IMPROVING FEASIBILITY VIA FAST AUTOENCODER-BASED PROJECTIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Enforcing complex (e.g., nonconvex) operational constraints is a critical challenge in real-world learning and control systems. However, existing methods struggle to efficiently and reliably enforce general classes of constraints. To address this, we propose a novel data-driven amortized approach that uses a trained autoencoder as an approximate projector to provide fast corrections to infeasible predictions. Specifically, we train an autoencoder using an adversarial objective to learn a structured, convex latent representation of the feasible set, enabling rapid correction of neural network outputs by projecting them onto a simple convex shape before decoding into the original feasible set. We test our approach on a diverse suite of constrained optimization and reinforcement learning problems with challenging nonconvex constraints. Results show that our method effectively improves constraint satisfaction at a low computational cost, offering a practical alternative to expensive feasibility correction techniques based on traditional solvers.

1 INTRODUCTION

Many learning and control systems, across areas such as robotics, energy systems, and industrial automation, must produce outputs that respect difficult-to-satisfy (often nonconvex) constraints. Enforcing these constraints reliably and efficiently is crucial for both safety and real-world usability. A number of approaches have been proposed to address this challenge within learning-based systems, such as penalty methods (Fioretto et al. (2021); Stooke et al. (2020)), differentiable projection and correction methods (Chen et al. (2021); Pan et al. (2022)), and post-hoc repair algorithms (Zamzam & Baker (2020); Nocedal & Wright (2006)). However, each of these approaches comes with distinct trade-offs in terms of reliability, generality, computational cost, and solution quality – for instance, failing to satisfy constraints in practice, not handling general classes of constraints, being prohibitively slow to deploy in real-world systems, and/or degrading end-to-end model performance.

To address this challenge, we propose a data-driven amortized alternative to traditional constraint enforcement algorithms: a trained autoencoder that acts as an approximate projector to provide fast corrections to infeasible predictions. Specifically, we train an autoencoder to learn a structured, convex latent representation of the feasible set, in a way that enables rapid mapping from infeasible to feasible points. This autoencoder can then be leveraged as a plug-and-play "attachment" to standard neural networks. While not aiming to supersede strict projections in regimes demanding hard feasibility, this approach is designed to improve feasibility in low-latency settings or in settings where moving predictions closer to the feasible set suffices to guarantee downstream system performance.

Our key contributions are as follows:

- **Framework for data-driven feasibility improvement.** We pose an approach for learning feasibility improvement mappings that can be appended to neural networks, as an alternative to expensive, albeit exact, constraint enforcement approaches. We propose a particular instantiation of this approach, FAB, which employs an autoencoder as the basis of the learned mapping.
- **Structured latent representation learning.** We propose a mechanism to learn faithful, feasibility-preserving latent representations of the feasible set, via adversarial training.
- **Empirical validation.** We test our method on a range of constrained optimization and reinforcement learning (RL) problems. For constrained optimization problems, the method consistently provides an efficient approximate projection, learning to map solutions to the feasible

set close to 100% of the time in a fraction of a millisecond (faster than any other method). For RL settings, the method consistently provides safer actions than methods like proximal policy optimization (PPO) and trust-region policy optimization (TRPO), as well as their constrained variants. Overall, this demonstrates the promise of our approach in providing fast, reliable feasibility improvements across a wide range of settings.

2 Related Work

Amortized optimization with constraints. Also known as *learning to optimize*, amortized optimization is a paradigm designed to accelerate the process of solving optimization problems by using machine learning models as fast function approximators for optimization and control problems (Amos, 2023). Models are trained either via supervised learning on a dataset of known solutions (Chen et al., 2022), or through unsupervised/self-supervised methods, where the model's loss function incorporates the optimization problem's objectives and constraints (Van Hentenryck, 2025). A primary challenge in amortized optimization is ensuring that the model's output satisfies constraints. Several lines of work have emerged to address this, including penalty methods, differentiable projection and/or correction methods, and post-hoc "repair techniques." Penalty methods turn constrained optimization problems into unconstrained ones by incorporating penalty terms for constraint violations in the objective terms (Fioretto et al., 2021; Stooke et al., 2020; Raissi et al., 2019); however, while they incentivize feasibility, they do not guarantee it, and may produce highly infeasible solutions in practice. In contrast, differentiable projection and/or correction methods embed exact solvers directly as layers in neural networks (Pham et al., 2018; Chen et al., 2021; Pan et al., 2022; Nguyen & Donti, 2025; Donti et al., 2021). While these methods do provide feasibility guarantees, they are often either expensive to run, or highly specialized to certain classes of constraints (Min & Azizan, 2024; Tordesillas et al., 2023). Likewise, post-hoc "repair" methods (Boyd et al., 2011; Douglas & Rachford, 1956; Zamzam & Baker, 2020) are often either expensive or highly specialized, and the inherent train-test mismatch can further degrade overall performance.

In this work, we propose an alternative, data-driven approach that can learn an inexpensive, non-iterative transformation from a latent convex set to any arbitrary constraint set, thereby speeding up feasibility improvement in practice (albeit at the expense of provable guarantees). The work closest in the literature to ours is (Liang et al., 2024; 2023), proposing a homeomorphic projection approach which learns an invertible mapping from a hypersphere to a topologically-equivalent constraint set, and then uses a bisection procedure to recover feasibility. However, this procedure is designed for post-hoc feasibility correction rather than end-to-end training, and is limited to ball-homeomorphic constraint sets. Our work presents a method that can learn a mapping between a latent hypersphere and any continuous constraint set, and is compatible with end-to-end training and inference.

Safe reinforcement learning. Safe reinforcement learning (RL) is an extension of standard RL, where an agent learns to make a sequence of decisions in an environment to maximize a cumulative reward signal while also aiming to minimize the costs associated with constraint violations (Garcıa & Fernández, 2015; Gu et al., 2024). While standard RL algorithms, such as Proximal Policy Optimization (PPO, Schulman et al., 2017) and Trust Region Policy Optimization (TRPO, Schulman et al., 2015) excel in unconstrained settings, they are not equipped to handle constraints. Prominent approaches to address this include Lagrangian relaxation (Stooke et al., 2020), safe exploration (Hans et al., 2008), and differentiable projection (Pham et al., 2018; Chen et al., 2021). However, mirroring the discussion on constraint enforcement in amortized optimization, these approaches come with distinct trade-offs in terms of reliability and computational cost.

