Learning A Continuous and Reconstructible Latent Space for Hardware Accelerator Design

Qijing Huang* *NVIDIA*
Charles Hong *UC Berkeley*
John Wawrzynek *UC Berkeley*
Mahesh Subedar *Intel Labs*
Yakun Sophia Shao *UC Berkeley*

***Abstract*—The hardware design space is high-dimensional and discrete. Systematic and efficient exploration of this space has been a significant challenge. Central to this problem is the intractable search complexity that grows exponentially with the design choices and the discrete nature of the search space. This work investigates the feasibility of learning a meaningful low-dimensional continuous representation for hardware designs to reduce such complexity and facilitate the search process. We devise a variational autoencoder (VAE)-based design space exploration framework called VAESA, to encode the hardware design space in a compact and continuous representation. We show that black-box and gradient-based design space exploration algorithms can be applied to the latent space, and design points optimized in the latent space can be reconstructed to high-performance realistic hardware designs. Our experiments show that performing the design space search on the latent space consistently leads to the optimal design point under a fixed number of samples. In addition, the latent space can improve the sample efficiency of the original algorithm by 6.8$\times$ and can discover hardware designs that are up to 5% more efficient than the optimal design searched directly in the high-dimensional input space.**

I. Introduction

Hardware design space exploration is a challenging problem due to its intractable search space with a large number of different design parameters, many of which cannot be made independently as they have intricate interactions with each other. The search space grows exponentially in size with the number of hardware configurations. In addition, a majority of the hardware design decisions are discrete, ranging from different dataflow choices to different buffer sizes across the memory hierarchy [1], [2]. Collectively, the nature of the design space exploration problem results in a non-linear and non-continuous performance function that relies heavily on the expert knowledge of designers.

Existing works approach the hardware design problem from its high-dimensional and discrete search space [3]–[11]. Typically, designers use heuristics-driven approaches to prune the large space [1], [3]. More recently, motivated by the great success of machine learning (ML) algorithms in performing difficult tasks such as image recognition [12]–[14], natural language processing [15], [16], and system optimization [17], [18], ML-based algorithms have also been used to search for optimal solutions in the discrete hardware design space [4]–[11]. However, these approaches focus on the development of intelligent search algorithms, with little attention to the structure of the search space.

In fact, the performance of a search algorithm heavily depends on the smoothness of the search landscape. In a continuous search space where points close to each other are also semantically similar, any continuous optimization algorithm can quickly move towards the optimal regions of the search space [19]. In addition, in the context of hardware design space exploration, the search space should also be reconstructible so that we can map the searched optimal design points back to the original hardware design space and generate valid configurations. Therefore, constructing a continuous and reconstructible search space of hardware design is key to reducing the dimensionality and improving the search efficiency of hardware design space exploration.

Recent advances in representation learning [19] have demonstrated that variational autoencoders (VAEs) are capable of learning a latent space and reconstructing the points in the latent space to the input space [20]. Specifically, VAEs compress the input information into a continuous latent distribution (encoding) and reconstruct it as accurately as possible (decoding). Different from vanilla autoencoders, the latent space of VAEs is regularized during training so that VAEs can be used in a generative process that produces new data by decoding points sampled from the latent space. Leveraging the generative property of VAEs, researchers have adopted VAEs in image synthesis [21], [22], chemical design [23], robot motion planing [24], neural architecture search [25], and other generative tasks.

In this paper, we investigate the feasibility of using VAEs for learning a meaningful reconstructible representation for hardware designs. The potential benefits of such representations are twofold. First, VAEs can reduce the dimensionality of the hardware design space as the encoded representation space typically has a lower dimension compared to the input space, making the overall search problem simpler. Second, the latent space features are normal distribution regularized and its performance surface is continuous, making it easier for optimization algorithms like gradient-based approaches [26] to navigate the search space.

Specifically, we present VAESA, a VAE-based design space exploration (DSE) framework for spatial accelerators. Unlike prior works that explore the original design space, VAESA learns a low-dimensional, continuous latent space representation that can be searched by any continuous optimization

*Work done when author was at UC Berkeley method. The searched high-quality solutions can also be reconstructed back to the original space to produce valid hardware configurations. To demonstrate the generality of the encoded representation, we apply two popular search algorithms, Bayesian Optimization (BO) [27] and Gradient Descent (GD), on the encoded design space and evaluate the searched hardware designs generated from the decoder. Our results show that the learned latent space significantly improve both the performance and sample efficiency for hardware design space exploration.

In summary, this paper makes the following contributions:

1) We propose VAESA, a novel VAE-based approach that learns a continuous and reconstructible latent space to compactly represent the high-dimension, discrete hardware design space.
2) We show that our approach is able to learn a well-structured latent search space through rigorous ablation studies.
3) We demonstrate that the learned search space significantly improve performance and sample efficiency by up to 5% and 6.8×, respectively, using two different search strategies.

II. Motivation and Background

This section discusses the challenges associated with hardware design space exploration, the state-of-the-art DSE approaches, and background on variational autoencoders (VAEs).

A. Hardware Design Space

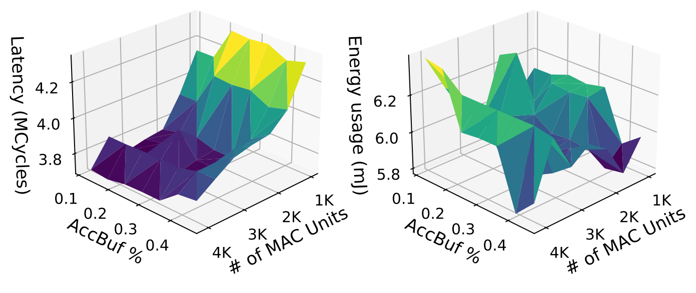
Hardware designs have evolved in both complexity and heterogeneity. The ever-growing and complex hardware exposes many intervening knobs to tune in its design. This work aims to optimize neural network accelerator designs described using a range of parameters, e.g. number of processing elements (PEs), number of multiply-and-accumulate (MAC) units per PE, and capacity of different buffers across the memory hierarchy. Many of these parameters have a large range of discrete values, resulting in a design space size on the order of 10^{17} with six different hardware parameters. Furthermore, the performance surface of the hardware design space is highly non-linear and non-convex, as demonstrated by the irregular shapes of the latency and energy landscapes, shown in Figure 1a and 1b. These properties make it challenging to quickly find optimal hardware configurations in the large and irregular hardware design space.

B. DSE Approaches

Design space exploration for hardware has been actively studied with a range of optimization techniques, shown in Table I. While these approaches significantly prune the hardware design space, they focus on developing effective search strategies to navigate the original high-dimensional design space itself.

Search Method	Original Input Space	Learned Search Space
Heuristics-Driven:	Interstellar [3]	N/A
Black-Box Optimization:	Bayesian Optimization [4] Apollo: P3BO [5] NAAS: Evolutionary [28]	**VAESA + BO**
Gradient-Based Optimization:	EDD [10] DiffTune [29]	**VAESA + GD**

TABLE I: Hardware design space exploration approaches. Prior works focus on developing intelligent search heuristics and algorithms while VAESA aims to learn a continuous and reconstructible latent search space to improve the search efficiency.



