GENERALIZABLE MOTION PLANNING VIA OPERATOR LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

In this work, we introduce a planning neural operator (PNO) for predicting the value function of a motion planning problem. We recast value function approximation as learning a single operator from the cost function space to the value function space, which is defined by an Eikonal partial differential equation (PDE). Therefore, our PNO model, despite being trained with a finite number of samples at coarse resolution, inherits the zero-shot super-resolution property of neural operators. We demonstrate accurate value function approximation at $16\times$ the training resolution on the MovingAI lab's 2D city dataset and compare with state-of-the-art neural value function predictors on 3D scenes from the iGibson building dataset. Lastly, we investigate employing the value function output of PNO as a heuristic function to accelerate motion planning. We show theoretically that the PNO heuristic is ϵ-consistent by introducing an inductive bias layer that guarantees our value functions satisfy the triangle inequality. With our heuristic, we achieve a 30% decrease in nodes visited while obtaining near optimal path lengths on the MovingAI lab 2D city dataset, compared to classical planning methods (A^*, RRT*).

1 INTRODUCTION

Classical tools for motion planning in complex environments face performance degradation as the dimension of the environment increases. Many modern approaches to motion planning introduce neural network models to enhance computational efficacy (Lehnert et al., 2024). Recent connections have been made between the motion planning and scientific machine learning communities in the context of solving partial differential equations (PDEs). In particular, the motion planning problem can be formulated from an optimal control perspective leading to the Eikonal PDE, a simplified Hamilton-Jacobi-Bellman (HJB) PDE whose solution yields the optimal value function. Recent works, including Ni & Qureshi (2023), bin Waheed et al. (2021), have explored solving the Eikonal equation via physics informed neural networks (PINNs) with the advantage that no expert training data is needed. However, as with classical PINNs, these approaches require retraining for each individual environment and, therefore, are computationally intractable for recomputing the solution quickly in dynamic environments governing the real-world.

In this work, we reformulate the solution to the Eikonal equation as an operator learning problem between function spaces. This reformulation enables the training of a single neural operator which maps an entire space of cost functions, with each cost function representing a separate environment, to a corresponding space of value functions. We design a new neural operator architecture, called Planning Neural Operator (PNO), that learns the solution operator of the Eikonal PDE. PNO approximates the optimal value function that can then be utilized for motion planning. Due to our architecture design, PNO can generalize to new environments with different obstacle geometries without retraining. Furthermore, we capitalize on the resolution invariance property of neural operators, enabling us to train using coarse resolution data (which is efficient) and apply the learned neural operator on new test maps with $16\times$ the training data resolution. This is illustrated in Fig. 1.

Our contributions can be summarized as follows.

(a) We formulate the motion planning problem as an optimal control problem whose value function satisfies the solution to the Eikonal PDE. We then prove to arbitrarily tight accuracy, the existence of a neural operator approximation to the Eikonal PDE solution operator.

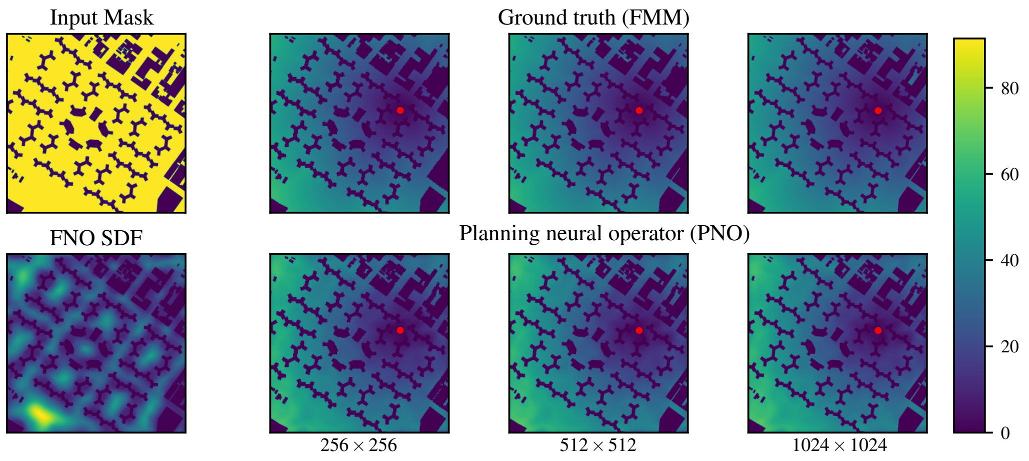


Figure 1: Example of super-resolution capabilities of PNO for motion planning on a map of NYC. The operator is trained on a dataset of resolution 64×64 and the examples shown here (resolutions 256×256, 512×512, and 1024×1024) were not seen during training. See Sec. 4 for details.

(b) We design a Planning Neural Operator (PNO) to approximate the solution operator of the Eikonal PDE and enable generalizable motion planning. By design, the architecture is (i) resolution invariant, (ii) encodes obstacle geometries into the learned weight space to enable generalization across different environments, and (iii) introduces a projection layer satisfying the triangle inequality to enable generalization across goal positions.

(c) We develop four experiments to highlight PNO's advantages, including zero-shot super resolution yielding $\sim 20\times$ speedups on the Moving AI city dataset, optimal path planning on real-world 3D iGibson building environments, and deployment as a heuristic function in A^* reducing the expanded nodes by 33% while maintaining near-optimal paths.

Related work. This work builds on ideas from two fields – PDEs and motion planning. Thus, we briefly review popular motion planning approaches while identifying connections between motion planning and PDEs. We then conclude with a brief introduction to operator learning.

Motion planning. Motion planning techniques can classically be split into two categories – sample-based algorithms and search-based algorithms. Sampling-based motion planners generate random samples in configuration space to construct graph structures whose edges connect to form collision-free paths. The two most widely known algorithms are the probabilistic roadmap (PRM) (Kavraki et al., 1996), which constructs a random graph in free space, and the rapidly-exploring random tree (RRT) (LaValle, 2006; Karaman & Frazzoli, 2011), which grows a tree stemming from a single start node to find feasible paths. Since the development of these methods, there have been substantial improvements to the efficiency using ideas such as biased sampling with heuristic functions (Gammell et al., 2015; 2014), parallelization (Sundaralingam et al., 2023), domain decomposition (Chamzas et al., 2021), and potential fields (Qureshi & Ayaz, 2015). We refer readers to Orthey et al. (2024) for a comprehensive review.

In contrast with sampling-based planners, search-based planners trade speed for accuracy to finding solutions of minimal cost. The most well known approaches are Dijkstra's algorithm (Dijkstra, 1959) and A* (Hart et al., 1968; Likhachev et al., 2003), which prioritizes the search order using a heuristic function. More recent search-based planners include Cohen et al. (2010), Liu et al. (2018) which plan in higher dimensional spaces using motion primitives. Lastly, we briefly mention planners based on Eikonal solvers such as Sethian (1996), Janson et al. (2015), Valero-Gomez et al. (2013), Pêtrès et al. (2005). These approaches, instead of planning by exploring nodes in configuration space, aim to solve the Eikonal PDE yielding an optimal function for the resulting environment and then perform corresponding gradient descent.

Despite the success of classical planning methods, they suffer from computational limitations when scaling to higher dimensions. Thus, with the recent success of deep learning, neural motion planners (NMPs) have exploded in popularity for reducing the computational burden of motion planning. For example, the earliest works such as value iteration networks (VIN) (Tamar et al., 2016) and motion planning networks (MPN) (Qureshi et al., 2021) as well as more recent approaches such as Zeng et al. (2019) and Fishman et al. (2022) attempt to plan completely from neural networks designs. Additionally, many approaches instead augment classical algorithms. For example, Chen et al. (2020), Zhang et al. (2022) speedup the collision checker in RRT and Ichter et al. (2019), Wang et al. (2020) learn sampling biases. Please refer to McMahon et al. (2022) for a comprehensive review. Other approaches use gradient information directly from either Neural Distance Fields (Ortiz et al., 2022; Liu et al., 2022) or Neural Radiance Fields (NeRF) (Adamkiewicz et al., 2022). Lastly, we mention two approaches, which are closely related to our formulation. The first, NTFields (Ni & Qureshi, 2023) learns a PINN for motion planning based on the Eikonal equation, but requires retraining for new environments. Meanwhile Li et al. (2022) learns implicit functions for planning but has no guarantees of obstacle avoidance and pays large startup costs for data generation.

Operator learning. The goal of operator learning is the design of models for approximating infinite-dimensional mappings across function spaces. Neural operators were first introduced in a seminal paper by Chen & Chen (1995) that designed the first architecture and a corresponding universal operator approximation theorem for approximating PDE solutions. This work was recently rediscovered and extended using modern neural network models under the DeepONet framework (Lu et al., 2021; Deng et al., 2022; Lanthaler et al., 2022). Almost concurrently, a new operator learning architecture, built on the Fourier transform was introduced in Li et al. (2021); Kovachki et al. (2023; 2021) for learning PDE solutions. Since, there have been a series of expansive papers addressing numerous challenges from non-uniform geometries (Liu et al., 2023; Li et al., 2023a;b; Fang et al., 2024) to architectural limitations (Seidman et al., 2022; You et al., 2022a; Gupta et al., 2021; Hao et al., 2023; Furuya et al., 2023) of the original works. Furthermore, neural operators have been employed in an eclectic range of applications spanning, but not limit to, weather forecasting (Kurth et al., 2023), material modeling (You et al., 2022b), seismic wave propagation (Yang et al., 2021), and control of PDEs (Bhan et al., 2023). Lastly, from a theoretical perspective, Lanthaler et al. (2023) unified the neural operator framework, introducing an abstract formulation of the operator learning problem and a corresponding universal approximation theorem. This framework, named the *nonlocal neural operator*, provides universal approximation theorems (see Theorem 2) for a wide array of architectures, including Fourier neural operator (FNO) and DeepONet.

2 Eikonal PDE formulation of the motion planning problem

2.1 Motion planning as an optimal control problem

Consider a continuous-time dynamical system with state $\boldsymbol{x}(t) \in \mathcal{X} \subset \mathbb{R}^n$, compact state space $\mathcal{X}$, control input $\boldsymbol{u}(t) \in \mathcal{U} \subset \mathbb{R}^k$, compact control space $\mathcal{U}$, and dynamics:

$$\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t), \boldsymbol{u}(t)). \tag{1}$$

Given an initial condition $\boldsymbol{x}(0) = \boldsymbol{x}_0$, we aim to design a control policy $\pi : \mathcal{X} \to \mathcal{U}$ that maintains the system state $\boldsymbol{x}(t)$ in a *safe set* $\mathcal{S} \subset \mathcal{X}$ for all t and drives it to a *goal set* $\mathcal{G} \subset \mathcal{S}$. We formulate this problem through an infinite-horizon first-exit optimal control problem (OCP) as Bertsekas (2017)

$$\min_{\pi} \; c_\tau(\boldsymbol{x}(\tau)) + \int_0^\tau c(\boldsymbol{x}(t), \pi(\boldsymbol{x}(t)))dt\,, \tag{2a}$$

$$\text{s.t.} \;\; \tau = \inf\{t \in \mathbb{R}_{\geq 0} \mid \boldsymbol{x}(t) \in \mathcal{G}\}, \tag{2b}$$

$$\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t), \pi(\boldsymbol{x}(t)))\,,\; \boldsymbol{x}(0) = \boldsymbol{x}_0, \tag{2c}$$

$$\boldsymbol{x}(t) \in \mathcal{S}\,,\; \forall t, \tag{2d}$$

where τ is the *first-exit time*, the smallest time at which the system reaches the goal region $\mathcal{G}$, $c : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$ is the *stage cost function* that measures the quality of the system trajectory, and $c_\tau : \mathcal{X} \to \mathbb{R}$ is a *terminal cost function* evaluated when the system reaches the goal region. To ensure the well-posedness of (2), we enforce the following standard assumption (Liberzon, 2012).

