BENCHMARKING FLOW-BASED MODELS FOR INVERSE DESIGN PROBLEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Inverse design problems are pervasive across science and engineering, including inverse kinematics, materials design, and drug design. Additionally, ill-posed inverse design problems often admit multiple design solutions that produce similar target responses, causing traditional deep learning methods to fail and motivating the development of specialized deep inverse models (DIMs). Recent flow-based models, particularly Flow Matching (FM), have achieved state-of-the-art performance across diverse generative modeling tasks and have recently been applied to specific inverse-design applications, yet their behavior relative to existing DIMs remains poorly characterized. In this work, we systematically benchmark several recent flow-based inverse models, including single-step variants, against competitive DIMs across a diverse collection of publicly available inverse-design benchmarks. We characterize each method by both inference speed and the accuracy of the inverse designs it produces. Our results reveal a pronounced accuracy-speed tradeoff among existing DIMs, with flow-based models frequently achieving competitive performance and occupying portions of the Pareto-optimal frontier. We further find that the relative advantages of different flow-based approaches vary substantially across tasks, highlighting the importance of considering both solution quality and inference cost when selecting inverse models.

1 INTRODUCTION

Inverse design problems are pervasive throughout science and engineering (e.g., inverse kinematics Ren et al. (2020), materials design Ren et al. (2021), and drug design Sanchez-Lengeling & Aspuru-Guzik (2018); Ma et al. (2021)). In this work we study the problem of *data-driven inverse design*, where we assume that there exists some observable forward process given by $y = f(x)$, where f is some unknown function, termed the *forward function* and $x \in \mathcal{X}$ is some vector of real numbers that we can choose (e.g., a design). Then, given some desired output of the forward function, $y^* \in \mathcal{Y} = f(\mathcal{X})$, the goal of inverse design is to find some setting of x, denoted x^* such that $f(x^*) \approx y^*$. More formally, our goal is to solve the following optimization problem:

$$x^* = \arg\min_x \mathcal{L}(f(x), y^*). \tag{1}$$

Here $\mathcal{L} : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ is some loss function that assigns a scalar indicating the relative error between $f(x)$ and y^* (e.g., squared error, Ren et al. (2020); Deng et al. (2021)). The quantity being minimized in Eq. 1 is referred to as the resimulation error of the design x, and $f(x)$ is referred to as the resimulation of x.

In this work we study a special case of inverse design, termed *data-driven inverse modeling*, where we assume no knowledge about f, except that we may evaluate it for specific settings of x, so that we can construct a dataset of the form $D = (x_i, y_i)_{i=1}^N$, with N input-output pairs. Our goal is then to use D to construct some mapping of the form $g_\theta : \mathcal{Y} \to \mathcal{X}$, termed the *inverse model*, with the property that $f(g_\theta(y^*)) \approx y^*$. The subscript θ refers to any parameters of the inverse model that are inferred from the dataset, D. Formally then we define the inverse modeling problem as

$$r = \min_\theta \mathbb{E}_{Y^*}[\mathcal{L}(f(g_\theta(Y^*), Y^*)]. \tag{2}$$

Here we assume that $Y^* \sim P_{Y^*}$ is a distribution over the domain $\mathcal{Y}$, which may vary depending upon the goals of the user. Often, P_{Y^*} is sampled by assuming X is uniformly distributed or Gaussian, and then passing samples of X through f.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Data-driven inverse modeling is challenging because inverse problems can be ill-posed. Machine learning problems can be ill-posed for several reasons (e.g., any violation of the Hadamard conditions Hadamard (1902)), but most research has focused on addressing the so-called non-uniqueness problem Ardizzone et al. (2019); Ren et al. (2021), where there are multiple solutions (i.e., settings of x) that can realize a given resimulation target, y^*. This has led to the development in recent years of numerous *Deep Inverse Models* (DIMs), which are designed to address these challenges Kruse et al. (2019); Liu et al. (2018); Ardizzone et al. (2019); Ho et al. (2020); Ren et al. (2020; 2021); Spell et al. (2023). These methods differ considerably in their formulation and can be taxonomized in different ways, but one widely-utilized class of DIMs in science and engineering are so-called probabilistic DIMs (e.g., Ren et al. (2020; 2021)). These are DIMs that learn to model the conditional probability of x, given some setting of y, so that $g_\theta(y) \approx Pr(x|y^*)$. Probabilistic DIMs are typically constructed by adapting popular generative models for the task of inverse modeling. Notable examples include DIMs based on variational auto-encoders (VAEs) Ma et al. (2019), generative adversarial networks (GANs) So & Rho (2019), Mixture Density Networks (MDNs) Unni et al. (2020), and invertible neural networks (INNs)Ardizzone et al. (2019); Ren et al. (2020). Recent benchmarks have found that, *among generative models*, INNs and cINNs often achieve the best inverse accuracy Ren et al. (2020; 2021); Spell et al. (2023).

In recent years Flow-based models have emerged as a state-of-the-art class of generative models Lipman et al. (2023). Flow models utilize a continuous-time velocity field that transports samples from a simple initial distribution to a target distribution. In this regard they are similar to normalizing flows (the generative model used within INNs), but do not suffer from their strict architectural restrictions that potentially limit their efficacy Papamakarios et al. (2021); Wildberger et al. (2023). Consequently flow Matching has recently been applied to specific inverse-design problems in aerodynamic Zhao et al. (2026), optical Ghapanvari (2026), and materials design Liu et al. (2026). However, to our knowledge, there has been no systematic or comprehensive comparison of flow-based inverse models with existing generative DIMs, making it unclear whether, or to what extent, flow-based DIMs outperform them.

1.1 Contributions of This Work

In this work, for the first time, we perform a rigorous benchmarking of flow-based models for the problem of data-driven inverse modeling. Specifically we adapt the following three flow-based models for use in inverse modeling: the conventional flow matching model (FM)Lipman et al. (2023), Conditional denoising diffusion probabilistic model (cDDPM)Ho et al. (2020), and the improved mean flow matching (iMF)Geng et al. (2026). The cDDPM model can be viewed as a stochastic flow-based model and important competitor of more recent deterministic flow models. The iMF is a very recent single-step flow model, enabling extremely fast inference. To our knowledge we are the first to investigate iMF for inverse modeling, in addition to benchmarking its performance for the task. We compare these three flow-based models against several baselines adopted from prior benchmark studies. All models are compared on six different public benchmark problems, adopted from prior inverse benchmarking publications. Where possible, we also follow the same scoring and optimization procedures adopted by prior work.

