterized shapes in multiple dimensions and the accelerators as multi-dimension containers with shapes defined under the same standard to accommodate them.

4.1 Mapping Principle

As mentioned in Section 3.1, mapping of an NN is described by loop blocking. Normally, the accelerator possesses two or more spatial dimensions as in Table 3. The memory manifests multiple levels but the temporal dimension is unified because it is hierarchical as shown in Fig. 7. When putting NNs into accelerators, each spatial dimension is filled independently.

In AINNS, neural network mapping shows great rotatability. Specifically, the GCONV model is semantically symmetric in all dimensions. The accelerator formalization model also defines each dimension individually. Therefore,

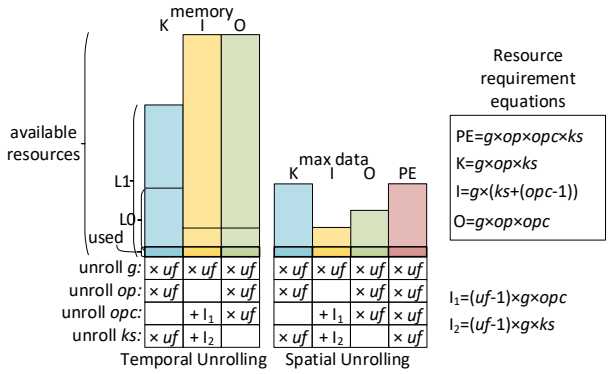

Fig. 7. Resource requirement in temporal and spatial unrolling in one dimension. *uf*: unrolling factor. L0/L1: high/low memory level.

there is no fixed matching between the dimensions of NN computation and the accelerator. The parameter of any dimension of NN can be unrolled in any dimension of the accelerator with no enforced order. Thanks to this feature, the following principles apply to universal NN acceleration.

To closely pack the NN computation into the accelerator and thus optimize the performance and energy efficiency, the scheduler should improve the utilization of both computation and memory resources. There are two major goals: (1) increase the computation performed in parallel by maximizing the loops unrolled spatially; (2) reduce accesses to the costly low-level memory by maximizing the data reuse and the loops unrolled temporally in the blocking of high-level memory.

Loop blocking is constrained by the accelerator hardware structure. The data reuse functions in the accelerator PE array model determines whether a parameter can be unrolled in a certain dimension of the accelerator according to Table 2. If the parameter is allowed or mandatory, the key limitation for the unrolling is the resource requirement, i.e., how many resources are occupied when a certain parameter is unrolled.

Here, we mainly consider the PE size and memory capacity. The temporal unrolling is constrained by the memory capacity and the spatial unrolling is constrained by both the PE size and the maximal data in that dimension, which is dictated by the available operator registers, i.e., the lowest-level memory. As shown in the "used" box in Fig. 7, when nothing is unrolled (unrolling factor is 1), only one PE and the storage for one data of each type are required. For each unrolling, the resource requirements expand with the unrolling factor differently based on the exact parameter. The occupation of PEs is simply the same as the unrolled loop points because each point corresponds to a MAC computation. For capacity requirement of the memory including the maximal data in PE array, each unrolling occupies the memory for all three kinds of data unless there is a data reuse opportunity. The blank cells in Fig. 7 indicate the parallel reuse and $+I_1/I_2$ entries indicate the convolution reuse which is smaller than $\times uf$ only when the unrolling factors of both *opc* and *ks* are above 1. Note that the resource requirments are multiplied on blockings of different parameters (e.g., *ks*, *op*) and DNN dimensions (e.g., *C*, *H*, *W*). The temporal unrolling may span multiple memory levels for different data types. For example, the unrolled *op* loop in Fig. 4 can keep the input stationary in local scratchpad but fectch the kernel from the global buffer.

Despite bandwidth's impact on the data loading time, it does not constrain the mapping of the network. Therefore, we only increase the data reuse in all the spatial dimensions to reduce the bandwidth requirement.

TABLE 5
DEFINITION OF SIZE AND SHAPE FEATURES

Type	Feature		Description
	Accelerator	Network	
size	$\underline{npe} = \prod_{ped}$ pe_size	$\underline{ncomp} = \prod_{opd}$ ($ks \times opc \times op \times g$)	total PEs and total computation
	$\underline{nmem}_{k/i/o} = \sum_{meml}$ (mem_size/cost)	$\underline{ndata}_{k/i/o} = \prod_{opd}$ required data	memory weighted by the costs and total data
shape	$\underline{pbr}_{k/i/o} = npe/\mathrm{bw}_{k/i/o}$	$\underline{cdr}_{k/i/o} = ncomp/ndata_{k/i/o}$	PE/bandwidth ratio and computation/data ratio
	$\underline{cpr} = (\prod_{conv\text{-}ped}$ pe_size$)/npe$	$\underline{ccr} = (\prod_{conv\text{-}opd}$ ($ks \times opc$))$/ncomp$	convolution (*diagonal*, *shift*) PE ratio and convolution computation
	$\underline{rpr} = (\prod_{red\text{-}ped}$ pe_size$)/npe$	$\underline{rcr} = (\prod_{opd} ks)/ncomp$	*reduction* functioned PE ratio and *reduction* computation ratio

ped/opd (conv/red- ped/opd): PE/NN dimension (with convolution/reduction); meml: memory level. The required data can be derived from Fig. 7.

4.2 Dispatching Principle

Though we can increase the utilization of an accelerator by optimizing the loop blocking of the neural network, the upper bound is limited. For example, as mentioned in Section 4.1, the hardware structure of accelerator determines which NN parameters can be unrolled to exploit certain data reuse opportunities. However, if the NN computation does not exhibit sufficient reuse opportunities, the utilization will be low. On the other hand, if the abundant reuse opportunities of an NN cannot be exploited by the accelerator, unnecessary data movement will still be involved. Therefore, high utilization of the entire system relies on smart matching of the NN tasks and accelerators.

