Leveraging Domain Information for the Efficient Automated Design of Deep Learning Accelerators

Chirag Sakhuja*
The University of Texas at Austin
chirag.sakhuja@utexas.edu

Zhan Shi*
The University of Texas at Austin
zshi17@cs.utexas.edu

Calvin Lin
The University of Texas at Austin
lin@cs.utexas.edu

***Abstract*—Deep learning accelerators are important tools for feeding the growing demand for deep learning applications. The automated design of such accelerators—which is important for reducing development costs—can be viewed as a search over a vast and complex design space that consists of all possible accelerators and all the possible software that could run on them. Unfortunately, this search is complicated by the existence of many ordinal and categorical values, which are critical to explore for the ultimate design but are not handled well by existing search techniques.**

This paper presents a technique for efficiently searching this space by injecting domain information—in this case information about hardware/software (HW/SW) co-design—into the automated search process. Specifically, this paper introduces a novel Bayesian optimization framework called daBO (domain-aware BO) that accepts domain information as input, including those describing ordinal and categorical values.

This paper also introduces Spotlight, a design tool based on daBO, and this paper empirically shows that Spotlight produces accelerator designs and software schedules that are orders of magnitude better than those created by the state-of-the-art. For example, for the ResNet-50 deep learning model, Spotlight produces a HW/SW configuration that reduces delay by $135\times$ over the configuration produced by ConfuciuX, a state-of-the-art HW/SW co-design tool, and Spotlight reduces energy-delay product (EDP) by $44\times$ over an Eyeriss-like accelerator, which is an edge-scale hand-designed accelerator. In the realm of cloud-scale accelerators, Spotlight reduces the EDP of a scaled-up Eyeriss-like accelerator by $23\times$. Our evaluation shows that Spotlight benefits from the efficiency of daBO, which allows Spotlight to identify accelerator designs and software schedules that prior work cannot identify.

I. Introduction

Deep learning (DL) models have had tremendous impact in fields such as computer vision, natural language processing, and speech recognition [73]. The varying demands of these models have driven an explosion of DL accelerators, which provide significant energy and performance benefits over CPUs and GPUs [11]–[14], [20], [26], [29], [33], [43], [45], [53].

Because DL accelerators can be deployed in a variety of situations—from datacenters to edge devices—they come in a large variety of shapes and sizes. And because DL models continue to rapidly evolve—sometimes incorporating new types of layers [22], [59] that do not execute efficiently on existing DL accelerators [13]—there is a constant desire to design new DL accelerators. Unfortunately, hardware design is expensive because of its lengthy design cycle [26], and the software stack often requires bespoke compilers and optimizations [35].

To significantly reduce the cost of accelerator design, we would ideally employ design automation. However, both the hardware design space—which considers architectural parameters such as buffer sizes and processing element (PE) arrangement—and the software design space—which considers loop optimization parameters such as permutations and tiling factors—are massive. Still, both prior work [46], [53] and our own results show that hardware/software (HW/SW) co-design is essential to the design of efficient DL accelerators.

Unfortunately, the co-design space exhibits unique characteristics that make it challenging to search automatically: (1) the co-design space is massive, e.g. a single layer of the ResNet-50 [22] DL model on a spatial array of PEs has $O(10^{18})$ configurations, (2) the co-design space is complex, as hardware and software parameters have complex interactions that render large and unpredictable parts of the co-design space infeasible or invalid, and (3) some parameters are ordinal (sortable but discontinuous values) or categorical (a set of arbitrary options), so performance and energy can vary wildly and unpredictably with minor changes to their values, i.e., there are performance and energy cliffs.

To search this vast co-design space, prior work has employed intelligent search algorithms, such as reinforcement learning [27], [66] or Bayesian optimization [24], [44], [48], [61], [66], [72]. Unfortunately, these techniques have largely relied on *off-the-shelf algorithms* which struggle with the complex portions of the design space, particularly with ordinal and categorical parameters [24], [44].

In this paper we introduce a novel *customized* Bayesian optimization framework, daBO (domain-aware BO), that overcomes the challenges of searching the HW/SW co-design space. Our key insight is that the search algorithm, which conventionally evaluates numerous samples to learn the shape of the co-design space, can be made more efficient by bootstrapping it with domain information. For example, a domain expert knows that the degree of parallelism, which is derived from the spatially unrolled dimension, the shape of the DL model, and the arrangement of processing elements, is a more accurate predictor of delay than any of the constituent parts alone. In designing daBO, we introduce a flexible method of providing high-level correlations, i.e. domain information, to the search algorithm. As a result, daBO is sample efficient—

*These authors contributed equally to this work.

i.e., it converges to a solution faster than prior techniques.

Because daBO is sample efficient, it can be applied to massive HW/SW co-design spaces, enabling it to find—in the same amount of time—solutions that are superior to those identified by other search techniques. Because it can leverage domain information, daBO can learn complex interactions between parameters. And because daBO can handle ordinal and categorical values, it can consider important design parameters that other techniques struggle with.

We use daBO as the basis for a new automated HW/SW co-design tool called Spotlight, which takes as input a set of DL models and a hardware budget. Spotlight then evaluates configurations using the MAESTRO [31] analytical model, and Spotlight produces as output (1) optimized microarchitectural parameters for a programmable DL accelerator and (2) optimized software schedules for each layer of the DL model.

This paper makes the following contributions:

- We present daBO (domain-aware BO), a novel Bayesian optimization framework that effectively deals with the ordinal and categorical search parameters that lead to discontinuities in the design space. In particular, daBO leverages domain information to efficiently learn correlations among categorical search parameters.
- We illustrate the benefits of daBO by presenting Spotlight, an open-source* automated HW/SW co-design tool that is built on daBO. We show that for the ResNet-50 DL model, Spotlight produces DL accelerator designs with $44\times$ lower energy-delay product (EDP) than an Eyeriss-like hand-designed accelerator and $135\times$ lower delay than a design created by ConfuciuX, a state-of-the-art HW/SW co-design tool. For the Transformer DL model, Spotlight achieves $902\times$ lower EDP than an NVDLA-like hand-designed accelerator and $52\times$ lower delay than a cloud-scale Eyeriss-like accelerator.
- We demonstrate that automated HW/SW co-design is critical for designing efficient DL accelerators. A significant part of Spotlight's benefit comes from the co-design of loop tile sizes with scratchpad sizes—a strategy that is made possible by daBO, which can efficiently explore the search space of tile sizes through the use of domain information.
- We empirically demonstrate that Spotlight exhibits several desirable properties.
  1) It is extremely sample efficient. We show that it can effectively search a co-design space of $O(10^{18})$ design points using just 100 hardware samples and 100 software samples per layer.
  2) It can find configurations that prior work completely ignores. Specifically, Spotlight considers both loop permutations and loop tiling factors for each dimension, while prior work in automated HW/SW co-design prunes this part of the co-design space.
  3) It is highly flexible and can be used in diverse design settings that include both edge-scale and cloud-scale designs: (1) It supports single-model co-design of accelerator parameters and software schedules, which is useful for FPGA deployment, and (2) it produces programmable accelerators that are able to efficiently execute DL models that they were not co-designed for—a property that is useful for ASIC deployment.

*https://github.com/chiragsakhuja/spotlight

```
for n := 0 to N
  for k := 0 to K
    for c := 0 to C
      for y := 0 to Y
        for x := 0 to X
          for r := 0 to R
            for s := 0 to S
              prod = Weights[k][c][s][r] *
                     Inputs[n][c][y][x]
              Outputs[n][k][y-s][x-r] += prod
```

Fig. 1: The 7-level loop used to compute a CONV layer.

The remainder of this paper is organized as follows. Sections II and III present background and related work. In Section IV we discuss our HW/SW co-design space and introduce our concept of a feature space. Section V introduces daBO, and Section VI describes Spotlight, which is evaluated in Section VII before we conclude in Section VIII.

II. Background

Deep learning (DL) models have diverse layer types such as convolutional, attention, and fully-connected layers that can be represented with the primitive operations of a convolution (CONV) or matrix multiplication (GEMM). Since these primitive operations are highly regular and constitute the majority of DL model inference time, they are popular targets for acceleration. In this work, we primarily focus on accelerating CONV operations, which can compute GEMM operations without loss of generality.

A. DL Layers

The CONV operation has many uses in DL models, including those that target image processing. Moreover, other types of layers, such as fully connected, GEMM, and depth-wise separable convolutions, can be represented as CONV layers.

The building block of CONV layers is the 3-D convolution operation, which operates on a weight tensor of size $R \times S \times C$ and an input tensor of size $X \times Y \times C$ to produce an output tensor of size $(X - R + 1) \times (Y - S + 1) \times 1$. The CONV operation is repeated for each of K weight tensors and N input tensors to produce $K \times N$ output tensors. Output tensors are reshaped such that subsequent layers are presented with N input tensors of size $(X - R + 1) \times (Y - S + 1) \times K$.

The CONV operation is implemented in software as a 7-level nested loop as shown in Figure 1.

Other operations can also be represented by CONV. For example, GEMM can be transformed, without loss of generality, to a CONV using an algorithm called col2im [1]. Also, depth-wise separable convolutions, which consists of a depth-wise convolution followed by a point-wise convolution, can be

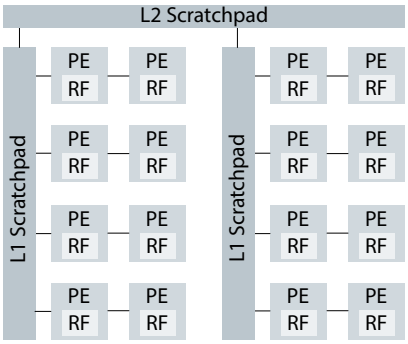

Fig. 2: An abstract DL accelerator architecture with 16 PEs, which are arranged in 2 clusters of 4×2 spatial arrays and 2 levels of scratchpads.

computed with CONV by computing each of its constituent parts independently. In both cases, some inefficiency may be introduced during the transformation.