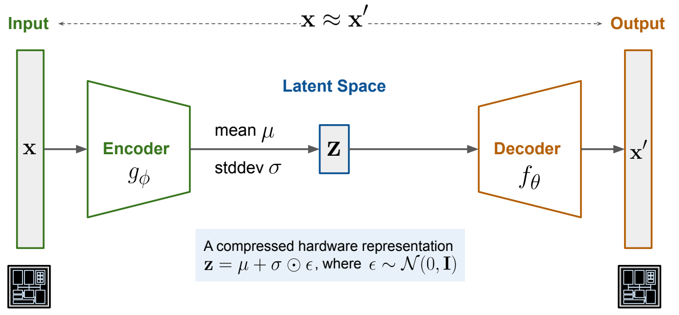
(a) Latency (b) Energy

Fig. 1: The irregular landscapes of the latency and energy functions across a small slice of the design space for ResNet50. Accumulation buffer size is represented as a percentage of the total buffer capacity (2.7 MB). All other hardware parameters are held constant.

1) Heuristics-Driven Approaches: Common DSE practice leverages heuristics of designers to prune the search space, followed by a random or brute-force search. However, the performance of this approach heavily relies on the insights of the designers, and the heuristically pruned space can limit the discovery of new optimal design points. For instance, Interstellar [3] confines its search to specific dataflows and adjacent memory buffer ratios that are known to be efficient.

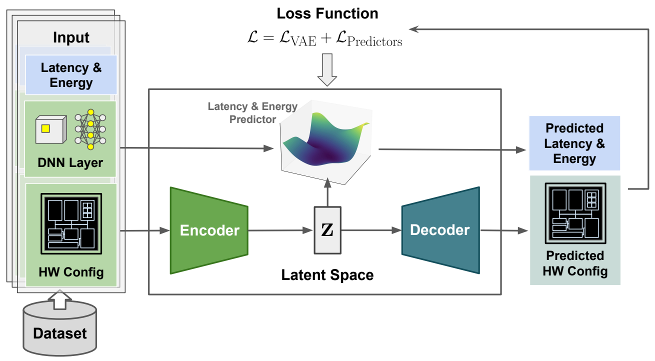
2) Reinforcement Learning (RL): RL obtains observations from the environment that the agent continually interacts with in addition to the reward after sampling each configuration. RL can learn a policy that maximizes the expected reward for a sequence of decisions. Recent works illustrates how RL co-optimizes the hardware design and the DNNs by continually updating the policy based on the reward feedback [6]–[8].

3) Black-box Optimization: Black-box optimization is a class of algorithms designed for optimizing tasks with unknown objective functions but achievable reward feedback given different inputs. For example, Bayesian optimization (BO) iteratively updates a statistical model to approximate the unknown objective function and uses an acquisition function to decide which input to sample next. Recent work [4] shows that BO achieves better performance more sample-efficiently compared to random search and algorithms used in TVM (XGBoost and TreeGRU) [30], [31].



Input
$\mathbf{x} \approx \mathbf{x}'$
Output
$\mathbf{x}$
Encoder
g_ϕ
Latent Space
mean μ
stddev σ
$\mathbf{z}$
Decoder
f_θ
$\mathbf{x}'$
A compressed hardware representation
$\mathbf{z} = \mu + \sigma \odot \epsilon$, where $\epsilon \sim \mathcal{N}(0, \mathbf{I})$

Fig. 2: Example variational autoencoder (VAE) model to learn a low dimensional representation $\mathbf{z}$ for hardware design. g_ϕ is the encoder function that takes input design parameters $\mathbf{x}$ and outputs a mean μ and a standard deviation σ for sampling the compressed latent feature $\mathbf{z}$. The sampled latent feature $\mathbf{z}$ is then fed to the decoder f_θ to reconstruct $\mathbf{x}'$. The training of VAE aims to minimize the difference between the input $\mathbf{x}$ and the output $\mathbf{x}'$.



Loss Function
$\mathcal{L} = \mathcal{L}_{\text{VAE}} + \mathcal{L}_{\text{Predictors}}$
Input
Latency & Energy
DNN Layer
HW Config
Dataset
Latency & Energy Predictor
Encoder
z
Decoder
Latent Space
Predicted Latency & Energy
Predicted HW Config

Fig. 3: The VAESA training pipeline to learn the latent space of hardware design. The dataset includes the latency and energy costs of DNN layers running on different DNN accelerator architectures. The encoder encodes the hardware configurations into the latent space, while the decoder reconstructs the hardware configurations from the latent space representation. Latency and energy predictors are trained together with the VAE pipeline to add semantics to the latent space construction. The overall loss function used in the VAE training consists of both the VAE loss and the MSE losses of the latency and energy predictors.

4) Gradient-based Optimization: Gradient-based approaches are commonly used for optimizing sophisticated objective functions that are differentiable, where the search directions are determined by the gradient of the function at the current point. Recent works demonstrate differentiable surrogate models can be constructed or trained to represent the performance of hardware designs [11], [29].

C. Variational Autoencoder

An autoencoder is a type of ML algorithm designed to learn a compressed representation of input data and the corresponding encoding and decoding functions [32], [33] . As shown in Figure 2, it is a feed-forward model that predicts output $\mathbf{x}'$ from input $\mathbf{x}$ through a bottleneck layer and the encoder g_ϕ and decoder f_θ are trained to minimize the difference between $\mathbf{x}$ and $\mathbf{x}'$, where $\mathbf{x}' = f_\theta(g_\phi(\mathbf{x}))$. $\mathbf{z}$ is a representation of $\mathbf{x}$ in the latent space, usually with reduced dimensionality. Auto-encoders are used in multiple domains including image compression, image generation [21], [22], and representation learning [23], [25], [34].

Different from an autoencoder, a variational autoencoder [20] (VAE) maps the input to a distribution instead of a fixed vector. Figure 2 demonstrates how a VAE is implemented in practice. Assuming that the data distribution of the latent space is a multivariate Gaussian, the encoder network first predicts the means μ and the standard deviations σ of the distribution from input $\mathbf{x}$. A latent point is sampled from the multivariate Gaussian distribution and decoded to the output during training. The output feature is then used to compare to the input feature to calculate the reconstruction loss. In addition, the predicted multivariate Gaussian distribution is regularized by the Kullback-Leibler (KL) divergence [35], which measures the difference between the learned distribution and the standard normal distribution and is minimized during training. The overall loss function can be writen as:

$$L_{\text{VAE}} = L_{\text{recon}} + \alpha L_{\text{kld}} \tag{1}$$

where the L_{recon} is the mean square error (MSE) between $\mathbf{x}$ and $\mathbf{x}'$, and L_{kld} is the KL divergence loss weighted by the α coefficient. VAEs turn the encoded point into a distribution and use the KL divergence measure to improve the continuity of the latent space, which is a critical property that can be exploited by many search algorithms. Therefore, in this work, we integrate VAE into existing hardware DSE frameworks to complement the search algorithms with an easy-to-navigate search space.

