

Exploring the Boundary of Diffusion-based Methods for Solving Constrained Optimization

Shutong Ding^{1,5} Yimiao Zhou¹ Ke Hu¹ Xi Yao³ Junchi Yan²
Xiaoying Tang⁴ Ye Shi^{1,5,*}

¹ShanghaiTech University ²Shanghai Jiao Tong University

³China Mobile Communications Company Limited Research Institute

⁴The Chinese University of Hong Kong, Shenzhen

⁵MoE Key Laboratory of Intelligent Perception and Human Machine Collaboration

{dingsht, zhouym3, huke2024, shiye}@shanghaitech.edu.cn
yaoxi@cmsr.chinamobile.com
yanjunchi@sjtu.edu.cn
tangxiaoying@cuhk.edu.cn

Abstract

Diffusion models have achieved remarkable success in generative tasks such as image and video synthesis, and in control domains like robotics, owing to their strong generalization capabilities and proficiency in fitting complex multimodal distributions. However, their full potential in solving Continuous Constrained Optimization problems remains largely underexplored. Our work commences by investigating a two-dimensional constrained quadratic optimization problem as an illustrative example to explore the inherent challenges and issues when applying diffusion models to such optimization tasks and providing theoretical analyses for these observations. To address the identified gaps and harness diffusion models for Continuous Constrained Optimization, we build upon this analysis to propose a novel diffusion-based framework for optimization problems called **DiOpt**. This framework operates in two distinct phases: an initial warm-start phase, implemented via supervised learning, followed by a bootstrapping training phase. This dual-phase architecture is designed to iteratively refine solutions, thereby improving the objective function while rigorously satisfying problem constraints. Finally, multiple candidate solutions are sampled, and the optimal one is selected through a screening process. We present extensive experiments detailing the training dynamics of DiOpt, its performance across a diverse set of Continuous Constrained Optimization problems, and an analysis of the impact of DiOpt’s various hyperparameters. The official implementation of DiOpt is provided in <https://github.com/wadx2019/diopt>.

1 Introduction

Constrained optimization with hard constraints constitutes a cornerstone of real-world decision-making systems, spanning critical applications from power grid operations [35, 13, 6, 40] and wireless communications [16] to robotic motion planning [25, 9]. Traditional numerical methods [33, 52, 44] face a fundamental trade-off: either simplify problems through restrictive relaxations (e.g., linear programming approximations) or endure prohibitive computational costs, both unsuitable for safety-

*Corresponding author.

critical and real-time systems. Learning-based approaches [15, 36, 48, 49, 21, 51] emerged as promising alternatives by training neural networks to predict solutions directly, yet they suffer from two critical limitations, as shown in Figure 1: 1) Deterministic one-to-one generation methods exhibit poor performance in handling multi-value mappings [31]. 2) The learned constraint points show limited overlap with the feasible region.

Recent efforts to address these limitations have turned to diffusion models [20, 41, 42], leveraging their multimodal sampling capacity to generate diverse solution candidates [25, 34]. While this mitigates the single-point failure risk, state-of-the-art methods still struggle with systematic constraint satisfaction due to persistent distributional misalignment. In addition, they rely heavily on supervised training with labeled datasets, a practical bottleneck given the NP-hard nature of many constrained optimization problems. Additionally, these diffusion-based optimization methods [25, 34] often require complex guidance or projection procedures to refine solutions during sampling, resulting in slow inference speeds that scale poorly with problem dimensionality. Besides, some diffusion-based methods attempt to generate constraint-satisfying points through gauge mapping [28]. However, these methods rely on a large number of feasible samples for training, despite the fact that the optimal solution in typical optimization problems is a single point in the distribution.

To address the absence of an efficient way that bridges the diffusion model and constrained optimization. We first analyze the problem of directly applying the supervised diffusion training paradigm in optimization. Then, to address these problems, we present DiOpt, a self-supervised diffusion framework that fundamentally rethinks how generative models interact with constrained solution spaces. Concretely, DiOpt introduces a target distribution designed to maximize overlap with the constrained solution manifold and develops a bootstrapped self-training mechanism that assigns weights to candidate solutions based on the severity of constraint violations and optimality gaps. Due to the instability of sampling, the current round of diffusion model sampling may not necessarily outperform its previous round, and directly using these points for training could degrade the model. Thus, DiOpt also introduces a look-up table to store historically optimal sampling points, thereby ensuring effective training. Our contributions are threefold:

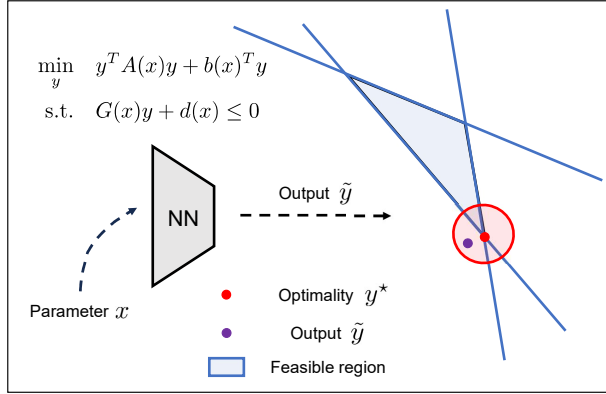


Figure 1: A schematic geometric interpretation of feasibility challenges in learning-based optimization. The feasible region (blue) and neural network’s output distribution (red), e.g., a Gaussian output by a diffusion model, fundamentally exhibit a small overlap, particularly under high dimensionality and multiple constraints. This distributional misalignment breaks constraint satisfaction.

- We investigate the performance of diffusion models in solving optimization problems and empirically demonstrate that training diffusion models via supervised learning leads to a distribution resembling a Gaussian centered around optimal points. Building on this observation, we theoretically analyze why such diffusion models encounter infeasibility issues in constrained optimization.
- To address this, we propose DiOpt, a diffusion model training framework that integrates both supervised and self-supervised learning. By introducing a weighted bootstrapping mechanism during the self-supervised phase, our approach effectively enhances feasibility while maintaining a tight optimality gap.
- We evaluate DiOpt across diverse tasks, including Quadratic Programming (QP), QP with Sine Regularization (QPSR), Concave QP (CQP), Alternating Current Optimal Power Flow (ACOPF), and Motion Retargeting (Retargeting), demonstrating its training dynamics, comparative advantages over baseline methods, and comprehensive ablation studies. Our results confirm that the framework achieves a balanced trade-off between optimality and feasibility across various optimization scenarios.

2 Related Works

Learning to Optimize. To address the high computational cost of classical optimization solvers, Learning to Optimize (L2O) has emerged as a promising approach that leverages machine learning techniques to solve real-world constrained optimization problems. The L2O methods can be generally categorized into two groups: 1) assisting traditional solvers with machine learning techniques to improve their efficiency or performance; 2) approximating the input-output mapping of optimization problems using data-driven models. In the first category, reinforcement learning (RL) has been widely adopted to design better optimization policies for both continuous [27] and discrete decision variables [32, 43]. Additionally, neural networks have been used to predict warm-start points for optimization solvers, significantly reducing convergence time [1, 14]. In the second category, deep learning models have been employed to directly approximate solutions for specific problems. For instance, [18, 8] utilize neural networks to solve the OPF problems efficiently. To further improve constraint satisfaction, recent works have integrated advanced techniques into the training process. For example, [15] introduced gradient-based correction, while [36] incorporated primal-dual optimization methods to ensure the feasibility of the learned solutions.

Neural Solvers with Hard Constraints. Despite the challenge of devising general-purpose neural solvers for arbitrary hard constraints, there are also some tailored neural networks (with special layers) for constrained optimization, especially for combinatorial optimization. In these methods, the problem-solving can be efficiently conducted by a single forward pass inference. For instance, in graph matching, or more broadly the quadratic assignment problem, there are a series of works [46, 17] introducing the Sinkhorn layer into the network to enforce the matching constraint. Another example is the cardinality-constrained problem; similar techniques can be devised to ensure the constraints [5, 45, 7]. However, as aforementioned, these layers are specifically designed and cannot be used in general settings as addressed in this paper. Moreover, it often requires ground truth for supervision, which cannot be obtained easily in real-world cases.

Generative Models for Constrained Optimization. Generative methods, characterized primarily by sampling from noise, involve models that transform random noise (typically a standard Gaussian distribution) into a specified distribution. To date, a considerable number of studies with diverse methodologies have focused on this area. One category of methods is derived from modifications to the sampling process of classical diffusion models [50, 24, 34]. By directly transforming the optimization problem into a probability function to replace the original score function in the sampling process, this class of methods that do not require training for solutions has been developed. Another category employs neural networks to process noise for transformation into a specified distribution, such as methods based on CVAE [26] or GAN [37]. Additionally, there are methods based on diffusion models; [4] simulates the distribution of optimization problems through compositional operations on multiple score functions. [25] forces the model to learn feasible solutions by adding violation penalties. [31] provides a theoretical guarantee for such methods. This process can be applied multiple times to enhance the quality of the solution. To enforce the feasibility of generated solutions, PDM [10] performs a projection after each diffusion step, while CGD [23] proposes a targeted post-processing method for the problems it addresses. For combinatorial optimization, T2T [29] and Fast T2T [30] also propose a training-to-testing framework.

Remark. However, most of the above methods are trained in a supervised learning paradigm, which tends to suffer from the small overlapping problem mentioned in Figure 1. In contrast, the proposed DiOpt trained in a bootstrapping paradigm can converge to the mapping to the near-optimal feasible region that has a large overlap with the feasible region and does not introduce extra cost on supervised data collection. Besides, it is worth noting that DiOpt focuses on constrained nonconvex optimization with continuous (real-valued) variables with smooth or nonconvex objectives/constraints (e.g., robotics, smart grids, communication systems), which fundamentally differs from combinatorial optimization [29, 30, 22, 39, 38] (e.g., TSP) in both problem structure and methodology.

3 Preliminaries

Problem Statement. Learning-to-optimize attempts to solve a family of optimization problems as,

$$\begin{aligned} \min_{\mathbf{y}} \quad & f(\mathbf{y}; \mathbf{x}) \\ \text{subject to} \quad & g_i(\mathbf{y}; \mathbf{x}) \leq 0 \quad i = 1, \dots, m \\ & h_j(\mathbf{y}; \mathbf{x}) = 0 \quad j = 1, \dots, n \end{aligned} \tag{1}$$

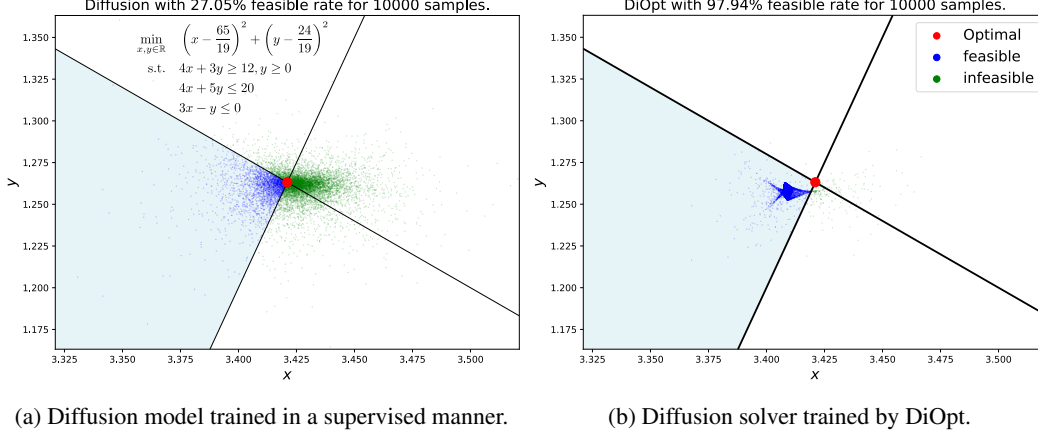


Figure 2: **Comparison between supervised diffusion and DiOpt on a toy example.** It can be observed that the distribution of diffusion trained in a supervised manner (a) approximates a Gaussian distribution centered around the optimal point, which leads to a low feasibility rate. For detailed settings of this toy example, please refer to the supplementary materials for more details.

where $\mathbf{y}$ is the decision variable of the optimization problem parameterized by $\mathbf{x}$. We can use machine learning techniques to learn the mapping from x to its corresponding solution $\mathbf{y}^*$ in an optimization problem family with a similar problem structure. With this mapping, the solution can be calculated faster and more efficiently compared with the classical optimization solver.

Diffusion Models. Denoising diffusion probabilistic models (DDPM) [20] are generative models that create high-quality data by learning to reverse a gradual forward noising process applied to the training data. Given a dataset $\{\mathbf{x}_0^i\}_{i=1}^N$ for $\mathbf{x}_0^i \sim q(\mathbf{x}_0)$, the forward process $\{\mathbf{x}_{0:T}\}$ adds Gaussian noise to the data with pre-defined schedule $\{\beta_{1:T}\}$:

$$q(\mathbf{x}_t | \mathbf{x}_{t-1}) := \mathcal{N}(\mathbf{x}_t; \sqrt{1 - \beta_t} \mathbf{x}_{t-1}, \beta_t \mathbf{I}). \quad (2)$$

Using the Markov chain property, we can obtain the analytic marginal distribution of conditioned on $\mathbf{x}_0$:

$$q(\mathbf{x}_t | \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t} \mathbf{x}_0, (1 - \bar{\alpha}_t) \mathbf{I}), \forall t \in \{1, \dots, T\}, \quad (3)$$

where $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{s=0}^t \alpha_s$. Given $\mathbf{x}_0$, it's easy to obtain a noisy sample by the reparameterization trick.

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \boldsymbol{\epsilon}, \boldsymbol{\epsilon} \in \mathcal{N}(\mathbf{0}, \mathbf{I}). \quad (4)$$

DDPMs use parameterized models $p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \Sigma_\theta(\mathbf{x}_t, t))$ to fit $q(\mathbf{x}_t | \mathbf{x}_{t-1}, \mathbf{x}_0)$ to reverse forward diffusion process, where θ denotes the learnable parameters. The practical implementation involves directly predicting the Gaussian noise $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$ to minimize the evidence lower bound loss. With $\boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, the loss in DDPM takes the form of:

$$\mathbb{E}_{t \sim [1, T], \mathbf{x}_0, \boldsymbol{\epsilon}_t} [\|\boldsymbol{\epsilon}_t - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \boldsymbol{\epsilon}_t, t)\|^2]. \quad (5)$$

After training, DDPM generates samples according to

$$\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) + \sigma_t \mathbf{z} \right), \quad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \quad t = T, \dots, 1 \quad (6)$$

In this paper, we consider the diffusion process to be deterministic when $\sigma_t = 0$.

4 Why Supervised Diffusion Tends to Generate Infeasible Solutions

As mentioned in Section 1, while diffusion models have achieved significant success in image editing, video generation, and robotic control thanks to their powerful multimodal fitting capabilities; how to apply them to constrained continuous optimization problems effectively and then generate near-optimal solutions without violating the complex (nonconvex) constraints, remains largely unexplored.

To bridge this gap, we conduct a toy example, which applies diffusion models trained in a supervised paradigm to a two-dimensional QP problem with linear constraints. Figure 2 shows that this kind of diffusion model tends to generate points around the optimal solution, and the distribution is an approximate Gaussian distribution. Nevertheless, the feasible region only has a small overlapping with the Gaussian ball, which leads to the diffusion solver violating the constraints with a high probability (i.e., the number of blue points (feasible) is greater than that of the green points).

Based on this finding, we also present a theoretical proof, as Theorem 1, that in high-dimensional optimization problems, exemplified by linear programming, diffusion models trained via supervised learning become increasingly prone to constraint violations, with the violation probability growing exponentially toward one as the dimensionality increases.