Adversarial training. Adversarial training, widely known for its application in Generative Adversarial Networks (GANs) (Goodfellow et al., 2014) and adversarially robust deep learning (Bai et al., 2021), offers a general framework for learning in the face of complex data distributions. In the canonical GAN setup, a *generator* network learns to produce realistic data samples from a noise vector, while a *discriminator* network is trained to distinguish between real and generated samples. In particular, minimax training of the generator and the discriminator drives the generator to produce increasingly plausible outputs. Inspired by this approach, our work leverages an adversarial training paradigm to learn feasible sets, where the "discriminator" is trained to distinguish between feasible and infeasible outputs produced by another model, and that output-producing model is thereby forced to learn more robust decision boundaries between feasible and infeasible points.

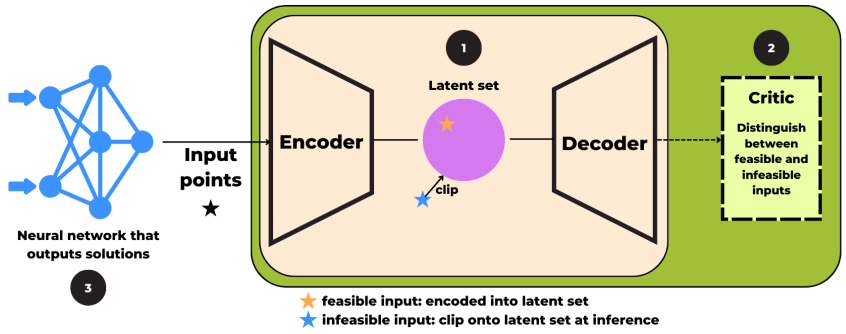

Figure 1: A schematic of FAB approximate projections. (1) Phase 1 of autoencoder training aims to enable reconstructions of the feasible set. (2) Phase 2 of autoencoder training introduces a discriminator to enable further structuring and refinement of the latent representation. (3) The trained autoencoder can be utilized as a plug-and-play attachment to another neural network model.

3 Fast Autoencoder-Based (FAB) Feasibility Improvement

In this paper, we consider the task of repeatedly solving parametric optimization problems of the form:

$$y^\star(x) \in \underset{y \in \mathcal{C}(x)}{\arg\min}\ f(y; x), \tag{1}$$

where $x \in \mathcal{X} \subseteq \mathbb{R}^m$ are the problem parameters, $y \in \mathcal{Y} \subseteq \mathbb{R}^n$ are decision variables, $f : \mathcal{Y} \times \mathcal{X} \to \mathbb{R}$ is the objective function, $\mathcal{C}(x) \subseteq \mathcal{Y}$ is a (parameter-dependent) constraint set, and $y^\star(x) \in \mathcal{C}(x)$ is a solution. Because the optimal solutions of the problems change as the parameters x change, this suggests a mapping between parameters x and optimal solutions. In addition to offline optimization, this formulation also captures RL settings, where $\mathcal{X}$ and $\mathcal{Y}$ are the state and action spaces, respectively (Amos, 2023). Our aim is to use neural networks to approximate a mapping from x to $y^\star(x)$ such that the resultant solution has low objective value, satisfies constraints, and is fast to compute at inference time.

We specifically consider neural networks of the form:

$$\hat{y}_\theta := \phi_x \circ N_\theta, \tag{2}$$

where $N_\theta : \mathcal{X} \to \mathbb{R}^d$ is a standard feedforward network with parameters θ, and $\phi_x : \mathbb{R}^d \to \mathcal{C}(x)$ is a differentiable feasibility improvement procedure that aims to map to a point in the constraint set. Prior work has considered the design of mappings ϕ_x that can ensure exact feasibility, e.g., by solving a differentiable orthogonal projection or via exact iterative procedures (see §2). While such approaches are indeed useful for, e.g., safety-critical settings where provable constraint satisfaction is a must, they tend to be relatively expensive for general classes of constraints. Instead, we propose to learn a fast, data-driven, *approximate* mapping for use in settings where (near-)feasibility is important, but where the benefits of reduced latency outweigh the need for strict constraint satisfaction.

The particular choice of ϕ_x that we propose in this work is motivated by the observation that while $\mathcal{C}(x)$ may be expensive to enforce directly, we can potentially learn a transformation between this set and some specially-structured set that is much cheaper to work with. In particular, let $E_\gamma : \mathcal{I} \to \mathcal{Z}$ be an encoder with parameters γ, where $\mathcal{I} := \mathcal{Y} \times \mathcal{X}$ and $\mathcal{Z} \subseteq \mathbb{R}^k$, and let $R_\psi : \mathcal{Z} \to \mathcal{Y}$ be a decoder ("reconstructor") with parameters ψ, where both E_γ and R_ψ are standard feedforward neural networks.[1] In addition, let $\mathcal{S} \subseteq \mathcal{Z}$ be a set that is by construction simple to map onto (e.g., a simplex or a ball), and let $\phi^{\mathcal{S}} : \mathcal{Z} \to \mathcal{S}$ be a cheap exact mapping to points in $\mathcal{S}$ (e.g., a softmax operation or closed-form projection). We then define our feasibility improvement procedure as

$$\phi_x := R_\psi \circ \phi^{\mathcal{S}} \circ E_\gamma. \tag{3}$$

In other words, given an output from N_θ alongside its corresponding input, we feed these into an encoder, map the encoded latent point into $\mathcal{S}$, and then decode the resultant point. The idea is that once E_γ and R_ψ are trained, each step of ϕ_x is cheap to execute, leading to fast overall inference.

[1] We focus our discussion in the main text on the case of input-varying constraints $\mathcal{C}(x)$, but note that the formulations for input-independent constraints are similar. In the latter case, $\mathcal{I} := \mathcal{Y}$; our autoencoder is simply a standard, rather than conditional, autoencoder; and all mentions of x in §3.1 can be dropped.

Algorithm 1 Two-Phase Autoencoder Training for FAB Feasibility Improvement

input: Dataset of feasible points $\mathcal{T}_{\text{feas}} = \{(y^{(i)}, x^{(i)}) \mid y^{(i)} \in \mathcal{C}(x^{(i)})\}$
input: Discriminator dataset $\mathcal{T}_{\text{disc}} := \left\{\left((y^{(i)}, x^{(i)}), c^{(i)}\right) \mid c^{(i)} = 1 \text{ if } y^{(i)} \in \mathcal{C}(x^{(i)}), 0 \text{ otherwise}\right\}$

procedure PHASE1($\mathcal{T}_{\text{feas}}$) *// Constraint set reconstruction*
  init encoder E_γ, decoder R_ψ
  while not converged **do** for batch of $(y, x) \in \mathcal{T}_{\text{feas}}$
    compute $\mathcal{L}_{\text{recon}}(y, x)$ via Eq. 4
    update γ, ψ using $\nabla_\gamma \mathcal{L}_{\text{recon}}, \nabla_\psi \mathcal{L}_{\text{recon}}$
  end while
  return E_γ, R_ψ
end procedure

