# NASA: Neural Architecture Search and Acceleration for Hardware Inspired Hybrid Networks

Huihong Shi[1,2*], Haoran You[1], Yang Zhao[1], Zhongfeng Wa ng[2], and Yingyan Lin[1]
[1]Rice University, [2]Nanjing University
{eiclab,hy34,zy34,yingyan.lin}@rice.edu,zfwang@nju.edu.cn

## ABSTRACT

Multiplication is arguably the most cost-dominant operation in modern deep neural networks (DNNs), limiting their achievable efficiency and thus more extensive deployment in resource-constrained applications. To tackle this limitation, pioneering works have developed handcrafted multiplication-free DNNs, which require expert knowledge and time-consuming manual iteration, calling for fast development tools. To this end, we propose a **N**eural **A**rchitecture **S**earch and **A**cceleration framework dubbed NASA, which enables automated multiplication-reduced DNN development and integrates a dedicated multiplication-reduced accelerator for boosting DNNs' achievable efficiency. Specifically, NASA adopts neural architecture search (NAS) spaces that augment the state-of-the-art one with hardware inspired multiplication-free operators, such as shift and adder, armed with a novel progressive pretrain strategy (PGP) together with customized training recipes to automatically search for optimal multiplication-reduced DNNs; On top of that, NASA further develops a dedicated accelerator, which advocates a chunk-based template and auto-mapper dedicated for NASA-NAS resulting DNNs to better leverage their algorithmic properties for boosting hardware efficiency. Experimental results and ablation studies consistently validate the advantages of NASA's algorithm-hardware co-design framework in terms of achievable accuracy and efficiency tradeoffs. Codes are available at https://github.com/shihuihong214/NASA.

## 1 INTRODUCTION

Modern deep neural networks (DNNs) have achieved great success in various computer vision tasks [7, 14, 15, 17], which has motivated a substantially increased demand for DNN-powered solutions in numerous real-world applications. However, the extensively used multiplications in DNNs dominate their energy consumption and have largely challenged DNNs' achievable hardware efficiency, motivating multiplication-free DNNs that adopt hardware-friendly operators, such as additions and bit-wise shifts, which require a smaller unit energy and area cost as compared to multiplications (see Table 1 [26]). In particular, pioneering works of multiplication-free DNNs include (1) DeepShift [6] which proposes to adopt merely shift layers for DNNs, (2) AdderNet [20] which advocates using adder layers to implement DNNs for trading the massive multiplications with lower-cost additions, and (3) ShiftAddNet [26] which combines both shift and adder layers to construct DNNs for better trading-off the achievable accuracy and efficiency.

Despite the promising hardware efficiency of multiplication-free DNNs, their models' expressiveness capacity and thus achievable accuracy are in general inferior to their multiplication-based counterparts. As such, it is highly desired to develop hybrid multiplication-reduced DNNs that integrate both multiplication-based and multiplication-free operators (e.g., shift and adder) to boost the hardware efficiency while maintaining the task accuracy. Motivated by the recent success of neural architecture search (NAS) in automating the design of efficient and accurate DNNs, one natural thought is to leverage NAS to automatically search for the aforementioned hybrid DNNs for various applications and tasks, each of which often requires a different accuracy-efficiency tradeoff and thus calls for a dedicated design of the algorithms and their corresponding accelerators.

**Table 1: Unit energy/area from 45nm implementations [26].**

| Operator | Format | Energy (pJ) | Improv. | Area ($\mu m^2$) | Improv. |
|---|---|---|---|---|---|
| Mult. | FP32 | 3.7 | - | 7700 | - |
| | FXP32 | 3.1 | - | 3495 | - |
| | FXP8 | 0.2 | - | 282 | - |
| Adder | FP32 | 0.9 | 4.1× | 4184 | 1.8× |
| | FXP32 | 0.1 | 31× | 137 | 25.5× |
| | FXP8 | 0.03 | 6.7× | 36 | 7.8× |
| Shift | FXP32 | 0.13 | 24× | 495 | 7.1× |
| | FXP8 | 0.024 | 8.3× | 38 | 7.3× |

In parallel, various techniques [1, 5, 16, 21, 28, 29] have been proposed to boost the hardware efficiency of DNNs, promoting their real-world deployment from the hardware perspective. For example, Eyeriss [5] proposes a row stationary dataflow and a micro-architecture with hierarchical memories to enhance data locality and minimize the dominant data movement cost; and [21] explores a low-bit quantization algorithm paired with a minimalist hardware design for AdderNet [20] to leverage its algorithmic benefits for boosted hardware efficiency. While it has been shown that dedicated accelerators can achieve up to three orders-of-magnitude efficiency improvement as compared to general computing platforms, such as GPUs and CPUs, existing accelerators are customized for either multiplication-based or multiplication-free DNNs, and thus could not fully leverage the algorithmic properties of the aforementioned hybrid DNNs for maximal efficiency. Thus, it is promising and desirable to develop dedicated accelerators for hybrid DNNs consisting of both multiplication-based and multiplication-free operators, which yet is still underexplored.

To marry the best of both worlds - the higher achievable accuracy of multiplication-based DNNs and the better hardware efficiency of multiplication-free DNNs, we target the ***exploration*** *and* ***acceleration*** *of* ***hybrid DNNs***, and make the following contributions:

- We propose **NASA**, a **N**eural **A**rchitecture **S**earch and **A**cceleration framework (see Fig 1) to search for and accelerate hardware inspired hybrid DNNs. To the best of our

[*] Work done when Huihong was a visiting scholar at Rice University. Correspondence should be addressed to: Zhongfeng Wang and Yingyan Lin.

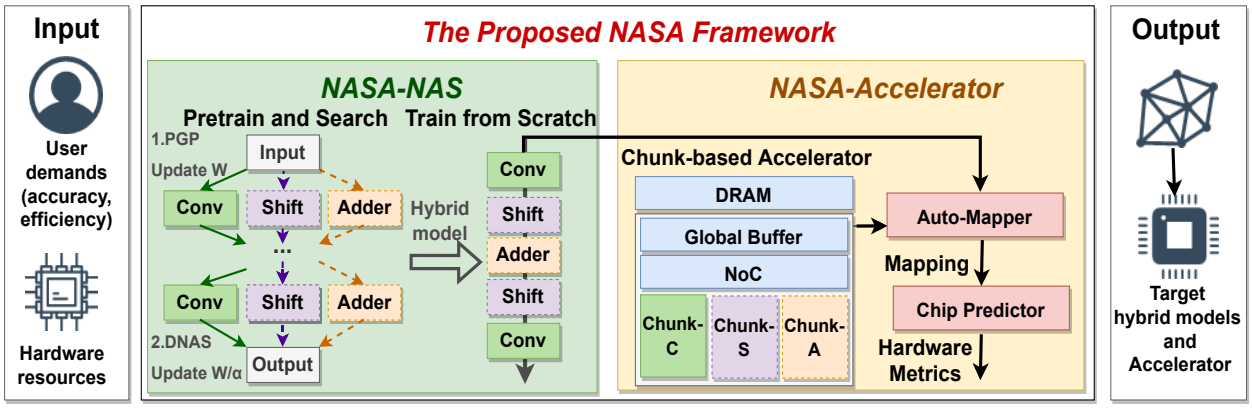


**Figure 1: An overview of our NASA framework integrating a neural architecture search (NAS) and acceleration engine dedicated for hybrid DNNs, where NASA-NAS searches for hybrid models via NAS with the proposed progressive pretrain strategy (PGP) while NASA-Accelerator advocates a dedicated chunk-based accelerator armed with an auto-mapper to support hybrid models searched by NASA-NAS.**

knowledge, NASA is ***the first*** algorithm and hardware co-design framework dedicated for hybrid DNN models.