Additionally, DL models incorporate important intermediate operations such as pooling and activation functions. Our work does not support these types of layers because of limitations in the analytical model backend, MAESTRO [31]. Fortunately, these layers do not contribute significantly to runtime of the DL model [7] MAESTRO also does not model cross-layer optimizations such as layer fusion, which are important components of modern DL models. These shortcomings have non-trivial implications, but we leave a solution to future work.

B. DL Accelerators

At a high level, DL accelerators comprise a spatial array of processing elements (PEs) or reduction trees that compute one or more Multiply-Accumulate (MAC) operations and are connected via multiple levels of on-chip scratchpad buffers and interconnects. Figure 2 shows an abstract DL accelerator with 16 PEs, 2 levels of scratchpads, and a register file (RF) within each PE. The architecture is a tree-like structure of scratchpads with leaves that are 2D spatial arrays of PEs. In this figure, there are two leaves of 2×4 PEs. Communication within a row of PEs occurs through a dedicated interconnect that is capable of uni-casting and multi-casting, and communication across rows occurs through queues in each scratchpad. This simple abstract accelerator design captures the high-level microarchitectural parameters of many popular DL accelerators [11], [12], [20], [26], [29], [33], [43], [45]. Edge-scale accelerators generally have one or two levels of scratchpads.

For a two-level DL accelerator, each of the seven loops in Figure 1 is broken into three tile sizes. The outer-most loops describe the data that is stored in the L2 scratchpad, and the inner-most loops describe the data that is stored in the RF. At each tile level, one dimension is spatially unrolled and is distributed evenly among the lower level processing clusters. If a dimension cannot be fully unrolled, then the partial tiles are streamed temporally.

C. Bayesian Optimization

Bayesian optimization (BO) is a search algorithm that converges to solutions in fewer total number of samples than competing algorithms, such as reinforcement learning [27], [66] and genetic algorithms [28], [66]. The key mechanism behind BO is the *surrogate model*, which is a probabilistic approximation of the cost function. The surrogate model is cheaper to query than the cost function, so it is consulted first when evaluating a candidate. To identify candidates that are worth evaluating on the slower cost function, BO generates a batch of candidates, queries their values on the surrogate model, ranks the candidates using an *acquisition function*, and selects the most promising candidates. Thus, if the surrogate model can accurately approximate the cost function, BO mostly selects high quality samples to evaluate. Unfortunately, the cost function of co-design has many invalid regions and behaves erratically, so it is difficult to train the surrogate model. In this work, we introduce a technique that overcomes these challenges to quickly and effectively train the surrogate model.

III. Related Work

The deep learning stack consists of (1) a DL model [22], [52], [54], [58], (2) a software optimizer [23], [28], [42], [46] or DL compiler [10], [36], which additionally performs code generation, and (3) a DL accelerator [9], [45], [53], [56].

Because each component of the stack has an enormous number of design points, prior work has focused on automating the co-design of two of the three components: either the accelerator and software mapping (HW/SW co-design) or the accelerator and DL model (HW/Model co-design).

End-to-end frameworks are a tangential type of automated design that iteratively transform a DL model into a fixed-function DL accelerator by projecting high-level representations into low-level representations until synthesizable hardware is produced. By contrast, HW/SW and HW/Model co-design simultaneously explore the joint space of two components of the deep learning stack. End-to-end frameworks can be augmented with HW/SW or HW/Model co-design, so we consider our work to be orthogonal to end-to-end frameworks.

A. HW/SW Co-Design

HW/SW co-design of DL accelerators aims to optimize the microarchitectural parameters of the accelerator alongside the loop structure of a single layer of a DL model. Interstellar [70] searches for the optimal loop to spatially unroll in the X and Y dimensions of a systolic array, but there are only a few hundred possibilities, so the design space is limited. dMazeRunner [16] and ZigZag [41] both present a vast software design space but only search a small hardware design space. MAGNet [61] uses off-the-shelf Bayesian optimization by first using heuristics to prune the software search space and then applying BO to the reduced hardware design space. In Section VII we compare Spotlight against HASCO [66], which uses reinforcement learning and Bayesian optimization, and ConfuciuX [27], which uses reinforcement learning and genetic algorithms,

but neither explores loop tiling options. Hypermapper [44] is a custom Bayesian optimization framework that accepts a limited and somewhat unintuitive form of domain information as input. VAESA [24] automatically learns a transformation of the complex design space into one that is easier for a search algorithm to explore. However, to learn the transformation, VAESA requires many samples.

Spotlight uses hand-provided domain information to transform the complex design space into a space that is more suitable for automated HW/SW co-design. Consequently, Spotlight can explore a large HW/SW co-design space comprised of complicated ordinal and categorical parameters, such as loop tiling sizes.

B. HW/Model Co-Design

Neural Architecture Search is the process of automatically designing the neural architecture of a DL model [5], [39], [49], [72], [75], and recent work has incorporated hardware design parameters into the search space. Reagen et al. [48] show that BO can effectively be used to co-design a model and accelerator, but their framework only searches over limited hardware design parameters. Other work limits the hardware design space by using hardware templates [21], [38], [69] or a few hardware parameters [3]. EDD [37] formulates the joint hardware-model design space as a differentiable search but only searches for a single parameter in the hardware design space.

HW/Model co-design is orthogonal to our work because our work accepts the DL model as input and searches the HW/SW design space for an optimized design.

C. End-to-End Frameworks

End-to-end frameworks, i.e., high-level synthesis tools, transform a high-level algorithmic description written in TensorFlow [2], C++, or a domain-specific language into fixed-function hardware. End-to-end frameworks, particularly those that target FPGAs and CGRAs, have been extensively studied [15], [18], [40], [60], but such work is orthogonal to ours, which aims to produce a programmable DL accelerator that can execute DL models that it was not explicitly designed for.

Hadjis and Olukotun [19] present a convenient framework that consumes a DL model and automatically deploys a specialized DL accelerator on an Amazon Web Services FPGA instance. The framework uses a series of independent tools, such as the Spatial [30] hardware design language and Vitis HLS [67], so there is limited room for co-design. Aurora [57] and REVAMP [6] use custom-designed CGRAs, which provide an effective tradeoff between reusability and performance, to generate workload-specific hardware. Other work restricts the hardware design space either by using hardware templates [17], [68], by limiting the design parameters that are searched [71], or by generating hardware that is highly specialized for the given algorithm [62], [64], [74].

IV. Co-Design Space

We now describe our co-design space, which is the Cartesian product of the hardware and software space of DL accelerators. We select a set of hardware parameters that, as prior work [25], [31], [32], [46], [61] has shown, captures a wide variety of DL accelerators and software optimizations. The resulting co-design space is massive—$O(10^{18})$ for a single layer of ResNet-50 running on a parameterizable accelerator (see Figure 3 for details).

Parameter	Range
SIMD Lanes	2 to 16
Bandwidth	64 to 256
PEs	128 to 300

(a) Cardinal parameters.

Parameter	Range	Stride
Scratchpad Size	64 to 256 KB	8
Register File Size	64 to 256 KB	8
PE Aspect Ratio	Divisors of PE Count	N/A
Tiling Factors†	Divisors of layer shape	N/A

(b) Ordinal parameters.

Parameter	Values
Loop Order†	Permutations of loops
Unroll Dimension†	N, K, C, R, S, X, Y

(c) Categorical parameters.

†Independent values per scratchpad level.

Fig. 3: HW/SW co-design parameter values.

We then present the notion of a feature space, which is our technique for reducing the complexity of the co-design space by using domain information.

A. Parameter Space

Our parameter space is composed of (1) high-level microarchitectural parameters for DL accelerators and (2) the full set of loop transformations that can be applied to the 7-level loop to compute a CONV.

1) Hardware Parameters: The hardware design space comprises the following prominent characteristics of DL accelerators: processing elements (PEs) count and arrangement in a 2D spatial array; the number of SIMD lanes in each PE; the size of the register files (RFs) that are in each PE; the size of a single global scratchpad; and the bandwidth of the simple interconnect, which supports uni-cast and multi-cast. To compare fairly against prior work, we use a fixed 8-bit precision. Figure 3 shows the range of values that Spotlight uses when designing an edge-scale accelerator.

2) Software Parameters: The software design space, which is independent for each layer of the DL model, consists of all loop transformations that can be applied to the CONV layer's 7-level loop. We consider three loop transformations: loop tiling, loop reordering, and spatial unrolling.

Loop tiling [65] is a common compiler optimization that improves data locality by splitting large loops into smaller loops that fit into on-chip caches or scratchpads. Each of the 7 loops in the CONV computation can be independently tiled. Naively, there are $(N \times K \times C \times R \times S \times X \times Y)^2$ options for loop tiling, but many of these options are invalid or require insertion of edge cases in the loops or padding in the memory

Feature	Calculation
Raw Cardinal Parameters	SIMD Lanes, On-Chip Bandwidth, Total # of PEs, Width of PE Array
Total Amount of On-Chip SRAM	Register File Size + Scratchpad Size
Parallelism Available in Kernel	$R_0 \times S_0$
Degree of Spatial Unrolling	Outer Loop Unrolled Tile Size × Inner Loop Unroll Tile Size
PE Utilization	$\frac{\text{DRAM Tile Size}}{\text{Outer Loop Unrolled Tile Size} \times \text{Height of PE Array}} \times \frac{\text{Outer Loop Unrolled Tile Size}}{\text{Inner Loop Unrolled Tile Size} \times \text{Width of PE Array}}$
Number of Loop Iterations to Completion	$\lceil \frac{\text{Outer Loop Unrolled Tile Size}}{\text{Height of PE Array}} \rceil \times \lceil \frac{\text{Inner Loop Unrolled Tile Size}}{\text{Width of PE Array}} \rceil$
Approximate Transfers from DRAM	$(X_0/X_2) \times (Y_0/Y_2) \times (\text{Width of PE array} + \text{Height of PE array})$
Size of Commonly Unrolled Dimensions	$2 \times X_0 + 3 \times Y_0 + 5 \times K_0 + 7 \times K_1 + 11 \times K_2$

Fig. 4: Features used as domain information by the search algorithm.

footprint. Our design space only considers loop tiling options that evenly divide the size of the layer.

