MAGNet: A Modular Accelerator Generator for Neural Networks

Rangharajan Venkatesan,[†] Yakun Sophia Shao,[†] Miaorong Wang,[‡] Jason Clemons,[†] Steve Dai,[†] Matthew Fojtik,[†]
Ben Keller,[†] Alicia Klinefelter,[†] Nathaniel Pinckney,[†] Priyanka Raina,[◇] Yanqing Zhang,[†] Brian Zimmer,[†]
William J. Dally,[†◇] Joel Emer,[†‡] Stephen W. Keckler,[†] Brucek Khailany[†]
NVIDIA[†] Massachusetts Institute of Technology[‡] Stanford University[◇]
Email: rangharajanv@nvidia.com

Abstract—**Deep neural networks have been adopted in a wide range of application domains, leading to high demand for inference accelerators. However, the high cost associated with ASIC hardware design makes it challenging to build custom accelerators for different targets. To lower design cost, we propose MAGNet, a modular accelerator generator for neural networks. MAGNet takes a target application consisting of one or more neural networks along with hardware constraints as input and produces synthesizable RTL for a neural network accelerator ASIC as well as valid mappings for running the target networks on the generated hardware. MAGNet consists of three key components: (i) MAGNet Designer, a highly configurable architectural template designed in C++ and synthesizable by high-level synthesis tools. MAGNet Designer supports a wide range of design-time parameters such as different data formats, diverse memory hierarchies, and dataflows. (ii) MAGNet Mapper, an automated framework for exploring different software mappings for executing a neural network on the generated hardware. (iii) MAGNet Tuner, a design space exploration framework encompassing the designer, the mapper, and a deep learning framework to enable fast design space exploration and co-optimization of architecture and application. We demonstrate the utility of MAGNet by designing an inference accelerator optimized for image classification application using three different neural networks—AlexNet, ResNet, and DriveNet. MAGNet-generated hardware is highly efficient and leverages a novel multi-level dataflow to achieve 40 fJ/op and 2.8 TOPS/mm^2 in a 16nm technology node for the ResNet-50 benchmark with $<$1% accuracy loss on the ImageNet dataset.**

I. Introduction

Deep neural networks (DNNs) have emerged as a key approach to solving complex problems across many application spaces, including image recognition, natural language processing, robotics, health care, and autonomous driving. DNN inference applications can differ significantly in their demands. For example, typical data center inference applications such as image recognition may prioritize performance and scalability at low latency and may be willing to sacrifice classification accuracy, while DNN inference for autonomous driving workloads may prioritize energy efficiency within real-time constraints while maintaining the best achievable network accuracy. Since custom inference accelerators can offer significant performance and power advantages compared to general-purpose processors and FPGAs, an ideal approach to DNN inference accelerator design would deploy custom accelerator hardware instances for each target market and range of supported workloads. Many different specialized accelerator architectures [1]–[8] have been proposed in literature. These accelerators typically employ large numbers of multiply-accumulate (MAC) units, but vary significantly in their interconnection network, memory hierarchy, and dataflow, demonstrating the large size of the DNN inference accelerator design space. However, the high engineering cost and long design cycles of ASICs make the task of choosing among these many design points to meet diverse performance and power targets a significant challenge.

This research was, in part, funded by the U.S. Government, under the DARPA CRAFT program. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government.

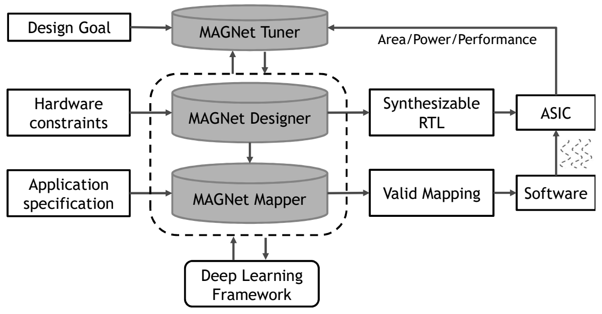


Fig. 1: MAGNet overview.

To address these challenges, we propose MAGNet, a Modular Accelerator Generator for neural Networks. As shown in Figure 1, MAGNet takes as input: 1) a target application specification consisting of a variety of neural networks; 2) hardware constraints, such as an area budget or a latency target; and 3) a design goal in terms of performance and/or energy. MAGNet produces as outputs synthesizable RTL for a DNN accelerator ASIC and a valid mapping for running DNN inference on a particular network. MAGNet consists of three components: MAGNet Designer, MAGNet Mapper, and MAGNet Tuner. The MAGNet designer consists of a highly configurable architecture template with many design-time parameters, allowing the generation of DNN accelerators specialized for specific workloads and use cases. The MAGNet mapper handles the mapping of different neural networks onto the generated hardware and enables optimization of mapping strategies at run-time. The MAGNet tuner uses Bayesian optimization, a well-known technique for searching complex spaces [9], to rapidly explore the design space and perform hardware-software co-optimization. It integrates the designer, the mapper, and a deep learning framework like Pytorch to optimize the design for different use cases. To reduce design cost, MAGNet leverages High Level Synthesis (HLS) and a generator-based design methodology [10] to model building blocks in C++ and SystemC. Dozens of different accelerator implementations can be easily prototyped through HLS, logic synthesis, and power analysis, enabling optimal design parameter selection for desired DNN accuracy, performance, and energy targets.

The main contributions of this work are as follows:

1) We propose MAGNet, a DNN inference accelerator generator structured around a highly configurable Processing Element (PE) architectural template written in C++ and synthesizable by HLS tools.
2) We use MAGNet to explore different dataflows and propose two novel multi-level dataflows within each MAGNet PE to exploit

data reuse and optimize energy efficiency in convolutional neural network layers.
3) We demonstrate an integrated framework for performing design space exploration, enabling co-design of PE hardware parameters, tiling strategies, and neural networks.