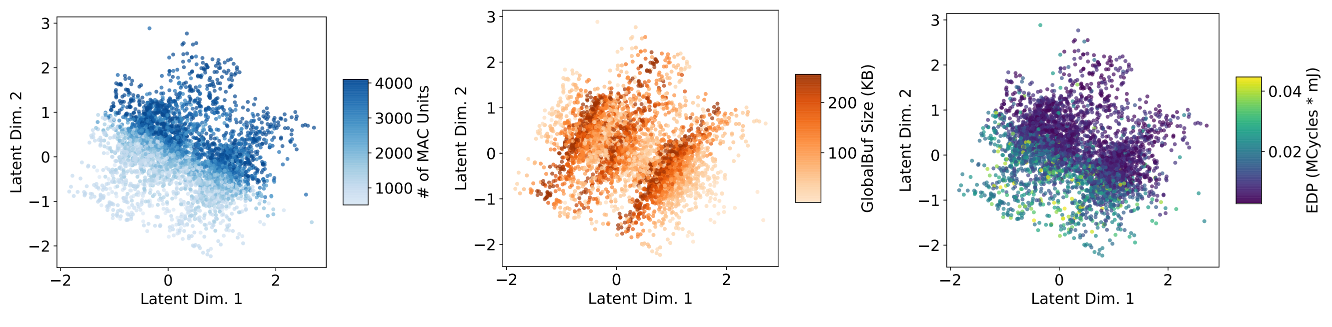
III. VAESA Framework

To improve the hardware DSE efficiency and leverage existing hardware design data, we introduce VAESA, an end-to-end DSE framework that enables the DSE to run on a low-dimensional and continuous latent space where optimized latent points can be decoded to realistic hardware designs.

A. Overview

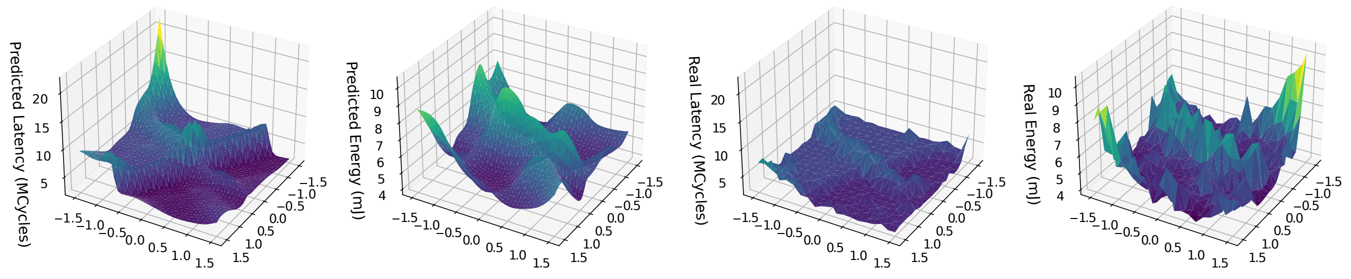
The VAESA framework takes problem specifications and existing hardware designs as input, performs design space exploration over the trained latent space, then outputs optimized hardware designs and the corresponding mappings to boost the overall performance and efficiency of the system. This work specifically targets the hardware DSE towards DNNs due to their popularity.

The VAESA framework decomposes the hardware design space exploration process into two separate steps. The first step (discussed in Section III-B) is a VAE training pipeline



(a) Number of MAC units (b) Global buffer size (c) Energy-delay product

Fig. 4: Visualization of training data (about 5000 randomly selected points) after encoding to a 2-dimensional latent space.



(a) Predicted performance of decoded accelerator (b) Real performance of decoded accelerator

Fig. 5: Visualization of the two predictors and how they compare to real ResNet-50 performance. *Left:* latency and energy values, predicted from points across the latent space. *Right:* decoded and evaluated latency and energy values of points from the same area of the latent space.

which learns to construct a continuous, low-dimensional latent space, where each point of the latent space can be reconstructed through the decoder. The second step (discussed in Section III-C) applies different search algorithms onto the latent space to efficiently find optimal and reconstructible design points. VAESA utilizes four key components for hardware accelerator DSE:

1) A VAE training pipeline to learn the compressed latent search space,
2) Hardware design space exploration algorithms to perform search over the latent space,
3) A scheduler to efficiently map the execution of the task to hardware
4) An evaluator to measure the quality of hardware and mapping

To evaluate the performance of different DNNs on a wide range of DNN hardware, we harness CoSA, a constrained optimization-based scheduler [36], to automatically generate high-performance mappings and Timeloop, an accurate latency and energy simulator [1], to estimate the performance of different design points.

B. Latent Space Generation: VAE Training and Visualization

To obtain a new latent space for hardware DSE, we first need to design and train a VAE model. The VAE is the key component in VAESA but hasn't been studied by the prior hardware DSE work. In this section, we take a deep dive into our VAE model design, training, as well as visualization of the latent space.

1) VAE Model Design: The VAE model, as shown in Figure 3, takes a real input hardware configuration as input, encodes it to a latent distribution, and decodes from a sample drawn from this distribution to an output. Training of VAE minimizes the difference between the predicted output and input as well as regularizes the latent space data distribution. The VAE model used in VAESA is a feed-forward multilayer perceptron (MLP) network with symmetric encoder and decoder designs. It uses leaky ReLU in between the MLP layers to introduce non-linearity in the model.

2) Performance Predictors: A drawback to the existing vanilla VAE design is that its latent space has no semantic meaning. Shown in Figure 3, we augment the VAE design with two performance predictors in order to introduce structure to the latent space. Conditional on different input layer features, the predictors are used to estimate the latency and energy of the latent design points. Such a design is motivated by the prior VAE work [23] that demonstrates the structuring effect of latent space property predictors.