Theorem 1. (Feasibility in Linear Programming.) *Given a d -dimensional linear programming problem of the form:*

$$\min_{x \in \mathbb{R}^d} c^T x \quad \text{subject to} \quad a_i^T x \leq b_i, \quad i = 1, \dots, N,$$

where a_i represents a unit normal vector, drawn independently and uniformly from the unit sphere S^{d-1} . Let x^* be the unique solution to this linear programming problem, and we define a neighborhood ball $B_\epsilon(x^*) = \{x : \|x - x^*\| \leq \epsilon\}$. For sufficiently small $\epsilon > 0$, there is an asymptotic bound on the probability that a point uniformly sampled from $B_\epsilon(x^*)$ lies in the feasible region $\mathcal{C}$.

$$\mathbb{P}_{x \sim B_\epsilon(x^*)} (x \in \mathcal{C}) \approx \frac{1}{2^d}.$$

For more general nonlinear inequality constraints, one can similarly linearize them within an infinitesimal neighborhood and reduce to the case described in this theorem. This theorem also confirms our findings in the toy example in Figure 2 where 27.05% feasible rate is close to $\frac{1}{2^2}$. The proof leverages results from stochastic geometry [11, 47], with details provided in supplementary materials. Hence, it suggests that training diffusion models with supervised samples is not an ideal choice for constrained optimization problems, and a more effective method needs to be provided to better incorporate the diffusion model into constrained optimization.

5 Methodology

To address the issue mentioned in Section 4, we propose DiOpt, a self-supervised **D**iffusion-based learning framework for Constrained **O**ptimization in this section. As shown in Figure 3, DiOpt trains the diffusion model in a bootstrapping mechanism via weighted variational loss of diffusion and applies the candidate *solution selection* technique during the evaluation stage to further boost the solution quality. The training procedure of DiOpt is presented in Algorithm 1.

Target Distribution for Diffusion Training. To apply the diffusion model to optimization problems, we train it using the following objective:

$$\mathbb{E}_{\mathbf{x}, \mathbf{y}^*, \epsilon, t} [\|\epsilon - \epsilon_\theta(\mathbf{y}_t, \mathbf{x}, t)\|_2^2] \quad (7)$$

where $\mathbf{y}^* \in \mathbb{R}^{d_y}$ denotes the optimal solution corresponding to $\mathbf{x} \in \mathbb{R}^{d_x}$ and ϵ_θ is the noise network of diffusion model $\mathcal{D}_\theta$. The near-optimal distribution learned by the diffusion model trained in this manner is ultimately similar to that shown in Figure 1. However, in high-dimensional settings, the intersection between the near-optimal region of the diffusion model and the feasible region becomes too small, which leads to infeasibility issues when applying the diffusion model to optimization problems. In that case, it is necessary to define another target distribution for diffusion models to enforce their constraint satisfaction. Referring to [31], we define the target distribution that corresponds to the near-optimal feasible region as

$$p(\mathbf{y}; \mathbf{x}) \sim \mathbb{I}_{\mathcal{C}(\mathbf{x})}(\mathbf{y}) \exp(-\beta f(\mathbf{y}; \mathbf{x})), \quad (8)$$

where $\mathbf{y} \sim \mathcal{D}_\theta(\mathbf{x})$, $\mathcal{C}(x)$ indicates the constraint region that satisfies $g_i(\mathbf{y}; \mathbf{x}) \leq 0$, $h_j(\mathbf{y}; \mathbf{x}) = 0$ and $\mathbb{I}_{\mathcal{C}(\mathbf{x})}$ is the indicator function that judges the satisfaction of the constraint.

Training Diffusion with Bootstrapping. While [31] has argued that it is essential to approximate the above target distribution rather than merely an optimal point, it is difficult to sample from (8) efficiently. Therefore, constructing a dataset based on the target distribution for training diffusion

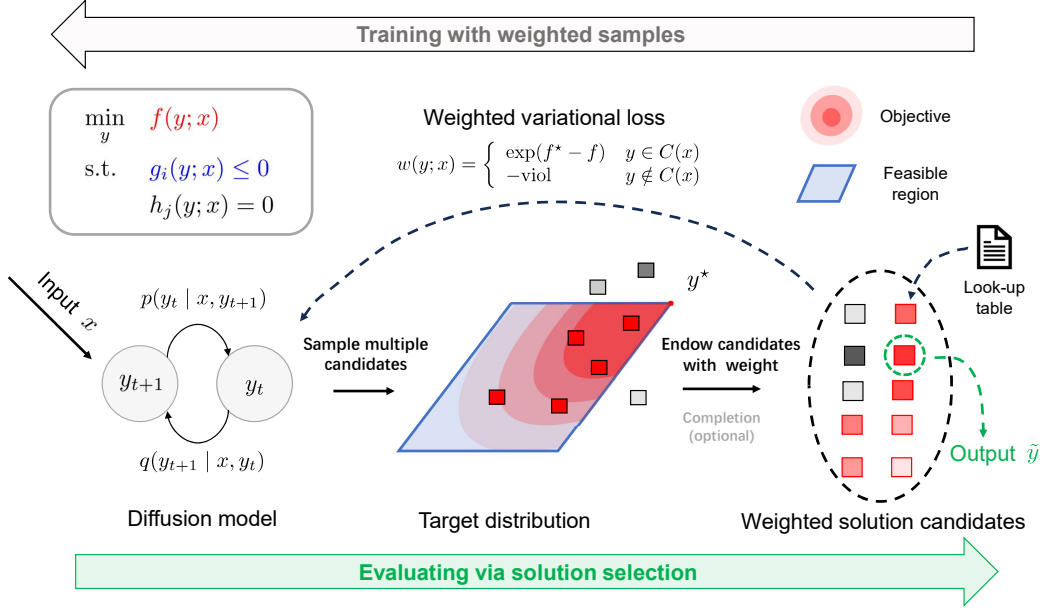


Figure 3: Training and evaluating procedure of DiOpt.

models is also impossible. To avoid this problem, we divert our attention to self-supervised learning. Motivated by [12], we design a diffusion-based learning framework for constrained optimization, which implements diffusion training in a bootstrapping manner. In this way, it can naturally converge to the target distribution without manufacturing a corresponding dataset.

Concretely, we try to utilize the parallelism of the diffusion model and generate a certain number of candidate points for one specific problem, and then endow the candidate points with weights related to the constraint violation and objective value. The diffusion model will be trained with these weighted candidate points according to:

$$\mathcal{L}(\theta) := \mathbb{E}_{\mathbf{x}, \mathbf{y}, \epsilon, t} \left[\omega(\mathbf{y}; \mathbf{x}) \|\epsilon - \epsilon_\theta(\mathbf{y}_t, \mathbf{x}, t)\|_2^2 \right], \quad (9)$$

Here, the weight can be viewed as the importance of training points. Hence, the diffusion model can approximately converge to the distribution defined by the weight function after enough iterations. Thus, the weight function design is essential to ensure that the diffusion model to converge to the target distribution. Based on that, we classify points into two cases and design two different weight functions for them:

$$\omega(\mathbf{y}; \mathbf{x}) := \begin{cases} \exp(f^*(\mathbf{x}) - f(\mathbf{y}; \mathbf{x})) & \mathbf{y} \in \mathcal{C}(\mathbf{x}) \\ -\sum_i \max(g_i(\mathbf{y}; \mathbf{x}), 0) & \mathbf{y} \notin \mathcal{C}(\mathbf{x}) \end{cases}, \quad (10)$$

where $f^*(\mathbf{x})$ indicates the optimal objective value of the optimization problem with parameter $\mathbf{x}$. One of the principal ideas for this weight function is that all the feasible points have positive weights and the infeasible points have negative weights. In that case, the diffusion model will converge to the feasible region and then consider the optimality of the points inside the constraint region. Besides, it is worth noting that $f^*(\mathbf{x})$ can be replaced with the estimated lower bound of the objective function. This term actually avoids the numerical explosion of the exponential function. Furthermore, to ensure that the trained model prioritizes feasibility over optimality, we reset the weight function every training epoch as follows:

$$\omega(\mathbf{y}; \mathbf{x}) = -\sum_i \max(g_i(\mathbf{y}; \mathbf{x}), 0) \quad (11)$$

Remark. If we continue updating diffusion with the weight of (10), the output of the diffusion solver will resemble that of supervised learning and will be prone to generating infeasible points. The detailed proof can be referred to in the supplementary materials. Hence, this formulation penalizes constraint violations ($g_i(\mathbf{y}; \mathbf{x}) > 0$), systematically steering the model toward the feasible region.

Algorithm 1 Bootstrapping-based training process of DiOpt

Input: training dataset X , the objective function $f(\mathbf{y}; \mathbf{x})$ and constraints $h(\mathbf{y}; \mathbf{x}), g(\mathbf{y}; \mathbf{x})$, the noise network ϵ_θ of diffusion model $\mathcal{D}_\theta$, look-up table $\mathcal{B}$, Number of Training Epochs N , Supervised Ratio r_s , Number of Training Samples K_t .

```
for  $n = 0$  to  $N - 1$  do
  if  $n \leq \lfloor r_s \cdot N \rfloor$  then
    Train  $\mathcal{D}_\theta$  by (7)
    Continue
  end if
  if  $n \bmod 2 = 0$  then
    Reset the weight function as (10)
  else
    Reset the weight function as (11)
    // reset diffusion with feasible points
  end if
  for  $\mathbf{x}_i$  in  $X$  do
     $\mathbf{y}_1, \dots, \mathbf{y}_{K_t} \stackrel{\text{i.i.d}}{\sim} \mathcal{D}_\theta(\mathbf{x}_i)$ 
     $\mathbf{y}_{\text{best}} \leftarrow \mathcal{B}(\mathbf{x}_i)$ 
    Endow  $\mathbf{y}_1, \dots, \mathbf{y}_{K_t}, \mathbf{y}_{\text{best}}$  with weights according to (12)
    Train  $\mathcal{D}_\theta$  by (13)
    Update  $\mathcal{B}(\mathbf{x}_i)$  with  $\mathbf{y}_1, \dots, \mathbf{y}_{K_t}$  by (14)
  end for
end for
Return  $\mathcal{D}_\theta$ 
```

However, there is still a problem to be resolved in our weight function. As illustrated in [12], the weight in (9) must always be positive. Hence, we perform a modification on the final weight when there exists a candidate point with a negative weight.

$$\tilde{\omega}(\mathbf{y}; \mathbf{x}) = \max(\omega(\mathbf{y}; \mathbf{x}) - \bar{\omega}, 0), \quad \text{where } \bar{\omega} = \frac{1}{K} \sum_{i=1}^K \omega(\mathbf{y}_i; \mathbf{x}), \quad \mathbf{y}_i \stackrel{\text{i.i.d}}{\sim} \mathcal{D}_\theta(\mathbf{x}) \quad (12)$$

As shown in [12], $\tilde{\omega}$ is equivalent to ω for diffusion training, we can ensure the diffusion model converges to the target distribution with the modified weight $\tilde{\omega}$. This leads to the following approach:

$$\mathbb{E}_{\mathbf{x}, \tilde{\mathbf{y}}, \epsilon, t} \left[\tilde{\omega}(\tilde{\mathbf{y}}, \mathbf{x}) \|\epsilon - \epsilon_\theta(\tilde{\mathbf{y}}_t, \mathbf{x}, t)\|_2^2 \right], \quad \tilde{\mathbf{y}} = \arg \max_{\mathbf{y} \in \{\mathbf{y}_1, \dots, \mathbf{y}_K\}} \omega(\mathbf{y}; \mathbf{x}) \quad (13)$$

Due to the curse of dimensionality [2], training the model purely based on bootstrapping may make it difficult for the diffusion model to explore sufficiently good solutions. Therefore, at the initial stage of training, we utilize (7) to train the diffusion model and guide it towards the vicinity of the optimal point. Afterwards, (13) is employed to further train the model, ensuring feasibility. Throughout all training epochs, the proportion of supervised learning is denoted as r_s .

Remark. Note that for problems that have equality constraints $h(\mathbf{y}; \mathbf{x}) = 0$, we choose to generate partial variables via the diffusion model and use an equation solver to complete it like [15, 13]. This is because the nature of equality constraints is the reduction of free variables. So it is not appropriate to split them into two corresponding inequality constraints here.

Quality Assurance Mechanism. Although diffusion models can employ bootstrapping mechanisms for self-supervised training, not every sampling iteration guarantees improvement over previous results. Therefore, a look-up table is required to archive historically optimal solutions, preventing performance degradation. We implement this through a look-up table $\mathcal{B} : \mathbb{R}^{d_x} \rightarrow \mathbb{R}^{d_y}$ that stores prior optimal solutions. Let $\mathbf{y}_{\text{best}}$ denote the optimal solution stored in $\mathcal{B}$ for a given problem parameter $\mathbf{x}$. The update rule for $\mathcal{B}$ is formulated as:

$$\mathcal{B}(\mathbf{x}) = \arg \max_{\mathbf{y} \in \{\mathbf{y}_1, \dots, \mathbf{y}_K, \mathbf{y}_{\text{best}}\}} \omega(\mathbf{y}; \mathbf{x}) \quad (14)$$

After Training, we employ *Solution Selection* for Inference:

$$\tilde{\mathbf{y}} = \arg \max_{\mathbf{y} \in \{\mathbf{y}_1, \dots, \mathbf{y}_K\}} \omega(\mathbf{y}; \mathbf{x}), \quad \mathbf{y}_1, \dots, \mathbf{y}_K \stackrel{\text{i.i.d}}{\sim} \mathcal{D}_\theta(\mathbf{x}) \quad (15)$$

Table 1: Performance comparison (mean $\pm$ std. dev.) across different methods.

Problem	Method	Objective $\downarrow$	Gap(%) $\downarrow$	Ineq Mean $\downarrow$	Ineq Max $\downarrow$	Ineq Num Viol $\downarrow$	Feasibility(%) $\uparrow$
QP	DC3 [15]	-7.47 ± 0.78	31.56 ± 6.11	0.00 ± 0.00	0.10 ± 0.26	0.57 ± 1.36	73.83 ± 0.00
	MBD [34]	N/A	N/A	N/A	N/A	N/A	0.05 ± 0.00
	DiOpt	-10.53 ± 0.47 ✓	3.45 ± 1.23	0.00 ± 0.00	0.02 ± 0.06	0.28 ± 0.51	79.11 ± 0.00
QPSR	DC3	-6.30 ± 1.02 ✗	35.57 ± 9.60	0.01 ± 0.01	0.45 ± 0.68	2.20 ± 3.36	45.38 ± 0.00
	MBD	N/A	N/A	N/A	N/A	N/A	0.02 ± 0.00
	DiOpt	-9.53 ± 0.43 ✓	2.48 ± 0.76	0.00 ± 0.00	0.01 ± 0.04	0.24 ± 0.48	82.30 ± 0.00
CQP	DC3	-62.07 ± 15.25 ✗	67.32 ± 40.73	0.19 ± 0.06	6.20 ± 3.43	49.09 ± 6.92	0.00 ± 0.00
	MBD	N/A	N/A	N/A	N/A	71.18 ± 3.77	0.00 ± 0.00
	DiOpt	-34.56 ± 0.64 ✓	7.06 ± 1.37	0.00 ± 0.00	0.05 ± 0.20	0.74 ± 1.74	70.35 ± 0.00
RETARGETING	DC3	1.78 ± 0.49	4.05 ± 4.52	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	100.00 ± 0.00
	MBD	4.41 ± 1.06 ✗	164.58 ± 56.41	0.07 ± 0.01	2.54 ± 0.58	0.98 ± 0.06	1.79 ± 0.00
	DiOpt	1.77 ± 0.49 ✓	3.26 ± 5.74	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	100.00 ± 0.00
ACOPF57	DC3	3.82 ± 0.63	0.40 ± 0.77	0.00 ± 0.00	0.00 ± 0.00	0.06 ± 0.28	95.00 ± 0.00
	MBD	4.77 ± 1.32 ✗	23.07 ± 14.17	0.02 ± 0.01	1.28 ± 0.53	4.21 ± 1.39	0.00 ± 0.00
	DiOpt	3.81 ± 0.64 ✓	0.18 ± 0.73	0.00 ± 0.00	0.00 ± 0.01	0.18 ± 0.52	87.00 ± 0.00
ACOPF118	DC3	13.44 ± 1.24 ✗	2.50 ± 0.20	0.00 ± 0.00	0.03 ± 0.03	0.98 ± 1.05	37.00 ± 0.00
	MBD	17.39 ± 0.34 ✗	33.71 ± 11.84	0.03 ± 0.01	3.87 ± 2.29	21.89 ± 0.97	0.00 ± 0.00
	DiOpt	13.39 ± 1.23 ✓	2.18 ± 0.10	0.00 ± 0.00	0.01 ± 0.03	0.19 ± 0.46	86.60 ± 0.00

By selecting a sufficiently large K , we ensure a high probability of obtaining a near-optimal solution.