In contrast to prior benchmarks, we characterize the performance of inverse models using Pareto plots of inference speed and inverse accuracy. As we show, this characterization reveals a tradeoff among existing DIMs between inference speed and accuracy, which can confound comparisons using only one of these two metrics alone (i.e., a model may offer suprerior accuracy, but only because it has substantially greater latency). We utilize this performance characterization to identify which models comprise the Pareto-optimal frontier (i.e., which models provide the best accuracy, among models with the same or longer latency). All of the inverse models considered in our benchmark can produce $T > 1$ candidate inverse solutions for a given target, y^*, and it is often feasible in practice to evaluate the quality of all T candidate solutions using the forward model (i.e., compute its resimulation error), and then subsequently utilize the best candidate Ren et al. (2020). Therefore, it can also be useful to characterize resimulation error as a function of T. We therefore adopt the resimulation error from prior work (e.g., Ren et al. (2020)) that takes the minimum error among T candidate solutions. We now summarize our contributions:

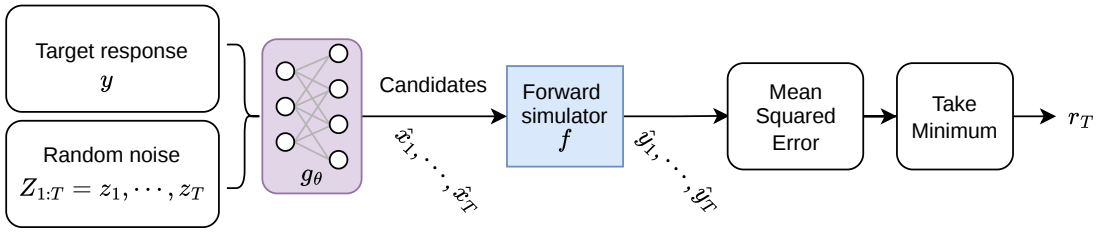

Figure 1: Overview of the framework used to score DIMs using the resimulation error r. The resimulated responses $\hat{y}_1, \cdots, \hat{y}_1$ are computed by passing the T candidates to the forward simulator that we also call the oracle.

- We perform the first rigorous benchmark of flow-based models for data-driven inverse design. This includes six public benchmarks from prior work, and several recent baseline models from prior work.
- We benchmark improved mean flow model (iMF) Geng et al. (2026) for inverse modeling for the first time for inverse modeling. iMF is a very recent single-step flow-based model.
- In contrast to prior benchmarks of inverse models, we utilize Pareto plots to provide a richer and more fair performance assessment. We also allow models to produce T candidate solutions, and evaluate their performance as T varies.

2 Related Work

Deep inverse models and benchmarks. Ren et al. introduced a time-based evaluation of deep inverse models and proposed the neural-adjoint method, which searches for designs by optimizing a learned forward surrogate (Ren et al., 2020). Their subsequent work examined inverse design for artificial electromagnetic materials, comparing methods across tasks with different degrees of ill-posedness and under constraints on repeated simulation (Ren et al., 2021). Spell et al. proposed Mixture Manifold Networks, which combine multiple inverse models coordinated through a common forward model, and evaluated them against other generative approaches to inverse modeling (Spell et al., 2023). The MMN study focuses on comparisons among generative inverse models. We use these works as methodological and benchmark foundations, and compare recent flow-based generators, including multi-step Flow Matching and one-step iMF, against established baselines such as cINN, MDN, and cDDPM across a broader set of design tasks. We report both cumulative best-of-T re-simulation error and the end-to-end time needed to generate the candidates.

Recent flow matching in inverse design. Flow-matching-based methods have been applied to airfoil design (Zhao et al., 2026), optical coating design (Ghapanvari, 2026), and metal-organic framework design within a reinforcement-learning framework (Liu et al., 2026). Some recent studies have also compared flow matching with other inverse-design methods. Krüger et al. (2026) compare four approaches on a gas-turbine combustor problem, emphasizing accuracy, diversity, and training-data size. de Campos et al. (2026) compare three flow and invertible models across five tasks, focusing on inverse accuracy and uncertainty-based candidate selection. We compare six methods across six tasks to examine how proposal budget and candidate-generation time affect the preferred method across problems.

3 Methods

3.1 Benchmark methods

We compare inverse models that map a target response y^* to one or more candidate designs x. We use θ to denote the trainable parameters of each model; training estimates these parameters from the paired dataset $D = \{(x_i, y_i)\}_{i=1}^N$. For stochastic models, candidate generation also depends on a

random input z, which determines which design the model proposes for a given target. Although the models use different training objectives, we evaluate their candidate designs using the same forward re-simulation metric.

Established baselines. We evaluate the conditional invertible neural network (cINN), and a mixture density network (MDN), following the benchmark implementations of Ren et al. (2020; 2021). We refer readers to those papers for the methods' architectures and objectives. While the original works provide additional baselines, we retain these methods as complementary reference points with different approaches to inverse prediction and candidate generation. cINN is competitive when multiple proposals can be evaluated, with strong results on several benchmark task. MDN provides a conditional-density modeling baseline; its training estimates model parameters θ from paired design–response data.

Flow Matching (FM). FM learns a response-conditioned velocity field that transports a Gaussian sample to a design. For a training design x, Gaussian noise ϵ, and $t \in [0, 1]$, we define the straight-line path from noise to data as $z_t = (1 - t)\epsilon + tx$. The network is trained to match its constant velocity:

$$\mathcal{L}_{\mathrm{FM}} = \mathbb{E}_{x,y,\epsilon,t} \left\| v_\theta(z_t, t, y) - (x - \epsilon) \right\|_2^2 . \tag{3}$$

At inference, for target response y^*, we draw $z_0 = \epsilon \sim \mathcal{N}(0, I)$ and solve $\dot{z}_t = v_\theta(z_t, t, y^*)$ forward from $t = 0$ to $t = 1$. As a default, we use the classical fourth-order Runge–Kutta (RK4) method with 50 equal steps. For step size $h = 1/50$ and $t_n = nh$, each step computes

$$\begin{aligned} k_1 &= v_\theta(z_n, t_n, y), \\ k_2 &= v_\theta(z_n + \tfrac{h}{2}k_1, t_n + \tfrac{h}{2}, y^*), \\ k_3 &= v_\theta(z_n + \tfrac{h}{2}k_2, t_n + \tfrac{h}{2}, y^*), \\ k_4 &= v_\theta(z_n + hk_3, t_n + h, y^*), \\ z_{n+1} &= z_n + \tfrac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4). \end{aligned} \tag{4}$$

Thus, the endpoint z_{50} is the generated design; RK4 combines velocity evaluations at the beginning, midpoint, and end of each step to approximate the ODE solution (Lipman et al., 2023).