Assumption 1. *There exists a policy $\pi : \mathcal{X} \to \mathcal{U}$ such that, for any initial condition $\boldsymbol{x}(0) \in \mathcal{S}$, the trajectories of the closed-loop system with $\boldsymbol{u}(t) = \pi(\boldsymbol{x}(t))$ in (1), reach the goal region $\mathcal{G}$ by a finite first-exit time $\tau < \infty$.*

As standard, the optimal value function $V : \mathcal{X} \to \mathbb{R}$ is the minimum cost attained in (2) given by

$$V(\boldsymbol{x}) := \min_{\pi}\{c_\tau(\boldsymbol{x}(\tau)) + \int_0^\tau c(\boldsymbol{x}(t), \pi(\boldsymbol{x}(t)))dt\}, \tag{3}$$

subject to the constraints in (2). A sufficient condition for optimality is that the function V satisfies the Hamilton-Jacobi-Bellman equation:

$$0 = \min_{\boldsymbol{u} \in \mathcal{U}} \left\{ \nabla V(\boldsymbol{x})^\top f(\boldsymbol{x}, \boldsymbol{u}) + c(\boldsymbol{x}, \boldsymbol{u}) \right\}, \qquad \forall \boldsymbol{x} \in \mathcal{S} \setminus \mathcal{G}, \tag{4a}$$

$$V(\boldsymbol{x}) = c_\tau(\boldsymbol{x}), \qquad \forall \boldsymbol{x} \in \mathcal{G}, \tag{4b}$$

where ∇V denotes the gradient of V. As in (4a), (4b), to simplify notation, we omit the time argument for $\boldsymbol{x}(t)$ and $\pi(\boldsymbol{x}(t))$ in the remainder of the paper.

2.2 Eikonal equation for optimal motion planning

In this work, we capitalize on the structure of common motion planning problems to simplify the HJB equation. In practice, one commonly splits the OCP described above into two subproblems: i) a motion planning problem, which determines a reference path from the initial state $\boldsymbol{x}(0)$ to the goal region $\mathcal{G}$ without considering the dynamics constraint in (2c), and ii) a control problem, which determines a control policy for the system to track the planned reference path. We focus on the planning problem, which simplifies the system dynamics to be fully actuated, $\dot{\boldsymbol{x}}(t) = \boldsymbol{u}(t)$.

Since a path planned with these simplified dynamics may not be dynamically feasible for the original system, it is common to limit the magnitude of the control input $\boldsymbol{u}(t)$. Thus, we constrain the input to an admissible control set $\mathcal{U} = \{\boldsymbol{u} \in \mathbb{R}^k | \|\boldsymbol{u}\| = 1\}$ of unit vectors. Finally, when the input magnitude is already constrained by $\mathcal{U}$, it is not necessary to penalize it in the stage cost. Hence, we let $c(\boldsymbol{x}, \boldsymbol{u}) = c(\boldsymbol{x})$ be a control-independent stage cost, and let $c_\tau(\boldsymbol{x}) = c(\boldsymbol{x})$ to simplify the presentation. With all the above simplifications, the optimal control problem in (2) reduces to the *optimal motion planning* problem:

$$\min_{\pi} \; c(\boldsymbol{x}(\tau)) + \int_0^\tau c(\boldsymbol{x}(t)) dt, \tag{5a}$$

$$\text{s.t. } \tau = \inf\{t \in \mathbb{R}_{\geq 0} \mid \boldsymbol{x}(t) \in \mathcal{G}\}, \tag{5b}$$

$$\dot{\boldsymbol{x}}(t) = \pi(\boldsymbol{x}(t)), \; \boldsymbol{x}(0) = \boldsymbol{x}_0, \tag{5c}$$

$$\boldsymbol{x}(t) \in \mathcal{S}, \quad \|\pi(\boldsymbol{x}(t))\| = 1, \quad \forall t, \tag{5d}$$

and the associated HJB equation in (4) simplifies to:

$$0 = \min_{\boldsymbol{u} \in \mathcal{U}} \left\{ \nabla V(\boldsymbol{x})^\top \boldsymbol{u} + c(\boldsymbol{x}) \right\}, \qquad \forall \boldsymbol{x} \in \mathcal{S} \setminus \mathcal{G}, \tag{6a}$$

$$V(\boldsymbol{x}) = c(\boldsymbol{x}), \qquad \forall \boldsymbol{x} \in \mathcal{G}. \tag{6b}$$

Note that the minimization in (6a) is now linear in $\boldsymbol{u}$ with the constraint $\|\boldsymbol{u}\| = 1$. The minimizer is readily computable in closed-form $\boldsymbol{u} = -\frac{\nabla V(\boldsymbol{x})}{\|\nabla V(\boldsymbol{x})\|}$, yielding a PDE in the Eikonal class:

$$\|\nabla V(\boldsymbol{x})\| = c(\boldsymbol{x}), \qquad \forall \boldsymbol{x} \in \mathcal{S} \setminus \mathcal{G}, \tag{7a}$$

$$V(\boldsymbol{x}) = c(\boldsymbol{x}), \qquad \forall \boldsymbol{x} \in \mathcal{G}. \tag{7b}$$

2.3 Properties and universal approximation of the Eikonal solution operator

The solution to the Eikonal PDE in (7) can be viewed as an infinite-dimensional nonlinear operator Ψ that maps cost functions $c(\boldsymbol{x})$ into corresponding value function solutions $V(\boldsymbol{x})$. Given an arbitrary cost function c, there is no known analytical representation for the operator $\Psi(c) = V$ and in fact, V is not guaranteed to be continuous.

In this work, we ask is it possible to learn an approximation to the nonlinear operator Ψ and, if so, how can we design a resolution invariant neural network architecture to approximate Ψ? To do so, we begin by theoretically justifying our methodology, proving the existence of such a neural operator approximation under mild regularity assumptions.

We begin by recalling the Universal Approximation Theorem of Neural Operators (Lanthaler et al., 2023, Theorem 2.1), which provides the foundation for showing the existence of a neural operator $\hat{\Psi}$ that is an ϵ-close approximations to a nonlinear operator Ψ. A *neural operator* $\hat{\Psi}$ is a neural network architecture consisting of a lifting neural network, a kernel approximation neural network, and a projection neural network. A formal definition of neural operator and statement of its universal approximation theorem is provided in Appendix B.1 and Theorem 2 in Appendix B.1.

Following the universal approximation theorem, there are two challenges to establishing the existence of a neural operator $\hat{\Psi}$ for approximating the Eikonal PDE solution. First, we require that the domain is a compact set of continuous functions and second we require continuity of the operator. For the first condition, since the solution to the Eikonal PDE is not continuous, we aim to learn the continuous and unique viscosity solution (see Appendix C). Let $\mathcal{F}_c$ be the function space of costs $\mathcal{X} \to \mathbb{R}$, namely $c(\cdot) \in \mathcal{F}_c$. Likewise, let $\mathcal{F}_v$ be the function space of value functions $\mathcal{X} \to \mathbb{R}$, that is $V(\cdot) \in \mathcal{F}_v$. To ensure well-posed solutions of Problem (7), we require the following assumption.

Assumption 2 (Continuity and uniform boundedness of cost function space)**.** *The cost function space $\mathcal{F}_c$ is uniformly equicontinuous. Further, there exists a constant $\theta > 0$ such that*

$$\inf_{\boldsymbol{x} \in \mathcal{X}} c(\boldsymbol{x}) > 1/\theta, \quad \sup_{\boldsymbol{x} \in \mathcal{X}} c(\boldsymbol{x}) < \theta, \quad \forall c \in \mathcal{F}_c. \tag{8}$$

The set $\mathcal{F}_c$ needs to be equicontinuous to ensure the continuity of the operator Ψ. The uniform positivity of the cost c is required to ensure the existence and uniqueness of viscosity solutions to the Eikonal PDE. Lastly, we note by the continuity of cost functions in Assumption 2 and the Arzelá Ascoli theorem (Folland, 1999, Theorem 4.43), $\mathcal{F}_c$ is compact. From these assumptions, it is well established that a viscosity solution $\Psi : \mathcal{F}_c \to \mathcal{F}_v$ to the Eikonal PDE in (7) exists (see Appendix C). We are now ready to show the existence of a neural operator approximation to Ψ.

Theorem 1. *Let Assumptions 1, 2 hold and consider $\Psi : \mathcal{F}_c \to \mathcal{F}_v$ as the solution to (7). Then, for any $\epsilon > 0$, there exists a neural operator $\hat{\Psi} : \mathcal{F}_c \to \mathcal{F}_v$ such that*

$$\sup_{c \in \mathcal{F}_c} \|\Psi(c) - \hat{\Psi}(c)\| \leq \epsilon. \tag{9}$$

Proof of Theorem 1 is presented in Appendix A.1. While this shows the *existence* of a neural operator for approximating the solution of the Eikonal PDE, it does not tell us how to design the neural operator $\hat{\Psi}$. This is the focus of the next section.

3 Designing Neural Operators for Eikonal PDEs

We introduce a new neural operator architecture, which we call *Planning Neural Operator* (PNO), to approximate the solution operator $\Psi(c) = V$ of the Eikonal PDE in (7). We ensure that our architecture achieves three goals: (i) invariance to resolution differences between train and test maps, (ii) generalization across environment geometries, and (iii) generalization across goal positions.

To achieve property (i), we employ the popular resolution-invariant Fourier neural operator (FNO) architecture (Li et al., 2021) with two key extensions. First, to enable generalization across environments, we *hard encode* the environment geometries into the operator structure. Second, to achieve goal generalization, we redesign the output layer of the FNO model to ensure that the predicted value function satisfies the triangle inequality. Assuming that the goal set is a singleton $\mathcal{G} = \{\boldsymbol{g}\}$, to simplify the presentation, let $V(\boldsymbol{x}, \boldsymbol{g})$ denote the solution to (7) with goal $\boldsymbol{g}$. By the principle of optimality (Bellman, 1957), the value function V satisfies:

$$V(\boldsymbol{x}, \boldsymbol{g}) \leq V(\boldsymbol{x}, \boldsymbol{y}) + V(\boldsymbol{y}, \boldsymbol{g}), \quad \forall \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{g} \in \mathcal{S}. \tag{10}$$

Thus, we seek a neural operator $V(\boldsymbol{x}, \boldsymbol{g}) = (\hat{\Psi}(\boldsymbol{c}; \boldsymbol{g}))(\boldsymbol{x})$ that encodes this property of the value function. Next, we review the structure of the FNO model, which ensures property (i). Then, we present our PNO architecture that extends the FNO model to enable properties (ii) and (iii).