- We develop a dedicated NAS engine called NASA-NAS integrated in NASA, which incorporates hardware-friendly shift layers [6] and/or adder layers [20] into a state-of-the-art (SOTA) hardware friendly NAS search space [22] to construct *hybrid DNN search spaces*. Furthermore, to enable effective NAS on top of the hybrid search space, we propose a ***Pro**Gressive **P**retrain strategy (PGP)* paired with *customized training recipes*.
- We further develop a dedicated accelerator called NASA-Accelerator to better leverage the algorithmic properties of hybrid DNNs for improved hardware efficiency. Our NASA-Accelerator advocates *a dedicated chunk-based accelerator* to better support the heterogeneous layers in hybrid DNNs, and integrates *an auto-mapper* to automatically search for optimal dataflows for executing hybrid DNNs in the above chunk-based accelerators to further improve efficiency.
- Extensive experiments and ablation studies validate the effectiveness and advantages of our NASA in terms of achievable accuracy and efficiency tradeoffs, against both SOTA multiplication-free and multiplication-based systems. We believe our work can open up an exiting perspective for the exploration and deployment of multiplication-reduced hybrid models to boost both task accuracy and efficiency.

## 2 RELATED WORKS

### 2.1 Multiplication-free DNNs

To favor hardware efficiency, pioneering efforts have been made to replace the cost-dominant multiplications in vanilla DNNs with more hardware-friendly operators, e.g., bit-wise shift and adder, for enabling handcrafted multiplication-free DNNs. For instance, ShiftNet [23] treats shift operations as a zero flop/parameter alternative to spatial convolutions and advocates DNNs featuring shift layers; DeepShift [6] substitutes multiplications with bit-wise shifts; AdderNets [20] trades multiplications with lower-cost additions and employs an $\ell 1$-normal distance as a cross-correlation substitute to measure the similarity between input features and weights; and inspired by a common hardware practice that implements multiplications with logical bitwise shifts and additions [25], ShiftAddNet [26] unifies both shift and adder layers to design DNNs with merely shift and adder operators. However, multiplication-free DNNs in general are still inferior to their multiplication-based counterparts in terms of task accuracy, motivating our NASA framework aiming to marry the best of both worlds from powerful multiplication-based and hardware efficient multiplication-free DNNs.

### 2.2 Neural Architecture Search

Early NAS methods [13, 31, 32] utilize reinforcement learning (RL) to search for DNN architectures, which gained great success but were resource- and time-consuming. To tackle this limitation, weight sharing methods [3, 11, 12] have been proposed. Among them, differentiable NAS (DNAS) algorithms [3, 11, 22] have achieved SOTA results by relaxing the discrete search space to be continuous and then applying gradient-based optimization methods to find optimal architectures from a pre-defined differentiable supernet. Specifically, FBNet [22] employs a Gumbel Softmax sampling method [9] and gradient-based optimization to search for efficient and accurate DNNs targeting mobile devices; FBNetV2 [18] designs a masking mechanism for feature map reuses in both spatial and channel dimensions, expanding the search space greatly at a cost of a small memory overhead; and alternatively, ProxylessNAS [3] activates only a few paths during the forward and backward processes of search, making it possible for DNAS to optimize with large search spaces. Despite the prosperity of NAS for vanilla DNNs, there still exists a lack of effort in exploring NAS designs for hybrid DNNs.

### 2.3 DNN Accelerators

Apart from algorithmic optimization, many works [1, 5, 16, 21, 28, 29] have proposed dedicated DNN accelerators to boost DNNs' hardware efficiency. Among them, Eyeriss [5] proposes a row stationary dataflow and a micro-architecture with hierarchical memories to enhance data locality and minimize dominant data movement costs; [24] proposes AutoDNNchip, a DNN chip generator that can automatically generate both FPGA- and ASIC-based DNN chip implementation given DNNs from machine learning frameworks for a designated application and dataset; and [16] builds a DNN

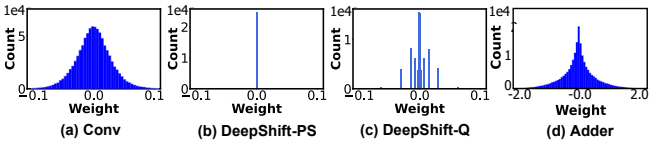


**Figure 2: Illustrating the weight distributions of (a) convolutions, shift layers constructed with (b) DeepShift-PS and (c) DeepShift-Q, respectively, and (d) adder layers, from a NASA-NAS searched hybrid-all model on CIFAR100.**

accelerator to increase the overall acceleration throughput by partitioning the FPGA resources into multiple specialized processors for various convolution layers. Nevertheless, existing mainstream accelerators are dedicated for either multiplication-based DNNs or multiplication-free DNNs (e.g., [21] for AdderNet [20]), which thus can not fully leverage our target hybrid DNNs' hardware efficiency, calling for dedicated accelerators to unleash hybrid DNNs' full efficiency potential.

## 3 PROPOSED NASA-NAS ENGINE

In this section, we first introduce our proposed search space that integrates both multiplication-free operators and multiplication-based convolutions, and then present our progressive pretrain strategy (PGP) proposed to enable effective training of hybrid supernets (see Sec. 3.2). Finally, Sec. 3.3 describes the adopted NAS algorithm based on PGP to search for hardware-inspired hybrid DNNs.

### 3.1 NASA-NAS's Search Spaces

**Basic operators.** To push forward the frontier of achievable task accuracy and efficiency tradeoffs, our NASA-NAS search space unifies both hardware-friendly multiplication-free operators, such as coarse-grained *shift layers* [6] and fine-grained *adder layers* [20], and multiplication-based *convolutions* to construct hybrid search spaces. Next, we will introduce the basic shift and adder layers.

- **Shift layers.** The promising hardware efficiency of bit-wise shift operations has motivated implementing DNNs with shift layers [6] as formulated in Eq. (1), where the weight $W_{shift}$ can be constructed through two ways: *DeepShift-PS* and *DeepShift-Q*. As formulated in Eq. (2), DeepShift-PS intuitively parametrizes $W_{shift}$ with sign flipping $s$ and bit-wise shift $p$, in which $s \in [-1, 0, 1]$ and $p$ is an integer with an absolute value normally greater than one. Given the example that $W_{shift} = -0.25/-0.125$, then the parameter $s = -1$ and $p = -2/-3$, and thus we can easily conclude that the smaller $W_{shift}$ comes with the larger absolute value of $p$. As such, parameters of shift layers that have larger absolute values are naturally incompatible with the small weights in convolutions (see Fig. 2a), yielding $W_{shift}$ of shift layers in hybrid DNNs to be zero (see Fig. 2b).

