NTopo: Mesh-free Topology Optimization using Implicit Neural Representations

Anonymous Author(s)
Affiliation
Address
email

Abstract

Recent advances in implicit neural representations show great promise when it comes to generating numerical solutions to partial differential equations. Compared to conventional alternatives, such representations employ parameterized neural networks to define, in a mesh-free manner, signals that are highly-detailed, continuous, and fully differentiable. In this work, we present a novel machine learning approach for topology optimization—an important class of inverse problems with high-dimensional parameter spaces and highly nonlinear objective landscapes. To effectively leverage neural representations in the context of mesh-free topology optimization, we use multilayer perceptrons to parameterize both density and displacement fields. Our experiments indicate that our method is highly competitive for minimizing structural compliance objectives, and it enables self-supervised learning of continuous solution spaces for topology optimization problems.

1 Introduction

Deep neural networks are starting to show their potential for solving partial differential equations (PDEs) in a variety of problem domains, including turbulent flow, heat transfer, elastodynamics, and many more [1, 2, 3, 4, 5]. Thanks to their smooth and analytically-differentiable nature, implicit neural representations with periodic activation functions are emerging as a particularly attractive and powerful option in this context [4]. In this work, we explore the potential of implicit neural representations for structural topology optimization—a challenging inverse elasticity problem with widespread application in many fields of engineering [6].

Topology optimization (TO) methods seek to find designs for physical structures that are as stiff as possible (i.e. least compliant) with respect to known boundary conditions and loading forces while adhering to a given material budget. While TO with mesh-based finite element analysis is a well-studied problem [7], we argue that mesh-free methods provide unique opportunities for machine learning. We propose the first self-supervised, fully mesh-free method based on implicit neural representations for topology optimization. The core of our approach is formed by two neural networks: a displacement network representing force-equilibrium configurations that solve the forward problem, and a density network that learns optimal material distributions in the domain of interests. To leverage the power of these representations, we cast TO as a stochastic optimization problem using Monte Carlo sampling. Compared to conventional mesh-based TO, this setting introduces new challenges that we must address. To account for the nonlinear nature of implicit neural representations, we introduce a convex density-space objective that guides the neural network towards desirable solutions. We furthermore introduce several concepts from FEM-based topology optimization methods into our learning-based Monte Carlo setting to stabilize the training process and to avoid poor local minima.

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

We evaluate our method on a set of standard TO problems in two and three dimensions. Our results indicate that neural topology optimization with implicit representations is able to match the performance of state-of-the-art mesh-based solvers. To further explore the potential advantages of this approach over conventional methods, we show how our formulation enables self-supervised learning of continuous solution spaces for this challenging class of problems.

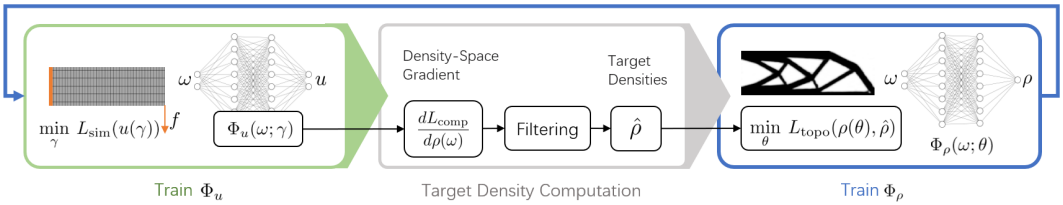

Figure 1: Neural topology optimization pipeline. We compute optimal material distributions by alternately training two neural networks: the displacement network Φ_u and the density network Φ_ρ, mapping spatial coordinates ω to equilibrium displacements u and optimal densities ρ, respectively. In each iteration, we first update Φ_u by minimizing the total potential energy of the system. We then perform sensitivity analysis to compute density-space gradients which, after applying our sensitivity filtering, give rise to target density fields $\hat{\rho}$. Finally, we update Φ_ρ by minimizing the convex objective L_{topo} based on mean squared error between current and target densities.

2 Related work

Neural Networks for Solving PDEs Deep neural networks have been widely used in different fields to provide solutions for partial differential equations for both forward simulation and inverse design problems [1, 4, 8]. In this context, PDEs can be solved either in their strong form [9, 10, 11] or variational form [12, 13]. We refer to DeepXDE [5] for a detailed review. Explorations into using deep learning alongside conventional solvers for simulation have been conducted with the goal of accelerating computations [14] or learning the governing physics [15, 16, 17, 18, 19, 20]. With their continuous and analytically-differentiable solution fields, neural implicit representations with periodic activation functions [4] offer a promising alternative to mesh-based finite element analysis. We leverage this new representation to solve high-dimensional inverse elasticity problems in a fully mesh-free manner.

Differentiable Simulation for Machine Learning There is growing interest in differentiable simulation methods that enable physics-based supervision in learning frameworks [21, 22, 23, 24, 25, 15]. Liang *et al*. [22] proposed a differentiable cloth simulation method for optimizing material properties, external forces, and control parameters. Hu *et al*. [21] targets reinforcement learning problems with applications in soft robotics. Geilinger *et al*. [24] proposed an analytically differentiable simulation framework that handles frictional contacts for both rigid and deformable bodies. To reduce numerical errors in a traditional solver Um *et al*. [25] leverage a differentiable fluid simulator inside the training loop. Similar to these existing methods, our approach relies on differentiable simulation at its core, but targets mesh-free, stochastic integration for elasticity problems.

Neural Representations Using implicit neural representation for complex signals has been an ongoing topic of research in computer vision [26, 27], computer graphics [28, 29], and engineering [4, 1]. PIFu [26] learns implicit representations of human bodies for monocular shape estimation. Sitzmann *et al*. [4] use MLPs with sinusoidal activation functions to represent signed distance fields in a fully-differentiable manner. Takikawa *et al*. [30] introduced an efficient neural representation that enables real-time rendering of high-quality surface reconstructions. Mildenhall *et al*. [28] describe a neural radiance field representation for novel view synthesis. We leverage the advantages of implicit neural representations, demonstrated in these previous works, to learn the high-dimensional solutions of topology optimization problems.

