Efficient Continuous Logic Optimization with Diffusion Model

Yikang Ouyang[1*], Xiaofei Yu[1*], Jiadong Zhu[1], Tinghuan Chen[2], Yuzhe Ma[1†]
[1]The Hong Kong University of Science and Technology (Guangzhou)
[2]The Chinese University of Hong Kong, Shenzhen
Email: yuzhema@hkust-gz.edu.cn

2025 62nd ACM/IEEE Design Automation Conference (DAC) | 979-8-3315-0304-8/25/$31.00 ©2025 IEEE | DOI: 10.1109/DAC63849.2025.11133282

***Abstract*—The logic synthesis optimization flow is crucial to the quality of results (QoR), which applies a sequence of transformations to a design. Recently, there has been a growing focus on the automatic optimization of synthesis flows to improve QoR, utilizing techniques such as Bayesian optimization and reinforcement learning, which may fall short in efficiency due to the exponentially large search space. In contrast, continuous optimization offers notable efficiency advantages by leveraging the explicit gradient. However, despite its potential, several significant concerns remain to be addressed. On one hand, it is essential to obtain a reliable gradient. On the other hand, a major challenge arises from the fact that searching within a continuous space can yield solutions that deviate from feasible ones. In this paper, we propose an efficient approach to optimize synthesis sequences within a continuous latent space. Specifically, the gradient information is derived from a QoR surrogate model, while the discrepancies between solutions and feasible transformations are minimized by a diffusion model. Experimental results on extensive benchmarks demonstrate that the proposed method not only achieves lower area and delay but also improves efficiency by 5X to 130X, compared with previous methods.**

I. Introduction

Logic synthesis is an important process that transforms circuits from register-transfer level design to gate-level netlist. Previous works usually represent circuits as AIG (And-Inverter Graph) [1] or similar boolean networks like MIG [2]. Then the synthesis process applies a sequence of logic optimization transformations like *rewrite*, *restructure* to circuits [1], [3]. For the same circuit, different sequences can lead to a variation in quality-of-result (QoR) of up to 40% [4]. Thus, it is imperative to optimize those synthesis sequences to improve the quality of synthesized netlists.

Previous works are devoted to optimizing sequences with techniques like reinforcement learning (RL) [5]–[7], multi-arm bandit (MAB) [8], and discrete Bayesian optimization (BO) [9]. Both RL and MAB methods treat the sequence optimization task as a sequential decision-making problem. They explore the probability of which transformation to take by updating the Q-function, policy function, etc. The discrete BO [9] has to sample sequences by discretely altering the transformations for evaluating the acquisition function. While automated methods yield significant QoR improvements over general heuristics, their dependence on extensive discrete searches and iterative nested updates leads to considerable

*Equal contribution
†Corresponding author

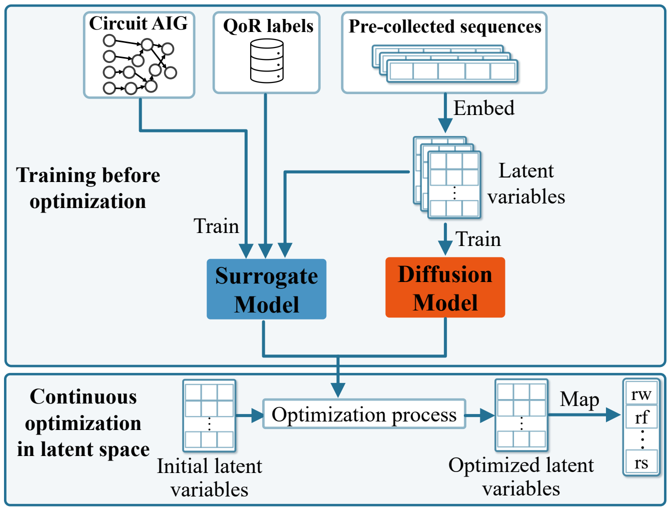


Fig. 1 Overview of the proposed framework. Upper part: Pre-training a surrogate model and a diffusion model. Lower part: Continuous optimization of sequence in latent space.

overhead and inefficiency during the optimization process, particularly as the number of transformation options increases and the sequence lengthens. This raises a pertinent question: Can we directly optimize sequences without searching in the discrete space to achieve faster turnaround time?

In contrast to discrete search, continuous search transforms the discrete search space into continuous latent space and leverages numerical optimization techniques like gradient descent to directly optimize the objective function guided by the gradient from a surrogate model [10]. Upon convergence, the final solution is retrieved by mapping the latent representation in continuous latent space back to the original space.

For synthesis sequence optimization, a deep learning-based surrogate model should first be built, which takes the circuit and sequence embeddings as input and predicts the QoR. However, it is observed that solely relying on the gradient from the surrogate model is problematic [11], [12]. Note that the final sequence for synthesis should be retrieved based on the optimized sequence embedding in the latent space. Unfortunately, even though the continuous optimization converges in the first phase, the sequence may hardly be retrieved due to a large discrepancy between the obtained embeddings and the feasible transformation embeddings. Some previous works explored to conduct an additional mapping based on a distance metric [13] or a neural network-based decoder [14]. Nevertheless, neither of these methods can achieve the desired performance, as revealed in [15]–[17], since the additional mapping operation is already out of the optimization loop, offering no assurance against performance degradation. Therefore, to facilitate the retrieval of final transformations from continuous space, it is essential to ensure that the embedding discrepancy between the final solutions and feasible transformations is minimized, such that the final optimized sequence can be instantly retrieved.

