CircuitVAE: Efficient and Scalable Latent Circuit Optimization

Jialin Song[*]
NVIDIA
Aidan M. Swope[*†]
Robert Kirby
NVIDIA
Rajarshi Roy
NVIDIA
Saad Godil[†]
Jonathan Raiman
NVIDIA
Bryan Catanzaro
NVIDIA

Abstract

Automatically designing fast and space-efficient digital circuits is challenging because circuits are discrete, must exactly implement the desired logic, and are costly to simulate. We address these challenges with CircuitVAE, a search algorithm which embeds computation graphs in a continuous space and optimizes a learned surrogate of physical simulation by gradient descent. By carefully controlling overfitting of the simulation surrogate and ensuring diverse exploration, our algorithm is highly sample-efficient, yet gracefully scales to large problem instances and high sample budgets. We test CircuitVAE by designing binary adders across a large range of sizes, IO timing constraints, and sample budgets. Our method excels at designing large circuits, where other algorithms struggle: compared to reinforcement learning and genetic algorithms, CircuitVAE typically finds 64-bit adders which are smaller and faster using less than half the sample budget. We also find CircuitVAE can design state-of-the-art adders in a real-world chip, demonstrating that our method can outperform commercial tools in a realistic setting.

Keywords: Circuit Design, Generative Models, Experimental Design, Latent Space Optimization

1. Introduction

As the workhorses of today's parallel processors, optimizing the design of binary adders is an important and well-studied problem. However, because the physical characteristics of a given design change as manufacturing technology improves, and because each adder in a larger design may face different constraints, classical adder designs which minimize analytical properties such as circuit depth often perform poorly in practice. More recent approaches use physical synthesis and simulation to optimize adders for real-world area,

*. Equal contribution. Names listed alphabetically. Correspondence to: `jialins@nvidia.com`.
†. Work conducted while at NVIDIA.

delay, and power consumption Roy et al. (2021); Song et al. (2022). Such algorithms remain expensive, though, because physical synthesis is slow, the problem is discrete, and the search space grows exponentially with the number of bits to be summed. Therefore, optimizing larger adders has generally remained intractable.

We present CircuitVAE, a highly sample-efficient algorithm for optimizing binary adders which outperforms human designs and commercial tools while requiring fewer simulations than competing approaches. CircuitVAE solves the two key challenges of this domain, discrete search and an expensive objective function, by embedding circuits in a continuous space and learning to predict the results of physical simulation. We introduce two domain-agnostic improvements to the standard latent-space optimization framework. Our first, prior-regularized search, prevents search points from "overfitting" the cost predictor far from the data manifold. The second, cost-weighted sampling, helps balance quality and diversity in the explored points by initializing them from high-performing prior evaluations. Through extensive ablations, we demonstrate that these improvements enable gradient-based search to outperform Bayesian optimization in the latent space, contrary to the standard approach.

We evaluate CircuitVAE in numerous settings, both on standard benchmarks and in the real world. Across various sizes of adders, and emulating various cost tradeoffs between area and delay, we find that CircuitVAE outperforms human designs, commercial tools, and the prior state-of-the-art reinforcement learning algorithm in cost and simulation requirements. Finally, we integrate CircuitVAE into a real-world chip design workflow and show that it outperforms commercial tools in a realistic setting.

2. CircuitVAE

In this section, we describe our CircuitVAE algorithm for optimizing prefix adders. In subsection 2.1, we describe how to train CircuitVAE by combining the standard VAE training objective with a cost predictor loss. In subsection 2.2, we describe how to search in the latent space of CircuitVAE, including our proposed prior-regularized search and cost-weighted sampling techniques. Due to the space constraint, please refer to subsection 5.1 and subsection 5.4 in the Appendix for a detailed introduction to the prefix adder design problem.

2.1 Training

We denote by $\mathcal{X}$ the discrete search space for all N-bit adders. Optimizing over $\mathcal{X}$ is challenging: computation graphs with many nodes or edges in common may not have similar costs because removing one node is sufficient to change the critical path. Therefore, we embed $\mathcal{X}$ into a continuous latent space $\mathcal{Z}$ with a VAE augmented with a cost prediction head.

Concretely, we learn an encoding function $q_\phi(z|x)$ that encodes an input x to a distribution over $\mathcal{Z}$, and a decoding function $p_\theta(x|z)$ that decodes a latent variable z to a distribution over $\mathcal{X}$. We optimize the parameters θ and ϕ by maximizing the evidence lower-bound (ELBO) (Kingma and Welling, 2013): $\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}(q_\phi(z|x)||p_\theta(z))$. Our prior $p_\theta(z)$ is a diagonal unit Gaussian. To balance adherence to the prior with other training objectives, we use a β-VAE (Higgins et al., 2017; Bozkurt et al., 2021). Our training loss is:

$$\mathcal{L}_{\theta,\phi}(x) = \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - \beta D_{KL}(q_\phi(z|x)||p_\theta(z)) \tag{1}$$

Given a dataset of prefix adders and their costs $\mathcal{D} = \{(x_i, c_i)\}_{i=1}^n$, we can fit the VAE parameters by maximizing $\sum_{i=1}^n \mathcal{L}_{\theta,\phi}(x_i)$ via gradient descent using the reparameterization trick (Kingma and Welling, 2013).