In AINNS, since the NN computation and accelerator structures are modeled consistently, we can introduce comparable feature pairs, as listed in Table 5, to help the scheduler efficiently match them. The shape features indicate how the NN would fit into the accelerator. They can constrain the utilization of the accelerator resources. The size features measure the total required and available resources of an NN and an accelerator, which determine the performance when the shape features match. These features capture the intrinsic characteristics of any neural network and accelerator. They can be leveraged to perform dispatching based on various algorithms.

5 AN AINNS ARCHITECTURE

With the formalization model and scheduling principles proposed in Sections 3 and 4, we can design various

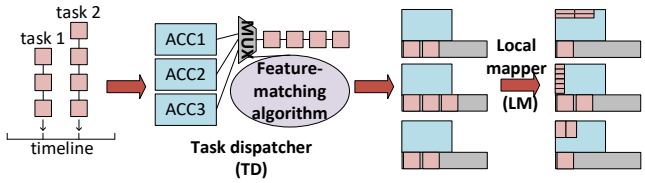

Fig. 8. AINNS system overview.

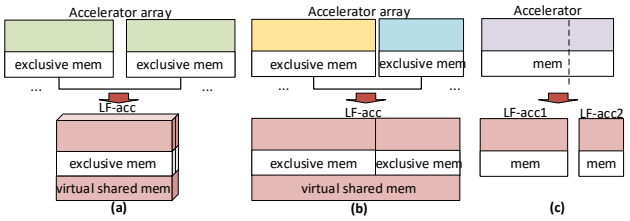

Fig. 9. Three types of logical formalized accelerators (LF-acc): **(a)** homogeneous accelerators; **(b)** heterogeneous accelerators; **(c)** accelerator partitioning.

efficient scheduling and optimization algorithms in the heterogeneous NN acceleration system. In this section, we present the AINNS that we implemented as an illustrative demonstration of the viability of the model.

Fig. 8 shows an overview of our AINNS system. AINNS is equipped with a task dispatcher and a local mapper. It takes a series of NNs (converted to GCONV Chains as in [3]) as input workloads. The task dispatcher (TD) schedules the NNs in the queue one by one by finding the most matched accelerator in terms of the required and available resources to guarantee high utilization. Then local mapper (LM) determines the exact mapping of each network on the corresponding accelerator, i.e., to find the spatial and temporal unrolling plans with the best performance and energy efficiency. Both TD and LM work in real time.

5.1 Logical Formalized Accelerator (LF-acc)

In a multi-accelerator-multi-NN system, an accelerator can be shared by more than one networks and a network can be distributed among accelerators [5][27]. In both cases, we focus on the logical accelerator that each NN runs on. This section introduces the logical formalized accelerators (LF-acc) so that the multi-tenant execution and network partitioning can be implicitly supported in AINNS.

Fig. 9 shows three types of LF-accs in AINNS. First, a set of homogeneous accelerators with uniform connections is viewed as an LF-acc with one more spatial dimension (Fig. 9(a)). All levels of memory in the accelerators are not associated in the new dimension. However, a virtual memory associated in all the spatial dimensions is added to the memory hierarchy to model the inter-accelerator connection. Its capacity is the sum of the lowest-level memory of all the accelerators while its bandwidth and cost are the communication bandwidth and cost between the accelerators. Fig. 9(b) illustrates when an NN is distributed to heterogeneous accelerators. In this case, the LF-acc is asymmetric with a different configuration for each physical accelerator and a virtual shared memory that models the communication information is also added. In this work, we only consider up to two heterogeneous physical accelerators in a LF-acc since this provides sufficient system utilization improvement in our experiments. When an NN cannot exploit all the the PEs and memory resources, the physical accelerator is partitioned into LF-accs to process more tasks as shown in Fig. 9(c). In AINNS, different types of LF-accs can be combined. For example, LF-acc in Fig. 9(a) and LF-acc2 in Fig. 9(c) can be scheduled as a LF-acc with heterogeneous physical accelerators as in Fig. 9(b).

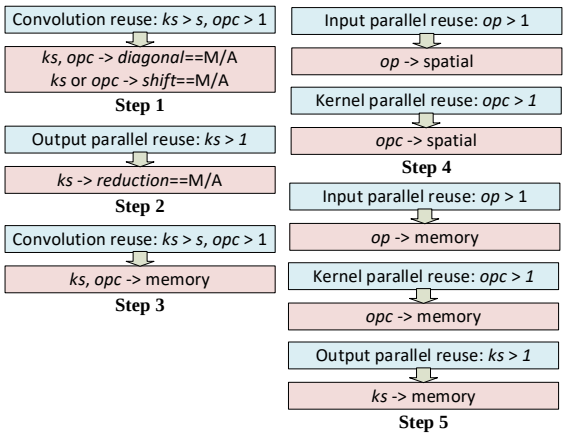

Fig. 10. Local mapping steps. *g* is unrolled only when there is no other parameter. The algorithm is applicable to both homogeneous and heterogeneous LF-accs.

5.2 Local Mapper (LM)

When a network is assigned to an LF-acc, LM schedules how the loops of the NN are unrolled and executed in the LF-acc. Instead of recognizing the dataflow and performing the specific mapping for each accelerator, our LM adopts just one uniform mapping algorithm for all the workloads and accelerators. The critical insight is to fill the most exclusive functions in the LF-acc with the corresponding NN loops first in case they will be left idle.