To address the above challenges, we incorporate a novel generative diffusion model together with a surrogate model into the continuous logic optimization framework. The overview of it is shown in Fig. 1. Particularly, the diffusion model is designed to minimize that discrepancy. Different from the previous two-stage workflows, the proposed method can seamlessly integrate the two stages into a unified stage. Moreover, all the computations are within the optimization loop, which provides a higher level of assurance regarding the quality of the final solution. The discrepancy is minimized explicitly through a series of denoising steps, each of which is performed together with a gradient descent step. Upon convergence, the discrepancy between the optimized embedding and the feasible embedding is minimized, and thus the final optimized sequence can be retrieved instantly without degrading QoR. Our contributions are as follows:

1) To the best of our knowledge, it is the first work to propose a continuous optimization approach for the discrete logic sequence optimization problem.
2) We optimize the QoR together with the discrepancy between optimized embeddings and feasible transformations to achieve a unified framework.
3) We leverage the gradient obtained from a surrogate model for QoR optimization and the denoising steps based on a diffusion model for discrepancy minimization.
4) Experimental results demonstrate that the proposed continuous optimization framework achieves substantial improvements in runtime and QoR compared with previous baseline methods on logic optimization.

II. Preliminaries

Some frequently used notations are shown in TABLE I for a clearer illustration.

A. Logic Synthesis Sequence Optimization

In logic synthesis, a sequence of logic transformations is applied to the circuit to achieve a better QoR. A sequence $\boldsymbol{s}$ with length L is composed of transformations $[s_1, \dots s_L]$, where each transformation s_i belongs to an available set $\mathbb{S}$, i.e., $s_i \in \mathbb{S}$. In our work, $\mathbb{S} = \{\texttt{rw, rwz, rf, rfz, rs, rsz, b}\}$. These transformations have different effects on circuits represented in AIG by reducing the number of nodes and reducing the levels. Although ABC [1] is a relatively fast synthesis tool, the exponentially increasing search space ($|\mathbb{S}|^L$) still hinders efficient optimization. Given the fact that longer sequences may not necessarily lead to better QoR [8], we focus on finding a sequence with a fixed length L that leads to minimal area or delay, or both. It is formulated as:

TABLE I Notations in this work.

Notation	Explanation
$\boldsymbol{s}$	A sequence of discrete transformations with length L.
$g(\cdot)$	The function that embeds sequence into latent space.
$\boldsymbol{x}$	The sequence of transformation embeddings.
$\boldsymbol{x}_t$	The sequence of latent variables at denoising step t. We use $t = T$ to denote the start of denoising steps and $t = 0$ as the end to align with notations in diffusion models.
$\hat{F}(\cdot)$	Surrogate model to predict QoR.
$\boldsymbol{\epsilon}_t$	The added noise at the t-th step in diffusion process.
$\boldsymbol{\epsilon}_\theta(\cdot, t)$	The predicted noise at the t-th step by the diffusion model.
$\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$	A Gaussian distribution with mean $\boldsymbol{\mu}$ and variance $\boldsymbol{\sigma}^2$.
$\boldsymbol{\sigma}_t, \boldsymbol{\alpha}_t$	Some constants dependent on t in the diffusion model.

$$\begin{aligned} \min_{\boldsymbol{s}} \quad & F(\boldsymbol{s}) \\ \text{s.t.} \quad & s_i \in \mathbb{S}, \quad i = 1, 2 \dots, L. \end{aligned} \tag{1}$$

B. Diffusion Models

Diffusion models have become the mainstream generative models [18]–[20]. Such popularity is mainly due to their extraordinary ability to capture the distribution of training data and then generate data with a similar distribution as the training data. Specifically, diffusion models will convert the training data into pure Gaussian noise by gradually adding Gaussian noise $\boldsymbol{\epsilon}_t$. Then they will learn to restore the data from pure Gaussian through denoising steps by removing noise $\boldsymbol{\epsilon}_\theta(\boldsymbol{x}, t)$ predicted by a neural network parameterized with θ.

III. Methodologies

Our continuous optimization framework involves two models, including a surrogate model to predict QoR and a diffusion model to learn the distribution of feasible transformation embeddings. The optimization process will optimize latent variables with both the gradient from the surrogate model and the denoising update from the diffusion model. Finally, it maps optimized latent variables back to discrete transformations. We illustrate the details in the following sections.

A. Training Surrogate Model

Since QoR after logic synthesis cannot be analytically formulated, a surrogate model is required to bridge the gap. Moreover, it is imperative that the surrogate model can generate the gradient information to enable continuous optimization. Recently, various works have investigated deep learning-based QoR prediction for logic synthesis [4], [21], [22]. Generally, these frameworks will first encode the sequence into an embedding in the latent space, i.e., $\boldsymbol{x} = g(\boldsymbol{s})$. Then a deep neural network is trained to predict the QoR. For each design, given a dataset comprising N data points that include sequences and QoR labels, the surrogate model is trained by minimizing the mean squared error between the predicted values and the corresponding labels as shown in the following equation:

$$\min \mathcal{L} = \frac{1}{N} \sum_{j=1}^{N} (\hat{y}^{(j)} - y^{(j)})^2, \tag{2}$$

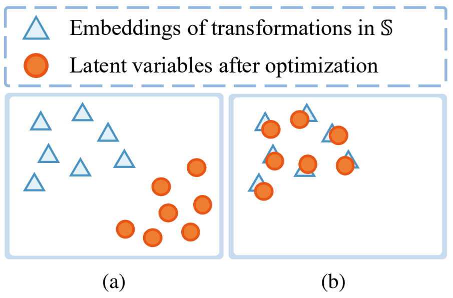


Fig. 2 (a) Latent variables optimized only w.r.t QoR. (b) Latent variables optimized w.r.t QoR and discrepancy.

where the $\hat{y}^{(j)}$ and $y^{(j)}$ are the predicted QoR and the label of the sample j, respectively.