  To tackle the issue above, we leverage a more training friendly way dubbed *DeepShift-Q* to build our hybrid DNNs, which generates $W_{shift}$ by quantizing the vanilla weights in convolutions $w^*$ to power of two following Eq. (3) instead of directly optimizing $p$ and $s$, and Fig. 2c demonstrates its effectiveness.

$$Y = \sum X^{\mathrm{T}} * W_{shift}. \tag{1}$$

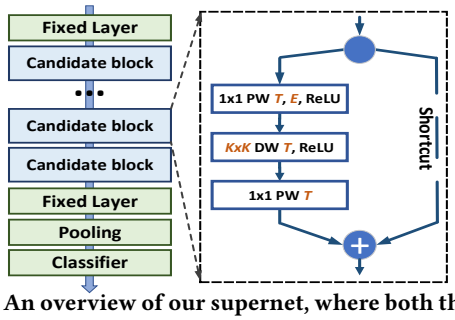


**Figure 3: An overview of our supernet, where both the macro-architecture (left) and the detailed candidate blocks (right) are shown. Here PW, and DW denote pointwise and depthwise layers, and $E, K$, and $T$ donate the channel expansion ratio of candidate blocks, kernel size in depthwise layers, and layer type, respectively.**

$$W_{shift} = s * 2^{p}. \tag{2}$$

$$\hat{s} = sign(w^*),\ \hat{p} = round(log_2|w^*|),\ w_{shift} = \hat{s} * 2^{\hat{p}}. \tag{3}$$

- **Adder layers.** The multiplication-intensive convolutions in DNNs are essentially cross-correlation functions used to measure the relevance between activations and weights. Alternatively, AdderNet [20] proposes adder layers, which leverage computational efficient additions and calculate the $\ell 1$-norm distance to measure the relevance between activations and weights, as expressed in Eq. (4). Due to the different properties of adder layers and convolutions, existing pretrain and NAS algorithms are not directly applicable to handle our hybrid-adder search space, motivating our proposed *progressive pretrain strategy* introduced in Sec. 3.2

$$Y = \sum -|X - W_{adder}|. \tag{4}$$

Based on the above shift and adder layers, we construct three hybrid search spaces: ***hybrid-shift***, ***hybrid-adder***, and ***hybrid-all***, as summarized in Table 2, by integrating them separately or jointly into the SOTA convolutional one [22].

**Supernet.** Built upon the proposed search spaces above, we construct our supernet in Fig. 3. As depicted in Fig. 3 (left), we adopt a macro-architecture following [22] for ease of benchmarking with SOTA efficient DNNs like [22], where the first and last three layers are fixed and the rest candidate blocks are searchable from our pre-defined search spaces. As shown in Fig. 3 (right), each candidate block consists of two pointwise layers (PW) and one depthwise layer (DW), and is characterized by three hyperparameters: the channel expansion ratio of the block $E$, kernel size of the DW layer $K$, and layer type $T$. The configurations of these hyperparameters are summarized in Table 2: All three search spaces contain the same options for $E$ and $K$ but *different choices for* $T$, e.g., the hybrid-shift/adder search space integrates convolutions with only shift/adder layers, while the hybrid-all one includes both of them; Additionally, a *skip* operator is included to skip unnecessary blocks and thus allow shallower DNNs. It is worth noting that we adopt shared weights between candidate blocks with the same layer type $T$ and kernel size $K$ among the channel dimension $E$ to enable effective NAS, inspired by [19].

In summary, there are a total of 22 layers to be searched in our supernet as [22], each of them can select from 13 or 19 (corresponding to the num. of $(E, K) \times$ the num. of $T + 1$) candidate blocks from

**Table 2: Configurations of candidate blocks in our pre-defined search spaces, where $E, K, T$, and Skip donate the expansion ratio, kernel size, layer type, and skip operators, respectively, and Conv, Shift, Adder represent convolutions, shift layers, and adder layers.**

| Hyperparameters | Choices | Search Spaces |
|---|---|---|
| $(E, K)$ | (1,3),(3,3),(6,3)<br>(1,5),(3,5),(6,5) | All Search Spaces |
| $T$ | Conv, Shift<br>Conv, Adder<br>Conv, Shift, Adder | Hybrid-Shift<br>Hybrid-Adder<br>Hybrid-All |
| Skip | – | All Search Spaces |

Table 2 to construct the hybrid-shift/hybrid-adder or hybrid-all search spaces, respectively. For instance, there are (1) 6 choices for $(E, K)$, pairing the expansion ratio $E$ and kernel size $K$, (2) 2 (or 3) choices for the layer type $T$, and (3) 1 skip operation in the hybrid-shift/adder (or hybrid-all) search space based on Table 2; As such, each searchable layer in our hybrid-shift/adder (or hybrid-all) supernet can choose from $6 \times 2 + 1 = 13$ (or $6 \times 3 + 1 = 19$) candidate blocks. Hence, there are a total of $13^{22}$ or $19^{22}$ potential architectures in our pre-defined search spaces.

## 3.2 NASA-NAS's Progressive Pretrain Strategy

We empirically find that directly applying the SOTA NAS pretrain and search method [22] to our *hybrid-adder* search space consistently suffers from the non-convergence issue, and identify that this is caused by the distinct weight distributions of adder and convolutional layers (denoted as conv). Specifically, weights of adder layers tend to follow a Laplacian weight distribution (see Fig. 2d), while those of convolutional layers normally follow a Gaussian distribution (see Fig. 2a). To tackle this issue, we propose a *progressive pretrain strategy (PGP)* that can enable effective pretraining of our target hybrid-adder supernets through *three stages*: (1) conv pretraining, (2) adder pretraining with fixed conv, and (3) mixture pretraining, paired with our *customized training recipe* (i.e., bigger learning rate (lr) and initialization).

Next, we elaborate the three stages above. Specifically, in the first (1) conv pretraining stage, we only forward and backward the convolution-based blocks to first leverage the high-speed convergence benefit of vanilla CNNs, providing a good initialization for the target hybrid supernet; in stage (2), we freeze the well-pretrained weights in convolutional layers to facilitate training the adder layers, in which we forward both convolutional and adder layers but only backward the latter during the backward process; and finally in the (3) mixture pretraining stage, we free up the previously fixed convolutional layers and simultaneously optimize all the candidate blocks to coordinate their parameters towards optimal model accuracy. As for the customized *training recipe*, we empirically observe that the lr can greatly affect the optimization of hybrid-adder supernets, and a *bigger lr* can accelerate the convergence; Additionally, we find that setting the learnable scaling coefficient $\gamma$ in the last batch normalization layer of each candidate block to 0, similar to [27], favors effective training of hybrid supernets.

Note that while we introduce the non-convergence issue above using the *hybrid-adder* search space, it also exists in the *hybrid-all* search space where our PGP strategy is equally effective. On the other hand, for the *hybrid-shift* search space, the vanilla pretrain method in [22] is sufficient. More details can be found in Sec. 5.3.