Remark. Considering the impact on training time, using an excessively large K during training can lead to prohibitively long training durations. Therefore, we introduce the notion of the Number of Training Samples, denoted as K_t , to represent the number of samples used during training, while K refers to the number of samples used during inference.

6 Experiment

Building upon the illustrative Toy Example demonstration in Figure 2b, we now conduct a comprehensive empirical evaluation of DiOpt on a range of more complex and challenging optimization tasks. These tasks include manually constructed problems (QP, QPSR, and CQP) as well as challenging real-world benchmarks such as AC Optimal Power Flow (ACOPF), a non-convex optimization problem in power systems, and Motion Retargeting [19], which maps human motion to a humanoid robot under kinematic constraints. Spanning diverse domains, these problems highlight DiOpt’s versatility and generalization ability. We compare DiOpt against supervised MLPs to demonstrate the advantages of our diffusion-based approach. Furthermore, we analyze DiOpt’s training dynamics, showing how the learned diffusion process balances feasibility and near-optimality. Detailed benchmark formulations, experimental settings, and ablation results are provided in supplementary materials.

6.1 Benchmarking Results: Performance Comparison

Table Notes. Bold numbers indicate the best metric. “ N/A ” denotes an invalid or abnormal result. Objective marked with ✗ indicate that the objective value is not meaningful due to infeasibility (Feasibility $< 50\%$). Conversely, an objective marked with ✓ indicates that the objective value is optimal with feasibility (Feasibility $\geq 50\%$). We define an inequality constraint $g_i(\mathbf{y}; \mathbf{x})$ as violated if $g_i(\mathbf{y}; \mathbf{x}) > \epsilon$, where ϵ is the threshold for inequality violation. The following metrics quantify inequality constraint violations: “Ineq Mean” (mean violation value), “Ineq Max” (maximum violation value), and “Ineq Num Viol” (number of violated constraints). The threshold ϵ is fixed at 0.01 for all experiments.

Results. Table 1 shows DiOpt achieves the best Gap across all tasks and the highest feasibility on most, except for ACOPF57. This suggests DiOpt effectively leverages diffusion’s multi-solution nature and bootstrapping for a strong balance between feasibility and near-optimality. In contrast, MBD’s abnormal results stem from incomplete equality constraints, causing point selection issues (see supplementary materials for results without these). DC3 struggles on CQP because its objective gradient increases sharply with $\mathbf{y}$, a challenge DiOpt is less affected by, demonstrating its robustness.

Further comparison in Table 2 shows Diffusion trained in a supervised manner (i.e., DiOpt with $r_s = 1$) significantly outperforms MLP in feasibility and optimality gap on most tasks. This

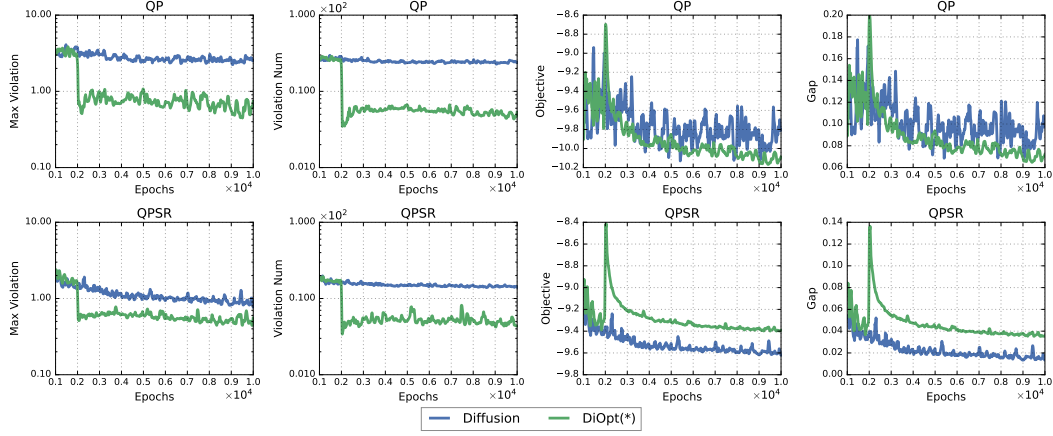


Figure 4: Training progression across different optimization tasks, with each row representing a unique optimization problem (indicated in subtitles) and each column showing specific performance metrics. The horizontal axes track training epochs, while the vertical axes display corresponding metric values with detailed scales labeled for each plot. Notably, DiOpt achieves sudden gains in constraint satisfaction approximately at 2000 epochs when compared with Diffusion. The bootstrapping mechanism initially causes an upward perturbation in the objective function and optimality gap, which then progressively decays to near-optimal.

Table 2: Comparison between Diffusion Model and Vanilla Neural Network

Problem	Method	Objective↓	Gap(%)↓	Ineq Mean↓	Ineq Max↓	Ineq Num Viol↓	Feasibility(%)↑
ACOPF57	Diffusion	3.81 ± 0.64 ✓	0.13 ± 0.61	0.00 ± 0.00	0.02 ± 0.02	0.56 ± 0.72	57.40 ± 0.00
	MLP	3.81 ± 0.63 ×	0.33 ± 0.89	0.00 ± 0.00	0.07 ± 0.07	1.30 ± 1.12	25.00 ± 0.00
ACOPF118	Diffusion	13.95 ± 1.32 ×	6.42 ± 0.35	0.01 ± 0.00	1.49 ± 0.52	9.42 ± 1.33	0.00 ± 0.00
	MLP	13.48 ± 1.24 ×	2.83 ± 0.29	0.00 ± 0.00	0.18 ± 0.14	7.44 ± 2.18	0.00 ± 0.00
QP	Diffusion	-10.61 ± 1.04 ×	2.75 ± 8.85	0.03 ± 0.09	0.86 ± 1.41	15.60 ± 7.67	0.00 ± 0.00
	MLP	-7.92 ± 4.70 ×	27.70 ± 43.17	0.35 ± 0.40	6.37 ± 4.03	36.95 ± 15.28	0.00 ± 0.00
QPSR	Diffusion	-9.76 ± 0.42 ×	0.19 ± 0.35	0.00 ± 0.00	0.21 ± 0.14	9.75 ± 1.80	0.00 ± 0.00
	MLP	-5.93 ± 5.65 ×	39.57 ± 58.02	0.34 ± 0.38	6.11 ± 4.02	33.77 ± 13.77	0.00 ± 0.00
CQP	Diffusion	-39.97 ± 12.79 ×	12.52 ± 32.88	0.10 ± 0.23	4.82 ± 8.51	18.63 ± 10.40	0.00 ± 0.00
	MLP	-98.55 ± 33.07 ×	165.21 ± 89.29	1.01 ± 0.37	30.03 ± 10.76	68.54 ± 8.40	0.00 ± 0.00
RETARGETING	Diffusion	1.74 ± 0.50 ✓	0.99 ± 1.57	0.00 ± 0.00	0.00 ± 0.00	0.02 ± 0.11	97.93 ± 0.00
	MLP	1.74 ± 0.50 ×	1.90 ± 3.65	0.00 ± 0.00	0.05 ± 0.09	0.54 ± 0.50	45.52 ± 0.00

highlights that the Diffusion models’ multi-solution sampling property is key to effective performance improvement. The high feasibility achieved by Diffusion on the Retargeting task, characterized by its relatively few and simple constraints, demonstrates that Solution Selection is enough to improve feasibility for a few simple constraints.

6.2 Train procedure

This section illustrates the training dynamics of DiOpt and Diffusion as observed on the QP and QPSR tasks. For training dynamics results across all tasks, please refer to the supplementary materials. As shown in Figure 4, once bootstrapping begins (2000 epochs), DiOpt immediately exhibits a significant reduction in constraint violation metrics.

Meanwhile, the objective-related metrics initially experience a slight increase, followed by a gradual decrease toward near-optimal values. This behavior indicates that, after bootstrapping is activated, DiOpt first guides the sampling distribution into the interior of the feasible region, and subsequently steers it progressively toward the neighborhood of the optimal solution. In this way, DiOpt achieves feasibility improvement while preserving near-optimality.

7 Conclusion and Outlook

This work first explored diffusion models for constrained optimization, revealing that purely supervised methods struggle with feasibility as constraint dimensionality increases. To address this, we proposed DiOpt, a diffusion training framework combining supervised and self-supervised learning. DiOpt guides sampling towards feasibility using a target distribution and improves optimality via weighted self-supervised training. Extensive experiments validate DiOpt’s effectiveness, demonstrating improved feasibility and preserved objective quality compared to baselines. Despite its strengths, DiOpt faces limitations in training efficiency due to its self-supervised nature and the costly feasibility completion needed for equality constraints.

Acknowledgement

This work was supported by National Natural Science Foundation of China (62303319, 62406195), Shanghai Local College Capacity Building Program (23010503100), ShanghaiTech AI4S Initiative SHTAI4S202404, HPC Platform of ShanghaiTech University, and MoE Key Laboratory of Intelligent Perception and Human-Machine Collaboration (ShanghaiTech University), and Shanghai Engineering Research Center of Intelligent Vision and Imaging.

References

- [1] Kyri Baker. Learning warm-start points for ac optimal power flow. In *2019 IEEE 29th International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6. IEEE, 2019.
- [2] R. Bellman, Rand Corporation, and Karreman Mathematics Research Collection. *Dynamic Programming*. Rand Corporation research study. Princeton University Press, 1957.
- [3] Christian Bingane, Miguel F. Anjos, and Sébastien Le Digabel. Tight-and-cheap conic relaxation for the ac optimal power flow problem. *IEEE Transactions on Power Systems*, 33(6):7181–7188, 2018.
- [4] Julia Briden, Breanna J Johnson, Richard Linares, and Abhishek Cauligi. Diffusion policies for generative modeling of spacecraft trajectories. In *AIAA SCITECH 2025 Forum*, page 2775, 2025.
- [5] Nataly Brukhim and Amir Globerson. Predict and constrain: Modeling cardinality in deep structured prediction. In *International Conference on Machine Learning*, pages 659–667. PMLR, 2018.
- [6] Mary B Cain, Richard P O’neill, Anya Castillo, et al. History of optimal power flow and formulations. *Federal Energy Regulatory Commission*, 1:1–36, 2012.
- [7] Xinwei Cao and Shuai Li. Neural networks for portfolio analysis with cardinality constraints. *IEEE Transactions on Neural Networks and Learning Systems*, 35(12):17674–17687, 2024.
- [8] Minas Chatzos, Ferdinando Fioretto, Terrence WK Mak, and Pascal Van Hentenryck. High-fidelity machine learning approximations of large-scale optimal power flow. *arXiv preprint arXiv:2006.16356*, 2020.
- [9] Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *arXiv preprint arXiv:2303.04137*, 2023.
- [10] Jacob K Christopher, Stephen Baek, and Ferdinando Fioretto. Constrained synthesis with projected diffusion models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.
- [11] Thomas M. Cover and Bradley Efron. Geometrical probability and random points on a hypersphere. *The Annals of Mathematical Statistics*, 38(1):213–220, 1967.
- [12] Shutong Ding, Ke Hu, Zhenhao Zhang, Kan Ren, Weinan Zhang, Jingyi Yu, Jingya Wang, and Ye Shi. Diffusion-based reinforcement learning via q-weighted variational policy optimization. *arXiv preprint arXiv:2405.16173*, 2024.
- [13] Shutong Ding, Jingya Wang, Yali Du, and Ye Shi. Reduced policy optimization for continuous control with hard constraints. *Advances in Neural Information Processing Systems*, 36, 2024.

- [14] Wenqian Dong, Zhen Xie, Gokcen Kestor, and Dong Li. Smart-pgsim: Using neural network to accelerate ac-opf power grid simulation. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15. IEEE, 2020.
- [15] Priya L Donti, David Rolnick, and J Zico Kolter. Dc3: A learning method for optimization with hard constraints. *arXiv preprint arXiv:2104.12225*, 2021.
- [16] Hongyang Du, Ruichen Zhang, Yinqiu Liu, Jiacheng Wang, Yijing Lin, Zonghang Li, Dusit Niyato, Jiawen Kang, Zehui Xiong, Shuguang Cui, et al. Enhancing deep reinforcement learning: A tutorial on generative diffusion models in network optimization. *IEEE Communications Surveys & Tutorials*, 2024.
- [17] Matthias Fey, Jan E. Lenssen, Christopher Morris, Jonathan Masci, and Nils M. Kriege. Deep graph matching consensus. In *International Conference on Learning Representations*, 2020.
- [18] Ferdinando Fioretto, Terrence WK Mak, and Pascal Van Hentenryck. Predicting ac optimal power flows: Combining deep learning and lagrangian dual methods. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 630–637, 2020.
- [19] Tairan He, Zhengyi Luo, Wenli Xiao, Chong Zhang, Kris Kitani, Changliu Liu, and Guanya Shi. Learning human-to-humanoid real-time whole-body teleoperation. *arXiv preprint arXiv:2403.04436*, 2024.
- [20] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.
- [21] Bozhen Jiang, Qin Wang, Shengyu Wu, Yidi Wang, and Gang Lu. Advancements and future directions in the application of machine learning to ac optimal power flow: A critical review. *Energies*, 17(6):1381, 2024.
- [22] Nikolaos Karalias and Andreas Loukas. Erdos goes neural: an unsupervised learning framework for combinatorial optimization on graphs. *Advances in Neural Information Processing Systems*, 33:6659–6672, 2020.
- [23] Kota Kondo, Andrea Tagliabue, Xiaoyi Cai, Claudius Tewari, Olivia Garcia, Marcos Espitia-Alvarez, and Jonathan P How. Cgd: Constraint-guided diffusion policies for uav trajectory planning. *arXiv preprint arXiv:2405.01758*, 2024.
- [24] Vince Kurtz and Joel W Burdick. Equality constrained diffusion for direct trajectory optimization. *arXiv preprint arXiv:2410.01939*, 2024.
- [25] Anjian Li, Zihan Ding, Adji Bousso Dieng, and Ryne Beeson. Efficient and guaranteed-safe non-convex trajectory optimization with constrained diffusion model. *arXiv e-prints*, pages arXiv–2403, 2024.
- [26] Anjian Li, Amlan Sinha, and Ryne Beeson. Amortized global search for efficient preliminary trajectory design with deep generative models. *CoRR*, 2023.
- [27] Ke Li and Jitendra Malik. Learning to optimize. *arXiv preprint arXiv:1606.01885*, 2016.
- [28] Xinpeng Li, Enming Liang, and Minghua Chen. Gauge flow matching for efficient constrained generative modeling over general convex set. In *ICLR 2025 Workshop on Deep Generative Model in Machine Learning: Theory, Principle and Efficacy*, 2025.
- [29] Yang Li, Jinpei Guo, Runzhong Wang, and Junchi Yan. From distribution learning in training to gradient search in testing for combinatorial optimization. *Advances in Neural Information Processing Systems*, 36, 2024.
- [30] Yang Li, Jinpei Guo, Runzhong Wang, Hongyuan Zha, and Junchi Yan. Fast t2t: Optimization consistency speeds up diffusion-based training-to-testing solving for combinatorial optimization. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.
- [31] Enming Liang and Minghua Chen. Generative learning for solving non-convex problem with multi-valued input-solution mapping. In *The Twelfth International Conference on Learning Representations*, 2024.
- [32] Defeng Liu, Matteo Fischetti, and Andrea Lodi. Learning to search in local branching. In *Proceedings of the aaai conference on artificial intelligence*, volume 36, pages 3796–3803, 2022.
- [33] Jorge Nocedal and Stephen J Wright. *Numerical optimization*. Springer, 1999.