Topology Optimization with Deep Learning Topology optimization methods aim to find an optimal material distribution of a design domain given a material budget, boundary conditions, and applied forces. Building on a large body of finite-element based methods [31, 6, 32, 33, 34, 35, 36,

37, 38, 39, 40, 41, 42, 43, 44], recent efforts have explored the use of deep learning techniques in this context. One line of work leverages mesh-based simulation data and convolutional neural networks (CNNs) to accelerate the optimization process [45, 46, 47, 48, 49, 50, 51]. Perhaps closest to our work is the method by Hoyer *et al*. [52], who reparameterize design variables with CNNs, but use mesh-based finite element analysis for simulation. While Hoyer *et al*. [52] map latent vectors to discrete grid densities, Chandrasekhar *et al*. [53] use fully connected layers to learn a continuous mapping from spatial locations to density values. However, as we show in our comparisons, their choice of using ReLU activation functions leads to overly simplified solutions whose structural performance is not on par with results from conventional methods [54, 55, 56]. In addition, both methods use an explicit mesh for forward simulation and sensitivity analysis, whereas our method is entirely based on neural implicit representations.

3 Problem Statement and Overview

Given applied forces, boundary conditions, and a target material volume $\hat{V}$, the goal of topology optimization is to find the material distribution that leads to the stiffest structure. This task can be formulated as a constrained optimization problem,

$$\min_{\rho,u} \; L_{\text{comp}} = \int_\Omega e(\rho, u, \omega)\, d\omega \qquad \text{s.t.} \quad c_{\text{sim}}(\rho, u) = 0, \;\; \rho(\omega) \in \{0, 1\}, \;\; \int_\Omega \rho \, d\omega = \hat{V} \;, \tag{1}$$

where ρ is the material density field, ω runs over the domain Ω, c_{sim} are constraints requiring that the displacement u must be in static equilibrium, and e is the pointwise compliance, which is equivalent to the internal energy $e = \frac{1}{2}\,\varepsilon : \sigma$; see Section 4.1 for details. Although manufacturing typically demands binary material distributions, densities are often allowed to take on continuous values while convergence to binary solutions is encouraged. We follow the same strategy and parameterize densities ρ and displacements u using implicit neural representations, $\Phi_\rho(\omega; \theta)$ and $\Phi_u(\omega; \gamma)$, respectively. By sampling a batch of locations ω^b and $\rho^b = \rho(\omega^b)$, we compute an estimate of L_{comp} using Monte Carlo integration, $L_{\text{comp}} \approx \frac{|\Omega|}{n} \sum_i^n e(\omega_i^b)$, where $|\Omega|$ is the volume of Ω. If u is a displacement in force equilibrium, we can compute the total gradient of the compliance loss with respect to the densities of the batch as

$$s = \frac{dL_{\text{comp}}}{d\rho} = \frac{\partial L_{\text{comp}}}{\partial \rho} + \frac{\partial L_{\text{comp}}}{\partial u}\frac{du}{d\rho} = -\frac{\partial L_{\text{comp}}}{\partial \rho} \;. \tag{2}$$

We will refer to this expression as the *density-space gradient*; see the supplemental document for a detailed derivation. The density-space gradient indicates how compliance changes w.r.t. changes in the density values, assuming that the force equilibrium constraints remain satisfied. On this basis, we compute the total gradient of the compliance loss with respect to the neural network parameters as $\frac{dL_{\text{comp}}}{d\theta} = \frac{dL_{\text{comp}}}{d\rho}\frac{\partial \rho}{\partial \theta}$.

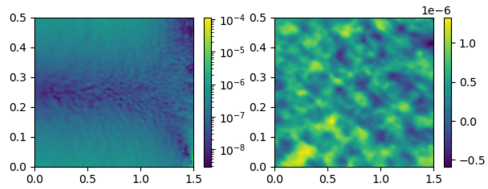

Figure 2: Failure of naive gradient descent. *Left*: negative density-space gradient on the domain for the beam example in the first iteration. Log-scale coloring is used to emphasize structure. *Right*: after one step of gradient descent (learning rate 10^{-5}), the neural network output has lost all structure from the density-space gradient.

Using this gradient together with a penalty on the volume constraint would be one potential option for solving Equation (1). In practice, however, we observed that this approach does not lead to satisfying behavior. We elaborate on this problem below. TO is a non-convex optimization problem, whose solutions depend on the optimization method [52] and the path density values take. Much research has been done on developing optimization strategies that converge to *good* local minima for FEM-based solvers. The density-space gradient is generally considered a good update direction for mesh-based approaches. However, when using standard optimizers to update the parameters for our density network, the resulting change in densities is not at all aligned with the density-space gradient; see Figure 2. We tested both ADAM [57] and SGD with little difference in the results—eventually, both approaches converge to local minima that are meaningless from a structural point of view; see Section 5.4.

To analyze this unexpected behavior, we consider the change in densities induced by a step in the negative direction of the density-space gradient $\Delta\rho = -\alpha \frac{dL_{\text{comp}}}{d\rho}$, where α is the learning rate. The corresponding first-order change in network parameters $\Delta\theta$ is $-\alpha \frac{dL_{\text{comp}}}{d\rho} \frac{\partial\rho}{\partial\theta}$. However, using this parameter change in the Taylor series expansion of the network output, we obtain

$$\Delta\rho' = \rho(\theta + \delta\theta) - \rho(\theta) = -\alpha \frac{dL_{\text{comp}}}{d\rho} \frac{\partial\rho}{\partial\theta} \frac{\partial\rho}{\partial\theta}^T + O(\alpha^2) \,. \tag{3}$$

It is evident that, unless the Jacobian $\partial\rho/\partial\theta$ of the neural network is a unitary matrix, the direction of density change $\Delta\rho'$ is different from $\Delta\rho$ even as $\alpha \to 0$.