To examine the effect of the numerical solver, we evaluated each dataset-specific Flow Matching checkpoint using RK4 with 1, 20, and 50 steps. We kept the trained checkpoint fixed across these inference configurations and evaluated each using $T = 1$ proposals.

Improved MeanFlow (iMF). MeanFlow (MF) predicts an average velocity u_θ over an interval $[r, t]$, allowing a sample to move across the interval in one model evaluation (Geng et al., 2025). Under MF's data-to-noise convention, the path is $z_t = (1 - t)x + t\epsilon$, so the average velocity points from data toward noise. This is the reverse of the noise-to-data time convention used above for FM. For a target response y^*, inference starts from $z_1 = \epsilon \sim \mathcal{N}(0, I)$ and uses the backward-time update

$$\hat{x} = z_0 = z_1 - u_\theta(z_1, 0, 1, y^*). \tag{5}$$

Improved MeanFlow (iMF) in Geng et al. (2026) retains this one-step inference rule but changes the training objective. It trains a compound velocity V_θ to match the conditional straight-line velocity $\epsilon - x$. The instantaneous-velocity estimate is defined by

$$v_\theta(z_t, t, y) = u_\theta(z_t, t, t, y).$$

The objective is

$$\begin{aligned} V_\theta &= u_\theta(z_t, r, t, y) + (t - r)\ \mathrm{sg}\big[\mathrm{JVP}\big(u_\theta;\ v_\theta(z_t, t, y), 0, 1\big)\big] , \\ \mathcal{L}_{\mathrm{iMF}} &= \mathbb{E}_{x,y,\epsilon,r,t} \left\| V_\theta - (\epsilon - x) \right\|_2^2 . \end{aligned} \tag{6}$$

Here JVP differentiates u_θ along its input, start-time, and end-time directions, and sg stops gradients through that correction. The compound quantity V_θ is used in training; inference uses u_θ directly. Thus, for target y^*, iMF generates a candidate with $\hat{x} = z_1 - u_\theta(z_1, 0, 1, y^*)$ (Geng et al., 2026).

Conditional denoising diffusion probabilistic model (cDDPM). Lastly, the cDDPM learns to remove Gaussian noise from a design, conditioning on its paired training response y and diffusion timestep. It uses uniformly sampled timesteps and the unweighted noise-prediction loss

$$\mathcal{L}_{\text{cDDPM}} = \mathbb{E}_{(x,y)\sim D,\, t,\, \epsilon} \left\| \epsilon_\theta(x_t, t, y) - \epsilon \right\|_2^2, \tag{7}$$

where x_t is the noised design. Candidate designs are generated by iteratively reversing the diffusion process using 1000 steps.

It is worth noting that our implementation for FM, iMF, and cDDPM conditioning is done by simply concatenation of y^* with the model inputs. Conceptually, all of these models have an MLP backbone with L layers of hidden dimensions h. We also embed the condition y^* to some embedding space of dimension c before concatenating to the model inputs. Selection of these hyperparameters was done through a grid search sweep. This process is explained in more details in Appendix A where we also detail the selected backbone architectures, the training time for each, and the number of parameters.

3.2 Benchmark datasets

We evaluate the inverse models on six benchmark datasets spanning analytic functions, equation-based physical systems, and metamaterial design. The Sine, Ballistics, and Robotic Arm tasks are inherited from Kruse et al. (2019); Ren et al. (2020); ADM, Stack, and Shell originate from Ren et al. (2021); Peurifoy et al. (2018); Deng et al. (2021); Chen et al. (2019). We use the same settings and data generation scripts as the code repository provided by (Ren et al., 2020; 2021).

Table 1: Inverse-design benchmark datasets. "Paired data" denotes the samples available for inverse-model training and validation; following Ren et al. (2020; 2021) the test column denotes a separately generated (for ADM, Stack, Shell) or held-out set from validation (for sine-wave, ballistics, robotic arm).

Dataset	d_x	d_y	Paired data	Test	Forward evaluator	Source
D1:Ballistics	4	1	10,000	500	Projectile simulator	Kruse et al. (2019)
D2:Sine	2	1	10,000	500	Analytic function	Ren et al. (2020)
D3:Robotic Arm	4	2	10,000	500	Analytic forward kinematics	Kruse et al. (2019)
D4:ADM	14	2,000	10,000	500	Neural surrogate ensemble	Deng et al. (2021)
D5:Stack	5	256	50,000	500	Transfer-matrix simulator	Chen et al. (2019)
D6:Shell	8	201	50,000	500	Electromagnetic scattering simulator	Peurifoy et al. (2018)

For every benchmark, inverse-design accuracy is computed using the evaluator listed in Table 1.

3.3 Evaluation metrics

Inverse-design accuracy. An inverse model with parameters θ maps a target response y^* to a design candidate. For stochastic models, the candidate also depends on a random input z, so we write $\hat{x} = g_\theta(y^*, z)$. For target y_i, let $\hat{x}_{i,j} = g_\theta(y_i, z_{i,j})$ denote candidate j, where $z_{i,j}$ is the random input used to generate that candidate. The model-specific expected best-of-T re-simulation error is

$$r_T(\theta) = \mathbb{E}_{Y^*, Z_{1:T}} \left[\min_{1\le j\le T} \mathcal{L}(f(g_\theta(Y^*, Z_j)), Y^*) \right], \tag{8}$$

where Y^* is a test target and $Z_{1:T}$ denotes the T random inputs drawn according to the model's candidate-generation procedure. The expectation is evaluated for the trained parameters θ; it does not minimize over model parameters.

