AC-Refiner: Efficient Arithmetic Circuit Optimization Using Conditional Diffusion Models

Chenhao Xue[1], Kezhi Li[2], Jiaxing Zhang[1], Yi Ren[1,3], Zhengyuan Shi[1],
Chen Zhang[4], Yibo Lin[1,5,6], Lining Zhang[7], Qiang Xu[2,8], Guangyu Sun[1,5,6,*]

[1]*School of Integrated Circuits, Peking University*, Beijing, China
[2]*Department of Computer Science and Engineering, The Chinese University of Hong Kong*, Sha Tin, Hong Kong S.A.R.
[3]*School of Software and Microelectronics, Peking University*, Beijing, China
[4]*Shanghai Jiao Tong University*, Shanghai, China
[5]*Institute of Electronic Design Automation, Peking University*, Wuxi, China
[6]*Beijing Advanced Innovation Center for Integrated Circuits*, Beijing, China
[7]*School of Electronic and Computer Engineering, Peking University*, Shenzhen, China
[8]*National Center of Technology Innovation for EDA*, Nanjing, China
{xch927027, yibolin, eelnzhang, gsun}@pku.edu.cn, {yiren20,zjx}@stu.pku.edu.cn
{kzli24, zyshi21, qxu}@cse.cuhk.edu.hk, chenzhang.sjtu@sjtu.edu.cn

Abstract—Arithmetic circuits, such as adders and multipliers, are fundamental components of digital systems, directly impacting the performance, power efficiency, and area footprint. However, optimizing these circuits remains challenging due to the vast design space and complex physical constraints. While recent deep learning-based approaches have shown promise, they struggle to *consistently explore high-potential design variants*, limiting their optimization efficiency. To address this challenge, we propose *AC-Refiner*, a novel arithmetic circuit optimization framework leveraging conditional diffusion models. Our key insight is to reframe arithmetic circuit synthesis as a conditional image generation task. By carefully conditioning the denoising diffusion process on target quality-of-results (QoRs), AC-Refiner consistently produces high-quality circuit designs. Furthermore, the explored designs are used to fine-tune the diffusion model, which focuses the exploration near the Pareto frontier. Experimental results demonstrate that AC-Refiner generates designs with superior Pareto optimality, outperforming state-of-the-art baselines. The performance gain is further validated by integrating AC-Refiner into practical applications.

Index Terms—Diffusion models, Arithmetic circuits, Design automation

Fig. 1. (a) Image generation with a conditional diffusion model. (b) AC-Refiner represents arithmetic circuits similar to images, and conditionally samples circuit designs with desired properties, enabling efficient optimization.

I. INTRODUCTION

In digital circuit designs, arithmetic circuits—such as adders and multipliers—serve as fundamental building blocks. These components play a critical role in determining the performance, power consumption, and area efficiency of hardware systems, particularly in computation-intensive applications like digital signal processing and neural networks. As a result, optimizing arithmetic circuits is essential for achieving high-performance and energy-efficient systems.

The optimization of arithmetic circuits, however, presents significant challenges due to the vast design space, which grows exponentially with input bit width. Over the years, a range of manual design techniques [1]–[5] and algorithmic methods [6]–[13], which focus on minimizing theoretical properties such as logic depth and size. Despite these efforts, a substantial gap remains between theoretical optimization and real-world performance. This discrepancy stems from the inherent difficulty of accurately modeling physical implementation constraints (e.g. capacitive loading and routing congestion) at early design stages [14], [15]. Consequently, purely theoretical optimizations cannot guarantee optimal quality-of-results (QoRs) in practice.

Recent advancements have introduced learning-based approaches to address the limitations of theoretical optimization, facilitating more effective design space 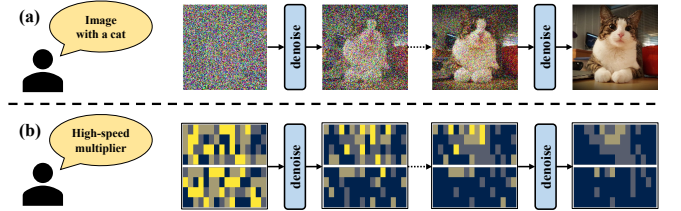exploration (DSE) of adders [15]–[18] and multipliers [16], [19], [20] for physically optimized implementations. Nevertheless, existing learning-based methods still face fundamental limitations in optimization efficiency, particularly when scaling to large and complex arithmetic units such as high-performance multipliers.

On the one hand, reinforcement learning (RL) has been applied to optimize designs through iterative interactions with physical synthesis tools [15], [16], [19], [20]. While RL-based methods demonstrate potential, they often require extensive trial-and-error before converging on optimized solutions. This iterative process can be prohibitively time-consuming and computationally expensive, particularly for large-scale multipliers [16]. Consequently, RL-agents often struggle to effectively navigate the vast design space within limited budgets, yielding suboptimal design quality.

On the other hand, generative models offer a promising alternative to address the efficiency limitation of RL. By capturing the structural patterns of prior arithmetic circuit designs, they can rapidly sample diverse promising candidates for parallel evaluation. Prior research has explored variational autoencoders (VAEs) for prefix adder optimization [17], [18]. However, VAEs exhibit limited capability to model fine-grained structural details [21]. For arithmetic modules with complex design constraints (e.g. multiplier compressor trees), it is difficult for VAEs to sample valid designs.

In this paper, we aim to address the limitations of existing learning-based arithmetic circuit optimization by leveraging recent advancements of generative models. To this end, we propose **AC-Refiner**, which employs conditional diffusion models [22]–[24] for the comprehensive and efficient optimization of multipliers, as shown in Fig. 1. The key contributions of AC-Refiner are summarized as follows:
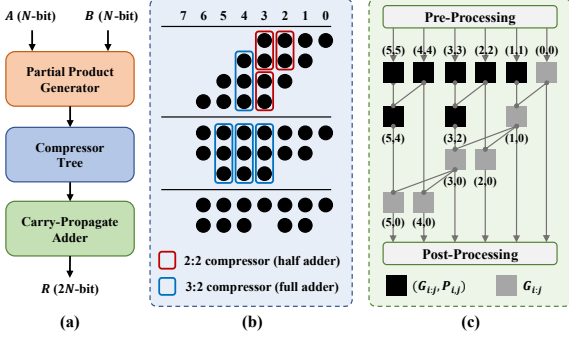
Fig. 2. (a) Multiplier architecture. AC-Refiner targets critical submodules of multipliers, including (b) compressor tree and (c) prefix adder.
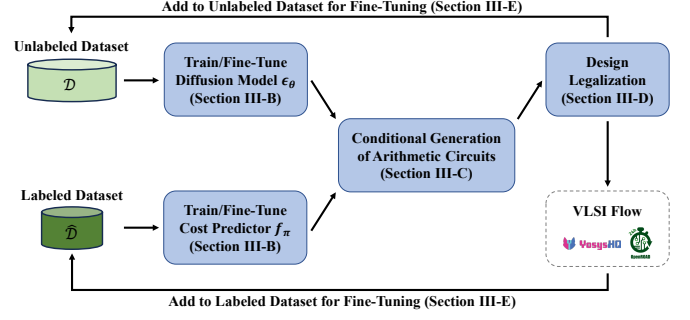

Fig. 3. AC-Refiner framework overview. After training the models, we use gradient-guided sampling to explore high-potential arithmetic circuit designs, which undergo legalization and VLSI synthesis. The explored high-quality designs are leveraged to fine-tune the model for the next optimization round.