In order for the predictors to impact the VAE behavior, we need to train the VAE and the predictor models together in an end-to-end manner. The overall loss function of the VAE

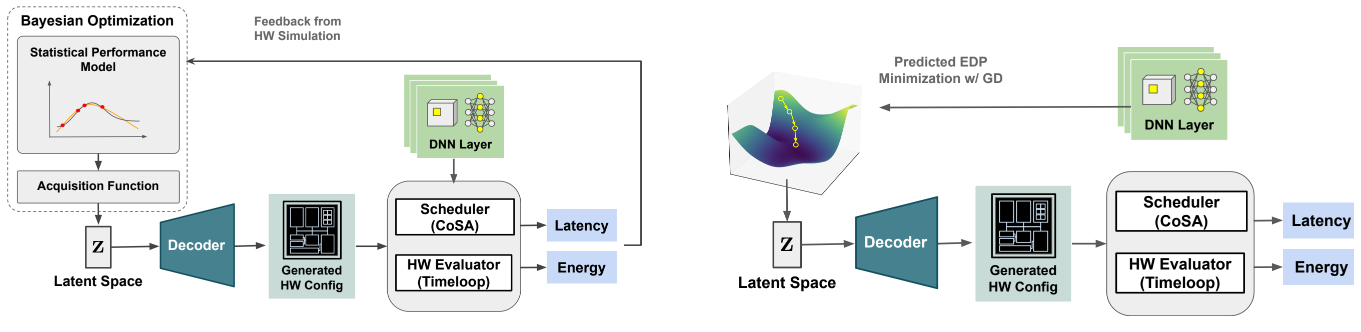


Fig. 6: Design space exploration with VAESA. Once the latent representation and the decoder are trained, different search algorithms can be directly applied to the latent space, where the searched optimal points in the latent space will be decoded back to the hardware configurations. We demonstrate two search algorithms: Bayesian Optimization (BO) and Gradient Descent (GD), as illustrated in Figure 6a and Figure 6b, respectively.

training pipeline thus becomes:

$$L = L_{\text{VAE}} + L_{\text{predictors}} \tag{2}$$

where L_{VAE} is the VAE loss as in Equation 1. $L_{\text{predictors}}$ is the sum of the MSE losses of the latency and energy predictors.

Another usage of the performance predictors is to enable gradient-based optimization over the latent space. During inference, the differentiable MLP-based performance predictor model with fixed model weights can be used as a proxy of the unknown real performance function over the latent space for gradient-based optimization. The accuracy of the predictor models is thus of critical importance to the gradient-based search algorithms. We will illustrate the use of predictor models by GD in Section III-C2.

3) Dataset Construction and Training: Training VAE requires a large amount of data. In VAESA, we collect a dataset of 500K samples to train the VAE with the per layer performance prediction. Each data sample consists of hardware design features, DNN layer features, and the corresponding energy and latency labels. We gather these data points by running grid search and random search over the original design space. We only add valid design points to the training dataset because we want the VAE to learn the distribution of valid designs and generate points that resemble realistic designs. As we explore more and more hardware designs during DSE, we can expand the dataset and retrain or fine tune the VAE and predictor models.

Normalization is a critical technique for training with the hardware and DNN features as the scales of their values can vary massively. We need to bring the magnitudes of different features down to a similar range to speed up the training process. In VAESA, we perform min-max normalization for all hardware design features and DNN layer features. We then train the model end-to-end until the loss function converges. Once we finish training, the VAE can predict hardware configurations that resemble realistic hardware designs. We also obtain a compact and continuous latent space where samples can be decoded to the original design space. Retraining of VAE is not required for new workloads or new hardware designs in our flow, but is needed if we change the number of original hardware features as we currently represent hardware configurations as fixed-length vectors. Adding a sequence or graph-based feature embedding to VAE can address this limitation.

4) Visualization of the Latent Space: A natural question arises from our system: how does the trained VAE map discrete design points to the continuous latent space? To visualize the latent design space, we train a VAE with a 2-dimensional latent space. Figure 4a and 4b demonstrate that even with a two-dimensional latent space, the VAE successfully encodes the structure it finds in the training data, as input data points are clearly grouped by feature values when they are encoded in the latent space.

Inspecting the encoder also reveals some of the relationships between architectural parameters and hardware performance metrics. The plots provide evidence that for the current workload, a large number of compute elements is necessary for a more efficient accelerator design, as the purple points (smaller energy-delay product) in Figure 4c tend to overlap with the dark blue points in Figure 4a. We can also see that some of the least efficient designs in the training data, drawn in yellow, have both a small number of PEs (light blue) and small global buffer sizes (light orange).

5) Visualization of Predictor Performance: We can also visualize our trained MLP-based predictors using the same 2-D latent space as Section III-B4. In Figure 5, plotting the predicted latency and energy usage of designs decoded from across the latent space against the real latency and energy usage of these designs allows us to visually inspect how similar these two surfaces are. In areas where there is plenty of training data (within a radius of about 1.5 units from the origin), the predictors for both latency and energy accurately represent the performance surface. Outside of this region, the predictor shows some potential to extrapolate beyond our training data at points such as (latent dim. 1, latent dim. 2) = (1.5, -1.5), where latency and energy are accurately predicted to have higher values than at neighboring points. However, the predictor fails at other points such as (-1.5, -1.5), where latency is correctly predicted to increase relative to neighboring points, but the value itself is incorrect by a multiple of 5. Despite the fact that there is a relatively high degree of quantifiable error in the predictor, this visualization allows us to see the qualitative accuracy of the latency and energy predictors, which match closely with the contours in the real performance surface. This indicates that gradient-based optimization in the latent space will be effective in quickly identifying relatively well-performing designs within a few samples. Additionally, the presence of several different areas of similarly low latency and energy usage in the predictors suggests that if a more exhaustive search is carried out, usage of the performance predictors will not hamper exploration of unique but similarly well-performing designs.

C. Latent Design Space Exploration: BO and GD

Once the latent space and the real-world design mapping is established through training, DSE can then be performed with respect to the latent design space. We implement two representative search strategies under two different categories of algorithms for DSE over the latent space: the black-box Bayesian optimization (BO) and the predictor-based gradient descent (GD).

1) Black-box BO: The first search strategy we employ is Bayesian Optimization (BO). BO assumes the objective function is a black box or unknown and maintains its own statistical model for the objective function using Gaussian processes. It updates its model every time it receives a new pair of latent feature to performance label sample. Based on the statistical model, the acquisition function makes the exploration-exploitation trade-off to determine the next sample. Since BO models the objective function using Gaussian processes, it is hard for the model to predict the performance of discrete inputs. The continuous and KL-regularized latent space intuitively should be more amenable for BO compared to the original design space.

Figure 6a illustrates how BO performs the latent space DSE in VAESA: BO first samples a design from the latent space, then invokes the decoder to map the latent design to the original design space. With the reconstructed hardware configuration, BO runs the scheduling and evaluation tools to obtain performance feedback. BO updates its statistical performance model and the acquisition function (computed using the statistical model) with the new performance feedback for the sampled latent design. In the next iteration of DSE, BO selects the latent space design that optimizes the updated acquisition function value. This search process continues iteratively to perform exploration and exploitation on the latent search space.