Fig. 10 shows the main steps of LM. **Step 1** and **Step 2** first spatially map NN dimensions with convolution reuse or output parallel reuse to the spatial dimensions with *diagonal*, *shift* or *reduction* functions. The mandatory functions are filled first to avoid underutilization. Then in **Step 3**, if there is still NN dimension with convolution reuse not mapped to any LF-acc function, the *ks* and *opc* of that dimension are both unrolled temporally for stationary reuse. In **Step 4**, the *op* and *opc* loops are unrolled to fill the spare spatial dimensions. And in **Step 5**, the *op*, *opc* and *ks* loops are unrolled temporally within the hierarchy of memory to reuse the data stationarily.

Note that LM is applicable to both homogeneous and heterogeneous LF-accs. The unrolling factor of each loop is determined by the required resources, e.g., PE size and memory capacity, as discussed in Section 4. For the mappings in each step, there is no required order for the dimensions or parameters to be unrolled. Our experiments show the mapping order affects the average latency and data movement by less than 5%.

5.3 Task Dispatcher (TD)

Since most of the neural networks arrive on the fly, TD is

Algorithm 1: Algorithm for feature-matching dispatching. It is called when a new task arrives or a task commits.

```
1:  for task in scheduling_queue:
2:    if available_home_accelerators not empty:
3:      for acc in available_home_accelerators:
4:        if task.cdr > acc.pbr × matching_threshold and task.ccr
          > acc.cbr × matching_threshold and task.rcr > acc.rpr × match-
          ing_threshold:
5:          home_acc ← acc
6:          break
7:      if not home_acc:
8:        home_acc ← available_home_accelerators[0]
9:      task.LF-acc ← home_acc
10:     deduct home_acc from all LF-acc
11: for task in running_tasks:
12:   while len(task.LF-acc) < partition_rate × len(available_LF-
      accs)
13:     task.LF-acc.add from available_LF-accs
14:     if task.LF-acc.pbr > unusable_threshold or task.LF-acc.
      nmem < unusable_threshold:
15:       task.LF-acc.deduct (added_portion)
16:   task.LF-acc.deduct (unused_portion)
17:   available_LF-accs.add (unused_portion)
```

designed to schedule them at run time heuristically. Algorithm 1 lists the algorithm of TD:

Home designation: The scheduler assigns a home accelerator to each task. It keeps a list of available home accelerators ordered by *npe* or *nmem* based on whether the policy is throughput or energy optimized. When a new task arrives, the scheduler iterates until an accelerator that matches the shape of NN within a threshold is located and assigned to the task (lines 3 to 6). If no matching accelerator is found, the scheduler simply assigns the first accelerator in the list to the task (lines 7 and 8).

NN partitioning: The tasks are distributed when there are idle accelerators. This is materialized by adding idle accelerators to the LF-acc of the tasks. The available_LF-accs list records the idle accelerators or LF-accs that are not in the LF-acc of any task. Starting from the home accelerator (line 9), each running task can add idle accelerators or LF-accs to its own LF-acc. The maximal number of accelerators that can be added by each task is determined by the real-time task arriving rate in the system (line 12). When adding LF-accs, the shape matching should also be checked. Especially, to avoid overwhelming interconnection communication, we check the memory-related features of the newly formed LF-acc and abandon if the overhead is too large (lines 14 and 15). When an accelerator is assigned to a new task as home, it should be deducted from the LF-accs of all (line 10).

Accelerator partitioning: After changing LF-acc, LM is performed to map the NN to its LF-acc. If it cannot exploit the entire LF-acc, the unused portion will be set idle and added to the available_LF-accs list (lines 16 and 17).

Preemption: Preemption is performed in a non-interruptible way in our scheduler to avoid starvation of the low-priority tasks. Specifically, the new tasks can preempt the resources of tasks with lower priority but the home accelerators cannot be preempted.

TABLE 6
MANUAL DATAFLOWS OF ACCELERATORS

Accelerator	Loop order		
	PE Dim1	PE Dim2	Temporal
Eyeriss	$ks_H\_ks_C$	$opc_H\_op_C$	$ks_W\_ks_C\_opc_W$ ...
NLR	ks_C	op_C	$ks_W\_ks_H$ ...
TPU	op_C	$ks_C\_ks_W\_ks_H$	$opc_B\_opc_W\_opc_H$ ...
Eager Pruning	$ks_W\_ks_H\_op_C$	ks_C	$opc_W\_opc_H$ ...

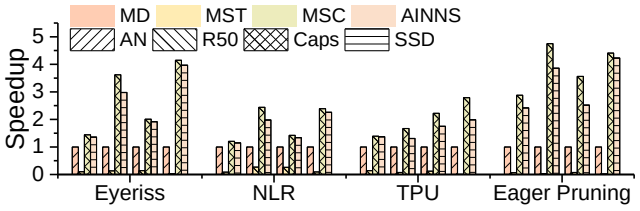

Fig. 11. Speedup of single DNNs, normalized to MD.

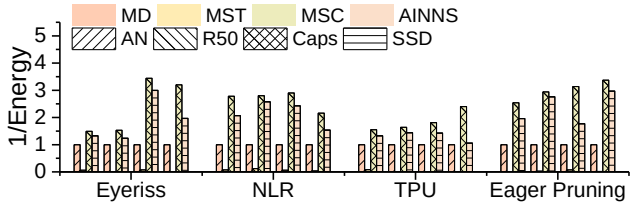

Fig. 12. Data movement energy of single DNNs, normalized to MD.

6 EVALUATION

6.1 Methodologies

In the following sections, the mapping and diapatching functions of AINNS are evaluated independently first. Then we evaluate the overall AINNS system in two case studies where the accelerators and NN workloads are both heterogeneous to show its performance in the AIoT applications.