- **Circuit Generation:** Unlike VAEs, which generate arithmetic circuit designs in a single inference step, diffusion models employ an *iterative inference process*. Starting from a random initial design, they progressively refine the structure, ensuring close adherence to design constraints. Incorporated with proper legalization procedures, diffusion models allow AC-Refiner to deliver valid and diverse candidate designs.
- **Guided Optimization:** To achieve efficient optimization, AC-Refiner introduces *conditional sampling mechanism* to explore high-potential designs. Specifically, AC-Refiner adopts neural cost predictors whose gradient steer the inference process toward promising outcomes. Additionally, AC-Refiner leverages discovered high-quality designs to fine-tune the models, which facilitates focused exploration near the Pareto frontier.

We apply AC-Refiner to the comprehensive optimization of multipliers, covering both compressor trees and carry-propagate adders. Our experimental results show that AC-Refiner discovers multipliers that Pareto-dominate all baseline methods at various bit-widths, achieving reductions of up to 15% in delay and 10% in area, respectively. The effectiveness of AC-Refiner is further validated through implementing multipliers within systolic arrays, highlighting its practical applicability in real-world designs. We extensively ablate the proposed techniques, including gradient-guided sampling and model fine-tuning, and demonstrate their effectiveness for improving optimization efficiency.

The remainder of this paper is organized as follows: Section II provides background on multiplier design and diffusion models. Section III details the AC-Refiner framework. Section IV presents the evaluation results. Finally, Section V concludes the paper.

II. BACKGROUND

A. Multiplier Architecture

As shown in Fig. 2(a), a multiplier consists of three main components: a partial product generator (PPG), a compressor tree (CT), and a carry-propagate adder (CPA).

- *Partial Product Generator:* The PPG is responsible for generating partial products (PPs), which are shifted into different columns to represent distinct power-of-two weights. A commonly employed AND gate-based PPG generates N^2 PPs for an N-bit multiplier.
- *Compressor Tree:* As illustrated in Fig. 2(b), the CT compresses PPs for multiple stages until no more than two PPs remain in each column. It is typically composed of various types and quantities of compressors, with full adders (FAs) and half adders (HAs) being predominantly used. These basic compressors take multiple bits (3 for FA and 2 for HA) of the same weight as input, producing one bit of the original weight and another of higher weight.

- *Carry-Propagate Adder:* The CPA aggregates the compressed PPs from the CT to produce the final multiplication result. To optimize for performance and accommodate the non-uniform signal arrival time [13], AC-Refiner adopts prefix adders to implement CPA. Prefix adders exhibit a tree-based structure due to the inherent associativity of prefix computation, as shown in Fig. 2(c).

B. Diffusion Model

Diffusion models are strong generative models, demonstrating superior performance over VAEs when introduced to controllable generation of high-quality images [23]. Due to their exceptional capabilities, diffusion models have been extended to various domains [25]–[29]. Conceptually, diffusion models consist of a T-step forward process and a T-step reverse process. In the forward process, the diffusion model gradually destroys clean training data x_0 with Gaussian noise for multiple timesteps:

$$x_t = \sqrt{\alpha_t} \cdot x_0 + \sqrt{1 - \alpha_t} \cdot \epsilon, \quad \epsilon \sim \mathcal{N}(0, \mathbf{I}) \quad (1)$$

where x_t represents the noisy data at timestep $t \in [1, T]$, and $\{\alpha_t\}_{t=1}^{T}$ denotes the noise schedule. In the reverse process, the diffusion model learns to recover x_0 by training a neural network ϵ_θ to predict the injected noise ϵ:

$$\epsilon_\theta(x_t, t) \approx \epsilon = \frac{x_t - \sqrt{\alpha_t} \cdot x_0}{\sqrt{1 - \alpha_t}} \quad (2)$$

At inference time, the diffusion model starts from a randomly sampled noise $x_T \sim \mathcal{N}(0, \mathbf{I})$ and gradually denoises the data using ϵ_θ. Since the vanilla sampling typically requires $T = 1000$ steps [23], many accelerated methods are proposed [24], [30]. These methods share a similar formulation, which first predicts the injected noise $\hat{\epsilon} = \epsilon_\theta(x_t, t)$, and produces a less noisy data x_{t-1} w.r.t. x_t, $\hat{\epsilon}$, and t:

$$x_{t-1} = \mathcal{S}(x_t, \hat{\epsilon}, t) \quad (3)$$

III. METHODOLOGY

A. Framework Overview

In this section, we describe the proposed AC-Refiner framework for optimizing arithmetic circuits. As illustrated in Fig. 3, we commence by training diffusion models and cost predictors, where the compressor trees and prefix trees are encoded in image-like binary bitmaps (Section III-B). Next, we employ guided sampling to generate promising design candidates (Section III-C), and adopt the legalization procedure to ensure design correctness (Section III-D). Finally, we leverage the explored high-quality designs to fine-tune the models before proceeding to the next optimization round (Section III-E).
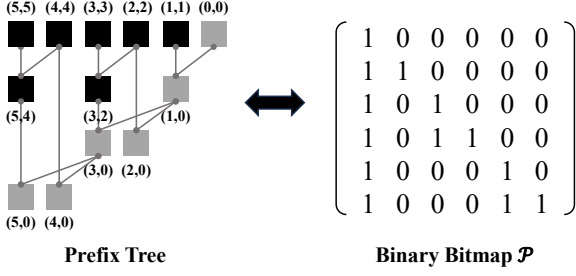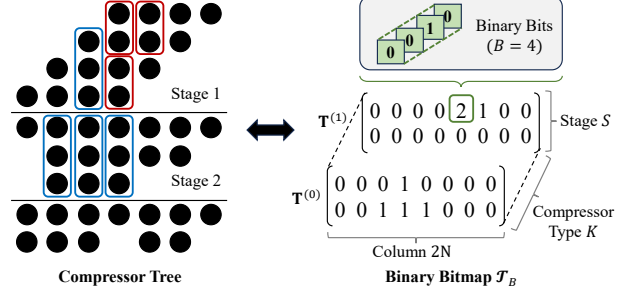
Fig. 4. Data representation of prefix adder.