2) Predictor-based GD: GD, on the other hand, requires a known and differentiable objective function to perform DSE. The trained predictors in our VAESA model suit this need. GD is an iterative method to approximate the optimal design by taking gradient steps. The gradient in GD is computed based on a subset of randomly drawn samples from the training set. To achieve the global optimum, it relies on the objective function to

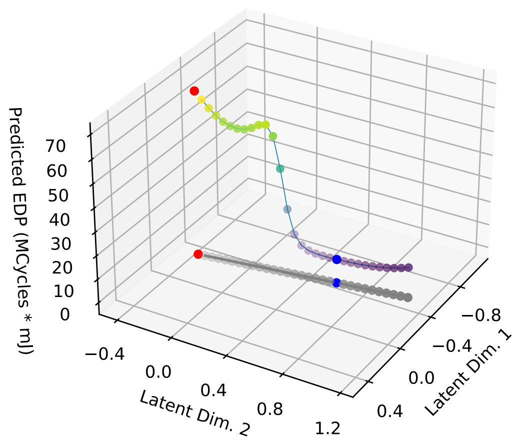
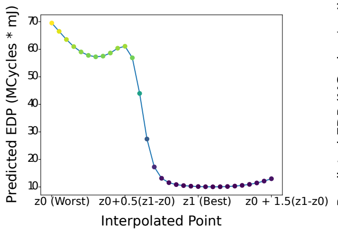
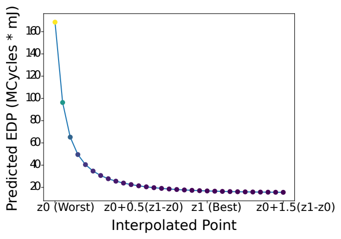


Fig. 7: Interpolation in the latent space between the worst (red) and best (blue) points in the training data.



(a) 2-dimensional latent space. L2 distance between worst and best points: 0.96



(b) 4-dimensional latent space. L2 distance between worst and best points: 2.58

Fig. 8: Contour of the predicted surface "above" (projected onto) the worst point-best point axis.

be very smooth. Otherwise, GD may get stuck at local minima. In VAESA, the latent search space can potentially also address the non-smooth objective proxy issue by rearranging design points in latent space based on their performance.

In Figure 6b, we show the VAE + GD flow that optimizes the latent space design points towards a performance proxy. Unlike the VAE + BO flow, the GD-based DSE flow performs iterative gradient updates on the latent space performance predictor, and only invokes the scheduler and simulator after an optimized latent space design is found with respect to the predictor performance.

As discussed in Section III-C, continuity and smoothness are two important indicators to show how good a search space is for both BO and gradient-based search algorithms. In this section, we thus visualize the latent space to qualitatively evaluate its smoothness and usefulness for accelerator design space exploration.

For gradient-based optimization methods to be effective, we require that there are not too many local minima where optimization can stall. With a 2-D latent space, the performance surface can be visually inspected for local minima. However, this can not easily be done for latent spaces of higher dimensionality. In higher dimensions, we can instead interpolate between the worst and best points to predict whether a gradient-based search would have successfully arrived at a good accelerator design even if it started at a poor one. Figure 7 shows how our visualization works. First, we encode the worst point (red, call it $z0$) and best point (blue, call it $z1$) for the predicted workload in our training data. Then, for each point $z0 + \frac{i}{N}(z1 - z0)$, where i ranges from 1 to N, between $z0$ and $z1$, we predict the energy-delay product $pEDP$ of the corresponding output accelerator. We also include some points $z0 + \frac{j}{N}(z1 - z0)$, where $j > N$, in order to understand whether a gradient-based search would have stopped near the best-known point, or continued elsewhere. Finally, we project each $pEDP$ onto its corresponding point on worst-best point axis, and plot this in 2 dimensions. In Figure 7, the worst-best point axis is the line of gray dots parallel to the x-y plane; in Figure 8, it corresponds to each x-axis.

We carried out this study for VAESA trained with 2- and 4-dimensional latent spaces. The result of the interpolation shows that for both variants, the contours of the predictor surface are conducive to an improvement in real performance of the decoded architecture, as the gradient of the predicted performance surface tends to be negative in the direction of the worst-best point axis. However, we also see that with a 2-dimensional latent space, there is a local minimum between the worst and best known points, where gradient descent, if carried out starting from the worst known point, could become trapped. This indicates that a 4-dimensional latent space's greater representational capacity may create a more monotonic performance surface where gradient descent can find better solutions. We further investigate the potential costs and benefits of changing latent space dimensionality in Section IV-B1.

IV. Experiments

In order to evaluate the effectiveness of VAESA, we first justify the design of the framework through the following experiments:

- **VAE hyperparameter tuning**. In this experiment, we vary design points specific to VAE, namely KL divergence vs reconstruction loss weighting and latent space dimensionality, and explain the selection of each hyperparameter for the subsequent evaluation of VAESA (Section IV-B1).

Then, we examine the optimization benefit of VAESA in hardware accelerator design space exploration with the following use cases:

- **Bayesian optimization**. We test the effectiveness of VAESA as tool to augment current popular iterative optimization techniques. We can use the latent space as a continuous and compressed space that increases the efficiency of Bayesian Optimization on the accelerator design space (Section IV-C).
- **Predictor-based gradient descent**. We explore the potential of VAESA to optimize for new neural network workloads within a very small number of samples (Section IV-D).

Parameter	Max	# Possible Values
No. of PEs	64	5
No. of MAC units	4096	64
Accum. buffer size	96 KB	128
Weight buffer size	8 MB	32768
Input buffer size	256 KB	2048
Global buffer size	256 KB	131072

TABLE II: Summary of the design space.

Target Workload	# of Unique Layers
AlexNet [12]	8
ResNet-50 [13]	24
ResNeXt-50 [37]	25
DeepBench [38] (OCR and Face Recognition)	9

TABLE III: Summary of DNN workloads whose layers are used to train the VAE (Section III-B3). We optimize the performance of these models in the VAE-BO study in Section IV-C.

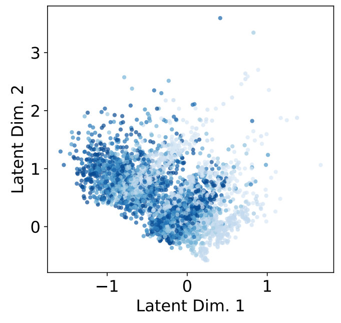
Layer #	Dimensions							
1.	1	1	1	1	2208	1000	1	1
2.	1	1	1	1	512	256	1	1
3.	1	1	28	28	512	512	1	1
4.	3	3	14	14	192	48	1	1
5.	3	3	14	14	512	512	1	1
6.	3	3	28	28	192	48	1	1
7.	3	3	28	28	512	512	1	1
8.	3	3	350	80	64	64	1	1
9.	3	3	56	56	192	48	1	1
10.	3	3	56	56	256	256	1	1
11.	3	3	7	7	192	48	1	1
12.	5	5	700	161	1	64	2	2

TABLE IV: Twelve different DNN layer workloads used in the VAE-GD study in Section IV-D. Format (8-columns): weight width, weight height, output width, output height, input channel, output channel, stride width and stride height.

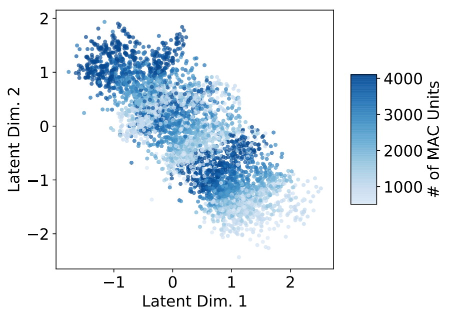
A. Experimental Setup

1) Hardware Design Space: We apply VAESA to explore the hardware design space of a Simba [39]-like DNN accelerator. We considered six design parameters to search (given in Table II) including the number of PEs/MACs and the size of different memory buffers. Various design parameters have a different number of possible discrete values ranging from 5 to 131K. The total design space size of this architecture is 3.6×10^{17}.