- [34] Chaoyi Pan, Zeji Yi, Guanya Shi, and Guannan Qu. Model-based diffusion for trajectory optimization. *arXiv preprint arXiv:2407.01573*, 2024.
- [35] Xiang Pan, Tianyu Zhao, Minghua Chen, and Shengyu Zhang. Deepopf: A deep neural network approach for security-constrained dc optimal power flow. *IEEE Transactions on Power Systems*, 36(3):1725–1735, 2020.
- [36] Seonho Park and Pascal Van Hentenryck. Self-supervised primal-dual learning for constrained optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 4052–4060, 2023.
- [37] Antoine Salmona, Valentin De Bortoli, Julie Delon, and Agnes Desolneux. Can push-forward generative models fit multimodal distributions? *Advances in Neural Information Processing Systems*, 35:10766–10779, 2022.
- [38] Sebastian Sanokowski, Wilhelm Berghammer, Martin Ennemoser, Haoyu Peter Wang, Sepp Hochreiter, and Sebastian Lehner. Scalable discrete diffusion samplers: Combinatorial optimization and statistical physics. *arXiv preprint arXiv:2502.08696*, 2025.
- [39] Sebastian Sanokowski, Sepp Hochreiter, and Sebastian Lehner. A diffusion model framework for unsupervised neural combinatorial optimization. *arXiv preprint arXiv:2406.01661*, 2024.
- [40] Ye Shi, Hoang Duong Tuan, Hoang Tuy, and S Su. Global optimization for optimal power flow over transmission networks. *Journal of Global Optimization*, 69:745–760, 2017.
- [41] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv:2010.02502*, October 2020.
- [42] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020.
- [43] Yunhao Tang, Shipra Agrawal, and Yuri Faenza. Reinforcement learning for integer programming: Learning to cut. In *International conference on machine learning*, pages 9367–9376. PMLR, 2020.
- [44] Andreas Wächter and Lorenz T Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106:25–57, 2006.
- [45] Runzhong Wang, Li Shen, Yiting Chen, Xiaokang Yang, and Junchi Yan. Towards one-shot neural combinatorial solvers: Theoretical and empirical notes on the cardinality-constrained case. In *ICLR*, 2023.
- [46] Runzhong Wang, Junchi Yan, and Xiaokang Yang. Learning combinatorial embedding networks for deep graph matching. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 3056–3065, 2019.
- [47] J. G. WENDEL. A problem in geometric probability. *Mathematica Scandinavica*, 11(1):109–111, 1962.
- [48] Ahmed S. Zamzam and Kyri Baker. Learning optimal solutions for extremely fast ac optimal power flow. In *2020 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, pages 1–6, 2020.
- [49] Ling Zhang and Baosen Zhang. Learning to solve the ac optimal power flow via a lagrangian approach. In *2022 North American Power Symposium (NAPS)*, pages 1–6, 2022.
- [50] Yanbo Zhang, Benedikt Hartl, Hananel Hazan, and Michael Levin. Diffusion models are evolutionary algorithms. *arXiv preprint arXiv:2410.02543*, 2024.
- [51] Meng Zhao and Masoud Barati. Synergizing machine learning with acopf: A comprehensive overview, 2024.
- [52] Ray Daniel Zimmerman, Carlos Edmundo Murillo-Sánchez, and Robert John Thomas. Matpower: Steady-state operations, planning, and analysis tools for power systems research and education. *IEEE Transactions on Power Systems*, 26(1):12–19, 2011.

A Proof for Theorem 1

Theorem 1 (Feasibility in Linear Programming.) *Given a d -dimensional linear programming problem of the form:*

$$\min_{x \in \mathbb{R}^d} c^T x \quad \text{subject to} \quad a_i^T x \leq b_i, \quad i = 1, \dots, N,$$

where a_i represents a unit normal vector, drawn independently and uniformly from the unit sphere S^{d-1} . Let x^* be the unique solution to this linear programming problem, and we define a neighborhood ball $B_\epsilon(x^*) = \{x : \|x - x^*\| \leq \epsilon\}$. For sufficiently small $\epsilon > 0$, there is an asymptotic bound on the probability that a point uniformly sampled from $B_\epsilon(x^*)$ lies in the feasible region $\mathcal{C}$.

$$\mathbb{P}_{x \sim B_\epsilon(x^*)}(x \in \mathcal{C}) \approx \frac{1}{2^d}.$$

Proof. Motivated by [11], we can find that, for each linear equation $a_i^T x = 0$, it defines a hyperplane that divides $\mathbb{R}^d$ into two half-spaces, i.e.,

$$A_i := \{x : \text{sgn}(a_i^T x) = 1\}, -A_i := \{x : \text{sgn}(a_i^T x) = -1\}. \quad (16)$$

In that case, we can observe that d linearly independent linear equations will partition the n -dimensional space $\mathbb{R}^d$ into 2^d distinct regions, intersecting at the origin. In other words, let $\delta_i = \text{sgn}(a_i^T x)$ and then we have 2^d choices for $\{\delta_1, \dots, \delta_d\} = \{\pm 1, \dots, \pm 1\}$. Considering the equivalence of each region, a random point sampled uniformly from the unit ball in the n -dimensional space falls into any particular region with an expected probability of $\frac{1}{2^d}$.

Back to the linear programming (LP) problem (16), it follows from the Fundamental Theorem of Linear Programming [33], that if an LP problem has a unique optimal solution $x^* \in \mathbb{R}^d$, the optimal solution must lie at the vertex, which is the intersection of d linearly independent inequality constraints.

By translating the coordinate system with the optimal solution x^* as the origin, we can find that computing the probability $\mathbb{P}_{x \sim B_\epsilon(x^*)}(x \in \mathcal{C})$ is actually equivalent to the problem described above. In that case, we have

$$\mathbb{E}_{x \sim B_\epsilon(x^*)} [\mathbb{P}(x \in \mathcal{C})] = \frac{1}{2^d}.$$

□

B Analysis for Function of Reset Operation

As we mentioned in Section 5, if we continue updating diffusion with the weight of (10), the output of the diffusion solver will resemble that of supervised learning and will be prone to generating infeasible points. For simplicity, we consider the case that the diffusion solver has been well trained and all the points generated by it are feasible in one optimization problem parameterized by $\mathbf{x}$. Then, the weight function will actually be

$$\omega(\mathbf{y}) = \exp(f^*(\mathbf{x}) - f(\mathbf{y}; \mathbf{x})). \quad (17)$$

Let the initial distribution of generated points from the well-trained diffusion solver be $\rho_0(\mathbf{y}; \mathbf{x})$. Then, after N iterations of weighted bootstrapping using (17), the distribution of generated points will be

$$\begin{aligned} \rho_N(\mathbf{y}; \mathbf{x}) &\propto \rho_0(\mathbf{y}; \mathbf{x}) \prod_{i=1}^N \exp(\beta(f^*(\mathbf{x}) - f(\mathbf{y}; \mathbf{x}))) \\ &= \rho_0(\mathbf{y}; \mathbf{x}) \exp(\beta N \cdot (f^*(\mathbf{x}) - f(\mathbf{y}; \mathbf{x}))), \end{aligned} \quad (18)$$

according to the reweighting technique, where $\beta > 0$ is a small value determined by the learning rate. Hence, when $N \rightarrow \infty$, $\rho_N(\mathbf{y}; \mathbf{x})$ will converge to the Dirac distribution $\delta(x - x^*)$. This is equivalent to the supervised diffusion solver trained with optimal points. In contrast, we can avoid this problem by redistributing the probability density across the feasible region with $\rho_1 \propto \rho_0 \exp(\beta(f^*(\mathbf{x}) - f(\mathbf{y}; \mathbf{x}))) \approx \rho_0$ using the reset operation.

C Hyperparameter Settings

In this paper, the following hyperparameters are involved:

Parameter Name	Symbol
Supervised learning ratio	r_s
Number of sampling points	K
Number of training samples	K_t
Number of training epochs	N
Number of diffusion steps	T
Noise Schedule Coefficient	η

The number of sampling points refers to the number of samples generated by the model $\mathcal{D}_\theta$ on the validation and test sets. The number of training samples, K_t , denotes the number of samples used during training. Note that $K_t \neq K$, as a large training sample size would significantly increase training time. The supervised learning ratio r_s determines the portion of training steps that use supervised learning. Specifically, DiOpt is trained with optimal solutions during the first $N \cdot r_s$ steps. After that, the buffer $\mathcal{B}$ is initialized, and training continues in the same manner. By setting $r_s = 1$, the procedure reduces to standard Diffusion as described in the paper. The number of diffusion steps corresponds to the variable T in (6). All experiments in Table 2 use $K = 64$. The parameter T is set to 5 for Retargeting and ACOPF118 tasks, and 100 for all other tasks. Unless stated otherwise, we fix $N = 10000$, $r_s = 0.2$, $T = 5$, $K_t = 16$, $K = 32$ ($N = 5000$ for ACOPF118).

Our noise network ϵ_θ consists of a time encoding module and a backbone network. The time encoding module maps each timestep into a 32-dimensional vector using sinusoidal positional embeddings, followed by a two-layer MLP with 512 hidden units and Mish activation. The backbone network comprises four linear layers, each with 512 hidden units. The input dimension is $d_y + d_t + d_x$, where d_x , d_t , and d_y denote the dimensions of $\mathbf{x}$, t , and $\mathbf{y}$, respectively. Each layer is followed by a Mish activation function. The final output has dimension d_y . In all experiments, we apply equality constraint completion, but do not perform inequality correction for diffusion-based methods.

The experiments were performed on a high-performance workstation featuring an Intel Core i9-14900K processor (24 cores, 32 threads, 6.0 GHz turbo frequency) paired with dual NVIDIA GeForce RTX 4090D GPUs (24GB GDDR6X VRAM each) for accelerated deep learning computations. The system was equipped with 128GB DDR5 RAM. The Ubuntu 24.04 LTS operating system with Linux kernel 6.11 and NVIDIA driver 550.144.03 provided an optimized environment for GPU-accelerated workloads.

D Ablation Study

In this section, we conduct a series of ablation experiments to analyze the impact of various hyperparameters on the experimental results.

D.1 Different Diffusion Steps

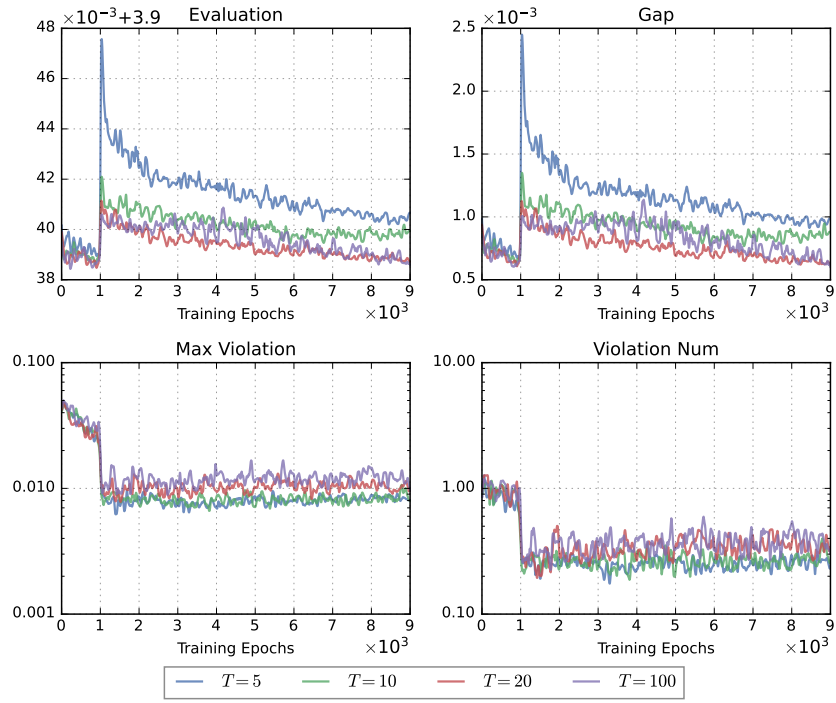
Recent studies [41] have pointed out that the number of diffusion steps can significantly affect the quality of the generated solutions. In addition, according to (6), different values of T introduce different levels of noise into the sampling process.

To investigate the effect of diffusion steps T on performance, we visualize the training dynamics of DiOpt under different diffusion steps in Figure 5. As shown in the figure, for the four tasks considered, the performances of $T = 10$ and $T = 20$ vary across tasks. Only $T = 100$ consistently achieves the smallest optimality gap across all tasks. Furthermore, for all values of T , a significant improvement in feasibility is observed immediately after the bootstrapping phase begins. This observation is consistent with the conclusions drawn in the main text.