To avoid converging to bad local minima, we seek to update the network parameters such that the network output changes along the density-space gradient. To this end, we define point-wise density targets $\hat{\rho}^b$ indicating how the network output should change. We then minimize the convex loss function

$$L_{\text{topo}}(\theta) = \frac{1}{n_b n} \sum_{b}^{n_b} \sum_{j}^{n} \left\| \rho(\omega_j^b; \theta) - \hat{\rho}(\omega_j^b) \right\|^2 \,. \tag{4}$$

While our density-space optimization strategy greatly improves results, we have still observed convergence to minima with undesirable artefacts. Drawing inspiration from mesh-based TO methods [58], we solve this problem with a sensitivity filtering approach (termed $\mathcal{FT}$ in Algorithm 1). To further accelerate convergence, we also adapt the optimality criterion method [7] to our setting ($\mathcal{OC}$ in Algorithm 1). Our resulting neural topology optimization algorithm, which we dubbed *NTopo*, is summarized in Algorithm 1 and further explained in the following Section.

Algorithm 1 NTopo: neural topology optimization.

Initialize $\gamma^{(-1)}$, $\theta^{(0)}$ for Φ_ρ, Φ_u; Initialize two optimizers: $\text{opt}_\theta, \text{opt}_\gamma$
Run initial simulation: $\gamma^{(0)} \leftarrow \arg\min_\gamma L_{\text{sim}}(\gamma, \theta^{(0)})$
for n_{opt} iterations **do**
  for n_{sim} iterations **do** $\gamma^{(l+1)} \leftarrow \text{opt}_\gamma.\text{step}(\gamma^{(l)}, \partial L_{\text{sim}}/\partial\gamma)$ **end for**
  for batch $b = 1, .., n_b$ **do** compute ρ^b, s^b **end for**
  $\hat{s} \leftarrow \mathcal{FT}(\rho^{1,\ldots,n_b}, s^{1,\ldots,n_b})$; $\hat{\rho} \leftarrow \mathcal{OC}(\rho^{1,\ldots,n_b}, \hat{s}^{1,\ldots,n_b}, \hat{V})$
  for batch $b = 1, .., n_b$ **do** $\theta^{(l+1)} \leftarrow \text{opt}_\theta.\text{step}(\theta^{(l)}, \hat{\rho}^b, \partial L_{\text{topo}}/\partial\theta)$ **end for**
end for
return Φ_ρ

4 Neural Topology Optimization

We start the technical description of our algorithm with a brief overview of the neural representation that we use. We then explain how to compute equilibrium configurations and how to update the density field. Finally, we introduce an extension of our method from individual solutions to entire solution spaces for a given continuous parameter range.

We use SIREN [4] as neural representation, which is a dense multilayer perceptron (MLP) with sinusoidal activation functions. A SIREN-MLP with l layers, $l-1$ hidden layers, and h neurons in each hidden layer is defined as

$$\Phi(x) = W_l(\phi_{l-1} \circ \phi_{l-2} \circ \ldots \circ \phi_0)(\omega_0 x) + b_l, \qquad \phi_i(x) = \sin(W_i x_i + b_i) \tag{5}$$

where x is the input vector, y is the output vector, W_i are weight matrices, b_i are biases, and ω_0 is a frequency dependent factor. We use a standard five-layer MLP with residual links and no regularization. The weights of all layers are initialized as suggested by Sitzmann *et al*. [4].

4.1 Computing Static Equilibrium Solutions

We parametrize the displacement field u using a neural network $\Phi_u(\omega; \gamma)$. To find the displacement field u in static equilibrium, we minimize the variational form of linear elasticity which is given by

$$\min_\gamma \; L_{\text{sim}}(u(\gamma)) = \int_\Omega \frac{1}{2} \varepsilon(u) : \sigma(u) \, d\omega - \int_\Omega u^T f \, d\omega \qquad \text{s.t.} \quad u|_{\partial\Omega_d} = u^* \tag{6}$$

where $\partial\Omega_d$ is the part of the boundary of the domain with prescribed displacement u^* and the tensor contraction operator ":" is defined as $\sigma : \epsilon = \sum_{ij} \sigma_{ij}\epsilon_{ij}$. In two dimensions we compute the stress tensor under plane stress assumption $\sigma = \frac{E}{1-\nu^2}\left((1-\nu)\varepsilon + \nu\,\mathrm{trace}(\varepsilon)\mathbf{I}\right)$ where ν is Poisson's ratio, E is Young's modulus, $\mathbf{I} \in \mathbb{R}^{2\times 2}$ is the identity matrix and $\varepsilon = \frac{1}{2}\left(\nabla u + \nabla u^T\right)$ is the linear Cauchy strain. In 3D we use Hooke's law $\sigma = \lambda\,\mathrm{trace}(\varepsilon)\mathbf{I} + 2\mu\epsilon$ where $\lambda = E\nu/((1+\nu)(1-2\nu))$ and $\mu = E/(2(1+\nu))$. Following SIMP [37], we parameterize the Young's modulus E using the density field as $E(\rho) = \rho^p E_0$. Larger values for the exponent p together with the volume constraint encourage binary solutions.

Sampling To evaluate the integral in Equation (6) we resort to a Quasi-Monte Carlo sampling approach. We generate stratified samples on a grid with $n_x \times n_y$ cells in 2D (and $n_x \times n_y \times n_z$ in 3D). We adjust n_x, n_y, n_z to match the aspect ratio of the domain.

Enforcing Dirichlet Boundary Conditions By constructing a function d that is zero on the fixed boundary and an interpolator of the function u^*, we enforce the displacement field $u(\omega) = d(\omega)\Phi_u(\omega) + (I \circ u^*)(\omega)$ to always satisfy the essential boundary conditions, thus turning the constrained problem into an unconstrained one [59]. We use simple boundaries in our examples for which analytic functions d are readily available; see the supplemental document for more details.