procedure PHASE2($\mathcal{T}_{\text{disc}}$, E_γ, R_ψ) *// Latent set structuring*
  init discriminator D_ξ
  while not converged **do** for batch of $((y, x), c) \in \mathcal{T}_{\text{disc}}$
    // Update discriminator D_ξ
    compute $\mathcal{L}_{\text{disc}}(y, x, c)$ via Eq. 5
    update ξ using $\nabla_\xi \mathcal{L}_{\text{disc}}$
    // Update autoencoder (E_γ, R_ψ)
    compute $\mathcal{L}_{\text{recon}}(y, x)$, $\mathcal{L}_{\text{hinge}}(y, x, c)$ via Eq. 4, 7
    sample batch of $z \sim \mathcal{S}$
    compute $\mathcal{L}_{\text{latent}}(z)$, $\mathcal{L}_{\text{geom}}$(batch of z) via Eq. 8–9
    compute $\mathcal{L}_{\text{struc}}$ from $\mathcal{L}_{\text{recon}}, \mathcal{L}_{\text{hinge}}, \mathcal{L}_{\text{latent}}, \mathcal{L}_{\text{geom}}$ via Eq. 6
    update γ, ψ using $\nabla_\gamma \mathcal{L}_{\text{struc}}, \nabla_\psi \mathcal{L}_{\text{struc}}$
  end while
  return E_γ, R_ψ
end procedure

The key challenge with this approach is that we must now design the autoencoder parameters (γ, ψ) such that the output of ϕ_x is actually feasible, i.e., actually lies within $\mathcal{C}(x)$. To do this, we propose a two-stage autoencoder training procedure aimed at structuring the latent space $\mathcal{Z}$ such that, to the extent possible, latent points decode to our feasible set $\mathcal{C}(x)$ if and only if they lie within $\mathcal{S} \subseteq \mathcal{Z}$. A schematic of our overall approach is provided in Figure 2. We now provide additional detail on the two-stage training procedure.

3.1 Two-phase autoencoder training

We train the encoder-decoder pair (E_γ, R_ψ) in two phases: (1) a *constraint set reconstruction phase* leveraging standard autoencoder training, and (2) a *latent space structuring phase* involving adversarial training with a discriminator. Together, these aim to enable the autoencoder to faithfully reconstruct the feasible set, and encourage latent points from $\mathcal{S}$ to decode to feasible points. Pseudocode for both training phases is given in Algorithm 1

Phase 1: Constraint set reconstruction. We first train the autoencoder to reconstruct the feasible set. Specifically, we train on a dataset of exclusively feasible points, $\mathcal{T}_{\text{feas}} := \{(y^{(i)}, x^{(i)}) \mid y^{(i)} \in \mathcal{C}(x^{(i)})\}$. As standard in autoencoder training, the encoder E_γ and the decoder R_ψ are jointly trained to minimize a standard L_2 reconstruction loss to prioritize the fidelity of decoded points:

$$\mathcal{L}_{\text{recon}}(y, x) = \|y - R_\psi(E_\gamma(y, x))\|_2^2. \tag{4}$$

Phase 2: Latent space structuring. The second autoencoder training phase aims to structure the latent space such that any point sampled from $\mathcal{S} \subseteq \mathcal{Z}$, when decoded, results in a feasible point. To do this, we draw loose inspiration from generative adversarial network (GAN) training, and train our autoencoder alternatingly with a discriminator $D_\xi : \mathcal{I} \to [0, 1]$ whose role is to distinguish between feasible and infeasible points. Specifically, the discriminator maps from inputs in $\mathcal{I}$ to an estimated probability that the input is feasible. We let D_ξ be a standard feedforward neural network.

We leverage two datasets during this training phase: (a) a labeled dataset of feasible *and* infeasible points, $\mathcal{T}_{\text{disc}} := \left\{\left((y^{(i)}, x^{(i)}), c^{(i)}\right) \mid c^{(i)} = 1 \text{ if } y^{(i)} \in \mathcal{C}(x^{(i)}), 0 \text{ otherwise}\right\}$, and (b) samples $z^{(j)} \sim \mathcal{S}$ from our latent subset. Unlike in GANs, where the generator and discriminator are trained via a minimax loss, here, the autoencoder and discriminator are trained using distinct loss functions.

Specifically, the discriminator D_ξ is trained to distinguish between feasible and infeasible points, by minimizing the negative log likelihood on $\mathcal{T}_{\text{disc}}$:

$$\mathcal{L}_{\text{disc}}(y, x, c) = -\left(c \log D_\xi(y, x) + (1 - c) \log\left(1 - D_\xi(y, x)\right)\right). \tag{5}$$

The autoencoder is trained to minimize a composite loss function leveraging both $\mathcal{T}_{\text{disc}}$ and $z^{(j)} \sim \mathcal{S}$, defined as:

$$\begin{aligned}\mathcal{L}_{\text{struc}} = \frac{1}{N}\sum_{i=1}^{N}\Big[\lambda_{\text{recon}}\mathcal{L}_{\text{recon}}(y^{(i)}, x^{(i)})\Big] \;+\; \frac{1}{N}\sum_{i=1}^{N}\lambda_{\text{hinge}}\Big[\mathcal{L}_{\text{hinge}}(y^{(i)}, x^{(i)}, c^{(i)})\Big] \\ +\; \frac{1}{M}\sum_{j=1}^{M}\Big[\lambda_{\text{latent}}\mathcal{L}_{\text{latent}}(z^{(j)})\Big] \;+\; \lambda_{\text{geom}}\mathcal{L}_{\text{geom}}(\{z^{(1)}, \ldots, z^{(M)}\}),\end{aligned} \tag{6}$$

where $\lambda_{\text{recon}}, \lambda_{\text{latent}}, \lambda_{\text{hinge}}, \lambda_{\text{geom}} \in \mathbb{R}$, where $\mathcal{L}_{\text{recon}}$ is defined similarly as above, and where the remaining loss terms are defined as follows:

- The hinge loss $\mathcal{L}_{\text{hinge}}$ structures the latent space by mapping feasible points to the interior of a hypersphere and infeasible points to its exterior. Here, we write the hinge loss for the specific choice of $\mathcal{S} := \{z : \|z\|_2 \leq r\}$ for some radius r:

$$\mathcal{L}_{\text{hinge}}(y, x, c) = c\,\text{ReLU}\left(\|E_\gamma(y, x))\|_2 - r\right) + (1 - c)\,\text{ReLU}\left(r - \|E_\gamma(y, x)\|_2\right). \tag{7}$$