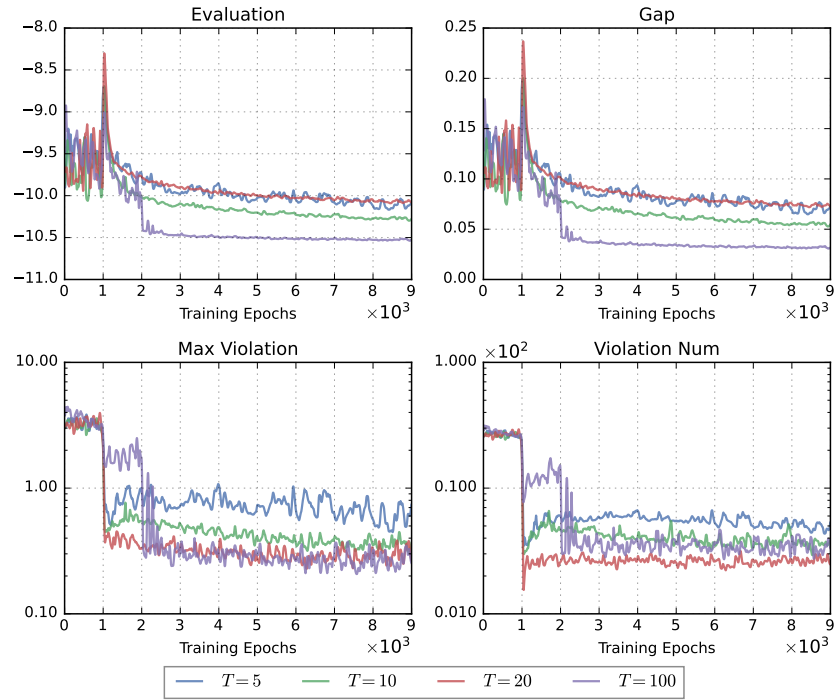
D.2 Different Supervised Ratios

Here r_s critically determines how closely DiOpt approaches the optimal value before bootstrapping initialization. It should be noted that due to the curse of dimensionality [2], the sampling capability

(a) ACOPF57



(b) QP



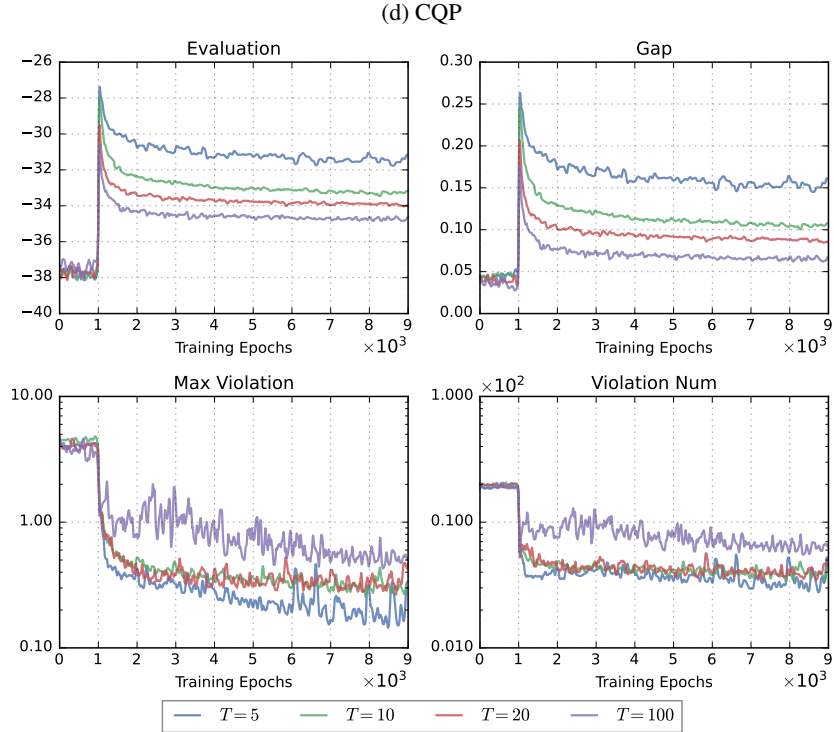
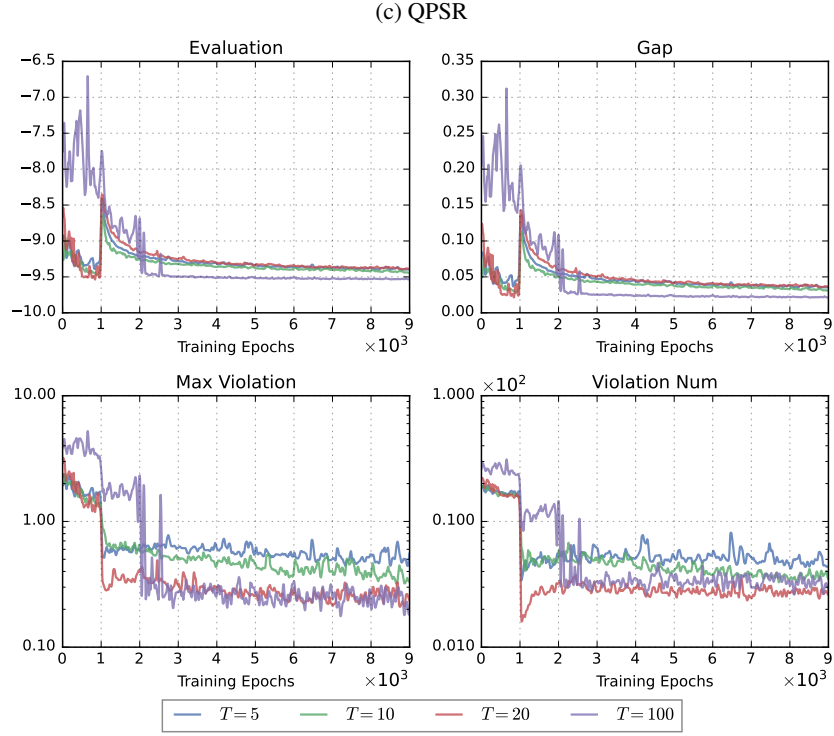


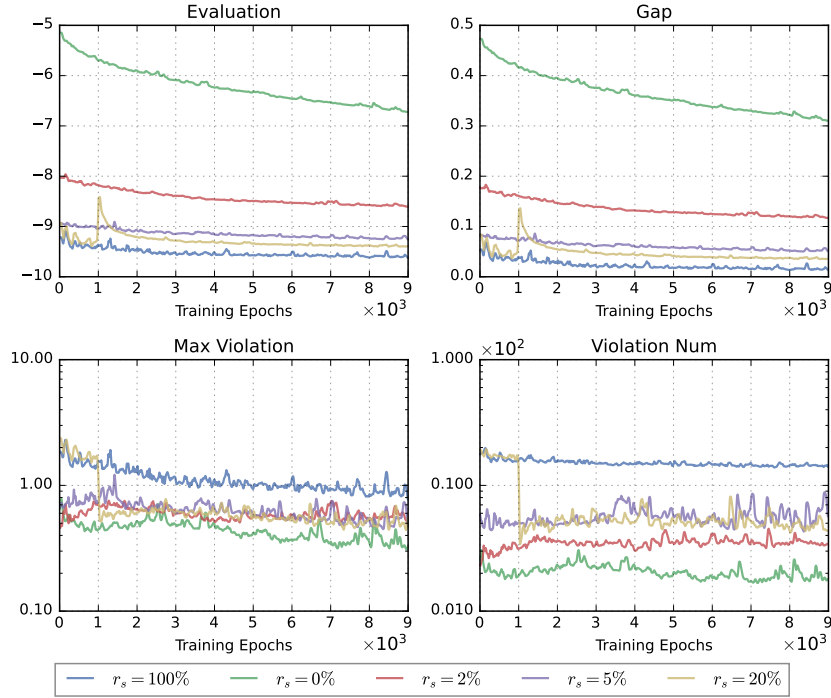
Figure 5: **Effects of Different Diffuion Steps.** In this experiment, we examine how varying the number of diffusion steps T affects performance. All other hyperparameters are fixed as follows: $r_s = 0.2$, $K_t = 16$, and $K = 32$ for all values of T .

of Diffusion-based methods under limited computational resources cannot guarantee sufficient exploration of the solution space. This limitation often leads to suboptimal objective function performance when supervision is inadequate.

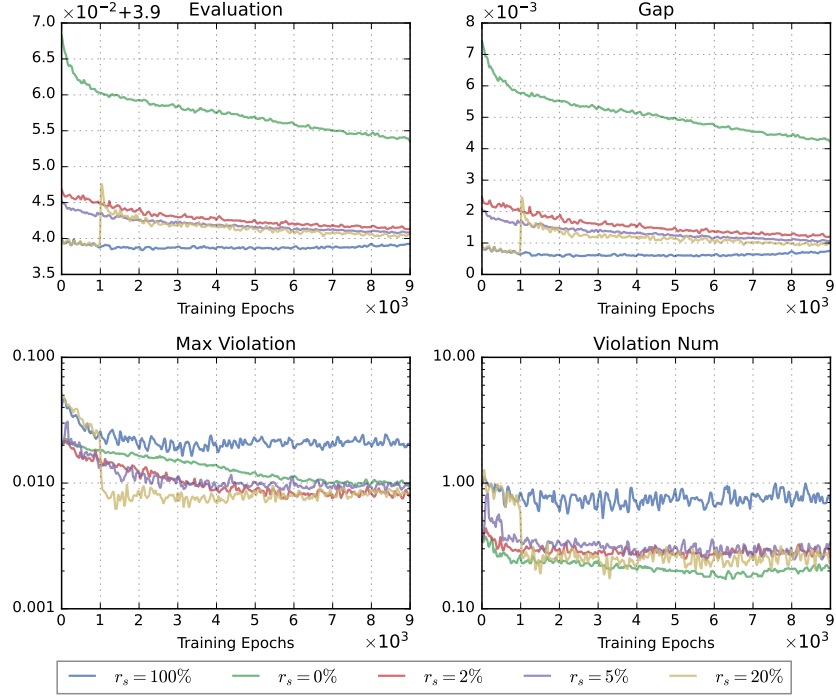
As demonstrated in Figure 6, models with $r_s = 0.05, 0.02$, and 0.2 achieve comparable feasibility performance. However, across both QP and QPSR tasks (with Dimension 100), the $r_s = 0.02$ configuration demonstrates relatively inferior objective function performance compared to $r_s = 0.05, 0.2$, attributable to insufficient supervised learning. In contrast, both $r_s = 0.05, 0.2$ maintain objective function values comparable to Diffusion while achieving satisfactory feasibility.

These results highlight the importance of selecting an appropriate supervised ratio for DiOpt. In this work, we set the default r_s to 0.2 to ensure: (1) model convergence before bootstrapping, and (2) a clear demonstration of bootstrapping’s effectiveness through performance improvements.

(a) QPSR



(b) ACOPF57



(c) QP

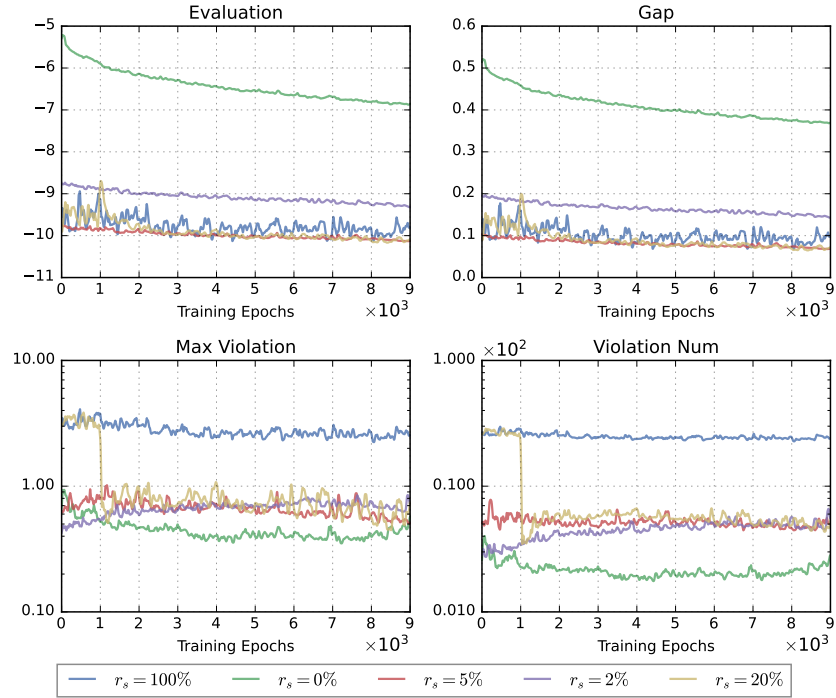
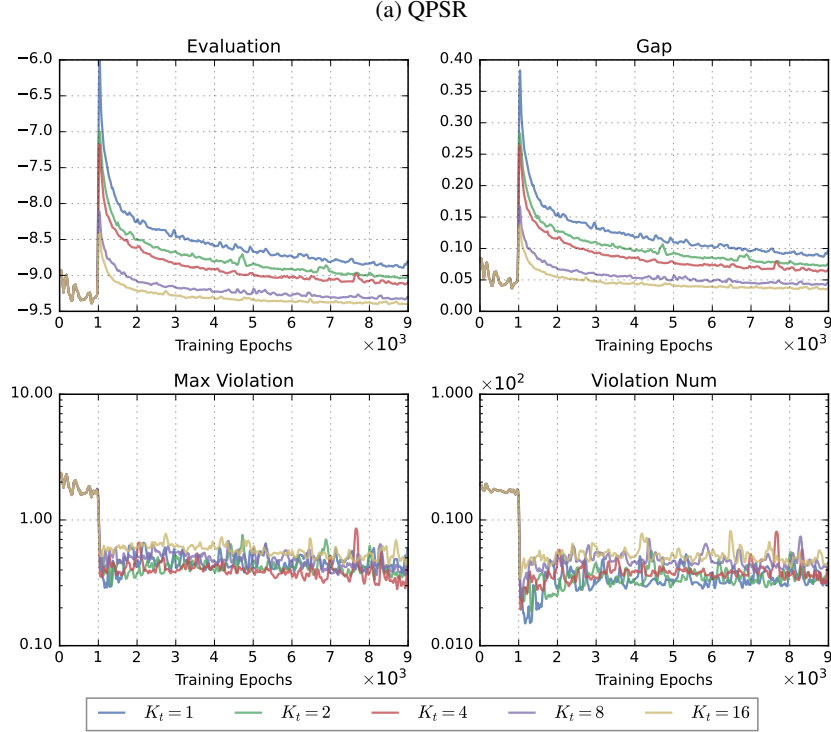


Figure 6: **Effects of Different Supervised Ratio.** In this experiment, we examine how varying the number of diffusion steps T affects performance. All other hyperparameters are fixed as follows: $T = 5$, $K_t = 16$, and $K = 32$ for all values of r_s .

D.3 Different Training Samples

During the training phase of DiOpt, we set $K_t = 1$ for the supervised learning stage (in contrast to $K_t = 16$ used in Diffusion). Once bootstrapping begins, K_t is set to 16. Figure 4 demonstrates that DiOpt maintains performance parity with Diffusion prior to bootstrapping, confirming that no additional K_t adjustment is required during supervised learning.

Figure 7 reveals that increased K_t values yield improved objective function performance at the cost of slight feasibility degradation. This trade-off emerges because larger K_t values concentrate the model’s sampling distribution nearer to optimal points, consequently narrowing the feasible region. However, given the negligible impact on overall feasibility, we establish $K_t = 16$ as DiOpt’s default configuration. The decision against higher K_t values stems from computational overhead, as they would proportionally increase both training duration and GPU memory requirements.