B. Optimizing Sequences in Continuous Space

After a surrogate model is trained to predict the delay and/or area after synthesis, we can minimize the predicted delay or area by finding a sequence embedding in the latent space. For simplicity, we use $\hat{F}(\cdot)$ to represent the surrogate model, and $\boldsymbol{x}$ to represent the embedded sequence to be optimized.

$$\boldsymbol{x}^* = \underset{\boldsymbol{x}}{\operatorname{argmin}}\, \hat{F}(\boldsymbol{x}). \tag{3}$$

Since the surrogate model is based on a DNN, we can get the gradient by back-propagation. However, relying on gradient may lead to a large discrepancy between optimized latent variables and embeddings of feasible transformations. Fig. 2 presents an illustrative example that visualizes the embeddings in latent space by t-SNE [23]. It poses a great challenge for sequence retrieval, i.e., mapping the embedding in latent space to the real sequence for logic synthesis, given the results with large discrepancy shown in Fig. 2(a).

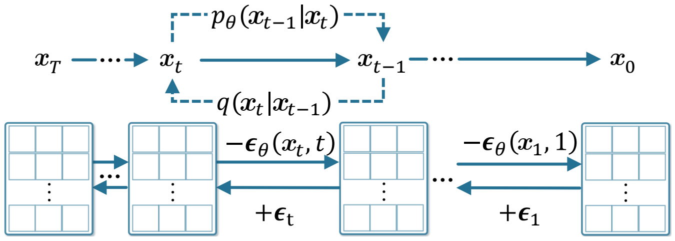
In this work, we propose to bridge the gap by introducing another auxiliary objective $H(\boldsymbol{x})$ that minimizes the discrepancy. Specifically, minimizing $H(\boldsymbol{x})$ implies that the optimized embedding $\boldsymbol{x}^*$ follows the same distribution as valid transformation embeddings so that we can eliminate the additional yet intricate mapping to retrieve the sequence, as depicted in Fig. 2(b). To this end, we reformulate our optimization problem in the latent space as:

$$\boldsymbol{x}^* = \underset{\boldsymbol{x}}{\operatorname{argmin}}[\hat{F}(\boldsymbol{x}) + H(\boldsymbol{x})], \tag{4}$$

where $H(\boldsymbol{x})$ represents the discrepancy between an embedding $\boldsymbol{x}$ and the valid transformation embeddings in latent space.

A widely used option for $H(\boldsymbol{x})$ is the negative log-likelihood (NLL). It is a commonly used objective to be optimized in generative models to ensure the generated data follows the same distribution as the training data. If minimized, the optimized latent variables should be in the distribution of the feasible transformation embeddings and the discrepancies between them are minimized. Ideally, we can directly compute the gradient of the objective in Equation (4) w.r.t $\boldsymbol{x}$, i.e., $\nabla_{\boldsymbol{x}}\hat{F}(\boldsymbol{x}) + \nabla_{\boldsymbol{x}}H(\boldsymbol{x})$.

The gradient $\nabla_{\boldsymbol{x}}\hat{F}(\boldsymbol{x})$ is easy to obtain since it can be computed via back-propagation. However, minimizing NLL through gradient descent leads to tractability issues for computing $\nabla_{\boldsymbol{x}}H(\boldsymbol{x})$. Instead, a variational bound is optimized [24], which is the mainstream approach to train generative models, including diffusion models [18]. Therefore, minimizing $H(\boldsymbol{x})$ can be resolved through training a diffusion model.


$p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)$
$\boldsymbol{x}_T$
$\boldsymbol{x}_t$
$\boldsymbol{x}_{t-1}$
$\boldsymbol{x}_0$
$q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})$
$-\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t, t)$
$+\boldsymbol{\epsilon}_t$
$-\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_1, 1)$
$+\boldsymbol{\epsilon}_1$
Fig. 3 The diffusion process (from right to left) and denoising process (from left to right) of a diffusion model.

C. Optimization with Diffusion Model

Diffusion models have shown extraordinary abilities in generating high-quality data in the training set distribution by minimizing a variational bound. The diffusion model encompasses a diffusion process and a denoising process as shown in Fig. 3. The diffusion process gradually adds noise to the feasible transformation embeddings as shown in Equation (5) with $\boldsymbol{\alpha}_t$ as some constants.

$$\boldsymbol{x}_t = \sqrt{\boldsymbol{\alpha}_t}\boldsymbol{x}_{t-1} + \sqrt{1-\boldsymbol{\alpha}_t}\boldsymbol{\epsilon}_t, \quad \boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}). \tag{5}$$

This diffusion process converts the original data into pure Gaussian noise following the conditional distribution:

$$q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}) = \mathcal{N}(\boldsymbol{x}_t; \sqrt{\boldsymbol{\alpha}_t}\boldsymbol{x}_{t-1}, \sqrt{1-\boldsymbol{\alpha}_t}\boldsymbol{I}). \tag{6}$$

