Skip the Benchmark: Generating System-Level High-Level Synthesis Data using Generative Machine Learning

Yuchao Liao
Electrical and Computer Engineering
University of Arizona
Tucson, Arizona, USA
yuchaoliao@arizona.edu

Tosiron Adegbija
Electrical and Computer Engineering
University of Arizona
Tucson, Arizona, USA
tosiron@arizona.edu

Roman Lysecky
Electrical and Computer Engineering
University of Arizona
Tucson, Arizona, USA
rlysecky@arizona.edu

Ravi Tandon
Electrical and Computer Engineering
University of Arizona
Tucson, Arizona, USA
tandonr@arizona.edu

ABSTRACT

High-Level Synthesis (HLS) Design Space Exploration (DSE) is a widely accepted approach for efficiently exploring Pareto-optimal and optimal hardware solutions during the HLS process. Several HLS benchmarks and datasets are available for the research community to evaluate their methodologies. Unfortunately, these resources are limited and may not be sufficient for complex, multi-component system-level explorations. Generating new data using existing HLS benchmarks can be cumbersome, given the expertise and time required to effectively generate data for different HLS designs and directives. As a result, synthetic data has been used in prior work to evaluate system-level HLS DSE. However, the fidelity of the synthetic data to real data is often unclear, leading to uncertainty about the quality of system-level HLS DSE. This paper proposes a novel approach, called *Vaegan*, that employs generative machine learning to generate synthetic data that is robust enough to support complex system-level HLS DSE experiments that would be unattainable with only the currently available data. We explore and adapt a Variational Autoencoder (VAE) and Generative Adversarial Network (GAN) for this task and evaluate our approach using state-of-the-art datasets and metrics. We compare our approach to prior works and show that Vaegan effectively generates synthetic HLS data that closely mirrors the ground truth's distribution.

CCS CONCEPTS

• **Hardware** → *Software tools for EDA*; • **Computing methodologies** → *Machine learning*.

KEYWORDS

High-Level Synthesis, Synthetic Data Generation, Variational Autoencoder, Generative Adversarial Network

GLSVLSI '24, June 12–14, 2024, Clearwater, FL, USA

ACM ISBN 979-8-4007-0605-9/24/06
https://doi.org/10.1145/3649476.3658738

ACM Reference Format:
Yuchao Liao, Tosiron Adegbija, Roman Lysecky, and Ravi Tandon. 2024. Skip the Benchmark: Generating System-Level High-Level Synthesis Data using Generative Machine Learning. In *Great Lakes Symposium on VLSI 2024 (GLSVLSI '24), June 12–14, 2024, Clearwater, FL, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3649476.3658738

1 INTRODUCTION

High-level synthesis (HLS) is a popular approach to designing, synthesizing, and optimizing hardware systems. HLS is often used to design embedded systems, such as medical devices, autonomous vehicles, and, more generally, the Internet of Things (IoT). Using existing HLS tools (like Vitis HLS), designers can develop application-specific embedded systems using high-level languages (e.g., C/C++) and map them to hardware register-transfer level (RTL) languages (e.g., Verilog, VHDL), thereby improving design productivity and reducing the design time/cost [16, 20]. HLS tools allow designers to select different directives such as loop unrolling factors, memory binding, function inline, target frequency, etc. Each modification of directives creates a different design configuration, leading to a larger design space.

HLS design space exploration (DSE) [20] aims to identify the Pareto-optimal or optimal design solutions, considering factors such as performance, area, and power at both the system and component levels. State-of-the-art HLS DSE approaches use machine learning (ML) or heuristic-based methods to identify component-level [11, 21] or system-level (multi-component) [16] Pareto-optimal configurations. Training ML models and evaluating each heuristic requires extensive data. Several HLS benchmarks and datasets are available to the community for evaluating these methodologies [2, 4, 5, 8–10]. However, these existing benchmarks and datasets focus on component/function-level computations without including system characteristics (e.g., end-to-end timing constraints) that may be required in real-world HLS usage scenarios. As such, these datasets cannot meet all experimental conditions, particularly for complex real-world embedded systems that require meeting timing constraints and minimizing energy consumption. For instance, as illustrated in Fig. 1, performing a system-level HLS DSE analysis of a multi-component system (e.g., a wearable pregnancy monitoring

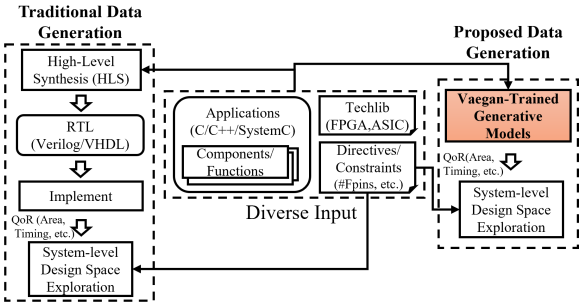


Figure 1: Proposed Vaegan approach to generating synthetic data for system-level HLS design space exploration compared to the traditional approach

system or an autonomous braking system) requires extensive synthesis, implementation, and validation of each component. These systems, initially generated using an HLS tool (e.g., Vitis HLS), are synthesized and implemented with an RTL tool (e.g., Xilinx Vivado) targeting a specific Field-Programmable Gate Array (FPGA). Collecting data for each system configuration is often prohibitively time-consuming.