We derive the experimental results by an in-house simulator. Specifically, the simulator generates the computation and data loading cycles as well as data movement in each memory level. The model is in accordance with those used in previous works [3][8][19]. The inter-accelerator connection latency and data movement model is consistent with [5]. Then the system computation and data movement energy are calculated based on the power estimation obtained by RTL synthesis in Synopsys Design Compiler and memory simulation in CACTI [26]. The scheduling time and energy are derived directly from the processors we run.

We compare the LM and TD of AINNS with different baselines respectively.

Local mapping baselines: LM is compared against two baselines:

Manual dataflow (MD): The scheduler follows the manually optimized dataflow as discussed in Section 3.1 to map the NN on the accelerator. The dataflow of each accelerator is listed in Table 6.

Mapping search (MS): The loop order and unrolling factors are randomly chosen. The optimal plan is selected based on the evaluation results. MST refers to the results of

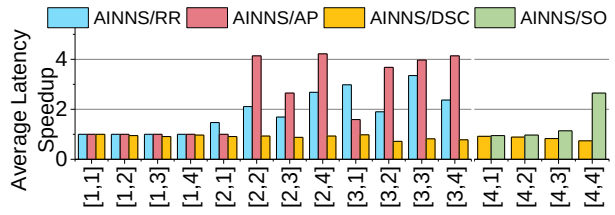

Fig. 13. Average latency speedup when the system is throughput-optimized.

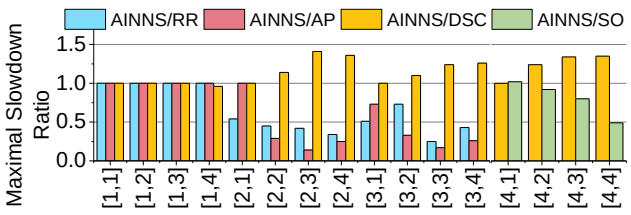

Fig. 14. Maximal slowdown ratio when the system is throughput-optimized.

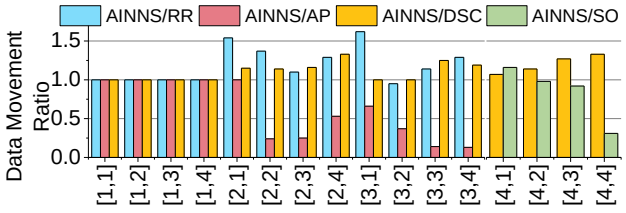

Fig. 15. Data movement ratio when the system is energy-optimized.

MS within the same time as AINNS and MSC refers to the results of MS until convergence.

Task dispatching baselines: For TD, there are two naïve baselines and two mainstream scheduling schemes:

Round-robin (RR): Each NN task is assigned to the next available accelerator and no partitioning is performed.

All-partition (AP): The tasks are processed in sequence. When each task is being processed, its LF-acc is all the accelerators in the system.

Size only (SO) [4][5]: The scheduler only takes the size features (i.e., PE size and memory capacity) of the accelerators into consideration but not the shape features.

Dispatching search (DS) [6]: The scheduler randomly searches for the optimal dispatching plans within the design space. DST and DSC refer to the results within the same time as AINNS and until convergence respectively.

6.2 Mapping Results

We compare the mapping results of four networks, i.e., AlexNet (AN) [28], ResNet50 (R50) [29], CapsuleNet (Caps) [30], SSD-MobileNet (SSD) [31], on the four accelerators in Table 3. Fig. 11 shows the speedup results of three mappers and Fig. 12 shows the total data movement energy consumption. AINNS gains 2.3x speedup and 1.9x energy reduction on average over MD. Compared with the optimum-guaranteed MSC, AINNS achieves 87% (up to 99%) and 78% (up to 94%) of its speedup and data movement energy efficiency. Note that MSC takes more than 1000x longer than AINNS to converge to the optimal results. Within the same scheduling time, MST only generates 6% and 2% speedup and data movement energy efficiency. MD also requires heavy human labor to optimize the dataflow for each accelerator while AINNS can generate effective mapping for any NN on any accelerator in real time.

6.3 Dispatching Results

To demonstrate that AINNS is all-inclusive, we evaluate the dispatcher in all the 16 regions of the NN-accelerator heterogeneity plane shown in Fig. 3. In the single accelerator/NN HPRs, only 1 accelerator is chosen from Eyeriss [19], Eager Pruning [21], NLR [22], TPU v2 [25] or TPU v3 [15] and only 1 NN is chosen from LeNet [32], AlexNet [28], GoogLeNet [33], ResNet34, ResNet50, ResNet101 [29], DenseNet [34] or CapsuleNet [30]. In the other cases, 10 accelerators and 50 networks are randomly chosen following the heterogeneity requirements. Arrival of the tasks follows continuous uniform distribution. To isolate the performance of the dispatcher, we apply AINNS mapping to all the dispatchers.

TABLE 7
TOTAL SCHEDULING TIME

Scheduler	RR	AP	SO	AINNS	DSC
Time (min)	2	2	6	8	185

We compare AINNS with the baselines in terms of average latency for quality of service, maximal slowdown (actual latency/ideal latency of a task) for fairness, total data movement cost for energy efficiency in Fig. 13 to 15. The results in single/homogeneous HPRs are averaged on all the NNs/accelerators and the results in (semi-) heterogeneous HPRs are averaged on 5 different benchmark configurations. SO has the same scheduling results as AINNS in HPRs[1,1] to [3,4] because there is no shape difference in the accelerators. Therefore, AINNS is only compared with RR, AP and DSC in these regions. In HPRs[4,1] to [4,4], since SO and DSC perform the best among the baselines, we omit the others. Table 7 compares the total time to perform scheduling in all the HPRs for once.