4.2 Density Field Optimization

We reparameterize the density field using a neural network Φ_ρ which maps spatial locations to their corresponding density values. The bound constraints on the densities are enforced by applying a scaled logistic sigmoid function to the network output, specifically $\rho(\omega) = \mathrm{sigmoid}(5\,\Phi_\rho(\omega))$. The total volume constraint is satisfied by the optimality criterion method described below.

Moving Mean Squared Error. Equation (4) can be interpreted as a moving mean squared error

$$\frac{1}{|\Omega|}\int_\Omega ||\rho(\omega;\theta) - \hat{\rho}(\omega)||^2 d\omega\ . \tag{7}$$

We minimize this loss using a mini-batch gradient descent strategy, where we use every batch only once. We collect multiple batches of data from $\hat{\rho}$ before we update ρ and $\hat{\rho}$, specifically $\hat{\rho}$ changes once every outer iteration. For this reason, we refer to this updating scheme as moving mean squared error (MMSE) in the following.

Sensitivity filtering $\mathcal{FT}$. In conventional TO algorithms, filtering is an essential component for discovering desirable minima that avoid artefacts such as checkerboarding [33]. While our neural representation does not suffer from the same discretization limitations that give rise to checkerboard patterns, we have nevertheless observed convergence to undesirable minima. Indeed, the neural representation alone does not remove the inherent reason for such artefacts: TO is an underconstrained optimization problem with a high-dimensional null-space. Isolating good solutions from this null-space requires additional priors, filters, or other regularizing information.

In order to address this problem, we propose a *continuous* sensitivity filtering approach that, instead of using discrete approximations [58], is based on continuous convolutions. Following this strategy, we obtain filtered sensitivities as

$$\hat{s}(\omega) = \frac{\int H(\Delta\omega)\rho(\omega+\Delta\omega)s(\omega+\Delta\omega)\,d\Delta\omega}{\max(\epsilon,\rho(\omega))\int H(\Delta\omega)\,d\Delta\omega} \tag{8}$$

where ϵ is set to 10^{-3} and H is the kernel $H(\Delta\omega) = \max\left(0, r - \|\Delta\omega\|\right)$, with radius r. Since the samples ω^i are distributed inside a grid, we can compute an approximation to the continuous filter as

$$\hat{s}^i_j = \frac{\sum_{k\in N^j} H(\omega^i_j - \omega^i_k)\rho^i_k s^i_k}{\max(\epsilon,\rho^i_j)\sum_{k\in N^j} H(\omega^i_j - \omega^i_k)} \tag{9}$$

where the neighborhood N is defined by cell sizes and radius r such that points inside the footprint of the kernel H are in the neighborhood N. Although this approximation is not an unbiased estimator of Equation (8), it led to satisfying results in all our experiments.

Multi Batch-based Optimality Criterion Method $\mathcal{OC}$ We leverage the optimality criterion method [36] to compute density targets that automatically satisfy volume constraints, thus avoiding penalty functions or other constraint enforcement mechanisms. To this end, we extend the discrete, mesh-based formulation to the continuous Monte-Carlo setting. First a set of multiplicative updates B^i are computed, which then are applied to compute the target densities $\hat{\rho}^i = \text{clamp}(\rho^i (B^i)^\eta, \max(0, \rho^i - m), \min(1, \rho^i + m))$, where m limits the maximum movement of a specific target density and η is a damping coefficient. We used $m = 0.2$ and $\eta = 0.5$. The updated B^i are computed using $B^i = \frac{-\hat{s}^i}{\lambda \frac{\partial V}{\partial \rho^i}}$ where λ is found using a binary search such that the estimated volume of the updated densities using Monte Carlo integration matches the desired volume across all batches $\frac{|\Omega|}{n_b n} \sum_b^{n_b} \sum_j^n \hat{\rho}_j^b = \hat{V}$.

4.3 Continuous Solution Space

Apart from solving individual TO problems for fixed boundary conditions and material budgets, our method can be readily extended to learn entire spaces of optimal solutions, *e.g.*, a continuous range of material volume constraints $\{\hat{V}^i\}$ or boundary conditions such as force locations. To this end, we seek to find a density function $\rho(\omega, q)$ which yields the optimal density at any point ω in the domain for any parameter vector q representing, e.g., material volume $q = [\hat{V}]$ in the target range $[\underline{V}, \overline{V}]$. In a supervised setting, a common approach is to first compute k solutions corresponding to different parameters $\{q^k\}$ and then fit the neural network using a mean squared error. By contrast, our formulation invites a fully self-supervised approach based on a modified moving mean squared error,

$$\int_{\underline{q}}^{\overline{q}} \int_\Omega \left\| \rho(\omega, q; \theta) - \hat{\rho}(\omega, q) \right\|^2 d\omega dq \; . \tag{10}$$

We minimize this loss by sampling q at random and then update the density network using the same method as described for the single target volume case.

5 Results

To analyze our method and evaluate its results, we start by comparing material distributions obtained with our approach on a set of 2D examples. We demonstrate the effectiveness of our method through comparisons to a state-of-the-art FEM solver (Section 5.1). We then investigate the ability of our formulation to learn continuous spaces of solutions in a fully self-supervised manner. Comparisons to a data-driven, supervised approach indicate better efficiency for our method, suggesting that our approach opens new opportunities for design exploration in engineering applications (Section 5.2). We then turn to TO problems with non-trivial boundary conditions and demonstrate generalization to 3D examples (Section 5.3). An ablation study justifies our choices of using sensitivity filtering and casting the nonlinear topology objective into an MMSE form (Section 5.4). We further compare the impact of different activation functions and provide detailed descriptions of our learning settings (Section 5.5).

5.1 Comparisons with FEM Solutions

We demonstrate that our results are competitive to those produced by a SIMP, a reference FEM approach [36] for mesh-based topology optimization. As can be seen in Figure 3, results are qualitatively similar, but our method often finds more complex supporting structures that lead to lower compliance values (see Table 1). To put these results in perspective, there is no topology optimization method fundamentally better than SIMP, despite decades of research. For fair comparisons, the compliance values of these structures are evaluated using the FEM solver and we consistently use fewer degrees of freedom (DoFs). The DoFs in these two methods are the number of network weights and the number of finite element cells, respectively. We refer to the supplemental material for details of these examples.