To generate in-distribution data, the denoising process tries to remove the same scale of Gaussian noise $\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t, t) \sim \mathcal{N}(\boldsymbol{\mu}_\theta(\boldsymbol{x}_t, t), \boldsymbol{\sigma}_t^2\boldsymbol{I})$ generated with a mean $\boldsymbol{\mu}_\theta(\boldsymbol{x}_t, t)$ parameterized by θ:

$$\boldsymbol{x}_{t-1} = \frac{1}{\sqrt{\boldsymbol{\alpha}_t}}(\boldsymbol{x}_t - \frac{1-\boldsymbol{\alpha}_t}{\sqrt{1-\bar{\boldsymbol{\alpha}}_t}}\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t, t)) + \boldsymbol{\sigma}_t\boldsymbol{z}, \tag{7}$$

where $\boldsymbol{\sigma}_t$ are some constants, $\bar{\boldsymbol{\alpha}}_t = \sum_{s=1}^{t}\boldsymbol{\alpha}_s$ and $\boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$. This process follows the conditional distribution:

$$p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t) = \mathcal{N}(\boldsymbol{x}_{t-1}; \boldsymbol{\mu}_\theta(\boldsymbol{x}_t, t), \boldsymbol{\sigma}_t^2\boldsymbol{I}). \tag{8}$$

The objective of training the diffusion model is to minimize the variational bound over the negative log-likelihood through the denoising process, as shown in Equation (9).

$$\min \mathcal{L}_d = \mathbb{E}[-\log q(\boldsymbol{x}_T) - \sum_{t=1}^{T}\log\frac{p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)}{q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})}] \geq \mathbb{E}[-\log p_\theta(\boldsymbol{x}_0)]. \tag{9}$$

Since the diffusion process with $q(\cdot)$ is irrelevant to parameters θ of the neural network, optimizing the loss function in Equation (9) is equivalent to learning a denoising process with parameters θ that can minimize the variational bound over the NLL. In practice, the diffusion model is trained by minimizing

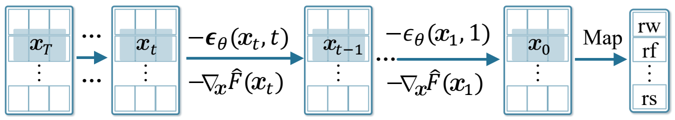


Fig. 4 The optimization process that gradually subtracts gradient from the surrogate model and noise from the diffusion model.

Algorithm 1 Training Diffusion Model

Input: N embedded latent sequences of transformations.
Output: A trained diffusion model parameterized by θ.

```
1: repeat
2:     $t \leftarrow \text{Random}(1, T)$, $j \leftarrow \text{Random}(1, N)$.
3:     $\bar{\alpha}_t \leftarrow \sum_{s=1}^{t} \alpha_s$.
4:     $\mathcal{L}_d \leftarrow \text{Loss}(t, j, x_0^{(j)})$.        ▷ Equation (10).
5:     $\theta \leftarrow \text{Update}(\theta, \mathcal{L}_d)$.
6: until Diffusion model converges.
```

the difference between the added noise and the predicted noise on the N pre-collected embedded sequences [18]:

$$\min \mathcal{L}_d = \mathbb{E}_{t,j,x_0,\epsilon_t}[||\epsilon_t - \epsilon_\theta(\sqrt{\bar{\alpha}_t} x_0^{(j)} + \sqrt{1-\bar{\alpha}_t}\epsilon_t, t)||^2], \tag{10}$$

where $x_0^{(j)}$ means the original embedded sequence of sample j. The training details are also illustrated in Algorithm 1. The predicted noise $\epsilon_\theta(x_t, t)$ with a mean $\mu_\theta(x_t, t)$ that dependent on θ is updated by gradient descent on computed loss as shown in Line 5.

Given a well-trained diffusion model, the generated latent variable x_0 through the denoising process should be in the distribution of embeddings of feasible transformations. The discrepancies between them in the latent space are also minimized. To minimize $\hat{F}(x) + H(x)$ in Equation (4), the corresponding update rule can be calculated by back-propagation and Equation (7), respectively. Therefore, we can combine them as the optimization process starting from a sequence of initial latent variables following pure Gaussian distribution, i.e., $x_T \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$, and proceed with the iterative optimization in a continuous latent space as:

$$x_{t-1} = x_t - \eta(\epsilon_\theta(x_t, t) + \omega \nabla_x \hat{F}(x_t)) + \sigma_t z, \tag{11}$$

where the constant coefficients are combined into η and ω for brevity. This optimization process is also shown in Fig. 4. At each optimization step, we will subtract the predicted noise $\epsilon_\theta(x_t, t)$ computed by the diffusion model and the gradient w.r.t predicted QoR $\nabla_x \hat{F}(x_t)$.