## 3.3 NASA-NAS's Search Algorithm

In NASA-NAS, we adopt SOTA differentiable NAS (DNAS) algorithm [22], which relaxes the discrete search space to be continuous and then applies gradient-based optimization to optimize the weights $w$ and architectural parameters $\alpha$ in pre-defined differentiable supernets, to enable effective search for our target hybrid models. Specifically, we update weights $w$ in both hybrid-adder and hybrid-all supernets on top of the proposed *PGP* strategy, which alleviates the non-convergence issue via progressively optimizing heterogeneous layers and thus promoting the integration of adder and other operators, such as convolutions and shift layers. The loss function is formulated as:

$$\min_{\alpha} \ \mathcal{L}_{ce}^{val}(\alpha, w*) + \lambda \mathcal{L}_{hw}(\alpha), \ s.t. \ \ w* = \min_{w} \mathcal{L}_{ce}^{train}(\alpha, w), \quad (5)$$

where $\mathcal{L}_{ce}^{train}$ and $\mathcal{L}_{ce}^{val}$ denote the cross-entropy loss on the training and validation datasets, respectively. We apply FLOPs (floating-point operations per second) as a proxy hardware metric to calculate the hardware-aware loss $\mathcal{L}_{hw}$, and $\lambda$ is the coefficient applied to trade-off the task accuracy and efficiency. Note that for the shift and adder layers where the FLOP metric is not applicable, we first treat them as normal convolutional layers, and then scale the measured FLOPs down based on the computational cost of shift and adder layers normalized to that of corresponding multiplications to encourage hardware-friendly layers under the constraint of $\mathcal{L}_{hw}$.

However, the required search time and memory cost of vanilla DNAS grow linearly with the number of search candidates, challenging the exploration capability of our proposed NASA-NAS engine for the large hybrid search spaces defined in Table 2. To tackle this limitation, we apply a masking mechanism to activate and optimize only several blocks within the supernet, so that the search cost is only determined by the number of active paths and thus agnostic to the total supernet size, inspired by [3]. Specifically, we activate blocks with top-$k$ sampling probabilities:

$$Y_l = \sum_{i=1}^{n} GS(M(\alpha_l^i)) w_l^i, \ l \in (0, 22), \quad (6)$$

where $Y_l$ is the output of searchable layer $l$ with $n$ candidate blocks, each of which is equipped with weights $w_l^i$ and architectural parameters $\alpha_l^i$ that represent the corresponding sampled probability. $M(\cdot)$ and $GS(\cdot)$ donate the masking and Gumbel Softmax (GS) [9] functions, respectively, which can be formulated as follows:

$$\begin{aligned} M(\alpha_l^i) &= \begin{cases} \alpha_l^i & if \ \alpha_l^i \in top_k(\alpha_l) \\ 0 & others \end{cases}, \\ GS(\alpha_l^i|\alpha_l) &= \frac{exp(\alpha_l^i + g_l^i)/\tau}{\sum_{j=1}^{n} exp(\alpha_l^j + g_l^j)/\tau}, \end{aligned} \quad (7)$$

where $g_l^i$ is a random noise that obeys Gumbel(0, 1), and $\tau$ is a temperature parameter that controls the distribution of the GS function: $\tau$ is first set as a large number to force the GS function to approximate a uniform distribution, and then gradually decreased for guiding the GS function towards a discrete sampling function.

After identifying the best architecture via our NASA-NAS engine, we train it from scratch to obtain its task accuracy.

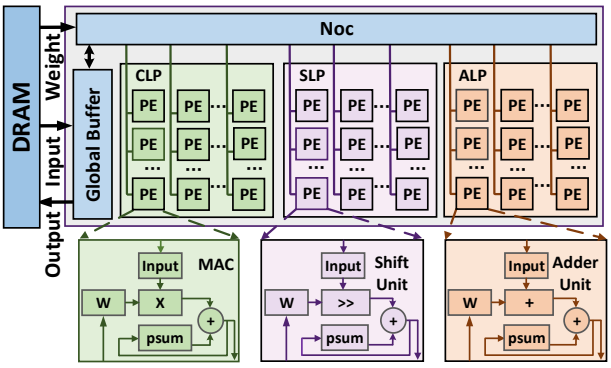


**Figure 4: Micro-architecture of our NASA-Accelerator engine, which integrates dedicated sub-processors/chunks called CLP, SLP, and ALP to process the convolutional, shift, and adder layers in NASA-NAS searched hybrid models, respectively. A four-level memory hierarchy (i.e., DRAM, global buffer, NoC, and register files within each PE) is considered to facilitate data reuses, while customized PE units (i.e., MAC, Shift Units, and Adder Units) are designed to accelerate convolutional, shift, and adder layers.**

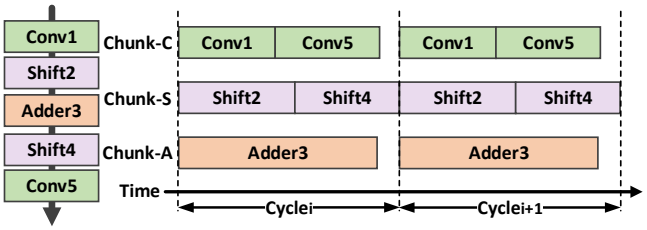


**Figure 5: Temporal processing schedule of our chunk-based accelerator across cycles, in which each sub-accelerator sequentially processes assigned layers corresponding to independent data to optimize the overall hardware utilization.**

## 4 PROPOSED NASA-ACCELERATOR ENGINE

In this section, we present our NASA-Accelerator aiming to better leverage the properties of NASA-NAS searched hybrid models for improved hardware efficiency. Specifically, Sec. 4.1 introduces NASA-Accelerator's chunk-based architecture, which integrates dedicated sub-processors for heterogeneous hybrid layers. Then, we highlight the auto-mapper in Sec. 4.2 that automatically schedules the searched hybrid models into dedicated accelerator to further improve the hardware efficiency.

### 4.1 NASA-Accelerator's Micro-architecture

**Architecture overview.** Our NASA-Accelerator adopts (1) a multi-chunk micro-architecture to facilitate designing customized processing elements (PEs) for the heterogeneous layers in NASA-NAS searched hybrid models with (2) a four-level memory hierarchy to enhance data locality. Specifically, as shown in Fig. 4, our NASA-Accelerator architecture consists of an off-chip DRAM, an on-chip global buffer, a network on chip (NoC), and three sub-processors/chunks. Specifically, the global buffer caches the computational inputs and outputs on chip to reduce costly off-chip data accesses from DRAM, the NoC serves as a bridge between the global buffer and PE array for further enhanced data reuses and can read weights directly from the off-chip DRAM, and the three sub-processors denoting convolution/shift/adder layer processors (CLP/SLP/ALP) customize their PEs to better accelerate the convolutional, shift, and adder layers in hybrid models, respectively. In particular, multiplication and accumulation (MAC) units are utilized in CLP while bit-wise shift and accumulation units (Shift Units) and addition and accumulation units (Adder Units) are designed in SLP and ALP, respectively. Additionally, while the three sub-processors share the DRAM, global buffer, and NoC, each of their PE has its own register files (RFs) for storing inputs, weights, and partial sums, respectively. Overall, our NASA-Accelerator architecture aims to (1) tailor their PE designs for assigned algorithmic layers in NASA-NAS searched hybrid DNNs and (2) maximize data locality and thus the overall acceleration efficiency.