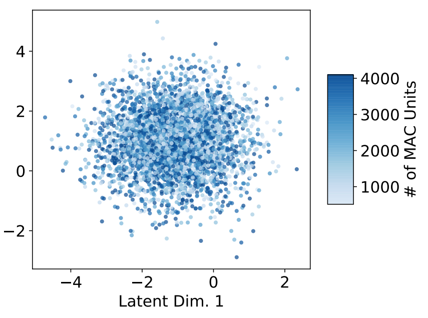
2) Workloads and Evaluation Metrics: Four of the main workloads evaluated are described in Table III. Three workloads (ResNet-50, ResNeXt-50-32x4d, and AlexNet) are convolutional neural networks used for computer vision applications, while the last, DeepBench, is a collection of different layer types purposed for benchmarking [38]. Layers from these four neural networks are used for training, as described in Section III-B3, and in the Bayesian optimization study in Section IV-C. An unseen set of 12 convolutional and fully connected layers (Table IV), selected for diversity from networks other than these four, is used in the gradient descent



(a) $\alpha = 0$



(b) $\alpha = 0.0001$



(c) $\alpha = 0.01$

Fig. 9: Encoders trained with different values of α. For $\alpha = 0$, encoder does not learn a continuous representation for the training data, while $\alpha = 0.01$ generates an almost normal distribution. We select an intermediate value of $\alpha = 0.0001$, which generates a distribution that is both continuous and reconstructible.

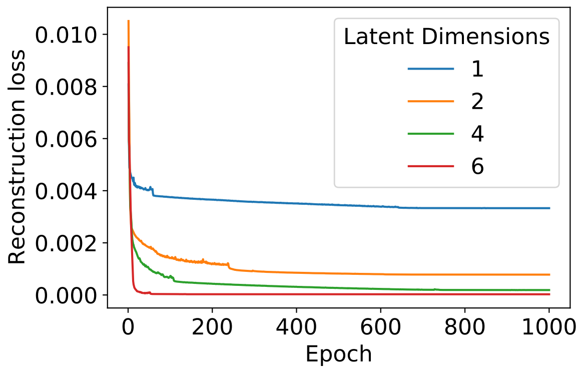


Fig. 10: Value of the autoencoder reconstruction loss term during training for different latent space dimensions. For this problem, there are diminishing returns beyond 4 dimensional latent space.

study in Section IV-D. We use a batch size of 1 for evaluating the workloads.

In our experiments, the objective is to minimize the energy-delay product (EDP, i.e. latency $\times$ energy) of deep neural network workloads. Although our flow can optimize the latency and energy separately, we select EDP as the hardware performance metric because it allows us to investigate Pareto-optimal design points that trade off latency and energy, creating a difficult but important search problem.

To evaluate search performance, we use the following metrics:

- Best EDP found, among all searched points. This evaluates the ability of each search method to explore the design space and identify high-performance points that other methods may not achieve.
- Sample efficiency, which we estimate by the rate at which each search method finds an accelerator within 3% of best known EDP for a workload. This evaluates the ability of each search method to exploit existing knowledge about accelerator performance and quickly identify a near-optimal solution to the search problem.

Given the hardware DSE is a stochastic process, the DSE performance is affected by randomness. Therefore, in our study, we run each experiment three times with different random seeds and show the average performance and the standard deviation of the runs in the figures.

3) Scheduler and Simulation Tool: We use optimization-based scheduler CoSA [36] to solve for mappings that minimize the overall data transfer size and maximize the buffer and PE utilization. Given a problem and an architecture specification, CoSA returns a high-performance mapping in one shot, significantly reducing the mapping search time compared to iterative schedulers.

Once a mapping for the architecture of interest is generated, we pass it, the architecture specification, and the target problem to a fast and accurate analytical performance model from Timeloop [1]. Timeloop statically analyzes the data access patterns of the input tensor-algebra workloads and reports the latency, energy, and area estimation. It derives the operation counts from the data access patterns and multiplies them with the energy per operation to estimate its total energy. In our experiments, we use the energy numbers gathered from the 40nm technology node.

4) Normalization of Features and Labels: To facilitate the training of the VAE and the performance predictors, we use two normalization schemes to scale the feature and label values to the desired range. First, we take the logarithm of the values, as they change by orders of magnitude. Then, we perform min-max normalization of the values by looking for the minimum and maximum log values in the dataset and dividing the values by the difference between these. After a series of normalization operations, the range of the features and labels should be close to [0,1).

B. VAE Hyperparameter Tuning

1) Weighting KL Divergence: Kullback-Leibler (KL) divergence is one of the loss terms minimized during training of VAESA. During training, we backpropagate on the KL divergence between the normal distribution of training data and the standard normal, which regularizes the encoder-decoder

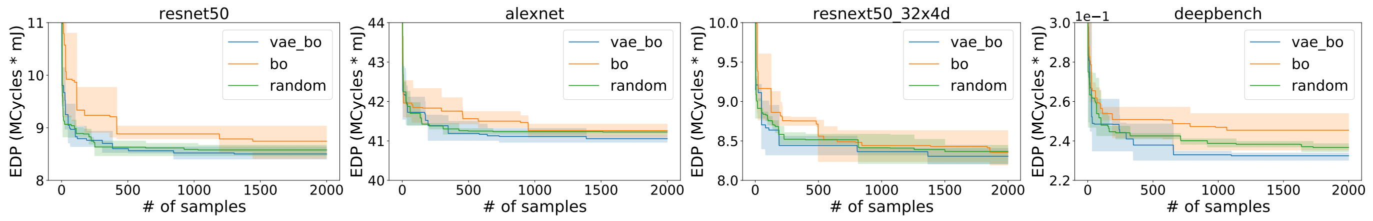


Fig. 11: DSE performance comparison of BO running with the VAE-generated latent space (*vae_bo*), BO, and random search. Each line represents the mean EDP across three random seeds and the shaded region shows the standard deviation across the three runs. *vae_bo* converges to an optimal design point fastest for all four DNNs.

	ResNet-50		AlexNet		ResNeXt-50		DeepBench	
DSE Method	Search Performance (SP)	Sample Efficiency (SE)	SP	SE	SP	SE	SP	SE
Random Search (*random*)	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Bayesian Optimization (*bo*)	0.98	0.61	1.00	0.31	1.00	0.94	0.96	1.00
VAESA + Bayesian Optimization (*vae_bo*)	**1.01**	**4.17**	1.00	**2.00**	**1.01**	**1.27**	**1.01**	**4.46**

TABLE V: Search performance and sample efficiency of DSE methods. *Search performance* is measured by the EDP of the best design point achieved over 2000 samples, relative to the average random search result. *Sample efficiency* is estimated by measuring the rate at which each search method finds an accelerator within 3% of best known EDP for a workload.

network by encouraging the encoder to cluster points closer to the origin. Reconstruction loss, on the other hand, tends to encourage distinct data to be encoded further apart so that they can be reconstructed more accurately [40]. During training, we can weight these losses and their backpropagated gradients, so that they are prioritized accordingly.