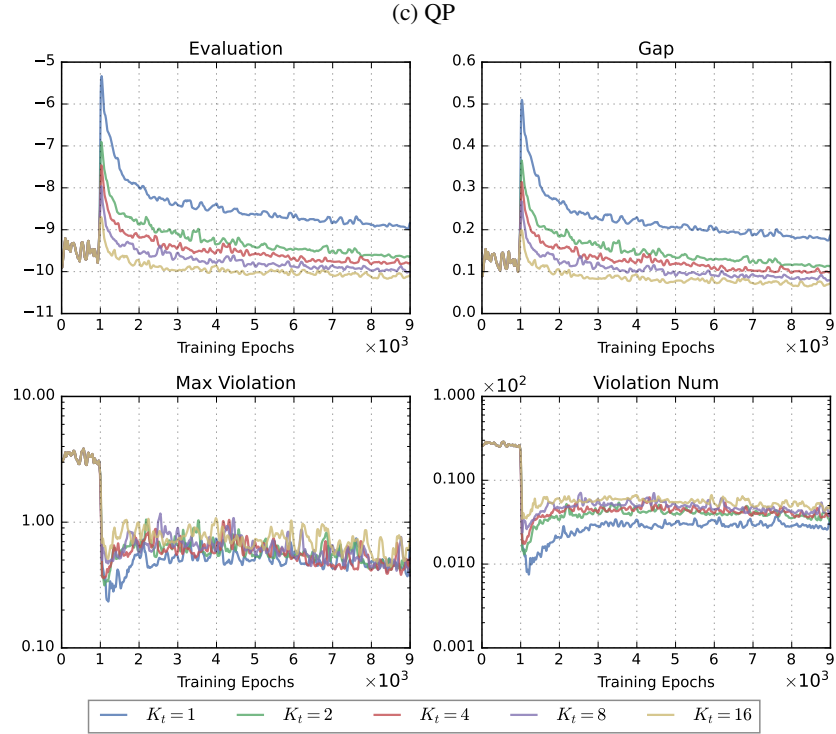
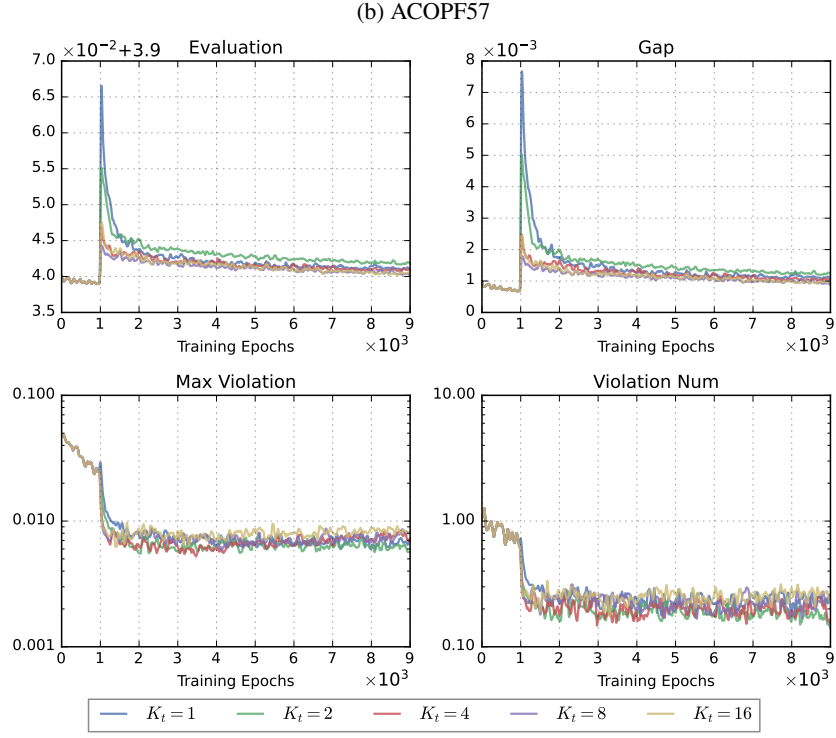


Figure 7: **Effects of different Training Samples.** In this experiment, we examine how varying the number of training samples K_t affects performance. All other hyperparameters are fixed as follows: $r_s = 0.2$, $T = 5$, and $K = 32$ for all values of K_t .

D.4 Different Sampling Points

In this chapter, we examine the impact of sampling points on DiOpt and Diffusion. As shown in Figure 9, increasing K fails to improve the feasibility of Diffusion across all tasks except ACOPF57. Although the dimension of $\mathbf{y}$ in ACOPF57 is 128, the equality completion mechanism reduces the effective prediction dimension to 14. This makes it easier for Diffusion to sample feasible solutions for ACOPF57. Therefore, using $T = 5$ for Diffusion in the main text does not compromise the fairness of result comparisons, as varying T has a negligible impact.

Regarding Figure 8, we observe that increasing K consistently enhances feasibility rates. Except for ACOPF118, both $T = 100$ and $K = 512$ achieve optimal gaps and feasibility rates on all tasks. Consequently, we adopt $T = 100, K = 64$ for DiOpt across all tasks except ACOPF118 and Retargeting in the main text. The $T = 5$ setting for Retargeting is maintained to ensure fair comparison with Diffusion.

D.5 Different Training Procedures

In this subsection, we primarily analyze the training dynamics of DC3, Diffusion, and DiOpt in the main text. It is worth reiterating that the term "Diffusion" here refers specifically to DiOpt with $r_s = 1$. For all experiments in this section, hyperparameters are uniformly set as $K = 32$, $K_t = 16$, $r_s = 0.2$, and $T = 5$. Notably, the noise level η is configured as 0 in all figures presented, a setting that significantly impacts performance outcomes. This aspect will be further elaborated in Section D.6.

From the Figure 10a and 10b analyses, we observe that both DiOpt and Diffusion exhibit superior performance gaps compared to DC3 throughout the training process, while maintaining comparable constraint violation levels. However, the Figure 10d experiments demonstrate that DC3 and Diffusion achieve better performance than DiOpt in retargeting tasks. As illustrated in Figure 2b, this discrepancy arises because DiOpt tends to distribute sampling points near interior regions at a certain distance from the optimal solution, thereby underperforming in tasks with simple constraints. For the CQP problem (Detailed in Section E.3), DC3 exhibits particularly poor performance due to the inherent characteristics of its objective function, where larger $\mathbf{y}$ values correspond to steeper gradients. This gradient amplification phenomenon causes DC3 to deviate progressively from feasible regions during iterative updates, ultimately resulting in severe feasibility violations and substantial optimality gaps. In contrast, both Diffusion and DiOpt demonstrate remarkable resilience to such gradient interference, suggesting that diffusion-based optimization frameworks are inherently more robust against sampling point divergence caused by high-gradient objective functions.

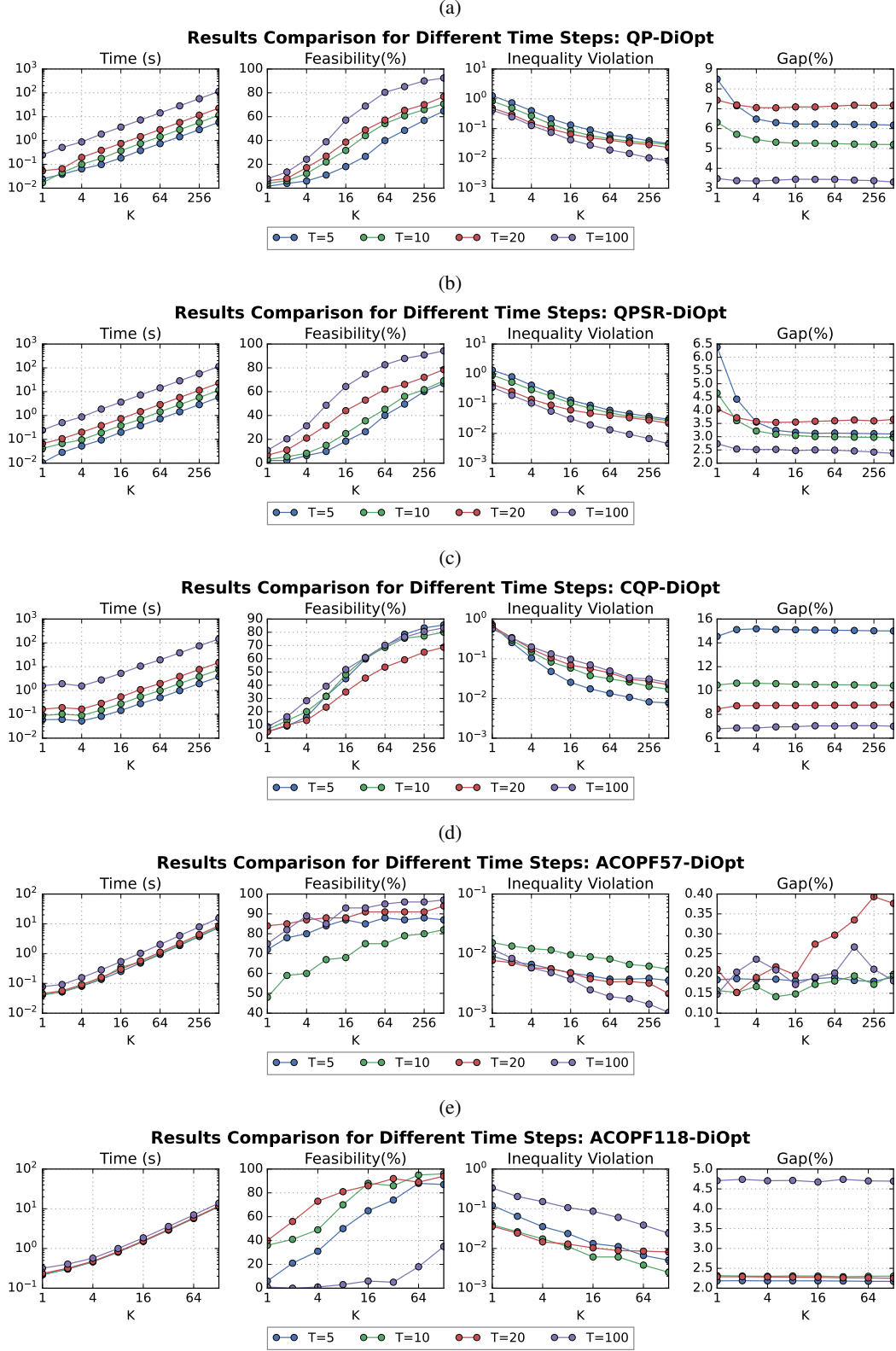


Figure 8: **Effects of number of sampling samples.** In this experiment, we examine how varying the number of sampling points K affects performance. r_s is set as 0.2 for all experiments and $T = 5$ for ACOPF118.

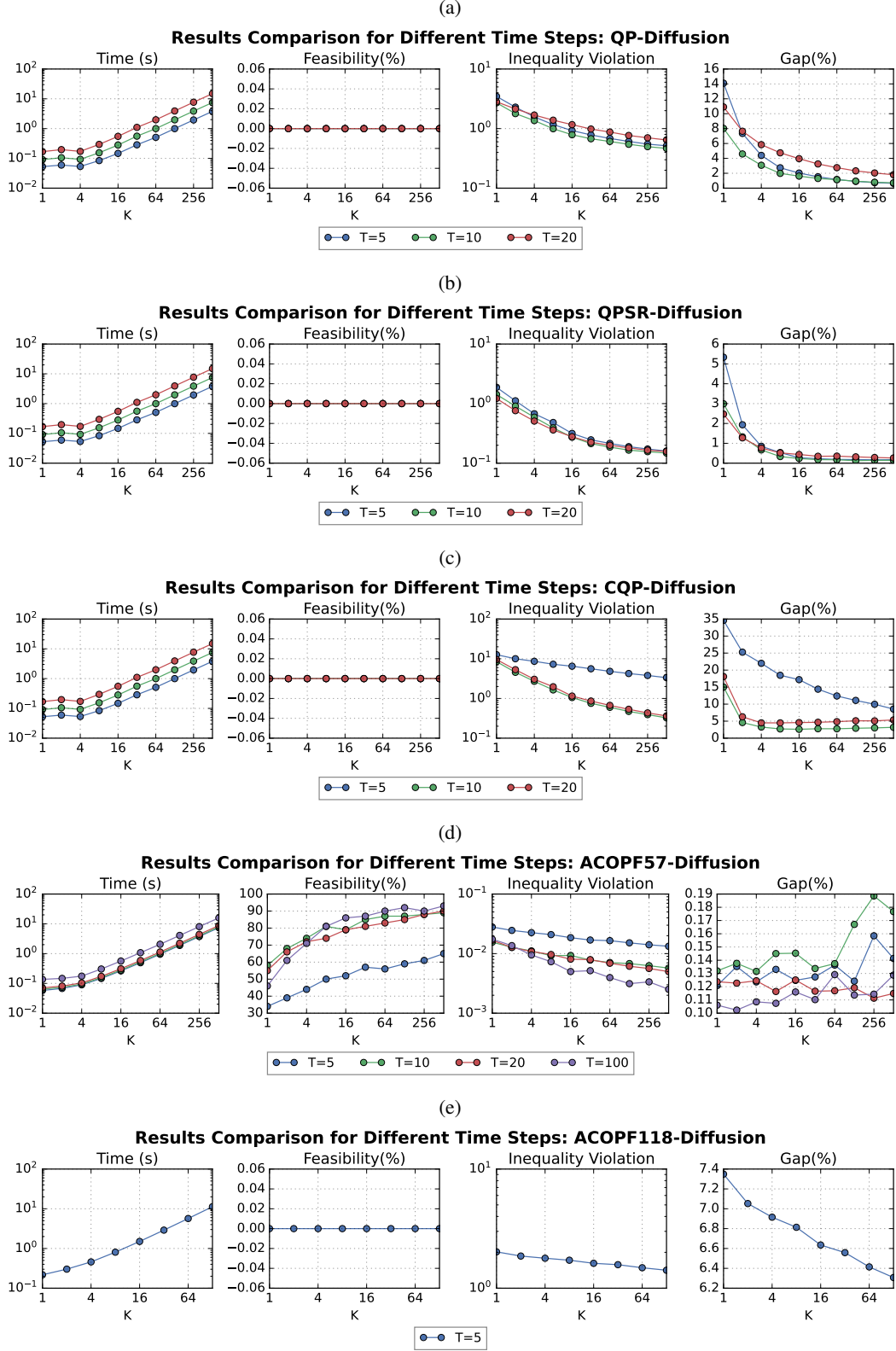
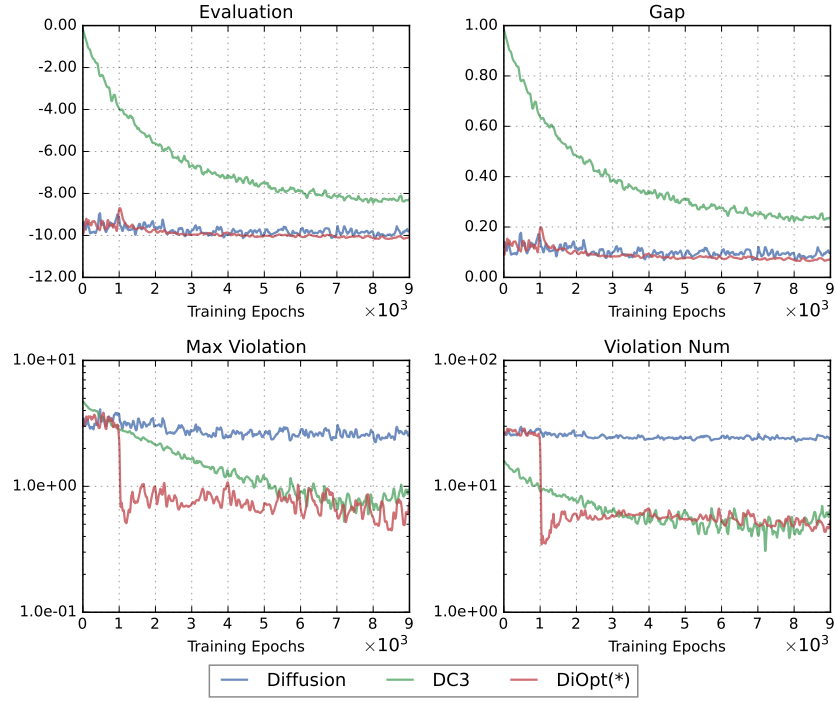
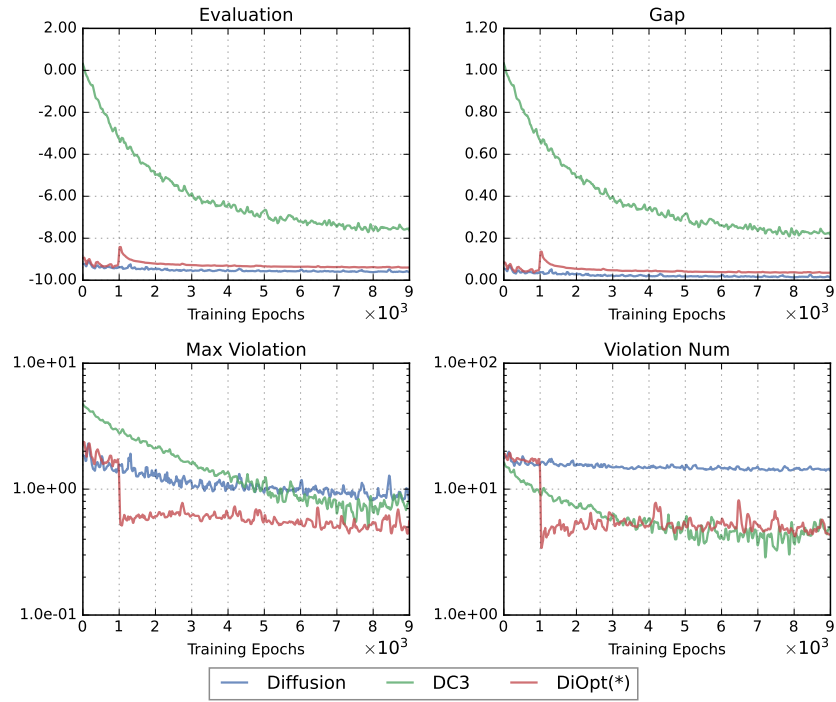


Figure 9: **Effects of number of sampling samples.** In this experiment, we examine how varying the number of sampling points K affects performance. r_s is set as 0.2 for all experiments and $T = 5$ for ACOPF118.