We also present an evaluation of using MAGNet and our design space exploration framework to optimize the PE in a DNN inference accelerator targeted for image classification using three different neural networks—AlexNet, ResNet-50, and DriveNet. By searching the accelerator design space and optimizing for PE design parameters and dataflow, we identify an accelerator architecture that achieves 69 fJ/op and 2.1 TOPS/mm^2 with 8-bit weight and activation precision for ResNet-50 using ImageNet dataset in a 16nm FinFET technology, a 43% reduction in energy and 2.1× improvement in performance per unit area compared to an NVDLA-like [1] design. Furthermore, by re-training the network at lower precision, we find a design that achieves 40 fJ/op and 2.8 TOPS/mm^2 using 4-bit weight and activation precision while maintaining accuracy within 1% of the full-precision baseline.

II. Related Work

A number of research efforts focused on improving the performance and energy efficiency of DNN inference accelerators. At the architecture level, early research efforts explored several accelerator designs [1]–[8] with different memory hierarchies, dataflow, and interconnection networks. These architectures exploit different reuse patterns to optimize the design for different target applications. Another class of research efforts explores trade-offs across different dataflows and flexible mappings to execute a DNN on target hardware using analytical models [11]–[13].

In contrast to previous efforts that describe individual design instances, MAGNet addresses the design automation challenges associated with specialized deep learning accelerators. MAGNet enables systematic design space exploration and co-optimization of accelerator design-time and run-time parameters, including different micro-architectural parameters, dataflows, tiling strategies, and quantization techniques. Our generator-based approach leverages a high-productivity design methodology to ensure fast and accurate design space exploration.

Earlier research efforts have also proposed the use of FPGAs and corresponding automation tools to accelerate DNNs [14]–[16]. These efforts primarily focus on accelerating a single neural network and focus on optimal FPGA resource management techniques. In addition, they support a fixed dataflow and do not consider run-time mapping strategies for executing different neural networks on the hardware. MAGNet targets ASIC hardware accelerators and supports configuration of various design-time and run-time parameters including hardware parameters, dataflows, and mappings to optimize the design across a wide range of neural networks.

III. MAGNet Overview

Figure 1 shows the automated MAGNet design flow. Internally, MAGNet consists of three components: the MAGNet designer that generates synthesizable RTL for the target architecture, the MAGNet mapper that produces valid mappings for a given network running on the target architecture, and the MAGNet tuner that enables efficient design space exploration.

The MAGNet designer, discussed in Section IV, is based on a tiled architecture with an array of interconnected PEs each containing local weight and activation memories as shown in Figure 2(a). The PEs perform the bulk of the computation for the convolutional and

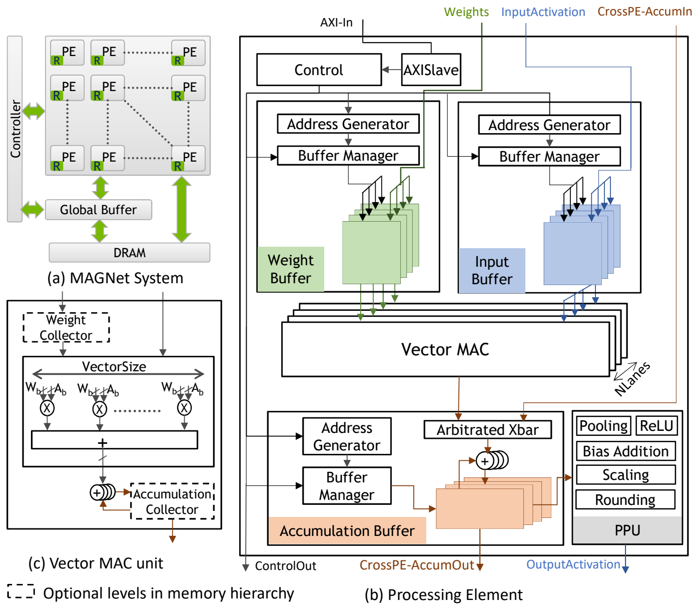


Fig. 2: MAGNet architecture template.

fully-connected layers found in typical DNNs. Highly configurable design-time parameters enable optimization and specialization.

The MAGNet mapper, discussed in Section V, maps DNN inference workloads onto the array of PEs at run time. The mapper utilizes tiling strategies for distributing work amongst an array of PEs in order to optimize for utilization and energy efficiency.

Finally, the MAGNet tuner, discussed in Section VI, integrates the MAGNet designer, the mapper, and neural network design tools like Caffe and PyTorch in a unified framework, enabling co-optimization across hardware parameters, tiling strategies, and the neural network.

IV. The MAGNet Designer

The foundation of the MAGNet designer is a highly configurable DNN accelerator template that leverages objected-oriented HLS to generate different types of accelerators based on user requirements. Figure 2 shows an overview of the MAGNet architecture template. It consists of a three-tiered compute hierarchy: 1) vector MAC, 2) processing element (PE), and 3) system. Table I illustrates the design-time parameters at each level. Each level in the hierarchy features configurable memory arrays for data staging and reuse. MAGNet also presents a range of micro-architectural parameters ($NPEs$ at the system level, $NLanes$ at the PE level, and $VectorSize$ at the vector MAC level) to scale the compute capabilities at each level. In addition, MAGNet allows configurability of dataflows, precision, and the interconnect network to suit the design requirements of different target applications. The following subsections present a detailed description of the MAGNet architecture template and novel dataflows proposed in this work.

A. System Architecture

Figure 2(a) shows the system architecture consisting of an array of PEs and a global buffer (GB). The PEs are the workhorse of the accelerator, performing the most compute-intensive tasks. The GB offers high-bandwidth memory to feed data into the array of PEs. At the system level, MAGNet supports a configurable number of PEs and flexible GB memory capacity and bandwidth to enable performance scalability depending on the application. It also supports a flexible interconnection topology to connect the different components. Smaller

TABLE I: MAGNet design-time parameters.