There are several prior works on determining how to weight the two VAE loss terms, including some that introduce schemes for scheduling these values [41], [42]. However, it is non-obvious how changing these weights affects VAESA, due to the introduction of predictor networks that are trained in tandem with the VAE. In order to study these effects, we train a 2-dimensional latent space VAESA with three different options for the weight α on the second term in Equation 1.

Visualization of the encoded training data in Figure 9 illuminates some key changes that occur as α changes. The smallest selected value, 0, removes the variational component of VAESA's autoencoder and creates a clear gradient in the encoding of accelerator configurations with different number of MAC units. However, some points in the training data are encoded far apart from one another, as there is no KL divergence term to drive latent space encodings towards the origin. This creates regions of high uncertainty, leading to inaccurate performance prediction. With $\alpha = 0.0001$, accelerators with different numbers of MAC units are encoded to a distinctly patterned, but mostly continuous cloud of points. An α value of 0.01 is too large, weighting the KL divergence loss term so heavily that the distribution of encoded data appears completely random (as it approaches the standard normal). Quantitatively, VAESA trained with $\alpha = 0.0001$ also reconstructs input data the most accurately of these three options. Therefore, we use $\alpha = 0.0001$ for most of our studies.

2) Latent Space Dimensionality: VAEs learn a lower-dimensional representation of input data in the latent space. However, there are several tradeoffs involved when selecting the dimensionality of the latent space. In Section III-B4 we used 2 dimensions to interpretably visualize the latent space, but Figure 10 shows that increasing latent space dimensionality increases reconstruction accuracy (the first term in Equation 1). We VAESA with a latent space dimensionality of 4 as it reduces the dimensionality of the design space without compromising a significant amount of accuracy.

C. Bayesian Optimization with VAE

For the first experiment, we utilize VAESA to augment Bayesian optimization, an iterative black-box search algorithm. We evaluate VAESA with Bayesian optimization (*vae_bo*) in the latent space in comparison to the same implementation of Bayesian optimization (*bo*) in the original hardware input feature space. Random search (*random*) is used as another baseline. In Figure 11, we show that *vae_bo* finds the optimal hardware configuration that minimizes EDP within 2000 samples for all target DNN workloads. The curves in Figure 11 also show that *vae_bo* in general converges faster than the two other baseline approaches.

Table V summarizes the results of combining the latent space of VAESA with Bayesian optimization. We measure both the EDP of the best architecture found by each method over 2000 samples, relative to the result found by random search, as well as the number of samples taken to reach an EDP within 3% of the known best architecture, to estimate sample efficiency. Sample efficiency is important in cases where an accelerator must quickly be tuned to a new neural network workload, such as when accelerator and neural network architecture are co-designed. The use of VAESA improves the search performance by up to 5% and the sample efficiency of hardware design space exploration by up to 6.8× versus Bayesian optimization in the original design space. On average, best performance

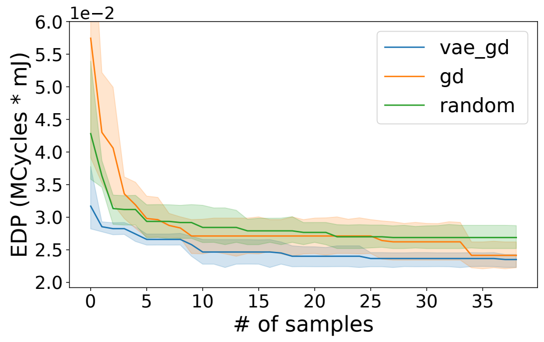


Fig. 12: Average EDP improvement of *vae_gd* compared to *gd* and *random* over 12 test layers. Experiments repeated for 5 random seeds.

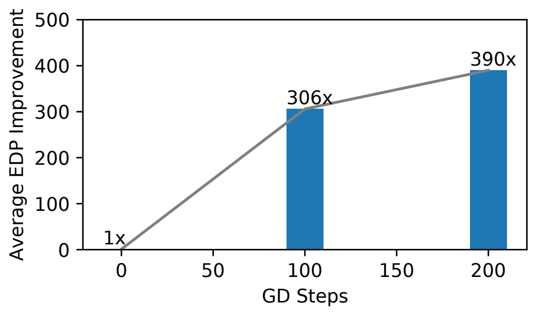


Fig. 13: Average GD improvement over different number of gradient update steps over 200 randomly generated sample points.

increases by 2% and sample efficiency increases by 4.8× versus Bayesian optimization. *vae_bo* also improves sample efficiency by 3.0× on average versus random search on the original search space. This demonstrates that the continuous, compressed latent space in VAESA increases the rate at which black-box optimization finds a good solution to the hardware search problem, while still allowing exploration of the design space over a large number of samples.

D. Gradient Descent with VAE

Because VAESA predicts the performance for an accelerator on an arbitrary convolutional or fully connected layer, we can attempt to optimize accelerator design for a certain layer or combination of layers by using gradient descent. With GD, we start from randomly selected points in the latent space and minimize predicted performance using the gradient of VAESA's predictors, then decode the selected latent space point. If the performance predictor is accurate, this method should be able to produce relatively good accelerator designs in just a few samples. Ideally, a user who wants to quickly optimize an accelerator for an arbitrary neural network design could predict performance for the full network by summing latency and energy predictions for multiple layers. Since neural network architecture search is not the objective of this work, we evaluate this method on individual layers and leave the exploration of neural network architectures to future work. These individual layers, listed in Table IV, are not part of the dataset used to train VAESA's performance predictors.

We first evaluate the effectiveness of GD in the latent space (*vae_gd*) as opposed to GD in the original discrete space (*gd*). For *gd*, we train a separate performance predictor taking input from the original design space and round the GD optimized input to valid discrete values. Figure 12 shows that on average, *vae_gd* consistently outperforms both *gd* and *random* for any small number (approx. $\leq$ 30) of samples. For example, it achieves 16% lower EDP in 10 samples than *random*. To remove the possible effects of confounding factors such as random selection of GD start points, we also examine the performance of decoded accelerator designs after 0, 100, and 200 steps of gradient descent. Figure 13 shows that *vae_gd* improves EDP of decoded accelerator designs by 306× after 100 steps and 390× after 200 steps relative to the corresponding randomly selected start points. This improvement explicitly demonstrates how VAESA allows users to improve accelerator performance before even beginning simulation.