3.1 Review of FNO and DAFNO

An FNO model consist of three components: (1) a lifting neural network that maps to high dimensional space, (2) a series of Fourier layers, and (3) a projection neural network to the target resolution. It can be viewed as a mapping $\hat{\Psi} = Q \circ L_M \circ \cdots \circ \circ L_1 \circ R$, where $R(c(\boldsymbol{x}), \boldsymbol{x})$ is the lifting network, L_m for $m = 1, \ldots, M$ are the hidden Fourier layers, and Q is the projection network. Details of the FNO architecture are provided in Appendix B.1. The Fourier layers consist of applying a Fast

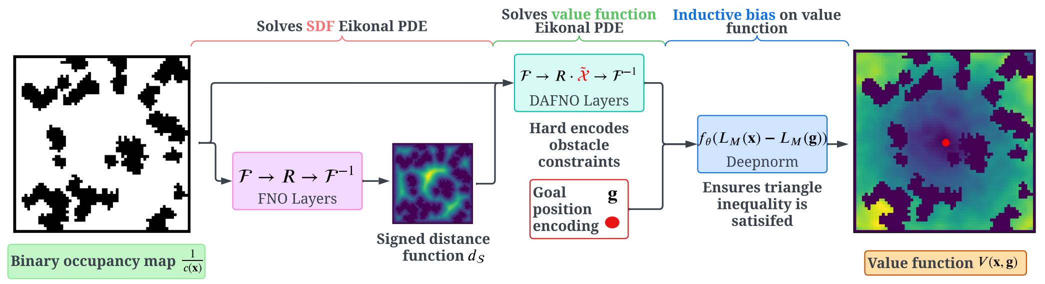

Solves SDF Eikonal PDE
Solves value function Eikonal PDE
Inductive bias on value function
Binary occupancy map $\frac{1}{c(\mathbf{x})}$
$\mathcal{F} \to R \to \mathcal{F}^{-1}$
FNO Layers
Signed distance function d_S
$\mathcal{F} \to R \cdot \tilde{\mathcal{X}} \to \mathcal{F}^{-1}$
DAFNO Layers
Hard encodes obstacle constraints
Goal position encoding $\mathbf{g}$
$f_\theta(L_M(\mathbf{x}) - L_M(\mathbf{g}))$
Deepnorm
Ensures triangle inequality is satisifed
Value function $V(\mathbf{x}, \mathbf{g})$

Figure 2: PNO network architecture. The input to a PNO is a binary occupancy grid, which is transformed into a sign distance function (SDF) via an independently trained FNO. This, along with the original binary map is passed to a series of modified FNO layers which hard encode the obstacles. Finally, the result, along with the goal, is then fed to a projection layer (ensuring satisfaction of the triangle inequality) obtaining the final value function prediction.

Fourier transform (FFT) to the input, multiplying that transform by a learnable weight matrix and then transforming back via an Inverse FFT. The key idea is that, by learning in frequency space, the operator is resolution invariant as one can project any desired resolution to and from a pre-specified number of Fourier modes.

However, to perform a Fourier transform, the FNO model needs a rectangular domain and thus cannot be applied to non-uniform geometries. Recently, a new model, titled Domain Agnostic FNO (DAFNO) (Liu et al., 2023) addressed this issue. DAFNO circumscribes the non-uniform geometry of the solution operator into a rectangle and then encodes, in the Fourier layer, where the true solution domain is active. To achieve such an encoding, DAFNO explicitly introduces an indicator matrix into Fourier layer multiplication explicitly forcing the points in the domain padding to 0. Details of the DAFNO architecture are provided in Appendix B.2. We take inspiration from this idea to explicitly encode the *obstacles* in the motion planning problem using an indicator function.

3.2 PNO ARCHITECTURE: GENERALIZING ACROSS ENVIRONMENTS

The PNO model maps a cost function to the corresponding value function of the motion planning problem. We consider a minimum-time motion planning problem by choosing the cost function as

$$c(\boldsymbol{x}) = \begin{cases} 1 & \boldsymbol{x} \in \mathcal{S}, \\ \infty & \boldsymbol{x} \in \mathcal{X} \backslash S, \end{cases} \tag{11}$$

where we penalize the unsafe set with infinite cost. In practice, to avoid numerical instability, we use $1/c(\boldsymbol{x})$, or a binary occupancy map as the input function. We, then, employ a neural network for our lifting layer as in FNO. To achieve property (ii), we ensure that PNO explicitly captures the obstacle configuration of the environment. That is, we *hard encode* the obstacle locations in the Fourier layer of our architecture. To do so, taking inspiration from DAFNO, we modify the Fourier layer by multiplying the learnable weight matrix with the following smoothed indicator function

$$\tilde{\chi}(\boldsymbol{x}) := \tanh(\beta d_{\mathcal{S}}(\boldsymbol{x}))(1/c(\boldsymbol{x}) - 0.5) + 0.5\,. \tag{12}$$

where β is a smoothing hyperparameter and $d_{\mathcal{S}}$ is the sign distance function (SDF) given by

$$d_{\mathcal{S}}(\boldsymbol{x}) = \begin{cases} \inf\limits_{\boldsymbol{y} \in \partial \mathcal{S}} \|\boldsymbol{x} - \boldsymbol{y}\|, & \text{if } \boldsymbol{x} \in \mathcal{S} \\ -\inf\limits_{\boldsymbol{y} \in \partial \mathcal{S}} \|\boldsymbol{x} - \boldsymbol{y}\|, & \text{if } \boldsymbol{x} \notin \mathcal{S}\,. \end{cases} \tag{13}$$

where $\partial\mathcal{S}$ is the boundary of $\mathcal{S}$. Multiplication of the Fourier layer weights by $\tilde{\chi}$ achieves two goals. First, it ensures that the indicator approximation in (12) is continuous and thus retains the guarantees of Theorem 1. Second, for each environment, PNO ignores the unsafe space in the weight matrix multiplication, which greatly improves its ability to generalize to new obstacle configurations.

Computing the SDF in the smoothing function (12) itself can be computationally challenging, especially in large motion planning problems. However, note a key observation that *a SDF $d_{\mathcal{S}}(\boldsymbol{x})$ is itself a solution to an Eikonal PDE*:

$$\|\nabla d_{\mathcal{S}}(\boldsymbol{x})\| = 1, \quad \boldsymbol{x} \in \mathcal{X}, \qquad d_{\mathcal{S}}(\boldsymbol{x}) = 0, \quad \boldsymbol{x} \in \partial\mathcal{S}\,. \tag{14}$$

Thus, under Theorem 1, there exists a neural operator approximation to (14). As such, we introduce a second neural operator, in the form of an FNO, trained independently, that maps from a binary occupancy map to the corresponding SDF, namely $1/c(\boldsymbol{x}) \mapsto d_{\mathcal{S}}(\boldsymbol{x})$. Thus, the occupancy function $1/c(\boldsymbol{x})$ and the SDF $d_{\mathcal{S}}(\boldsymbol{x})$ are taken as inputs to generate the smoothed indicator $\tilde{\chi}$ in (12), which in turn is used to modulate the kernel in the Fourier layers of PNO. Since we use an FNO to generate the SDF function, our entire architecture maintains property (i), namely resolution invariance.

3.3 PNO ARCHITECTURE: GENERALIZING ACROSS GOAL POSITIONS

Given the lifting layer and the modified Fourier layers discussed above, we move to design the projection layer of PNO. The projection layer contains two inputs, namely the output from the last modified Fourier layer and the goal location $\boldsymbol{g}$. This goal location is equivalent to passing the boundary condition (7b) for the solution operator Ψ into our network. As aforementioned, to enable goal generalization (iii), we exploit the fact that an optimal value function for a motion planning problem must satisfy the triangle inequality. Thus, we parameterize our output with an inductively biased Deepnorm layer $Q(\cdot,\cdot)$ (Pitis et al., 2020) given by:

$$Q(L_M(\boldsymbol{x}), L_M(\boldsymbol{g})) = f_\theta(L_M(\boldsymbol{x}) - L_M(\boldsymbol{g}))\,, \tag{15}$$

where $L_M(\boldsymbol{x})$, $L_M(\boldsymbol{g})$ are the outputs from last modified Fourier layer at location $\boldsymbol{x}$ and goal $\boldsymbol{g}$, f_θ consists of regular neural network layers with a *non-negative* activation function (e.g., ReLU) and a *positive* weight matrix W^+. This ensures that our value function satisfies the triangle inequality with respect to goal position $\boldsymbol{g}$ (Pitis et al., 2020) and greatly improves the generalization performance across different goal positions, achieving property (iii). The entire PNO architecture is summarized in Fig. 2, starting from a binary occupancy input $1/c(\boldsymbol{x})$ and providing a value function as an output.

4 LEARNING MOTION PLANNING VALUE FUNCTIONS

To validate the efficacy of our PNO architecture, we design three learning experiments. First, we test our method on grid-world environments where the operator is compared against learning-based motion planners. Although, many planners have been successful on the small grid world dataset, the environments lack the complexity of real-world tasks. Thus, we introduce two other experiments - both on *real world* datasets. In the first experiment, we highlight the super-resolution properties of PNO by training on small synthetic maps and evaluating on large real-world maps from the Moving AI 2D city maps dataset (Sturtevant, 2012). In the second experiment, we showcase the scalability of our approach on 3D environments from the iGibson dataset (Shen et al., 2021).

We compare our method against the fast marching method (FMM) (Sethian, 1996) which solves the Eikonal PDE numerically, state-of-the-art neural motion planners: VIN (Tamar et al., 2016), NTFields (Ni & Qureshi, 2023), IEF2D (Li et al., 2022), and two operator learning architectures: FNO (Li et al., 2021), DAFNO (Liu et al., 2023). We review the baselines and provide additional results in Appendix D. All code is available on our Github repository (available post review period).

Grid-world environments. In the first experiment, we consider a small Grid-World dataset as in Tamar et al. (2016), consisting of 5k training maps and 1k testing maps. We compare with VIN and IEF2D. For planning, we perform gradient descent on the test-map value function predictions from VIN, IEF2D, and PNO. In Table 1, we present the computation time needed for value function prediction and the success rate of reaching the goal. We can see that all baselines perform quite well, with PNO outperforming both methods on the larger 28×28 sized mazes while achieving an improved computational time. In Appendix D, we give an example of the learned value function and corresponding map for the PNO. The superior performance of PNO on the Grid World dataset is primarily due to the fact PNO satisfies the triangle inequality and guarantees obstacle avoidance where as both VIN and IEF2D lack such properties.

Table 1: Comparison of planning on learned value functions on the Grid World dataset at various sizes. The success rate and times are averaged over 1k maps.

	Avg. success rate $\uparrow$			**Avg. computation time (ms) $\downarrow$**		
	8^2	16^2	28^2	8^2	16^2	28^2
VIN	99.6	**99.3**	96.7	3.9	22.7	82.1
IEF2D	99.7	98.3	97.0	13.3	14.8	20.4
PNO (ours)	**99.9**	97.3	**99.3**	**2.6**	**4.7**	**6.4**

Moving AI 2D city maps. To highlight the advantage of reformulating motion planning as an operator learning problem in continuous function space, we demonstrate that our PNO architecture can be trained with a coarse resolution and deployed at a finer resolution in the Moving AI 2D real-world city maps (see Fig. 1). We train our architecture on a synthetic map dataset generated using a random map generator at 64×64 resolution (see Appendix D.3). We, then, evaluate the super-resolution performance of our method in comparison with two operator learning architectures in Table 2. The importance of encoding the obstacle geometry explicitly can be seen in that both DAFNO and PNO clearly outperform FNO on the synthetic and real-world datasets. Furthermore, we see the effect of the inductive-bias projection layer (ensuring satisfaction of the triangle inequality) over DAFNO - particularly in the unseen real-world city dataset achieving a 50% reduction in L_2 error. For the city dataset, PNO is the only method to produce value functions that are viable for planning as the errors in the other operator learning methods create large local minima. In the third column of Table 2, we see the computational speedup at scale of the PNO architecture. The SDF generation significantly slows DAFNO and we achieve almost $10\times$ speedup over the fast marching method (FMM). Lastly, we provide an example of navigating in a New York City map in Fig. 1. In Fig. 1, we see that the FNO is able to learn the SDF accurately (achieving a relative L_2 test error of 0.064) and that the corresponding value function is accurately computed at all three super-resolution scales.