Despite researchers dedicating considerable time to data collection, datasets targeting only a few FPGA boards and a specific HLS tool may prove insufficient for developing accurate ML models or heuristics for DSE. These models and heuristics are crucial for predicting hardware implementation results from HLS directives (e.g., loop unrolling factor, loop pipeline, array partition) or evaluating heuristic algorithms across diverse HLS tools and FPGA boards. Given the complexity of these unexplored conditions (e.g., embedded systems' time constraints), previous works [16, 21] have demonstrated that synthetic data can be effective in expanding the design space and evaluating HLS DSE methodologies.

Synthetic data offers numerous benefits for both ML-based and heuristic-based HLS DSE. For instance, it is cost effective and time efficient, and provides data diversity. Using both synthetic and real data for training can enhance the robustness of ML models, among other advantages [17]. However, creating synthetic HLS data and achieving *high fidelity* (i.e., a similar data distribution between real and synthetic data) is challenging. Current works that use synthetic data for HLS DSE do not quantify the fidelity of the synthetic data they use, thus calling into question the effectiveness of the DSE evaluation. Existing approaches for generating synthetic data may result in low-fidelity data. We address this critical challenge using an innovative method for generating high-fidelity synthetic HLS data to support HLS DSE research.

In this paper, we propose a novel approach, called *Vaegan*[1], to simplify the generation of synthetic data for system-level HLS DSE (Fig. 1). Vaegan initially formulates and transforms diverse real HLS data, sourced from HLS directives, HLS report estimation, post-synthesis, and post-implementation data into a binary input format. The approach then employs generative machine learning—we explored Variational Autoencoder (VAE) [13] and Generative Adversarial Network (GAN) [3])—to generate and analyze the synthetic data. Vaegan allows designers to use and transform their own ground truth data to generate synthetic data. We evaluate our work using widely recognized metrics, and compare it both qualitatively and quantitatively to real HLS data and two prior works involving synthetic data [7, 16]. Experimental results show that, compared to prior work, Vaegan effectively generates high-fidelity synthetic data that improves the Maximum Mean Discrepancy (MMD) score from real HLS data by 44.05%. We also demonstrate the practical usefulness of the Vaegan approach through a case study that involves generating complex, synthetic HLS data for a wearable pregnancy monitoring system. Our results indicate that this synthetic data expands the Pareto frontier, uncovering new Pareto-optimal solutions that were not present in the original dataset, and extending both the quality and the quantity of the original dataset for design space exploration.

[1]The code is available at https://github.com/yuchaoliao/VAEGAN.git.

2 RELATED WORK

State-of-the-art high-level synthesis (HLS) benchmarks feature many low- and top-level kernels. Examples include Rosetta [9], MachSuite [2], and Polybench [5]. While recent HLS datasets, such as db4hls [4] and HLSDataset [10], are highly valuable, they partially implement the above benchmarks and are limited in their diversity of experimental scenarios. For instance, HLSDataset omits data for execution cycles and post-implementation critical paths which are essential for energy calculation, while db4hls lacks post-synthesis and post-implementation results. Both datasets concentrate on a limited set of workloads and devices, with none considering real-world multi-component embedded systems. Consequently, HLS DSE research has turned to using synthetic data (in addition to real data) for evaluation [15, 16, 21]. For example, Liao et al. [16] randomly generated post-implementation data based on real-world systems for system-level HLS DSE and Wu et al. [21] randomly generated data flow graphs (DFGs) during early-stage HLS exploration. However, these works do not analyze the fidelity of the generated synthetic data to real HLS data. Our examination of the approaches employed in these works reveals a critical discrepancy between the synthetic and real HLS data, highlighting the need for a novel approach to generate and assess synthetic HLS data.

There are many related works on leveraging generative ML models for generating synthetic data in fields such as Natural Language Processing (NLP), vision, healthcare, voice, etc. [17]. However, none of them pertain to the generation of HLS data. Variational Autoencoder (VAE) [13] and Generative Adversarial Network (GAN) [3] are the most common ML frameworks used for synthetic data generation. VAE is a type of generative model that uses a probabilistic approach to compress data into a lower-dimensional space (encoding) and then generates new data by decoding from this latent space. GAN is composed of two neural networks—a generator and a discriminator—competing against each other, where the generator attempts to produce synthetic data that can deceive the discriminator network into believing it is real. Both models are explained in Section 3.2 along with how we tailor them for HLS data. To our knowledge, ours is the first work that tackles the challenge of generating high-fidelity system-level synthetic data for HLS DSE.