Arguably, the surrogate model may not provide accurate prediction on a sequence consisting of *noisy* latent variables. Following [25], for each optimization step, we revert the *noisy* variables x_t to a noise-free version $\hat{x}_t$ to obtain a more accurate gradient to optimize the QoR. This can be done by the reparameterization trick used in [25], [26]:

$$\hat{x}_t = \frac{x_t - (\sqrt{1-\bar{\alpha}_t})\epsilon_\theta(x_t, t)}{\sqrt{\bar{\alpha}_t}}. \tag{12}$$

Algorithm 2 Continuous Logic Optimization

Input: Trained surrogate model and diffusion model; target circuit.
Output: Optimized sequence for logic synthesis on the target circuit.

```
1: Randomly sample a sequence of latent variables from
   Gaussian noise: $x_T \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$.
2: for $t = T$ to 1 do
3:     Run optimization step with Equation (13).
4: end for
5: Retrieve a sequence of transformations.   ▷ Section III-D.
```

Combining Equation (12) and Equation (11), we can have the final update rule for solving the problem in Equation (4):

$$x_{t-1} = x_t - \eta(\epsilon_\theta(x_t, t) + \omega \nabla_x \hat{F}(\hat{x}_t)) + \sigma_t z. \tag{13}$$

This equation illustrates our idea of optimizing sequences in continuous space with the optimized variables in the distribution of valid transformation embeddings. It generates latent variables in a sequence to minimize QoR and prevent them from having large discrepancies from the feasible transformation embeddings. The optimization process is illustrated in Algorithm 2.

D. Retrieval of Optimized Transformations

Upon convergence of the above optimization, we obtain the optimized x^* in latent space. Owing to the denoising process involved, the separate embeddings in x^* will be almost aligned with the embeddings of feasible transformation from $\mathbb{S}$ with minimal discrepancy. Therefore, the final optimized transformation sequence can be retrieved instantly. Our framework can be concluded in Algorithm 1. This optimized sequence will be sent to ABC [1] for validating the actual QoR.

IV. Experimental Results

In this section, we demonstrate the efficiency of our continuous optimization method with several baselines in terms of QoR and runtime. We also ablate the effectiveness of our continuous optimization with different choices of surrogate models and the necessity of the diffusion model.

A. Experiment Setup

Target circuits are taken from EPFL15 and ISCAS89 benchmarks [27], [28]. The PDK for technology mapping is the open-source ASAP7 PDK [29]. Our surrogate model is an MTL-based from [22], which consists of a 2-layer GNN, an LSTM, and two 2-layer attention modules. It is trained by 20000 randomly generated sequences synthesized by ABC [1]. Our diffusion model has a 1-D U-Net structure [18], [30]. It has 500 diffusion/denoising steps. The experiments are conducted on a machine with two AMD EPYC 7543 32-core CPUs and one Nvidia RTX 3090 GPU. We target optimizing synthesis sequences with length $L = 20$. For each circuit, we train a corresponding surrogate model and a diffusion model, which takes about 11 minutes and is a one-time effort.

We choose several state-of-the-art methods of logic optimization as baselines, including DRiLLS [5], abcRL [6],

TABLE II Delay and Area Comparison. Unit of area: μm^2. Unit of delay: ps.

Circuit	Original		DRiLLS [5]		abcRL [6]		BOiLS [9]		FlowTune [8]		**Ours**	
	Area	Delay	Area	Delay	Area	Delay	Area	Delay	Area	Delay	Area	Delay
adder	867.43	1006.42	807.56	1005.60	941.10	1004.28	945.74	1005.69	**797.05**	1004.28	**797.05**	**1003.60**
arbiter	**1299.64**	**347.26**	1301.25	**347.26**	**1299.64**	**347.26**	**1299.64**	**347.26**	**1299.64**	**347.26**	**1299.64**	**347.26**
bar	2077.62	**75.37**	1680.94	76.39	**1669.79**	**75.37**	1680.94	76.39	1680.94	**75.37**	**1669.79**	76.39
cavlc	213.22	76.18	235.02	74.17	206.06	73.36	229.57	76.18	208.65	**72.97**	**198.77**	73.77
ctrl	66.29	40.44	54.33	35.34	53.86	32.94	56.68	40.49	48.03	**31.09**	**41.78**	**31.09**
dec	**289.42**	**26.28**	**289.42**	**26.28**	**289.42**	**26.28**	**289.42**	**26.28**	**289.42**	**26.28**	**289.42**	**26.28**
div	12888.45	17251.69	12531.56	16871.04	12130.13	16766.32	13488.61	16898.22	13768.78	**16756.87**	**11900.00**	16765.37
hyp	176053.57	89237.98	168154.61	89319.01	**167755.80**	89207.74	181291.68	89269.65	176048.90	89237.98	168070.20	**89168.95**
i2c	556.44	73.08	491.11	61.86	491.81	**60.78**	549.74	**60.78**	469.11	61.11	**464.22**	**60.78**
int2float	90.45	71.06	91.16	70.00	75.98	71.06	85.82	71.06	**74.37**	**68.04**	78.10	71.06
log2	**20590.64**	1614.88	22084.77	1613.17	23092.16	1571.92	24947.17	1571.80	22412.07	**1570.75**	20997.60	1606.55
max	1968.73	896.81	1332.00	747.25	1369.65	681.15	1580.82	637.20	1340.26	657.73	**1317.22**	**612.07**
mem_ctrl	13377.19	439.94	11705.69	371.81	11138.74	360.04	10517.57	363.30	11016.25	360.52	**10289.00**	**353.37**
multiplier	**19738.85**	1100.52	21870.08	1098.25	21924.85	1049.30	23028.97	1093.97	22468.65	**1046.04**	19747.90	**1046.04**
priority	382.75	969.31	254.58	555.05	**208.43**	**331.00**	237.33	436.86	218.65	337.10	211.90	331.10
router	246.68	114.32	103.09	74.01	100.51	74.53	95.63	82.41	95.83	69.46	**82.35**	**67.57**
sin	4576.88	752.97	4478.85	726.67	4619.88	706.06	4896.39	714.22	4486.30	**702.59**	**3921.83**	704.74
sqrt	12109.51	21604.47	11683.06	21318.41	13991.26	21330.84	13685.81	21385.74	11736.63	**21316.52**	**11569.62**	21320.39
square	11284.68	984.70	13611.74	935.58	12658.78	934.25	13382.14	978.47	12637.56	977.54	**11086.65**	**929.45**
voter	17367.28	333.12	11398.64	294.34	11359.52	305.71	12254.71	290.33	**10986.33**	285.26	11150.58	**283.87**
c17	**3.73**	**18.52**	**3.73**	**18.52**	**3.73**	**18.52**	**3.73**	**18.52**	**3.73**	**18.52**	**3.73**	**18.52**
c432	165.77	128.00	137.13	119.37	**88.42**	118.82	135.29	124.42	112.19	119.55	88.67	**108.55**
c499	487.36	105.30	524.26	98.59	514.01	98.59	487.36	105.30	487.65	97.51	**480.45**	**97.25**
c880	207.21	92.22	230.74	91.97	221.61	91.09	226.78	94.59	**197.17**	90.54	206.74	**89.90**
c1355	494.16	107.81	524.88	100.12	557.81	97.51	507.74	114.78	502.96	**97.25**	**483.95**	99.89
c1908	465.92	142.06	452.96	130.87	436.59	131.66	447.51	131.73	**410.87**	129.25	426.20	**127.96**
c2670	631.64	88.95	522.35	94.14	506.68	88.95	554.93	99.59	491.30	86.28	**463.09**	**83.37**
c3540	871.97	175.82	705.59	164.89	692.83	160.31	777.00	169.21	**627.92**	158.88	677.16	**157.01**
c5315	1069.64	175.32	822.22	128.71	824.82	121.18	938.61	131.33	758.96	120.25	**726.85**	**116.76**
c6288	2379.73	424.29	2577.50	417.55	2594.44	**415.73**	2298.16	424.29	2546.76	416.58	**2269.26**	424.39
c7552	1635.09	151.14	1221.87	118.99	1072.60	107.16	1252.75	146.18	1058.71	105.75	**1027.64**	**103.44**
Mean	9821.22	4471.81	9415.57	4422.75	9448.09	4401.93	10070.14	4418.91	9386.73	4400.48	**9097.98**	**4396.99**
Ratio	1.079	1.017	1.035	1.006	1.038	1.001	1.107	1.005	1.032	1.001	**1.000**	**1.000**
Geo. mean	1212.81	319.57	1088.54	292.09	1056.72	281.97	1117.81	296.23	1029.10	279.15	**983.51**	**276.69**
Ratio	1.233	1.155	1.107	1.056	1.074	1.019	1.137	1.071	1.046	1.009	**1.000**	**1.000**