- The latent loss $\mathcal{L}_{\text{latent}}$ encourages outputs decoded from latent points $z \in \mathcal{S}$ to be feasible, via supervision from the discriminator:

$$\mathcal{L}_{\text{latent}}(z) = -\log D_\xi(R_\psi(z)). \tag{8}$$

- The Jacobian regularization term $\mathcal{L}_{\text{geom}}$ encourages uniform coverage of the feasible set by the decoder (Nazari et al., 2023), and is computed over a set of latent points $\hat{\mathcal{S}} \subseteq \mathcal{S}$:

$$\mathcal{L}_{\text{geom}}(\hat{\mathcal{S}}) = \text{Variance}_{z \sim \hat{\mathcal{S}}}\left[\log\det(J_z J_z^\top + \varepsilon I_k)\right], \tag{9}$$

  where $J_z = \nabla_z R_\psi(z)$ is the Jacobian of the decoder with respect to z, I_k is the $k \times k$ identity matrix, and ε is a small scalar value. This loss term measures how the decoder's output changes under small changes in the latent code (εI_k). Specifically, it calculates the determinant of the Gram matrix $J_z J_z^\top$ and measures how much the decoder locally stretches or shrinks the space.

3.2 Leveraging the trained feasibility mapping

After autoencoder training, we fix the weights of the encoder and decoder and use them to construct the feasibility mapping $\phi_x = R_\psi \circ \phi^{\mathcal{S}} \circ E_\gamma$, as also given by Equation 3. We then append the feasibility mapping to our base neural network as $\hat{y}_\theta = \phi_x \circ N_\theta$, as given by Equation 2. The resultant neural network $\hat{y}_\theta$ (with learnable parameters θ) can then be trained as usual to address the amortized optimization problem at hand, i.e., to learn approximate solutions to Problem 1. In other words, our learned mapping serves as a plug-and-play attachment to standard deep learning models that be used during both training and inference to improve model feasibility, while also being fast to run. (While in principle the encoder and decoder parameters could be further adjusted end-to-end alongside the base neural network parameters θ, rather than being fixed, we do not consider that case here.) In the following sections, we demonstrate the performance of $\hat{y}_\theta$ in learning feasible solutions to amortized optimization problems across both offline optimization and RL settings.

4 Experiments on Constrained Optimization Problems

Problem classes. We test our approach on 4 nonconvex constraint families (Fig. 2) and 3 objective types: linear, quadratic, and distance minimization. The nonconvex constraint sets are defined analytically; we consider input-independent constraint sets $\mathcal{C} \subseteq \mathcal{Y}$. The problems are formulated as:

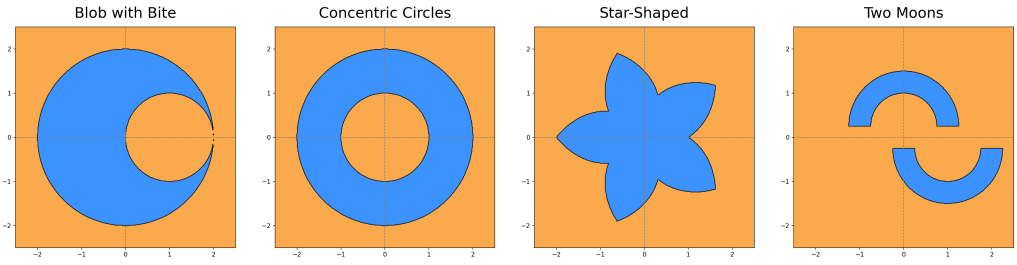

Figure 2: The nonconvex constraint sets tested in our constrained optimization settings.

Linear:

$$\min_y a^\top y \quad \text{s.t. } y \in \mathcal{C}$$

Quadratic:

$$\min_y y^\top Q y + a^\top y \quad \text{s.t. } y \in \mathcal{C}$$

Distance minimization:

$$\min_y \|y - t\|^2 \quad \text{s.t. } y \in \mathcal{C},$$

where $y \in \mathcal{Y} := \mathbb{R}^n$ is the decision variable, and the problem parameters $a \in \mathbb{R}^n$, $Q \in \mathbb{R}^{n \times n}$, $t \in \mathbb{R}^n$ vary between instances. Our goal is to learn a neural network approximator $\hat{y}_\theta$ for each problem class that minimizes the objective function while satisfying constraints.

Baselines. We compare FAB feasibility improvement against:

- **Projected Gradient Descent**. This algorithm iteratively takes a gradient descent step and then projects the current iterate onto the feasible set to enforce constraints.
- **Penalty**. A popular method that solves constrained problems by adding a penalty term to the objective that heavily penalizes constraint violations.
- **Augmented-Lagrangian**. The method combines penalty terms and Lagrange multipliers to handle constraints more robustly, allowing both primal and dual updates.
- **Interior Point** (a.k.a. barrier method). Ensures feasibility by adding barrier functions at the constraint boundaries, guiding iterates through the interior of the feasible set.
- **FSNet** (Nguyen & Donti, 2025). This method combines NNs with an exact solution of an iterative constraint violation-minimization optimization problem.
- **Homeomorphic Projection** (Liang & Chen, 2025). This method learns an invertible mapping between a hypersphere and a ball-homeomorphic constraint set, using a bisection procedure to minimize distortion.

Implementation details. We run experiments over 5 seeds with 300 problems per seed (trained 500 epochs with a batch size of 32, tested on 1,500 problems), totalling 18,000 optimization problems across the entire testing suite. All methods were run on a workstation equipped with one NVIDIA RTX 5090 GPU. We choose $\mathcal{S}$ to be a 0.5-radius hypersphere. Hyperparameters are given in Appendix A.1.

Evaluation metrics. We evaluate the methods by their feasibility rates, optimality gaps, and inference times over the 4 constraint types and 3 objective types.

Results

As presented in tables 1 and 2 (as well as the tables in Appendix B, our methods achieve a substantial reduction in computation time, with sub-millisecond inference times. In addition, they also do well at finding feasible solutions even with disjoint or non-ball-homeomorphic sets. As shown in the 2 and 3 decoder ablations, their performance improves for sets with multiple components, as each decoder "specializes" in a distinct feasible region.