**The PE allocation strategy.** One important question for effectively designing the aforementioned multi-chunk micro-architecture is how to partition and then allocate the limited on-accelerator resources for the sub-processors dedicated for different types of layers in NASA-NAS searched hybrid models. For better understanding our allocation strategy, we first introduce the temporal processing schedule of our accelerator, where each sub-processor sequentially processes assigned layers with independent input data to ensure concurrent processing of all sub-processors for optimizing the overall hardware utilization. Fig. 5 shows the temporal processing schedule for an NASA-NAS searched hybrid model with five layers as an illustrative example. During each cycle, sub-processors CLP, SLP, and ALP sequentially process the convolutional layers (Conv1 and Conv5), shift layers (Shift2 and Shift4), and adder layers (Adder3), respectively, each corresponds to different input data, e.g., the output of Conv1 processed by CLP in $cycle_i$ serves as input of Shift2 computed by SLP in $cycle_{i+1}$. From the above schedule and example, we can see that the achievable throughput of our accelerator is limited by the dominant latency across cycles. Hence, our PE allocation strategy strives to allocate PE resources to all three sub-processors for balancing their throughput and thus minimizing the latency of each cycle.

Formally, our PE allocation strategy above can be formulated as:

$$\begin{aligned} &N_{CLP}/O_{Conv} = N_{SLP}/O_{Shift} = N_{ALP}/O_{Adder}, \\ &s.t.\ A_{CLP} + A_{SLP} + A_{ALP} = Area\ Constraint. \end{aligned} \tag{8}$$

Simply, it ensures that the number of PEs allocated for sub-accelerators CLP ($N_{CLP}$), SLP ($N_{SLP}$), and ALP ($N_{ALP}$) are proportional to the total number of operations in the convolutional layers ($O_{Conv}$), shift layers ($O_{Shift}$), and adder layers ($O_{Adder}$), respectively, under the area budget. Note that the PEs of sub-accelerators SLP and ALP will be integrated jointly or separately with CLP to execute NASA-NAS searched hybrid models based on their layer types $T$ pre-defined in Table. 2, under the same resource budget.

Thanks to the area- and energy- efficient shift and adder layers (see Table. 1), our chunk-based accelerator dedicated for multiplication-reduced hybrid models can partially trade higher-cost MACs for convolutional layers with lower-cost Shift and/or Adder Units under the same area budget, increasing the parallelism and thus reducing the overall latency and energy cost.

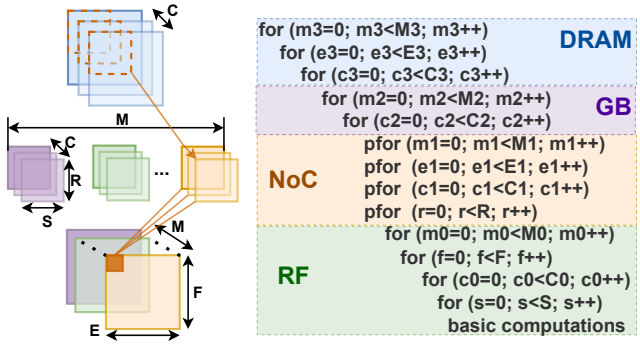


**Figure 6: An example illustrating a nested for-loop description, where a row stationary dataflow is considered in a four-level memory hierarchy including DRAM, GB, NoC, and RF, where $M, C, R, S, E, F$ donate the input/output channel dimensions, kernel width/height, and output feature map width/height, respectively.**

## 4.2 NASA-Accelerator's Auto-Mapper

It is well-known that dataflows (i.e., temporal and spatial algorithm to accelerator mapping methods) can largely impact an accelerator's hardware efficiency, while it is challenging to figure out an optimal choice which requires jointly considering both the model structure and the hardware design. Considering that the larger dataflow space resulting from more heterogeneous layers in our target hybrid models, it is nontrivial and even more challenging to manually identify the optimal dataflow for a hybrid model to be processed in our accelerator. As such, our NASA framework integrates an automated dataflow search engine called auto-mapper to automatically search for optimal dataflows of a given hybrid model when being executed in our accelerator.

In auto-mapper, we leverage the widely adopted *nested for-loop description* [4, 8, 30] that are characterized by *loop ordering factors* and *loop tiling factors* to construct the dataflow search space. Specifically, loop ordering factors correspond to how to schedule the computations in the target PE array and within each PE, and thus determine the data reuse patterns, while loop tiling factors determine how to store data within each memory hierarchy to effectively accommodate the above loop tiling factors. For better understanding the nested for-loop description, an example is shown in Fig. 6, where a layer with six dimensions is executed in an accelerator with four memory hierarchies using a row-stationary dataflow. Here the loop ordering and loop tiling factors in NoC and RF indicate that $M1 \times E1 \times C1 \times R$ PEs in the PE array are activated to process in parallel, and each of them cashes $C0 \times M0$ rows of weights to generate $M0$ *rows* of partial sums; Thus, it can be easily recognized that the data reuse pattern is row stationary and the tiling factors are constraint by the buffer size within each memory hierarchy, PE array, and dimensions of the given DNN model. Finally, the dataflow search space in our auto-mapper can be summarized as:

- **Loop ordering factors:** Determine the data reuse patterns. Here we search from four patterns: *row stationary (RS), input stationary (IS), weight stationary (WS), and output stationary (OS)* for each chunk, and thus have a total of *64 (4*4*4) choices* for reuse pattern of the three sub-accelerators in our accelerator.
- **Loop tiling factors:** Determine how to store data within each memory hierarchy (e.g., DRAM, global buffer, NoC, and RF in our accelerator) to effectively accommodate the above data reuse patterns, and can be derived from *all possible choices under the resource budget* (e.g., memory and computation resource budgets).

# 5 EXPERIMENTAL RESULTS

In this section, we first describe the experiment setup in Sec. 5.1 and the overall comparisons between our NASA framework and prior SOTA systems in Sec. 5.2, and then evaluate NASA's algorithm and hardware enablers, i.e., NASA-NAS and NASA-Accelerator, in Sec. 5.3 and Sec. 5.4, respectively.

## 5.1 Experiment Setup

**Datasets, baselines and evaluation metrics.** To verify the effectiveness of our proposed NASA framework, which integrates (1) NASA-NAS to enable effective search for multiplication-reduced hybrid models; and (2) NASA-Accelerator to boost hardware efficiency of NASA-NAS resulting models, we consider **two datasets**: CIFAR10 and CIFAR100, and **three SOTA baselines**: (1) SOTA multiplication-based NAS work FBNet [22] on SOTA multiplication-based accelerator Eyeriss [5]; (2) SOTA handcrafted multiplication-free networks, i.e., DeepShift [6] and AdderNet [20], on Eyeriss [5] with MACs in PEs replaced by the corresponding Shift and Adder Units (See Sec. 4.1); and (3) AdderNet [20] on its dedicated accelerator [21]. We compare our NASA over the above SOTA systems in terms of **two evaluation metrics**: accuracy and Energy-Delay Product (EDP), a well-known hardware metric considering both energy and latency, *under the same hardware budget for a fair comparison.*