In addition to learning the encoder and decoder, we also learn a cost predictor model $f_\pi : \mathcal{Z} \to \mathbb{R}$. For a datapoint (x, c), we first map x to a latent space distribution $q_\phi(z|x)$, next we sample a latent variable $z \sim q_\phi(z|x)$ (again using the reparametrization trick), and finally we predict its cost $f_\pi(z)$. Our cost prediction loss is $\mathcal{L}_\pi(x, c) = (f_\pi(z) - c)^2$. The cost predictor enables optimization, but it also helps shape the latent space: observe that if two circuits x_1, x_2 with very different costs have overlapping posteriors $q_\phi(\cdot \mid x_1), q_\phi(\cdot \mid x_2)$, the cost predictor will fail to distinguish them. Therefore, the training loss is minimized when circuits with similar costs are grouped together, which aids optimization.

Following (Tripp et al., 2020), we reweight datapoints according to their cost to give promising points more volume in latent space. Specifically, the weight of a datapoint $(x, c) \in \mathcal{D}$ is

$$w(x; \mathcal{D}, k) \propto \frac{1}{kn + \mathrm{rank}_\mathcal{D}(x)}, \quad \mathrm{rank}_\mathcal{D}(x) = |\{x_i : c_i < c,\ (x_i, c_i) \in \mathcal{D}\}|\ , \tag{2}$$

where k is a hyperparameter controlling the relative weights among the datapoints. For simplicity, we use $w_i(\mathcal{D})$ to denote the normalized weight for a datapoint (x_i, c_i). Note that we need to recompute these weights after acquiring new datapoints because of the dependency on $\mathcal{D}$.

Finally, we can put the two losses together to obtain the overall loss objective

$$\mathcal{L}_{\theta,\phi,\pi}(\mathcal{D}) = \sum_{i=1}^n w_i(\mathcal{D})\mathcal{L}_{\theta,\phi}(x_i) + \lambda\mathcal{L}_\pi(x_i, c_i) \tag{3}$$

where $\lambda \in \mathbb{R}^+$ is a hyperparameter to balance the VAE training loss and the cost prediction loss. In all experiments, we set $\beta = 0.01$, $\lambda = 10.0$, and $k = 0.001$, and optimize $\mathcal{L}_{\theta,\phi,\pi}$ with Adam (Kingma and Ba, 2014).

2.2 Optimization

Once we fit the parameters for the VAE with the cost predictor, we can choose new designs to query by minimizing the predicted costs with f_π. In our experiments, we instantiate f_π with a feed-forward neural network and perform gradient descent directly in the latent space by differentiating through f_π with respect to its inputs.

Naively optimizing the predicted cost without constraints yields poor results (Nguyen et al., 2016; Gómez-Bombarelli et al., 2018; Griffiths and Hernández-Lobato, 2020) because the cost predictor is only accurate near regions where training examples are available. We propose *prior regularization*, a means of softly constraining optimized latents to stay near the origin where the majority of the dataset lies. We optimize latents according to a linear combination of predicted cost and prior log-probability:

$$g(z) = f_\pi(z) - \gamma \log p_\theta(z) \tag{4}$$

Algorithm 1 CircuitVAE

```
1: Input: $\mathcal{D}_0$ initial dataset; $f$ a blackbox function available for queries; $M$ the number
   of data acquisition rounds; $m$ the number of parallel latent search; $T$ the number of
   gradient descent steps for latent space optimization; $t$ the interval to capture latents
   during optimization
2: $\mathcal{D} \leftarrow \mathcal{D}_0$
3: for $i = 1...M$ do
4:     Compute sample weights for $\mathcal{D}$ (Equation 2)
5:     Fit parameters $(\theta, \phi, \pi)$ of VAE with the cost predictor on $\mathcal{D}$ with the weighted
       training objective (Equation 3)
6:     Sample $m$ points from $\mathcal{D}$ proportional to sample weights
7:     Sample $m$ initial latents with $q_\phi$
8:     Optimize $g(z)$ (Equation 4) with gradient descent from the initial latents for $T$ steps
       and capture a set of latents $Z_i$ along the optimization trajectory after every $t$ gradient
       steps
9:     Sample a new set of $X_i$ by decoding $Z_i$ through $p_\theta$
10:    Query $f$ on $\mathcal{X}_t$ to obtain $\mathcal{D}_i$
11:    $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_i$
12: end for
13: return the lowest cost point in $\mathcal{D}$
```

where γ is a hyperparameter to control the strength of the prior regularization. While (Tripp et al., 2020) propose constraining search to a box around the origin for the same reason, we note that in high-dimensional space a box has exponentially many corners, and so a box large enough to contain most of the data mass is likely to have many uninhabited regions.

To balance quality and diversity of our samples, we also initialize the starting latent variables close to valid and high-performing circuits by a cost-weighted sampling method. Specifically, we sample circuits from the current dataset (Line 6 in Algorithm 1) proportional to their datapoint weights (Equation 2). For a sample design x, we obtain an initial latent variable z_0 from the posterior $q_\phi(z|x)$ (Line 7) and the gradient descent on $g(z)$ starts at z_0. This procedure ensures that latents are initialized in high-probability and low-cost regions, while being diverse enough to provide good training data for the next round.

We perform gradient descent for a fixed number of steps and capture the latent variable values after every t steps to get z_t, z_{2t}, etc. (Line 8). For each z_t, we decode to a distribution over $\mathcal{X}$ with $p_\theta(x|z_t)$ and sample a design x to query its cost c (Line 9 and 10). CircuitVAE (Algorithm 1) repeats the training and optimization loop multiple times. In practice, we can parallelize latent space gradient descent (Line 8) to further accelerate the search.