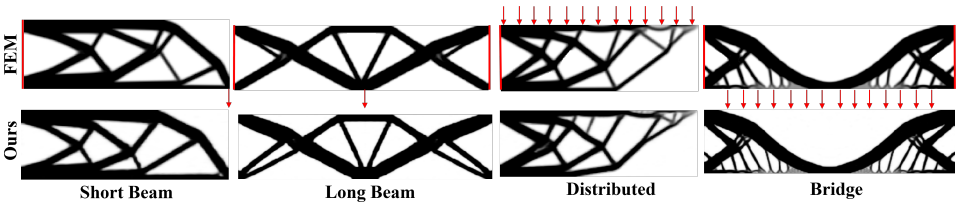

Figure 3: 2D comparison on four example problems. Solutions produced by our method are qualitatively equivalent to corresponding FEM solutions. Force and Dirichlet boundary conditions are visualized in the top-row images.

Table 1: Statistics of 2D comparisons. Our results achieve quantitatively better (lower) compliance value than the reference solver (SIMP) for all examples.

EXP.	MTHD.	COMP.	DOFS	EXP.	MTHD.	COMP.	DOFS
SHORTBEAM	SIMP	1.173×10^{-3}	30,000	DISTRIBUTED	SIMP	2.026×10^{1}	30,000
	OURS	1.166×10^{-3}	28,300		OURS	2.012×10^{1}	28,300
LONGBEAM	SIMP	2.595×10^{-4}	31,250	BRIDGE	SIMP	4.441×10^{0}	31,250
	OURS	2.592×10^{-4}	28,300		OURS	4.385×10^{0}	28,300

5.2 Learning Continuous Solution Spaces

Optimal designs for different volume constraints Here we demonstrate the capability of our method to learn continuous spaces of optimal density distributions for a continuous range of material volume constraints. We minimize the objective defined in Equation (10) for volume constraint samples drawn randomly from the range $[30\%, 70\%]$. To evaluate the accuracy of our learned solutions, we apply our single-volume algorithm for 11 discrete material volume constraints sampled uniformly across the target range. As can be seen in Figure 5, our solution space network does not compromise the quality of individual designs. The mean and maximum errors in compliance and volume violation are 0.75%, 3.83%, 0.5% and 2.83%, respectively. We therefore conclude that our model successfully learned the continuous solution space. Furthermore, we argue that the level of accuracy is acceptable for design exploration in many engineering applications.

Different solution spaces To further analyze the behavior of our solution space approach, we conducted two additional experiments: the beam example with fixed volume but varying location for the applied forces and the bridge example with varying density constraint; see Figure 4. Both examples confirmed our initial observations, showing smoothly evolving topology and compliance values close to the single-solution reference. For both examples, we sample 25 volume fractions/force locations during each iteration using stratified sampling. In addition to our single solution setup, we shuffle the density pairs randomly during the MMSE fitting step. The training takes 280 iterations in total, leading to 37.3 hours. Once trained, the inference enables real-time exploration of the solution space (0.014s/71.4FPS for 300×100 samples).

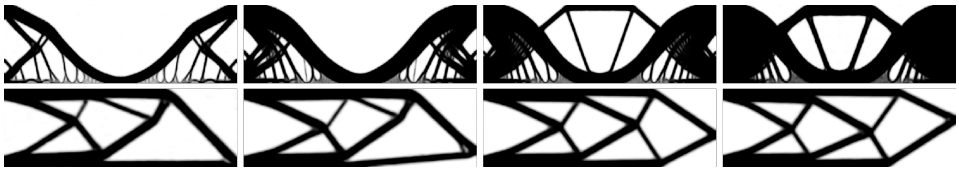

Figure 4: Different solution spaces. Here we show additional results varying the volume fraction constraints for the bridge example (first row) as well as varying the applied force location for the cantilever beam (second row). See Figure 3 for boundary conditions.

Comparison with supervised setting We further compare our self-supervised training techniques with a supervised setting under the same computational budget. The cost of performing $1,000$ optimization steps ($5.25h$) with our larger solution space network is similar to (but somewhat lower

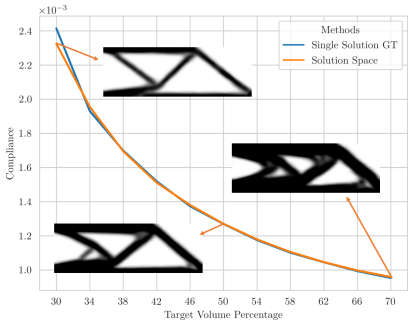

Figure 5: Results and compliance comparisons between evaluating our solution space network at discrete volume locations and the reference solution produced by our method in the single volume constraint case. As can be seen, the learned solution space does not compromise the quality of individual solutions.

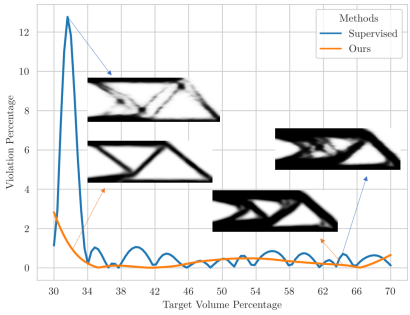

Figure 6: Comparisons with supervised setting. Our self-supervised learning methods outperforms its supervised counterparts with less computational cost. The target volume constraint is shown on the x-axis and the constraint violation (in percentage) of the resulting structures is given on the y-axis.

than) the cost of computing 11 solutions under single volume constraints. We select 11 volume constraint values uniformly and train the network from these solutions in a supervised fashion. As can be seen in Figure 6, the network performs poorly for unseen data leading to infeasible designs and significant volume violations. On the contrary, our self-supervised approach leads to physically valid material distributions.