After loop tiling is applied, the resulting 14 loops can be reordered in any of $(7!)^2$ permutations, and each permutation is a viable option. One loop of each level of loop tiling can also be spatially unrolled along a dimension of the spatial array. Our search space considers all 7^2 options.

3) Cardinal, Ordinal, and Categorical Parameters: Cardinal parameters, which take on integral values within a specified range, are straightforward for search algorithms to explore because they tend to exhibit appreciable trends. For example, as on-chip bandwidth is increased, energy consumption and area increase, while delay decreases. Ordinal parameters, which take on ordered values, are more complex to search if they have inconsistent spacing, but they can still exhibit appreciable trends. Categorical parameters, however, are problematic for search algorithms because they represent arbitrary values that have no correlation among them, so changes in their value have unpredictable implications. Figure 3 defines the type of each parameter in our parameter space of our co-design space.

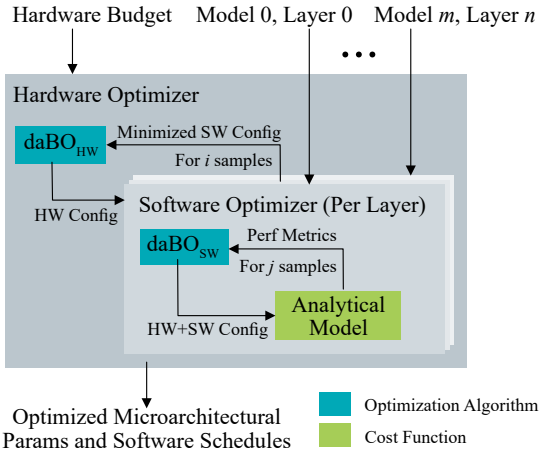

Fig. 5: Spotlight takes as input a hardware budget and a DL model and performs a nested optimization using our novel Bayesian optimization framework, daBO, to produce optimized microarchitectural parameters and software schedules.

B. Feature Space

The HW/SW co-design space of DL accelerators exhibits three unique challenges: (1) the co-design space is vast, (2) the co-design space is complex, with interactions among parameters rendering large portions of the space invalid, and (3) changes to the numerous ordinal and categorical parameters can result in erratic changes in behavior of the resulting design. Our technique of injecting domain information into the search overcomes these challenges.

1) Overview: To understand how domain information can improve a search algorithm's learning process, consider an example: It is well known that end-to-end delay is directly proportional to PE count and utilization, and given enough sample points, a search algorithm can learn this correlation on its own. However, it is sample efficient for an expert to explicitly highlight this correlation. Thus, domain information can be used (1) to guide the search toward profitable regions and away from invalid regions of the co-design space, and (2) to provide information on the behavior of parameters so that changes to these parameters are more predictable.

Typically, a search algorithm explores the parameter space directly, but we introduce the notion of a *feature space*, which comprises *features*, which are defined as an arbitrary transformation over the parameter space.

Concretely, let $\mathbb{P}$ be the set of HW/SW co-design parameters. The cost function, C, maps a point in $\mathbb{P}$ to its performance. The feature space is defined as any transformation $T : \mathbb{P} \rightarrow \mathbb{F}$, where $\mathbb{F}$ is the feature space and comprises individual features $f_i : \mathbb{P} \rightarrow \mathbb{R}$. The transformed cost function, which is what is learned by Spotlight, maps the performance of a point, $p \in \mathbb{P}$, as follows: $C'(T(p))$.

It is easier for a search algorithm to find correlations in C' than C. For example, it is unreasonable for a search algorithm to learn much useful information about delay from just the spatially unrolled dimension, which is a categorical parameter that takes on 7^2 unrelated values. By contrast, it is much more apparent that there is an inverse relationship between delay and degree-of-parallelism, which is a feature derived from the spatially unrolled dimension, the tiling factors, and the PE arrangement.

2) Feature Selection: The quality of the features determines the quality of the search, so thorough feature selection is critical. The selection of relevant and meaningful features is domain-specific, so we follow four general guidelines. (1) We ensure that categorical parameters are incorporated into one

or more features so that it is easier for the search algorithm to find correlations among them. (2) We encode domain information, i.e. well-known complex interactions among hardware and software parameters, as features. Examples of domain information are: the cost of data transfer among parts of the memory hierarchy and knowledge about the infeasible regions of the co-design space. (3) We design features that have linear trends so that the surrogate model in our Bayesian optimization framework can use a linear kernel, which can be computed more quickly than other common kernels such as Radial Basis Function and Matérn. (4) We verify the usefulness of each feature by computing permutation importance [4], [8].

We use these principles to brainstorm an initial set of 15 intuitive features, including buffer utilization, reuse volume, PE perimeter, and those in Figure 4. To ensure that the features are of high quality, we measure each feature's correlation with performance metrics by (1) computing each feature's value for millions of random HW/SW samples, and (2) visualizing the feature values against the performance of those samples. We discard any features that do not exhibit a strong correlation. Furthermore, to ensure that removal of a feature does not affect search quality, we evaluate our automated HW/SW co-design tool, Spotlight, both with and without these weakly- or uncorrelated features (see Section VI).

Figure 4 shows the final results of our feature selection process, including the equations used to compute each feature. We validate each of these features by ensuring that the correlations learned by the surrogate model are the same as those that we observe with our offline samples. The first features are simply raw cardinal parameters, which our search algorithm is already able to correlate well with performance metrics. Next, the total amount of on-chip SRAM is directly correlated with power consumption. The next three features—parallelism available in kernel, degree of parallelism in the spatially unrolled dimension, and PE utilization—measure available parallelism, which is a property of both the hardware and software, and is strongly correlated with delay. Next, some configurations can produce many edge cases that lead to a large tail latency, so we incorporate as features an approximation for the number of loop iterations for a layer to completely execute and the number of transfers of the input and kernel matrices from DRAM. Finally, we incorporate commonly unrolled spatial dimensions that are correlated with delay. We observe that each independent parameter—X_0, Y_0, K_0, etc.—has a weak, but notable, correlation with delay because the parameters generally take on fewer than 32 unique values, making it difficult to disambiguate them. For this feature, we spread out the number of unique values by using the prime numbers as the "basis vectors" to compute a linear combination of these parameters.

V. Domain-Aware BO

Our novel Bayesian optimization framework utilizes the notion of a feature space to efficiently search the co-design space.

As an optimizer, Bayesian optimization consists of two major components: (1) a surrogate model that predicts a Bayesian posterior probability distribution over the values of a cost function, and (2) an acquisition function that leverages the posterior distribution to suggest a design point to evaluate.

A. Surrogate Model

Conventionally, the surrogate model learns the characteristics of the parameter space to predict the cost function. With daBO, the surrogate model is trained on features instead of parameters. Candidate configurations are randomly generated in the parameter space, and daBO transforms them into the feature space before evaluating them on the surrogate model.

As is common practice, daBO uses Gaussian process (GP) as the surrogate model [47]. At a high level, GP learns a probabilistic approximation of the cost function by maintaining a Gaussian distribution for each point in the domain. More concretely, GP takes as input the features, denoted by $\mathbf{x}$, and predicts a posterior distribution based on prior distribution over the space of functions comprised of a mean function $m(\mathbf{x})$ and a covariance, or kernel function, $k(\mathbf{x}, \mathbf{x}')$. If the covariance for every point in the domain is 0, then GP exactly matches the function it is learning.

Typically, a Matérn or Radial Basis Function (RBF) kernel is employed because they can approximate a wide variety of cost functions, but both kernels have complexity of $O(N^3)$, and we find that they overfit to the evaluated samples. Instead, daBO employs a simple linear kernel, which has $O(N)$ complexity, takes far fewer samples to accurately model the trends of the cost function, and fits well with our feature selection.

B. Acquisition Function

The acquisition function selects the next configuration to evaluate on the cost function. A batch of candidate configurations is randomly generated in parameter space; each candidate is then transformed into feature space and evaluated on the posterior predictive distribution predicted by the surrogate model. daBO then uses Lower Confidence Bound [55] as the acquisition function, which is maximized to determine the next configuration to evaluate.

VI. Spotlight

Spotlight is a design automation tool that employs multiple instances of daBO to conduct automated HW/SW co-design. At a high level, Spotlight accepts as input a hardware budget and a set of layers from one or more DL models; for each input layer, Spotlight produces as output microarchitectural parameters for an optimized DL accelerator, along with optimized software schedules. Spotlight uses the MAESTRO [31] analytical model to evaluate configurations. Spotlight does not perform code generation or hardware synthesis. Figure 5 provides an overview of Spotlight.

A. Layerwise Optimization

It is challenging to optimize multiple layers of a model simultaneously, so Spotlight iteratively optimizes the hardware

and software configurations using a layerwise approach. Independent instances of daBO are used as the search algorithms for both hardware and software, so we denote the instances as daBO_{HW} and daBO_{SW}.

We use $\mathbf{x}_h$ and $\mathbf{x}_s$ to denote the set of hardware and software parameters in the parameter space. In Spotlight's layerwise approach, a hardware search is first performed by daBO_{HW} with the objective being to minimize $f(\mathbf{x}_h \mid \text{layers})$, which can be the energy-delay product (EDP) or delay of executing the DL model layers on the hardware configuration. Given the hardware configuration, Spotlight optimizes the software schedule by applying daBO_{SW} to each layer independently, with the objective being to minimize $f(\mathbf{x}_s \mid \mathbf{x}_h, \text{layer}_j)$, which is defined as the EDP or delay of running the layer j on the fixed hardware configuration. The software search produces a configuration that represents the best software schedule for each layer on the hardware configuration. The layerwise energies and delays are then summed to represent the cost of the hardware to compute aggregate EDP or delay, which is fed back to daBO_{HW} to generate the next hardware configuration. This concludes one iteration of search. The iterative search between hardware and software repeats for a user-defined number of trials.