3. Experiments

3.1 Training and evaluation details

We evaluated circuits following Roy et al. (2021). Prefix graphs generated by CircuitVAE are first legalized by inserting missing parents of existing nodes—this may be considered part of the objective function, so our cost predictor learns to infer the same value for equivalent

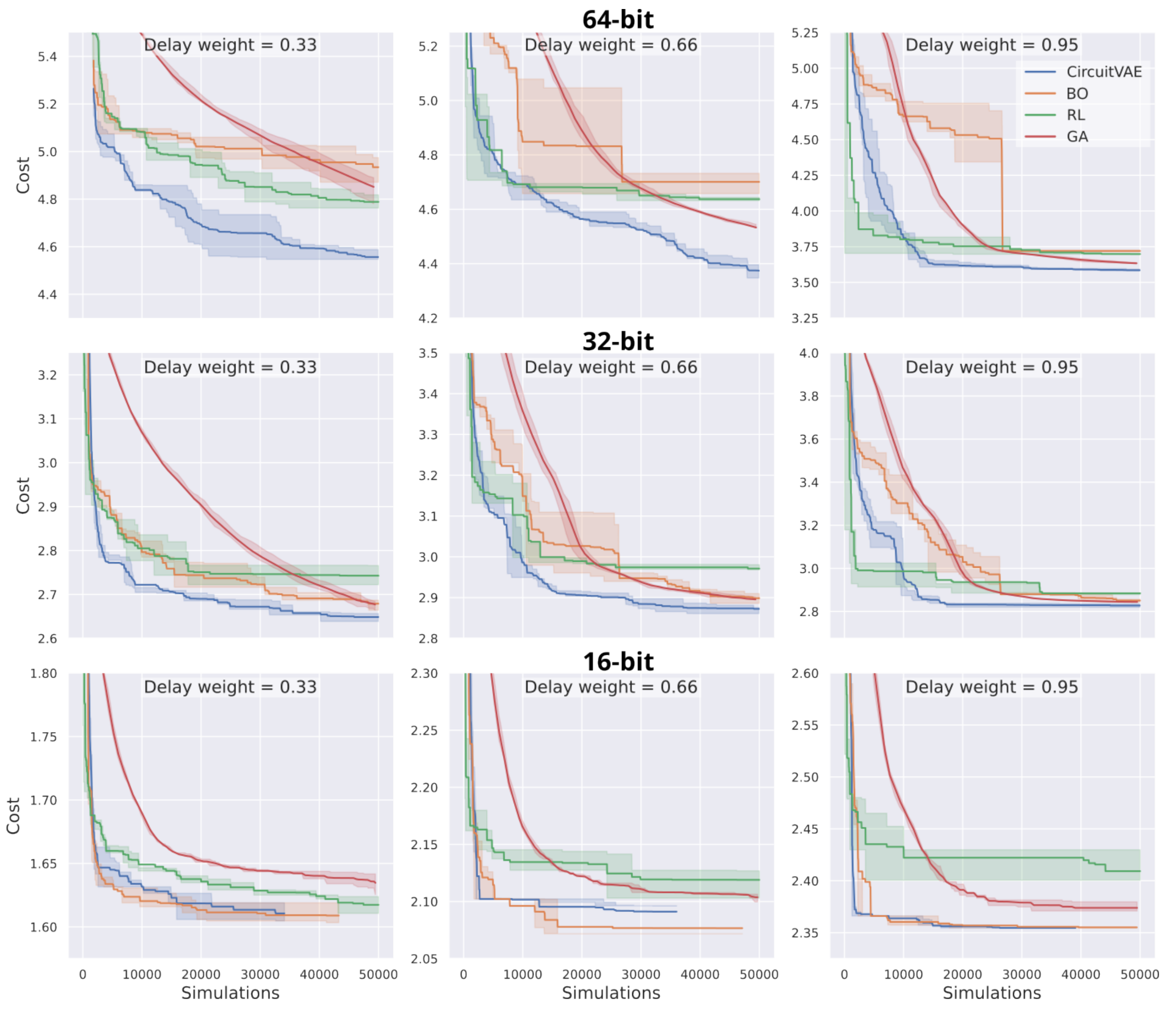

Figure 1: Curves of circuit cost (lower is better) vs simulation budget across a range of circuit sizes (rows) and timing constraints (columns). CircuitVAE consistently achieves lower costs at fewer simulations.

circuits. We then compile the legalized graph into a netlist using the 45-nanometer Nangate45 cell library (Ajayi et al., 2019) and then physically synthesize the circuit using OpenPhySyn (Agiza and Reda, 2020). Our encoder and decoder were each $\sim$ 1M-parameter CNNs autoencoding the computation graph with a 2-layer MLP as the cost predictor. We represent circuits using the grid format described by Roy et al. (2021). All experiments used one A100 GPU and 24 CPU cores. For a full description of the model architecture and hyperparameters, please refer to the subsection subsection 5.5 in the Appendix.

3.2 Comparing search algorithms

We compared CircuitVAE to three alternative search algorithms on the task of optimizing adders across a range of simulation budgets. Our results are summarized in Figure 1: in all settings but 16-bit, CircuitVAE outperforms all other methods at every budget.