5.3 Irregular BCs and 3D Results

Our formulation extends to more complex boundaries, which we illustrate on a set of additional examples. Figure 7 shows multiple examples where densities are constrained on circular sub-domains and the Dirichlet boundary $\partial\Omega_d$ is also circular in one of the two examples. Our method also generalizes to 3D, as demonstrated on the two examples shown in Figure 8. Due to the symmetry of the configuration, we apply symmetric boundary conditions to reduce the domain of interests to half and quarter for the cantilever beam and bridge example respectively to save computational cost and memory usage. Our method finds smooth solutions with various supporting features for the two tasks.

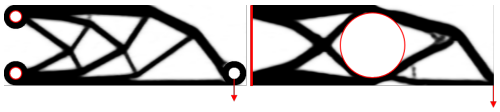

Figure 7: Curved boundaries: In the left example 3 holes have been put in the design using constraints on the density field. On the right, the density field is constrained to have a hole in the middle.

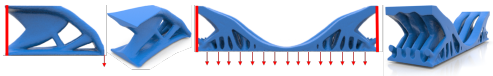

Figure 8: The solutions for two 3D examples to demonstrate the generality of our method. A 3D cantilever beam (left) and a 3D bridge (right). The mesh has been generated using a marching-cubes algorithm [60].

5.4 Ablation Studies

Here we provide evidence for the necessity of using a sensitivity filter, our moving mean squared error loss during the optimization process, and the influence of different neural networks. In the first test, we do not rely on the optimality criterion method to compute the target density field nor do we use the mean squared error. Rather, we add a soft penalty term to satisfy the volume constraint and update the neural network only once per iteration. In the second test, we adopt the proposed method without filtering. Comparing with our reference solutions at different iterations, the alternatives either converge significantly slower or arrive at poor local minima with many artefacts, such as disconnected struts or rough boundaries. We further compare our network structure with a ReLU-MLP as proposed in TOuNN [53]. As can be seen from Figure 10, this approach fails to capture much of the geometric features that our method (and SIMP) produce, leading to more compliant (i.e., less optimal) designs

(compliance: 0.001196). The Fourier feature network [29] leads to comparable results (compliance: 0.001172) and is thus also a valid option for our method.

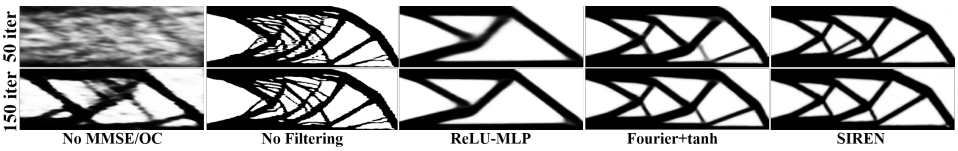

Figure 9: Ablation studies on the beam example. $1st$ column: without moving mean squared error and optimality criterion—the density-space gradient is directly applied to the network, which is updated once per optimization iteration. $2nd$ column: proposed method without filtering. $3rd$ column: proposed method using ReLU-MLP as the density network, SIREN is adopted as simulation network to obtain accurate equilibrium displacement. $4th$ column: Proposed method with Fourier feature network. $5th$ column: Proposed method with SIREN network. We verify that our moving mean squared error, filtering, and choice of activation function are all quintessential.

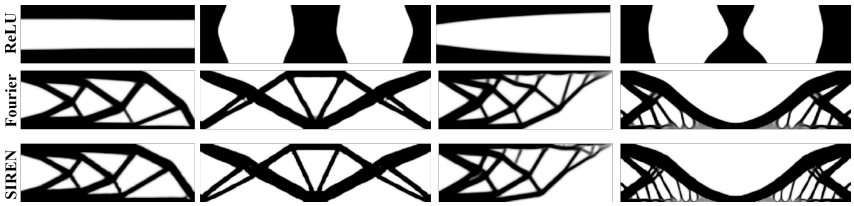

Figure 10: We use different network architectures to parameterize both the density and displacement field. Since the ReLU-MLP fails to capture the high-frequency details, its solution for the forward problem is far from being accurate, resulting in meaningless structures for the inverse problems. Fourier feature and SIREN networks produced similar results, thus, suitable for our method.

5.5 Training Details.

We use Adam [57] as our optimizer for both displacement and density networks and the learning rate of both is set to be $3 \cdot 10^{-4}$ for all experiments. We use $\omega_0 = 60$ for the first layer and 60 neurons in each hidden layer in 2D, and 180 hidden neurons in 3D. For the solution space learning setup, we use 256 neurons in each hidden layer in the density network to represent the larger solution space. For all experiments, we initialize the output of the density network close to a uniform density distribution of the target volume constraint by initializing the weights of the last layer close to zero and adjusting the bias accordingly. We used $E_0 = 1, \nu = 0.3, p = 3, n_b = 50$ and $[n_x, n_y] = [150, 50]$ in 2D and $[n_x, n_y, n_z] = [80, 40, 20]$ in 3D. The training iterations for the *ShortBeam*, *LongBeam*, *Distributed*, *Bridge* are 200, 200, 400, 200 respectively with a cost of $10.08s$ per iteration. The training iterations for the 3D examples are 100 with a per iteration cost of $131.7s$. All timings in the paper are reported on a GeForce RTX 2060 SUPER graphics card.

6 Conclusion

We propose a novel, fully mesh-free TO method using implicit neural representations. At the heart of our method lies two neural networks that represent force-equilibrium configurations and optimal material distribution, respectively. Experiments demonstrate that our solutions are qualitatively on par with the standard FEM method for structural compliance minimization problems, yet further enables self-supervised learning of solution spaces. The propose method can handle irregular boundary conditions due to its mesh-free nature and can generalize well to 3D. As such we consider it a stepping stone towards solving many varieties of inverse design problems using neural networks.

Limitations and Future Works As sigmoid function is adopted to enforce box constraints, it naturally leads do small gradients when approaching 0 or 1. Although it did not lead to convergence issue for us in practice, but better ways of enforcing box constraints in density space is an interesting avenue for future work. Similar to other Monte Carlo-based methods, advanced sampling strategies, *e.g.*, importance sampling can also be explored to speed up the optimization process.