For the reported benchmark results, we use mean squared response error. Its sample estimate is

$$r_T(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \min_{1\le j\le T} \frac{1}{d_y} \left\| f(\hat{x}_{i,j}) - y_i \right\|_2^2, \qquad \hat{x}_{i,j} = g_\theta(y_i, z_{i,j}). \tag{9}$$

Here $N = 500$ is the number of test targets, d_y is the response dimension, and $\hat{x}_{i,j}$ is candidate j for target y_i. We report $r_T(\theta)$ for $T \in \{1, 10, 50, 100, 200\}$. Lower values are better, and the cumulative minimum ensures that the measured error cannot increase with T. Because response scales differ across datasets, we compare r_T within each dataset, rather than across datasets.

Inference time. We measure the wall-clock time t required to generate $T = 1$ candidate designs. Timing excludes forward re-simulation, file I/O, plotting, and result aggregation. Each method is measured over three repetitions following one warm-up run, with CUDA synchronization before and after each timed call. All methods are evaluated on the same hardware.

Accuracy–speed tradeoff. We summarize joint performance using the coordinates

$$\left(\frac{1}{r_1}, \frac{1}{t}\right), \tag{10}$$

where higher values indicate greater accuracy and faster inference, respectively. A method is Pareto optimal if no other method is both more accurate and faster, with a strict improvement in at least one criterion.

3.4 Training and evaluation protocol

In order to ensure fairness and "interpretabilit", our experiment protocols follow the same ones as Ren et al. (2020; 2021). We build on top of the original work's repository [1] to ensure that the baselines are the same and experiments follow the same constraints and logic. For each dataset mentioned in Table1, we use the same implemented data loading and splitting functions as the provided repository – although random seeds can be different. Each dataset is split into a training and validation set. The training set is used to optimize the inverse model's parameters, while the validation set is used to select the best configuration of FM, iMF, cDDPM.

Indeed, for the established baselines, we use the configurations already provided by the repository and that are reported in the papers with no further grid search Ren et al. (2020; 2021). On the other hand, we use the same method used by these papers to select the best configuration of MLP structure. To do that we use the validation set to compute r_1 for all configurations determined by a hyperparameter grid. An optimal configuration that is then retained is the one that produces the lowest validation r_1. The complete sweep spaces and selected configurations are reported in Appendix A.

Following the original benchmark studies, all models are trained using the *Adam* optimizer with a mini-batch training procedure and a validation-based learning-rate scheduler using NVIDIA RTX PRO 6000 Blackwell Max-Q GPUs. The tuned models are trained for 500 epochs, with validation performed every 20 epochs. This is slightly longer than the 300-epoch schedule reported in the original benchmark paper, but the same training schedule is applied consistently within the present comparison and was sufficient for the monitored training and validation losses to converge.

4 Experiments and Results

Accuracy versus the Number of Inverse Design Candidates Proposed, T Table 2 shows that the most accurate method depends on both the benchmark and proposal budget. At $T = 1$, cINN has the lowest error on Ballistics and Sine Wave, while Flow Matching has the lowest error on Robotic Arm, ADM, Stack, and Shell. At $T = 200$, cINN has the lowest error on Ballistics, Sine Wave, and Robotic Arm, while Flow Matching has the lowest error on ADM, Stack, and Shell. No method has the lowest error on all six benchmarks at either budget.

A more detailed view of the ranking across values of T ranging from $T = 1$ to $T = 200$ is provided in Appendix B in Figure 3.

Accuracy and inference time. Figure 2 compares accuracy and candidate-generation time at $T = 1$. iMF is the fastest method and lies on the Pareto frontier on all six benchmarks. cINN and MDN