Fig. 5. Data representation of compressor tree.

B. Model Training

In this section, we first explain how to represent arithmetic circuits with image-like multi-dimensional tensors. Then, we introduce how to train the diffusion models and neural cost predictors.

Prefix Adder Representation. As shown in Fig.4, we represent N-bit prefix adders as bitmap $\mathcal{P} \in \{0,1\}^{N \times N}$, where $\mathcal{P}[i,j] = 1$ indicates the presence of prefix signals $G_{i,j}$ $(i \geq j)$ and $P_{i,j}$ $(i > j)$. The upper-right triangular region of $\mathcal{P}$ are padded with zeros.

The prefix structure imposes the following design constraint: each prefix node must have exactly one pair of upper/lower parent:

$$\mathcal{P}[i,j] = 1 \implies \exists k \in [j+1,i] \cap \mathbb{Z}, \quad \mathcal{P}[i,k] = \mathcal{P}[k-1,j] = 1 \quad (4)$$

When multiple k satisfy Equation 4, the minimal k determines the canonical parent pair. For example, node $(5,0)$ in Fig. 4 have two valid parent pairs $(5,5)/(4,0)$ and $(5,4)/(3,0)$, with the latter selected as the parent nodes. Consequently, each bitmap $\mathcal{P}$ uniquely defines a valid parallel prefix tree.

Compressor Tree Representation. As illustrated in Fig.5, we represent a compressor tree using a three-dimensional tensor $\mathcal{T} \in \mathbb{R}^{K \times 2N \times S}$, where K denotes the number of compressor types, N denotes the multiplier bit-width, and S denotes the number of stages. Without loss of generality, we consider $K = 2$ and only employ full adders and half adders to construct CT in AC-Refiner. The maximum compression stage S is computed as $S_{min} + \Delta S$. S_{min} is the stage count of an N-bit Wallace multiplier, which is also the theoretical lower bound [4]. $\Delta S = 1$ is a hyperparameter that balances design quality against the number of feasible solutions.

To formalize the CT design rules, we introduce tensor $\mathcal{V} \in \mathbb{R}^{2N \times (S+1)}$ that tracks the number of partial products at each compression stage, with $\mathcal{V}[\cdot, 0]$ determined by PPG and $\mathcal{V}[\cdot, S]$ as the final partial product counts. Using the established notations, a valid compressor tree design $\mathcal{T}$ should satisfy the following constraints:

$$\mathcal{T}[k,c,s] \geq 0, \quad \forall k \in \{0,1\}, c \in [0,2N), s \in [0,S) \quad (5)$$

$$3 \cdot \mathcal{T}[0,c,s] + 2 \cdot \mathcal{T}[1,c,s] \leq \mathcal{V}[c,s], \quad \forall c \in [0,2N), s \in [0,S) \quad (6)$$

$$\begin{aligned} \mathcal{V}[c,s+1] = \mathcal{V}[c,s] - (2 \cdot \mathcal{T}[0,c,s] + \mathcal{T}[1,c,s]) \\ + (\mathcal{T}[0,c-1,s] + \mathcal{T}[1,c-1,s]), \forall c \in [1,2N), s \in [0,S) \end{aligned} \quad (7)$$

$$\mathcal{V}[0,s+1] = \mathcal{V}[0,s] - (2 \cdot \mathcal{T}[0,0,s] + \mathcal{T}[1,0,s]), \quad \forall s \in [0,S) \quad (8)$$

$$0 \leq \mathcal{V}[c,S] \leq 2, \quad \forall c \in [0,2N) \quad (9)$$

Given a valid CT configuration $\mathcal{T}$, the mapping from $\mathcal{T}$ to RTL implementation is not unique, as input assignments of co-located compressors can be permuted without affecting the multiplication correctness. To obtain a unique and high-quality CT design, we adopt a deterministic interconnection scheme that prioritizes assigning input bits with minimal signal arrival time to FAs over HAs. The internal delay of compressors is estimated on the selected technology node.

Training Diffusion Model. The training process of diffusion models proceeds as follows:

1) We convert the circuit representations into binary bitmaps. For CT designs, we follow Analog-Bit [31] to split integers into multiple binary bits with different weights. The resulting bitmap is denoted as $\mathcal{T}_B$, as shown in Fig. 5.
2) We cast the input bitmap to a real-valued tensor, denoted as x_0. Each binary bit $b \in \{0,1\}$ is mapped to a corresponding real value $r \in \{-1.0, 1.0\}$. Conversely, a real-valued r can be quantized back to a discrete b according to each element's sign.
3) We randomly sample Gaussian noise $\epsilon \sim \mathcal{N}(0, \mathbf{I})$ and obtain noisy data x_t according to the noise schedule in Equation (1).
4) We train the diffusion model ϵ_θ to predict the injected noise as in Equation (2). ϵ_θ is implemented with U-Net backbones [32]. The training process can be formulated as minimizing the mean-square error (MSE) loss:

$$\min_\theta \mathcal{L}_\theta(x_t, \epsilon, t) = (\epsilon_\theta(x_t, t) - \epsilon)^2. \quad (10)$$

Training Cost Predictor. Apart from training diffusion models, AC-Refiner also utilizes neural cost predictors to guide the circuit generation. Specifically, we define function f as the mapping from an arithmetic circuit design to the post-synthesis Quality-of-Results (QoR). To enable gradient-based guidance, f takes in the same continuous tensor x as the diffusion model ϵ_θ. To trade off multiple competing QoR metrics and obtain Pareto-optimal circuit design, we conduct multiple synthesis flows to evaluate the arithmetic circuit design x. The considered design scenarios include timing-driven, area-driven, and balanced optimization. The total cost $f(x)$ is defined as a weighted sum of delay and area across all the scenarios:

$$f(x) = (w \sum_{i=1}^{n} \text{delay}_i(x) + (1-w) \sum_{i=1}^{n} \text{area}_i(x))/n. \quad (11)$$

where delay_i and area_i are the post-synthesis metrics in i-th design scenario, and w is the weight to trade off the competing goals.