Method	Quadratic			Linear			Dist. Min.		
	Feas (%)	Time (ms)	Gap (gap)	Feas (%)	Time (ms)	Gap (gap)	Feas (%)	Time (ms)	Gap (gap)
Projected Gradient	100.0	38.73	0.60	100.0	19.80	1.04	100.0	32.34	0.95
Std Dev	*0.0*	*28.68*	*0.88*	*0.0*	*31.73*	*1.11*	*0.0*	*49.68*	*2.42*
Penalty Method	59.7	64.76	0.97	43.7	38.72	1.29	55.1	55.01	5.37
Std Dev	*49.0*	*47.86*	*1.39*	*49.6*	*52.87*	*1.28*	*49.7*	*156.36*	*6.47*
Augmented Lagrangian	58.5	71.64	0.98	44.5	42.68	1.33	56.5	45.49	5.54
Std Dev	*49.3*	*64.04*	*1.42*	*49.7*	*49.49*	*1.29*	*49.6*	*33.47*	*6.73*
Interior Point	95.3	61.82	2.50	92.6	45.66	1.63	94.5	40.50	11.89
Std Dev	*21.2*	*35.51*	*2.61*	*26.2*	*56.51*	*1.66*	*22.7*	*41.82*	*11.35*
FSNet	99.5	2.66	1.15	98.7	2.49	2.09	99.8	2.31	13.63
Std Dev	*7.3*	*11.09*	*1.67*	*11.2*	*1.43*	*1.98*	*4.5*	*1.33*	*13.30*
NN 1 Decoder	100.0	0.69	0.53	100.0	0.50	1.14	100.0	0.39	2.89
Std Dev	*0.0*	*1.44*	*0.71*	*0.0*	*0.48*	*0.96*	*0.0*	*0.17*	*4.26*
NN 2 Decoder	62.0	1.14	0.52	55.9	0.59	1.09	86.0	0.45	2.59
Std Dev	*48.5*	*4.08*	*0.69*	*49.7*	*0.43*	*0.95*	*34.7*	*0.15*	*4.23*
NN 3 Decoder	95.6	1.15	0.53	95.3	0.60	1.11	94.6	0.55	3.59
Std Dev	*20.5*	*3.18*	*0.70*	*21.2*	*0.35*	*0.92*	*22.6*	*0.42*	*4.49*
FAB: Only phase 1	66.7	0.77	0.51	64.1	1.16	1.06	76.7	0.56	1.32
Std Dev	*47.1*	*0.34*	*0.62*	*48.0*	*2.88*	*0.83*	*42.3*	*0.13*	*1.48*
FAB: No hinge	100.0	0.94	0.63	100.0	1.15	1.29	100.0	0.61	6.47
Std Dev	*0.0*	*1.79*	*0.79*	*0.0*	*1.58*	*1.11*	*0.0*	*0.40*	*5.28*
FAB: No D_ξ	80.8	0.74	0.52	80.6	0.82	1.11	77.9	0.73	4.79
Std Dev	*39.4*	*0.50*	*0.69*	*39.5*	*0.47*	*0.92*	*41.5*	*0.58*	*4.34*
Homeomorphic Projection	100.0	196.631	4.37	100.0	196.247	2.58	100.0	196.021	13.36
Std Dev	*0.0*	*193.4*	*4.28*	*0.0*	*195.8*	*2.30*	*0.0*	*195.5*	*12.73*

Table 1: Mean and std. dev. for feasibility (%), time (ms), and optimality gap for each method: Blob with Bite constraint family. 5 seeds, 300 problems each per problem per objective type.

5 Experiments on Safety Gym (Safe RL)

Safe RL problem formulation. Safe RL can be framed within our parametric optimization setup from Equation 1. We consider a discrete-time dynamical system where the state evolves according to:

$$s_{k+1} = f(s_k, u_k), \tag{10}$$

where $s_k \in \mathcal{X} \subseteq \mathbb{R}^m$ is the current state and $u_k \in \mathcal{Y} \subseteq \mathbb{R}^n$ is the control action at timestep k. In this context, the actions u_k are the decision variables, and the states s_k correspond to the problem parameters. The constraint set $\mathcal{C}(s_k)$ is state-dependent. An action u_k is considered safe, i.e., $u_k \in \mathcal{C}(s_k)$, if executing it from state s_k does not lead to an immediate constraint violation (e.g. collision). The objective is to learn a policy $\pi_\theta : \mathcal{X} \to \mathcal{Y}$, that maximizes the expected cumulative reward. The goal is, therefore, to solve for a policy π_θ such that for any given state s_k, the chosen action $u_k = \pi_\theta(s_k)$ is both optimal long-term and safe.

Environment. We evaluate our approach using the SafetyPointGoal2-v0 environment from the Safety Gymnasium benchmark suite (Ji et al., 2023). In this task, an agent must navigate to a series of goal locations while avoiding randomly-placed hazards. The state vector comprises simulated sensor readings (e.g., accelerometers, gyroscope, and a LiDAR-like sensor for hazard detection), while the action vector controls the agent's forward/backward movement, as well as its rotational velocities.

Baselines. We compare FAB projections against:

- **PPO** (Schulman et al., 2017). A state-of-the-art unconstrained RL algorithm.
- **TRPO** (Schulman et al., 2015). Another robust unconstrained RL algorithm.

Method	Quadratic			Linear			Dist. Min.		
	Feas (%)	Time (ms)	Gap (gap)	Feas (%)	Time (ms)	Gap (gap)	Feas (%)	Time (ms)	Gap (gap)
Projected Gradient	80.5	79.07	1.18	76.3	48.23	0.90	79.2	69.79	4.65
Std Dev	*39.6*	*91.65*	*1.58*	*42.5*	*52.86*	*0.99*	*40.6*	*25.09*	*5.87*
Penalty Method	20.4	78.39	1.36	10.4	61.58	1.52	14.6	38.67	6.39
Std Dev	*40.3*	*71.27*	*1.67*	*30.5*	*111.21*	*1.49*	*35.3*	*10.02*	*7.07*
Augmented Lagrangian	20.9	85.71	1.33	10.9	56.98	1.56	15.7	42.58	6.43
Std Dev	*40.7*	*119.17*	*1.59*	*31.1*	*62.66*	*1.56*	*36.3*	*6.69*	*6.70*
Interior Point	78.3	82.80	1.64	77.3	46.37	1.03	79.9	42.00	6.12
Std Dev	*41.2*	*110.99*	*2.02*	*41.9*	*33.68*	*1.03*	*40.0*	*12.03*	*6.41*
FSNet	98.7	4.42	1.13	98.3	3.40	0.96	99.3	4.03	5.30
Std Dev	*11.5*	*2.77*	*1.45*	*12.8*	*2.08*	*1.01*	*8.5*	*1.67*	*5.51*
NN 1 Decoder	96.8	0.55	0.85	83.5	0.43	0.73	88.3	0.41	4.22
Std Dev	*17.6*	*0.40*	*1.16*	*37.1*	*0.27*	*0.73*	*32.2*	*0.20*	*4.15*
NN 2 Decoder	100.0	0.59	1.31	100.0	0.55	0.94	100.0	0.50	5.64
Std Dev	*0.0*	*0.35*	*1.36*	*0.0*	*0.37*	*0.89*	*0.0*	*0.25*	*5.73*
NN 3 Decoder	100.0	0.80	1.02	100.0	0.62	0.90	100.0	0.60	5.44
Std Dev	*0.0*	*1.03*	*1.15*	*0.0*	*0.36*	*0.84*	*0.0*	*0.32*	*4.84*
FAB: Only phase 1	82.5	0.64	0.86	69.9	1.07	0.72	64.1	0.54	3.83
Std Dev	*38.0*	*0.14*	*1.21*	*45.9*	*4.12*	*0.69*	*48.0*	*0.09*	*4.07*
FAB: No hinge	28.4	0.63	1.13	18.8	0.76	0.85	28.1	0.56	5.14
Std Dev	*45.1*	*0.16*	*1.23*	*39.1*	*0.84*	*0.80*	*44.9*	*0.16*	*4.96*
FAB: No D_ξ	75.1	0.64	0.86	54.4	0.67	0.70	13.6	0.55	3.61
Std Dev	*43.2*	*0.22*	*1.21*	*49.8*	*0.31*	*0.69*	*34.3*	*0.19*	*4.00*