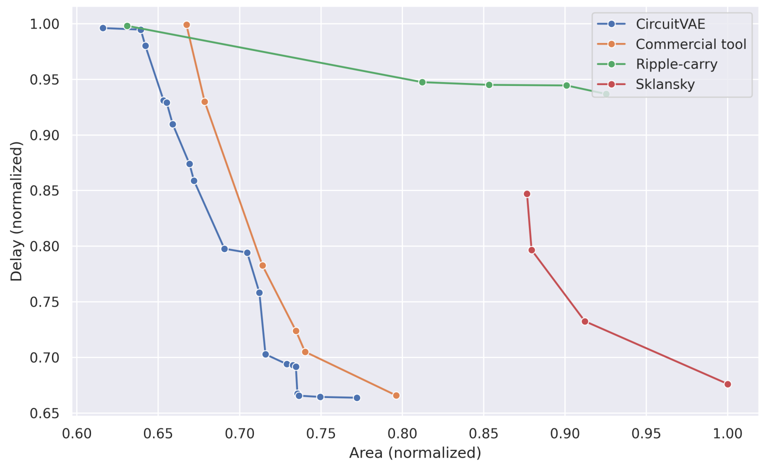

Figure 2: Comparing the area-delay Pareto frontiers of 8nm circuits in a realistic setting. CircuitVAE's designs Pareto-dominate both human-designed circuits and a commercial design tool.

Our primary baseline is PrefixRL ("RL"), the prior state-of-the-art reported by Roy et al. (2021). We also compared against a genetic algorithm ("GA") directly optimizing a bitvector representation of the circuit; we used the first few generations of GA as the initial data to train CircuitVAE. Finally, we compared against a variant of CircuitVAE which employs BO in the latent space, a practice which has become common (Tripp et al., 2020).

We repeated this experiment for bitwidths in {16, 32, 64} and delay weights in {0.33, 0.66, 0.95}. For CircuitVAE and Bayesian experiments, we launched runs with approximately 1,000, 5,000, 10,000, and 30,000 initial datapoints and grouped these runs into a single curve to report performance across a range of budgets; the initial simulations required to build the dataset were counted against these methods' budgets. We ran each experiment with five different random seeds and independently collected initial datasets, and report the median and interquartile ranges across these runs.

It is interesting to note how gradient-based search outperforms latent Bayesian search in most settings. We hypothesize that our higher-capacity neural score predictor can learn more from large datasets than the Bayesian surrogate model, and that mitigating overoptimization with prior-regularized search enables quickly identifying promising candidates.

3.3 Designing real-world circuits

To evaluate CircuitVAE in a more realistic setting, we tried using it in place of a commercial tool in a real-world datapath design. We used CircuitVAE to design 31-bit adders at delay weights in {0.3, 0.6, 0.95} using OpenPhySyn as described above; however, we generated netlists with a proprietary 8nm cell library, and used bit input and output timings captured from a complete datapath. Then, we synthesized the most promising designs using a commercial design tool. Note the domain gap in the cost function between training and evaluation: the commercial tool makes different choices with respect to netlist buffering, gate sizing, cell placement, etc. Nevertheless, as Figure 2 shows, we managed to Pareto-dominate both the design tool's provided adders and common human-designed adders.

4. Conclusion

In this work, we demonstrated that CircuitVAE can efficiently optimize binary adders, even in the difficult cases of large circuits and tight timing constraints. However, the authors optimistically believe that the impact of this work extends beyond adders. Our method may be applied unchanged to optimize other prefix computations, such as leading zero detectors; by replacing the prefix graph with another data structure, one might also optimize multipliers or other circuits. Finally, two of our key innovations, prior-regularized search and cost-weighted sampling, can apply to any gradient-based latent-space optimization algorithm, such as Gómez-Bombarelli et al. (2018) use to design chemicals. We hope to address these exciting possibilities in future work.

References

Ahmed Agiza and Sherief Reda. Openphysyn: An open-source physical synthesis optimization toolkit. 2020.

T Ajayi, D Blaauw, TB Chan, CK Cheng, VA Chhabria, DK Choo, M Coltella, S Dobre, R Dreslinski, M Fogaça, et al. Openroad: Toward a self-driving, open-source digital layout implementation tool chain. *Proc. GOMACTECH*, pages 1105–1110, 2019.

Alican Bozkurt, Babak Esmaeili, Jean-Baptiste Tristan, Dana Brooks, Jennifer Dy, and Jan-Willem van de Meent. Rate-regularization and generalization in variational autoencoders. In *International Conference on Artificial Intelligence and Statistics*, pages 3880–3888. PMLR, 2021.

R.P. Brent and H.Y. Kung. A regular layout for parallel adders. *IEEE Transactions on Computer*, C-31:260–264, 1982.

L Davis, editor. *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, 1991.

Rafael Gómez-Bombarelli, Jennifer N Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, Ryan P Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS central science*, 4(2):268–276, 2018.

Ryan-Rhys Griffiths and José Miguel Hernández-Lobato. Constrained bayesian optimization for automatic chemical design using variational autoencoders. *Chemical science*, 11(2): 577–586, 2020.

Rafael Gómez-Bombarelli, Jennifer N. Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D. Hirzel, Ryan P. Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS Central Science*, 4(2):268–276, 2018. doi: 10.1021/acscentsci.7b00572. URL `https://doi.org/10.1021/acscentsci.7b00572`. PMID: 29532027.