Table 2: Average L_2 value function error of FNO, DAFNO, and PNO versus FMM. All models were trained on the synthetic obstcale dataset at 64×64 resolution.

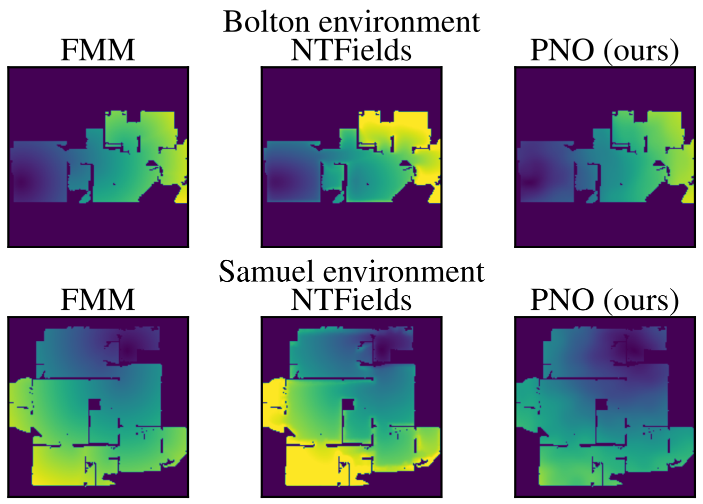
	Avg. relative L_2 error ↓								**Avg. computation time ↓**			
Map size	64^2	256^2	512^2	1024^2	64^2	256^2	512^2	1024^2	64^2	256^2	512^2	1024^2
	Synthetic obstacle dataset (100 maps, in-distribution)				MovingAI real-world city dataset (90 maps, out-of-distribution)				Synthetic + real-world city (1000 maps, ms)			
FNO	0.1996	0.5771	0.6214	0.6405	—	0.7188	0.7519	0.7692	1.62	**1.64**	**1.66**	**2.28**
DAFNO	**0.0985**	0.3868	0.4060	0.4120	—	0.4090	0.4259	0.4315	3.10	4.97	11.44	49.82
PNO (ours)	0.1136	**0.1197**	**0.1190**	**0.1194**	—	**0.1748**	**0.1885**	**0.2034**	5.57	5.31	6.33	8.31
Numerical solver (FMM)	—	—	—	—	—	—	—	—	**0.34**	5.96	25.66	104.62

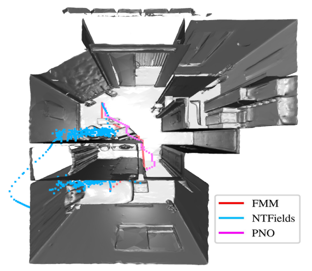
3D planning in iGibson environments. Finally, we scale the planning problem to 3D real-world iGibson environments (details in Appendix D.4). For comparison, we independently train NTFields models on the Bolton and Samuel environments. Each NTFields model takes 6 hours to train *per environment* (NVIDIA $A100$ GPU). To validate the generalization of our PNO, we ensure that our training dataset contains no instances of the Samuel environment, but does contain instances of the Bolton environment with different start goal positions than during testing. Generating our dataset takes approximately 1 hour and the model training takes 4 hours (NVIDIA $A100$ GPU).

Fig. 3 shows predicted value functions sliced at $z = 0$, where all three methods capture the general structure of the value function. However, both NTFields and PNO struggle when far from the target leaving opportunities for future work. On the top right, we showcase an example of planning. NTFields, due to the lack of obstacle encoding, is unable to guarantee valid paths whereas both PNO and FMM reach the goal successfully. Lastly, the table of Fig. 3, we see that quantitatively that our method performs well compared to NTFields but may still have room for improvement especially on the unseen Samuel environment. Excitingly, it is clear that both neural solvers perform faster then the numerical solver achieving speedups of $2\times$ (NTFields) and $3\times$ (PNO) over FMM.

5 Learned operator value functions as neural heuristics

Although one can plan on the learned value functions directly, they may exhibit local minima due to approximation error. To address this challenge, we capitalize on the rich set of classical planning algorithms by employing our learned value function as a heuristic to guide the planning algorithm exploration. Typically, motion planners use the Euclidean norm as a heuristic given that it is both fast to compute and guarantees an optimal path (1-consistent). However, the Euclidean norm does not capture the geometry of the obstacles and thus regions surrounding obstacles can require extensive exploration before an optimal path is found. Thus, we propose using the learned value functions from PNO as an alternative heuristic to the Euclidean norm. We first prove that our heuristic is





	Average relative L_2 error ↓	Average computation time (100 inst, ms) ↓
	Bolton Environment	
PNO (ours)	0.0877	33.60
NTFields	0.2529	56.07
FMM	—	101.05
	Samuel Environment	
PNO (ours)	0.2363	31.67
NTFields	0.3119	58.04
FMM	—	180.07

Figure 3: (left) Comparison of 3D value function approximations for two iGibson environments (sliced at $z = 0$). (right) On top, we show an example path in the Bolton environment. On the bottom, the value function approximations are compared quantitatively. The computation time only includes the forward inference time of the model - no training time.

$\epsilon-$consistent and then showcase the advantage numerically, achieving a 33% decrease in the number of explored nodes compared to the Euclidean norm when used in the A^* algorithm.

We begin by showing that PNO value function is an ϵ-consistent heuristic, which is an important property for the A^* algorithm to compute an ϵ-optimal path (Likhachev et al., 2003).

Definition 1. *Let $V(\boldsymbol{x}, \boldsymbol{g})$ be the value function with goal position $\boldsymbol{g}$. A heuristic function $h(\boldsymbol{x}) : \mathcal{X} \to \mathbb{R}$ is said to* ***admissible*** *if $h(\boldsymbol{x}) \leq V(\boldsymbol{x}, \boldsymbol{g})$ for any $\boldsymbol{x} \in \mathcal{X}$. It is said to be* ***consistent*** *if $h(\boldsymbol{x}) \leq V(\boldsymbol{x}, \boldsymbol{y}) + h(\boldsymbol{y})$. Furthermore, for any $\epsilon > 1$, a heuristic is* ***ϵ-consistent*** *if $h(\boldsymbol{x}) \leq \epsilon V(\boldsymbol{x}, \boldsymbol{y}) + h(\boldsymbol{y})$.*

Lemma 1 (ϵ-consistency of neural heuristic)**.** *Let Assumption 1 hold and let $\mathcal{F}_c$ be the cost function space satisfying Assumption 2. Let ϵ_{NO} be the neural operator approximation error as in (9). Then, $\hat{V}(\boldsymbol{x})$, generated from the neural operator is an ϵ-consistent heuristic with:*

$$\epsilon = \max_{\{\boldsymbol{x}, \boldsymbol{y} \in \mathcal{S} | \boldsymbol{x} \neq \boldsymbol{y}\}} 1 + 2\epsilon_{NO}/V(\boldsymbol{x}, \boldsymbol{y}) \,. \tag{16}$$

Further, if one is interested in a neural operator satisfying a specific $\epsilon-$consistency for $\epsilon > 1$, then the neural operator must have error no worse than:

$$\epsilon_{NO} \leq \min_{\{\boldsymbol{x}, \boldsymbol{y} \in \mathcal{S} | \boldsymbol{x} \neq \boldsymbol{y}\}} (\epsilon - 1)/V(\boldsymbol{x}, \boldsymbol{y}) \,. \tag{17}$$

In continuous space, it is likely that $\epsilon \to \infty$ in Lemma 1. However, motion planning algorithms, such as A^*, discretize the space yielding $\epsilon < \infty$. Further, in Lemma 1, it is impossible to identify the true approximation error ϵ_{NO}. Thus, we introduce a second, practical improvement. Our key idea is to "erode" the obstacles (as in Fig. 4) with the intuition the neural operator applied to the eroded environment is more likely to be an under-approximation to the true value function. This encourages *admissibility* which can improve the paths generated under the heuristic (Pearl, 1984). To conduct the erosion, we remove the outermost layer from each obstacle (Serra, 1983) and repeat this operation several times depending on the environment size. Formally, this can be expressed as increasing the safe space as $\tilde{\mathcal{S}} \supset \mathcal{S}$, where $\tilde{\mathcal{S}}$ is the safe set after erosion. The eroded cost function then becomes

$$\tilde{c}(\boldsymbol{x}) = \begin{cases} 1 & \boldsymbol{x} \in \tilde{\mathcal{S}}, \\ \infty & \boldsymbol{x} \in \mathcal{X} \setminus \tilde{\mathcal{S}}. \end{cases} \tag{18}$$

The following result guarantees that our eroded PNO heuristic does not harm the ϵ-consistency.

Lemma 2 (Eroded heuristic is "more" consistent)**.** *Let Assumptions 1, 2 hold. For any $\epsilon_{NO} > 0$, let $\hat{\Psi}(c), \hat{\Psi}(\tilde{c})$ be the learned solutions to (7) satisfying Theorem 1 with ϵ_{NO} according to costs $c, \tilde{c}$ defined in (11), (18) respectively. Let ϵ and $\tilde{\epsilon}$ be the $\epsilon-$consistent value functions generated by operators $\hat{\Psi}(c)$ and $\hat{\Psi}(\tilde{c})$ with ϵ_{NO} error as in Lemma 1. Then, $\epsilon \geq \tilde{\epsilon}$.*

Lastly, to ensure we do not significantly underestimate the value function, we combine our heuristic with the Euclidean norm via $h(\boldsymbol{x}) = \max\{\|\boldsymbol{x} - \boldsymbol{g}\|, \hat{\Psi}(\tilde{c}(\boldsymbol{x}))\}$ while preserving ϵ-consistency.

We evaluate the impact of our PNO heuristic for motion planning on 1024×1024 Moving AI lab city maps. For planning, we use the A^* algorithm with our PNO estimated value function as the heuristic. In Table 3, we see that the PNO heuristic with erosion achieves nearly optimal paths while expanding 33% fewer nodes compared to the Euclidean norm heuristic. Furthermore, the value of the eroded heuristic is clear as without erosion, our PNO heuristic generates sub-optimal paths. The effect of erosion is further analyzed in Appendix E. Lastly, in Fig. 4, we highlight an example of planning where the PNO heuristic yields a similar path as the Euclidean norm despite expanding fewer nodes.

Table 3: Comparison of heuristics in the A^* algorithm, RRT and RRT* over 2D maps (256^2 implies 256×256). The computational time includes both the time for heuristic generation as well as the time for planning. The number of layers eroded is 12, 14, and 18 for 256^2 , 512^2, and 1024^2, respectively.