B. Candidate Evaluation

To evaluate the cost of each design, we use MAESTRO [31] to report delay, energy, throughput, power, and area of DL accelerators. MAESTRO has been validated against RTL simulation, and our hardware and software design spaces naturally translate into MAESTRO's data-centric loop representation. MAESTRO models primitives, such as interconnects and convolutional layers, that are building blocks for DL accelerators and DL models.

Spotlight performs single objective optimization to minimize delay or EDP, which is a common metric for comparing DL accelerators [28]. From the pareto-optimal frontier, Spotlight selects the configuration that is closest to the inputted area and power budgets without exceeding them.

VII. Evaluation

We evaluate Spotlight in a variety of settings and against a variety of baselines. Unless otherwise specified, we evaluate Spotlight with 100 hardware samples and, for each hardware design and each layer, 100 software samples.

DL Models: We co-design separate DL accelerators with each of five DL models. Four models—VGG16 [54], ResNet-50 [22], MobileNetV2 [52], and MnasNet [58]—are popular for image processing and span nearly a decade of progress, including one model, MnasNet, which is automatically generated by neural architecture search (NAS). The fifth model is a single Transformer [59], which is a building block for the state-of-the-art natural language processing model, ALBERT [34].

Hand-Designed Accelerators: We compare Spotlight's optimized DL accelerator designs against three hand-designed accelerators: NVDLA-like [45], Eyeriss-like [12], and MAERI-like [33][†]. NVDLA and Eyeriss are popular edge-scale DL accelerators that have been fabricated. Both accelerators suffer from rigid dataflows that cannot always run modern DL models efficiently [13], [32], while MAERI, which is a more recent edge-scale accelerator that has not been fabricated, is designed to be highly flexible. For fairness, we evaluate Spotlight-generated accelerators and the hand-designed accelerators under our layerwise software optimizer, daBO_{SW} and we scale all accelerators so that they fit in the same area.

HW/SW Co-Design Tools: Where possible, we compare Spotlight against two state-of-the-art HW/SW co-design frameworks that also use the MAESTRO [31] ecosystem: ConfuciuX [27] and HASCO [66]. ConfuciuX optimizes with a combination of reinforcement learning and genetic algorithms, and HASCO uses a combination of Bayesian optimization and reinforcement learning. Both tools search limited software schedules; ConfuciuX selects among Eyeriss-like, NVDLA-like, and ShiDianNao-like, and HASCO uses a fixed software schedule. We evaluate ConfuciuX and HASCO with their out-of-the-box configurations. We do not show comparisons against Hypermapper [44], which is a domain-specific Bayesian optimization framework, because most runs do not terminate within four days of runtime (far longer than the scale of our evaluated results), and those that do produce designs on par with Eyeriss.

DL Accelerator Size: We generally use Spotlight to generate edge-scale accelerators with the parameters specified in Figure 3. Additionally, we optimize for a cloud-scale setting and compare against scaled-up hand-designed accelerators. To explore cloud-scale accelerators, the only change to Spotlight is the range of the parameter values that Spotlight explores—Spotlight works out-of-the-box without any other change to configuration.

Performance Metrics: Spotlight can minimize either delay or energy-delay product (EDP) under area and power constraints.

Design Scenarios: We present results for two different scenarios, which are described in more detail in their respective sections: single-model co-design (Section VII-A) and multi-model co-design (Section VII-B).

We conclude the evaluation with a discussion of Spotlight's benefits (Section VII-C), a deeper dive into daBO's behavior (Section VII-D), an ablation study (Section VII-E), and a comparison of results from a different analytical model than MAESTRO (Section VII-F).

A. Single-Model Co-Design

One use case for Spotlight is to co-design an accelerator with a full DL model. The generated accelerator can be deployed on an FPGA, which can be reconfigured for each new model, or it can be deployed as a highly specialized ASIC,

[†]We refer to the hand-designed accelerators as Eyeriss-like, NVDLA-like, and ShiDianNao-like because the MAESTRO model can only approximate their behavior.

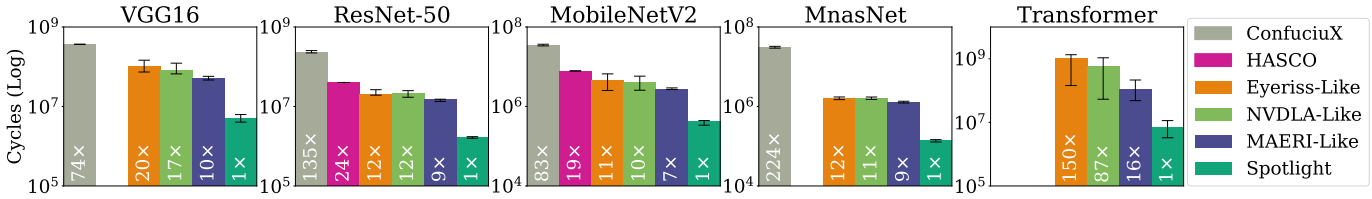

Fig. 6: Comparison of Spotlight against edge-scale hand-designed accelerators and those designed by state-of-the-art HW/SW co-design tools. The missing data is due to limitations of HASCO—which does not accept VGG16, MnasNet, or Transformer as inputs—and ConfuciuX—which cannot optimize Transformer. Lower is better.

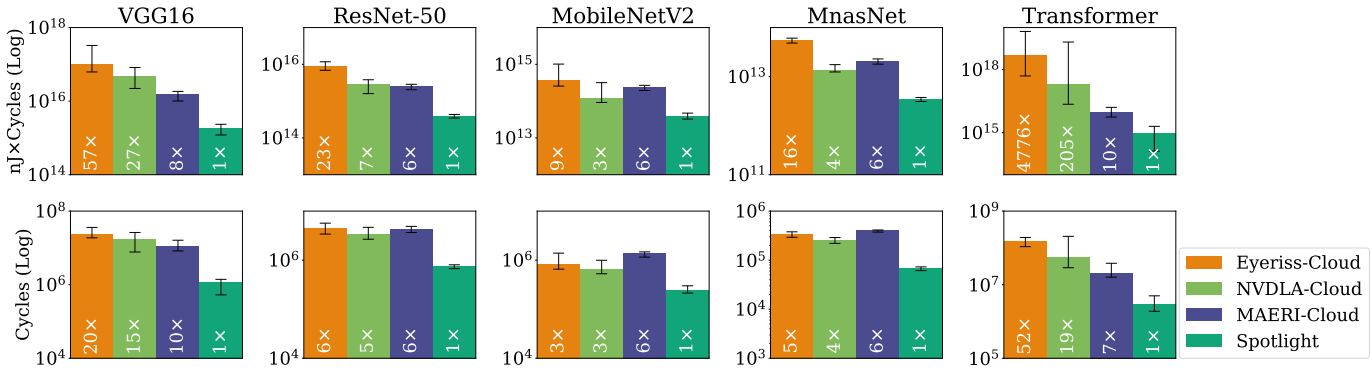

Fig. 7: Comparison of EDP (nJ×Cycles) and delay (Cycles) of Spotlight against scaled-up versions of hand-designed accelerators. Lower is better.

for example, in a low-power IoT device with a long lifetime and static workload.

The key takeaway from this first set of experiments: When co-designing with a single DL model, Spotlight produces designs that achieve significantly lower delay than hand-designed accelerators and those produced by other co-design tools.

Figure 6 shows the results when Spotlight co-designs edge-scale accelerators. Each bar represents the median delay of 10 independent trials, and the error bars indicate min/max of the trials. The missing data is due to limitations of HASCO and ConfuciuX, which cannot run all the selected DL models. This figure focuses on delay because HASCO and ConfuciuX cannot minimize energy-delay product (EDP). Notably, the trends when minimizing EDP are identical.

ConfuciuX and HASCO produce inefficient designs primarily because of their limited design spaces—neither aims to co-design loop tile sizes with scratchpad sizes, and we show in Section VII-C that co-design of these parameters is the primary reason that Spotlight performs well. Additionally, ConfuciuX and HASCO explore a severely limited set of software schedules, but we show in Section VII-E that this is not a crippling limitation.

Not surprisingly, of the hand-designed accelerators, MAERI generally achieves the lowest delay, followed by NVDLA and then Eyeriss. MAERI is highly flexible, so it can efficiently execute a wider variety of layer shapes than NVDLA and Eyeriss. NVDLA achieves lower delay than Eyeriss because it spatially unrolls the K and C dimensions, which exhibit higher parallelism in the mid and late layers of every evaluated model than the X and Y dimensions that Eyeriss unrolls. Eyeriss performs especially poorly on Transformer because we convert the GEMM operations that compose Transformer into CONV operations, which results in layer shapes that Eyeriss is not designed for efficiently executing.

Figure 7 presents results for cloud-scale accelerators when Spotlight minimizes EDP (top graphs) and delay (bottom graphs). We do not compare against HASCO or ConfuciuX because they do not support cloud-scale accelerators out-of-the-box. For this experiment, our only change to Spotlight was to change the range of parameters; we did not change the feature space or otherwise tune BO for the cloud setting. These results follow the same trends as the edge-scale accelerators.

B. Multi-Model Co-Design

Spotlight can also be used to co-design one accelerator with many DL models. Such an accelerator might be deployed as an ASIC, so it must efficiently execute a variety of DL models and remain efficient as new DL models are found.

Specifically, we consider two realistic deployment scenarios: (1) We assume that all the DL models are known at design-time, which is common for dedicated IoT accelerators; and (2) we assume that only a limited set of models is known at design-time, and the hardware is expected to generalize to unseen models.

The key takeaway: Spotlight can automatically design programmable DL accelerators that frequently outperform programmable hand-designed accelerators.