BOiLS [9], and FlowTune [8], all of which have open-source implementations. We spare all 64 CPU cores when running them for each circuit and one GPU if needed. We modify them meticulously so that both baseline methods and ours use the same set of transformations $\mathbb{S}$ to form a sequence with a length of 20 for a fair comparison. The numbers of optimization iterations of baselines are properly set according to their original papers. We repeat the evaluation 30 times for all methods and select the best sequence for comparison. For DRiLLS [5] and BOiLS [9], which support multi-objective optimization (delay and area), we evaluate them in a multi-objective scenario. For abcRL [6] and FlowTune [8], designed for single-objective optimization, we select the best results for area and delay separately.

B. QoR Comparison

Here we compare the delay and area after logic synthesis between our method and baseline methods. The results are shown in TABLE II. Note that the circuit scale in these benchmarks may vary significantly, and thus computing the mean value over all results may undermine the significance of small circuits (e.g., `ctrl`) and be biased towards large circuits (e.g., `hyp`). To mitigate this bias, we also list the geometric mean results, which reduce the disproportionate impact of substantially large values on the overall relative comparison. It can be seen that the proposed method can always dominate all baselines. Particularly, we can achieve a 4% to 13% reduction in the area and a 1% to 7% reduction in delay. This result validates that our continuous optimization is effective and can find superior sequences compared with baseline methods.

C. Runtime Comparison

For a fair comparison, for DRiLLS [5], abcRL [6], and BOiLS [9], we subtract the runtime of ABC [1] interleaved in the optimization loop so that the comparison is solely on the optimization algorithm itself. For our method, despite that model training may take time in our approach, it is a one-time effort. Therefore, the actual sequence optimization process in our approach is highly efficient. The runtime of our method compared with previous methods is shown in Fig. 5. It can be seen that the proposed continuous optimization method can achieve a 5X to 130X speedup compared with previous methods. In contrast, baseline methods like RL and Bayesian optimization require the model to be updated during the optimization process, and MAB searches a discrete space without explicit gradient information. Thus the efficiency of the baseline methods is not so high. The abcRL [6] is much slower as it requires building graphs from AIG being synthesized every after a transformation is applied. It is clearly shown that our continuous space optimization method is much more efficient than previous discrete optimization methods as it directly optimizes the sequence in a continuous space that avoids extensive discrete search.