Dan Hendrycks and Kevin Gimpel. Bridging nonlinearities and stochastic regularizers with gaussian error linear units. *CoRR*, abs/1606.08415, 2016. URL `http://arxiv.org/abs/1606.08415`.

Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. In *International conference on learning representations*, 2017.

Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation. In *International conference on machine learning*, pages 2323–2332. PMLR, 2018.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Peter M. Kogge and Harold S. Stone. A parallel algorithm for the efficient solution of a general class of recurrence equations. *IEEE Transactions on Computers*, 22(8):786–793, 1973.

Yibo Lin, Zixuan Jiang, Jiaqi Gu, Wuxi Li, Shounak Dhar, Haoxing Ren, Brucek Khailany, and David Z. Pan. Dreamplace: Deep learning toolkit-enabled gpu acceleration for modern vlsi placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40:748–761, 2020.

Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe W. J. Jiang, Ebrahim M. Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Sungmin Bae, Azade Nazi, Jiwoo Pak, Andy Tong, Kavya Srinivasa, William Hang, Emre Tuncer, Anand Babu, Quoc V. Le, James Laudon, Richard Ho, Roger Carpenter, and Jeff Dean. Chip placement with deep reinforcement learning. *CoRR*, abs/2004.10746, 2020. URL `https://arxiv.org/abs/2004.10746`.

Takayuki Moto and Mineo Kaneko. Prefix sequence: Optimization of parallel prefix adders using simulated annealing. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, 2018.

Anh Nguyen, Alexey Dosovitskiy, Jason Yosinski, Thomas Brox, and Jeff Clune. Synthesizing the preferred inputs for neurons in neural networks via deep generator networks. *Advances in neural information processing systems*, 29, 2016.

Rajarshi Roy, Jonathan Raiman, Neel Kant, Ilyas Elkin, Robert Kirby, Michael Siu, Stuart Oberman, Saad Godil, and Bryan Catanzaro. Prefixrl: Optimization of parallel prefix circuits using deep reinforcement learning. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 853–858, 2021. doi: 10.1109/DAC18074.2021.9586094.

Subhendu Roy, Mihir R. Choudhury, Ruchir Puri, and David Z. Pan. Towards optimal performance-area trade-off in adders by synthesis of parallel prefix structures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33(10):1517–1530, 2014.

J. Sklansky. Conditional-sum addition logic. *Ire Transactions on Electronic Computers*, 9 (2):226–231, 1960.

Jialin Song, Rajarshi Roy, Jonathan Raiman, Robert Kirby, Neel Kant, Saad Godil, and Bryan Catanzaro. Multi-objective reinforcement learning with adaptive pareto reset for prefix adder design. *Workshop on ML for Systems at NeurIPS*, 2022.

Austin Tripp, Erik Daxberger, and José Miguel Hernández-Lobato. Sample-efficient optimization in the latent space of deep generative models via weighted retraining. *Advances in Neural Information Processing Systems*, 33:11259–11272, 2020.

Hanrui Wang, Jiacheng Yang, Hae-Seung Lee, and Song Han. Learning to design circuits, 2020.

5. Appendix

5.1 Background

In this section, we provide a brief introduction to important concepts in digital design for a machine learning audience. For a more complete exposition, please refer to the appendix.

Many kinds of circuits, notably including binary adders, can be compactly represented as *prefix graphs*, which describe the pattern of carries within the circuit. The design space of prefix graphs is large—$O(2^{N^2})$ for N-bit designs—and expresses tradeoffs between circuit area and delay, the two main desiderata. For example, the ripple-carry structure represents schoolbook addition, computing carries one bit at a time, and has the lowest possible area but is relatively slow. Faster adders such as Sklansky (Sklansky, 1960) and Kogge-Stone (Kogge and Stone, 1973) compute redundant carry bits, which enables some parallelism at the cost of area. While regular structures minimizing analytical properties like graph depth and connectivity are well-known, the actual delay of a fully synthesized and laid-out circuit depends in a complicated way on many other physical factors, so designing these circuits remains an important industrial problem.

Because real-world designs may have different requirements for area and latency, we define a scalar cost function $f(x) = \omega \cdot \text{delay}(x) + (1 - \omega) \cdot \text{area}(x)$. We call ω the *delay weight*, a hyperparameter trading off these competing goals. In the cost function, we measured a circuit's total area in square microns divided by 100 and the delay of its longest ("critical") path in nanoseconds multiplied by 10, as we found this yielded smooth changes in optimization as ω was swept from 0 to 1. To ensure our results generalize, we conducted our experiments for $\omega \in \{0.33, 0.66, 0.95\}$ and for 16-, 32-, and 64-bit adders.

Prefix graphs are translated into physical circuits through cell mapping (which translates the logical graph into a list of electrical components with a lookup table), physical synthesis, and layout. In this work we experiment exclusively with binary adders, but the algorithm we describe could straightforwardly apply to any parallel prefix circuit by altering the cell mapping process.

5.2 Related work

5.2.1 Machine learning for EDA

The most closely related work to ours is Roy et al. (2021), which also optimizes parallel prefix adders with deep learning. While their reinforcement learning (RL) approach outperforms conventional tools, we show that RL is hindered by the difficulty of searching directly in the input space. In head-to-head comparison (subsection 3.2), CircuitVAE is typically more than twice as data-efficient as RL due to learning its own well-structured search space. Classical approaches to this problem include heuristic search (Moto and Kaneko, 2018) and pruning (Roy et al., 2014) methods.