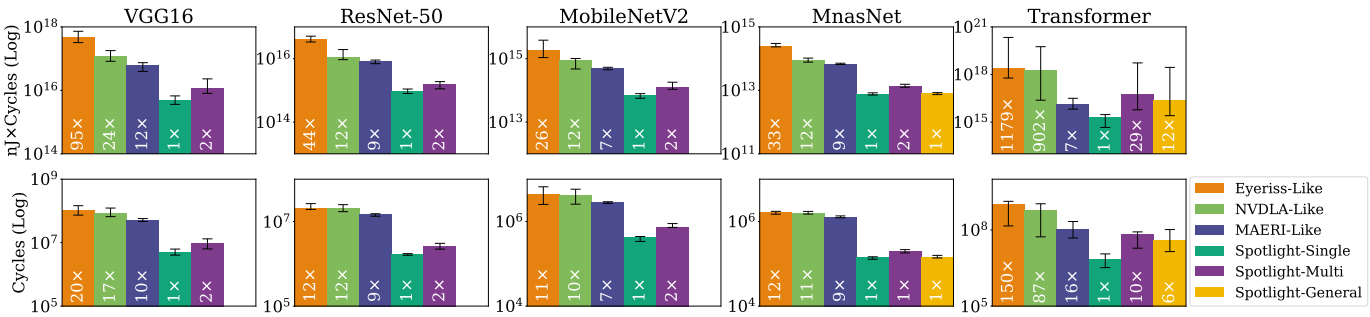

Fig. 8: The EDP (nJ×Cycles) and delay (Cycles) of the best designs found in the single-model co-design (green), the multi-model co-design (purple), and the generalization (yellow) scenarios. For the generalization scenario, we co-design the accelerator with VGG16 ResNet-50 and MobileNetV2, and we evaluate it on MnasNet and Transformer. Thus, only MnasNet and Transformer have yellow bars. Lower is better.

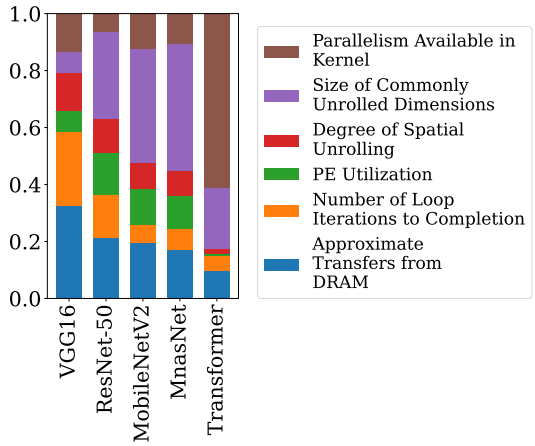

Fig. 9: The relative importance of each feature in daBO$_{SW}$.

Figure 8 shows results for both EDP (top graphs) and delay (bottom graphs), comparing Spotlight's design against hand-designed accelerators that are designed to generalize well. Spotlight-Single shows the results of single-model co-design, as described in Section VII-A, Spotlight-Multi shows the results of deployment scenario (1), and Spotlight-General shows the results of deployment scenario (2).

To emulate the first scenario, we co-design a DL accelerator with all five DL models as input to Spotlight and then re-run Spotlight's layerwise optimizer (daBO$_{SW}$) for each model independently on the resulting accelerator. Unsurprisingly, Spotlight-Multi has higher EDP and delay than Spotlight-Single because Spotlight-Single finely tunes each accelerator for a single model. However, Spotlight-Multi still almost always outperforms each hand-designed accelerator, highlighting the benefits of automated design.

To emulate the second scenario, we evaluate whether the hardware co-designed with a subset of DL models—VGG16, ResNet-50, and MobileNetV2—generalizes well to other DL models—MnasNet and Transformer. We co-design a DL accelerator by providing the first three models as input to Spotlight, and then for the resulting accelerator we run daBO$_{SW}$ independently for each of the last two models. We find that Spotlight-General has slightly higher EDP and delay than Spotlight-Single. Rather counterintuitively, we see that Spotlight-General has *lower* delay and EDP than Spotlight-Multi. We conjecture that when simultaneously co-designing for five models, daBO$_{HW}$ is unable to learn correlations among the complex software space spanning hundreds of unique layers, so the resulting accelerator is no longer as efficient for any single model.

C. Discussion

To understand the benefit of Spotlight, we compare its optimized configurations with the behavior of the hand-designed accelerators and HW/SW co-design tools.

The single most significant benefit of using Spotlight is its ability to co-design scratchpad sizes with tile sizes and loop unrolling properties, which leads to improved data locality. For example, given the same area and power budget, when Spotlight's optimized configuration, called Spotlight-Opt, runs ResNet-50, it achieves 26× higher throughput per Joule than Eyeriss, 28× higher than NVDLA, and 8.3× higher than MAERI. The main source of this improvement is greater input and weight reuse, computed as reads per fill, in the L1 scratchpad and RF. Eyeriss and NVDLA, which have rigid software schedules and fixed hardware, are unable to adjust unrolling parameters or on-chip memory sizes, so they cannot maintain high on-chip memory utilization for diverse layer shapes. MAERI supports flexible dataflows but still has fixed on-chip memory sizes, so it loses a degree of freedom compared to Spotlight, which finds a better balance between PE count and on-chip memory space than MAERI, so Spotlight-Opt has higher average utilization of both.

Qualitatively, the same reasoning explains Spotlight's improvement over HASCO and ConfuciuX. Neither HASCO nor ConfuciuX searches for tile sizes nor spatial unroll dimension, so these tools struggle to produce accelerator designs that match the efficiency of Spotlight-Opt.

Additionally, Spotlight achieves good results through a series of small wins, which designers often do not consider, during the execution of each layer. For example, we find that

Spotlight often produces accelerators with a long and narrow PE array, resulting in two benefits: (1) on the narrow side of the array, network latency is lower and there are fewer unicast operations, and (2) the layer edge cases, which result in low utilization and add tail latency, are smaller and thus have smaller impact on overall runtime. These results (1) illustrate the importance of co-design and (2) the benefits of automated co-design over manual co-design.

D. Feature Space Analysis

We have demonstrated that Spotlight can efficiently co-design DL accelerators and software schedules. We now peer into daBO to understand the source of Spotlight's benefits.

Surrogate Model Accuracy: We quantify the accuracy of the surrogate model, Guassian Process (GP), in predicting the behavior of the cost function. GP does not need to predict the absolute EDP or delay values, but it should be able to predict trends in these metrics so that the acquisition function can accurately select promising configurations.

To measure GP's predictability, we use a dataset of thousands of HW/SW samples and their respective EDP and delay. We use 90% of the dataset to train GP in two configurations—with a linear kernel and with a Matérn kernel—using the features described in Figure 4. We then predict the EDP and delay of the remaining 10% of the dataset. Rather than comparing directly with ground truth, we sort the predicted samples and the ground truth and compare the ordering using the Spearman rank correlation coefficient (ρ) [63], which measures the difference in ordering between vectors such that a score of 1 indicates a strong correlation and -1 indicates an inverse correlation.

Across the test set, ρ is equal to 0.0822 and 0.1127 for the linear and Matérn kernels, respectively. In both cases the correlation is quite low, but roughly 24% of the top 20% of samples are correctly predicted, which we find is sufficient for the acquisition function to select a high quality candidate. Though the Matérn kernel achieves a slightly higher correlation than the linear kernel, when we run Spotlight with the Matérn kernel we find no noticeable difference in search quality, so we opt for the simpler linear kernel. Nonetheless, the low correlation may warrant the use of a more sophisticated surrogate model.

Feature Importance: We rank the importance of each feature. For each instance of daBO_{SW} in single-model configuration, we conduct a commonly used experiment called permutation importance [4], [8]. After the GP is trained, we randomly perturb each feature in turn and measure the resulting change in the surrogate model's prediction. Features that cause large changes are considered to be more important.

Figure 9 shows the relative importance of each feature normalized for each model. Aside from Transformer, for which "parallelism available in the kernel" is dominant, no single feature is the sole indicator of performance. Parallelism is especially important for the Transformer model because Transformer is dominated by GEMM operations, which when converted to CONV operations result in large and uneven kernel sizes. In general, though, the most important feature varies.

We repeat this experiment with two modified configurations of Spotlight: (1) with only vanilla parameters instead of features (Spotlight-V) and (2) with the union of all features and raw parameters (Spotlight-A). We find the exact same result: There are typically a few features, which are different for each model, that are the most indicative of performance. We find that Spotlight-A produces accelerators that are on par with Spotlight, and both Spotlight and Spotlight-A produce better accelerators than Spotlight-V. This observation indicates that while good feature selection is still critical, Spotlight is somewhat resilient to the precise feature selection.

E. Ablation Study

To isolate the benefits of the daBO framework we compare sample convergence against ConfuciuX and four different search algorithms within the Spotlight tool, i.e., we replace daBO_{HW} and daBO_{SW} with each of the following five algorithms: genetic algorithm (Spotlight-GA), random search (Spotlight-R), vanilla BO (Spotlight-V), and BO with fixed software schedule options (Spotlight-F). More specifically, Spotlight-V is identical to off-the-shelf BO because it directly searches the parameter space instead of the feature space. Spotlight-F searches the Spotlight feature space, but it only searches among the three software schedules supported by ConfuciuX—namely, Eyeriss-like, NVDLA-like, and ShiDianNao-like—and it only searches for tiling factors in the K and C dimensions.

The key takeaway: Bayesian optimization is a strong starting point and is further enhanced by the introduction of the feature space. Moreover, most of the configurations selected by Bayesian optimization are superior to the best configuration produced by competing algorithms.

Figure 10 shows how each search algorithm, including ConfuciuX, converges—as a function of wall clock time—to a minimized EDP and delay when co-designing a single model. The shaded region represents the minimum and maximum of 10 search trials, and the solid line represents the median. We are unable to collect per-sample data with HASCO, so we denote with a dashed line the best result of HASCO's 10 trials.