D. Effectiveness of the Diffusion Model

We devise an ablation study to further validate the effectiveness of the diffusion model. To this end, we perform extensive

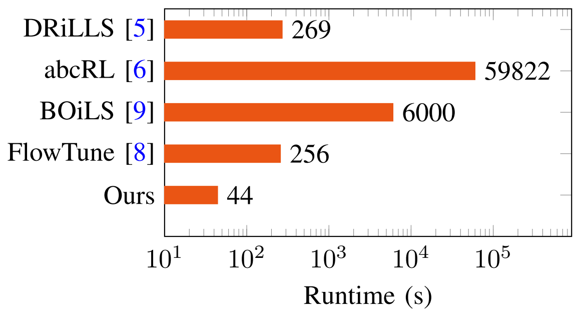


Fig. 5 Comparison of runtime.

validation not only on the MTL-based model [22], but also hybrid graph-based model (LOSTIN) [21] and convolutional neural network-based model (CNN) [4]. Regarding the counterpart without diffusion model, we perform the optimization using only the gradient from the surrogate model, i.e.,

$$\boldsymbol{x}_{t-1} = \boldsymbol{x}_t - \omega \nabla_{\boldsymbol{x}} \hat{F}(\boldsymbol{x}_t). \tag{14}$$

Fig. 6 presents the QoR results of continuous optimizations with various surrogate models, with and without the incorporation of the diffusion model. It can be seen that no matter which surrogate model we choose, incorporating the diffusion model can always result in a substantial performance improvement, which validates the general applicability of the proposed framework. Moreover, it is noted that solely utilizing a surrogate model to perform continuous optimization can hardly exceed the performance of baseline methods, e.g., FlowTune [8]. When the diffusion model is integrated, all the results can surpass the baselines. Please note that our continuous optimization method equipped with different surrogate models has approximately the same runtime (~40 s) and is still much faster than baselines.

Next, we demonstrate a visualization of the optimized embeddings in latent space with t-SNE [23] in Fig. 7. With the diffusion model, the optimized variables are aligned with the embeddings of feasible transformations, allowing them to be mapped back to transformations instantly. The corresponding sequence in Fig. 7 is [`rw; rw; rsz; rfz; rw; rf; rf; rfz; b; rs; rw; rsz; rwz; rw; rw; rsz; rsz; rsz; rw; rsz`], which yields an area of 11900.00 μm^2. In contrast, without the diffusion model, optimized variables diverge from embeddings of feasible transformations. Mapping these latent variables back to the nearest feasible transformations with large discrepancies gives a sequence as [`rfz; rfz; rwz; rs; rs; rsz; rs; rs; rs; rs; rs; rwz; rs; rwz; rs; rs; rwz; rw; rwz; rwz`] and leads to an area of 22654.15 μm^2, which is 1.9 times larger than the area with a diffusion model.

V. Conclusion

In this paper, we propose to efficiently optimize logic synthesis sequences in a continuous space. The gradient-based optimization in continuous space alleviates the heavy runtime burden of extensive discrete search of transformations. We additionally incorporate a diffusion model to keep latent variables in feasible embedding distributions, allowing instant retrieval of feasible transformations without degrading QoR. Our experimental results show that our work is not only significantly faster than previous methods but also reaches better QoR.

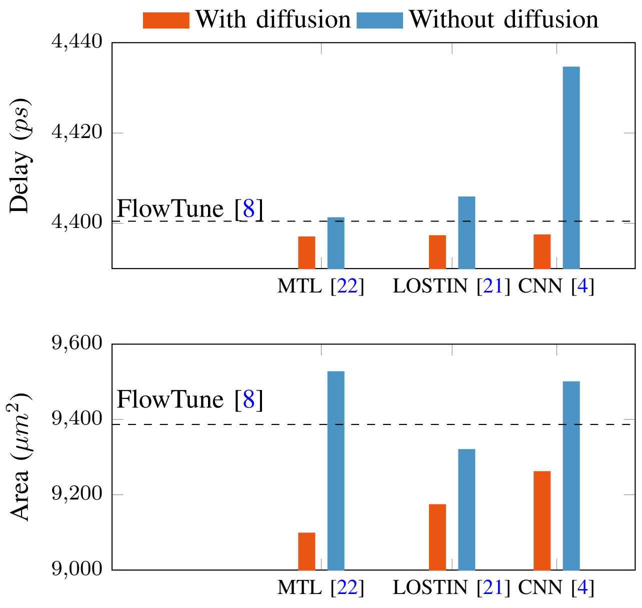


Fig. 6 Area and delay with and without diffusion model for various surrogate models.

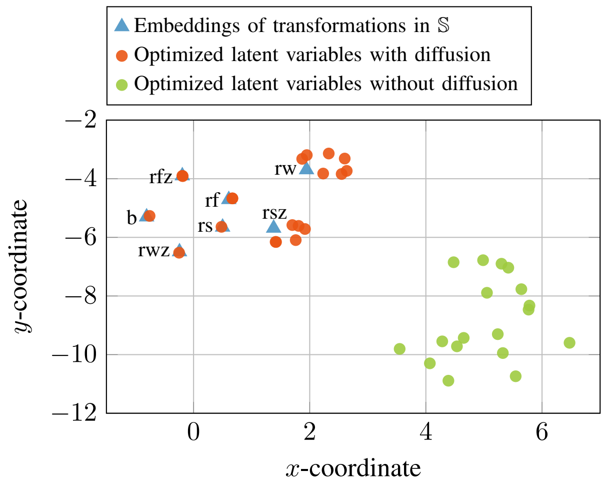


Fig. 7 The t-SNE projection of transformation embeddings from `div` design. The optimized latent variables without diffusion model deviate from feasible transformation embeddings.

Acknowledgment

This work is supported in part by the Department of Education of Guangdong Province (No. 2024KTSCX037) and the Guangzhou Municipal Science and Technology Project (Municipal Key Laboratory Construction Project, Grant No.2023A03J0013).