[1] https://github.com/BensonRen/BDIMNNA

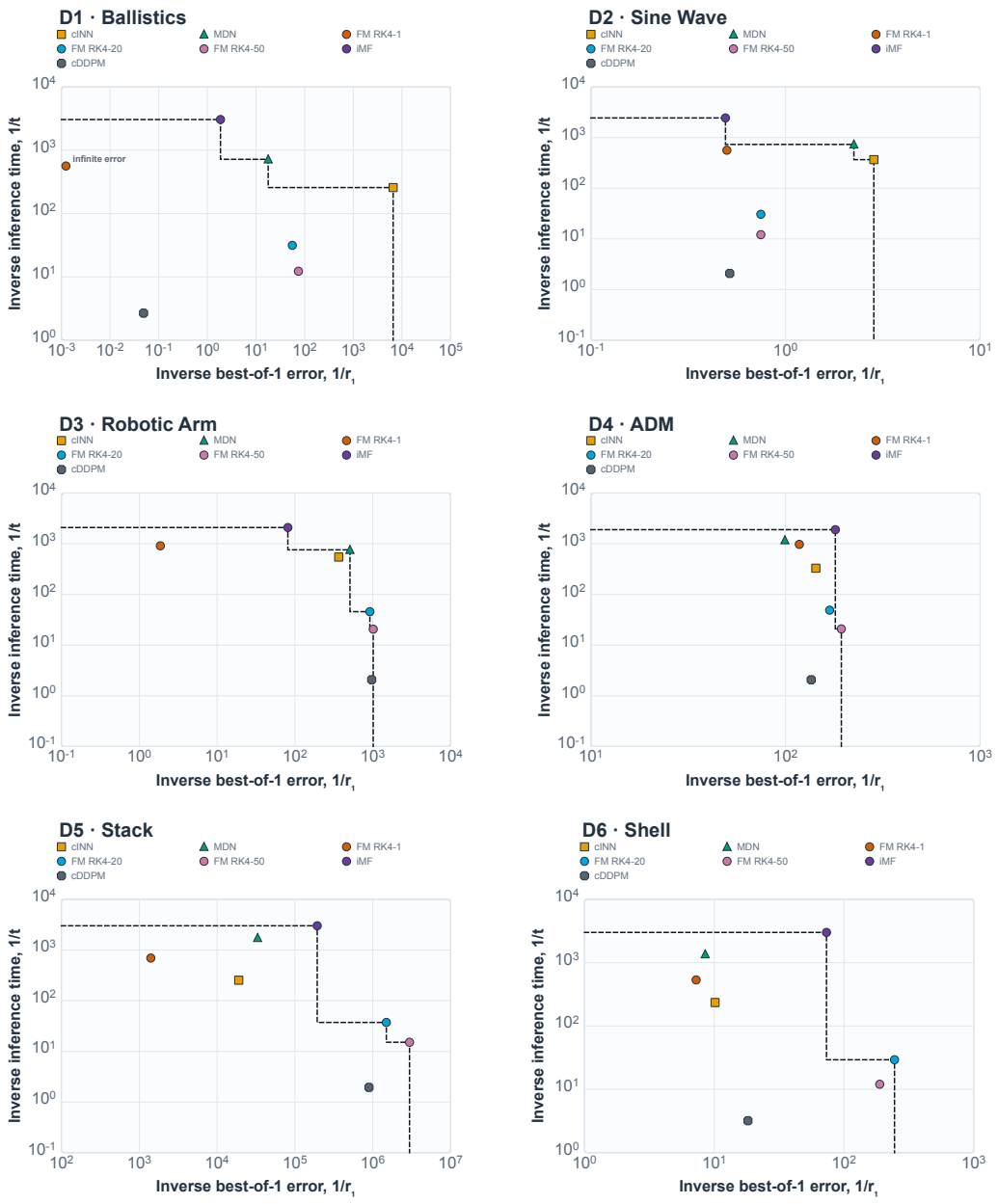

Figure 2: Pareto frontiers at $T = 1$ for the six benchmarks: D1 Ballistics, D2 Sine Wave, D3 Robotic Arm, D4 ADM, D5 Stack, and D6 Shell. Dashed staircases extend each frontier to the plot boundaries.

also lie on the frontier for Ballistics and Sine Wave. On Robotic Arm, the frontier includes MDN and Flow Matching with RK4-20 and RK4-50; on ADM, it includes RK4-50; on Stack, RK4-20 and RK4-50; and on Shell, RK4-20. Furthermore, we find that on ADM dataset, iMF has competitive accuracy compared to FM despite having a one candidate budget and predicting the design in one step (5.55×10^{-3} vs 5.17×10^{-3} respectively for iMF and FM).

The solver comparison shows a consistent cost-accuracy tradeoff between one and 50 RK4 steps: RK4-1 has higher error than RK4-50 on all six benchmarks, with infinite error on Ballistics. RK4-50 also has lower error than RK4-20 on five of the six benchmarks; Shell is the exception, where RK4-20 has lower error. The difference between RK4-20 and RK4-50 on Sine Wave is small.

Table 2: Inverse-design accuracy at $T \in \{1, 200\}$ proposals, measured as best-of-T re-simulation error r_T (lower is better). **Bold** marks the best method for each dataset and proposal budget; underlining marks the second best. Response scales differ across datasets, so values are comparable within, not across, rows.

Dataset	T	cINN	MDN	Flow Matching	iMF	cDDPM
D1 Ballistics	1	$\mathbf{1.52 \times 10^{-4}}$	5.62×10^{-2}	1.35×10^{-2}	5.37×10^{-1}	2.05×10^{1}
	200	$\mathbf{8.05 \times 10^{-9}}$	4.09×10^{-6}	6.22×10^{-7}	6.68×10^{-5}	2.00×10^{1}
D2 Sine Wave	1	$\mathbf{3.52 \times 10^{-1}}$	4.45×10^{-1}	1.34×10^{0}	2.04×10^{0}	1.93×10^{0}
	200	$\mathbf{5.43 \times 10^{-6}}$	2.83×10^{-5}	1.04×10^{-4}	2.56×10^{-4}	1.68×10^{0}
D3 Robotic Arm	1	2.74×10^{-3}	1.96×10^{-3}	$\mathbf{9.86 \times 10^{-4}}$	1.23×10^{-2}	1.03×10^{-3}
	200	$\mathbf{2.74 \times 10^{-7}}$	6.75×10^{-6}	1.82×10^{-6}	6.25×10^{-5}	2.71×10^{-6}
D4 ADM	1	6.99×10^{-3}	1.01×10^{-2}	$\mathbf{5.17 \times 10^{-3}}$	5.55×10^{-3}	7.38×10^{-3}
	200	1.63×10^{-3}	2.21×10^{-3}	$\mathbf{1.14 \times 10^{-3}}$	1.31×10^{-3}	1.73×10^{-3}
D5 Stack	1	5.27×10^{-5}	3.02×10^{-5}	$\mathbf{3.37 \times 10^{-7}}$	5.18×10^{-6}	1.12×10^{-6}
	200	6.40×10^{-7}	3.35×10^{-7}	$\mathbf{1.27 \times 10^{-8}}$	1.18×10^{-7}	3.84×10^{-8}
D6 Shell	1	9.85×10^{-2}	1.17×10^{-1}	$\mathbf{5.30 \times 10^{-3}}$	1.36×10^{-2}	5.50×10^{-2}
	200	9.53×10^{-3}	1.04×10^{-2}	$\mathbf{4.50 \times 10^{-4}}$	1.17×10^{-3}	2.17×10^{-3}

Table 3: Accuracy–speed tradeoff at T=1. Each cell reports single-candidate re-simulation error r_1 (top, lower is better) and wall-clock inference time t in seconds (bottom, lower is faster). Flow Matching uses the RK4-50 integrator. **Bold** marks the lowest error in the top row and the fastest inference time in the bottom row; underlining marks the second-lowest error. Times are averaged over three runs after one warm-up on identical hardware.