References

[1] Oliver Hennigh, Susheela Narasimhan, Mohammad Amin Nabian, Akshay Subramaniam, Kaustubh Tangsali, Max Rietmann, Jose del Aguila Ferrandis, Wonmin Byeon, Zhiwei Fang, and Sanjay Choudhry. Nvidia simnet^{TM}: an ai-accelerated multi-physics simulation framework. *arXiv preprint arXiv:2012.07938*, 2020.

[2] 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.

[3] Chengping Rao, Hao Sun, and Yang Liu. Physics informed deep learning for computational elastodynamics without labeled data. *arXiv preprint arXiv:2006.08472*, 2020.

[4] Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33, 2020.

[5] Lu Lu, Xuhui Meng, Zhiping Mao, and George E Karniadakis. Deepxde: A deep learning library for solving differential equations. *arXiv preprint arXiv:1907.04502*, 2019.

[6] Martin Philip Bendsoe and Ole Sigmund. *Topology optimization: theory, methods, and applications*. Springer Science & Business Media, 2013.

[7] Martin P Bendsøe and Ole Sigmund. *Optimization of structural topology, shape, and material*, volume 414. Springer, 1995.

[8] Maziar Raissi, Alireza Yazdani, and George Em Karniadakis. Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. *Science*, 367(6481):1026–1030, 2020.

[9] MWMG Dissanayake and Nhan Phan-Thien. Neural-network-based approximations for solving partial differential equations. *communications in Numerical Methods in Engineering*, 10(3):195–201, 1994.

[10] Isaac E Lagaris, Aristidis Likas, and Dimitrios I Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE transactions on neural networks*, 9(5):987–1000, 1998.

[11] Justin Sirignano and Konstantinos Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375:1339–1364, 2018.

[12] Juncai He, Lin Li, Jinchao Xu, and Chunyue Zheng. Relu deep neural networks and linear finite elements. *arXiv preprint arXiv:1807.03973*, 2018.

[13] E Weinan and Bing Yu. The deep ritz method: a deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics*, 6(1):1–12, 2018.

[14] Tianju Xue, Alex Beatson, Sigrid Adriaenssens, and Ryan Adams. Amortized finite element analysis for fast pde-constrained optimization. In *International Conference on Machine Learning*, pages 10638–10647. PMLR, 2020.

[15] Philipp Holl, Vladlen Koltun, and Nils Thuerey. Learning to control pdes with differentiable physics. *arXiv preprint arXiv:2001.07457*, 2020.

[16] Peter W Battaglia, Razvan Pascanu, Matthew Lai, Danilo Rezende, and Koray Kavukcuoglu. Interaction networks for learning about objects, relations and physics. *arXiv preprint arXiv:1612.00222*, 2016.

[17] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. Learning to simulate complex physics with graph networks. In *International Conference on Machine Learning*, pages 8459–8468. PMLR, 2020.

[18] Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W Battaglia. Learning mesh-based simulation with graph networks. *arXiv preprint arXiv:2010.03409*, 2020.

[19] Radek Grzeszczuk, Demetri Terzopoulos, and Geoffrey Hinton. Neuroanimator: Fast neural network emulation and control of physics-based models. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 9–20, 1998.

[20] Daniel Holden, Bang Chi Duong, Sayantan Datta, and Derek Nowrouzezahrai. Subspace neural physics: Fast data-driven interactive simulation. In *Proceedings of the 18th annual ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 1–12, 2019.

[21] Yuanming Hu, Jiancheng Liu, Andrew Spielberg, Joshua B Tenenbaum, William T Freeman, Jiajun Wu, Daniela Rus, and Wojciech Matusik. Chainqueen: A real-time differentiable physical simulator for soft robotics. In *2019 International conference on robotics and automation (ICRA)*, pages 6265–6271. IEEE, 2019.

[22] Junbang Liang, Ming Lin, and Vladlen Koltun. Differentiable cloth simulation for inverse problems. 2019.

[23] Yuanming Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley, and Frédo Durand. Difftaichi: Differentiable programming for physical simulation. *arXiv preprint arXiv:1910.00935*, 2019.

[24] Moritz Geilinger, David Hahn, Jonas Zehnder, Moritz Bächer, Bernhard Thomaszewski, and Stelian Coros. Add: analytically differentiable dynamics for multi-body systems with frictional contact. *ACM Transactions on Graphics (TOG)*, 39(6):1–15, 2020.

[25] Kiwon Um, Philipp Holl, Robert Brand, Nils Thuerey, et al. Solver-in-the-loop: Learning from differentiable physics to interact with iterative pde-solvers. *arXiv preprint arXiv:2007.00016*, 2020.

[26] Shunsuke Saito, Tomas Simon, Jason Saragih, and Hanbyul Joo. Pifuhd: Multi-level pixel-aligned implicit function for high-resolution 3d human digitization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 84–93, 2020.

[27] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 165–174, 2019.

[28] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European Conference on Computer Vision*, pages 405–421. Springer, 2020.

[29] Matthew Tancik, Pratul P Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *arXiv preprint arXiv:2006.10739*, 2020.

[30] Towaki Takikawa, Joey Litalien, Kangxue Yin, Karsten Kreis, Charles Loop, Derek Nowrouzezahrai, Alec Jacobson, Morgan McGuire, and Sanja Fidler. Neural geometric level of detail: Real-time rendering with implicit 3d shapes. *arXiv preprint arXiv:2101.10994*, 2021.

[31] Martin Philip Bendsoe and Noboru Kikuchi. Generating optimal topologies in structural design using a homogenization method. 1988.

[32] Michael Yu Wang, Xiaoming Wang, and Dongming Guo. A level set method for structural topology optimization. *Computer methods in applied mechanics and engineering*, 192(1-2):227–246, 2003.

[33] Ole Sigmund. Morphology-based black and white filters for topology optimization. *Structural and Multidisciplinary Optimization*, 33(4-5):401–424, 2007.