Hierarchy	Parameter	Name
System	# of PEs	$NPEs$
	Network Topology	NetTopology
	Global Buffer Size	$GBSize$
PE	Dataflow	Df
	Weight Buffer Size	$WSize$
	Input Activation Buffer Size	$IASize$
	Accumulation Buffer Size	$AccumSize$
	# of Parallel Lanes	$NLanes$
VectorMAC	Vector Size	$VectorSize$
	Weight Collector Depth	$WColDepth$
	Accumulation Collector Depth	$AccumColDepth$
	Weight Precision	$WPrecision$
	Input Activation Precision	$IAPrecision$
	Accumulation Precision	$AccumPrecision$

mobile inference accelerators that consist of a few PEs can use direct point-to-point and broadcast links to deliver data to the PEs. For large-scale inference accelerator designs, the PEs can be connected together using a mesh Network-on-Chip (NoC) configuration. To implement this NoC, MAGNet employs a programmable, LUT-based wormhole router with configurable flit widths and virtual channels.

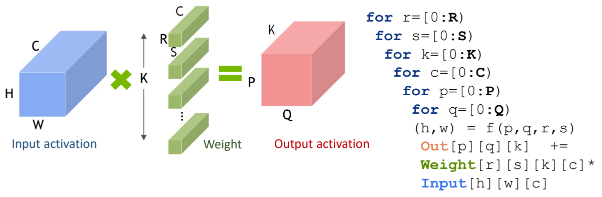
B. Processing Element

Figure 2(b) shows the micro-architecture of the PE. It contains vector MAC units, a weight buffer, an input activation buffer, an accumulation buffer, a post-processing unit (PPU), and control.

Each PE contains $NLanes$ **Vector MAC units** that perform matrix-vector product operations every cycle. A weight matrix of dimensions $NLanes \times VectorSize$ is multiplied with an input activation (IA) vector of size $VectorSize$ to produce a partial-sum vector of size $NLanes$. These operations are supported at various precisions with $WPrecision$-bit weights, $IAPrecision$-bit input activations, and $AccumPrecision$-bit partial sum outputs. Each of the Vector MAC units contains a $VectorSize$-wide dot-product calculation, weight collectors with depth $WColDepth$ and width $WPrecision \times VectorSize$, and accumulation collectors with depth $AccumColDepth$ and width $AccumPrecision$. Optional *collectors* are latch arrays that provide an additional level of memory hierarchy to reduce traffic between the vector MAC units and local SRAM buffers. As $VectorSize$ and $NLanes$ are adjusted for more parallelism and reuse, MAGNet automatically adjusts the parameters of local buffers within the PE to keep the vector MAC units fed. Increasing $VectorSize$ also increases the logic depth within the vector MAC unit, resulting in the automatic insertion of additional pipeline stages by the HLS tool depending on the target clock frequency.

Weight and IA buffers supply weight and IA data to the vector MACs. The weight buffer read port is $WPrecision \times NLanes \times VectorSize$ bits wide so it can supply distinct weight vectors to different vector MAC units each cycle. The IA buffer is $IAPrecision \times VectorSize$ bits wide, as an IA vector is shared across $NLanes$. Each buffer has an address generator that produces an address every cycle, as well as a buffer manager [17] that controls the availability of data. The address generation pattern and the granularity of data movement can be configured at run time.

Accumulation buffers are used to store partial sums across $NLanes$ vector MACs and are optimized to perform read-modify-write operations every cycle. Partial sums from the vector MAC units are packed into vectors of width $AccumPrecision \times NLanes$ to be stored in the accumulation buffer. From there, they can be sent either directly to another PE for cross-PE reduction or to the post-processing unit (PPU) to produce final output activations. The PPU supports not only scaling and quantization but also ReLU and pooling operations to allow layer fusion.



```
for r=[0:R)
 for s=[0:S)
  for k=[0:K)
   for c=[0:C)
    for p=[0:P)
     for q=[0:Q)
      (h,w) = f(p,q,r,s)
       Out[p][q][k]  +=
       Weight[r][s][k][c]*
       Input[h][w][c]
```

Fig. 3: Loop nest representation of a convolution layer.

C. Dataflow

Figure 3 shows an example of a convolutional layer. We use `H`, `W` to represent the input activation size, `R`, `S` for the weight kernel size, and `C` and `K` to represent the number of input and output channels, respectively. The computations in a convolutional layer can be represented by a loop nest as shown on the right of Figure 3.

Dataflow controls the temporal reuse of data across cycles. Prior work has demonstrated that the choice of dataflow has a significant impact on the energy efficiency and performance of an accelerator [18]. Therefore, MAGNet allows design-time configurability to choose different dataflows. In addition to previously published dataflows, i.e, weight stationary (WS), output stationary (OS), and input stationary (IS) [18], MAGNet features new multi-level weight-output stationary dataflows: Weight Stationary - Local Output Stationary (WS-LOS) and Output Stationary - Local Weight Stationary (OS-LWS). Unlike prior work that optimizes a single operand, MAGNet's multi-level dataflows provide the flexibility to capture reuse in both weights and outputs with minimal area overhead. This advantage is quantified in Section VII.

Figure 4 illustrates different dataflows in loop nest form. The loop bounds for each dataflow are determined by the MAGNet mapper. Figure 4(a) shows a weight-stationary (WS) dataflow, in which the outer loops use the weight dimensions $(R, S, K1, C1)$ to reuse weights across vector MAC operations with different input vectors. The loop nest is given for a single PE, so the dimensions $K1, C1$ are used instead of K, C to signify that each PE handles a portion of the entire range. To generate a DNN accelerator with a WS dataflow using MAGNet, we choose a single-entry weight collector and omit the accumulation collector in the vector MAC unit. The ordering of vector MAC operations is achieved by configuring the address generator logic to produce the correct sequence of addresses to the local buffers. Similarly, an OS (IS) dataflow can be realized by using a single-entry output (input) collector.