Dataset	cINN	MDN	Flow Matching (RK4-50)	iMF	cDDPM
D1 Ballistics	$\mathbf{1.52 \times 10^{-4}}$	5.62×10^{-2}	1.35×10^{-2}	5.37×10^{-1}	2.05×10^{1}
	3.89×10^{-3}	1.39×10^{-3}	8.15×10^{-2}	$\mathbf{3.28 \times 10^{-4}}$	3.74×10^{-1}
D2 Sine Wave	$\mathbf{3.52 \times 10^{-1}}$	4.45×10^{-1}	1.34×10^{0}	2.04×10^{0}	1.93×10^{0}
	2.73×10^{-3}	1.37×10^{-3}	8.26×10^{-2}	$\mathbf{4.10 \times 10^{-4}}$	4.80×10^{-1}
D3 Robotic Arm	2.74×10^{-3}	1.96×10^{-3}	$\mathbf{9.86 \times 10^{-4}}$	1.23×10^{-2}	1.03×10^{-3}
	1.83×10^{-3}	1.32×10^{-3}	4.85×10^{-2}	$\mathbf{4.78 \times 10^{-4}}$	4.82×10^{-1}
D4 ADM	6.99×10^{-3}	1.01×10^{-2}	$\mathbf{5.17 \times 10^{-3}}$	5.55×10^{-3}	7.38×10^{-3}
	3.04×10^{-3}	8.48×10^{-4}	4.79×10^{-2}	$\mathbf{5.27 \times 10^{-4}}$	4.83×10^{-1}
D5 Stack	5.27×10^{-5}	3.02×10^{-5}	$\mathbf{3.37 \times 10^{-7}}$	5.18×10^{-6}	1.12×10^{-6}
	3.95×10^{-3}	5.77×10^{-4}	6.62×10^{-2}	$\mathbf{3.33 \times 10^{-4}}$	5.15×10^{-1}
D6 Shell	9.85×10^{-2}	1.17×10^{-1}	$\mathbf{5.30 \times 10^{-3}}$	1.36×10^{-2}	5.50×10^{-2}
	4.26×10^{-3}	7.36×10^{-4}	8.32×10^{-2}	$\mathbf{3.33 \times 10^{-4}}$	3.13×10^{-1}

These results show that additional solver steps can improve accuracy, while increasing candidate-generation time.

5 Discussion

The results detailed above show that model choice depends on the task and the available inference budget. At $T = 1$, iMF provides the fastest candidate generation on every benchmark, while cINN and MDN remain on the accuracy-speed frontier for Ballistics and Sine Wave. Flow Matching with RK4-20 or RK4-50 provides additional frontier points on Robotic Arm, ADM, Stack, and Shell.

The integrator results are consistent with the role of the learned velocity field. Flow Matching predicts an instantaneous velocity at a given state and time; therefore, it does not directly predict the final design when starting from noise. A single RK4 step spans the full integration interval, so its large step may not resolve changes in the velocity field along the trajectory – the latter can be curved despite the fact that we use conditional straight paths in FM training. Using more, smaller steps lets

RK4 reevaluate the field along the path and can improve the resulting design. Nonetheless, classical RK4 uses four velocity-field evaluations per step, so increasing the step count also increases model evaluations and inference time. The results show this tradeoff without a strict improvement at every step count: RK4-50 has lower error than RK4-20 on five benchmarks, while RK4-20 performs better on Shell.

It is worth reminding that these comparisons use the same trained Flow Matching checkpoint for each solver setting, so they measure inference-time integration choices rather than solver-specific retraining. The Pareto plots report candidate-generation time and do not include the cost of forward re-simulation. Future work could investigate solver-aware training, adaptive integration, and end-to-end costs that include evaluating generated candidates.

6 Conclusion

We benchmarked flow-based inverse-design methods and established baselines across six tasks, evaluating both proposal-budget accuracy and candidate-generation time. The most accurate method varies with the task and proposal budget: cINN has the lowest error on Ballistics and Sine Wave at $T = 1$, while Flow Matching has the lowest error on the other four benchmarks. At $T = 200$, cINN leads on Ballistics, Sine Wave, and Robotic Arm, and Flow Matching leads on ADM, Stack, and Shell. Meanwhile, iMF provides the fastest candidate generation across the six tasks.

The integrator comparison shows that RK4-50 improves on RK4-1 in accuracy across all six benchmarks, at the cost of more velocity-field evaluations and longer inference time. The comparison between 20 and 50 steps is task dependent, with RK4-20 performing better on Shell. Future work could explore solver-aware training and adaptive step selection to reduce inference cost while retaining accuracy, as well as broader benchmarks and end-to-end evaluations that include forward re-simulation.

AI use statement

We used generative AI tools to improve the language of the manuscript, assist with code debugging, and write routine or repetitive scripts. We did not use generative AI to write the manuscript in full, generate images, develop research ideas, or develop the methodology. The authors reviewed and verified AI-assisted edits and code and take responsibility for the final content of this work.

Reproducibility statement

The experimental setup and evaluation protocol are described in the main text. Appendix A documents the architectures, training settings, and hyperparameter search for FM, iMF, and cDDPM. We intend to release the modified code, datasets, and trained model weights after publication once polished.

References

Lynton Ardizzone, Jakob Kruse, Carsten Rother, and Ullrich Köthe. Analyzing inverse problems with invertible neural networks. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rJed6j0cKX.

Yingshi Chen, Jinfeng Zhu, Yinong Xie, Naixing Feng, and Qing Huo Liu. Smart inverse design of graphene-based photonic metamaterials by an adaptive artificial neural network. *Nanoscale*, 11 (19):9749–9755, 2019. doi: 10.1039/C9NR01315F.

Miguel de Campos, Werner Krebs, and Hanno Gottschalk. Generative inverse design with abstention via diagonal flow matching, 2026. URL https://arxiv.org/abs/2603.15925. ICML 2026 camera-ready version.

Yang Deng, Simiao Ren, Kebin Fan, Jordan M. Malof, and Willie J. Padilla. Neural-adjoint method for the inverse design of all-dielectric metasurfaces. *Optics Express*, 29(5):7526–7534, 2021. doi: 10.1364/OE.419138.

Zhengyang Geng, Mingyang Deng, Xingjian Bai, J. Zico Kolter, and Kaiming He. Mean flows for one-step generative modeling. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-2534.

Zhengyang Geng, Yiyang Lu, Zongze Wu, Eli Shechtman, J. Zico Kolter, and Kaiming He. Improved mean flows: On the challenges of fastforward generative models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 30467–30476, 2026.

Erfan Ghapanvari. Yield-aware generative inverse design of anti-reflection coatings via optimal-transport flow matching with a conditional-value-at-risk objective. *Scientific Reports*, 2026. doi: 10.1038/s41598-026-65513-5.

Jacques Hadamard. Sur les problèmes aux dérivées partielles et leur signification physique. *Princeton University Bulletin*, 13(4):49–52, 1902.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems*, volume 33, pp. 6840–6851, 2020.