BO consistently achieves lower EDP and delay than random search, genetic algorithm, ConfuciuX, and HASCO. Furthermore, our results suggest that given unlimited runtime, ConfuciuX may never achieve the same quality of solutions that Spotlight can achieve in a few hours. Moreover, both Spotlight and Spotlight-F, which use domain information, outperform Spotlight-V, which does not use domain information, by up to $2\times$ in all cases except for Transformer. For Transformer, we compute permutation importance [4], [8], as described in Section VII-D, on the parameter space of Spotlight-V and the feature space of Spotlight, and we find, unexpectedly, that the raw parameters have a larger impact on the surrogate model's prediction than our selected features. This observation explains why Spotlight-V outperforms Spotlight, and it highlights the

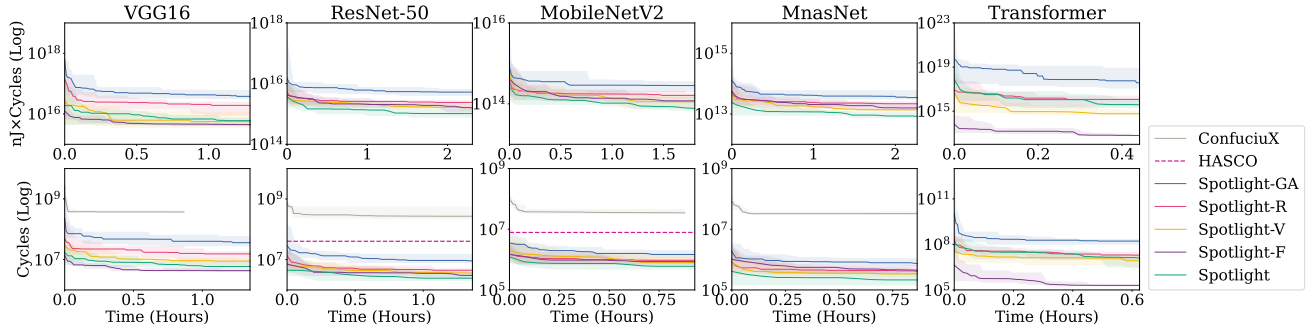

Fig. 10: The EDP (nJ×Cycles) and delay (Cycles) over time during single-model co-design for five search algorithms: Spotlight, three variations of Spotlight—random search (Spotlight-R), BO with fixed dataflow (Spotlight-F), and vanilla BO (Spotlight-V)—and two state-of-the-art co-design tools. For each layer of each hardware design, Spotlight and variations evaluate 100 sample points in the software design space. The solid line represents the median of 10 trials, and the shaded region represents the minimum and maximum. Lower is better.

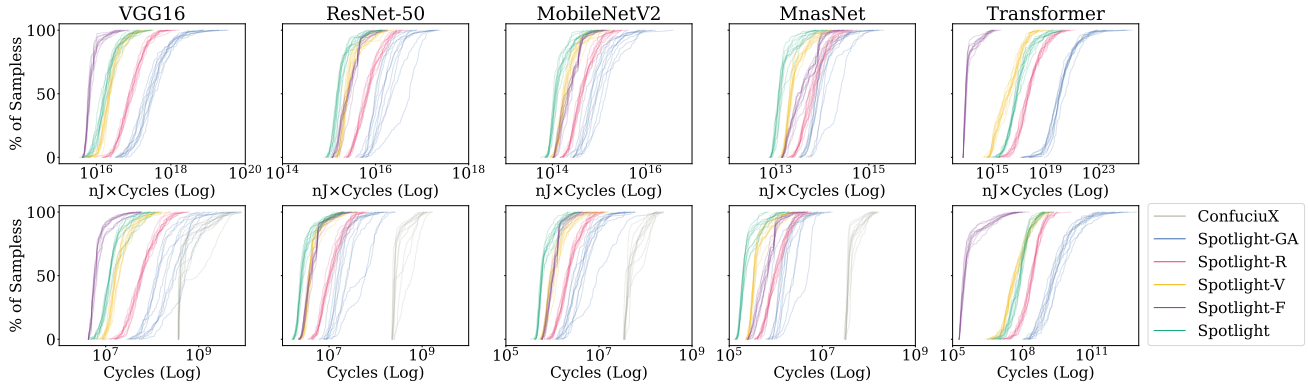

Fig. 11: Cumulative distribution function of hardware samples for each search algorithm. Each line represents the results from 1 of 10 trials. Further to the left is better.

importance of carefully selecting good for each specific application.

Our results also show that BO searches the co-design space more efficiently than other algorithms. The domain of the X axis of Figure 10 is set to the shortest wall clock time of the evaluated algorithms—in most cases, Spotlight-GA. Compared to Spotlight-GA, Spotlight-R evaluates 82% of the total number of samples, and Spotlight evaluates 52% of the total number of samples. Though Spotlight spends more time per-sample than Spotlight-GA and Spotlight-R, the improved sample efficiency of daBO results in superior results within the same wall clock time.

We find that Spotlight-F outperforms Spotlight for VGG16 and Transformer. Eyeriss is designed to be highly efficient when executing VGG16 [12], and indeed we find that when minimizing either EDP and delay, Spotlight-F selects an Eyeriss-like software schedule every time. Transformer is dominated by GEMM operations (converted to CONV), which NVDLA-like and ShiDianNao-like software schedules are able to execute efficiently. Because the software schedules that Spotlight-F explores are already tuned for the layers of VGG and Transformer, Spotlight-F has the advantage of exploring a simple yet high-quality co-design space that can be explored quicker than the co-design space of Spotlight, so Spotlight-F achieves superior results within the same wall clock time.

To further evaluate the quality of each search algorithm, we present Figure 11. This figure plots the cumulative distribution function (CDF) of hardware sample points, which shows the percentage of total sample points evaluated that achieve a given EDP or delay. Each line represents 1 of 10 trials.

The CDFs for Spotlight and Spotlight-F are further left than those of the competing algorithms, which indicates that Spotlight does not just find a single good configuration but rather consistently finds configurations that outperform the best configurations found by competing algorithms.

The CDF for Spotlight-R is Guassian, while the other search algorithms have a steep initial slope, which means that many of the sample points achieve EDP or delay that is similar to the final optimized configuration. Specifically, 81.7% of the hardware samples that Spotlight selects are better than the best results that Spotlight-R finds. So it is clear that BO is conducting a higher quality search.

F. Evaluations Using Timeloop

Because Spotlight uses an analytical model to evaluate candidate designs, there is a risk that Spotlight's designs overfit to the MAESTRO analytical model—i.e., the designs exploit features of MAESTRO that do not represent actual hardware. To evaluate this risk, we measure the performance of Spotlight's designs using a separate analytical model, namely, Timeloop [46].

Timeloop is an analytical model that given an accelerator specification, estimates the performance of a single layer of a DL model. Compared to MAESTRO, Timeloop provides more precise control over the accelerator specification and software schedule. Consequently, it is less intuitive and exposes larger hardware and software design spaces.

To compare Timeloop and MAESTRO, we take the results in Figure 6, which evaluates 100 samples for each layer of each DL model, and perform the following for each layer: (1) We compute the performance of the 100 samples with both MAESTRO and Timeloop, (2) we sort the results, and (3) we quantify the similarity between the two analytical models. On average, across all layers, 35% of the highest 20 and lowest 20 samples match. These results indicate that Spotlight does not overfit to MAESTRO but that some caution should be taken when considering the optimality of specific designs. If Spotlight's designs are to be applied to another medium, such as to actual hardware, our recommendation is to evaluate each of the top 20 designs on the other medium.

VIII. Conclusions

In this paper, we have presented Spotlight, an automated framework for performing hardware/software (HW/SW) co-design of deep learning accelerators. We have also presented daBO, our novel Bayesian optimization framework that is critical to Spotlight's success because it incorporates domain information into the automated search process. We have empirically demonstrated that Spotlight can produce highly efficient HW/SW co-designs that are orders of magnitude better than competing solutions, including both manually designed accelerators and those designed by state-of-the-art tools.

Philosophically, we observe that previous work [16], [61], [70] manually applies domain information to define dramatically smaller co-design spaces to search, but because the co-design space is so complex, this manual pruning apparently removes many of the best design points from the search space. By contrast, Spotlight gets great power by embracing a vast co-design space and incorporating the domain information into the automated search process, thereby giving Spotlight a mechanism for finding many of the best design points.

As we look to the future, we see many potential uses for Spotlight. For example, we believe that Spotlight's sample efficiency will be amplified by more costly but more accurate evaluation backends, such as FPGA emulation. Moreover, Spotlight can be integrated with widely-studied neural architecture search techniques to fully explore the joint space of hardware, software, and neural models.

Acknowledgments

This work was funded in part by NSF Grant CCF-1823546, a gift from Intel Corporation through the NSF/Intel Partnership on Foundational Microarchitecture Research, and an NXP fellowship. We thank Kevin Swersky and Milad Hashemi for their collaboration on foundational concepts that this paper builds on, and we thank Minesh Patel and Molly O'Neil for their comments on early drafts of this paper.

References

[1] col2im. [Online]. Available: https://www.mathworks.com/help/images/ref/col2im.html

[2] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," in *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. USENIX Association, 2016, pp. 265–283.

[3] M. S. Abdelfattah, Ł. Dudziak, T. Chau, R. Lee, H. Kim, and N. D. Lane, "Best of both worlds: Automl codesign of a cnn and its hardware accelerator," in *Proceedings of the 57th IEEE/ACM Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[4] A. Altmann, L. Toloi, O. Sander, and T. Lengauer, "Permutation importance: a corrected feature importance measure," *Bioinformatics*, vol. 26, no. 10, pp. 1340–1347, 04 2010. [Online]. Available: https://doi.org/10.1093/bioinformatics/btq134

[5] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," *arXiv preprint arXiv:1611.02167*, 2016.

[6] T. K. Bandara, D. Wijerathne, T. Mitra, and L.-S. Peh, "Revamp: a systematic framework for heterogeneous cgra realization," in *Proceedings of the 27th International Conference on Architectural Support for Operating Systems (ASPLOS)*, 2022, pp. 918–932.