Figure 5 shows the energy breakdown across different components of the PE for WS and OS dataflows for all AlexNet layers summed together with $VectorSize = 8$, $NLanes = 8$, 8-bit precision for weights and input activations, and 24-bit precision for partial sums, evaluated using the setup described in Table IV. In WS dataflow, which is optimized for the temporal reuse of weights, the weight buffer contributes only a small fraction to energy consumption while the accumulation buffer contributes significantly. An OS dataflow, on the other hand, optimizes the accumulation buffer energy at the cost of increased weight buffer energy. In each case, the relative contribution

```
for k1=[0:K1):
 for r=[0:R):
  for s=[0:S):
   for c1=[0:C1):
    // WS
    for p1=[0:P1):
     for q1=[0:Q1):


        Vector MACs
```

(a) WS dataflow

```
for k1=[0:K1):
 for p1=[0:P1):
  for q1=[0:Q1):
   // OS
   for r=[0:R):
    for s=[0:S):
     for c1=[0:C1):


        Vector MACs
```

(b) OS dataflow

```
for h1=[0:H1):
 for w1=[0:W1):
  for c1=[0:C1):
   // IS
   for k1=[0:K1):
    for r=[0:R):
     for s=[0:S):


        Vector MACs
```

(c) IS dataflow

```
for k1=[0:K1):
 for r=[0:R):
  for s=[0:S):
   for c1=[0:C1):
    // WS
    for p1=[0:P1):
     for q1=[0:Q1):
      // LOS
      for c0=[0:C0):
        Vector MACs
```

(d) WS-LOS dataflow

```
for k1=[0:K1):
 for p1=[0:P1):
  for q1=[0:Q1):
   // OS
   for r=[0:R):
    for s=[0:S):
     for c1=[0:C1):
      // LWS
      for q0=[0:Q0):
        Vector MACs
```

(e) OS-LWS dataflow

Fig. 4: Loop nests for different dataflows supported by MAGNet.

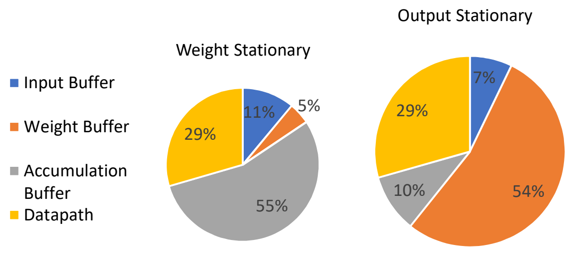


Fig. 5: Energy breakdown across different components of PE for weight and output stationary dataflows. Size of the pie chart is proportional to the total energy consumption.

TABLE II: Temporal data reuse across different dataflows.

Dataflow	Weight Reuse	Input Reuse	Output Reuse
WS	P1×Q1	0	0
OS	0	0	R×S×C1
IS	0	R×S×K1	0
WS-LOS	P1×Q1	0	C0
OS-LWS	Q0	0	R×S×C1

from the input buffer is small due to the narrow read/write bitwidth and spatial reuse across different lanes of the PE, so the energy savings from input stationary dataflow (not shown) are small. To address the challenge of optimizing the weight and accumulation buffer energy simultaneously, we propose multi-level dataflows.

Multi-level Dataflows: Multi-level dataflows are designed to optimize for reuse in both weights and outputs by using dedicated weight and accumulation collectors. Figures 4(d) and 4(e) show the loop nests for the multi-level WS-OS dataflows proposed in this work. The multi-level dataflows include an additional level in the loop nest compared to conventional dataflows. The WS-LOS dataflow reuses outputs in the innermost loop in a single-entry accumulation collector, while reusing weights in outer loops via a multi-entry weight collector. As illustrated in Figure 4(d), outputs are reused in the `C0` loop while weights are reused outside the `P1` loop. The OS-LWS dataflow prioritizes weight reuse in the innermost loop while capturing the output reuse in outer loops as shown in Figure 4(e). A MAGNet-generated accelerator with an OS-LWS dataflow provisions a single-entry weight collector and a multiple-entry accumulation collector. In this dataflow, a weight stored in the weight collector is reused `Q0` times to compute `Q0` different partial sums, which are stored in the accumulation collector. Next, each of these `Q0` partial sums values are reused `R×S×C1` times before they are written to the accumulation buffer.

TABLE III: MAGNet run-time parameters.

Parameter	Name
PE Tile size	P1, Q1, R, S, K1, C1, C0, Q0
Spatial mapping	PE-id
Temporal ordering	Address Gen dimensions
Layer fusion	$IsRelu$, $IsPool$

Table II presents a qualitative comparison of the data reuse for different dataflows at the PE level. Multi-level dataflows exploit additional reuse dimensions to achieve reuse in both weights and outputs. The amount of reuse is determined by the bounds of the loop-nest and can be controlled at run time by the MAGNet mapper.

V. The MAGNet Mapper

The MAGNet mapper determines how a neural network will be executed on the target hardware architecture. We use the term *mapping* to describe how a neural network layer is executed in hardware. A mapping describes several key aspects of the execution and controls the runtime-configurable parameters listed in Table III.

- Tile size: A mapping sets the PE tile size by specifying a valid setting for the bounds of the loop nest shown in Figure 4. This parameter determines the amount of reuse across different data types at the PE level.
- Spatial mapping: This parameter specifies how the computation is spatially distributed across the array of PEs. It determines the number of PEs used as well as the amount of computation performed in each PE. It also specifies if the output of a PE is sent to another PE for cross-PE reduction or the global buffer after post-processing.
- Temporal ordering: A mapping defines the temporal execution order for MAC operations in each PE. In MAGNet-generated hardware, temporal ordering is specified by configuring the address generators to produce the required sequence of addresses to the local buffers.
- Layer fusion: MAGNet also allows flexibility to fuse convolution layers with ReLU and pooling layers by controlling the configuration registers in the PPU.

Figure 6 shows the effects on performance and energy of different mapping choices for two different ResNet-50 layers executed on a single MAGNet design instance. Given a neural network layer and an architecture specification, a large number of valid mappings for executing a layer on the hardware exists, and different mappings can result in a 10× difference in performance and energy efficiency.

To explore different mapping strategies, MAGNet has an automated framework similar to TimeLoop [19] to help designers navigate the

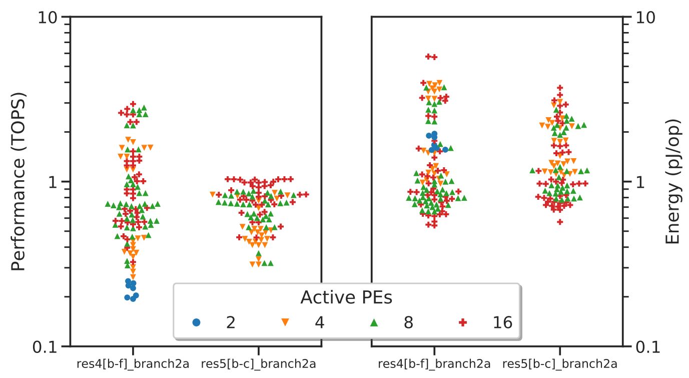


Fig. 6: Effect of mappings on performance and energy.