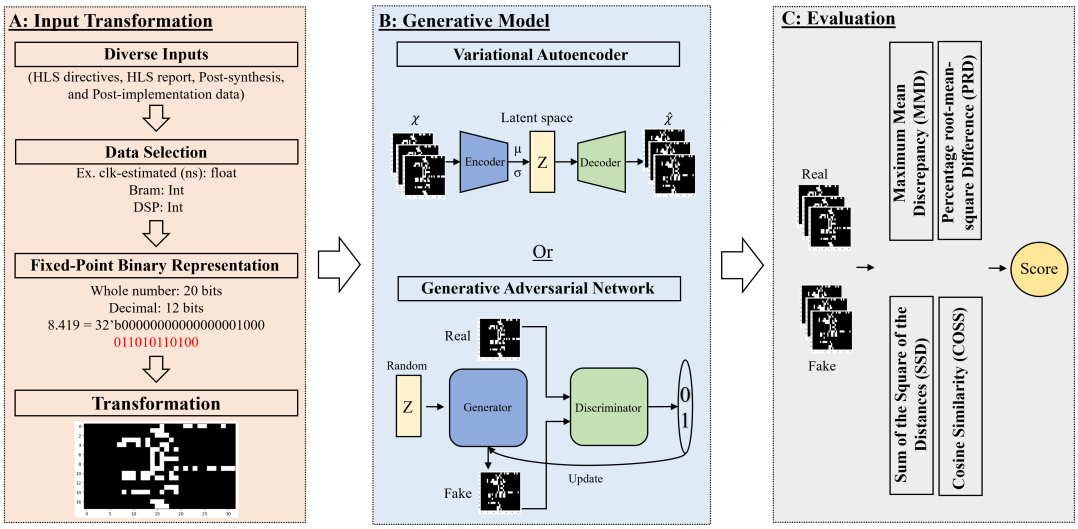


Figure 2: Vaegan comprises three stages: (A) formatting and transforming diverse input data (HLS directives, HLS report estimation, post-synthesis, and post-implementation data) into a format readable by the network. (B) employing ML (here, a Variational Autoencoder (VAE) or a Generative Adversarial Network (GAN)) to generate synthetic HLS data. (C) evaluating the generated synthetic HLS data.

3 METHODOLOGY

Fig. 2 illustrates the overall design flow of Vaegan, which consists of three stages. The first stage involves formalizing the input and output. Diverse input data, such as clock, area, and power are transformed into a format readable by the network. The second stage employs a state-of-the-art generative network, specifically a Variational Autoencoder (VAE) or a Generative Adversarial Network (GAN), fine-tuned to generate the desired synthetic HLS data. The final stage evaluates the models and the generated synthetic data using state-of-the-art metrics. Here, we elaborate on these stages.

3.1 Input Transformation

Fig. 2(A) shows the flow of input transformation. Vaegan begins by constructing the appropriate input sets, a task complicated by the intricate HLS process and the variety of data types. Our approach aims to enable designers to generate their own synthetic HLS data. As such, the first challenge in this step involves selecting critical variables for data generation. A typical HLS design point might comprise C/C++ source code, synthesis directives, Intermediate Representation (IR) of the source code, HLS report estimation data, post-synthesis data, and post-implementation data. Depending on their specific focus, designers may select any subset of these data. For instance, ML-based HLS DSE approaches often predict area and performance from the synthesis directives and IR to either post-synthesis or post-implementation data [11, 21]. To evaluate Vaegan, we use HLS directives, HLS report estimation, post-synthesis, and post-implementation data.

HLS synthesis directives usually consist of loop unrolling factors, loop pipelining, array partitioning, memory type, function inlining, target frequency, etc. HLS report estimation data includes estimated synthesis results. These estimated results resemble actual implementation results from tools like Xilinx Vivado. Post-synthesis and post-implementation data, derived from synthesis tool reports, include metrics for execution cycles, critical path, area, power, and target frequency. Designer-selected data are combined into a single model input. Given the limitations of existing benchmarks and datasets, designers may need to manually generate real data as a ground truth for the ML model.

The second challenge is transforming diverse input data, such as resource usage (integers), power (floating-point numbers), and HLS directives (options with integers), into a network-readable format. Firstly, We propose converting each input variable to a binary fixed-point format, with the precision determined by the real data. In our experiments (Section 4), we used a 32-bit representation, with 20 and 12 bits for the integer and fractional portions, respectively. Each HLS directive is represented using 32 bits, wherein each directive option (unrolling factors, array partitions, etc.) is represented using 4 bits. The options are combined and zero-padded, if needed, to form the 32-bit representation for the directive. Next, to prepare data for a network input, the binary values of all variables and directives in a solution are concatenated row-wise into a 2-D matrix. Advantages of using a fixed-point binary representation include seamlessly handling both continuous and discrete data types and achieving high precision while maintaining a dynamic range, thanks to the separation of integer and fractional values [12].

3.2 Generative Models

The second stage of Vaegan employs a generative model to create synthetic data. We explored a Variational Autoencoder (VAE) and a Generative Adversarial Network (GAN), but primarily use the VAE as it proved to be more effective for our purposes. We note that other state-of-the-art models like diffusion models can be used in this stage, and we plan to explore additional models in future work.