	Avg. path length ↓			**ϵ suboptimality estimate ↓**			**Avg. number of nodes expanded ↓**			**Avg. computational time (50 inst., s) ↓**		
Map size (2D)	256^2	512^2	1024^2	256^2	512^2	1024^2	256^2	512^2	1024^2	256^2	512^2	1024^2
A* - Euclidean norm	**144.05**	**311.21**	**605.14**	**1.000**	**1.000**	**1.000**	3310	14850	59965	0.186	0.868	3.606
A* - PNO (ours) w Erosion	144.05	311.72	608.54	**1.000**	**1.000**	**1.000**	2024	9869	41394	0.136	0.684	2.759
A* - PNO (ours) w/o Erosion	144.88	315.25	614.40	1.005	1.013	1.015	**1757**	**8882**	**39438**	0.121	0.587	2.703
RRT	191.98	418.61	800.31	1.333	1.345	1.322	-	-	-	**0.014**	**0.045**	**0.066**
RRT*	177.29	403.56	783.83	1.231	1.297	1.295	-	-	-	0.120	0.600	2.71

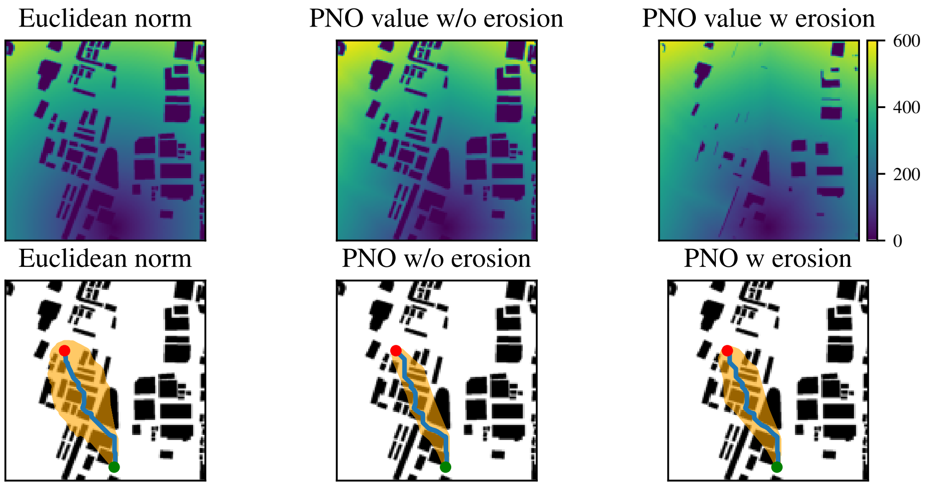


Figure 4: Path planning using various heuristics on the Moving AI Shanghai city map. The top showcases the heuristic generated and the bottom visualizes the A^* planning under the corresponding heuristic. The nodes expanded are in orange, the start in red, the goal in green, and the path in blue.

6 Conclusions, Limitations, and Future Work

We reformulated the motion planning problem as an Eikonal PDE and focused on learning its solution operator. To learn the operator, we developed the Planning Neural Operator (PNO) which is (i) resolution invariant, (ii) does not require retraining in new environments, and (iii) generalizes across goal positions. We evaluated our architecture on the $2D$ MovingAI city dataset and the $3D$ iGibson building dataset, showcasing super-resolution value function prediction while achieving speedups of $10\times$ over a numerical PDE solver. Lastly, we proved that our value function predictions can be used as ϵ-consistent heuristics for motion planning algorithms, and demonstrated a 33% decrease in expanded nodes in the A^* algorithm compared to planning with a Euclidean norm heuristic.

In the future, we hope to address some limitations of PNO. For example, our heuristic is computed once without updates during planning. An interesting question is whether we can introduce dynamic heuristic evaluation as PNO exhibits fast forward inference time. In addition, we plan to validate PNO on real robot motion plannig problems. To do so, we aim to extend the PNO formulation to include the physical constraints of the robot dynamics. Lastly, we considered cost functions with uniform step penalties. An exciting extension would be to train PNO in environments with non-uniform costs.

References

Michal Adamkiewicz, Timothy Chen, Adam Caccavale, Rachel Gardner, Preston Culbertson, Jeannette Bohg, and Mac Schwager. Vision-only robot navigation in a neural radiance world. *IEEE Robotics and Automation Letters*, 7(2):4606–4613, 2022. doi: 10.1109/LRA.2022.3150497.

Richard Bellman. *Dynamic programming*. Princeton University Press, 1957.

Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control: Vol. I*. Athena Scientific, 2017.

Luke Bhan, Yuanyuan Shi, and Miroslav Krstic. Neural operators for bypassing gain and control computations in PDE backstepping. *IEEE Transactions on Automatic Control*, pp. 1–16, 2023. doi: 10.1109/TAC.2023.3347499. https://ieeexplore.ieee.org/document/10374221.

Umair bin Waheed, Ehsan Haghighat, Tariq Alkhalifah, Chao Song, and Qi Hao. PINNeik: Eikonal solution using physics-informed neural networks. *Computers & Geosciences*, 155:104833, 2021. ISSN 0098-3004. doi: https://doi.org/10.1016/j.cageo.2021.104833. URL https://www.sciencedirect.com/science/article/pii/S009830042100131X.

Constantinos Chamzas, Zachary Kingston, Carlos Quintero-Peña, Anshumali Shrivastava, and Lydia E. Kavraki. Learning sampling distributions using local 3D workspace decompositions for motion planning in high dimensions. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1283–1289. IEEE Press, 2021. doi: 10.1109/ICRA48506.2021.9561104. URL https://doi.org/10.1109/ICRA48506.2021.9561104.

Binghong Chen, Bo Dai, Qinjie Lin, Guo Ye, Han Liu, and Le Song. Learning to plan in high dimensions via neural exploration-exploitation trees. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rJgJDAVKvB.

Tianping Chen and Hong Chen. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. *IEEE Transactions on Neural Networks*, 6(4):911–917, 1995. doi: 10.1109/72.392253.

Benjamin J. Cohen, Sachin Chitta, and Maxim Likhachev. Search-based planning for manipulation with motion primitives. In *2010 IEEE International Conference on Robotics and Automation*, pp. 2902–2908, 2010. doi: 10.1109/ROBOT.2010.5509685.

Beichuan Deng, Yeonjong Shin, Lu Lu, Zhongqiang Zhang, and George Em Karniadakis. Approximation rates of DeepONets for learning operators arising from advection–diffusion equations. *Neural Networks*, 153:411–426, 2022. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2022.06.019. URL https://www.sciencedirect.com/science/article/pii/S0893608022002349.

Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1 (1):269–271, 1959.

Lawrence C. Evans. *Partial differential equations*. American Mathematical Society, 2010.

Zhiwei Fang, Sifan Wang, and Paris Perdikaris. Learning only on boundaries: A physics-informed neural operator for solving parametric partial differential equations in complex geometries. *Neural Computation*, 36(3):475–498, 02 2024. ISSN 0899-7667. doi: 10.1162/neco_a_01647. URL https://doi.org/10.1162/neco_a_01647.

Adam Fishman, Adithyavairavan Murali, Clemens Eppner, Bryan Peele, Byron Boots, and Dieter Fox. Motion policy networks. In *6th Annual Conference on Robot Learning*, 2022. URL https://openreview.net/forum?id=aQnn9cIVTRJ.

Gerald B. Folland. *Real analysis : modern techniques and their applications*. Wiley, 1999.

Takashi Furuya, Michael Anthony Puthawala, Matti Lassas, and Maarten V. de Hoop. Globally injective and bijective neural operators. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=6cc69ArD3O.

Jonathan D. Gammell, Siddhartha S. Srinivasa, and Timothy D. Barfoot. Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2997–3004, 2014. doi: 10.1109/IROS.2014.6942976.

Jonathan D. Gammell, Siddhartha S. Srinivasa, and Timothy D. Barfoot. Batch informed trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, May 2015. doi: 10.1109/icra.2015.7139620. URL http://dx.doi.org/10.1109/ICRA.2015.7139620.

Gaurav Gupta, Xiongye Xiao, and Paul Bogdan. Multiwavelet-based operator learning for differential equations. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=LZDiWaC9CGL.

Zhongkai Hao, Zhengyi Wang, Hang Su, Chengyang Ying, Yinpeng Dong, Songming Liu, Ze Cheng, Jian Song, and Jun Zhu. GNOT: A general neural operator transformer for operator learning. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 12556–12569. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/hao23c.html.

Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968. doi: 10.1109/TSSC.1968.300136.

Brian Ichter, James Harrison, and Marco Pavone. Learning sampling distributions for robot motion planning, 2019. URL https://arxiv.org/abs/1709.05448.

Lucas Janson, Edward Schmerling, Ashley Clark, and Marco Pavone. Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions. *The International Journal of Robotics Research*, 34(7):883–921, 2015. doi: 10.1177/0278364915577958. URL https://doi.org/10.1177/0278364915577958. PMID: 27003958.

Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011. doi: 10.1177/0278364911406761. URL https://doi.org/10.1177/0278364911406761.

Nikos Katzourakis. *An Introduction To Viscosity Solutions for Fully Nonlinear PDE with Applications to Calculus of Variations in L_∞*. Springer Cham, 2014.

Lydia E. Kavraki, Petr Svestka, Jean-Claude Latombe, and Mark H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996. doi: 10.1109/70.508439.

Nikola Kovachki, Samuel Lanthaler, and Siddhartha Mishra. On universal approximation and error bounds for Fourier neural operators. *J. Mach. Learn. Res.*, 22(1), jan 2021. ISSN 1532-4435.

Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: learning maps between function spaces with applications to PDEs. *Journal of Machine Learning Research*, 24(89):1–97, 2023. URL http://jmlr.org/papers/v24/21-1524.html.

Thorsten Kurth, Shashank Subramanian, Peter Harrington, Jaideep Pathak, Morteza Mardani, David Hall, Andrea Miele, Karthik Kashinath, and Anima Anandkumar. FourCastNet: Accelerating global high-resolution weather forecasting using adaptive Fourier neural operators. In *Proceedings of the Platform for Advanced Scientific Computing Conference*, PASC '23, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400701900. doi: 10.1145/3592979.3593412. URL https://doi.org/10.1145/3592979.3593412.

Samuel Lanthaler, Siddhartha Mishra, and George E Karniadakis. Error estimates for DeepONets: a deep learning framework in infinite dimensions. *Transactions of Mathematics and Its Applications*, 6(1):tnac001, 03 2022. ISSN 2398-4945. doi: 10.1093/imatrm/tnac001. URL https://doi.org/10.1093/imatrm/tnac001.

Samuel Lanthaler, Zongyi Li, and Andrew M. Stuart. The nonlocal neural operator: Universal approximation, 2023. URL https://arxiv.org/abs/2304.13221.

Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, USA, 2006. ISBN 0521862051.

Lucas Lehnert, Sainbayar Sukhbaatar, Paul McVay, Michael Rabbat, and Yuandong Tian. Beyond A*: Better LLM planning via search dynamics bootstrapping. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*, 2024. URL https://openreview.net/forum?id=rviGTsl0oy.

Tingguang Li, Danny Ho, Chenming Li, Delong Zhu, Chaoqun Wang, and Max Q.-H. Meng. Houseexpo: A large-scale 2D indoor layout dataset for learning-based algorithms on mobile robots. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5839–5846, 2020. doi: 10.1109/IROS45743.2020.9341284.