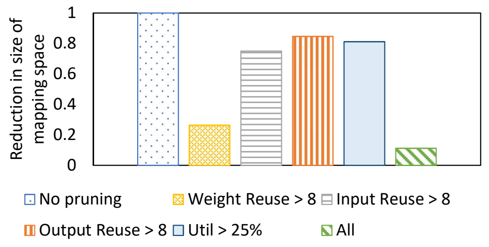


Fig. 7: Pruning heuristics and mapping space size.

mapping space and optimize the mapping for different application and use cases. The tool takes the dimensions of neural network layers and the hardware specifications as inputs and generates a list of valid mappings describing how those layers execute on the target architecture. The remainder of this section describes the MAGNet mapping space and how the MAGNet mapper prunes the mapping space with user-specified constraints.

A. Mapping Space

A mapping space is the set of valid mappings of a neural network layer onto an architecture. A valid mapping must satisfy three requirements. First, it must be compatible with the hardware dataflow. For example, a mapping for a weight-stationary dataflow would not run correctly on output-stationary hardware. Second, the mapping needs to match available compute resources. For example, a mapping requiring eight PEs would not run if only four PEs are in the system. Finally, it should satisfy the memory constraints at different levels in the hierarchy. The size of the buffers chosen at design-time places an upper bound on the tile size that can executed on the hardware.

Given an architecture specification and network layer dimensions, the MAGNet mapper generates valid mappings by enumerating all possible factorizations of the layer dimensions and compatible loop orders that satisfy the above constraints. The size of the mapping space can be extremely large, so exhaustively sweeping all possible mappings is often impractical. To address this challenge, we explore the mapping space pruning heuristics described below.

B. Mapping Space Pruning

The MAGNet mapper prunes the space by applying two user-defined constraints to prioritize either energy efficiency or performance. The first constraint is the *reuse factor*, which determines the minimum amount of temporal reuse for different data types. The reuse factor offers a knob to prune small tile sizes, which have poor energy efficiency due to low memory reuse. The second constraint is *utilization*, which determines the number of PEs utilized in the system. A higher reuse factor implies larger PE tile size and typically leads to better energy efficiency within a PE, but overall performance may be reduced if few PEs are utilized. A higher utilization tends to improve the system performance but may not be energy-optimal. Figure 7 shows the reduction in size of the mapping space with different pruning heuristics for ResNet-50 mapped to a MAGNet-generated hardware with a WS dataflow. As shown in the figure, applying a small reuse factor and utilization constraints leads to nearly 90% reduction in size of the mapping space. The pruning

Algorithm 1: Design Space Exploration in MAGNet Tuner

```
Input  : DesignGoal, DLNetworks, DesignSpace, HardwareConstraints,
         MappingConstraints, Target accuracy
Output: Optimal MAGNet design parameters and mapping
1  Initialize Bayesian Model
2  while DesignGoal not met do
3      DesignParam ← Get DesignParam from Bayesian Model
4      design ← Designer(DesignParam)
5      foreach network ∈ DLNetworks do
6          (accuracy, weights) ← DLFramework(DesignParam, network)
7          if accuracy > Target accuracy then
8              foreach layer ∈ network do
9                  MapSpace ← Mapper(design, layer,
                    MappingConstraints)
10                 foreach mapping ∈ MapSpace do
11                     Evaluate design goal
12                     Update Mapping[design][network][layer]
13                 end
14             end
15         end
16     end
17     Update Optimal Design Parameters and Optimal Mapping
18     Update BayesianModel(design, Mapping)
19 end
```

constraints are used only to prune easily-identified less-optimal points in the design space.

VI. The MAGNet Tuner

The combination of different configuration parameters in the MAGNet Designer results in a many-dimensional design space that is difficult to search manually. To enable effective and efficient co-design from architectures to networks, we propose the MAGNet tuner framework, which automatically explores the design space engendered by MAGNet and quickly identifies the most desirable

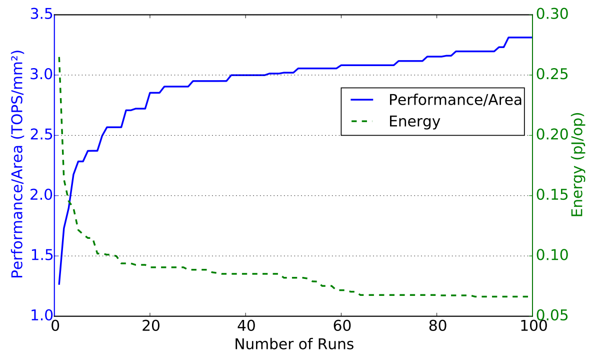


Fig. 8: Design quality achieved by iterating the MAGNet tuner. Results were averaged over 10 seeds.

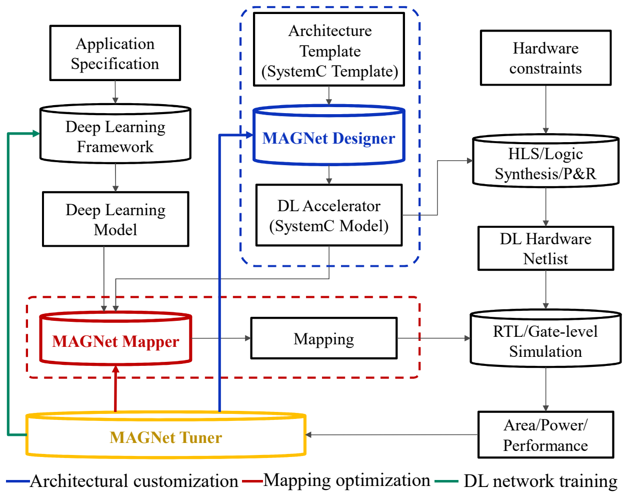


Fig. 9: Design space exploration using MAGNet.

design points within the large search space. The MAGNet tuner is built based on a Bayesian optimization strategy, a well-known technique for optimizing black-box functions within complex search spaces [9].