3.2.1 Variational Autoencoder (VAE). A VAE [13] is a probabilistic directed graphical model defined by a joint distribution over a set of latent random variables z and observed variables x, expressed as $p(x, z) = p(x|z)p(z)$. Fig. 2(B) shows a sample VAE. In VAE, an *encoder network* is leveraged to map the input variables to a continuous latent space. The parameters of a variational distribution defined within this latent space can spawn multiple different samples sharing the same underlying distribution. Typically, the prior distribution over the latent random variables, $p(z)$, is chosen as a standard Gaussian distribution, and the data likelihood $p(x|z)$ is generally a Gaussian or Bernoulli distribution whose parameters depend on z through a deep neural network known as the *decoder network*. This decoder is then employed to map the latent space back to the input space, leading to the generation of data points. The loss function ($L(x)$) of the encoder and decoder networks are jointly trained to maximize the evidence lower bound (ELBO):

$$L(x) = \mathbb{E}_{x\sim q(z|x)}[log(p(x|z))] - D_{KL}(q(z|x)\|p(z))$$

where D_{KL} is the Kullback–Leibler (KL) divergence between the latent distribution and standard normal distribution.

We employ a Multilayer Perceptron (MLP) network for both the encoder and decoder (MLPVAE). MLPVAE has two hidden layers and uses a ReLU activation function to provide non-linearity and symmetry for the VAE. A sigmoid activation function is used to reconstruct the input format for the decoder. MLPVAE encodes the real HLS data to a latent distribution by the mean and standard deviation and then decodes a sample from this distribution to reconstruct a new synthetic HLS data. We trained the model using binary cross entropy loss and Adam optimizer with an initial learning rate $\alpha = 0.0001$. A unique feature of the fixed-point input format used here is that we modified the loss function to place additional weight on changes to the most significant bit (MSB). This modification is made to account for the fact that each modification of the MSB causes a greater change in value compared to the lower bits.

3.2.2 Generative Adversarial Network (GAN). A GAN [3], exemplified in Fig. 2(B), comprises a generative model trained through a competitive interplay between a generator and a discriminator network. The generator function $G(z)$ is typically initialized by drawing the latent variable z from a basic prior distribution such as Gaussian, $p(z)$. The discriminator network, $D(x)$, outputs the probability of a given sample originating from the actual data distribution. It aims to differentiate between samples generated by the GAN and actual data. Concurrently, the generator endeavors to create samples with a high degree of realism, intending to deceive the discriminator into accepting these generated outputs as genuine. This iterative process between the generator and discriminator results in a zero-sum game, fostering an environment conducive to unsupervised learning. This contest between the two networks translates into a minimax problem where both networks strive to optimize their performances:

$$\min_G \max_D \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[1 - \log D(G(z))]$$

We employed Deep Convolutional Generative Adversarial Network (DCGAN) [19] which generally demonstrates superior performance over GAN in generating synthetic data. DCGAN uses convolutional neural networks (CNN) for both the generator and discriminator. The generator utilizes four deconvolution layers with Batch normalization and ReLU activation for the first three layers. A sigmoid activation function is used for the last layer to map to the [0,1] range for binary inputs. The generator takes a latent vector z of size 100 from normal distribution as input and generates output matching the real data. The discriminator has four convolution layers with Batch normalization and LeakyReLU activation for the first three layers. Because the discriminator needs to output the probability of whether the input data is real or fake, a sigmoid activation function is also used in the last layer. Like MLPVAE, the DCGAN uses the same loss function as MLPVAE and Adam optimizer. We use the initial learning rate $\alpha = 0.07$ for the generator and $\alpha = 0.0001$ for the discriminator.

3.3 Model Evaluation

Fig. 2(C) illustrates the evaluation process to verify the model's performance and the fidelity of the generated synthetic data to real data. The generated synthetic HLS data $\hat{x}$ is assessed relative to the real input data x using a variety of state-of-the-art metrics to ensure a robust evaluation. These metrics include the Maximum Mean Discrepancy (MMD) [1], Sum of the Square of the Distances (SSD), Percentage Root-mean-square Difference (PRD), and Cosine Similarity (COSS) [14, 18]. We found the Fréchet Inception Distance (FID) [6], which is commonly used for evaluating GANs with 3-channel image inputs, to be incompatible with the 1-channel binary data format used in our work. Except for COSS, where higher values mean greater fidelity, smaller values in all other metrics indicate that the generated synthetic data closely resembles the real data.

4 EXPERIMENTS

We performed experiments with input datasets from HLSDataset [10] as the original HLS data and used the MLPVAE and DCGAN models (Section 3.2) to generate synthetic HLS data. We compared the Vaegan-generated data to the original data and synthetic data generated using two prior works. The first prior work [16] (called 'Gaussian' herein) randomly generated synthetic HLS data separately from all samples in each benchmark and variable based on a normal distribution. The second [7] (called 'ABC' herein) generated genetic data using an approximate Bayesian computation (ABC) procedure. While this work did not generate HLS data, we found the approach used instructive for evaluating our work. We performed the experiments on an Intel i7 11700k @3.6GHz CPU with an NVIDIA RTX 3080 Ti GPU.