Table 2: Mean and std. dev. for feasibility (%), time (ms), and optimality gap for each method: Two Moons constraint family. 5 seeds, 300 problems each per problem per objective type.

Table 3: SafetyGym Results

Algorithm	**Reward Mean ↑**	**Reward Std ↓**	**Cost Mean ↓**	**Cost Std ↓**	**Time Mean (s) ↓**
PPO_FAB	-1.12	**1.11**	**24.32**	**37.84**	2.75
PPO	13.26	14.05	167.46	87.06	2.47
PPO_LAG	2.24	5.10	54.10	64.50	**2.40**
TRPO	**15.58**	10.31	164.14	88.43	2.59
TRPO_LAG	2.37	8.46	89.04	187.67	2.78

- **Lagrangian PPO, Lagrangian TRPO** (Stooke et al., 2020; Ray et al., 2019). Constrained versions of PPO and TRPO that use Lagrangian relaxation to penalize constraint violations during training.

Implementation details. The autoencoder was trained on a dataset of 100,000 state-action pairs, approximately evenly balanced between safe and unsafe examples, which were collected by running a random policy in the environment. The performance of these algorithms was evaluated over 50 independent seeds. All methods were run on a workstation equipped with one NVIDIA RTX 5090 GPU. We choose $\mathcal{S}$ to be a 0.5-radius hypersphere. Hyperparameters are given in Appendix A.2.

Evaluation metrics. We assess performance based on several metrics, averaged over the evaluation episodes: episode rewards, episode costs, and mean inference time. Episode reward is the cumulative rewards obtained by the agent in one episode (which is 1000 steps). Episode cost is the total number of constraint violations (i.e., entering hazard zones).

Results

As presented in 3, PPO_FAB achieves the lowest cost among the evaluated methods, substantially outperforming standard baselines like PPO (167.46) and TRPO (164.14). Furthermore, it also ex-

hibits the highest level of reliability with the lowest standard deviation in both cost (37.84) and reward (1.11). However, this focus on safety also corresponds to a more conservative reward-seeking policy, resulting in a lower reward mean (-1.12) compared to the other algorithms.

6 CONCLUSION

In this work, we introduced a novel-data driven method, Fast Autoencoder-Based (FAB) projections for feasibility improvement, to address the critical challenge of enforcing nonconvex constraints in learning-based systems. Our method leverages a specially trained autoencoder to learn an approximate projection from a convex latent set to a general (e.g., potentially nonconvex or disjoint) feasible set. By structuring the autoencoder's latent space to correspond to a simple convex shape that is easy to map into, we can perform a fast projection in this latent space at inference and decode the result back into a point that is highly likely to be feasible.

On constrained optimization benchmarks, FAB consistently achieved near-perfect feasibility rates (often approaching 100%) across a diverse set of challenging, nonconvex feasible sets. In the SafetyGym benchmark, it consistently achieved low costs, with the median cost being 0.0. Crucially, it achieved both of these with inference times that were significantly faster than those of other methods. While methods with formal guarantees can ensure exact feasibility, their computational overhead can be prohibitive for real-time applications. Therefore, FAB presents a compelling alternative for latency-sensitive domains where rapid, high-fidelity approximate projections may be sufficient.

Limitations of FAB include lack of hard guarantees, as well as its reliance on having a representative dataset of feasible and infeasible points in order for autoencoder training to perform well. Future work includes identifying ways to improve sample efficiency, distributional performance, and interpretability of the data-driven mapping – e.g., through the use of specialized neural network structures such as input-convex neural networks (Amos et al., 2017) or operator learning-based networks (Kovachki et al., 2024), or even simpler parameterized functions – as well as exploring toolkits such as formal verification to better understand and assess the behavior of the learned data-driven mapping after it is trained. Another future direction includes exploring adaptive co-training of the data-driven feasibility improvement mapping with the neural network to which it will be attached (rather than training the mapping fully separately), motivated by prior results on the benefits of end-to-end learning (Cameron et al., 2022). Overall, we believe that the paradigm of data-driven feasibility mapping offers a compelling yet underexplored middle ground between penalty methods and exact feasibility enforcement, and we look forward to future work that investigates alternative approaches within this space beyond the one presented here.

REFERENCES

Brandon Amos. Tutorial on amortized optimization. *Foundations and Trends® in Machine Learning*, 16(5):592–732, 2023.

Brandon Amos, Lei Xu, and J Zico Kolter. Input convex neural networks. In *International conference on machine learning*, pp. 146–155. PMLR, 2017.

Tao Bai, Jinqi Luo, Jun Zhao, Bihan Wen, and Qian Wang. Recent advances in adversarial training for adversarial robustness. *arXiv preprint arXiv:2102.01356*, 2021.

Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011. URL https://web.stanford.edu/~boyd/papers/admm_distr_stats.pdf.

Chris Cameron, Jason Hartford, Taylor Lundy, and Kevin Leyton-Brown. The perils of learning before optimizing. In *Proceedings of the AAAI conference on artificial intelligence*, volume 36, pp. 3708–3715, 2022.

Bingqing Chen, Priya L. Donti, Kyri Baker, J. Zico Kolter, and Mario Bergés. Enforcing policy feasibility constraints through differentiable projection for energy optimization. In *Proceedings of the Twelfth ACM International Conference on Future Energy Systems (e-Energy '21)*, pp. 199–210. ACM, 2021. ISBN 978-1-4503-8333-2. doi: 10.1145/3447555.3464874.

Wenbo Chen, Seonho Park, Mathieu Tanneau, and Pascal Van Hentenryck. Learning optimization proxies for large-scale security-constrained economic dispatch. *Electric Power Systems Research*, 213:108566, 2022.

Priya L. Donti, David Rolnick, and J Zico Kolter. DC3: A learning method for optimization with hard constraints. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=V1ZHVxJ6dSS.