Algorithm 1 details the design space exploration process within the MAGNet tuner. The tuner takes as inputs the target accuracy and hardware design goal (performance or energy efficiency), along with a user-defined design space for the MAGNet designer, hardware constraints such as target clock frequency, and mapping constraints for the MAGNet mapper to prune the mapping space. Given these inputs, the tuner leverages Bayesian optimization to iteratively search through the design space until the design goal is met. With Bayesian optimization, we leverage a probabilistic model to approximate the objective (performance or energy efficiency) as a function of the various design parameters based on previously synthesized and evaluated parameter configurations. In each iteration, a new parameter configuration is selected using the probabilistic model to maximize the expected improvement of the objective over the best achieved thus far. In addition to model-based selection, our algorithm also selects a fraction of new configurations at random to encourage exploration of a larger design space instead of exploiting only the subspace emphasized by the probabilistic model. This prevents the optimization process from losing sight of the global objective function and falling into a local optimum.

The optimization is constructed based on an existing Bayesian optimization framework [20]. The implementation consists of a configuration manager to constrain the legal space of parameters, a worker to invoke the evaluation of each design point, as well as an optimizer to model design quality over the parameter space and select the next optimal candidate configuration to evaluate. Figure 8 demonstrates the performance of the MAGNet tuner in end-to-end optimization for two different design goals, performance and energy efficiency. Each curve shows the respective design quality achieved after a specific number of runs. The MAGNet tuner achieves significant improvement in design quality and can find a desirable energy/performance point with a small number of runs.

Figure 9 shows how MAGNet, along with deep learning tools like Caffe and hardware design flows, enables exploration of the large DNN accelerator design space. From a given target application specification, we use deep learning tools to explore different weight and activation precisions and the MAGNet designer to generate different

TABLE IV: Experimental Setup

Benchmarks	
Networks	AlexNet, ResNet-50, DriveNet
Dataset	ImageNet
Design Tools	
HLS Compiler	Mentor Graphics Catapult HLS
Verilog simulator	Synopsys VCS
Logic synthesis	Synopsys Design Compiler Graphical
Place and Route	Synopsys ICC2
Power Analysis	Synopsys PT-PX
Design Space	
Dataflows	WS, OS, WS-LOS, OS-LWS
VectorSize/NLanes	4, 8, 16
Weight/Activation precision	4 bits, 8 bits
Accumulation precision	16 bits, 20 bits, 24 bits
Weight Collector Size	8B - 2KB
Accumulation Collector Size	8B - 384B
Input Buffer Size	2KB, 8KB, 16KB
Weight Buffer Size	4KB - 128KB
Accumulation Buffer Size	1KB - 6KB
Global Buffer Size	64KB
Target Frequencies	500 MHz, 1 GHz
Supply Voltage	0.6V

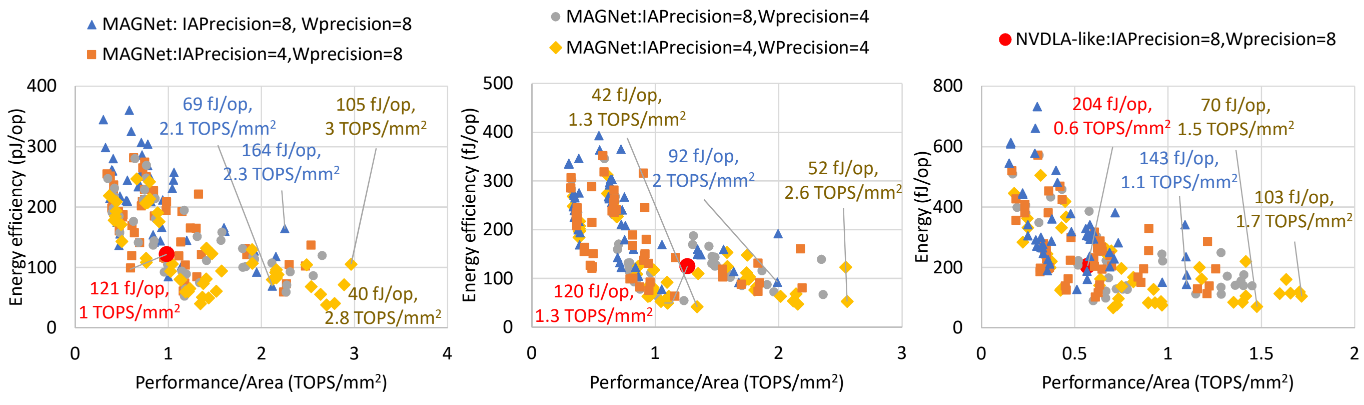
architectures with varying sizes, dataflows, and memory hierarchies. The MAGNet mapper then takes the accelerator architecture and neural network model as input and explores different mapping strategies. To reduce design effort, ease configurability, and enable fast and accurate design space exploration, we use an object-oriented high-level synthesis based methodology [10] for hardware generation. Architectural models are coded using synthesizable SystemC and C++ to autogenerate RTL using HLS tools. A standard logic synthesis flow generates gate-level netlists and area estimates, and standard power analysis tools use gate-level simulation results to estimate power dissipation. The MAGNet tuner takes these results to predict the next set of parameters for design space exploration.

VII. Evaluation

To evaluate MAGNet, we consider the case study of optimizing a DNN inference accelerator for image classification using three different neural networks—AlexNet [21], ResNet-50 [22], and DriveNet [23] using the ImageNet dataset. We study the energy and performance tradeoffs for each of its convolutional layers and evaluate a number of MAGNet-generated inference accelerator designs in a 16nm FinFET technology node for these neural networks. Table IV shows the design tools and parameters used in the evaluation.