Based on the results in Fig. 13 to 15 and Table 7, the performance of AINNS is comparable to DSC, which can be viewed as an optimal ground-truth scheduler, with significantly reduced compiling time (by 23x). For the real time schedulers, AP and RR have similar results as AINNS when the system has single accelerator or neural network. However, if the accelerator array and NN workloads are more complicated, AP and RR fail to render competitive speedup. AP suffers form the worst speedup, slowdown and data movement due to low utilization and high communication. Note that RR has the least data movement since it avoids any inter-accelerator communication but this is accompanied by worse speedup and slowdown because the NN computation cannot be distributed and accelerators with the most resources are not always busy. SO always assigns accelerators with the most resources to the workloads, so its scheduling results can be even better than AINNS in HPRs[4,1] to [4,3]. However, it is less comparable to AINNS in HPR[4,4]. This most challenging region requires accurate matching of the neural networks and the accelerators. The significant speedup (2.7x) and data movement reduction (69%) in this region prove the effectiveness of the shape matching mechanism in AINNS.

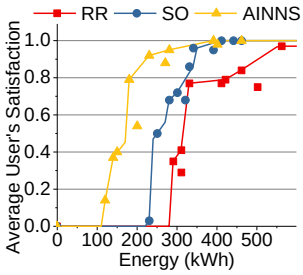

Fig. 16. Average user's satisfaction as the total energy consumption grows.

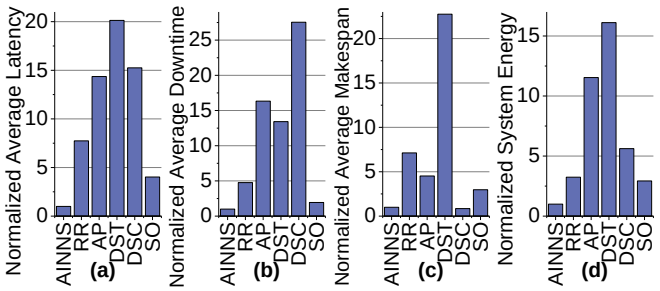

Fig. 17. Comparison of different schedulers in the datacenter.

In the following, we will evaluate the entire AINNS system in two case studies where the accelerators and NN workloads are both heterogeneous to show its potential application in the future AIoT environment.

6.4 Case Study 1: Data Center

The first case study simulates the workloads in the data center. The NN tasks are randomly chosen from MLPerf [35] (an industry-wide standard machine learning benchmark including ResNet50, MobileNet, SSD-ResNet34, SSD- MobileNet and GNMT) and configured as training (low priority), batch inference (medium priority) or real-time inference (batch size is 1, high priority). And their arrival follows the Poisson's distribution.

Since both the quality of service and the service provider's profit should be optimized in the data center [36], we evaluate the user's satisfaction with the NN tasks and the total energy consumption of the system. The user's satisfaction with the latency is calculated based on the model proposed in [37]. Concretely, the satisfaction is scored 1 or 0 if the latency is within an imperceptible latency threshold or beyond an unusable latency threshold. And the satisfaction degrades linearly with the response time between the two thresholds. In the evaluation, we set 1 second and 6 seconds per input for the imperceptible and unusable thresholds respectively for inference [38]. For the training tasks, the imperceptible latency threshold is set to 7 days for the heavy neural networks (i.e., ResNet50, SSD-ResNet34 and GNMT) and 2 days for the light neural networks (i.e., MobileNet and SSD-MobileNet) [28]. The unusable threshold for training is 15 days.

For comprehensiveness, we evaluate the overall performance of the scheduler considering both the mapper and dispatcher. Specifically, RR, AP and SO rely on MD mapper while DS is based on the MS mapper. The scheduler compiling time and energy consumption are counted into the results. They are derived from the power and compiling latency of the schedulers on an Intel Core i7 CPU.

We first study the average user's satisfaction as the system energy grows. Better schedulers can achieve same user's satisfaction with less energy because they have a better utilization of the accelerators and thus less accelerators should be employed. AINNS is evaluated with a heterogeneous accelerator array where all types of accelerators mentioned in Section 6.3 are available. RR is evaluated as a naïve baseline with a homogeneous TPU v2 array. SO is evaluated with a semi-homogeneous accelerator array with two generations of TPUs as a state-of-the-art scheduler. We change the numbers of the accelerators, test the energy and user satisfaction of each sample configuration and plot the upper envelopes of the samples in Fig. 16. As shown, with a more flexible accelerator choice in heterogeneous arrays, AINNS achieves 0.5, 0.8 and 1 user's satisfaction with 30%, 43% and 5% less energy consumption compared with state-of-the-art.

We then compare the performance of all the schedulers under the heterogeneous configuration where AINNS achieves 0.8 user's satisfaction in Fig. 17. With its effectiveness and efficiency in scheduling, AINNS improves the average latency, downtime, makespan and energy efficiency of almost all the baselines. Exhaustive search fails to provide efficient real-time scheduling as DSC suffers from extremely long downtime while DST provides the worst latency among all the baselines.

6.5 Case Study 2: Smart Electric Vehicle

The second case study lies in a smart plug-in hybrid electric vehicle that integrates energy management, traffic sign recognition and driver stress detection. Before the voyage, the energy management is performed by a three-layer feed-forward neural network for driving mode prediction, a two-layer RNN to estimate the co-state of the energy scheduling principles and a radical basis function neural network to predict the future velocity based on the past vehicle speed [39]. During driving, the traffic sign recognition is realized by a multi-task CNN, where a fast binary classification result can be obtained after the first convolution layer and the real classification is provided after the third layer [9]. Another neural network with two hidden layers is also embedded in the system to detect the stress level of the driver using the physiological signals [40]. In the system, the energy management is first performed as a real-time inference. The traffic sign recognition is performed as a batch inference with hard deadline whenever the regions of interests are proposed, which is modeled as a uniformly distributed arriving task in our evaluation. The real classification after the binary classification is performed as a new task and 70% of the real classification can be skipped due to a negative binary classification. The driver stress detection is set as an inference that is performed every second with medium priority. The system also trains the models of stress detection in the case of false positive and energy management if the mode or velocity change during driving. The training tasks are set as low priority.