**Search and training recipes.** We search for hardware-inspired hybrid models from the pre-defined three search spaces: *hybrid-shift*, *hybrid-adder*, and *hybrid-all*. For pretraining, we pretrain the hybrid-shift supernet for 60 epochs following the pretraining recipe as described in [22]. For hybrid-adder and hybrid-all supernets that include adder layers and therefore suffer from the non-convergence problem, we pretrain them with the proposed PGP strategy (See Sec. 3.2). Specifically, we pretrain the hybrid-adder supernet for 120 epochs, while pretrain the hybrid-all supernet for 150 epochs to accommodate the larger search space. For searching, we start from the pretrained supernet and iteratively train the model weights and architecture parameters for 90 epochs with a batch size of 128, following [22]. In each epoch, the weights are trained on 50% of the training dataset using the SGD optimizer with a momentum of 0.9. The initial learning rate (lr) is 0.05 for hybrid-shift and 0.1 for other search spaces, and will be gradually decayed following a cosine scheduler. The architecture parameters are optimized on the rest 50% of the training set by Adam optimizer [10] with a learning rate of 3e-4 and weight decay of 5e-4. The initial temperature $\tau$ of Gumbel Softmax is set to 5 and decayed by 0.956 every epoch following [22]. For training, we train the searched networks from scratch for 650 epochs with a batch size of 196, an initial lr of 0.02 with a cosine lr scheduler for hybrid-shift models, and an initial lr of 0.1 with a multi-step scheduler for hybrid-adder and hybrid-all models.

**Hardware experiment setup.** To verify the performance of NASA-Accelerator, we implement a cycle-accurate simulator on top of [30] as our chunk-based accelerator to obtain fast and reliable estimations. We verified it against the RTL implementation to ensure its correctness. The adopted unit energy and area are synthesized on CMOS 45$nm$ technology at a frequency of 250$MHz$. For fair comparisons, we also follow [30] to implement cycle-accurate simulators as benchmark accelerators: Eyeriss equipped with MACs for multiplication-based FBNet [22], equipped with Shift Units for multiplication-free DeepShift [6], and equipped with Adder Units for AdderNet [20]. We evaluate both NASA-Accelerator and the above baselines under the same hardware constraint, CMOS technology, and clock frequency. Notably, to execute our hybrid models and other baselines into corresponding hardware accelerators, we follow SOTA quantization method [2] to quantize both their weights and activations into 8 bits, except for those in shift and adder layers of our hybrid models, which are quantized into 6 bit.

## 5.2 NASA over SOTA systems

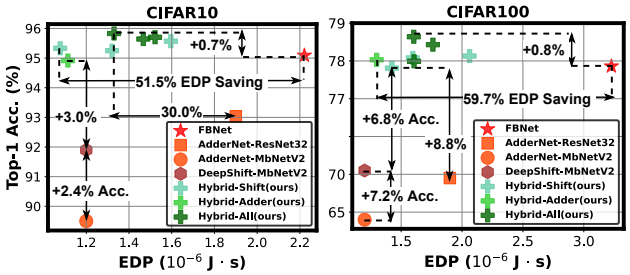


**Figure 7: Benchmarking multiplication-reduced NASA over SOTA multiplication-based and multiplication-free baselines on CIFAR10 and CIFAR100 in terms of accuracy and EDP.**

As shown in Fig. 7, the proposed NASA framework consistently surpasses all SOTA baselines, e.g., SOTA multiplication-based and multiplication-free models on Eyeriss with merely MACs and Shift/Adder Units (as introduced in Sec. 4.1) in terms of accuracy and EDP trade-off. Note that we compare under the same area and memory budget for fair. Specifically, (1) NASA achieves much higher accuracy over SOTA multiplication-free systems under comparable or even better hardware efficiency, thanks to the powerful convolutions in NASA-NAS resulting hybrid-models and the high-parallelism chunk-based design in NASA-Accelerator. In particular, NASA achieves up to **6.8%** and **14.0%** accuracy improvements over DeepShift-MobileNetV2 and AdderNet-MobileNetV2 on Eyeriss with customized multiplication-free units under the same hardware resource budget, with almost the same EDP (±**-0.1%**). Moreover, NASA offers **8.8%** higher accuracy with **25.0%** EDP saving against AdderNet-ResNet32 on [21], when testing on CIFAR100. (2) NASA also outperforms the SOTA multiplication-based system, i.e., SOTA NAS work FBNet [22] on Eyeriss [5] with MACs, by fully leveraging the algorithmic benefit of multiplication-reduced searched models and thus offering comparable or even higher accuracy with much lower EDP, e.g., we can achieve up to 0.24% and 0.20% higher accuracy with **51.5%** and **59.7%** lower EDP on CIFAR10 and CIFAR100, respectively. This set of experiments help to verify the effectiveness of NASA algorithm and hardware co-design system for boosting both the accuracy and hardware efficiency.

## 5.3 Evaluation of NASA-NAS

**Results and analysis.** We compare the NASA-NAS resulting multiplication-reduced models with SOTA handcrafted multiplication-free networks, including DeepShift-MobileNetV2 [6] and AdderNet-MobileNetV2 [20], and SOTA multiplication-based models searched by FBNet [22], on both CIFAR10 and CIFAR100. As shown in Table 3, we have three observations: (1) our hybrid models consistently outperform all baselines in terms of accuracy-efficiency trade-offs regardless of full precision (FP32) or 8-bit fixed-point (FXP8) settings, verifying NASA-NAS's effectiveness and the hypothesis that we can unify powerful convolutions with energy-efficient shift and adder layers to build hybrid models for boosting task accuracy; (2) NASA achieves on-average 0.13% and 0.34% accuracy improvements over full precision and FXP8 multiplication-based baselines, respectively, benefit from the hardware friendly operators such as shift and adder, *demonstrating that our hybrid models are powerful and robust to quantization*; and (3) as for the comparison within the three search spaces pre-defined in NASA, hybrid-all models consistently outperform other searched models by achieving a higher accuracy, showing the superiority of hybrid-all search space that includes both coarse-grained shift and fine-grained adder layers.

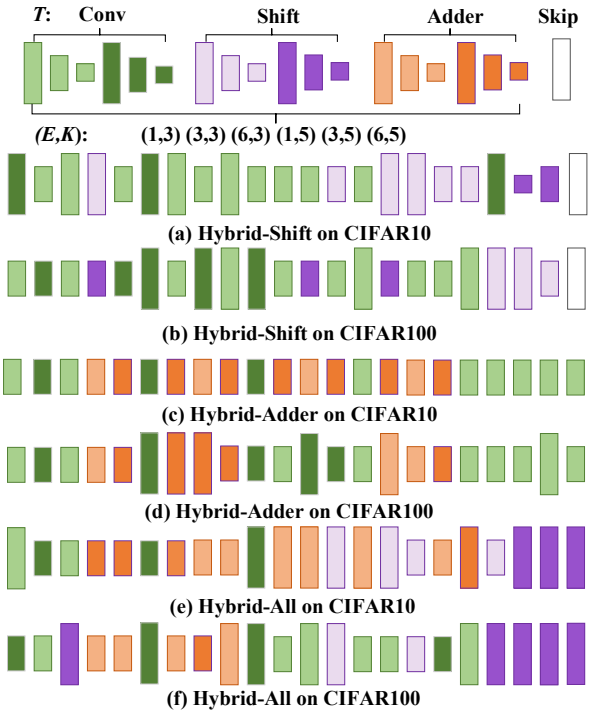


**Figure 8: The visualization of searched architectures, where $T$, $E$, and $K$ denote the layer type, expansion ratio, and kernel size of candidate blocks, respectively.**