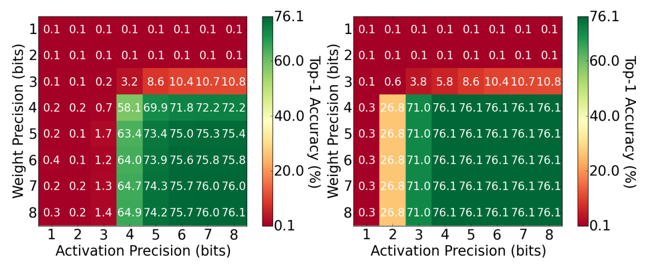
A. *Experimental Results*

Figure 10 presents the results of design space exploration. Results are shown as a tradeoff between energy efficiency (y-axis) and performance per unit area (x-axis); the lower right part of the graph is optimal. Four categories of MAGNet designs are shown, each corresponding to a particular pairing of weight and activation precisions. Each point on the graph reports metrics for a synthesized design instance selected from the set of parameter options in Table IV. For each category, we optimize the design across three different neural networks and prune the mapping space aggresively to identify an efficient mapping. Performance and energy results are averaged over convolutional layers of the networks, weighted by the number of operations in each layer. As a baseline for comparison, we also show results for a MAGNet instance configured to be similar



(a) ResNet (b) AlexNet (c) DriveNet

Fig. 10: Energy and performance of MAGNet-generated accelerators for different neural networks.



(a) Without Retraining (b) With Retraining

Fig. 11: Effect of precision scaling on the accuracy of ResNet-50 benchmark with ImageNet dataset.

to NVDLA [1], an open-source DNN accelerator that is used in commercial SoCs. NVDLA uses WS dataflow with 8-bit weight and activation precisions. Two different frequency targets are included in this work's parameter sweeps.

Figure 10 also highlights the best energy-efficiency and performance achieved for different networks. Using 8-bit precision for weights and activations, MAGNet's most energy-efficient design instance achieves 69 fJ/op (where a MAC is two ops) with 2.1 TOPS/mm^2 for the ResNet-50 benchmark, thereby achieving 43% reduction in energy and 2.1$\times$ improvement in performance per unit area compared to the NVDLA-like baseline. For AlexNet and DriveNet, the same design achieves a 24% and 30% reduction in energy. MAGNet's performance-efficient design achieves 2.3 TOPS/mm^2 for ResNet-50 at 8-bit precision, which is 2.3$\times$ higher than that of the NVDLA-like design.

Lower weight and input activation precision produces more efficient designs at the expense of reduced accuracy. By co-optimizing the application along with hardware, a MAGNet-generated instance achieves 40 fJ/Op and 2.8 TOPS/mm^2 using 4-bit weights and activations for ResNet-50. Without network re-training, this design would suffer an 18% loss in accuracy as shown in Figure 11(a). By re-training [24] the neural network, MAGNet achieves significant improvements in energy and performance with similar accuracy to that of an 8-bit implementation (Figure 11(b)). Compared to a full-precision floating point implementation, which achieves 76.5% top-1 accuracy for ResNet-50 on the ImageNet dataset, MAGNet-generated hardware has a small ($<1\%$) loss in accuracy.

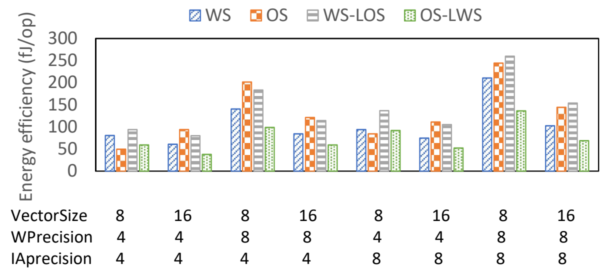


Fig. 12: Effect of weight precision, activation precision, vector size, and dataflow on energy efficiency.

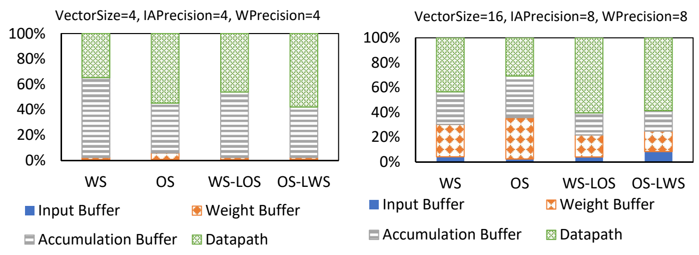


Fig. 13: Energy breakdown of the MAGNet PE.

B. Analysis

Figure 12 plots the effect of varying dataflow, vector size, and precision on the energy efficiency for ResNet-50.

Optimal Dataflow: We highlight three insights related to choice of dataflow from the results shown in Figure 12. The first insight is that the OS-LWS dataflow achieves best energy efficiency in nearly every case. The advantage is most apparent with 8-bit precision and large $VectorSize$. In these configurations, the weight buffer and the accumulation buffer contribute a significant fraction to the total energy consumption, as shown in Figure 13. Reducing accesses to these memories allows OS-LWS to achieve lower energy than the OS and WS dataflows.

A second insight is the surprising increase in energy consumption when comparing the WS-LOS dataflow to the WS dataflow. This effect can be understood by considering the mapping chosen by the MAGNet mapper. The mapper used the `C` and `K` dimensions of the convolutional layer loop-nest to map to $VectorSize$ and $NLanes$ of the PE. WS-LOS also uses the `C` dimension for temporal reuse

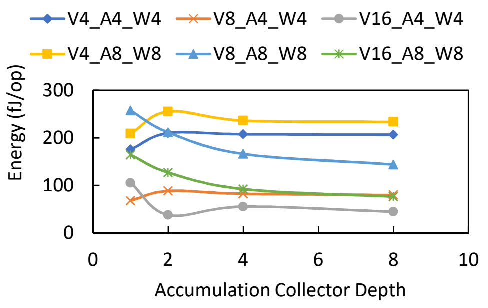


Fig. 14: Sensitivity to accumulation collector depth.