Traditionally, the neural network workloads at the edge and mobile ends adopt fixed schedule of an accelerator

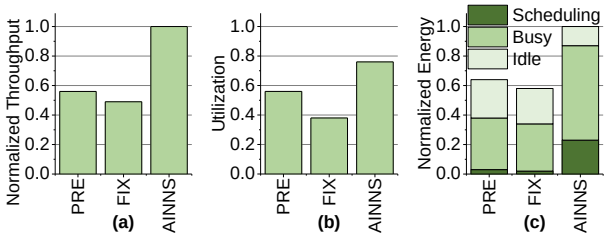

Fig. 18. Comparison of different schedulers in the smart plug-in hybrid electric vehicle.

TABLE 8
COMPUTATION DURING 30-MINUTE VOYAGE

Scheduler	TS	SD	SD-T	SD-A	EM-T	EM-A
AINNS	66	1765	35	5%	13	3%
PRE	66	942	9	1%	3	0%
FIX	66	725	20	3%	6	2%

TS: traffic sign recognition; SD: stress detection; EM: energy management prediction; -T: training; -A: accuracy improvement by training.

array [6] or preemption of a single accelerator [41]. However, since the neural networks are not running continuously but arrive randomly on the fly, the fixed allocation of the accelerators and the preemption may result in underutilization of the resources. In this case study, we compare the total throughput of our proposed AINNS with the fixed allocation (FIX) and preemption (PRE) mechanisms under the same computation resources.

In AINNS and FIX, assume there is an NLR and two Eyeriss chips for the neural network computation in the system. FIX in essence leverages DSC and MSC to determine the optimal schedules of all the possible tasks in ahead while AINNS schedules the workloads in real time. Here, to ensure that the most significant work, i.e., traffic sign recognition, can be completed within the hard deadline (i.e., 0.5 seconds [42]), FIX allocates sufficient resources to it and AINNS modifies its policy to allow traffic sign recognition to preempt all the other workloads. PRE is similar to AP by allocating all the resources to only one workload simultaneously while also allowing the most significant work to preempt. For best results, we configure the accelerator in PRE as a TPU with 28×28 PE array. In this case study, the scheduling is performed on an Arm Cortex-A78AE CPU.

Fig. 18 shows the average system throughput and system energy consumption of AINNS, PRE and FIX during a 30-minute voyage. Although AINNS consumes 36% more energy than PRE due to the scheduling overhead and more work performed, its average power (i.e., 9.1W) is still acceptable in a smart car system [43], let alone that it improves the throughput by 1.8x. Specifically, as shown in Table 8, AINNS performs 823 more driver stress detections, 15 and 7 more iterations of training for stress detection and energy management prediction, which improves the accuracies of the models by 2% and 1% respectively.

7 RELATED WORK

To cater to the heterogeneity requirement in the neural network applications, the computer architecture community has seen a trend that the accelerators and schedulers are more *inclusive*. During the early years, most works only focus on the acceleration of a single network on a fixed-size PE array. Located in HPR[1,1], these accelerators mainly explore the local dataflow to maximize the performance and energy efficiency of a single chip.

TABLE 9
COMPARISON OF REALTED WORKS

Scheduler	Heterogeneity		Real-time	Dis-patch	Map	Baseline
	NN	Accelerator				
AINNS	√	√	√	√	√	
Gavel [4]	√	○	√	√	-	SO
TVM [45]	√	-	-	-	√	MSC
PREMA [41]	√	-	√	√	-	AP
AccPar [5]	-	○	-	√	-	SO, AP
Herald [6]	√	○	-	√	√	DSC, MSC
MT [27]	√	-	-	√	√	
Interstellar [7]	-	√	-	-	√	MD, MSC

√: supports the corresponding function; ○: only supports the semi-heterogeneous accelerators/NNs; -: not supported.

The "accelerator wall" and the limited parallelism in a network then motivated several studies to scale up the system. [44] splits the computation of each layer and assign the portions to different accelerators while [23] deploys different layers to different accelerators and process the inputs in a pipeline. The other studies propose to accommodate several networks in one architecture at the same time to increase the utilization [27]. However, these works only manage accelerators with the same structure and handle the co-execution of NNs of the same type because the multiple tenants share the same dataflow (up to HPR[2,3]).

There are also mature data center machine learning compilers [45] that translate any given NN into standard intermediate representation and automatically map it to the underlying hardware. They can cover any heterogeneous workloads but are still not able to grasp the heterogeneity in the accelerators (up to HPR[2,4]). To this end, several works are recently proposed to schedule and map the networks in heterogeneous acelerator arrays. [4][5] are proposed to dictate the partitioning of neural network layers and the scheduling of the networks on the accelerators based on the performance or throughput of each pair of accelerator and workload. Unfortunately, without a formal approach to recognition of different accelerator architectures and the corresponding dataflows, it is difficult for them to accurately estimate the performance and throughputs, so they only make it to HPR[3,4]. [6] searches for the optimal templates and resource allocations for each accelerator as well as the mapping of each network. It takes heterogeneity in both NN and accelerator into consideration but is impossible to perform in real time. Table 9 compares the functions of the representative related works and the baselines that they resemble the most.