Jim Douglas, Jr. and H. H. Rachford. On the numerical solution of heat conduction problems in two and three space variables. *Transactions of the American Mathematical Society*, 82(2):421–439, 1956.

Ferdinando Fioretto, Pascal Van Hentenryck, Terrence WK Mak, Cuong Tran, Federico Baldo, and Michele Lombardi. Lagrangian duality for constrained deep learning. In *Machine learning and knowledge discovery in databases. applied data science and demo track: European conference, ECML pKDD 2020, Ghent, Belgium, September 14–18, 2020, proceedings, part v*, pp. 118–135. Springer, 2021.

Javier Garcıa and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015.

Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

Shangding Gu, Long Yang, Yali Du, Guang Chen, Florian Walter, Jun Wang, and Alois Knoll. A review of safe reinforcement learning: Methods, theories and applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

Alexander Hans, Daniel Schneegaß, Anton Maximilian Schäfer, and Steffen Udluft. Safe exploration for reinforcement learning. In *ESANN*, pp. 143–148, 2008.

Jiaming Ji, Borong Zhang, Jiayi Zhou, Xuehai Pan, Weidong Huang, Ruiyang Sun, Yiran Geng, Yifan Zhong, Josef Dai, and Yaodong Yang. Safety gymnasium: A unified safe reinforcement learning benchmark. *Advances in Neural Information Processing Systems*, 36:18964–18993, 2023.

Nikola B Kovachki, Samuel Lanthaler, and Andrew M Stuart. Operator learning: Algorithms and analysis. In Siddhartha Mishra and Alex Townsend (eds.), *Numerical Analysis Meets Machine Learning*, volume 25 of *Handbook of Numerical Analysis*, pp. 419–467. Elsevier, 2024. doi: https://doi.org/10.1016/bs.hna.2024.05.009. URL https://www.sciencedirect.com/science/article/pii/S1570865924000097.

Enming Liang and Minghua Chen. Efficient bisection projection to ensure NN solution feasibility for optimization over general set. 2025. URL https://openreview.net/forum?id=7TXdglI1g0.

Enming Liang, Minghua Chen, and Steven H Low. Low complexity homeomorphic projection to ensure neural-network solution feasibility for optimization over (non-) convex set. In *40th International Conference on Machine Learning (ICML 2023)*, pp. 20623–20649, 2023.

Enming Liang, Minghua Chen, and Steven H Low. Homeomorphic projection to ensure neural-network solution feasibility for constrained optimization. *Journal of Machine Learning Research*, 25(329):1–55, 2024.

Youngjae Min and Navid Azizan. Hardnet: Hard-constrained neural networks with universal approximation guarantees. *arXiv preprint arXiv:2410.10807*, 2024.

Philipp Nazari, Sebastian Damrich, and Fred A Hamprecht. Geometric autoencoders–what you see is what you decode. *arXiv preprint arXiv:2306.17638*, 2023.

Hoang T Nguyen and Priya L Donti. FSNet: Feasibility-seeking neural network for constrained optimization with guarantees. *arXiv preprint arXiv:2506.00362*, 2025.

Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, 2006.

Xiang Pan, Minghua Chen, Tianyu Zhao, and Steven H Low. DeepOPF: A feasibility-optimized deep neural network approach for AC optimal power flow problems. *IEEE Systems Journal*, 17 (1):673–683, 2022.

Tu-Hoa Pham, Giovanni De Magistris, and Ryuki Tachibana. Optlayer-practical constrained optimization for deep reinforcement learning in the real world. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6236–6243. IEEE, 2018.

Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707, 2019.

Alex Ray, Joshua Achiam, and Dario Amodei. Benchmarking safe exploration in deep reinforcement learning. *arXiv preprint arXiv:1910.01708*, 2019.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pp. 1889–1897. PMLR, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Adam Stooke, Joshua Achiam, and Pieter Abbeel. Responsive safety in reinforcement learning by pid lagrangian methods. In *International Conference on Machine Learning (ICML)*, 2020.

Jesus Tordesillas, Jonathan P How, and Marco Hutter. Rayen: Imposition of hard convex constraints on neural networks. *arXiv preprint arXiv:2307.08336*, 2023.

Pascal Van Hentenryck. Optimization learning. *arXiv preprint arXiv:2501.03443*, 2025.

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)*, pp. 1–6. IEEE, 2020.

A Additional Experimental Details

A.1 Constrained optimization hyperparameters

Phase 1 Training (phase1_training.py)

Batch size 256
Epochs 500
Learning rate 0.001
Optimizer Adam
Validation split 0.2
Hidden dimension 64
Number of decoders 1
Number of samples 60000

Phase 2 Training (phase2_training.py)

Batch size 256
Epochs 150
Learning rates AE=0.0005, Discriminator=0.001
Loss weights $\lambda_{\text{recon}} = 1.0$, $\lambda_{\text{feasibility}} = 1.0$, $\lambda_{\text{latent}} = 1.0$, $\lambda_{\text{hinge}} = 0.1$, $\lambda_{\text{geometric}} = 0.1$
Optimizer Adam
Discriminator type "absolute"
Critic steps 3
Validation split 0.2 (implicit from train_test_split)
Normalization enabled
Hidden dimension 64
Number of decoders 1

A.2 Safety Gym hyperparameters

Phase 1 Training (safety-gym/phase1_training.py
Batch size: 256
Epochs: 500
Learning rate: 0.001
Optimizer: Adam
Validation split: 0.2
Reconstruction weight: 1.0
Hidden dimension: 64
Number of decoders: 1
Number of samples: 100000
State dimension: 60 (safety_gym)
Action dimension: 2 (safety_gym)
Latent dimension: equal to action dimension
Phase 2 Training (safety-gym/phase2_training.py
Batch size: 256
Epochs: 100