**Visualization of searched hybrid models.** We sample and visualize the architecture of searched hybrid models in Fig. 8, from which we can see that: (1) models searched on CIFAR100 include more expensive layers, e.g., convolutions or candidate blocks with large kernel size or/and expansion ratios, than those searched on CIFAR10, which is consistent with the common sense that huge

**Table 3: The operation numbers (e.g. multiplications, bit-wise shifts, and additions) and accuracy comparisons of NASA-NAS resulting hybrid models over SOTA multiplication-free and multiplication-based models on CIFAR10 and CIFAR100.**

| Models | CIFAR10 | | | | | CIFAR100 | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Operation Numbers | | | Acc.(%) | | Operation Numbers | | | Acc.(%) | |
| | Mult. | Shift | Addition | FP32 | FXP8 | Mult. | Shift | Addition | FP32 | FXP8 |
| DeepShift-MobileNetV2[6] | 3.3M | 39.6M | 42.9M | – | 91.9(-3.2) | 3.35M | 39.6M | 42.9M | – | 71.0(-7.2) |
| AdderNet-MobileNetV2[20] | 3.3M | 0M | 82.5M | 90.5(-4.9) | 89.5(-5.6) | 3.35M | 0M | 82.5M | 64.1(-14.1) | 63.5(-14.4) |
| FBNet[22] | 47.2M | 0M | 47.2M | 95.4 | 95.1 | 56.6M | 0M | 56.6M | 78.2 | 77.9 |
| Hybrid-Shift-A | 29.6M | 21.6M | 51.2M | **95.5(+0.1)** | **95.6(+0.5)** | 34.3M | 24.1M | 58.5M | 78.2(+0.0) | **78.2(+0.3)** |
| Hybrid-Shift-B | 32.3M | 11.1M | 43.5M | **95.5(+0.1)** | **95.3(+0.2)** | 35.1M | 23.6M | 58.8M | **78.3(+0.1)** | **78.1(+0.2)** |
| Hybrid-Shift-C | 24.6M | 18.1M | 42.7M | 95.3(-0.1) | **95.3(+0.2)** | 33.9M | 14.6M | 48.6M | 78.0(-0.2) | 77.8(-0.1) |
| Hybrid-Adder-A | 35.4M | 0M | 43.8M | 95.0(-0.4) | 94.9(-0.2) | 29.9M | 0M | 60.1M | **78.4(+0.2)** | **78.1(+0.2)** |
| Hybrid-All-A | 24.1M | 23.8M | 63.5M | **95.7(+0.3)** | **95.7(+0.6)** | 27.5M | 19.9M | 64.7M | **78.4(+0.2)** | **78.0(+0.1)** |
| Hybrid-All-B | 27.7M | 17.2M | 68.5M | **95.9(+0.5)** | **95.7(+0.6)** | 25.2M | 20.5M | 64.5M | **78.7(+0.5)** | **78.7(+0.8)** |
| Hybrid-All-C | 21.0M | 20.3M | 64.6M | **95.8(+0.4)** | **95.8(+0.7)** | 28.2M | 23.9M | 68.1M | **78.3(+0.1)** | **78.5(+0.6)** |

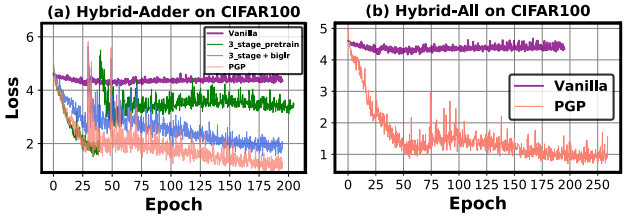


**Figure 9: Ablation study of the proposed progressive pretrain strategy on (a) hybrid-adder and (b) hybrid-all search spaces.**

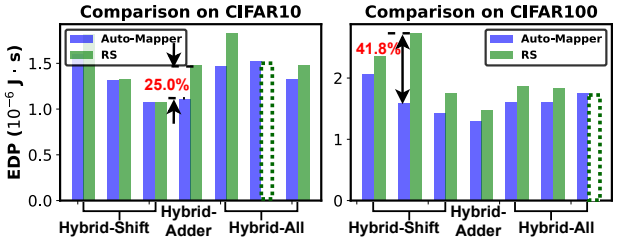


**Figure 10: Auto-Mapper over SOTA expert-crafted RS.**

networks have the better model expressiveness capacity to fit complex dataset; and (2) the layer types in searched hybrid models are consistent with these pre-defined ones in Table 2, e.g., hybrid-shift or hybrid-adder models integrate convolutions with only shift or adder layers, while hybrid-all models include both of them, further verifying the redundancy of original DNNs and the superiority of our NASA framework.

**Ablation study of PGP.** Fig. 9 (a) visualizes the training trajectories, from which we can see that: (1) the mismatch between adder layers and convolutions in terms of weight distribution and convergence speed yields the non-convergence problem when searching for hybrid-adder models, while the vanilla pretrain method in [22] can not handle this issue, calling for customized training strategy; (2) our proposed PGP accelerates the convergence during search, verifying PGP's effectiveness and our hypothesis that progressively training heterogeneous layers can promote the fusion of them; (3) the three-stage pretrain method in PGP sets a good initialization for hybrid-adder supernet via training convolutions and adder layers sequentially and then jointly, and the big lr alleviates the slow convergence problem of adder layers and thus accelerates the optimization process. Besides, the initialization that proposed to stabilize the search process in [27] also benefits the training of our hybrid supernet. We observe the same non-convergence problem when training the hybrid-all supernet, and PGP also works well as proofed in Fig. 9 (b).

## 5.4 Evaluation of the NASA-Accelerator

**Effectiveness of auto-mapper.** Fig. 10 shows the comparison between Row Stationary (RS) mapping and our proposed auto-mapper, from which we can observe that: (1) the heterogeneous dataflows suggested by auto-mapper consistently outperform the SOTA expert-crafted dataflow, e.g., RS for each chunk which is also included in our search space, verifying the auto-mapper's effectiveness; (2) in particular, auto-mapper can achieve up to **25.0%** and **41.8%** EDP saving on CIFAR10 and CIFAR100, respectively; and (3) it is worth noted that due to the competition between chunks, e.g., CLP, SLP, and ALP that are executed to process corresponding convolutions, shift, and adder layers in parallel, for hardware resources such as shared buffer, the fixed RS for all chunks fails to map hybrid models into the chunk-based accelerator under given hardware constraint in some cases (e.g., those indicated in green dotted line).

## 6 CONCLUSION

We propose, develop, and validate NASA, *the first* algorithm and hardware co-design framework dedicated for hybrid DNN models. Our NASA framework integrates a NAS and accelerator engine customized for developing and accelerating hybrid DNN models, respectively. Specifically, NASA-NAS fuses hardware friendly operators, such as bit-wise shifts and additions, into a SOTA FBNet search space, and equips a NAS algorithm with our proposed progressive pretrain strategy to identify optimal hybrid DNNs from our constructed hybrid search spaces; NASA-Accelerator advocates a chunk-based accelerator and integrates an auto-mapper to optimize dataflows for heterogeneous layers in hybrid DNNs. Extensive experimental results validate the effectiveness and advantages of our NASA framework in boosting DNN accuracy and efficiency. We believe our work can open up an exiting perspective for the exploration and deployment of multiplication-reduced hybrid models to boost both task accuracy and efficiency.