8 CONCLUSION

This paper aims to address the increasing heterogeneity in both neural network computation and accelerators. We

first build a model to formalize various NN computation and accelerators so that the NNs are transformed into a series of entities with parameterized shapes in multiple dimensions and the accelerators are described as multi-dimension containers with shapes defined under the same standard to accommodate them. This allows us to turn the complicated and ad hoc scheduling in the heterogeneous environment into a straightforward shape matching and accelerator filling algorithm. We exploit this opportunity to propose an all-inclusive neural network scheduler (AINNS) which effortlessly and universally dispatches and maps the neural network tasks in the accelerator array with any level of heterogeneity. Its simplicity and effectiveness allow AINNS to perform better than state-of-the-art approaches that either utilize exhaustive search or perform scheduling with no awareness of the heterogeneity in the architectures.

ACKNOWLEDGMENT

We appreciate the Department of Electrical and Computer Engineering, University of Florida and the Margaret A. Ross Fellowship for their support.

REFERENCES

[1] J. Zhang and D. Tao, "Empowering Things with Intelligence: A Survey of the Progress, Challenges, and Opportunities in Artificial Intelligence of Things," *IEEE Internet Things J.*, 2020.

[2] S. Liu *et al.*, "Cambricon: An Instruction Set Architecture for Neural Networks," in *Proceedings of the 43rd International Symposium on Computer Architecture (ISCA)*, 2016, pp. 393–405.

[3] J. Zhang, X. Chen, and S. Ray, "GCONV Chain: Optimizing the Whole-life Cost in End-to-end CNN Acceleration," *IEEE Trans. Comput.*, 2021.

[4] D. Narayanan, K. Santhanam, A. Phanishayee, F. Kazhamiaka, and M. Zaharia, "Heterogeneity-Aware Cluster Scheduling Policies for Deep Learning Workloads," in *USENIX*, 2020, pp. 481–498.

[5] L. Song, F. Chen, Y. Zhuo, X. Qian, H. Li, and Y. Chen, "AccPar: Tensor Partitioning for Heterogeneous Deep Learning Accelerators," in *Proceedings of International Symposium on High-Performance Computer Architecture (HPCA)*, 2020, pp. 342–355.

[6] H. Kwon, L. Lai, M. Pellauer, T. Krishna, Y. H. Chen, and V. Chandra, "Heterogeneous Dataflow Accelerators for Multi-DNN Workloads," in *Proceedings of International Symposium on High-Performance Computer Architecture (HPCA)*, 2021, vol. 2021-Febru, pp. 71–83.

[7] X. Yang *et al.*, "Interstellar: Using Halide's Scheduling Language to Analyze DNN Accelerators," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2020, pp. 369–383.

[8] A. Parashar *et al.*, "Timeloop: A Systematic Approach to DNN Accelerator Evaluation," in *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2019.

[9] H. Luo, Y. Yang, B. Tong, F. Wu, and B. Fan, "Traffic Sign Recognition Using a Multi-Task Convolutional Neural Network," *IEEE Trans. Intell. Transp. Syst.*, vol. 19, no. 4, pp. 1100–1111, Apr. 2018.

[10] T. Elsken, J. H. Metzen, and F. Hutter, "Neural Architecture Search: A Survey," *J. Mach. Learn. Res.*, vol. 20, pp. 1–21, 2019.

[11] H. Li, J. Li, and X. Han, "Robot Vision Model Based on Multi-Neural Network Fusion," in *Proceedings of 2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference, ITNEC 2019*, 2019, pp. 2571–2577.

[12] M. Tremblay, M. S. Gupta, C. Y. Google, M. N. Facebook, G. Diamos, and B. A. Iyer, "Panel-The Impact of AI Workloads on Datacenter Compute and Memory Samsung @ The Heart of Your Data." [Online]. Available: https://code.fb.com/ai-research/scaling-neural-machine-translation-to-bigger-data-sets-with-faster-training-and-inference. [Accessed: 18-Nov-2020].

[13] A. Prakash, N. Ramakrishnan, K. Garg, and T. Srikanthan, "Accelerating Computer Vision Algorithms on Heterogeneous Edge Computing Platforms," in *IEEE Workshop on Signal Processing Systems (SiPS)*, 2020, vol. 2020-October.

[14] "Heterogeneous Computing as a Next- Generation Architecture for Scaling Data Centers: Trends, Opportunities, Solutions | by Artavazd Khachatryan | Grovf | Medium." [Online]. Available: https://medium.com/grovf/heterogeneous-computing-as-a-next-generation-architecture-for-scaling-data-centers-trends-ae5c5f5725f9. [Accessed: 18-Nov-2020].

[15] "System Architecture | Cloud TPU | Google Cloud." [Online]. Available: https://cloud.google.com/tpu/docs/system-architecture. [Accessed: 18-Nov-2020].

[16] "Connecting the Dots: AI at the Edge :: Omdia." [Online]. Available: https://omdia.tech.informa.com/campaign/connecting-the-dots-ai-at-the-edge#Form. [Accessed: 27-Sep-2021].

[17] S. Wang, J. Xu, N. Zhang, and Y. Liu, "A Survey on Service Migration in Mobile Edge Computing," *IEEE Access*, vol. 6, pp. 23511–23528, Apr. 2018.

[18] L. Pacheco, H. Oliveira, D. Rosario, E. Cerqueira, L. Villas, and T. Braun, "Service Migration for Connected Autonomous Vehicles," in *Proceedings - IEEE Symposium on Computers and Communications*, 2020, vol. 2020-July.

[19] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," *IEEE SOLID-STATE CIRCUITS*, vol. 1, 2016.

[20] W. Lu, G. Yan, J. Li, S. Gong, Y. Han, and X. Li, "FlexFlow: A Flexible Dataflow Accelerator Architecture for Convolutional Neural Networks," in *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, 2017, pp. 553–564.

[21] J. Zhang, X. Chen, M. Song, and T. Li, "Eager Pruning: Algorithm and Architecture Support for Fast Training of Deep Neural Networks," in *Proceedings of the 46th International Symposium on Computer Architecture (ISCA)*, 2019, pp. 292–303.