Since f lacks a closed-form expression, we aim to train a differentiable regression model f_π to approximate the ground truth f. We implement f_π as a 3-layer CNN composed of convolutional residual blocks [33]. Given design x labeled with its corresponding QoR y, the training process is formulated as minimizing the MSE loss:

$$\min_\pi \mathcal{L}_\pi(x, y) = (f_\pi(x) - y)^2 \quad (12)$$

C. Conditional Design Generation

To consistently explore high-potential designs, AC-Refiner introduces a conditional sampling process. Conceptually, we incorporate the gradient guidance from the neural cost predictor f_π to steer the denoising direction towards high QoR outcomes. As illustrated in
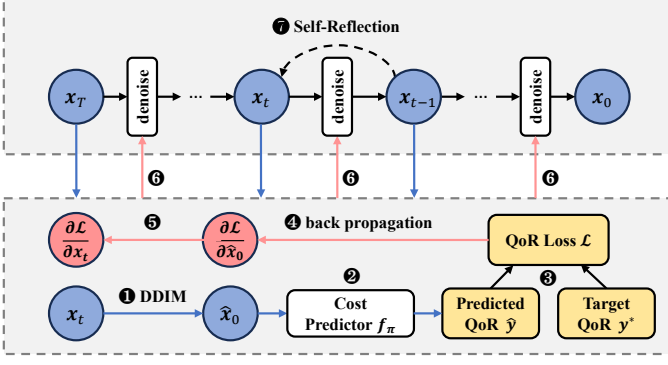
Fig. 6. Conditional generation of arithmetic circuits. Gradient-guided sampling (Step 1-6) drives the generated designs towards achieving the target quality-of-results (QoR). Self-reflection (Step 7) prevents the gradient guidance from compromising the structural correctness.

Fig. 6, the conditional design generation process comprises two key ingredients: gradient-guided sampling and self-reflection.

Gradient-Guided Sampling. A critical challenge in guiding the diffusion process is evaluating the quality of intermediate states. On the one hand, the noisy x_t lacks a well-defined structure. Since the cost predictor f_π is trained on unperturbed tensors, directly assessing x_t cannot reliably provide meaningful guidance. On the other hand, the desired x_0 is available upon completion of the diffusion process.

To address this dilemma, we draw inspiration from DDIM [24] to approximate x_0 at intermediate steps (Step ❶):

$$\hat{x}_0 = \frac{x_t - \sqrt{1 - \alpha_t} \cdot \epsilon_\theta(x_t, t)}{\sqrt{\alpha_t}}, \tag{13}$$

DDIM originally employs the predicted clean data $\hat{x}_0$ to accelerate sampling, whereas AC-Refiner repurposes this mechanism to acquire informative feedback from cost predictor f_π. Although $\hat{x}_0$ remains an imperfect approximation, it exhibits a more regularized structure over the noisy x_t, which enables cost predictor f_π to deliver a more accurate prediction on post-synthesis QoR (Step ❷).

For efficient design optimization, we expect to prune suboptimal candidate designs whose predicted QoR $\tilde{y} = f_\pi(\hat{x}_0)$ does not reach a desirable target y^*. This objective is formulated as minimizing loss $\mathcal{L}(y^*, \tilde{y})$, where we select mean-square error as the loss function (Step ❸). To achieve effective minimization, we backpropagate the gradient of loss $\mathcal{L}$ down to x_t (Steps ❹-❺), and add a gradient rectification term $\frac{\partial \mathcal{L}}{\partial x_t}$ in the denoising step from x_t to x_{t-1} (Step ❻):

$$x_{t-1} = \mathcal{S}(x_t, \hat{\epsilon}, t) - s(t) \cdot \nabla_{x_t} \mathcal{L}(y^*, f_\pi(\hat{x}_0)), \tag{14}$$

This adjustment attempts to progressively reduce $\mathcal{L}$ throughout the denoising process. In Equation (14), $s(t) > 0$ controls the guidance strength, and $\mathcal{S}$ adopts the sampling procedure from DDIM [24].

Self-Reflection. In practice, the gradient-guided diffusion process struggles to balance design correctness and quality. A radical guidance strength $s(t)$ may sabotage the normal denoising process and lead to invalid designs. Whereas a conservative one may not effectively minimize loss $\mathcal{L}(y, y^*)$ in limited denoising steps. To address this issue, we apply the self-reflection strategy following [34], which ensures QoR optimization without compromising correctness. Specifically, after gradient-guided denoising produces x_{t-1}, AC-Refiner reintroduces random Gaussian noise ϵ' back to x_{t-1} (Step ❼):

$$x_t = \sqrt{\alpha_t/\alpha_{t-1}} \cdot x_{t-1} + \sqrt{1 - \alpha_t/\alpha_{t-1}} \cdot \epsilon', \tag{15}$$

Algorithm 1 Gradient-Guided Sampling

Input: noisy data $x_T \sim \mathcal{N}(0, I)$, total timestep T, noise scale $\{\alpha_t\}_{t=1}^T$, diffusion model ϵ_θ, cost predictor f_π, criterion $\mathcal{L}$, target QoR y^*, guidance strength $s(t)$, and self-reflection step k.
Output: Sampled design x_0
1: **for** $t = T, T-1, \ldots, 1$ **do**
2:     **for** $n = 1, 2, \ldots, k$ **do**
3:         Predict noise $\hat{\epsilon} = \epsilon_\theta(x_t, t)$
4:         Predict final design $\hat{x}_0$ as Equation (13)
5:         Denoise design x_t to x_{t-1} as Equation (14)
6:         Perturb x_{t-1} to x_t as Equation (15)
7: **return** x_0

Algorithm 2 Compressor Tree Legalization

Input: Compressor tree $\mathcal{T}$ to be legalized
Output: Legalized compressor tree structure
1: **while** $\exists$ error $e \in E(\mathcal{T})$ **do**
2:     Initialize candidate action set $\mathcal{A} \leftarrow \emptyset$
3:     **if** e is over-compression at column c and stage s **then**
4:         **if** $c > 1$ **then**
5:             $\mathcal{A} \leftarrow \mathcal{A} \cup \{\texttt{SplitFA}_{(c-1),s'} | 1 \le s' < s\}$
6:         $\mathcal{A} \leftarrow \mathcal{A} \cup \{\texttt{ReplaceFA}_{c,s'} | 1 \le s' \le s\}$
7:         $\mathcal{A} \leftarrow \mathcal{A} \cup \{\texttt{DeleteHA}_{c,s}\}$
8:     **else if** e is under-compression at column c **then**
9:         **if** $c > 1$ **then**
10:            $\mathcal{A} \leftarrow \mathcal{A} \cup \{\texttt{FuseFA}_{(c-1),s'} | 1 \le s' < S\}$
11:         $\mathcal{A} \leftarrow \mathcal{A} \cup \{\texttt{ReplaceHA}_{c,s'} | 1 \le s' \le S\}$
12:         $\mathcal{A} \leftarrow \mathcal{A} \cup \{\texttt{AddHA}_{c,s'} | 1 \le s' \le S\}$
13:     Choose action $A^* = \arg\min_{A \in \mathcal{A}} |E(A(\mathcal{T}))|$
14:     Update design $\mathcal{T} \leftarrow A^*(\mathcal{T})$
15: **return** $\mathcal{T}$

The resulting intermediate state x_t has an in-distribution pattern at timestep t and better alignment with the gradient guidance. In plain words, self-reflection mechanism enables selective sampling of x_t instances that largely satisfy the gradient guidance. This iterative process is repeated multiple times before advancing to timestep $t - 2$.