References

[1] R. Brayton and A. Mishchenko, "Abc: An academic industrial-strength verification tool," in *Proc. CAV*. Springer, 2010, pp. 24–40.

[2] L. Amaru, P.-E. Gaillardon, and G. De Micheli, "Majority-inverter graph: A new paradigm for logic optimization," *IEEE TCAD*, vol. 35, no. 5, pp. 806–819, 2015.

[3] W. L. Neto, M. Austin, S. Temple, L. Amaru, X. Tang, and P.-E. Gaillardon, "Lsoracle: a logic synthesis framework driven by artificial intelligence: Invited paper," in *Proc. ICCAD*, 2019, pp. 1–6.

[4] C. Yu, H. Xiao, and G. De Micheli, "Developing synthesis flows without human knowledge," in *Proc. DAC*, 2018, pp. 1–6.

[5] A. Hosny, S. Hashemi, M. Shalan, and S. Reda, "Drills: Deep reinforcement learning for logic synthesis," in *Proc. ASPDAC*, 2020, pp. 581–586.

[6] K. Zhu, M. Liu, H. Chen, Z. Zhao, and D. Z. Pan, "Exploring logic optimizations with reinforcement learning and graph convolutional network," in *Proc. MLCAD*, 2020, p. 145–150.

[7] C. Lv, Z. Wei, W. Qian, J. Ye, C. Feng, and Z. He, "Gpt-ls: Generative pre-trained transformer with offline reinforcement learning for logic synthesis," in *Proc. ICCD*, 2023, pp. 320–326.

[8] W. L. Neto, Y. Li, P.-E. Gaillardon, and C. Yu, "Flowtune: End-to-end automatic logic optimization exploration via domain-specific multiarmed bandit," *IEEE TCAD*, vol. 42, no. 6, pp. 1912–1925, 2023.

[9] A. Grosnit, C. Malherbe, R. Tutunov, X. Wan, J. Wang, and H. B. Ammar, "Boils: Bayesian optimisation for logic synthesis," in *Proc. DATE*, 2022, pp. 1193–1196.

[10] 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, pp. 268–276, 2018.

[11] T. Wu, T. Maruyama, L. Wei, T. Zhang, Y. Du, G. Iaccarino, and J. Leskovec, "Compositional generative inverse design," *arXiv preprint*, 2024.

[12] Q. Zhao, D. B. Lindell, and G. Wetzstein, "Learning to solve pde-constrained inverse problems with graph networks," *arXiv preprint*, 2022.

[13] J. Jiang and J. A. Fan, "Global optimization of dielectric metasurfaces using a physics-driven neural network," *Nano letters*, vol. 19, no. 8, pp. 5366–5372, 2019.

[14] J. Lu, L. Lei, F. Yang, L. Shang, and X. Zeng, "Topology optimization of operational amplifier in continuous space via graph embedding," in *Proc. DATE*, 2022, pp. 142–147.

[15] T. White, "Sampling generative networks," *arXiv preprint*, 2016.

[16] Z. Dong, W. Cao, M. Zhang, D. Tao, Y. Chen, and X. Zhang, "Cktgnn: Circuit graph neural network for electronic design automation," *arXiv preprint*, 2023.

[17] J. Chu, J. Park, S. Lee, and H. J. Kim, "Inversion-based latent bayesian optimization," in *Proc. NIPS*, 2024.

[18] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *Proc. NIPS*, vol. 33, pp. 6840–6851, 2020.

[19] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," 2021.

[20] J. Ho, T. Salimans, A. Gritsenko, W. Chan, M. Norouzi, and D. J. Fleet, "Video diffusion models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 8633–8646, 2022.

[21] N. Wu, J. Lee, Y. Xie, and C. Hao, "Lostin: Logic optimization via spatio-temporal information with hybrid graph models," in *Proc. ASAP*, 2022, pp. 11–18.

[22] Y. Ouyang, S. Li, D. Zuo, H. Fan, and Y. Ma, "Asap: Accurate synthesis analysis and prediction with multi-task learning," in *Proc. MLCAD*, 2023, pp. 1–6.

[23] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne." *Journal of Machine Learning Research*, vol. 9, no. 11, 2008.

[24] D. P. Kingma, "Auto-encoding variational bayes," *arXiv preprint*, 2013.

[25] A. Bansal, H.-M. Chu, A. Schwarzschild, S. Sengupta, M. Goldblum, J. Geiping, and T. Goldstein, "Universal guidance for diffusion models," in *Proc. CVPR*, 2023, pp. 843–852.

[26] L. Wei, P. Hu, R. Feng, H. Feng, Y. Du, T. Zhang, R. Wang, Y. Wang, Z.-M. Ma, and T. Wu, "A generative approach to control complex physical systems," *arXiv preprint*, 2024.

[27] L. Amarú, P.-E. Gaillardon, and G. De Micheli, "The epfl combinational benchmark suite," in *Proc. IWLS*, 2015.

[28] M. C. Hansen, H. Yalcin, and J. P. Hayes, "Unveiling the iscas-85 benchmarks: A case study in reverse engineering," *IEEE Design & Test of Computers*, vol. 16, no. 3, pp. 72–80, 1999.

[29] X. Xu, N. Shah, A. Evans, S. Sinha, B. Cline, and G. Yeric, "Standard cell library design and optimization methodology for asap7 pdk," in *Proc. ICCAD*, 2017, pp. 999–1004.

[30] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Proc. MICCAI*, 2015, pp. 234–241.