[22] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2015, pp. 161–170.

[23] S. Venkataramani *et al.*, "Scaledeep: A Scalable Compute Architecture for Learning and Evaluating Deep Networks," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2017, vol. Part F1286, pp. 13–26.

[24] Z. Du *et al.*, "ShiDianNao: Shifting Vision Processing Closer to the Sensor," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture (ISCA)*, 2015.

[25] N. P. Jouppi *et al.*, "In-Datacenter Performance Analysis of a Tensor Processing Unit," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2017, pp. 1–12.

[26] "CACTI 6.0: A Tool to Model Large Caches." [Online]. Available: https://www.hpl.hp.com/techreports/2009/HPL-2009-85.html. [Accessed: 14-Mar-2021].

[27] E. Baek, D. Kwon, and J. Kim, "A Multi-Neural Network Acceleration Architecture," in *Proceedings of International Symposium on Computer Architecture (ISCA)*, 2020, pp. 940–963.

[28] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, Jun. 2017.

[29] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[30] S. Sabour, N. Frosst, and G. E. Hinton, "Dynamic Routing Between Capsules," *Adv. Neural Inf. Process. Syst.*, vol. 2017-Decem, pp. 3857–3867, Oct. 2017.

[31] W. Liu *et al.*, "SSD: Single Shot MultiBox Detector," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2016, vol. 9905 LNCS, pp. 21–37.

[32] Y. LeCun *et al.*, "Handwritten digit recognition: applications of neural network chips and automatic learning," *Commun. Mag.*, 1989.

[33] C. Szegedy *et al.*, "Going Deeper with Convolutions," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2015, vol. 07-12-June, pp. 1–9.

[34] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely Connected Convolutional Networks," in *Proceedings of the 30th Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, vol. 2017-Janua, pp. 2261–2269.

[35] V. J. Reddi *et al.*, "MLPerf Inference Benchmark," in *Proceedings - International Symposium on Computer Architecture*, 2020, vol. 2020-May, pp. 446–459.

[36] S. Jang, T. Y. Kim, J. Kim, and J. Lee, "The Study of Genetic Algorithm-based Task Scheduling for Cloud Computing," *undefined*, 2012.

[37] M. Song, Y. Hu, H. Chen, and T. Li, "Towards Pervasive and User Satisfactory CNN across GPU Microarchitectures," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2017, pp. 1–12.

[38] "Best Server and Application Response Time Monitoring Tools + Guide - DNSstuff." [Online]. Available: https://www.dnsstuff.com/response-time-monitoring. [Accessed: 21-Nov-2020].

[39] Y. Wu, Y. Zhang, G. Li, J. Shen, Z. Chen, and Y. Liu, "A Predictive Energy Management Strategy for Multi-Mode Plug-In Hybrid Electric Vehicles Based on Multi Neural Networks," *Energy*, vol. 208, p. 118366, Oct. 2020.

[40] A. Saeed and S. Trajanovski, "Personalized Driver Stress Detection with Multi-task Neural Networks using Physiological Signals," in *arXiv:1711.06116*, 2017.

[41] Y. Choi and M. Rhu, "PREMA: A Predictive Multi-Task Scheduling Algorithm for Preemptible Neural Processing Units," in *Proceedings - 2020 IEEE International Symposium on High Performance Computer Architecture, HPCA 2020*, 2020, pp. 220–233.

[42] J. Khoury, K. Amine, and R. A. Saad, "An Initial Investigation of the Effects of a Fully Automated Vehicle Fleet on Geometric Design," *J. Adv. Transp.*, vol. 2019, 2019.

[43] "Self-Driving Cars and Power Consumption — New Chip Designs | by Nitin Vaish | Medium." [Online]. Available: https://nitinvaish.medium.com/self-driving-cars-and-power-consumption-new-chip-designs-4c723659f8cd. [Accessed: 22-Nov-2020].

[44] L. Song, J. Mao, Y. Zhuo, X. Qian, H. Li, and Y. Chen, "HyPar: Towards hybrid parallelism for deep learning accelerator array," in *Proceedings of International Symposium on High-Performance Computer Architecture (HPCA)*, 2019, pp. 56–68.

[45] T. Chen *et al.*, "TVM: An Automated End-to-End Optimizing Compiler for Deep Learning," in *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation*, 2018, pp. 578–594.

Jiaqi Zhang received the B.S. degree in Communication Engineering from Beijing Jiaotong University in 2016. She is currently a Ph.D. candidate in the Department of Electrical and Computer Engineering, University of Florida. Her research interests lie in software and hardware acceleration of emerging algorithms and applications including machine learning and IoT.

Xiangru Chen received the B.S. degree in Electronic Information Engineering from Shandong University in 2016 and M.S. degree in Electrical and Computer Engineering from University of Florida in 2018. He is currently pursuing a Ph.D. degree in the Department of Electrical and Computer Engineering, University of Florida. His research focuses on the architecture support for ML applications.

Sandip Ray is an Endowed IoT Term Professor at the Department of Electrical and Computer Engineering, University of Florida. His research involves developing correct, dependable, secure, and trustworthy computing through cooperation of specification, synthesis, architecture and validation technologies. He focuses on next generation computing applications, including IoT, autonomous automotive systems, etc. Before joining University of Florida, he was a Senior Principal Engineer at NXP Semiconductors, where he led the R&D on security architecture and validation of hardware platforms for automotive and IoT applications. Prior to that, he was a Research Scientist at Intel Strategic CAD Labs, where he led research on validation technologies for security and functional correctness of SoC designs. Dr. Ray is the author of three books and over 90 publications in international journals and conferences. He has a Ph.D. from University of Texas at Austin and is a Senior Member of IEEE.