Algorithm 1 summarizes the overall conditional design generation process, which integrates the proposed gradient-guided sampling and self-reflection mechanism. While it largely respects the design constraints, it cannot perfectly guarantee structural integrity, which necessitates the legalization process introduced in Section III-D.

D. Design Legalization

To ensure the functional correctness of the generated arithmetic circuit designs, AC-Refiner introduces heuristic legalization procedures to eliminate design rule violations (DRVs) in the circuits' tensor representation. For prefix adders, we adopt the established approach from PrefixRL [15], which iterates through all the prefix nodes and fills the missing lower parents to ensure a valid bitmap $\mathcal{P}$. For compressor trees, GOMIL [12] utilizes integer linear programming (ILP) to produce valid designs satisfying Equation (5)-(9). However, the unpredictable behavior of ILP solver may negate the benefits from conditional circuit generation, rendering it unsuitable in our use case.

Empirically, we observe that CT designs generated by the diffusion models typically exhibit few DRVs, which provide opportunities to correct these errors via efficient heuristics. As shown in Algorithm 2, AC-Refiner adopts CT legalization procedure that takes a series of local adjustments to resolve design rule violations. We denote $E(\mathcal{T})$

as the set of error positions in the compressor tree design $\mathcal{T}$, consisting of over-compression (violating Equation (6)) and under-compression (violating Equation (9)). The algorithm handles over-compression by increasing partial products or reducing compressors (Lines 3-7), and addresses under-compression by adding compressors if needed (Lines 8-12). For quick convergence to a valid configuration, the algorithm greedily selects an action A^* from all candidate actions $\mathcal{A}$, which minimizes the size of $E(\mathcal{T})$ (Line 13 to 14). When no action can effectively shrink $E(\mathcal{T})$, we force the legalization process to take a detour, selecting an action that incurs the minimum new errors. In most cases, we observe that Algorithm 2 manages to quickly converge on valid solutions.

E. Model Fine-Tuning

To further improve optimization efficiency, AC-Refiner fine-tunes the models leveraging the explored high-quality designs. This process enables the models to better capture the intricate pattern of optimized designs, which increases the likelihood of discovering even superior ones. As shown in Fig. 3, AC-Refiner performs M rounds of optimization. In each round, the generated designs constitute a new unlabeled dataset $\mathcal{D}'$. A subset of $\mathcal{D}'$ is evaluated for QoR with physical synthesis tools, creating a labeled dataset $\hat{\mathcal{D}}'$. The newly acquired datasets are then merged with the initial dataset $\mathcal{D}$ and $\hat{\mathcal{D}}$ to fine-tune the diffusion model ϵ_θ and regression model f_π.

IV. EVALUATIONS

A. Setup

Platform. All experiments are conducted on a Linux-based platform equipped with a 2.9GHz 64-core Intel Xeon Gold 6226R CPU, 376GB of memory, and an NVIDIA A6000 GPU. We use open-sourced logic synthesis tool Yosys (version 0.27) [35] and the physical synthesis tool OpenROAD (version 2.0) [36] to assess the Quality-of-Results (QoR) of multiplier designs using the NanGate 45nm technology process library [37].

Dataset Preparation. Training robust generative models requires an extensive dataset of validated design configurations. To meet this demand, we utilize heuristic algorithms to modify manual designs [1]–[5] for improved theoretical properties. The modification operation and associated legalization process are adopted from RL-MUL [19] and PrefixRL [9] for CT and CPA, respectively. For CT designs, we add an extra optimization by randomly swapping compressors at different stages, yielding new CTs beyond the design space of RL-MUL.

Design Verification. For RTL generation of AC-Refiner, we implement a Python program to convert the compressor tree configurations $\mathcal{T}$ and prefix adder configurations $\mathcal{P}$ into Verilog HDL codes. These components are then composed to construct multipliers. The pre-synthesis Verilog codes of all multipliers, including those from baseline methods and from AC-Refiner, are converted to and-inverter graph with ABC [38] and formally verified with RevSCA-2.0 [39].

Baselines. We compare AC-Refiner with four baseline methods for multiplier optimization, including Wallace multiplier [4], ILP-based method GOMIL [12], and RL-based methods RL-MUL [19] and ArithmeticTree [16].

Hyperparameter Settings. We implement the diffusion models ϵ_θ and cost predictors f_π with PyTorch. To evaluate the QoR cost function as in Equation (11), we synthesize multiplier designs under area-driven scenario and timing-driven scenario, and normalize the delay and area with corresponding values of Wallace multiplier [4]. The trade-off weight is set to $w = 0.66$. For gradient-guided sampling, we set the target QoR objective $y = 0.7$, the guidance strength $s(t) = 10\sqrt{1 - \alpha_t}$, and the self-reflection step to $k = 25$. We
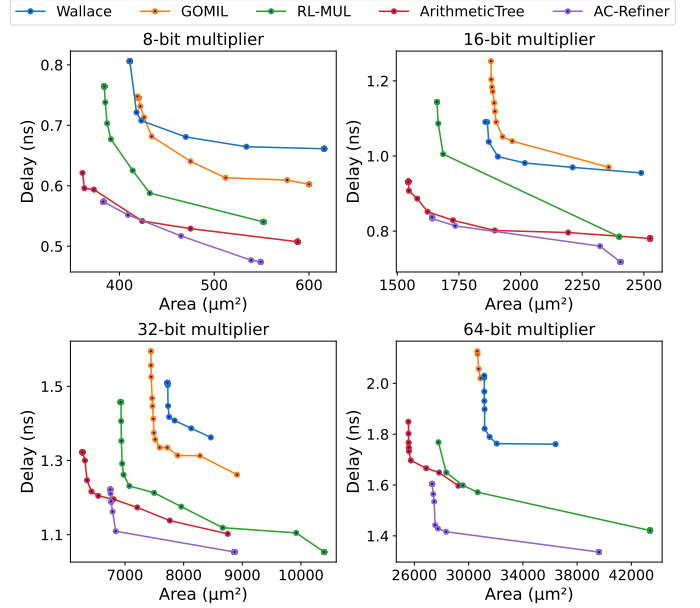

Fig. 7. Pareto frontiers of the synthesized results on multipliers.

conduct a comprehensive optimization of multipliers covering both CT and CPA. We commence by optimizing CT and fix CPA with the default adder from the synthesis tool. Then we adopt the best explored CT and refine the CPA. For CT and CPA, respectively, the initial dataset consists of $|D_0| \approx 15000$ unlabeled samples, and $|\hat{D}_0| = 1000$ designs labeled with post-synthesis QoR. Optimization runs for $M = 5$ rounds, where in each round we sample $|D_m| = 1000$ designs and randomly select $|\hat{D}_m| = 100$ designs for physical synthesis. In total, the synthesis tools are invoked 3000 times, which is the same as learning-based baselines RL-MUL and ArithmeticTree.