Several works have explored using machine learning for other parts of the electronic design automation (EDA) process. These include Mirhoseini et al. (2020) and Lin et al. (2020), who use deep learning for chip placement, and Wang et al. (2020) who use RL to design analog rather than digital circuits.

We build on the open-source EDA tools OpenROAD (Ajayi et al., 2019) and OpenPhySyn (Agiza and Reda, 2020), without which this work would not have been possible.

5.2.2 Latent-space optimization

CircuitVAE employs latent-space optimization (LSO), a method which has recently become popular for black-box optimization, most notably in the field of chemical design (Gómez-Bombarelli et al., 2018; Jin et al., 2018; Tripp et al., 2020). LSO consists of learning a latent-variable generative model over input structures, together with a neural predictor of the cost function which serves to shape the latent space and may be used for optimization. The latent space acts as a learned continuous search space, typically grouping semantically similar inputs and enabling continuous search algorithms. While many improvements to this scheme have been recently proposed, we build on the framework of Tripp et al. (2020), which interleaves optimization with retraining the generative model on new data.

While some LSO techniques optimize by gradient descent through the cost predictor, recently Bayesian optimization (BO) has been the preferred approach (Jin et al., 2018). In this work, we demonstrate that naive gradient descent suffers from over-optimizing the cost predictor far from the data manifold, yielding points with low predicted costs but high actual costs. However, we introduce two techniques to address this problem and show that, once appropriately regularized, gradient descent can outperform BO by a wide margin.

5.3 Dataset

We release a dataset of circuits designed by CircuitVAE, together with their latent representations and simulation results. This dataset would be appropriate for reproducing the figures in this work, developing methods that predict circuit properties, and analyzing the latent circuit representations that CircuitVAE learns. The dataset format is documented in Table 1.

Due to the file size and anonymity constraints of the NeurIPS supplement, we only release data from one experiment at this time. This experiment contains 64-bit adders designed by CircuitVAE with a delay weight of 0.95, a starting dataset of 30 GA generations, and a random seed of 1. Following de-anonymization, we will release all the remaining data.

Table 1: Format of released data

Column	Description
`index`	Number of syntheses before this circuit
`outer_loop`	Round of training and optimization this circuit came from
`step`	Gradient step of optimization this circuit was synthesized at
`batch_idx`	Which latent trajectory in the batch this circuit belonged to
`bitvector`	The circuit's prefix graph, represented as a bitvector (see subsection 5.4)
`latent`	CircuitVAE's latent encoding of this circuit, a 128-vector
`prior_logprob`	The log-probability of this latent under the variational prior
`true_score`	The circuit's cost according to OpenPhySyn at this delay weight
`predicted_score`	The circuit's cost as predicted by CircuitVAE's cost predictor

5.4 Circuit synthesis

In this section, we continue our discussion of the circuit synthesis flow used in this work in more detail.

Prefix graphs compactly represent a circuit's design in terms of carry *generation* and *propagation* Brent and Kung (1982). Each bit span `i:j` is associated with a generate bit $g_{i,j}$ and a propagate bit $p_{i,j}$. For all $i = 1 \ldots N$, computing the input bits $g_{i,i}$ and $p_{i,i}$ is straightforward; furthermore, given the output bits $(g_{1,1}; p_{1,1}) \ldots (g_{N,1}, p_{N,1})$, computing the final carries and summand is easy. Intermediate values may be computed recursively: $(g_{i,j}; p_{i,j}) = (g_{i,x}; p_{i,x}) \circ (g_{x-1,j}; p_{x-1,j})$ where $i \geq x > j$ and $\circ$ is the carry operator Brent and Kung describe. A prefix graph is exactly a tree determining the association order of $(g_{i,i}; p_{i,i}) \circ \ldots \circ (g_{1,1}; p_{1,1})$ for each $i = 1, \ldots, N$.

In the dataset we release, we compactly represent prefix graphs as *bitvectors* with one bit per possible node in the graph. For input to our CNN encoder, we reshape the bitvector into a matrix in $\{0, 1\}^{N \times N}$ where the upper triangular holds the bitvector values and the lower triangular holds zeroes; the CNN decoder predicts logits of this shape, and we extract the upper triangular to predict a bitvector. We found that this representation approximately colocates bits which are closely connected.

Before synthesizing a predicted bitvector, any missing nodes implied by parentless child nodes are inserted in a process we refer to as *legalization*. By legalizing before scoring vectors, our cost predictor effectively sees legalization as part of the cost function, and does not need to separately learn which vectors are valid.

A prefix graph may be converted into a circuit netlist and synthesized at a particular clock target to determine its area and delay. Because of decisions made within the synthesis tool, a given circuit may achieve a range of areas and delays when synthesized at different clock targets. In practice, we first synthesized each circuit with clock targets 0.0ns and 10.0ns to determine upper and lower bounds on achievable delay, and then synthesize twice more at clock targets linearly interpolated 4% and 36% between these bounds. We then fit a cubic interpolator to these four (area, delay) tuples to predict the full curve of area and delay, and score the circuit based on the minimum cost along this curve according to our given delay weight. We found this scheme predicted synthesis results almost perfectly at much lower cost than computing the entire curve, so we benchmarked all methods in this way.

5.5 Model architecture and hyperparameters

In this section, we document CircuitVAE's model architecture. All search, training, and model hyperparameters are listed in Table 2.