4.1 Input

We selected the HLSDataset due to its inclusion of partial post-implementation data, offering more diverse input data than db4hls [4]. HLSDataset targets two FPGA parts: xc7v585tffg1157-3 and xczu9eg-ffvb1156-2-i. We leveraged both parts to evaluate our approach's ability to generate distinct synthetic data for specific parts. HLSDataset provides several hardware configurations comprising HLS directives, HLS report estimation, post-synthesis, and partial

Table 1: Example input configuration with 20 variables from HLSDataset. The variables are generated from the HLS report and synthesis results and include statistics like the number of digital signal processors (DSPs), block RAMs (BRAM), lookup tables (LUTs), flip-flops (FF), dynamic power (DP), shift register LUTs (SRL), etc.

Project	Clk-estimated(ns)	BRAM	DSP	FF	LUT	c-num-arith
io1-l2n1n1-l4n1n1	8.419	32	5	727	1231	22
c-num-logic	**rtl-num-arith**	**rtl-num-logic**	**input-port**	**output-port**	**DP(mW)**	**Total LUTs**
10	19	10	128	32	17.103	654
Logic LUTs	**LUTRAMs**	**SRLs**	**FFs**	**RAMB36**	**RAMB18**	**DSP48**
654	0	0	498	16	0	5

Table 2: Hyperparameters for MLPVAE and DCGAN training

Model	MLPVAE	DCGAN-Generator	DCGAN-Descriminator
Learning Rate	1e-4	7e-2	1e-4
Feature Map	16	32	32
Batch Size	20	20	20
Number of Epochs	150	150	150
Optimizer	Adam	Adam	Adam
Layer Type	Linear	CNN	CNN
Number of Layers	4	4	4

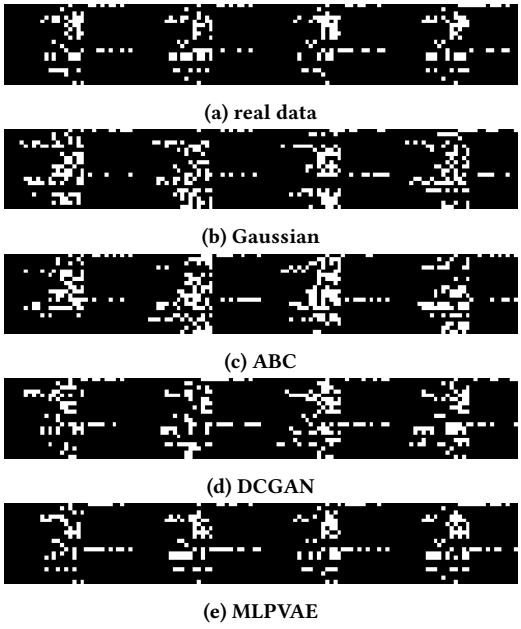
post-implementation data. We separated the data into two categories: with and without HLS directives. This separation was necessary because each benchmark exhibited a different total number of loops and arrays, resulting in a varying number of HLS directives. Initially, we combined the data from the HLS report, post-synthesis, and post-implementation that did not include HLS directives. Of the 23 variables available per configuration, we selected 20 variables and eliminated 3 variables—target clock period of $10ns$, clock uncertainty of $1.25ns$, and frequency of $100MHz$—that were fixed across all samples (see Table 1 for the details of variables). For the data with HLS directives, we used three directive options (loop pipeline, loop unrolling, and array partition) and eliminated the two (interface and resource) that were fixed for all benchmarks.

We collected a total of 9557 samples without HLS directives from two FPGA parts and 3717 samples with HLS directives for FPGA part xc7v585tffg1157-3 for Polybench. Of the 9 Polybench benchmarks, only 7 have directive files available and the directives are the same for both FPGA parts, hence the smaller number of samples. After selecting the variables from the dataset, we used a Python script[2] to convert each variable to the 32-bit input format.

Table 3: Evaluation of DCGAN and MLPVAE using MMD, SSD, PRD, and COSS for two FPGA parts without HLS directives. Larger numbers are better for COSS, while smaller are better for all other metrics. Results depict the mean over 5 runs of each model and the standard deviation (±).

Model	MMD	SSD	PRD%	COSS
FPGA part: xc7v585tffg1157-3				
Gaussian [16]	0.666 ± 0.001	99.697 ± 0.087	116.752 ± 0.102	0.390 ± 0.001
ABC [7]	0.667 ± 0.002	99.714 ± 0.082	116.754 ± 0.033	0.390 ± 0.001
DCGAN	0.614 ± 0.002	86.623 ± 0.098	110.707 ± 0.055	0.458 ± 0.001
MLPVAE	**0.370 ± 0.002**	**47.721 ± 0.185**	**82.830 ± 0.302**	**0.698 ± 0.002**
FPGA part: xczu9eg-ffvb1156-2-i				
Gaussian [16]	0.662 ± 0.001	98.405 ± 0.081	116.320 ± 0.155	0.393 ± 0.001
ABC [7]	0.663 ± 0.003	98.420 ± 0.177	116.207 ± 0.106	0.394 ± 0.001
DCGAN	0.660 ± 0.001	87.215 ± 0.100	114.031 ± 0.095	0.436 ± 0.001
MLPVAE	**0.374 ± 0.004**	**47.641 ± 0.166**	**82.953 ± 0.252**	**0.696 ± 0.001**