[34] Martin P Bendsøe and Ole Sigmund. Material interpolation schemes in topology optimization. *Archive of applied mechanics*, 69(9-10):635–654, 1999.

[35] Martin P Bendsøe. Optimal shape design as a material distribution problem. *Structural optimization*, 1(4):193–202, 1989.

[36] Erik Andreassen, Anders Clausen, Mattias Schevenels, Boyan S Lazarov, and Ole Sigmund. Efficient topology optimization in matlab using 88 lines of code. *Structural and Multidisciplinary Optimization*, 43(1):1–16, 2011.

[37] Ole Sigmund. A 99 line topology optimization code written in matlab. *Structural and multidisciplinary optimization*, 21(2):120–127, 2001.

[38] Grégoire Allaire, François Jouve, and Anca-Maria Toader. Structural optimization using sensitivity analysis and a level-set method. *Journal of computational physics*, 194(1):363–393, 2004.

[39] Grégoire Allaire, Frédéric De Gournay, François Jouve, and Anca-Maria Toader. Structural optimization using topological and shape sensitivity via a level set method. *Control and cybernetics*, 34(1):59, 2005.

[40] Xu Guo, Weisheng Zhang, and Wenliang Zhong. Doing topology optimization explicitly and geometrically—a new moving morphable components based framework. *Journal of Applied Mechanics*, 81(8), 2014.

[41] Weisheng Zhang, Jian Zhang, and Xu Guo. Lagrangian description based topology optimization—a revival of shape optimization. *Journal of Applied Mechanics*, 83(4), 2016.

[42] Weisheng Zhang, Junfu Song, Jianhua Zhou, Zongliang Du, Yichao Zhu, Zhi Sun, and Xu Guo. Topology optimization with multiple materials via moving morphable component (mmc) method. *International Journal for Numerical Methods in Engineering*, 113(11):1653–1675, 2018.

[43] Zhen Luo, Michael Yu Wang, Shengyin Wang, and Peng Wei. A level set-based parameterization method for structural shape and topology optimization. *International Journal for Numerical Methods in Engineering*, 76(1):1–26, 2008.

[44] Nico P van Dijk, Kurt Maute, Matthijs Langelaar, and Fred Van Keulen. Level-set methods for structural topology optimization: a review. *Structural and Multidisciplinary Optimization*, 48(3):437–472, 2013.

[45] Yiquan Zhang, Airong Chen, Bo Peng, Xiaoyi Zhou, and Dalei Wang. A deep convolutional neural network for topology optimization with strong generalization ability. *arXiv preprint arXiv:1901.07761*, 2019.

[46] Yonggyun Yu, Taeil Hur, Jaeho Jung, and In Gwun Jang. Deep learning for determining a near-optimal topological design without any iteration. *Structural and Multidisciplinary Optimization*, 59(3):787–799, 2019.

[47] Saurabh Banga, Harsh Gehani, Sanket Bhilare, Sagar Patel, and Levent Kara. 3d topology optimization using convolutional neural networks. *arXiv preprint arXiv:1808.07440*, 2018.

[48] Erva Ulu, Rusheng Zhang, and Levent Burak Kara. A data-driven investigation and estimation of optimal topologies under variable loading configurations. *Computer Methods in Biomechanics and Biomedical Engineering: Imaging & Visualization*, 4(2):61–72, 2016.

[49] Zhenguo Nie, Tong Lin, Haoliang Jiang, and Levent Burak Kara. Topologygan: Topology optimization using generative adversarial networks based on physical fields over the initial domain. *arXiv preprint arXiv:2003.04685*, 2020.

[50] Xin Lei, Chang Liu, Zongliang Du, Weisheng Zhang, and Xu Guo. Machine learning-driven real-time topology optimization under moving morphable component-based framework. *Journal of Applied Mechanics*, 86(1), 2019.

[51] Qiyin Lin, Jun Hong, Zheng Liu, Baotong Li, and Jihong Wang. Investigation into the topology optimization for conductive heat transfer based on deep learning approach. *International Communications in Heat and Mass Transfer*, 97:103–109, 2018.

[52] Stephan Hoyer, Jascha Sohl-Dickstein, and Sam Greydanus. Neural reparameterization improves structural optimization. *arXiv preprint arXiv:1909.04240*, 2019.

[53] Aaditya Chandrasekhar and Krishnan Suresh. Tounn: Topology optimization using neural networks. *Structural and Multidisciplinary Optimization*, pages 1–15, 2020.

[54] Haixiang Liu, Yuanming Hu, Bo Zhu, Wojciech Matusik, and Eftychios Sifakis. Narrow-band topology optimization on a sparsely populated grid. *ACM Transactions on Graphics (TOG)*, 37(6):1–14, 2018.

[55] Yue Li, Xuan Li, Minchen Li, Yixin Zhu, Bo Zhu, and Chenfanfu Jiang. Lagrangian–eulerian multidensity topology optimization with the material point method. *International Journal for Numerical Methods in Engineering*, March 2021.

[56] Niels Aage, Erik Andreassen, Boyan S Lazarov, and Ole Sigmund. Giga-voxel computational morphogenesis for structural design. *Nature*, 550(7674):84–86, 2017.

[57] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[58] Blaise Bourdin. Filters in topology optimization. *International journal for numerical methods in engineering*, 50(9):2143–2158, 2001.

[59] Kevin Stanley McFall and James Robert Mahan. Artificial neural network method for solution of boundary value problems with exact satisfaction of arbitrary boundary conditions. *IEEE Transactions on Neural Networks*, 20(8):1221–1233, 2009.

[60] Thomas Lewiner, Hélio Lopes, Antônio Wilson Vieira, and Geovan Tavares. Efficient implementation of marching cubes' cases with topological guarantees. *Journal of graphics tools*, 8(2):1–15, 2003.

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section **??**.
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See the last paragraph.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [N/A]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Section 5.5
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [N/A]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Section 5.5
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [N/A]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]