B. Multiplier Optimization Result

We compare the Pareto frontiers obtained using different optimization algorithms. Following established evaluation methodologies [16], [19], we select the optimal multiplier from each approach and synthesize them with a target delay ranging from 0ns to 2ns to demonstrate the trade-off between timing and area. As shown in Fig. 7, AC-Refiner achieves superior Pareto optimality over all baselines. Both the manually designed Wallace multiplier and the algorithmic method GOMIL fail to account for the complexities inherent in physical implementation, leading to suboptimal post-synthesis delay and area. RL-MUL modifies only the column-wise compressor count without considering detailed stage-specific compressor assignment or CPA optimization, yielding inferior performance. ArithmeticTree trains the RL agents to construct multipliers in a gate-by-gate manner. Despite fine-grained optimization, it suffers from exploration inefficiency as the RL agents must rely on extensive trial-and-error iterations. Compared with the state-of-the-art ArithmeticTree method, AC-Refiner achieves a delay reduction of up to 15% and an area reduction of up to 10%.

We also examine the QoR improvement by progressively incorporating diffusion-driven optimization. Fig. 8 shows that CTs and CPAs optimized by AC-Refiner (Opt) achieve better QoR over the ones in the initial dataset (Init). Notably, the initial CPAs may not integrate well with the optimized CT than the default adders from the synthesis tool (Default), yet AC-Refiner still identifies CPAs that enhance the overall QoR of multipliers.
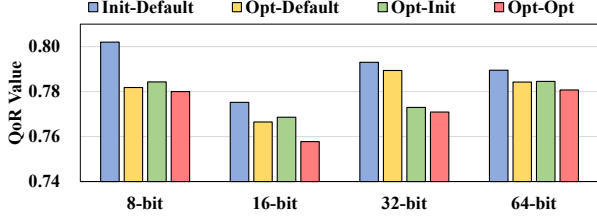
Fig. 8. Effect on QoR value upon progressive inclusion of diffusion-driven optimization. Labels indicate the source of CT and CPA.

TABLE I
SYSTOLIC ARRAY COMPARISON ON DELAY, AREA, AND AREA-DELAY
PRODUCT (ADP).

Objective	Method	8-bit			16-bit		
		Delay (ns)	Area (μm^2)	ADP ($\mu m^2 \cdot$ns)	Delay (ns)	Area (μm^2)	ADP ($\mu m^2 \cdot$ns)
Min Delay	Wallace [4]	1.0489	18896	19820	1.4164	60672	85935
	GOMIL [12]	1.1101	20384	22628	1.5559	58864	91586
	RL-MUL [19]	0.9746	18592	18119	1.3502	56176	75848
	ArithmeticTree [16]	0.9351	19440	18178	1.3311	60832	80973
	AC-Refiner (Ours)	**0.9317**	19376	**18052**	**1.2723**	55488	**70597**
Trade-off	Wallace [4]	1.2846	14816	19032	1.7833	45056	80348
	GOMIL [12]	1.2661	15056	19062	1.8992	45520	86451
	RL-MUL [19]	1.1825	14496	17141	1.6443	42048	69139
	ArithmeticTree [16]	1.1163	14144	15788	1.5504	40544	62859
	AC-Refiner (Ours)	**1.0479**	14432	**15123**	**1.3803**	41712	**57575**

To demonstrate the effectiveness of AC-Refiner in real-world designs, we incorporate multipliers from all approaches into multiply-accumulators of systolic arrays, which are commonly adopted in AI accelerators [40]. As shown in TABLE I, AC-Refiner retains its performance advantages across different scenarios, achieving the lowest delay when optimizing for timing, and the smallest area-delay product (ADP) when balancing timing and area.

C. Ablations

We perform ablation studies on the components of AC-Refiner to study their individual effects: gradient-guided sampling, self-reflection, and model fine-tuning.

Gradient-Guided Sampling. The QoR outcome $\tilde{y}$ of the conditional sampling process is influenced by two factors: target QoR y^* and guidance strength $s(t)$. We demonstrate their individual effects through the optimization of compressor trees across various bit-width.

In Fig. 9 we examine the impact of y^* on $\tilde{y}$. When y^* falls in the range of the initial dataset, the real QoR $\tilde{y}$ shows a strong correlation with y^*. This phenomenon shows that the neural cost predictor successfully captures the nuanced interplay between circuit structure and physical synthesis, offering effective guidance for conditional circuit generation. When y^* is set extremely low, the real QoR outcome generalizes beyond the minima of the initial dataset, highlighting the efficacy of diffusion-driven optimization in expanding Pareto frontiers.

Fig. 10 shows the effect of $s(t)$ on $\tilde{y}$, where the x-axis represents the logarithmic values of the coefficient in $s(t)$. Weak guidance strength leads to QoR distributions resembling the initial dataset. Whereas overly strong guidance disrupts the diffusion process, resulting in degraded QoR. Therefore, proper tuning of $s(t)$ is critical for achieving effective optimization outcomes.