Xueting Li, Sifei Liu, Shalini De Mello, Xiaolong Wang, Ming-Hsuan Yang, and Jan Kautz. Learning continuous environment fields via implicit functions. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=3ILxkQ7yElm.

Zongyi Li, Nikola Borislavov Kovachki, Kamyar Azizzadenesheli, Burigede liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=c8P9NQVtmnO.

Zongyi Li, Daniel Zhengyu Huang, Burigede Liu, and Anima Anandkumar. Fourier neural operator with learned deformations for PDEs on general geometries. *Journal of Machine Learning Research*, 24(388):1–26, 2023a. URL http://jmlr.org/papers/v24/23-0064.html.

Zongyi Li, Nikola Kovachki, Chris Choy, Boyi Li, Jean Kossaifi, Shourya Otta, Mohammad Amin Nabian, Maximilian Stadler, Christian Hundt, Kamyar Azizzadenesheli, and Animashree Anandkumar. Geometry-informed neural operator for large-scale 3D PDEs. In A. Oh, T. Neumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 35836–35854. Curran Associates, Inc., 2023b. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/70518ea42831f02afc3a2828993935ad-Paper-Conference.pdf.

Daniel Liberzon. *Calculus of Variations and Optimal Control Theory: A Concise Introduction*. Princeton University Press, 2012.

Maxim Likhachev, Geoffrey J Gordon, and Sebastian Thrun. ARA$*$: Anytime A$*$ with provable bounds on sub-optimality. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2003.

Ning Liu, Siavash Jafarzadeh, and Yue Yu. Domain agnostic Fourier neural operators. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=ubap5FKbJs.

Puze Liu, Kuo Zhang, Davide Tateo, Snehal Jauhri, Jan Peters, and Georgia Chalvatzaki. Regularized deep signed distance fields for reactive motion generation. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6673–6680, 2022. doi: 10.1109/IROS47612.2022.9981456.

Sikang Liu, Kartik Mohta, Nikolay Atanasov, and Vijay Kumar. Search-based motion planning for aggressive flight in SE(3). *IEEE Robotics and Automation Letters*, 3(3):2439–2446, 2018. doi: 10.1109/LRA.2018.2795654.

Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, 2021. ISSN 2522-5839. doi: 10.1038/s42256-021-00302-5. URL https://doi.org/10.1038/s42256-021-00302-5.

Troy McMahon, Aravind Sivaramakrishnan, Edgar Granados, and Kostas E. Bekris. A survey on the integration of machine learning with sampling-based motion planning. *Foundations and Trends® in Robotics*, 9(4):266–327, 2022. ISSN 1935-8261. doi: 10.1561/2300000063. URL http://dx.doi.org/10.1561/2300000063.

Ruiqi Ni and Ahmed H Qureshi. NTFields: Neural time fields for physics-informed robot motion planning. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=ApF0dmi1_9K.

Andreas Orthey, Constantinos Chamzas, and Lydia E. Kavraki. Sampling-based motion planning: A comparative review. *Annual Review of Control, Robotics, and Autonomous Systems*, 7(Volume 7, 2024):285–310, 2024. ISSN 2573-5144. doi: https://doi.org/10.1146/annurev-control-061623-094742. URL https://www.annualreviews.org/content/journals/10.1146/annurev-control-061623-094742.

Joseph Ortiz, Alexander Clegg, Jing Dong, Edgar Sucar, David Novotny, Michael Zollhoefer, and Mustafa Mukadam. iSDF: Real-time neural signed distance fields for robot perception. In *Robotics: Science and Systems*, 2022.

Judea Pearl. *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley Longman Publishing Co., Inc., USA, 1984. ISBN 0201055945.

Silviu Pitis, Harris Chan, Kiarash Jamali, and Jimmy Ba. An inductive bias for distances: Neural nets that respect the triangle inequality. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HJeiDpVFPr.

Clément Pêtrès, Yan Pailhas, Yvan Petillo, and Dave Lane. Underwater path planing using fast marching algorithms. In *Europe Oceans 2005*, volume 2, pp. 814–819 Vol. 2, 2005. doi: 10.1109/OCEANSE.2005.1513161.

Ahmed Hussain Qureshi and Yasar Ayaz. Potential functions based sampling heuristic for optimal path planning. *Autonomous Robots*, 40(6):1079–1093, November 2015. ISSN 1573-7527. doi: 10.1007/s10514-015-9518-0. URL http://dx.doi.org/10.1007/s10514-015-9518-0.

Ahmed Hussain Qureshi, Yinglong Miao, Anthony Simeonov, and Michael C. Yip. Motion planning networks: Bridging the gap between learning-based and classical motion planners. *IEEE Transactions on Robotics*, 37(1):48–66, 2021. doi: 10.1109/TRO.2020.3006716.

Jacob H Seidman, Georgios Kissas, Paris Perdikaris, and George J. Pappas. NOMAD: Nonlinear manifold decoders for operator learning. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=5OWV-sZvMl.

Jean Serra. *Image Analysis and Mathematical Morphology*. Academic Press, Inc., USA, 1983. ISBN 0126372403.

James A. Sethian. A fast marching level set method for monotonically advancing fronts. *Proceedings of the National Academy of Sciences of the United States of America*, 93(4):1591–1595, 1996. doi: 10.1073/pnas.93.4.1591.

Bokui Shen, Fei Xia, Chengshu Li, Roberto Martín-Martín, Linxi Fan, Guanzhi Wang, Claudia Pérez-D'Arpino, Shyamal Buch, Sanjana Srivastava, Lyne P. Tchapmi, Micael E. Tchapmi, Kent Vainio, Josiah Wong, Li Fei-Fei, and Silvio Savarese. igibson 1.0: a simulation environment for interactive tasks in large realistic scenes. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. accepted. IEEE, 2021.

Nathan R. Sturtevant. Benchmarks for grid-based pathfinding. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(2):144–148, 2012. doi: 10.1109/TCIAIG.2012.2197681.

Balakumar Sundaralingam, Siva Kumar Sastry Hari, Adam Fishman, Caelan Garrett, Karl Van Wyk, Valts Blukis, Alexander Millane, Helen Oleynikova, Ankur Handa, Fabio Ramos, Nathan Ratliff, and Dieter Fox. cuRobo: Parallelized collision-free minimum-jerk robot motion generation, 2023. URL https://arxiv.org/abs/2310.17274.

Aviv Tamar, YI WU, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value iteration networks. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper_files/paper/2016/file/c21002f464c5fc5bee3b98ced83963b8-Paper.pdf.

Alberto Valero-Gomez, Javier V. Gomez, Santiago Garrido, and Luis Moreno. The path to efficiency: Fast marching method for safer, more efficient mobile robot trajectories. *IEEE Robotics & Automation Magazine*, 20(4):111–120, 2013. doi: 10.1109/MRA.2013.2248309.

Jiankun Wang, Wenzheng Chi, Chenming Li, Chaoqun Wang, and Max Q.-H. Meng. Neural RRT*: Learning-based optimal path planning. *IEEE Transactions on Automation Science and Engineering*, 17(4):1748–1758, 2020. doi: 10.1109/TASE.2020.2976560.

Yan Yang, Angela F. Gao, Jorge C. Castellanos, Zachary E. Ross, Kamyar Azizzadenesheli, and Robert W. Clayton. Seismic wave propagation and inversion with neural operators. *The Seismic Record*, 1(3):126–134, 11 2021. ISSN 2694-4006. doi: 10.1785/0320210026. URL https://doi.org/10.1785/0320210026.

Huaiqian You, Yue Yu, Marta D'Elia, Tian Gao, and Stewart Silling. Nonlocal kernel network (NKN): A stable and resolution-independent deep neural network. *Journal of Computational Physics*, 469: 111536, 2022a. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2022.111536. URL https://www.sciencedirect.com/science/article/pii/S0021999122005988.

Huaiqian You, Quinn Zhang, Colton J. Ross, Chung-Hao Lee, and Yue Yu. Learning deep implicit fourier neural operators (IFNOs) with applications to heterogeneous material modeling. *Computer Methods in Applied Mechanics and Engineering*, 398:115296, 2022b. ISSN 0045-7825. doi: https://doi.org/10.1016/j.cma.2022.115296. URL https://www.sciencedirect.com/science/article/pii/S0045782522004078.

Wenyuan Zeng, Wenjie Luo, Simon Suo, Abbas Sadat, Bin Yang, Sergio Casas, and Raquel Urtasun. End-to-end interpretable neural motion planner. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8652–8661, Los Alamitos, CA, USA, jun 2019. IEEE Computer Society. doi: 10.1109/CVPR.2019.00886. URL https://doi.ieeecomputersociety.org/10.1109/CVPR.2019.00886.

Ruipeng Zhang, Chenning Yu, Jingkai Chen, Chuchu Fan, and Sicun Gao. Learning-based motion planning in dynamic environments using GNNs and temporal encoding. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=gQBetxnU4Lk.

Appendix

A Proofs of Lemmas and Theorems

A.1 Proof of Theorem 1

Proof. We show Ψ is a continuous operator in L^∞ and then invoke Theorem 2 noting that $\mathcal{F}_c$ is compact by Assumptions 2 and the Arzelá Ascoli theorem. Since, Ψ is the viscosity solution, it satisfies the comparison principle (Katzourakis, 2014, Theorem 1). Thus, let $c_1, c_2 \in \mathcal{F}_c$ such that $c_1 \leq c_2$. Then, via the comparison principle, $\Psi(c_1) \leq \Psi(c_2)$. Furthermore, by the Archimedean property, we can find a real number $\lambda > 0$ such that $\lambda c_1 \geq c_2$. A simple choice for such a λ is $\lambda = \sup_{\boldsymbol{x} \in \mathcal{X}} (\frac{c_2}{c_1})(\boldsymbol{x})$. Then we have that $\lambda \Psi(c_1) \geq \Psi(c_2)$ again by the comparison principle. Thus, we can say that

$$\begin{aligned}\|\Psi(c_2)-\Psi(c_1)\|_{L_\infty} &\le \|(\lambda-1)\Psi(c_1)\|_{L^\infty} \\ &\le \left(\sup_{\boldsymbol{x}\in\mathcal{X}}\left(\frac{c_2-c_1}{c_1}\right)(\boldsymbol{x})\right)\|\Psi(c_1)\|_{L^\infty} \\ &\le \theta_v\theta_u\|c_2-c_1\|_{L^\infty}\,, \end{aligned} \tag{19}$$

where we used substitution, Cauchy Schwartz, and then substitution again to obtain the bound. Thus Ψ, $\mathcal{F}_c$ satisfy the requirements of Theorem 2 completing the result. □

A.2 Proof of Lemma 1

Proof. Let $h^*(\boldsymbol{x}) = \Psi(c)(\boldsymbol{x})$ be the optimal value function. The optimal value function is consistent, namely for any $\boldsymbol{x},\boldsymbol{y}\in\mathcal{X}$, we have $h^*(\boldsymbol{x})-h^*(\boldsymbol{y})\le V^*(\boldsymbol{x},\boldsymbol{y})$. Now, for any ϵ_{NO}, there exists some $\hat{\Psi}(c)$ satisfying Theorem 1 such that $\|\Psi(c)-\hat{\Psi}(c)\|<\epsilon_{NO}$ for any $\boldsymbol{x},\boldsymbol{y}\in\mathcal{X}$. Then, we have

$$\begin{aligned} h(\boldsymbol{x})-h(\boldsymbol{y}) &\le h^*(\boldsymbol{x})-h^*(\boldsymbol{y})+2\epsilon_{NO} \\ &\le V^*(\boldsymbol{x},\boldsymbol{y})+2\epsilon_{NO}\,. \end{aligned} \tag{20}$$