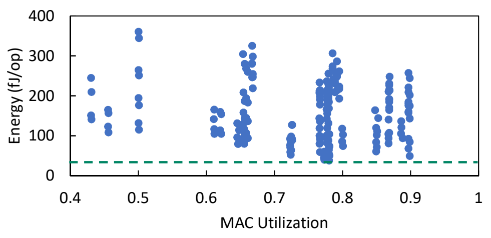


Fig. 15: Energy vs. Utilization for ResNet-50.

in the collectors. As a result, for Resnet-50 layers with smaller C dimensions, WS-LOS achieves little reuse but incurs an energy penalty from additional collectors. In contrast, OS-LWS exploits temporal reuse along the Q dimension in the collectors, thereby achieving better reuse and energy reduction. Additionally, one of the inputs of the Vector MAC units remains constant in the OS-LWS dataflow, leading to lower activity and better energy efficiency.

A third insight is that for designs with 4-bit precision and smaller vector size, the OS dataflow achieves slightly better energy efficiency than OS-LWS, because a larger fraction of the energy is used by the accumulation buffers.

Vector size: For all the dataflows except the OS dataflow, increasing the vector size leads to a reduction in energy consumption due to improved spatial reuse of partial sums. For the OS dataflow, the contribution of accumulation buffer energy is small and the incremental benefits of improved spatial reuse from a larger vector size is minimal, while other overheads from a larger vector size such as wider input memories and registers increase the total energy.

Accumulation collector depth: Figure 14 shows the effect of increasing the accumulation collector depth for different configurations in the OS-LWS dataflow. In the figure, Vx_Ay_Wz represents a design with $VectorSize = x$, $IAPrecision = y$, and $WPrecision = z$. The accumulator collector size controls the amount of weight reuse that can be exploited in the design. A larger accumulation collector enables more weight reuse and therefore minimizes the energy consumption of the weight buffer. However, it also increases the energy consumed in the accumulation collector registers. For configurations with smaller weight precision and lower spatial reuse of partial sums, because the weight buffer contributes a small fraction to total energy, increasing accumulation collector size leads to an increase in energy consumption. At higher weight precisions and larger vector sizes, increasing the accumulation collector size leads to a decrease in energy consumption due to better temporal reuse.

MAC Utilization: Figure 15 shows energy vs. MAC utilization for different MAGNet-generated designs. As shown in the figure, MAGNet designs can achieve nearly 90% MAC utilization, resulting in highly-performant designs for ResNet-50. Further, for a given utilization, a wide range of designs exists that offer different levels of energy efficiency, demonstrating the need for systematic design space exploration. Designs with very low utilization ($<50\%$) are sub-optimal for both performance and energy, so utilization offers a useful heuristic for pruning the design space.

VIII. Conclusion

This paper presents MAGNet, a DNN inference accelerator generator. We describe a highly configurable architectural template with configurable dataflows and a mapper for mapping neural networks onto a hardware architecture. We propose novel dataflows that exploit reuse across both weights and partial sums. We demonstrate the utility of MAGNet for optimizing energy efficiency and performance by generating actual hardware implementations, analyzing results on post-synthesis results, and measuring power on real workloads. By co-optimizing architecture, dataflow, and neural network, we show that a MAGNet-generated accelerator can achieve 40 fJ/op and 2.8 TOPS/mm^2 in a 16nm FinFET technology, outperforming state-of-the-art DNN inference accelerators.

References

[1] F. Sijstermans, "The NVIDIA deep learning accelerator," in *Hot Chips*, 2018.

[2] B. Zimmer *et al.*, "A 0.11 pJ/Op, 0.32-128 TOPS, Scalable Multi-Chip-Module-based Deep Neural Network Accelerator with Ground-Reference Signaling in 16nm," in *Proc. International Symp. on VLSI Circuits*, 2019.

[3] T. Chen *et al.*, "DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," in *Proc. ASPLOS*, 2014.

[4] S. Han *et al.*, "EIE: Efficient inference engine on compressed deep neural network," in *Proc. ISCA*, 2016.

[5] S. Venkataramani *et al.*, "ScaleDeep: A scalable compute architecture for learning and evaluating deep networks," in *Proc. ISCA*, 2017.

[6] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proc. ISCA*, 2017.

[7] A. Parashar *et al.*, "SCNN: An accelerator for compressed-sparse convolutional neural networks," in *Proc. ISCA*, 2017.

[8] Y. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *Proc. ISCA*, 2016.

[9] J. Mockus, *On bayesian methods for seeking the extremum and their application*. IFIP Congress, 1977.

[10] B. Khailany *et al.*, "A modular digital VLSI flow for high-productivity SoC design," in *Proc. DAC*, 2018.

[11] W. Lu *et al.*, "FlexFlow: A flexible dataflow accelerator architecture for convolutional neural networks," in *Proc. HPCA*, 2017.

[12] M. Alwani *et al.*, "Fused-layer CNN accelerators," in *Proc. MICRO*, 2016.

[13] B. Reagen *et al.*, "A case for efficient accelerator design space exploration via bayesian optimization," in *Proc. ISLPED*, 2017.

[14] H. Sharma *et al.*, "From high-level deep neural models to FPGAs," in *Proc. MICRO*, 2016.

[15] X. Zhang *et al.*, "DNNBuilder: An automated tool for building high-performance DNN hardware accelerators for FPGAs," in *Proc. ICCAD*, 2018.

[16] C. Zhang *et al.*, "Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks," in *Proc. FPGA*, 2015.

[17] M. Pellauer *et al.*, "Buffets: An efficient and composable storage idiom for explicit decoupled data orchestration," in *Proc. ASPLOS*, 2019.

[18] V. Sze *et al.*, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," *Proceedings of the IEEE*, Dec. 2017.

[19] A. Parashar *et al.*, "Timeloop: A systematic approach to dnn accelerator evaluation," in *Proc. ISPASS*, 2019.

[20] S. Falkner *et al.*, "BOHB: Robust and efficient hyperparameter optimization at scale," *CoRR*, vol. abs/1807.01774, 2018.

[21] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. NeuRIPS*, 2012.

[22] K. He *et al.*, "Deep residual learning for image recognition," in *Proc. CVPR*, 2016.

[23] M. Bojarski *et al.*, "Explaining how a deep neural network trained with end-to-end learning steers a car," *CoRR*, vol. abs/1704.07911, 2017.

[24] H. Wu, "Low precision inference on gpus," in *Proc. GTC*, 2019.