[2]https://github.com/yuchaoliao/VAEGAN/blob/main/DataTransformation.py



(a) real data

(b) Gaussian

(c) ABC

(d) DCGAN

(e) MLPVAE

Figure 3: Visualized HLS data comparison between (a) real data, (b) Gaussian [16], (c) ABC [7], (d) DCGAN, and (e) MLPVAE for part xc7v585tffg1157-3 without HLS directives

4.2 Model Configuration & Training Parameters

Table 2 presents the hyperparameters used for the MLPVAE model, as well as for the generator and discriminator components of the DCGAN model. Without HLS directives, both models were trained separately on two FPGA parts, and synthetic HLS data was generated to match the real data size of 20 variables, each with 32 bits (20x32). Results (Section 4.3) revealed the superiority of MLPVAE over both DCGAN and prior works. Thus, with HLS directives, for brevity, we report results for MLPVAE for part xc7v585tffg1157-3, with input widths of the number of directives (ranging from 19 to 35) plus 20 variables. The initial learning rate was the same $\alpha = 0.0001$ for both the generator and discriminator in DCGAN. We slowly increased the learning rate of the generator [6] to $\alpha = 0.07$ until we found the balance between the generator and the discriminator. In each training epoch, we evaluated the model and generated synthetic HLS data using the MMD, SSD, PRD, and COSS metrics.

4.3 Results

Fig. 3 depicts a visualization of the real HLS data without directives (for part xc7v585tffg1157-3) compared to the synthetic HLS data generated by our DCGAN and MLPVAE models and prior works

Table 4: Evaluation of MLPVAE using MMD, SSD, PRD, and COSS for part xc7v585tffg1157-3 with HLS directives for seven benchmarks.

Benchamrk (Directives)	MMD	SSD	PRD%	COSS
syrk(23)	0.528 ± 0.004	57.009 ± 0.193	74.764 ± 0.147	0.745 ± 0.001
syr2k(36)	0.489 ± 0.006	61.353 ± 0.213	70.074 ± 0.163	0.777 ± 0.001
k3mm(35)	0.532 ± 0.005	58.597 ± 0.315	69.191 ± 0.191	0.781 ± 0.001
k2mm(35)	0.516 ± 0.004	60.837 ± 0.296	70.027 ± 0.186	0.776 ± 0.001
gesummv(24)	0.535 ± 0.007	52.539 ± 0.282	71.623 ± 0.252	0.765 ± 0.001
gemm(24)	0.528 ± 0.011	57.643 ± 0.304	73.201 ± 0.186	0.757 ± 0.001
bicg(19)	0.579 ± 0.008	55.230 ± 0.219	78.418 ± 0.254	0.723 ± 0.001
Average	**0.529**	**57.601**	**72.471**	**0.761**

(*Gaussian* and *ABC*). As can be inferred from the visualization, both GAN and VAE are superior to the prior works in generating realistic synthetic data. A similar visualization with HLS directives (the figure is omitted for brevity) showed Vaegan's ability to generate data resembling the real data. However, we conducted additional quantitative assessments to more accurately evaluate the models.

Table 3 provides a detailed evaluation of MLPVAE, DCGAN, and prior works (Gaussian and ABC) using MMD, SSD, PRD, and COSS metrics for data without HLS directives across both FPGA parts. MLPVAE consistently outperforms both DCGAN and prior methods across all metrics. The two prior works (Gaussian and ABC) generally perform similarly across all metrics. Specifically, regarding MMD, which represents the distribution score between real and synthetic data, MLPVAE generates synthetic HLS data that is on average 44.0%, 44.1%, and 41.6% superior to that of Gaussian, ABC, and DCGAN for both FPGA parts. Considering the SSD, PRD, and COSS metrics, MLPVAE outperforms Gaussian by an average of 51.9%, 28.9%, and 78%, respectively, and outperforms DCGAN by an average of 45.1%, 26.2%, and 56%, respectively.

DCGAN underperforms MLPVAE primarily due to the inherent challenges of fine-tuning. This stems from the sensitivity of DCGAN's convolutional layers to architectural choices (number of layers, filter sizes, strides), requiring extensive experimentation for optimal results. As a result, finding the right set of hyperparameters that leads to convergence can be less intuitive than with MLPVAE. To achieve high-fidelity synthetic HLS data, we fine-tuned our DCGAN's convolutional layers and hyperparameters significantly more often than our VAE's—over a hundred iterations for DCGAN compared to less than twenty for VAE, to achieve high-fidelity synthetic HLS data. On average, each experiment using DCGAN ran for 31 minutes and 46 seconds, whereas MLPVAE experiments ran for 11 minutes and 25 seconds. As such, compared to DCGAN, MLPVAE achieves better results while taking less time.