To ensure, that (20) is $\le\ \epsilon V^*(\boldsymbol{x},\boldsymbol{y})$ for every $\boldsymbol{x},\boldsymbol{y}$, we can choose $\epsilon_{NO}\ \le \min_{\{\boldsymbol{x},\boldsymbol{y}\in\mathcal{X}|\boldsymbol{x}\neq\boldsymbol{y}\}}\dfrac{(\epsilon-1)V^*(\boldsymbol{x},\boldsymbol{y})}{2}$. Likewise, the smallest ϵ achieved is $\epsilon\ \ge \max_{\{\boldsymbol{x},\boldsymbol{y}\in\mathcal{X}|\boldsymbol{x}\neq\boldsymbol{y}\}} 1+\dfrac{2\epsilon_{NO}}{V^*(\boldsymbol{x},\boldsymbol{y})}$. □

A.3 Proof of Lemma 2

Proof. Let $V^*(\boldsymbol{x},\boldsymbol{y})$ and $\tilde{V}^*(\boldsymbol{x},\boldsymbol{y})$ be the optimal value functions for $c,\tilde{c}$ respectively. Then, $V^*\ \ge\ \tilde{V}^*$ by definition of c, $\tilde{c}$. Let $\epsilon(c),\epsilon(\tilde{c})$ be the minimum ϵ_2 satisfying Lemma 1, (20) for $c,\tilde{c}$ respectively. Explicitly

$$\epsilon(c) = \max_{\{\boldsymbol{x},\boldsymbol{y}\in\mathcal{X}|\boldsymbol{x}\neq\boldsymbol{y}\}}\frac{V^*(x,y)+2\epsilon_{NO}}{V^*(x,y)}\,, \tag{21}$$

$$\epsilon(\tilde{c}) = \max_{\{\boldsymbol{x},\boldsymbol{y}\in\mathcal{X}|\boldsymbol{x}\neq\boldsymbol{y}\}}\frac{\tilde{V}^*(x,y)+2\epsilon_{NO}}{V^*(x,y)}\,. \tag{22}$$

Now, noting that $\tilde{V}^*\le V^*$ everywhere yields $\epsilon(\tilde{c})\le\epsilon(c)$. □

B Neural Operators

B.1 Nonlocal Neural Operators

Formally, we provide a review of the nonlocal neural operator (NNO) as in Lanthaler et al. (2023) under the architecture of the general neural operator first introduced in Kovachki et al. (2023). Such a framework is useful as it encompasses almost all neural operator architectures including the well known DeepONet Lu et al. (2021) and FNO Li et al. (2021). Let $\mathcal{X}\subset\mathbb{R}^n$ be a bounded domain and define the following function spaces consisting of continuous functions $\mathcal{F}_c\subset C^0(\mathcal{X};\mathbb{R})$, $\mathcal{F}_v\subset C^0(\mathcal{X};\mathbb{R})$. Then, a NNO is defined as a mapping $\hat{\Psi}:\mathcal{F}_c(\mathcal{X}:\mathbb{R})\to\mathcal{F}_v(\mathcal{X};\mathbb{R})$ which can be written in the compositional form $\hat{\Psi}=\mathcal{Q}\circ\mathcal{L}_L\circ\cdots\circ\mathcal{L}_1\circ\mathcal{R}$ consisting of a lifting layer $\mathcal{R}$, hidden layers $\mathcal{L}_l, l=1,...,L$, and a projection layer $\mathcal{Q}$. Given a channel dimension d_c, the lifting layer $\mathcal{R}$ is given by

$$\mathcal{R}:\mathcal{F}_c(\mathcal{X};\mathbb{R})\to\mathcal{F}_s(\mathcal{X};\mathbb{R}^{d_c}),\quad c(\boldsymbol{x})\mapsto R(c(\boldsymbol{x}),\boldsymbol{x})\,, \tag{23}$$

where $\mathcal{F}_s(\mathcal{X};\mathbb{R}^{d_c})$ is a Banach space for the hidden layers and $R:\mathbb{R}\times\mathcal{X}\to\mathbb{R}^{d_c}$ is a learnable neural network acting between finite dimensional Euclidean spaces. For $l=1,...,L$ each hidden layer $\mathcal{L}_l$ is of the form

$$(\mathcal{L}_l v)(\boldsymbol{x}) := \sigma\left(W_l v(\boldsymbol{x})+b_l+(\mathcal{K}_l v)(\boldsymbol{x})\right) \tag{24}$$

where weights $W_l \in \mathbb{R}^{d_c \times d_c}$ and biases $b_l \in \mathbb{R}^{d_c}$ are learnable parameters, $\sigma : \mathbb{R} \to \mathbb{R}$ is a smooth, infinitely differentiable activation function that acts component wise on inputs and $\mathcal{K}_l$ is the nonlocal operator given by

$$(\mathcal{K}_l v)(\boldsymbol{x}) = \int_{\mathcal{X}} K_l(\boldsymbol{x}, \boldsymbol{y}) v(\boldsymbol{y}) dy \tag{25}$$

where $K_l(\boldsymbol{x}, \boldsymbol{y})$ is the kernel containing learnable parameters given in various forms. For example, in the FNO architecture, $K_l(\boldsymbol{x}, \boldsymbol{y}) = K_l(\boldsymbol{x} - \boldsymbol{y})$, $K_l(\boldsymbol{x}) = \sum_{|k| \le k_{\max}} \hat{P}_{l,k} e^{ik\boldsymbol{x}}$ is a trigonometric polynomial (Fourier) approximation with $k_{\max}$ nodes and $\hat{P}_{l,k}$ is a matrix of complex, learnable parameters $\hat{P}_{l,k} \in \mathbb{C}^{d_c \times d_c}$. Note that (24) is almost a traditional feed-forward neural network except for the kernel term (25), that is nonlocal - it depends on points over the entire domain rather then just $\boldsymbol{x}$. Lastly, the projection layer $\mathcal{Q}$ is defined as

$$\mathcal{Q} : \mathcal{F}_s(\mathcal{X}; \mathbb{R}^{d_c}) \to \mathcal{F}_v(\mathcal{X}; \mathbb{R}), \quad s(\boldsymbol{x}) \mapsto Q(s(\boldsymbol{x}), \boldsymbol{y}), \tag{26}$$

where Q is a finite dimensional neural network from $\mathbb{R}^{d_c} \times \mathcal{X} \to \mathbb{R}$ yielding the final value of the operator $(\hat{\Psi}c)$ $(c \in \mathcal{F}_c)$ at the point $\boldsymbol{x} \in \mathcal{X}$.

Theorem 2 (Neural operator approximation theorem Lanthaler et al. (2023, Theorem 2.1))**.** *Let $\mathcal{X} \subset \mathbb{R}^n$ be a bounded domain with Lipschitz boundary and $\overline{\mathcal{X}}$ its respective closure. Let $\Psi : C^0(\overline{\mathcal{X}}; \mathbb{R}) \to C^0(\overline{\mathcal{X}}; \mathbb{R})$ be a continuous operator, where $C^0(\overline{\mathcal{X}}; \mathbb{R})$ is the set of continuous functions $\overline{\mathcal{X}} \to \mathbb{R}$. Then for any $\epsilon > 0$ and some compact set $\mathcal{K} \subset C^0(\overline{\mathcal{X}}; \mathbb{R})$, there exists a nonlocal neural operator $\hat{\Psi} : \mathcal{K} \subset C^0(\overline{\mathcal{X}}; \mathbb{R}) \to C^0(\overline{\mathcal{X}}; \mathbb{R})$ such that*

$$\sup_{c \in \mathcal{K}} \|\Psi(c) - \hat{\Psi}(c)\|_\infty \le \epsilon. \tag{27}$$

B.2 Domain-Agnostic Fourier Neural Operators

Domain-Agnostic Fourier Neural Operators (DAFNO), first introduced in Liu et al. (2023), augment the FNO with a mask such that FNO's can be applied on non rectangular geometries despite the requirement of Fourier transforms to operate on periodic domains. For the motion planning problem, as above, consider partitioning $\mathcal{X}$ into two domains $\mathcal{S}$ for the safeset and $\mathcal{X} \backslash \mathcal{S}$ as the unsafe set. Furthermore, if $\mathcal{X}$ is not a box, we consider the smallest rectangle $\mathcal{T}$ that contains $\mathcal{X}$ (see Liu et al. (2023, Fig. 1)) and add all the padded points in $\mathbb{T} \backslash \mathcal{X}$ to the unsafe set. Then, consider the following two characteristic functions that encode the geometry

$$\chi(\boldsymbol{x}) := \begin{cases} 1 & \boldsymbol{x} \in \mathcal{S} \\ 0 & \boldsymbol{x} \in \mathcal{X} \backslash \mathcal{S} \end{cases}, \tag{28a}$$

$$\tilde{\chi}(\boldsymbol{x}) := \tanh(\beta d_{\mathcal{S}}(\boldsymbol{x}))(\chi(\boldsymbol{x}) - 0.5) + 0.5, \tag{28b}$$

where $\beta \in \mathbb{R}$ is a chosen hyper parameter parameter and $d_{\mathcal{S}}$ is the sign distance function (SDF) given by

$$d_{\mathcal{S}}(\boldsymbol{x}) = \begin{cases} \inf_{y \in \partial \mathcal{X}_{\mathcal{S}}} \|\boldsymbol{x} - \boldsymbol{y}\| & \text{if } \boldsymbol{x} \in \mathcal{S} \\ -\inf_{y \in \partial \mathcal{X}_{\mathcal{S}}} \|\boldsymbol{x} - \boldsymbol{y}\| & \text{if } \boldsymbol{x} \notin \mathcal{X}_{\mathcal{S}}\,. \end{cases} \tag{29}$$

The idea is that χ masks the geometry, setting the domain to 0 where obstacles are and $\tilde{\chi}$ is a smoothed version to ensure that the encoded geometry is continuous satisfying Theorem 2. Then, the DAFNO architecture uses the following instantiation of the kernel in (25) as

$$(\mathcal{K}_l v)(\boldsymbol{x}) = \int_{\mathbb{T}} \tilde{\chi}(\boldsymbol{x}) \tilde{\chi}(\boldsymbol{y}) K_l(\boldsymbol{x} - \boldsymbol{y})(v(\boldsymbol{y}) - v(\boldsymbol{x})) dy\,, \tag{30}$$

where $K_l(\boldsymbol{x} - \boldsymbol{y})$ is defined as the trigonometric polynomials as in the original FNO architecture. Lastly, we mention the subtraction of $v(\boldsymbol{x}) - v(\boldsymbol{y})$ in (30), was introduced in You et al. (2022a) and has shown superior performance over FNOs.