# REFERENCES


[1] Manoj Alwani, Han Chen, Michael Ferdman, and Peter Milder. 2016. Fused-layer CNN accelerators. *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)* (2016), 1–12.

[2] R. Banner, Itay Hubara, E. Hoffer, and Daniel Soudry. 2018. Scalable Methods for 8-bit Training of Neural Networks. In *NeurIPS*.

[3] Han Cai, Ligeng Zhu, and Song Han. 2019. ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. *ArXiv* abs/1812.00332 (2019).

[4] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. 2018. Eyeriss v2: A Flexible and High-Performance Accelerator for Emerging Deep Neural Networks. *arXiv preprint arXiv:1807.07928* (2018).

[5] Yu-Hsin Chen, T. Krishna, J. Emer, and V. Sze. 2017. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. *IEEE Journal of Solid-State Circuits* 52 (2017), 127–138.

[6] Mostafa Elhoushi, Farhan Shafiq, Y. Tian, Joey Li, and Zihao Chen. 2019. DeepShift: Towards Multiplication-Less Neural Networks. *ArXiv* abs/1905.13298 (2019).

[7] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), 770–778.

[8] Yu hsin Chen, Joel S. Emer, and Vivienne Sze. 2016. Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks. *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)* (2016), 367–379.

[9] Eric Jang, Shixiang Gu, and Ben Poole. 2017. Categorical Reparameterization with Gumbel-Softmax. *ArXiv* abs/1611.01144 (2017).

[10] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2015).

[11] Hanxiao Liu, K. Simonyan, and Yiming Yang. 2019. DARTS: Differentiable Architecture Search. *ArXiv* abs/1806.09055 (2019).

[12] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and J. Dean. 2018. Efficient Neural Architecture Search via Parameter Sharing. In *ICML*.

[13] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. 2019. Regularized Evolution for Image Classifier Architecture Search. In *AAAI*.

[14] Joseph Redmon and Ali Farhadi. 2017. YOLO9000: Better, Faster, Stronger. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017), 6517–6525.

[15] Joseph Redmon and Ali Farhadi. 2018. YOLOv3: An Incremental Improvement. *ArXiv* abs/1804.02767 (2018).

[16] Yongming Shen, Michael Ferdman, and Peter Milder. 2017. Maximizing CNN accelerator efficiency through resource partitioning. *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)* (2017), 535–547.

[17] Christian Szegedy, W. Liu, Y. Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, D. Erhan, V. Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), 1–9.

[18] Alvin Wan, Xiaoliang Dai, Peizhao Zhang, Zijian He, Yuandong Tian, Saining Xie, Bichen Wu, Matthew Yu, Tao Xu, Kan Chen, Péter Vajda, and Joseph Gonzalez. 2020. FBNetV2: Differentiable Neural Architecture Search for Spatial and Channel Dimensions. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2020), 12962–12971.

[19] Hanrui Wang, Zhanghao Wu, Zhijian Liu, Han Cai, Ligeng Zhu, Chuang Gan, and Song Han. 2020. HAT: Hardware-Aware Transformers for Efficient Natural Language Processing. *ArXiv* abs/2005.14187 (2020).

[20] Yunhe Wang, Mingqiang Huang, Kai Han, Hanting Chen, Wei Zhang, Chunjing Xu, and Dacheng Tao. 2020. AdderNet: Do We Really Need Multiplications in Deep Learning? *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2020), 1465–1474.

[21] Yunhe Wang, Mingqiang Huang, Kai Han, Hanting Chen, Wei Zhang, Chunjing Xu, and Dacheng Tao. 2021. AdderNet and its Minimalist Hardware Design for Energy-Efficient Artificial Intelligence. *ArXiv* abs/2101.10015 (2021).

[22] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Péter Vajda, Yangqing Jia, and Kurt Keutzer. 2019. FBNet: Hardware-Aware Efficient ConvNet Design via Differentiable Neural Architecture Search. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2019), 10726–10734.

[23] Bichen Wu, Alvin Wan, Xiangyu Yue, Peter H. Jin, Sicheng Zhao, Noah Golmant, Amir Gholaminejad, Joseph E. Gonzalez, and Kurt Keutzer. 2018. Shift: A Zero FLOP, Zero Parameter Alternative to Spatial Convolutions. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2018), 9127–9135.

[24] Pengfei Xu, Xiaofan Zhang, Cong Hao, Yang Zhao, Yongan Zhang, Yue Wang, Chaojian Li, Zetong Guan, Deming Chen, and Yingyan Lin. 2020. AutoDNNchip: An Automated DNN Chip Predictor and Builder for Both FPGAs and ASICs. In *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (Seaside, CA, USA) *(FPGA '20)*. Association for Computing Machinery, New York, NY, USA, 40–50. https://doi.org/10.1145/3373087.3375306

[25] Ping Xue and Bede Liu. 1984. Adaptive equalizer using finite-bit power-of-two quantizer. In *IEEE Trans. Acoust. Speech Signal Process.*

[26] Haoran You, Xiaohan Chen, Yong an Zhang, Chaojian Li, Sicheng Li, Zihao Liu, Zhangyang Wang, and Yingyan Lin. 2020. ShiftAddNet: A Hardware-Inspired Deep Network. *Thirty-fourth Conference on Neural Information Processing Systems (NeurIPS)* (2020).

[27] Jiahui Yu, Pengchong Jin, Hanxiao Liu, Gabriel Bender, Pieter-Jan Kindermans, Mingxing Tan, Thomas Huang, Xiaodan Song, and Quoc V. Le. 2020. BigNAS: Scaling Up Neural Architecture Search with Big Single-Stage Models. In *ECCV*.

[28] Xiaofan Zhang, Junsong Wang, Chao Zhu, Y. Lin, Jinjun Xiong, W. Hwu, and D. Chen. 2018. DNNBuilder: an Automated Tool for Building High-Performance DNN Hardware Accelerators for FPGAs. *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)* (2018), 1–8.

[29] Yang Zhao, Xiaohan Chen, Yue Wang, Chaojian Li, Haoran You, Yonggan Fu, Yuan Xie, Zhangyang Wang, and Yingyan Lin. 2020. SmartExchange: Trading Higher-cost Memory Storage/Access for Lower-cost Computation. *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)* (2020), 954–967.

[30] Yang Zhao, Chaojian Li, Yue Wang, Pengfei Xu, Yongan Zhang, and Yingyan Lin. 2020. DNN-Chip Predictor: An Analytical Performance Predictor for DNN Accelerators with Various Dataflows and Hardware Architectures. *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2020), 1593–1597.

[31] Barret Zoph and Quoc V. Le. 2017. Neural Architecture Search with Reinforcement Learning. *ArXiv* abs/1611.01578 (2017).

[32] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. 2018. Learning Transferable Architectures for Scalable Image Recognition. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2018), 8697–8710.