CircuitVAE uses a fully-convolutional encoder and decoder, with linear layers to map to and from the 128-dimensional latent space. Each trunk consists of four residual blocks, each having two 5x5 convolutions; these sizes were picked to give output units a full receptive field. We use GELU Hendrycks and Gimpel (2016) as the activation function and do not use batch or layer normalization. The latent space uses a diagonal unit normal prior. The cost predictor is an MLP on top of the latent vector, with one hidden layer of 32 units, followed by GELU and a linear scalar predictor. The encoder and decoder each have approximately 1M parameters, and the score predictor has about 4000. The input to the encoder is a bitmatrix as described in subsection 5.4, augmented with binary positional encodings indicating the location of input and output nodes.

Table 2: Hyperparameters

Parameter	Value	Description
Experimental		
Bitwidth	16, 32, 64	Sizes of circuits designed
Delay weight	0.33, 0.66, 0.95	Sensitivity of objective to delay vs area
GA generations	1, 5, 10, 30	Number of GA generations used as initial data
Search		
Steps	600	Total latent gradient steps
Synthesis period	100	Gradient steps between synthesizing circuits
Batch size	96	Number of parallel latent trajectories per search
γ	0.01 - 0.1	Strength of prior regularization
Learning rate	0.1	
Training		
Search period	5000	Training steps between latent optimization rounds
Batch size	64	
Learning rate	0.0002	
Gradient clipping	1.0	
Gradient skipping	400.0	Skip updates with pre-clip gradient norm $\geq$ 400.0
AE loss weight	0.03	Weight of autoencoding term in β-VAE loss
β	0.01	Weight of KL term in β-VAE loss
λ	10.0	Weight of cost prediction loss
KL warmup	2000	Steps to linearly warmup KL loss from 0
k	0.001	Data reweighting coefficient
Model		
Latent dimension	128	
CNN filters	64	
CNN kernel size	5	
CNN blocks	4	ResNetV2 residual blocks per encoder and decoder
f_π depth	1	Hidden layers in cost predictor
f_π width	32	Width of cost predictor hidden layers

5.6 Ablations

We ablated each of the components of CircuitVAE to understand their individual contributions. All of these experiments were conducted on 32-bit adders, with a delay weight of 0.66 and the largest initial dataset. In Figure 3, we tested:

- Removing data reweighting (Tripp et al., 2020), which leads training to get stuck when new datapoints have a negligible impact on the overall distribution.

- Replacing the cost-weighted latent distribution with the prior or the latent encoding of Sklansky. Starting the search from a good adder (Sklansky) outperforms sampling from the prior, but both underperform our adaptive initialization.

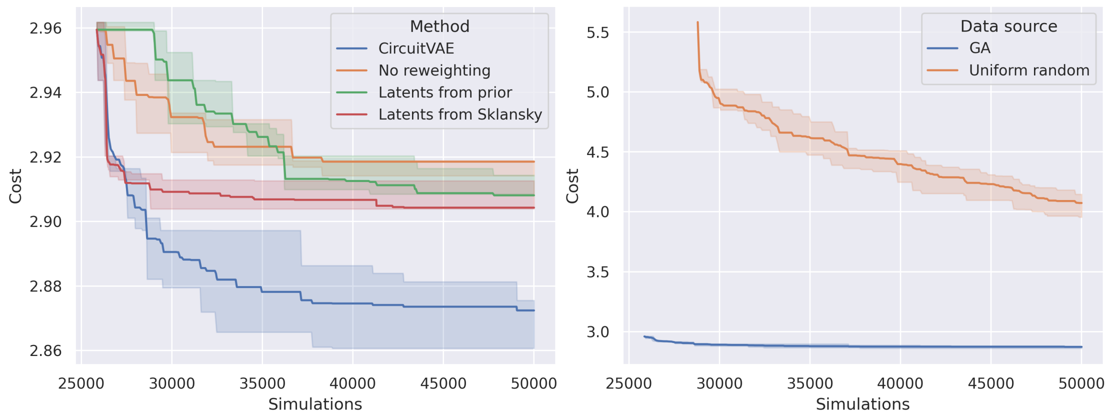

Figure 3: Ablating search and training methods (left) and data source (right).

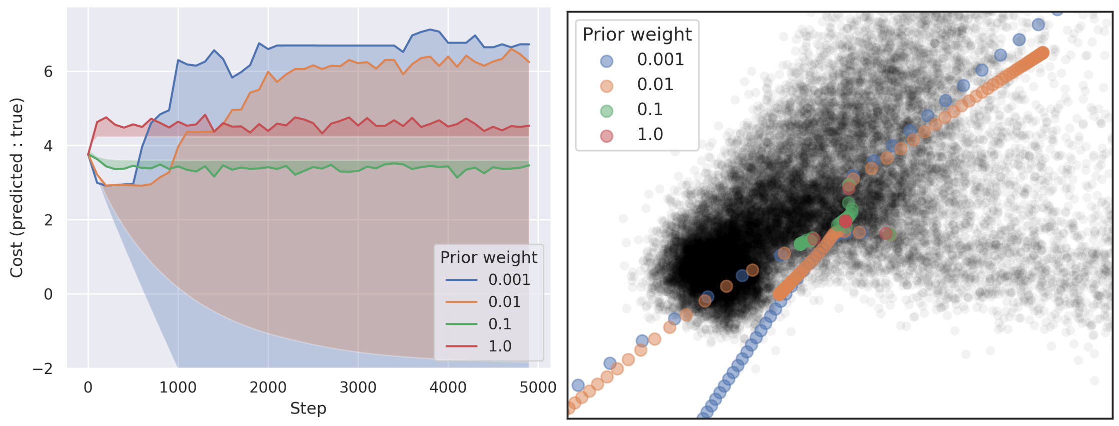

Figure 4: The effect of changing prior weight γ on cost (left) and latent search trajectories (right). Shaded regions show overfitting when the true cost (top) exceeds the model's prediction (bottom).