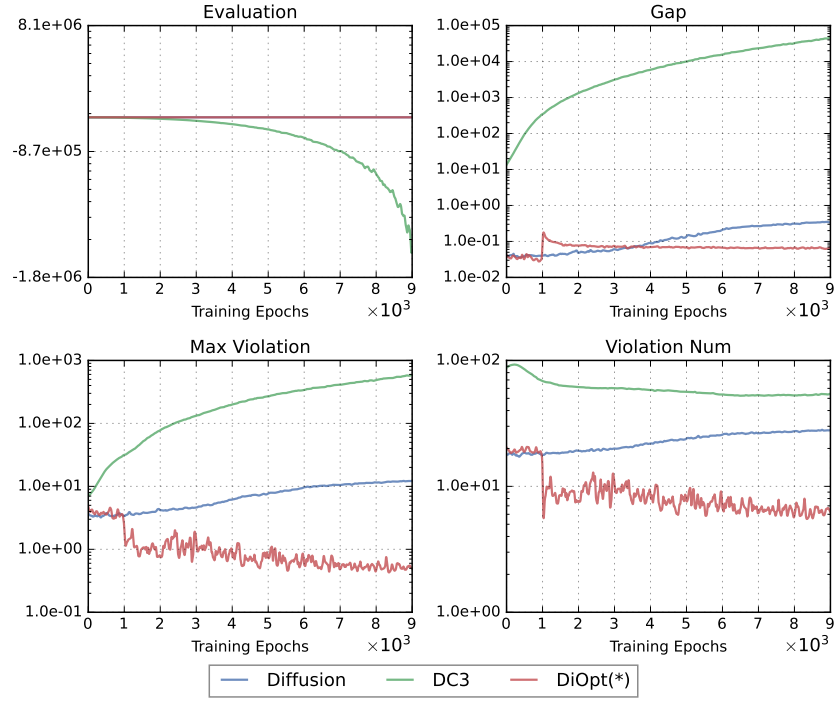
(a) QP



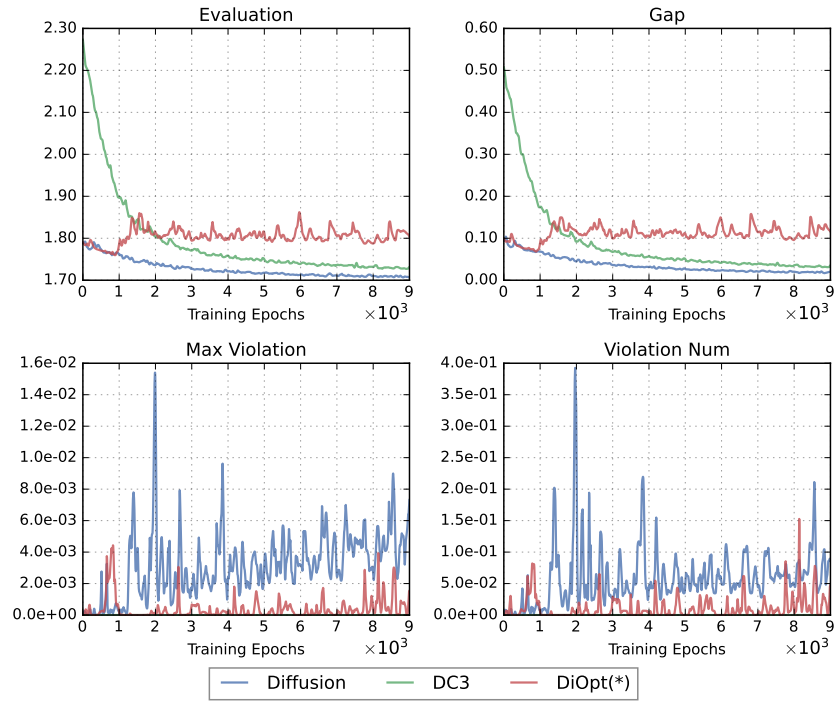
(b) QPSR



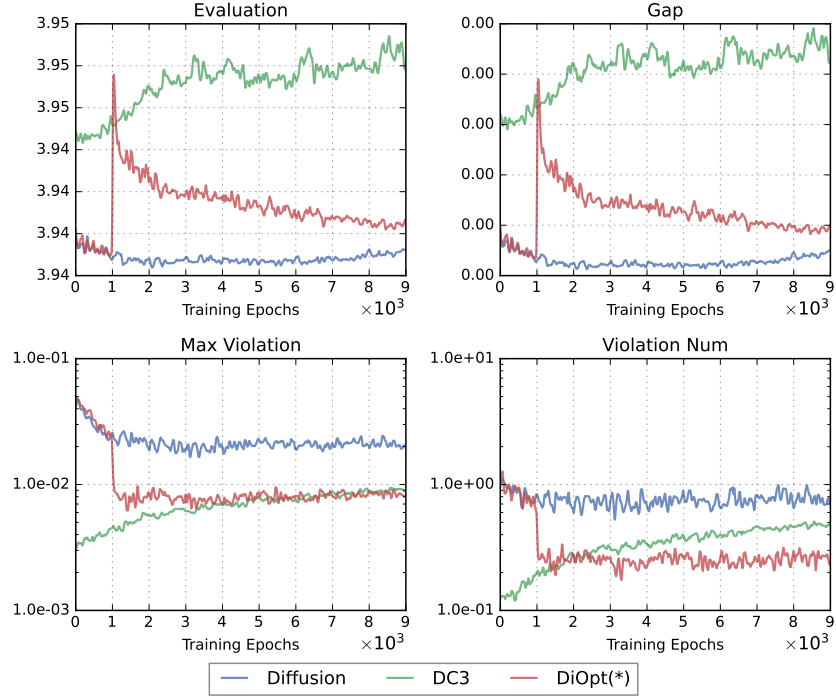
(c) CQP



(d) Retargeting



(e) ACOPF57



(f) ACOPF118

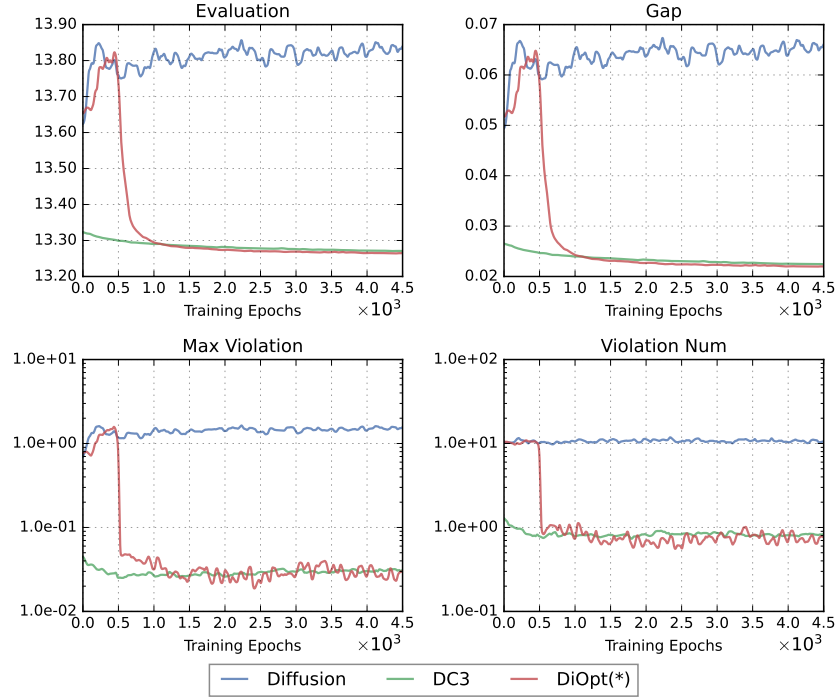


Figure 10: **Training Procedure of DiOpt and Diffusion.** In this experiment, all hyperparameters are fixed as follows: $r_s = 0.2$, $T = 5$, $K_t = 16$, and $K = 32$.

D.6 Different Noise Levels

Finally, we also investigate the influence of noise level η on feasibility rates. As previously discussed, varying η determines the "exploration range" of the diffusion model, which significantly impacts its performance on optimization tasks. As shown in Figure 11 and Figure 12, setting $\eta = 0$ implies that even varying K does not affect the results. This explains why, in Section D.5, DiOpt outperforms DC3 in overall performance but exhibits worse constraint violation curves. Additionally, we observe that the interplay between η , T , and K follows distinct patterns. For smaller T , although larger K is employed, feasibility rates sharply degrade when η exceeds a critical threshold. In contrast, configurations with $T = 100$ demonstrate a higher critical η threshold, and this threshold further increases with larger K . These findings justify why larger T values achieve higher feasibility rates in prior experiments: greater T allows the diffusion process to allocate more noise-injection steps, enabling more thorough exploration of the sampling space while maintaining solution quality.

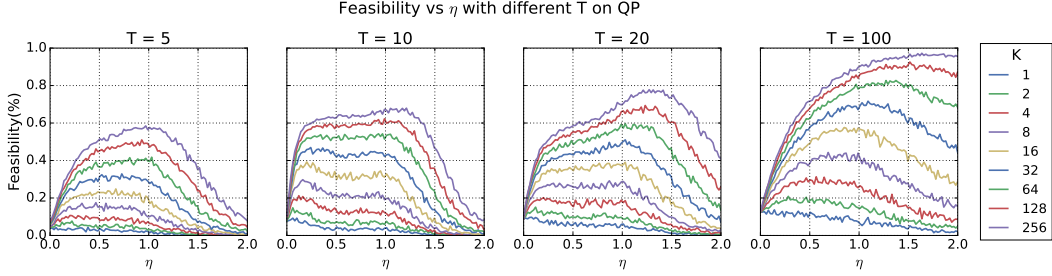


Figure 11: Feasibility of DiOpt under different η with different T and K .

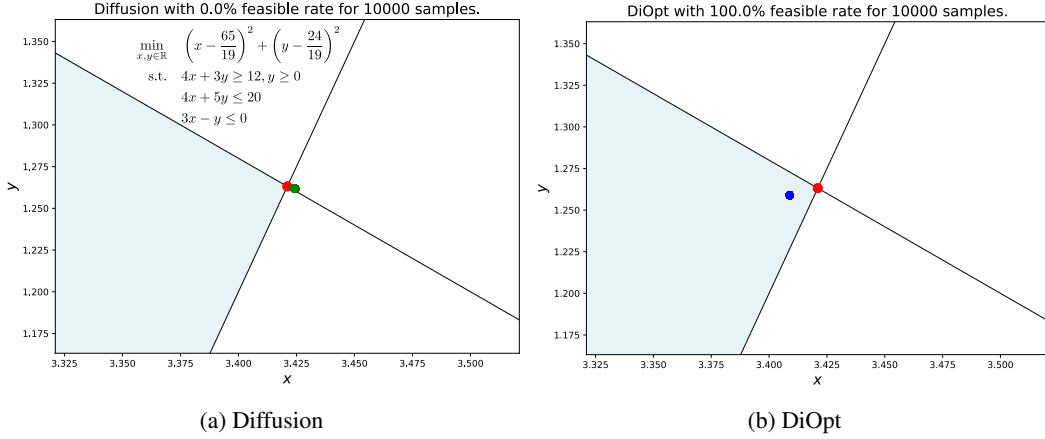


Figure 12: For the $\eta = 0$ configuration in the toy example, observations reveal that despite initial sampling starting from Gaussian-distributed random noise, all trajectories converge to nearly identical positions after denoising when $\eta = 0$. This demonstrates the necessity of noise injection during the sampling process. Directly employing acceleration methods like DDIM [41] for sampling may lead to significant performance degradation, as such approaches bypass critical noise-adding steps required for effective exploration.

E Benchmark configuration

In this section, we provide detailed descriptions of some tasks referenced in the main text. These include concave quadratic optimization problems, relatively complex nonconvex optimization problems with practical significance, and others.

E.1 Quadratic Programming (QP)

The Simple Problem discussed in the text is defined as follows:

$$\begin{aligned} \min_{y \in \mathbb{R}^n} \quad & \frac{1}{2} y^T Q y + p^T y, \\ \text{s.t.} \quad & A y = x, \\ & G y \leq h. \end{aligned} \tag{19}$$

Here, x is treated as a conditional parameter of the optimization problem, sampled uniformly from $[-1, 1]$ across all instances. Q , p , A , and h remain fixed. Q is a diagonal matrix whose diagonal elements are independently and identically sampled from $[0, 1]$. The vector p is generated using the same method as Q . Elements of matrices A and G are sampled from a standard normal distribution. To ensure the feasibility of $G y \leq h$, h is constructed as:

$$h_i = \sum_j |(G A^+)_{ij}|, \tag{20}$$

where A^+ denotes the pseudoinverse of A . The optimal solutions are generated through IPOPT [44]. 10000 examples have been generated for this task.

E.2 Quadratic Programming with Sine Regularization (QPSR)

The Nonconvex Problem in the text is formulated as:

$$\begin{aligned} \min_{y \in \mathbb{R}^n} \quad & \frac{1}{2} y^T Q y + \alpha \cdot p^T \sin(y), \\ \text{s.t.} \quad & A y = x, \\ & G y \leq h. \end{aligned} \tag{21}$$

Here, x similarly serves as a conditional parameter, and the generation methods for x , Q , p , A , G , and h align with those in the Simple Problem. In our experiment, α was set to 1. The optimal solutions are generated through IPOPT. 10000 examples have been generated for this task.

E.3 Concave Quadratic Programming (CQP)

The Concave Quadratic Problem discussed in the text is defined as follows:

$$\begin{aligned} \min_{y \in \mathbb{R}^n} \quad & \frac{1}{2} y^T Q y + p^T y, \\ \text{s.t.} \quad & A y = x, \\ & G y \leq h. \end{aligned} \tag{22}$$

Here, x is treated as a conditional parameter of the optimization problem, sampled uniformly from $[-1, 1]$ across all instances. Q , p , A , and h remain fixed. Q is a diagonal matrix whose diagonal elements are independently and identically sampled from $[-1, 0]$. The vector p is generated using the same method as Q . Elements of matrices A and G are sampled from a standard normal distribution. To ensure the feasibility of $G y \leq h$, h is constructed as:

$$h_i = \sum_j |(G A^+)_{ij}|, \tag{23}$$

where A^+ denotes the pseudoinverse of A . The optimal solutions are generated through IPOPT. 10000 examples have been generated for this task.