Self-Reflection. To demonstrate that self-reflection enables effective gradient-based guidance, TABLE II shows the legalization results for compressor trees generated with different methods. The unconditional diffusion model attempts to interpolate across distinct circuit
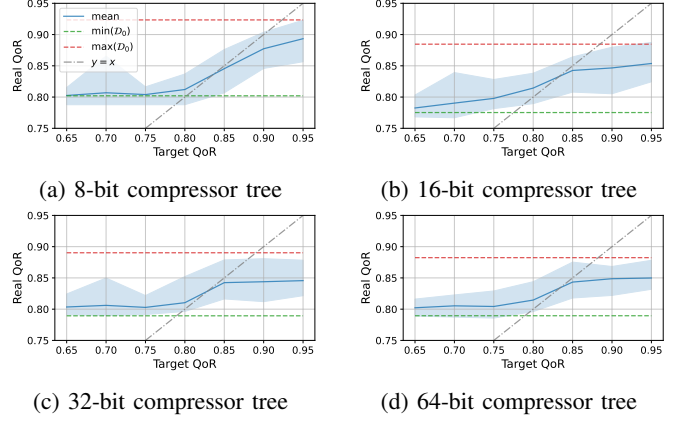

Fig. 9. Plot of real QoR versus target QoR for guided sampling. Dotted lines represent the min/max QoR of initial dataset $\tilde{D}_0$ and the ideal line $y = x$. The boundary of shaded regions indicates the min/max QoR of sampled designs.
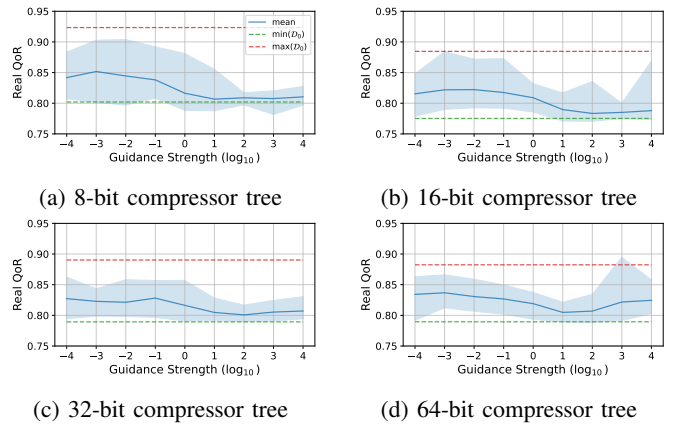

Fig. 10. Plot of real QoR versus guidance strength for guided sampling. Dotted lines represent the min/max QoR of initial dataset $\tilde{D}_0$. Shaded region boundaries indicates min/max QoR of sampled designs.

TABLE II
RESULTS OF AVERAGE STEPS TO LEGALIZE SAMPLED COMPRESSOR TREES.
THE MAXIMUM LEGALIZATION STEP IS SET TO 5000.

Method	8-bit	16-bit	32-bit	64-bit
Unconditional	0	4	33	103
Conditional w/o SR	7	4	179	81
Conditional w/ SR	0	2	1	7
VAE [41]	26	3757	N/A	N/A

structures, which may lead to substantial DRVs. For conditional models with conservative guidance strength, the discrepancy between predicted and target QoR cannot be effectively minimized, resulting in similar DRVs as the unconditional counterparts. By incorporating self-reflection to minimize $\mathcal{L}(y^*, \tilde{y})$, AC-Refiner restricts the diffusion model to interpolate on fewer targeted circuit structures. Consequently, the generated designs exhibit fewer DRVs and can be easily legalized.

Model Fine-Tuning. Fig. 11 shows the QoR optimization progress with iterative model fine-tuning. By enhancing the models with sampled high-quality designs, AC-Refiner achieves focused exploration near the Pareto frontier, with increased probability of generating high-potential circuit designs. As a result, AC-Refiner progressively improves multiplier QoR, until converging at an optimized state.

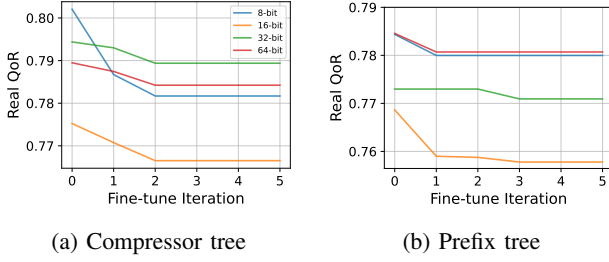(a) Compressor tree   (b) Prefix tree

Fig. 11. The effect of fine-tuning iteration on QoR optimization results. 0-th iteration represents minimum QoR from initial labeled dataset.
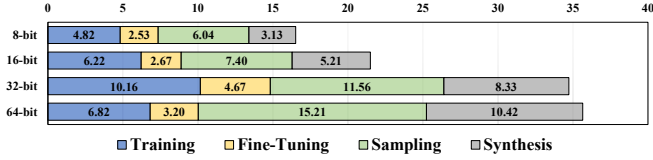

Fig. 12. Runtime of multiplier optimization (time unit: h).

D. Discussions

Runtime Analysis. Fig. 12 shows the runtime breakdown for comprehensive optimization of multipliers, including CT and CPA. The total runtime consists of four parts: (1) training diffusion models and cost predictors; (2) fine-tuning the models; (3) conditional sampling for new CT and CPA design; (4) QoR evaluation by running EDA tools. Unlike many RL-based approaches [15], [16], [19], AC-Refiner enables parallel synthesis on sampled designs, which significantly alleviates the bottleneck of EDA tool invocations. Although model training and design generation take comparable amount of time to synthesis in our current setup, both tasks exhibit high parallelism and can be accelerated proportionally with the number of GPUs employed.
Selection of Generative Models. In AC-Refiner, we adopt diffusion model over alternative generative approaches due to its high generation quality and high controllability. We observe that these advantages are crucial for achieving efficient optimization. As shown in TABLE II, we also apply VAEs [41] to generate CT designs with complex design constraints (Equation (5)-(9)). While VAEs have been adopted for prefix adder optimization in previous work [18], they perform poorly in generating correct CT designs. The produced CTs take excessive steps to legalize and even become irreparable. This suggests that the deficiency of VAEs in capturing fine-grained details [21] renders them unsuitable for optimizing complex arithmetic circuits.

V. CONCLUSION

This work introduces AC-Refiner, an arithmetic circuit optimization framework enabled with conditional diffusion models. By introducing conditional design generation process and model fine-tuning techniques, AC-Refiner consistently explores promising design candidates, enabling efficient design optimization. Future work will address the optimization for variable-width arithmetic components and explore extending the methodology to other components and broader applications.

ACKNOWLEDGEMENT

REFERENCES

[1] J. Sklansky, "Conditional-sum addition logic," *IRE Transactions on Electronic computers*, no. 2, pp. 226–231, 1960.

[2] Brent and Kung, "A regular layout for parallel adders," *IEEE Transactions on Computers*, vol. 100, no. 3, pp. 260–264, 1982.

[3] P. M. Kogge and H. S. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence equations," *IEEE Transactions on computers*, vol. 100, no. 8, pp. 786–793, 1973.

[4] C. S. Wallace, "A suggestion for a fast multiplier," *IEEE Transactions on electronic Computers*, no. 1, pp. 14–17, 1964.

[5] L. Dadda, *Some schemes for parallel multipliers*. IEEE Computer Society Press, 1990.