[7] A. Boroumand, S. Ghose, B. Akin, R. Narayanaswami, G. F. Oliveira, X. Ma, E. Shiu, and O. Mutlu, "Google neural network models for edge devices: Analyzing and mitigating machine learning inference bottlenecks," in *Proceedings of the 30th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE, 2021, pp. 159–172.

[8] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[9] M. Capra, B. Bussolino, A. Marchisio, M. Shafique, G. Masera, and M. Martina, "An updated survey of efficient hardware architectures for accelerating deep convolutional neural networks," *Future Internet*, vol. 12, no. 7, p. 113, 2020.

[10] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, H. Shen, M. Cowan, L. Wang, Y. Hu, L. Ceze *et al.*, "{TVM}: An automated end-to-end optimizing compiler for deep learning," in *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. IEEE, 2018, pp. 578–594.

[11] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," in *Proceedings of the 19th International Conference on Architectural Support for Operating Systems (ASPLOS)*. IEEE, 2014, pp. 269–284.

[12] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *Proceedings of the 43rd IEEE/ACM International Symposium on Computer Architecture (ISCA)*. IEEE, 2016, pp. 367–379.

[13] Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 292–308, 2019.

[14] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun *et al.*, "Dadiannao: A machine-learning supercomputer," in *Proceedings of the 47th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2014, pp. 609–622.

[15] S. Dave, R. Baghdadi, T. Nowatzki, S. Avancha, A. Shrivastava, and B. Li, "Hardware acceleration of sparse and irregular tensor computations of ml models: A survey and insights," *Proceedings of the IEEE*, vol. 109, no. 10, pp. 1706–1752, 2021.

[16] S. Dave, Y. Kim, S. Avancha, K. Lee, and A. Shrivastava, "dmazerunner: Executing perfectly nested loops on dataflow accelerators," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 18, no. 5s, pp. 1–27, 2019.

[17] Y. Guan, H. Liang, N. Xu, W. Wang, S. Shi, X. Chen, G. Sun, W. Zhang, and J. Cong, "FP-DNN: an automated framework for mapping deep neural networks onto FPGAs with rtl-hls hybrid templates," in *Proceedings of the 25th IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2017, pp. 152–159.

[18] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, "A survey of FPGA-based neural network accelerator," *arXiv preprint arXiv:1712.08934*, 2017.

[19] S. Hadjis and K. Olukotun, "Tensorflow to cloud fpgas: Tradeoffs for accelerating deep neural networks," in *Proceedings of the 29th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2019, pp. 360–366.

[20] T. J. Ham, L. Wu, N. Sundaram, N. Satish, and M. Martonosi, "Graphicionado: A high-performance and energy-efficient accelerator for graph analytics," in *Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2016, pp. 1–13.

[21] C. Hao, X. Zhang, Y. Li, S. Huang, J. Xiong, K. Rupnow, W.-m. Hwu, and D. Chen, "FPGA/DNN co-design: An efficient design methodology for iot intelligence on the edge," in *Proceedings of the 56th IEEE/ACM Design Automation Conference (DAC)*. IEEE, 2019, pp. 1–6.

[22] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the 2016 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2016, pp. 770–778.

[23] K. Hegde, P.-A. Tsai, S. Huang, V. Chandra, A. Parashar, and C. W. Fletcher, "Mind mappings: Enabling efficient algorithm-accelerator mapping space search," in *Proceedings of the 26th International Conference on Architectural Support for Operating Systems (ASPLOS)*. IEEE, 2021, pp. 943–958.

[24] Q. Huang, C. Hong, J. Wawrzynek, M. Subedar, and Y. S. Shao, "Learning a continuous and reconstructible latent space for hardware accelerator design," in *Proceedings of the 2022 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2022, pp. 277–287.

[25] L. Jia, Z. Luo, L. Lu, and Y. Liang, "Analyzing the design space of spatial tensor accelerators on FPGAs," in *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2021, pp. 230–235.

[26] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th IEEE/ACM International Symposium on Computer Architecture (ISCA)*, 2017, pp. 1–12.

[27] S.-C. Kao, G. Jeong, and T. Krishna, "ConfuciuX: autonomous hardware resource assignment for DNN accelerators using reinforcement learning," in *Proceedings of the 53rd IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 622–636.

[28] S.-C. Kao and T. Krishna, "GAMMA: automating the hw mapping of DNN models on accelerators via genetic algorithm," in *Proceedings of the 2020 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2020, pp. 1–9.

[29] S. Knowles, "Graphcore," in *2021 IEEE Hot Chips 33 Symposium (HCS)*. IEEE, 2021, pp. 1–25.

[30] D. Koeplinger, M. Feldman, R. Prabhakar, Y. Zhang, S. Hadjis, R. Fiszel, T. Zhao, L. Nardi, A. Pedram, C. Kozyrakis *et al.*, "Spatial: A language and compiler for application accelerators," in *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*. IEEE, 2018, pp. 296–311.

[31] H. Kwon, P. Chatarasi, V. Sarkar, T. Krishna, M. Pellauer, and A. Parashar, "MAESTRO: a data-centric approach to understand reuse, performance, and hardware cost of DNN mappings," *IEEE Micro*, vol. 40, no. 3, pp. 20–29, 2020.

[32] H. Kwon, M. Pellauer, A. Parashar, and T. Krishna, "Flexion: A quantitative metric for flexibility in DNN accelerators," *IEEE Computer Architecture Letters (CAL)*, vol. 20, no. 1, pp. 1–4, 2020.

[33] H. Kwon, A. Samajdar, and T. Krishna, "Maeri: Enabling flexible dataflow mapping over DNN accelerators via reconfigurable interconnects," in *Proceedings of the 23rd International Conference on Architectural Support for Operating Systems (ASPLOS)*. IEEE, 2018, pp. 461–475.

[34] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "Albert: A lite bert for self-supervised learning of language representations," *arXiv preprint arXiv:1909.11942*, 2019.

[35] C. Lattner, M. Amini, U. Bondhugula, A. Cohen, A. Davis, J. Pienaar, R. Riddle, T. Shpeisman, N. Vasilache, and O. Zinenko, "Mlir: Scaling compiler infrastructure for domain specific computation," in *Proceedings of the 2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. IEEE, 2021, pp. 2–14.

[36] M. Li, Y. Liu, X. Liu, Q. Sun, X. You, H. Yang, Z. Luan, L. Gan, G. Yang, and D. Qian, "The deep learning compiler: A comprehensive survey," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 32, no. 3, pp. 708–727, 2020.

[37] Y. Li, C. Hao, X. Zhang, X. Liu, Y. Chen, J. Xiong, W.-m. Hwu, and D. Chen, "Edd: Efficient differentiable dnn architecture and implementation co-search for embedded ai solutions," in *Proceedings of the 57th IEEE/ACM Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[38] Y. Lin, D. Hafdi, K. Wang, Z. Liu, and S. Han, "Neural-hardware architecture search," 2019, mL for Systems.

[39] Y. Lin, M. Yang, and S. Han, "Naas: Neural accelerator architecture search," in *Proceedings of the 58th IEEE/ACM Design Automation Conference (DAC)*. IEEE, 2021, pp. 1051–1056.

[40] L. Liu, J. Zhu, Z. Li, Y. Lu, Y. Deng, J. Han, S. Yin, and S. Wei, "A survey of coarse-grained reconfigurable architecture and design: Taxonomy, challenges, and applications," *ACM Computing Surveys (CSUR)*, vol. 52, no. 6, pp. 1–39, 2019.

[41] L. Mei, P. Houshmand, V. Jain, S. Giraldo, and M. Verhelst, "ZigZag: enlarging joint architecture-mapping design space exploration for DNN accelerators," *IEEE Transactions on Computers (TC)*, 2021.

[42] G. E. Moon, H. Kwon, G. Jeong, P. Chatarasi, S. Rajamanickam, and T. Krishna, "Evaluating spatial accelerator architectures with tiled matrix-matrix multiplication," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 4, pp. 1002–1014, 2021.

[43] T. Moreau, T. Chen, L. Vega, J. Roesch, E. Yan, L. Zheng, J. Fromm, Z. Jiang, L. Ceze, C. Guestrin *et al.*, "A hardware–software blueprint for flexible deep learning specialization," *IEEE Micro*, vol. 39, no. 5, pp. 8–16, 2019.

[44] L. Nardi, D. Koeplinger, and K. Olukotun, "Practical design space exploration," in *Proceedings of the 27th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, 2019, pp. 347–358.

[45] NVIDIA, "Nvidia deep learning accelerator." [Online]. Available: http://nvdla.org/

[46] A. Parashar, P. Raina, Y. S. Shao, Y.-H. Chen, V. A. Ying, A. Mukkara, R. Venkatesan, B. Khailany, S. W. Keckler, and J. Emer, "Timeloop: A systematic approach to dnn accelerator evaluation," in *Proceedings of the 2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2019, pp. 304–315.

[47] C. E. Rasmussen and C. K. Williams, *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[48] B. Reagen, J. M. Hernández-Lobato, R. Adolf, M. Gelbart, P. Whatmough, G.-Y. Wei, and D. Brooks, "A case for efficient accelerator design space exploration via bayesian optimization," in *Proceedings of the 2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, 2017, pp. 1–6.

[49] P. Ren, Y. Xiao, X. Chang, P.-Y. Huang, Z. Li, X. Chen, and X. Wang, "A comprehensive survey of neural architecture search: Challenges and solutions," *arXiv preprint arXiv:2006.02903*, 2020.

[50] C. Sakhuja, Z. Shi, and C. Lin, "Spotlight," Nov 2022. [Online]. Available: https://doi.org/10.5281/zenodo.7269574

[51] C. Sakhuja, Z. Shi, and C. Lin, "Spotlight," Nov 2022. [Online]. Available: https://github.com/chiragsakhuja/spotlight

[52] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 4510–4520.