- Replacing the initial dataset with uniformly random adders rather than the first 30 generations of GA, which performs poorly because uniformly random adders are typically low-quality.

In Figure 4, we examined the ability to control search by controlling the prior regularization term γ. At low values of γ (blue and orange), latent trajectories quickly exit the region around the training data (gray) and overfit the cost predictor, yielding much higher costs than the model predicts. At higher values (green and red), trajectories stay near the origin, which prevents overfitting but limits exploration and sample diversity. We found the best results from sampling values of γ per latent trajectory log-uniformly between 0.01 and 0.1, and used this setting for all other experiments.

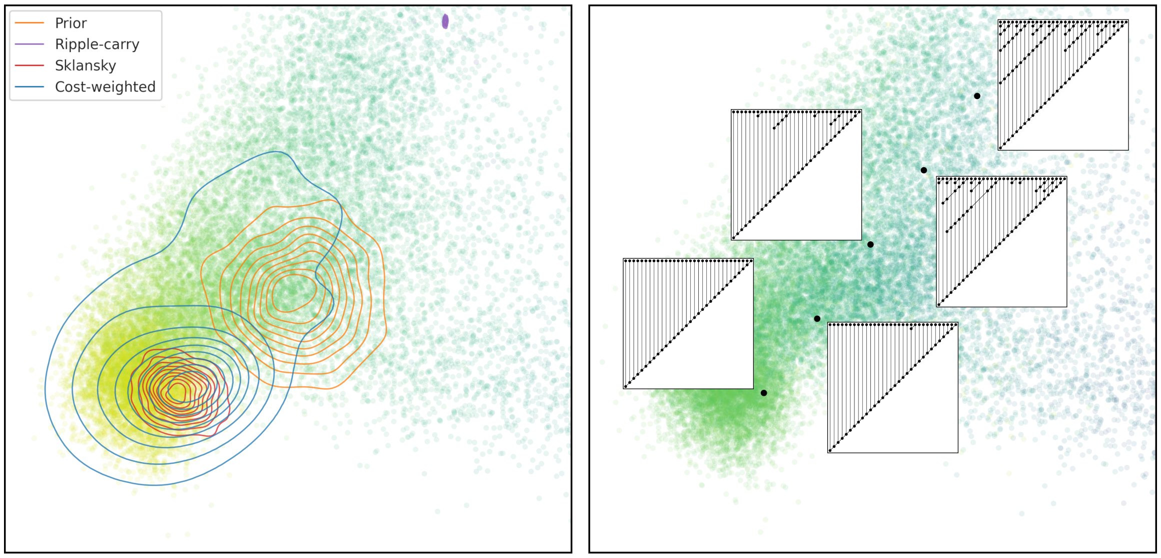

Figure 5: Left: Kernel density estimate plots of latent distributions. Right: interpolating between two circuits. Background points are training set adders, colored by cost (left) and area (right); lighter is better.

5.7 Analyzing CircuitVAE's latent space

To investigate properties of CircuitVAE's learned search space, we visualize it in two dimensions in Figure 5. We reduced 128-dimensional latent vectors down to two dimensions with PCA trained on encodings of training set points, and visualized dataset points and selected circuits in this space.

When the points are colored by cost (left) or area (right), it is clear that the latent space is self-organized according to both the objective function and physical properties of the circuits. This is in stark contrast to the input space: represented as discrete graphs, nearby circuits (defined by having many nodes in common) may have very different costs and physical properties, since adding a single node may change the critical path. While discrete search methods like GA and RL suffer from this poorly-structured search space, CircuitVAE learns its own space in which optimization is easy.

Visualizing different latent distributions (left) illustrates the benefits of our cost-weighted sampling approach. Latents sampled from the prior are diverse but have subpar cost, while latents sampled from the posterior of specific circuits (ripple-carry and Sklansky) are typically higher quality but much less diverse. Our cost-weighted distribution is diverse, covering much of the latent space, but is biased towards lower-cost adders.

5.8 GA baseline

Our genetic algorithm (GA) baseline used a standard genetic algorithm with crossover and mutation operations Davis (1991). We used a population size of 1000 for each generation. The individuals of the population were the bitvector representations of flattened prefix graphs with each bit representing whether or not a prefix node was present. A single mutation was represented by a random inversion of a single Boolean value. Each individual was initialized by randomly choosing either the ripple-carry or Sklansky prefix structure and performing 200 random mutations. We assigned each individual a fitness score as negative cost as described in subsection 5.4. Each successive generation was generated by taking the top 50% most fit individuals from the previous generation and from them generating a new population. The new population was generated 40% by a mutation procedure, 40% by a crossover procedure, 10% by preserving the top individuals from the previous generation unchanged and 10% by the random initialization procedure mentioned above. The mutation procedure created a new individual by randomly sampling a single parent and then performing up to 50 random mutations. The crossover procedure created a new individual by randomly sampling two parents and then randomly choosing one of those two parents to supply the value at each node location.