E.4 ACOPF

The AC Optimal Power Flow (ACOPF) [6, 40] is a core problem in power systems, aiming to minimize generation costs by adjusting active/reactive power outputs of generators, voltage magnitudes, and phase angles while satisfying constraints such as power balance, line flow limits, and voltage limits. Although the generation costs are merely simple quadratic functions, the intricate constraints render the ACOPF a highly non-convex problem. This results in traditional solution algorithms for ACOPF encountering issues such as global optimality and excessive computation times etc. Recent studies have proposed relaxation approaches [3] and machine learning-based approaches [48, 49, 21, 51] to address these issues.

More specifically, an ACOPF problem involves N nodes, including load buses $\mathcal{L}$, a reference bus $\mathcal{R}$, and generator buses $\mathcal{G}$. Variables include active power p_g , reactive power q_g , active demand p_d , reactive demand q_d , voltage magnitude $|v|$, and voltage phase angle θ . Load buses (representing non-generating nodes) satisfy $(p_g)_{\mathcal{L}} = (q_g)_{\mathcal{L}} = 0$. The reference bus provides a phase angle reference, with $\theta_{\mathcal{R}} = \theta_{\text{ref}}$. Network parameters are described by the admittance matrix Y . The ACOPF is formalized as follows, where $v = |v|e^{i\theta}$, and A, b are fixed parameters related to generation costs:

$$\begin{aligned}
& \min_{p_g, q_g, v, \theta} && p_g^T A p_g + b^T p_g, \\
& \text{s.t.} && \underline{p}_g \leq p_g \leq \overline{p}_g, \\
& && \underline{q}_g \leq q_g \leq \overline{q}_g, \\
& && \underline{|v|} \leq |v| \leq \overline{|v|}, \\
& && \theta_{\mathcal{R}} = \theta_{\text{ref}}, \\
& && (p_g)_{\mathcal{L}} = (q_g)_{\mathcal{L}} = 0, \\
& && (p_g - p_d) + i(q_g - q_d) = \text{diag}(v)Yv^*.
\end{aligned} \tag{24}$$

where A, b represent as the cost coefficient, the underline represent the lower bound, and overline represent the upper bound. In this formulation, nodal demands p_d and q_d act as conditional parameters. Our experiments test the 57-bus system (ACOPF57) and 118-bus system (ACOPF118), with optimal solutions obtained via MATPOWER [52].

E.5 Retargeting Problem

The motion retargeting task can be formulated as an optimization problem, where the objective is to minimize the discrepancy between the motion of the SMPL human model and the H1 robot model. This task involves not only the alignment of joint positions but also the consideration of differences in kinematic structure, body proportions, joint alignment, and end-effector positioning. Due to the significant differences between the kinematic structure of the SMPL model and the kinematic tree of the H1 humanoid robot, [19] proposed a two-step method for preliminary motion retargeting. In the first step, given that the body shape parameters β of the SMPL human model can represent a variety of body proportions, we optimize to find a body shape β' that best matches the humanoid robot's structure, thereby minimizing the joint position discrepancies between the models. This ensures that the joint positions of the SMPL model and H1 robot align as closely as possible, laying the foundation for subsequent retargeting.

Once the optimal β' is determined, the second step involves mapping the joint positions and postures of the H1 robot to their corresponding positions in the SMPL model using forward kinematics. This process takes into account the kinematic constraints of the robot, ensuring the validity of joint positions. Finally, to further refine the joint alignment, we minimize the differences in the positions of 11 key joints, adjusting the joint configuration between the SMPL model and the H1 robot. It is important to note that the retargeting process goes beyond adjusting joint positions-it also involves the alignment of end-effectors (such as ankles, elbows, and wrists). Special attention is given to the precise alignment of these key points to ensure that the human motion is smoothly transferred

to the humanoid robot. Given a sequence of motions expressed in SMPL parameters, which takes as input the joint positions $\mathbf{P}_{SMPL}$, root rotation $\mathbf{R}_{root}$, and transform offset $\mathbf{O}_{offset}$ from the SMPL model, and computes the global joint positions $\mathbf{P}_{H1}$ of the H1 robot model using forward kinematics. The loss function is defined as the difference between the computed H1 joint positions and the corresponding SMPL joint positions. The optimization problem is defined as follows:

$$\begin{aligned} \min_{\mathbf{P}_{H1}} \quad & \|\text{FK}(\mathbf{P}_{H1}, \mathbf{R}_{root}, \mathbf{O}_{offset}) - \mathbf{P}_{SMPL}\|_2^2 + \lambda \|\mathbf{P}_{H1}\|_2^2, \\ \text{s.t.} \quad & \mathbf{P}_{lower} \leq \mathbf{P}_{H1} \leq \mathbf{P}_{upper}, \\ & \|\mathbf{P}_{H1}\|_2^2 \leq 4. \end{aligned} \quad (25)$$

The ℓ_2 norm penalty ensures smoother values and prevents $\mathbf{P}_{H1}$ from becoming excessively large during optimization. Large control inputs could be impractical and could even damage the robot hardware. 1737 examples have been generated for this task via IPOPT.

Table 3: Results for Model-Based Diffusion with/without Completion. Method with (w.) means Method with completion. Method with (w.o) means Method without completion.

Problem	Method	Objective↓	Gap(%)↓	Ineq Max↓	Ineq Num Viol↓	Eq Max↓	Eq Num Viol↓	Feasibility(%)↑
ACOPF57	MBD(w.)	4.77 ± 1.32	23.07 ± 14.17	1.28 ± 0.53	4.21 ± 1.39	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
	MBD(w.o)	4.50 ± 0.02	23.68 ± 24.31	0.00 ± 0.00	0.00 ± 0.00	26.03 ± 0.53	50.59 ± 1.41	0.00 ± 0.00
	DiOpt	3.81 ± 0.64	0.18 ± 0.73	0.02 ± 0.06	0.28 ± 0.51	0.00 ± 0.00	0.00 ± 0.00	79.11 ± 0.00
ACOPF118	MBD(w.)	17.39 ± 0.34	33.71 ± 11.84	3.87 ± 2.29	21.89 ± 0.97	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
	MBD(w.o)	22.48 ± 0.02	72.96 ± 16.50	0.00 ± 0.00	0.00 ± 0.00	17.23 ± 1.94	121.60 ± 1.79	0.00 ± 0.00
	DiOpt	13.39 ± 1.23	2.18 ± 0.10	0.01 ± 0.03	0.19 ± 0.46	0.00 ± 0.00	0.00 ± 0.00	86.60 ± 0.00
QP	MBD(w.)	381.47 ± 396.82	3597.00 ± 3617.54	82.16 ± 47.93	96.43 ± 16.83	0.00 ± 0.00	0.00 ± 0.00	0.05 ± 0.00
	MBD(w.o)	30.47 ± 0.25	379.87 ± 11.84	5.61 ± 0.24	27.33 ± 0.85	10.53 ± 0.65	23.37 ± 1.04	0.00 ± 0.00
	DiOpt	-10.53 ± 0.47	3.45 ± 1.23	0.02 ± 0.06	0.28 ± 0.51	0.00 ± 0.00	0.00 ± 0.00	79.11 ± 0.00
QPSR	MBD(w.)	397.39 ± 405.76	4165.79 ± 4143.71	84.10 ± 48.39	96.97 ± 15.90	0.00 ± 0.00	0.00 ± 0.00	0.02 ± 0.00
	MBD(w.o)	29.54 ± 0.34	402.82 ± 13.40	5.63 ± 0.21	26.64 ± 0.91	10.42 ± 0.66	23.54 ± 0.95	0.00 ± 0.00
	DiOpt	-9.53 ± 0.43	2.48 ± 0.76	0.01 ± 0.04	0.24 ± 0.48	0.00 ± 0.00	0.00 ± 0.00	82.30 ± 0.00
CQP	MBD(w.)	-16741.82 ± 7398.29	44933.21 ± 19921.78	597.79 ± 164.51	71.18 ± 3.77	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
	MBD(w.o)	18.95 ± 0.07	150.98 ± 0.97	2.84 ± 0.22	2.33 ± 0.40	10.41 ± 0.67	23.84 ± 0.94	0.00 ± 0.00
	DiOpt	-34.56 ± 0.64	7.06 ± 1.37	0.05 ± 0.20	0.74 ± 1.74	0.00 ± 0.00	0.00 ± 0.00	70.35 ± 0.00
RETARGETING	MBD(w.)	4.41 ± 1.06	164.58 ± 56.41	2.54 ± 0.58	0.98 ± 0.06	0.00 ± 0.00	0.00 ± 0.00	1.79 ± 0.00
	MBD(w.o)	4.46 ± 1.07	167.56 ± 55.69	2.36 ± 0.26	1.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
	DiOpt	1.77 ± 0.49	3.26 ± 5.74	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	100.00 ± 0.00

F Baseline Settings

F.1 DC3

DC3 (Deep Constraint Completion and Correction) is a neural network-based constrained optimization solver. Unlike direct prediction of solutions via neural networks, DC3 incorporates two key components: an *equality completion* operator $\varphi_{\mathbf{x}}$ and an *inequality correction* operator $\rho_{\mathbf{x}}$. These mechanisms significantly improve the feasibility of the obtained solutions.

Moreover, DC3 adopts a self-supervised learning paradigm. Instead of requiring optimal solutions as supervision, it constructs its loss function as follows:

$$\ell_{\text{soft}}(\hat{\mathbf{y}}) = f(\hat{\mathbf{y}}; \mathbf{x}) + \lambda_g \cdot \|\text{ReLU}(g(\hat{\mathbf{y}}; \mathbf{x}))\|_2^2 + \lambda_h \cdot \|h(\hat{\mathbf{y}}; \mathbf{x})\|_2^2,$$

where $f(\cdot; \mathbf{x})$ denotes the objective function, while $g(\cdot; \mathbf{x})$ and $h(\cdot; \mathbf{x})$ represent inequality and equality constraints, respectively. The ReLU-based term penalizes constraint violations.

The training and sampling procedures of DC3 are outlined in Algorithm 2, adapted from [15]. In our experiments, we fix $\lambda_g = \lambda_h = 5$. In this work, we define N_θ as a three-layer neural network comprising two hidden layers (each with 512 neurons, ReLU activation, batch normalization, and dropout with $p = 0.2$).

F.2 MLP

In our experiments, we include an MLP baseline that shares the **identical network architecture** with DC3. However, the MLP differs in two key aspects: 1.) It only employs equality completion $\phi_{\mathbf{x}}$ and omits inequality correction $\rho_{\mathbf{x}}$. 2.) During training, it directly minimizes the MSE loss between predictions and ground-truth solutions:

$$\ell_{\text{mse}} = \|\tilde{\mathbf{y}} - \mathbf{y}^*\|_2^2.$$

Algorithm 2 Deep Constraint Completion and Correction (DC3)

```
1: Assume: Equality completion procedure  $\varphi_{\mathbf{x}} : \mathbb{R}^{d_{\mathbf{y}} - d_{\text{neq}}} \rightarrow \mathbb{R}^{d_{\text{neq}}}$ 
2: procedure TRAIN( $X$ )
3: Initialize neural network  $N_{\theta} : \mathbb{R}^{d_{\mathbf{x}}} \rightarrow \mathbb{R}^{d_{\mathbf{y}} - d_{\text{neq}}}$ 
4: while not converged do
5:   for  $\mathbf{x} \in X$  do
6:     Compute partial variables:  $\mathbf{z} = N_{\theta}(\mathbf{x})$ 
7:     Complete to full variables:  $\tilde{\mathbf{y}} = [\mathbf{z}^T \quad \varphi_{\mathbf{x}}(\mathbf{z})^T]^T \in \mathbb{R}^{d_{\mathbf{x}}}$ 
8:     Correct to feasible (or approx. feasible) solution:  $\hat{\mathbf{y}} = \rho_{\mathbf{x}}^{(\text{train})}(\tilde{\mathbf{y}})$ 
9:     Compute constraint-regularized loss:  $\ell_{\text{soft}}(\hat{\mathbf{y}})$ 
10:    Update  $\theta$  using  $\nabla_{\theta} \ell_{\text{soft}}(\hat{\mathbf{y}})$ 
11:   end for
12: end while
13: end procedure
14: procedure TEST( $\mathbf{x}, N_{\theta}$ )
15: Compute partial variables:  $\mathbf{z} = N_{\theta}(\mathbf{x})$ 
16: Complete to full variables:  $\tilde{\mathbf{y}} = [\mathbf{z}^T \quad \varphi_{\mathbf{x}}(\mathbf{z})^T]^T$ 
17: Correct to feasible solution:  $\hat{\mathbf{y}} = \rho_{\mathbf{x}}^{(\text{test})}(\tilde{\mathbf{y}})$ 
18: Return  $\hat{\mathbf{y}}$ 
19: end procedure
```

F.3 Model-Based Diffusion

We attempt to adopt the model-based diffusion method proposed in [34] as a baseline for this work. Since practical application scenarios may not strictly satisfy the conditions specified in the original paper, we adapt the method with specific modifications. Concretely, when calculating the probability score for each sample, we compute it as follows:

$$p_i = P(\mathbf{y}_i | \mathbf{x}) := f(\mathbf{y}_i; \mathbf{x}) + \lambda_h \|h(\mathbf{y}_i; \mathbf{x})\|_2 + \lambda_g \|\text{ReLU}(g(\mathbf{y}_i; \mathbf{x}))\|_2. \quad (26)$$

Here, $\mathbf{y}_i$ represents the i -th sample within the complete collection of samples in one diffuse step. The subsequent algorithmic steps remain consistent with Algorithm 1 in [34]. For all experiments, the number of samples is set to 256, the number of diffusion steps is set to 100, and $\lambda_h = \lambda_g = 10$. The specific process of the model-based diffusion with completion is outlined in Algorithm 3, where "Completion" denotes the task-specific completion procedure.

We offer the results of MBD without completion in Table 3. While the completion step eliminates violations of equality constraints, it simultaneously leads to higher violation levels in inequality constraints. Moreover, failing to employ completion would render the equality constraints unsolvable. These observations demonstrate that pure sampling approaches are insufficient for constraint satisfaction and thus neural-based diffusion models are essential for constrained optimization problems.

Algorithm 3 Model-based Diffusion with completion

```
Input:  $\mathbf{z}^{(N)} \sim \mathcal{N}(0, I)$ , Condition Parameter  $\mathbf{x}$ , Number of Diffusion Steps  $N$ , Number of Samples  $d$ .
for  $i = N$  to 1 do
  Sample  $d$  samples  $\mathcal{Z}^{(i)} = [\mathbf{z}_1^i, \dots, \mathbf{z}_d^i] \stackrel{i.i.d}{\sim} \mathcal{N}\left(\frac{\mathbf{z}^{(i)}}{\sqrt{\alpha_{i-1}}}, \left(\frac{1}{\alpha_{i-1}} - 1\right) I\right)$ 
  Get completion:  $\mathcal{Y}^{(i)} = [\mathbf{y}_1^i, \dots, \mathbf{y}_d^i] = \text{Completion}(\mathcal{Z}^{(i)}; \mathbf{x})$ 
  Calculate probability score:  $p_j = P(\mathbf{y}_j^i | \mathbf{x})$ 
  Estimate New Center:  $\mathbf{z}^{(i-1)} = \frac{\sum_{j=1}^d p_j \mathbf{z}_j^i}{\sum_{j=1}^d p_j}$ 
end for
Complete partial solution:  $\mathbf{y}^{(0)} = \text{Completion}(\mathbf{z}^{(0)}; \mathbf{x})$ 
Return Optimized solution  $\mathbf{y}^{(0)}$ 
```