[53] Y. S. Shao, J. Clemons, R. Venkatesan, B. Zimmer, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. Pinckney, P. Raina, S. G. Tell, Y. Zhang, W. J. Dally, J. Emer, and C. T. Gray, "Simba: Scaling deep-learning inference with multi-chip-module-based architecture," in *Proceedings of the 52nd IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2019, pp. 14–27.

[54] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.
[55] N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger, "Gaussian process optimization in the bandit setting: No regret and experimental design," *arXiv preprint arXiv:0912.3995*, 2009.
[56] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.
[57] C. Tan, C. Xie, A. Li, K. J. Barker, and A. Tumeo, "Aurora: Automated refinement of coarse-grained reconfigurable accelerators," in *Proceedings of the 2021 Design, Automation and Test in Europe Conference and Exhibition (DATE)*. IEEE, 2021, pp. 1388–1393.
[58] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," in *Proceedings of the 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2019, pp. 2820–2828.
[59] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.
[60] S. I. Venieris, A. Kouris, and C.-S. Bouganis, "Toolflows for mapping convolutional neural networks on FPGAs: A survey and future directions," *ACM Computing Surveys (CSUR)*, vol. 51, no. 3, pp. 1–39, 2018.
[61] R. Venkatesan, Y. S. Shao, M. Wang, J. Clemons, S. Dai, M. Fojtik, B. Keller, A. Klinefelter, N. Pinckney, P. Raina *et al.*, "Magnet: A modular accelerator generator for neural networks," in *Proceedings of the 2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2019, pp. 1–8.
[62] J. Wang, L. Guo, and J. Cong, "Autosa: A polyhedral compiler for high-performance systolic arrays on FPGA," in *Proceedings of the 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*. IEEE, 2021, pp. 93–104.
[63] E. W. Weisstein. Spearman rank correlation coefficient. From MathWorld—A Wolfram Web Resource. From MathWorld—A Wolfram Web Resource. [Online]. Available: https://mathworld.wolfram.com/SpearmanRankCorrelationCoefficient.html
[64] J. Weng, S. Liu, V. Dadu, Z. Wang, P. Shah, and T. Nowatzki, "Dsagen: Synthesizing programmable spatial accelerators," in *Proceedings of the 47th IEEE/ACM International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 268–281.
[65] M. Wolfe, *Optimizing Supercompilers for Supercomputers*. Cambridge, MA: MIT Press, 1989.
[66] Q. Xiao, S. Zheng, B. Wu, P. Xu, X. Qian, and Y. Liang, "Hasco: Towards agile hardware and software co-design for tensor computation," in *Proceedings of the 48th IEEE/ACM International Symposium on Computer Architecture (ISCA)*, 2021, pp. 1055–1068.
[67] Xilinx, "Vitis high-level synthesis." [Online]. Available: https://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html
[68] P. Xu, X. Zhang, C. Hao, Y. Zhao, Y. Zhang, Y. Wang, C. Li, Z. Guan, D. Chen, and Y. Lin, "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 (FPGA)*. ACM, 2020, pp. 40–50.
[69] L. Yang, Z. Yan, M. Li, H. Kwon, L. Lai, T. Krishna, V. Chandra, W. Jiang, and Y. Shi, "Co-exploration of neural architectures and heterogeneous asic accelerator designs targeting multiple tasks," in *Proceedings of the 57th IEEE/ACM Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.
[70] X. Yang, M. Gao, Q. Liu, J. Setter, J. Pu, A. Nayak, S. Bell, K. Cao, H. Ha, P. Raina *et al.*, "Interstellar: Using halide's scheduling language to analyze DNN accelerators," in *Proceedings of the 25th International Conference on Architectural Support for Operating Systems (ASPLOS)*, 2020, pp. 369–383.
[71] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*. ACM, 2015, pp. 161–170.
[72] D. Zhang, S. Huda, E. Songhori, K. Prabhu, Q. Le, A. Goldie, and A. Mirhoseini, "A full-stack search technique for domain optimized deep learning accelerators," in *Proceedings of the 27th International Conference on Architectural Support for Operating Systems (ASPLOS)*, 2022, pp. 27–42.
[73] D. Zhang, N. Maslej, E. Brynjolfsson, J. Etchemendy, T. Lyons, J. Manyika, H. Ngo, J. N. Carlos, M. Sellitto, E. Sakhaee, Y. Shoham, J. Clark, and R. Perrault, "The ai index 2022 annual report," Human-Centered AI Institute, Stanford University, Stanford, CA, Tech. Rep., Mar 2022. [Online]. Available: https://aiindex.stanford.edu/wp-content/uploads/2021/03/2021-AI-Index-Report_Master.pdf
[74] X. Zhang, J. Wang, C. Zhu, Y. Lin, J. Xiong, W.-m. Hwu, and D. Chen, "DnnBuilder: An automated tool for building high-performance DNN hardware accelerators for FPGAs," in *Proceedings of the 2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2018, pp. 1–8.
[75] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.

APPENDIX

A. Abstract

This artifact provides the source code for Spotlight as well as scripts that aid with reproduction of key results. The latter includes the raw values shown in Figures 6-8 and the end results of Figure 10.

B. Artifact check-list (meta-information)

- Runs Spotlight and Spotlight variants.
- Supports DL models: VGG16, ResNet-50, MobileNetV2, MnasNet, and Transformer.
- Supports hand-designed accelerator baselines: Eyeriss, NVDLA, and MAERI.
- Includes scripts to produce and compare results.
- Outputs all sample points and final results for architectural parameters and software schedules.
- Setup takes 20 minutes.
- Runtime takes 5-9 days, depending on the number of workloads being evaluated. Runtime can be significantly reduced if more parallelism is available.

C. Access to Artifact

The artifact is permanently archived on Zenodo [50] and is also publicly released on GitHub [51].

D. System Requirements and Dependencies

In this appendix, for convenience, we present a workflow using Docker. However, the README.md in the artifact additionally outlines a native installation process, which requires lower overhead and may have slightly higher performance.

System requirements:

- 10 GB of disk space
- 16+ GB of RAM recommended
- 8+ core CPU recommended

Software dependencies:

- Docker
- Linux installation recommended

E. Setup

Build the Docker image and set up a new container. (20 minutes)

```
$ docker build -t spotlight .
$ docker run -it spotlight /bin/bash
```

From within the Docker container, activate the Python environment and build Spotlight. (1 minute).

```
$ conda activate spotlight-ae
$ scons -j$(nproc)
```

F. Workflow

If the container is not running, then start it up.

```
$ docker container ls -a
  (To get Container ID)
$ docker start <Container ID>
$ docker exec -it <Container ID> /bin/bash
```

The starting point for all runs is the `run-ae.sh` script, which launches the experiments. There are four modes, each one corresponding to the key figures: Main-Edge (Figure 6), Main-Cloud (Figure 7), General (Figure 8), and Ablation (Figure 10).

The following is a brief description of each mode, including its expected runtime. The runtime can be fairly high in some cases, but it can be reduced if more parallelism is available. We were able to complete all runs within a couple of days by running on a cluster instead of a single machine. However, the `run-ae.sh` script requires modifications to support parallelism across machines.

- Main-Edge: Runs Spotlight to generate an edge-scale, fine-tuned DL accelerator. It fine-tunes separately for EDP and delay, and it runs Spotlight independently for each of the 5 models. Finally, it also searches for optimal software schedules and hardware configurations for Eyeriss, NVDLA, and MAERI. The expected runtime, if running a single trial for each configuration, is 1 day.
- Main-Cloud: Runs a similar set of workloads as Main-Edge, but it performs them for a cloud-scale accelerator, including cloud-scaled versions of Eyeriss, NVDLA, and MAERI. The cloud-scale design space contains many more invalid points than the edge-scale design space, so the expected runtime is anywhere between 3 and 7 days. It is recommended to perform other comparisons before Main-Cloud, because Main-Cloud can have highly variable runtimes.
- General: Runs Spotlight's software optimizer with accelerator parameters that were generated by Spotlight when fine-tuning an accelerator for either (1) all 5 DL models simultaneously or (2) VGG16, ResNet-50, and MobileNetV2 simultaneously. In scenario (1), all 5 DL models are independently optimized to see if the accelerator is flexible. In scenario (2), only MnasNet and Transformer are optimized to see if the accelerator can generalize to efficiently process models it wasn't fine-tuned for. Since only the software optimizer is running in this mode, its runtime is roughly 10 minutes.
- Ablation: Runs the remaining variants of Spotlight: Spotlight-GA, Spotlight-R, Spotlight-V, and Spotlight-F. The expected runtime is roughly 1 day.

All runs can be executed with the following simple commands.

```
$ ./run-ae.sh main-edge
$ ./run-ae.sh main-cloud
$ ./run-ae.sh ablation
$ ./run-ae.sh general
```

Each command can optionally be augmented with the `--trials N` flag, where `N` indicates the number of independent trials for which each experiment runs. Though we use 10 trials for all evaluations in this work, we recommend sticking with the default value of 1 unless more parallelism is available. As it stands, multiple trials are run in series.

Results are stored in the `results` directory.

G. Steps for Evaluation and Results

After the runs are complete, then results can be analyzed using the `compare-ae.sh` script. Similar to the `run-ae.sh` script, the user specifies a comparison mode that corresponds to Figures 6-8 and 10.

The script is run with the following simple commands.

```
$ ./compare-ae.sh main-edge
$ ./compare-ae.sh main-cloud
$ ./compare-ae.sh ablation
$ ./compare-ae.sh general
```

The `compare-ae.sh` script outputs a CSV file to standard output. The columns of the CSV are the configuration type, the minimum achieved performance across all trials, the maximum achieved performance across all trials, the median performance across all trials, and the median normalized to the median of Spotlight. This output can be used to evaluate Figures 6-8 directly, although numbers may vary slightly due to the randomness of the optimization process. The CSV output corresponds to the final endpoint of each line in Figure 10.

For more thorough details, see the README.md in the artifact.