Patrick Krüger, Patrick Materne, Werner Krebs, and Hanno Gottschalk. How well do generative models solve inverse problems? a benchmark study, 2026. URL https://arxiv.org/abs/2601.23238.

Jakob Kruse, Lynton Ardizzone, Carsten Rother, and Ullrich Köthe. Benchmarking invertible architectures on inverse problems. In *First Workshop on Invertible Neural Networks and Normalizing Flows (INNF), International Conference on Machine Learning (ICML)*, 2019. URL https://invertibleworkshop.github.io/INNF_2019/accepted_papers/pdfs/INNF_2019_paper_6.pdf.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2210.02747.

Zhaocheng Liu, Dayu Zhu, Sean P. Rodrigues, Kyu-Tae Lee, and Wenshan Cai. A generative model for the inverse design of metasurfaces. *Nano Letters*, 18(10):6570–6576, 2018. doi: 10.1021/acs.nanolett.8b03171.

Zicheng Liu, Ben Fei, and Di Huang. PRO-MOF: Policy optimization with universal atomistic models for controllable MOF generation. In *International Conference on Learning Representations*, 2026.

Wei Ma, Feng Cheng, Yihao Xu, Qinlong Wen, and Yongmin Liu. Probabilistic representation and inverse design of metamaterials based on a deep generative model with semi-supervised learning strategy. *Advanced Materials*, 31(35):1901111, 2019. doi: 10.1002/adma.201901111.

Wei Ma, Zhaocheng Liu, Zhaxylyk A. Kudyshev, Alexandra Boltasseva, Wenshan Cai, and Yongmin Liu. Deep learning for the design of photonic structures. *Nature Photonics*, 15(2):77–90, 2021. doi: 10.1038/s41566-020-0685-y.

George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *Journal of Machine Learning Research*, 22(57):1–64, 2021. URL https://jmlr.org/papers/v22/19-1028.html.

John Peurifoy, Yichen Shen, Li Jing, Yi Yang, Fidel Cano-Renteria, Brendan G. DeLacy, John D. Joannopoulos, Max Tegmark, and Marin Soljačić. Nanophotonic particle simulation and inverse design using artificial neural networks. *Science Advances*, 4(6):eaar4206, 2018. doi: 10.1126/sciadv.aar4206.

Simiao Ren, Willie J. Padilla, and Jordan M. Malof. Benchmarking deep inverse models over time, and the neural-adjoint method. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

Simiao Ren, Ashwin Mahendra, Omar Khatib, Yang Deng, Willie J. Padilla, and Jordan M. Malof. Inverse deep learning methods and benchmarks for artificial electromagnetic material design. *arXiv preprint arXiv:2112.10254*, 2021. URL https://arxiv.org/abs/2112.10254.

Benjamin Sanchez-Lengeling and Alán Aspuru-Guzik. Inverse molecular design using machine learning: Generative models for matter engineering. *Science*, 361(6400):360–365, 2018. doi: 10.1126/science.aat2663.

Sunae So and Junsuk Rho. Designing nanophotonic structures using conditional deep convolutional generative adversarial networks. *Nanophotonics*, 8(7):1255–1261, 2019. doi: 10.1515/nanoph-2019-0117.

Gregory P. Spell, Simiao Ren, Leslie M. Collins, and Jordan M. Malof. Mixture manifold networks: A computationally efficient baseline for inverse modeling. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 9874–9881, 2023. doi: 10.1609/aaai.v37i8.26178.

Rohit Unni, Kan Yao, and Yuebing Zheng. Deep convolutional mixture density network for inverse design of layered photonic structures. *ACS Photonics*, 7(10):2703–2712, 2020. doi: 10.1021/acsphotonics.0c00630.

Jonas Wildberger, Maximilian Dax, , Simon Buchholz, Stephen R. Green, Jakob H. Macke, and Bernhard Schölkopf. Flow matching for scalable simulation-based inference. In *Advances in Neural Information Processing Systems*, 2023. URL https://arxiv.org/abs/2305.17161.

Yanxuan Zhao, Guopeng Sun, Shenshen Liu, Jing Yu, Peng Zhang, Jianqiang Chen, and Yueqing Wang. Flow-matching framework for two-dimensional airfoil multipoint inverse design. *Journal of Aircraft*, 2026. doi: 10.2514/1.C038770.

A Training and Hyperparameter search details

For finding the best hyperparameters, we use grid search on a number of configurations. We also follow similar search and selection philosophy as Ren et al. (2020) to improve fairness. For Flow Matching, iMF, and cDDPM, the base search grid varied hidden dimension over $\{128, 256, 512\}$, number of layers over $\{4, 6, 8\}$, conditioning-embedding dimension over $\{128, 256\}$, dropout over $\{0, 0.1\}$, and learning rate over $\{10^{-3}, 10^{-4}\}$, for a total of 72 configurations. This matches roughly the same number of configurations (80) stated in Ren et al. (2020; 2021).

Table 4 summarizes which model–dataset pairs used these independent searches and which baseline settings were adopted from the original repositories.

Table 4: Hyperparameter-search coverage using this paper's dataset order. G denotes a grid search conducted here; R denotes a repository configuration used without a new search.

Model	D1 Ball.	D2 Sine	D3 Arm	D4 ADM	D5 Stack	D6 Shell
cINN	R	R	R	R	R	R
MDN	R	R	R	R	R	R
Flow Matching	G	G	G	G	G	G
iMF	G	G	G	G	G	G
cDDPM	G	G	G	G	G	G

Selection criterion. Just like Ren et al. (2020), we selected the best configuration using r_1 computed on the validation set of each dataset. For Stack and Shell datasets, we do not use a subset of the validation data of size $N = 500$ to decrease computation time because their respective forward models are extremely slow.

A.1 Selected configurations

Table 5 lists selected configurations. Here h denotes hidden dimension, L the number of layers, c the conditioning-embedding dimension, d dropout, η learning rate, P scheduler patience, and γ learning-rate decay.

Table 5: Best validation-r_1-selected configurations across the six benchmarks. Training time is reported in seconds. Parameter counts are trainable parameters in thousands (K).