Method	Quadratic			Linear			Dist. Min.		
	Feas (%)	Time (ms)	Gap (gap)	Feas (%)	Time (ms)	Gap (gap)	Feas (%)	Time (ms)	Gap (gap)
Projected Gradient	100.0	38.41	0.34	100.0	18.99	0.78	100.0	31.18	1.01
Std Dev	*0.0*	*39.50*	*0.41*	*0.0*	*10.97*	*0.69*	*0.0*	*27.47*	*1.86*
Penalty Method	75.8	61.37	0.87	38.0	40.03	1.17	57.3	39.41	4.84
Std Dev	*42.8*	*33.75*	*1.32*	*48.5*	*33.05*	*1.15*	*49.5*	*13.16*	*5.99*
Augmented Lagrangian	75.0	70.70	0.89	40.4	39.49	1.19	58.1	48.25	4.99
Std Dev	*43.3*	*74.02*	*1.38*	*49.1*	*20.97*	*1.10*	*49.3*	*32.70*	*6.17*
Interior Point	96.6	70.65	1.77	92.9	40.25	1.45	95.0	53.22	9.43
Std Dev	*18.1*	*75.77*	*1.99*	*25.7*	*21.66*	*1.46*	*21.8*	*62.60*	*8.70*
FSNet	99.9	3.13	1.03	99.7	3.76	1.05	99.7	5.41	7.24
Std Dev	*3.6*	*3.00*	*1.91*	*5.2*	*2.06*	*1.04*	*5.2*	*11.90*	*7.61*
FAB: NN 1 Decoder	99.9	0.56	0.34	99.9	0.60	0.77	95.4	0.64	2.01
Std Dev	*2.6*	*0.45*	*0.40*	*2.6*	*1.61*	*0.65*	*20.9*	*0.93*	*1.96*
FAB: NN 2 Decoder	100.0	0.60	0.37	99.5	0.49	0.73	94.7	0.71	2.30
Std Dev	*0.0*	*0.32*	*0.43*	*7.3*	*0.26*	*0.59*	*22.3*	*0.83*	*2.28*
FAB: NN 3 Decoder	99.9	0.65	0.34	99.9	0.55	0.75	94.5	0.93	2.02
Std Dev	*2.6*	*0.29*	*0.39*	*3.6*	*0.24*	*0.62*	*22.9*	*2.39*	*2.00*
FAB: Only phase 1	100.0	0.61	0.34	100.0	0.54	0.76	100.0	0.53	3.05
Std Dev	*0.0*	*0.08*	*0.41*	*0.0*	*0.08*	*0.65*	*0.0*	*0.08*	*3.04*
FAB: No hinge	100.0	0.60	0.38	100.0	0.54	0.83	100.0	0.52	5.00
Std Dev	*0.0*	*0.07*	*0.49*	*0.0*	*0.07*	*0.75*	*0.0*	*0.04*	*4.27*
FAB: No D_ξ	100.0	0.86	0.35	100.0	0.54	0.78	96.8	0.52	1.62
Std Dev	*0.0*	*2.05*	*0.40*	*0.0*	*0.07*	*0.65*	*17.6*	*0.04*	*1.64*
Homeomorphic Projection	100.0	128.653	3.70	100.0	126.350	2.35	100.0	126.455	12.49
Std Dev	*0.0*	*48.4*	*3.35*	*0.0*	*48.9*	*2.07*	*0.0*	*48.9*	*11.84*

Table 4: Mean and std. dev. for feasibility (%), time (ms), and optimality gap for each method: Star Shaped constraint family. 5 seeds, 300 problems each per problem per objective type.

Learning rates: AE=0.0001, Discriminator=0.0002

Loss weights: $\lambda\_recon = 1.0$, $\lambda\_feasibility = 0.5$, $\lambda\_latent = 0.5$, $\lambda\_hinge = 0.5$, $\lambda\_geometric = 0.1$

Optimizer: Adam

Discriminator type: "absolute"

Critic steps: 1

Validation split: 0.2

Hidden dimension: 64

Number of decoders: 1

B Experimental Tables

C Statement on use of Large Language Models (LLMs)

LLMs did *not* play a significant role in research ideation and/or writing for this paper. Perplexity was used to facilitate literature discovery as part of research process, alongside other (non-LLM-based) tools. LLMs were also used to aid in sentence-level rewording for a small number of sentences during the preparation of this manuscript.

Method	Quadratic			Linear			Dist. Min.		
	Feas (%)	Time (ms)	Gap (gap)	Feas (%)	Time (ms)	Gap (gap)	Feas (%)	Time (ms)	Gap (gap)
Projected Gradient	100.0	31.24	0.96	100.0	14.83	1.16	100.0	24.36	0.40
Std Dev	*0.0*	*13.55*	*1.27*	*0.0*	*14.92*	*1.16*	*0.0*	*17.34*	*0.88*
Penalty Method	35.5	56.68	1.12	42.5	39.15	1.35	49.0	40.05	5.55
Std Dev	*47.8*	*44.47*	*1.41*	*49.4*	*63.11*	*1.30*	*50.0*	*31.26*	*6.60*
Augmented Lagrangian	37.2	57.59	1.16	42.1	52.09	1.39	49.3	43.93	5.70
Std Dev	*48.3*	*43.84*	*1.43*	*49.4*	*85.00*	*1.30*	*50.0*	*46.15*	*6.91*
Interior Point	93.4	54.53	2.67	89.9	39.76	1.78	92.6	44.17	11.89
Std Dev	*24.8*	*34.56*	*2.71*	*30.2*	*29.55*	*1.79*	*26.2*	*38.55*	*11.28*
FSNet	99.9	1.75	1.29	98.5	3.04	1.72	99.1	2.03	9.72
Std Dev	*2.6*	*1.01*	*1.72*	*12.0*	*6.11*	*2.01*	*9.3*	*0.97*	*14.52*
NN 1 Decoder	100.0	0.45	1.75	100.0	0.38	1.48	99.9	0.48	5.72
Std Dev	*0.0*	*0.21*	*1.67*	*0.0*	*0.13*	*1.36*	*2.6*	*0.36*	*7.72*
NN 2 Decoder	0.2	0.50	1.49	8.3	0.49	1.52	81.4	0.52	1.54
Std Dev	*4.5*	*0.17*	*1.44*	*27.5*	*0.30*	*1.33*	*38.9*	*0.30*	*1.53*
NN 3 Decoder	0.7	0.86	1.50	3.1	0.61	1.60	82.5	0.71	1.25
Std Dev	*8.1*	*1.94*	*1.43*	*17.4*	*0.44*	*1.47*	*38.0*	*1.21*	*1.25*
FAB: Only phase 1	0.1	0.77	0.89	2.0	0.54	1.12	25.9	0.51	4.61
Std Dev	*3.6*	*2.51*	*0.82*	*14.0*	*0.11*	*0.81*	*43.8*	*0.04*	*3.63*
FAB: No hinge	1.0	0.74	1.30	1.9	0.58	1.48	68.7	0.51	7.08
Std Dev	*9.9*	*0.57*	*1.29*	*13.5*	*0.21*	*1.29*	*46.4*	*0.03*	*7.85*
FAB: No D_ξ	1.7	0.74	1.02	1.5	0.72	1.23	82.7	0.51	1.58
Std Dev	*12.8*	*0.67*	*0.92*	*12.3*	*2.67*	*0.96*	*37.9*	*0.03*	*2.15*

Table 5: Mean and std. dev. for feasibility (%), time (ms), and optimality gap for each method: Concentric Circles constraint family. 5 seeds, 300 problems each per problem per objective type.