Table 4 shows the evaluation of MLPVAE on HLS data with HLS directives, using four metrics for the part xc7v585tffg1157-3 across seven benchmarks in Polybench. For brevity, we focus on MLPVAE due to its superiority over DCGAN and prior works. We found that MLPVAE not only reduces training time (by up to 48×) compared to DCGAN but also significantly improves the quality of results. MLPVAE generates synthetic data that resembles the real data, achieving average MMD, SSD, PRD, and COSS scores of 0.529, 57.601, 72.471, and 0.761, respectively. These results show the promise of the proposed approach for generating high-fidelity synthetic data for HLS DSE. Furthermore, we observed that the results were benchmark-dependent because of variabilities in each benchmark's number of loops and loop levels, necessitating careful model tuning for each benchmark to optimize the synthetic data generation process.

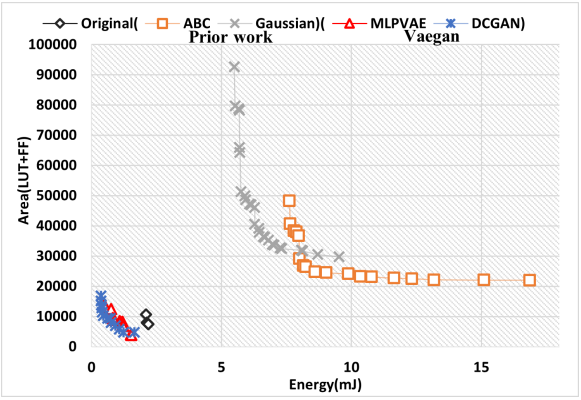


Figure 4: Pareto-optimal design points for the area (FF+LUT) and energy of the original three-component wearable pregnancy monitoring system compared with prior work (ABC and Gaussian) and Vaegan (MLVPAE and DCGAN).

5 CASE STUDY: WEARABLE SYSTEMS

We briefly demonstrate and compare the use of Vaegan to generate synthetic data for a complex wearable pregnancy monitoring (WPM) device using real data with prior approaches. This case study involves a system-level DSE using a genetic algorithm to determine the Pareto-optimal energy and area under a constraint of the number of frequencies available on the target FPGA board. The WPM device comprises three components to monitor and process data for *maternal heart rate*, *blood oxygen saturation*, and *abdomen contraction electromyography*. Each component runs with its period constraint, and communication is required between each component's controller and computation for each algorithm. The system-level design space comprises a combination of component design alternatives and their latencies, HLS directives to generate these design alternatives, and the interactions between the different components, resulting in a complex system-level DSE challenge. For the input HLS data, we used 69 manually generated post-implementation data points for the three-component WPM device. The design points are from different design alternatives (e.g., clock frequencies, latency constraints) yielding 1.92e+10 solutions, evaluated with four metrics: execution cycles, area (FF+LUT), critical path, and power.

Due to space constraints, we omit the detailed description of the genetic algorithm, but it is modeled after a recent HLS DSE algorithm in [16]. We use Vaegan, ABC, and Gaussian to generate four synthetic systems with the same size for the design alternatives. Fig. 4 presents the system-level Pareto-optimal configurations after applying the genetic algorithm to the original and synthetic systems using MLPVAE, DCGAN, ABC, and Gaussian. As seen in the figure, Vaegan generates a Pareto-optimal frontier that is much closer to the original system than ABC and Gaussian. We quantified the quality of the generated solutions using the Average Distance to Reference Set (ADRS) [20]. Compared to the original system, MLPVAE, DCGAN, ABC, and Gaussian achieved ADRS scores of 122.6%, 124.1%, 950.8%, and 1353.6%, respectively. A higher ADRS score implies a greater disparity from the original system in the

Pareto frontier. MLPVAE modestly improved the ADRS over DCGAN by 1.23%, and significantly outperformed ABC and Gaussian by 87.1% and 90.9%, respectively.

6 CONCLUSION

Existing benchmarks and datasets cannot cover all experimental conditions for system-level High-Level Synthesis (HLS) design space exploration (DSE). Consequently, utilizing synthetic data has emerged as a solution to this challenge. This paper proposes a novel approach—Vaegan—for generating realistic synthetic HLS data using generative machine learning models. To this end, we explored two kinds of models—a Multilayer Perception Variational Autoencoder (MLPVAE) and Deep Convolutional Generative Adversarial Networks (DCGAN)—and adapted them to generate synthetic HLS data. Experimental results show that Vaegan generates synthetic HLS data that closely mimics the ground truth distribution. Compared to prior work, Vaegan produced a Pareto front with superior design points, achieving energy and area values significantly closer to the original dataset. This demonstrates the effectiveness of our approach for improved design space exploration in complex systems.

Future work will explore the use of Vaegan as a machine learning predictor to bypass the High-Level Synthesis (HLS) process. Additionally, we will explore the integration of models such as the diffusion model into Vaegan for generating synthetic HLS data. This work will also be extended to accommodate various inputs, such as data flow graphs, for early-stage HLS exploration.