Dataset	Model	h	L	c	d	lr	Validation r_1	Time (s)	Params (K)
D1 Ballistics	FM	512	8	128	0.0	10^{-3}	1.3459×10^{-2}	44.68	9073.28
D1 Ballistics	iMF	512	4	256	0.0	10^{-3}	5.7798×10^{-1}	49.13	5000.45
D1 Ballistics	cDDPM	512	6	256	0.0	10^{-4}	1.3137×10^{-1}	87.44	7102.21
D2 Sine Wave	FM	128	8	256	0.0	10^{-3}	1.3392	30.11	1221.38
D2 Sine Wave	iMF	512	4	256	0.1	10^{-3}	1.5655	46.93	6969.47
D2 Sine Wave	cDDPM	128	8	256	0.1	10^{-4}	1.9450	55.25	1221.38
D3 Robotic Arm	FM	256	4	256	0.0	10^{-3}	9.8646×10^{-4}	24.36	1781.51
D3 Robotic Arm	iMF	512	8	256	0.0	10^{-4}	1.1867×10^{-2}	60.26	9206.02
D3 Robotic Arm	cDDPM	512	8	128	0.0	10^{-4}	8.6964×10^{-4}	107.79	9074.31
D4 ADM	FM	256	4	128	0.1	10^{-3}	5.3508×10^{-3}	37.60	3735.18
D4 ADM	iMF	256	8	128	0.0	10^{-3}	5.7045×10^{-3}	80.99	4789.13
D4 ADM	cDDPM	512	8	256	0.1	10^{-3}	7.6638×10^{-3}	123.31	11262.73
D5 Stack	FM	512	6	128	0.0	10^{-3}	6.2766×10^{-7}	143.80	7233.16
D5 Stack	iMF	512	4	128	0.0	10^{-3}	4.6792×10^{-6}	231.67	5131.40
D5 Stack	cDDPM	256	8	256	0.0	10^{-3}	2.0191×10^{-6}	149.79	3095.81
D6 Shell	FM	512	8	256	0.0	10^{-4}	4.2948×10^{-3}	179.04	9413.38
D6 Shell	iMF	512	4	256	0.0	10^{-3}	1.3826×10^{-2}	287.54	5209.35
D6 Shell	cDDPM	512	4	256	0.0	10^{-4}	5.6461×10^{-2}	121.23	5208.84

B SUPPLEMENTARY BEST-OF-T ACCURACY CURVES

Figure 3 shows the ranking of the models at different values of T. This is a more detailed view to Table 2.

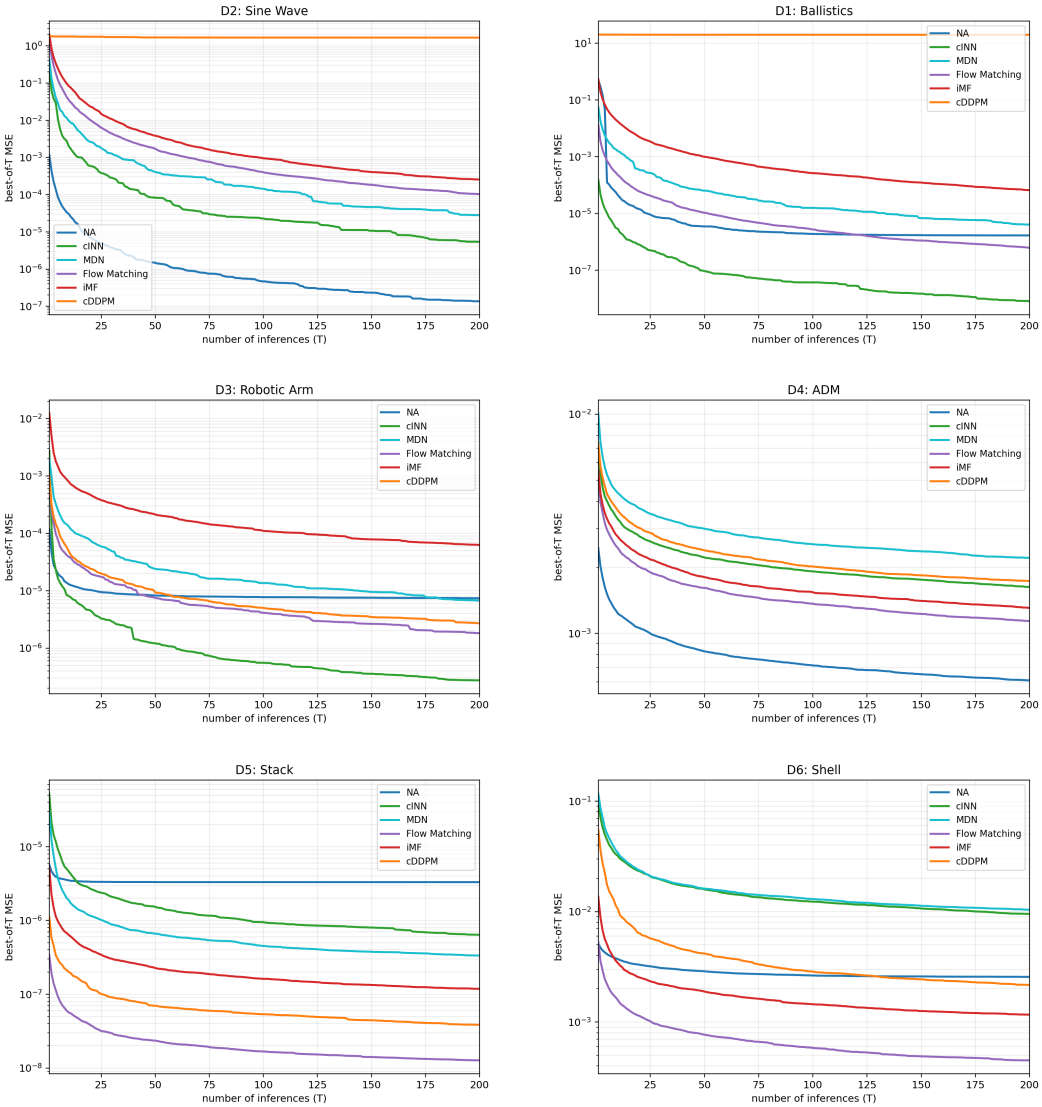

Figure 3: Best-of-T re-simulation error across proposal budgets for all six benchmarks: D1 Ballistics, D2 Sine Wave, D3 Robotic Arm, D4 ADM, D5 Stack, and D6 Shell. Lower error is better.