V. Conclusion

In this work, we introduce a new VAE-based hardware design space exploration framework called VAESA where the search is performed on a continuous and reconstructible latent space. Through experimentation, visualization, and ablation studies, we demonstrate properties of the latent space and study how they vary with different hyperparameter settings. We devise and train a rigorous VAE model based on our observations and use the trained models to enhance two state-of-the-art algorithms: the black-box BO and the predictor-based GD. Experiments show that algorithms running over the latent space converge faster and are more likely to discover the optimal design within a fixed number of samples. Performing search over the learned latent space achieves up to 5% performance improvement and 6.8× better sample efficiency for BO and improves the performance of GD consistently.

Acknowledgements

This work was supported in part by the UC Berkeley ADEPT/SLICE Lab industrial sponsors and a Facebook Faculty Research Award.

References

[1] 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 International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2019.

[2] H. Kwon, P. Chatarasi, M. Pellauer, A. Parashar, V. Sarkar, and T. Krishna, "Understanding reuse, performance, and hardware cost of dnn dataflow: A data-centric approach," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2019.

[3] X. Yang, M. Gao, Q. Liu, J. Setter, J. Pu, A. Nayak, S. Bell, K. Cao, H. Ha, P. Raina, C. Kozyrakis, and M. Horowitz, "Interstellar: Using halide's scheduling language to analyze dnn accelerators," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operation Systems (ASPLOS)*, 2020.

[4] Z. Shi, C. Sakhuja, M. Hashemi, K. Swersky, and C. Lin, "Using bayesian optimization for hardware/software co-design of neural accelerators," in *Workshop on ML for Systems at the Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

[5] A. Yazdanbakhsh, C. Angermueller, B. Akin, Y. Zhou, A. Jones, M. Hashemi, K. Swersky, S. Chatterjee, R. Narayanaswami, and J. Laudon, "Apollo: Transferable architecture exploration," *arXiv preprint arXiv:2102.01723*, 2021.

[6] S.-C. Kao, G. Jeong, and T. Krishna, "ConfuciuX: Autonomous Hardware Resource Assignment for DNN Accelerators using Reinforcement Learning," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2020.

[7] W. Jiang, L. Yang, S. Dasgupta, J. Hu, and Y. Shi, "Standing on the shoulders of giants: Hardware and neural architecture co-search with hot start," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020.

[8] W. Jiang, L. Yang, E. H.-M. Sha, Q. Zhuge, S. Gu, S. Dasgupta, Y. Shi, and J. Hu, "Hardware/software co-exploration of neural architectures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2020.

[9] Y. Zhou, X. Dong, B. Akin, M. Tan, D. Peng, T. Meng, A. Yazdanbakhsh, D. Huang, R. Narayanaswami, and J. Laudon, "Rethinking co-design of neural architectures and hardware accelerators," *arXiv preprint arXiv:2102.08619*, 2021.

[10] 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," *arXiv preprint arXiv:2005.02563*, 2020.

[11] P. Achararit, M. A. Hanif, R. V. W. Putra, M. Shafique, and Y. Hara-Azumi, "Apnas: Accuracy-and-performance-aware neural architecture search for neural hardware accelerators," *IEEE Access*, vol. 8, 2020.

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet Classification with Deep Convolutional Neural Networks," in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2012.

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

[14] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[15] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[16] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, 2020.

[17] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proceedings of the ACM workshop on hot topics in networks*, 2016.

[18] A. Mirhoseini, H. Pham, Q. V. Le, B. Steiner, R. Larsen, Y. Zhou, N. Kumar, M. Norouzi, S. Bengio, and J. Dean, "Device placement optimization with reinforcement learning," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2017.

[19] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE transactions on pattern analysis and machine intelligence*, 2013.

[20] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *2nd International Conference on Learning Representations (ICLR)*, 2014.

[21] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner, "beta-vae: Learning basic visual concepts with a constrained variational framework," 2017.

[22] D. P. Kingma and M. Welling, "An introduction to variational autoencoders," *Foundations and Trends® in Machine Learning*, vol. 12, no. 4, p. 307–392, 2019. [Online]. Available: http://dx.doi.org/10.1561/2200000056

[23] R. Gómez-Bombarelli, J. N. Wei, D. Duvenaud, J. M. Hernández-Lobato, B. Sánchez-Lengeling, D. Sheberla, J. Aguilera-Iparraguirre, T. D. Hirzel, R. P. Adams, and A. Aspuru-Guzik, "Automatic chemical design using a data-driven continuous representation of molecules," *ACS central science*, vol. 4, no. 2, 2018.

[24] B. Ichter, J. Harrison, and M. Pavone, "Learning sampling distributions for robot motion planning," 2019.

[25] J. Li, Y. Liu, J. Liu, and W. Wang, "Neural architecture optimization with graph vae," *arXiv preprint arXiv:2006.10310*, 2020.

[26] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT*. Springer, 2010.

[27] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," 2012.

[28] Y. Lin, M. Yang, and S. Han, "Naas: Neural accelerator architecture search," in *Design Automation Conference (DAC)*, 2021.

[29] A. Renda, Y. Chen, C. Mendis, and M. Carbin, "Difftune: Optimizing cpu simulator parameters with learned differentiable surrogates," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2020.

[30] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, H. Shen, M. Cowan, L. Wang, Y. Hu, L. Ceze, C. Guestrin, and A. Krishnamurthy, "TVM: An Automated End-to-end Optimizing Compiler for Deep Learning," in *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2018.

[31] T. Chen, T. He, M. Benesty, V. Khotilovich, Y. Tang, H. Cho *et al.*, "xgboost: extreme gradient boosting," *R package version 0.4-2*, 2015.

[32] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *science*, 2006.

[33] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2008.

[34] A. Hottung, B. Bhandari, and K. Tierney, "Learning a latent search space for routing problems using variational autoencoders," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

[35] S. Kullback, *Information Theory and Statistics*. Courier Corporation, 1968.

[36] Q. Huang, A. Kalaiah, M. Kang, J. Demmel, G. Dinh, J. Wawrzynek, T. Norell, and Y. S. Shao, "CoSA: Scheduling by Constrained Optimization for Spatial Accelerators," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2021.

[37] S. Xie, R. Girshick, P. Dollar, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[38] S. Narang and G. Diamos, "Baidu deepbench," *GitHub Repository*, 2017. [Online]. Available: http://www.github.com/baidu-research/DeepBench

[39] Y. S. Shao, J. Cemons, R. Venkatesan, B. Zimmer *et al.*, "Simba: Scaling deep-learning inference with chiplet-based architecture," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019.

[40] A. Alemi, B. Poole, I. Fischer, J. Dillon, R. A. Saurous, and K. Murphy, "Fixing a broken ELBO," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. PMLR, 10–15 Jul 2018, pp. 159–168.

[41] B. Dai and D. Wipf, "Diagnosing and enhancing VAE models," in *International Conference on Learning Representations (ICLR)*, 2019.

[42] A. Asperti and M. Trentin, "Balancing reconstruction error and kullback-leibler divergence in variational autoencoders," 2020.