ACKNOWLEDGMENTS

This work was partially supported by the Technology and Research Initiative Fund (TRIF) provided to the University of Arizona by the Arizona Board of Regents (ABOR) and National Science Foundation (NSF) Grant CNS-1844952. The work of R. Tandon was supported by NSF grants CAREER 1651492, CCF-2100013, CNS-2209951, and CNS-2317192.

REFERENCES

[1] Arthur Gretton et al. 2012. A kernel two-sample test. *The Journal of Machine Learning Research* 13, 1 (2012), 723–773.

[2] Brandon Reagen et al. 2014. MachSuite: Benchmarks for accelerator design and customized architectures. In *2014 IEEE International Symposium on Workload Characterization (IISWC)*. 110–119. https://doi.org/10.1109/IISWC.2014.6983050

[3] Ian Goodfellow et al. 2020. Generative adversarial networks. *Commun. ACM* 63, 11 (2020), 139–144.

[4] Lorenzo Ferretti et al. 2021. Db4hls: a database of high-level synthesis design space explorations. *IEEE Embedded Systems Letters* 13, 4 (2021), 194–197.

[5] Louis-Noël Pouchet et al. 2012. Polybench: The polyhedral benchmark suite. *URL: http://www. cs. ucla. edu/pouchet/software/polybench* 437 (2012), 1–1.

[6] Martin Heusel et al. 2017. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems* 30 (2017).

[7] Sophie Wharrie et al. 2022. HAPNEST: An efficient tool for generating large-scale genetics datasets from limited training data. In *NeurIPS 2022 Workshop on Synthetic Data for Empowering ML Research*.

[8] Yuko Hara et al. 2008. CHStone: A benchmark program suite for practical C-based high-level synthesis. In *2008 IEEE International Symposium on Circuits and Systems (ISCAS)*. 1192–1195. https://doi.org/10.1109/ISCAS.2008.4541637

[9] Yuan Zhou et al. 2018. Rosetta: A Realistic High-Level Synthesis Benchmark Suite for Software Programmable FPGAs. In *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '18)*. 269–278. https://doi.org/10.1145/3174243.3174255

[10] Zhigang Wei et al. 2023. HLSDataset: Open-Source Dataset for ML-Assisted FPGA Design using High Level Synthesis. In *2023 IEEE 34th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 197–204.

[11] Pingakshya Goswami, Benjamin Carrion Schaefer, and Dinesh Bhatia. 2023. Machine learning based fast and accurate High Level Synthesis design space exploration: From graph to synthesis. *Integration* 88 (2023), 116–124.

[12] Dhiraj Kalamkar, Dheevatsa Mudigere, Naveen Mellempudi, Dipankar Das, Kunal Banerjee, Sasikanth Avancha, Dharma Teja Vooturi, Nataraj Jammalamadaka, Jianyu Huang, Hector Yuen, et al. 2019. A study of BFLOAT16 for deep learning training. *arXiv preprint arXiv:1905.12322* (2019).

[13] Diederik P Kingma and Max Welling. 2022. Auto-Encoding Variational Bayes. arXiv:1312.6114 [stat.ML]

[14] Huayu Li, Gregory Ditzler, Janet Roveda, and Ao Li. 2023. DeScoD-ECG: Deep Score-Based Diffusion Model for ECG Baseline Wander and Noise Removal. *IEEE Journal of Biomedical and Health Informatics* (2023).

[15] Yuchao Liao, Tosiron Adegbija, and Roman Lysecky. 2022. A high-level synthesis approach for precisely-timed, energy-efficient embedded systems. *Sustainable Computing: Informatics and Systems* 35 (2022), 100741.

[16] Yuchao Liao, Tosiron Adegbija, and Roman Lysecky. 2023. Efficient System-Level Design Space Exploration for High-Level Synthesis Using Pareto-Optimal Subspace Pruning. In *Proceedings of the 28th Asia and South Pacific Design Automation Conference (ASPDAC '23)*. 567–572. https://doi.org/10.1145/3566097.3567841

[17] Yingzhou Lu, Huazheng Wang, and Wenqi Wei. 2023. Machine Learning for Synthetic Data Generation: a Review. *arXiv preprint arXiv:2302.04062* (2023).

[18] R. Nygaard, G. Melnikov, and A.K. Katsaggelos. 2001. A rate distortion optimal ECG coding algorithm. *IEEE Transactions on Biomedical Engineering* 48, 1 (2001), 28–40. https://doi.org/10.1109/10.900246

[19] Alec Radford, Luke Metz, and Soumith Chintala. 2016. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. arXiv:1511.06434 [cs.LG]

[20] Benjamin Carrion Schafer and Zi Wang. 2020. High-Level Synthesis Design Space Exploration: Past, Present, and Future. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 10 (2020), 2628–2639.

[21] Nan Wu, Yuan Xie, and Cong Hao. 2022. IronMan-Pro: Multiobjective Design Space Exploration in HLS via Reinforcement Learning and Graph Neural Network-Based Modeling. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 42, 3 (2022), 900–913.