[6] J. Liu, Y. Zhu, H. Zhu, C.-K. Cheng, and J. Lillis, "Optimum prefix adders in a comprehensive area, timing and power design space," in *2007 Asia and South Pacific Design Automation Conference*. IEEE, 2007, pp. 609–615.

[7] T. Matsunaga and Y. Matsunaga, "Area minimization algorithm for parallel prefix adders under bitwise delay constraints," in *Proceedings of the 17th ACM Great Lakes symposium on VLSI*, 2007, pp. 435–440.

[8] S. Roy, M. Choudhury, R. Puri, and D. Z. Pan, "Towards optimal performance-area trade-off in adders by synthesis of parallel prefix structures," in *Proceedings of the 50th Annual Design Automation Conference*, 2013, pp. 1–8.

[9] T. Moto and M. Kaneko, "Prefix sequence: Optimization of parallel prefix adders using simulated annealing," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2018, pp. 1–5.

[10] S. Lin, B. Jiang, W. Sheng, and E. Young, "Size-optimized depth-constrained large parallel prefix circuits," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.

[11] M. Kumm and P. Zipf, "Pipelined compressor tree optimization using integer linear programming," in *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2014, pp. 1–8.

[12] W. Xiao, W. Qian, and W. Liu, "Gomil: Global optimization of multiplier by integer linear programming," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2021, pp. 374–379.

[13] D. Zuo, J. Zhu, C. Li, and Y. Ma, "Ufo-mac: A unified framework for optimization of high-performance multipliers and multiply-accumulators," *arXiv preprint arXiv:2408.06935*, 2024.

[14] Y. Ma, S. Roy, J. Miao, J. Chen, and B. Yu, "Cross-layer optimization for high speed adders: A pareto driven machine learning approach," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 12, pp. 2298–2311, 2018.

[15] R. Roy, J. Raiman, N. Kant, I. Elkin, R. Kirby, M. Siu, S. Oberman, S. Godil, and B. Catanzaro, "Prefixrl: Optimization of parallel prefix circuits using deep reinforcement learning," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 853–858.

[16] Y. Lai, J. Liu, D. Z. Pan, and P. Luo, "Scalable and effective arithmetic tree generation for adder and multiplier designs," *arXiv preprint arXiv:2405.06758*, 2024.

[17] H. Geng, Y. Ma, Q. Xu, J. Miao, S. Roy, and B. Yu, "High-speed adder design space exploration via graph neural processes," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 8, pp. 2657–2670, 2021.

[18] J. Song, A. Swope, R. Kirby, R. Roy, S. Godil, J. Raiman, and B. Catanzaro, "Circuitvae: Efficient and scalable latent circuit optimization," *arXiv preprint arXiv:2406.09535*, 2024.

[19] D. Zuo, Y. Ouyang, and Y. Ma, "Rl-mul: Multiplier design optimization with deep reinforcement learning," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[20] Y. Feng and C. Wang, "Gomarl: Global optimization of multiplier using multi-agent reinforcement learning," in *2024 2nd International Symposium of Electronics Design Automation (ISEDA)*. IEEE, 2024, pp. 728–733.

[21] I. Gulrajani, K. Kumar, F. Ahmed, A. A. Taiga, F. Visin, D. Vazquez, and A. Courville, "Pixelvae: A latent variable model for natural images," *arXiv preprint arXiv:1611.05013*, 2016.

[22] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli, "Deep unsupervised learning using nonequilibrium thermodynamics," in *International conference on machine learning*. PMLR, 2015, pp. 2256–2265.

[23] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *Advances in neural information processing systems*, vol. 33, pp. 6840–6851, 2020.

[24] J. Song, C. Meng, and S. Ermon, "Denoising diffusion implicit models," *arXiv preprint arXiv:2010.02502*, 2020.

[25] X. Li, J. Thickstun, I. Gulrajani, P. S. Liang, and T. B. Hashimoto, "Diffusion-lm improves controllable text generation," *Advances in Neural Information Processing Systems*, vol. 35, pp. 4328–4343, 2022.

[26] R. Huang, M. W. Lam, J. Wang, D. Su, D. Yu, Y. Ren, and Z. Zhao, "Fastdiff: A fast conditional diffusion model for high-quality speech synthesis," *arXiv preprint arXiv:2204.09934*, 2022.

[27] K. E. Wu, K. K. Yang, R. van den Berg, S. Alamdari, J. Y. Zou, A. X. Lu, and A. P. Amini, "Protein structure generation via folding diffusion," *Nature communications*, vol. 15, no. 1, p. 1059, 2024.

[28] C. Chi, S. Feng, Y. Du, Z. Xu, E. Cousineau, B. Burchfiel, and S. Song, "Diffusion policy: Visuomotor policy learning via action diffusion," *arXiv preprint arXiv:2303.04137*, 2023.

[29] Z. Sun and Y. Yang, "Difusco: Graph-based diffusion solvers for combinatorial optimization," *Advances in Neural Information Processing Systems*, vol. 36, pp. 3706–3731, 2023.

[30] L. Liu, Y. Ren, Z. Lin, and Z. Zhao, "Pseudo numerical methods for diffusion models on manifolds," *arXiv preprint arXiv:2202.09778*, 2022.

[31] T. Chen, R. Zhang, and G. Hinton, "Analog bits: Generating discrete data using diffusion models with self-conditioning," *arXiv preprint arXiv:2208.04202*, 2022.

[32] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*. Springer, 2015, pp. 234–241.

[33] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[34] A. Bansal, H.-M. Chu, A. Schwarzschild, S. Sengupta, M. Goldblum, J. Geiping, and T. Goldstein, "Universal guidance for diffusion models," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 843–852.

[35] C. Wolf, "Yosys open synthesis suite," https://yosyshq.net/yosys/.

[36] T. Ajayi and D. Blaauw, "Openroad: Toward a self-driving, open-source digital layout implementation tool chain," in *Proceedings of Government Microcircuit Applications and Critical Technology Conference*, 2019.

[37] NanGate Inc, "Nangate freepdk45 open cell library," 2008.

[38] R. Brayton and A. Mishchenko, "Abc: An academic industrial-strength verification tool," in *Computer Aided Verification: 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings 22*. Springer, 2010, pp. 24–40.

[39] A. Mahzoon, D. Große, and R. Drechsler, "Revsca-2.0: Sca-based formal verification of nontrivial multipliers using reverse engineering and local vanishing removal," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 5, pp. 1573–1586, 2021.

[40] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th annual international symposium on computer architecture*, 2017, pp. 1–12.

[41] D. P. Kingma, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.