C Review of Viscosity Solutions for PDEs

Definition 2 (Viscosity solution (Katzourakis, 2014))**.** *A bounded, uniformly continuous function $V : \mathcal{X} \to \mathbb{R}$ is called a **viscosity solution** of the Eikonal initial-value Problem (7) provided* *i.* $V(\boldsymbol{x}) = c(\boldsymbol{x})$ *when* $x \in \mathcal{G}$

ii. Given any function $v \in C^1(\mathcal{X})$*, the following two hold*

a. If $V - v$ *has a local maximum at the point* $\boldsymbol{x} \in \mathcal{X}$*, then,*

$$\|\nabla v(\boldsymbol{x})\| - c(\boldsymbol{x}) \leq 0. \tag{31}$$

b. If $V - v$ *has a local minimum at the point* $\boldsymbol{x} \in \mathcal{X}$*, then,*

$$\|\nabla v(\boldsymbol{x})\| - c(\boldsymbol{x}) \geq 0\,. \tag{32}$$

We briefly mention that the existence of viscosity solutions can be shown in two ways - namely Perron's method via the maximum principle or via the vanishing viscosity method. We briefly detail the latter result and refer the reader to Evans (2010) for more formal analysis.

Lemma 3. *(Vanishing viscosity method Evans (2010)) Following the definition above, let* Ψ *be the viscosity solution to the given problem (7). Then, let* Ψ_ϵ *be the classical solution to the following viscous regularized Eikonal problem*

$$\|\nabla(\Psi_\epsilon c)(\boldsymbol{x})\| + \epsilon_v \Delta(\Psi_\epsilon c)(\boldsymbol{x}) = c(\boldsymbol{x})\,, \quad \forall c \in \mathcal{F}_c, \boldsymbol{x} \in \mathcal{X}, \tag{33a}$$

$$\Psi(\boldsymbol{g}) = c(\boldsymbol{g})\,, \qquad \forall \boldsymbol{g} \in \mathcal{G}\,. \tag{33b}$$

Then, Ψ_ϵ *uniformly converges to* Ψ *as* $\epsilon_v \to 0$*.*

D Learning Value Functions: Experimental details and additional results

D.1 A brief review of the baseline methods

To evaluate our models, we aimed to compare across a series of different modern motion planning methodologies. In particularly, we consider perspectives across reinforcement like methods such as VIN and IEF, operator learning architectures such as FNO and DAFNO, and a physics informed approach in NTFields. In this section, we use a series of different GPUs for training and testing. To clarify, all the 2D experiments use a NVIDIA 3090 Ti. The 3D iGibson experiments use a NVIDIA $A100$ for data-generation and training, while we employ a NVIDIA 4060 during testing.

- **Value Iteration Networks (VIN) Tamar et al. (2016)** VIN attempts to learn value functions by approximating the value iteration (VI) algorithm. To do so, they introduce two neural network components to their algorithm. The first model, deemed the VI module, takes in the previous value function estimate and the current reward observed to provide an estimate on the current value function. In implementation, for 2D experiments, the VI module is a classical CNN. They then combine this module a planning module that takes in the value function and current state and passes this through an attention mechanism coupled with a classical NN to achieve the best path direction.
- **Implicit Environment Functions (IEF) Li et al. (2022)** IN IEF2D/IEF3D, the authors take the classical neural implicit network architecture, which represents 2D and 3D scenes via their signed distance functions, and instead learns to represent the scene by obtaining distances between samples start goal pairs. As such their methodology is continuous, and is able to learn trajectories by analyzing these distance functions. However, to generalize across environment's, the authors first project the scene to a latent space via a auto-encoder and then learn a neural implicit function in that latent space.
- **Fourier Neural Operators (FNO) Li et al. (2021)** Like PNO, FNOs learn the solution of operator mappings across continuous function spaces. In particular, FNOs project the input map into a high dimensional latent space, of which an FFT is then performed. From here, the FNO learns a weight matrix in frequency space that multiplies with the encoded input before performing an IFFT and then reprojecting back to the desired output resolution. They have been extremely successful in learning operator solutions specifically in the context of weather (See Kurth et al. (2023)).
- **Domain-Agnostic Fourier Neural Operators (DAFNO) Liu et al. (2023)** DAFNO, an extension of FNO, maintains the same network structure as an FNO, but additionally encodes the domain geometry into the kernel calculation in frequency space. That is,

DAFNO, multiplies the encoded input in frequency space by an additional mask containing the domain geometry. This approach, taking advantage of the fact that Fourier transforms only operate on periodic grids, enable users to learn FNO approximations on non-uniform geometries.

- **Neural Time Fields (NTFields) Ni & Qureshi (2023)** NTFields is similar to the operator learning architectures in that the network aims to learn an operator, but specifically they constrain that operator to be the solution operator of an Eikonal PDE. In particular, they re-parameterize the value function as a factorized Eikonal equation $V(\boldsymbol{x}, \boldsymbol{g}) = \frac{\|\boldsymbol{x}-\boldsymbol{g}\|}{\tau(\boldsymbol{x},\boldsymbol{g})}$ where τ is learned via a neural network. They then perform training with a physics informed loss based on the satisfaction of the Eikonal PDE enabling NTFields to learn paths *without training data*. As such NTFields is able to generalize to high dimensional environments, but is constrained as it needs retraining for each new environment geometry.

D.2 Small-scale 2D maze experiments

To compare with state of the art value function predictors such as VIN and IEF2D, we explore the efficacy of a PNO on the small-scale 2D Grid-world experiments in Tamar et al. (2016). We used the same dataset used to test the VIN framework of which IEF2D also employs.

The parameters and the relative errors of our model are presented in Table 4. To develop our model, it took approximately 40 minutes of training on a NVIDIA RTX 3090 Ti GPU.

Table 4: Model parameters and performance metrics for PNO over the GridWorld dataset. The PNO for 8×8 was smaller than that trained for 16×16 and 28×28.

Map size	**PNO model number of parameters**	**Avg. L_2 relative error training data**	**Avg. L_2 relative error test data**
8×8	238816	0.019	0.022
16×16	690432	0.031	0.035
28×28	690432	0.027	0.045

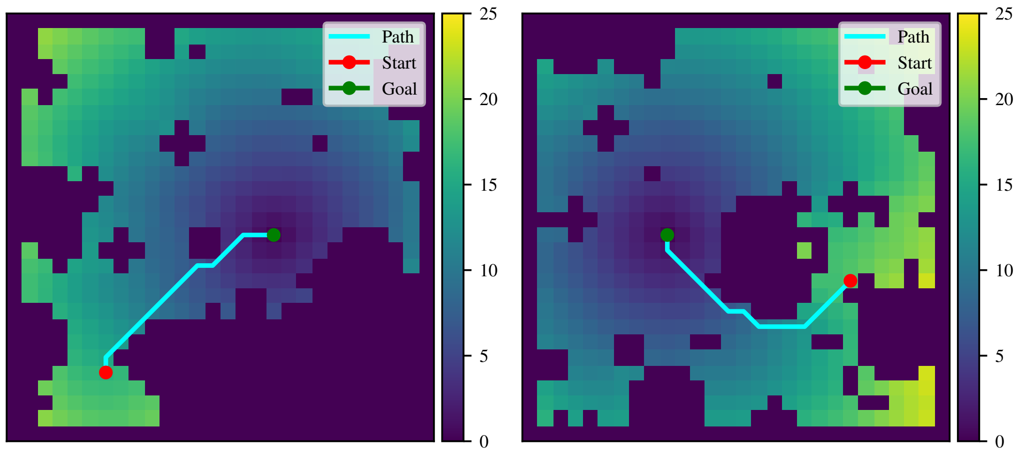


Figure 5: Two examples of PNO planning on 28×28 Grid-world dataset. Planning is done on the value function generated by PNO using classical gradient descent.

D.3 MovingAI city experiments

In this experiment, we highlight the super-resolution capabilities of the planning operator architecture. To do so, we created a custom 64×64 resolution dataset (will be publicly available) and trained the neural operator on the custom dataset. Note, this dataset consists of just objects on a rectangular grid as shown in the example in Fig. 2. Furthermore, our dataset did not include any of the city maps, but the synthetic object maps closely resemble a similar structure as the city maps. Our architecture consisted of 157808 parameters and we achieved an L_2 relative error of 0.1 for both training and testing taking approximately 10 minutes for training on a NVIDIA 3090 Ti GPU.

To showcase our the benefit of both the obstacle encoding and inductive bias, we perform comparisons across a wide set of operator learning benchmarks. We briefly summarize both the training and testing error in Table 1. Note all the testing sets in each category are the same maps structurally, just scaled to the target resolution via averaging. For the in-distribution obstacle dataset, we consider $1k$ maps with 10 different goals each for training. and 10 maps with 10 different goals each for testing. For the MovingAI city experiments, we include all 30 city maps with 3 goals each for testing. An example of the MovingAI city result along with comparison errors is given in Fig. 6 for a snapshot of Paris. Furthermore, to all models were trained and tested using the PyKonal FMM for the value function and a classic numerical solver for the SDF was used for DAFNO. For PNO, the FNO generated SDF was trained over 1k instances of the 64×64 dataset (achieving an relative L_2 error of 0.064) and is also performing super-resolution when applied.

Lastly, for the same calculations, we compute the speedup of the neural operator at various resolutions on a NVIDIA 3090 Ti GPU (ML models) and utilize the extremely powerful AMD Ryzen 9 7950X CPU for the numerical solvers. In Table 5, we break down exactly the calculation time of both the SDFs as well as the value function where on small scaled maps, it is clear the numerical solver performs best; however, as the scale increases, we can see that the operator architectures do not lose much computation expenditure while the numerical solvers scale poorly.

Table 5: Computation times for super-resolution calculations average over 1000 instances on the Moving AI lab $2D$ dataset (64^2 indicates 64×64). The DAFNO SDF was calculated using the SciPy numerical solver. The numerical solver used for FMM is via Pykonal.

	Avg. computation time signed distance function(1000 inst, ms) $\downarrow$				**Avg. computation time value function(1000 inst, ms) $\downarrow$**				**Avg. computation time total function(1000 inst, ms) $\downarrow$**			
Map size	64^2	256^2	512^2	1024^2	64^2	256^2	512^2	1024^2	64^2	256^2	512^2	1024^2
FNO	—	—	—	—	1.6168	**1.6356**	**1.6563**	**2.281**	1.6168	**1.6356**	**1.6563**	**2.281**
DAFNO	**0.1734**	1.9806	8.4207	46.1614	2.9232	2.9889	3.024	3.6626	3.0966	4.9695	11.4447	49.825
PNO	1.6168	**1.6356**	**1.6563**	**2.281**	3.954	3.6729	4.6735	6.0332	5.5708	5.3085	6.3298	8.3142
Numerical solver (FMM)	—	—	—	—	**0.3454**	5.9612	25.6555	104.6184	**0.3454**	5.9612	25.6555	104.6184

D.4 3D iGibson dataset

For our 3D experiments, we use the iGibson Dataset Shen et al. (2021). The dataset consists of only 10 maps which are not sufficient to train an accurate model. As such, we perform two augmentations for generating sufficient training data. First, we consider different training instances by rotating the different maps across about the z-axis by 90 degrees creating four versions of the same map. Additionally, we augmented the dataset with 32 maps from the HouseExpo dataset Li et al. (2020). The maps were extruded to form 3D maps. Then, for each map, we randomly sampled 5 start goal pairs leading to a total dataset size of 360. We then performed a $90/10$ train test split explicitly ensuring that the training dataset did not contain any Samuel environments. Further, no start goal positions in the evaluation dataset in Fig. 3 were seen during training. Our model consisted of 418528 parameters of which we achieved a 0.08 L_2 relative training error and a 0.19 L_2 relative testing error for learning the value function. The computational calculations for value function generation were computed using a NVIDIA 4060 GPU.

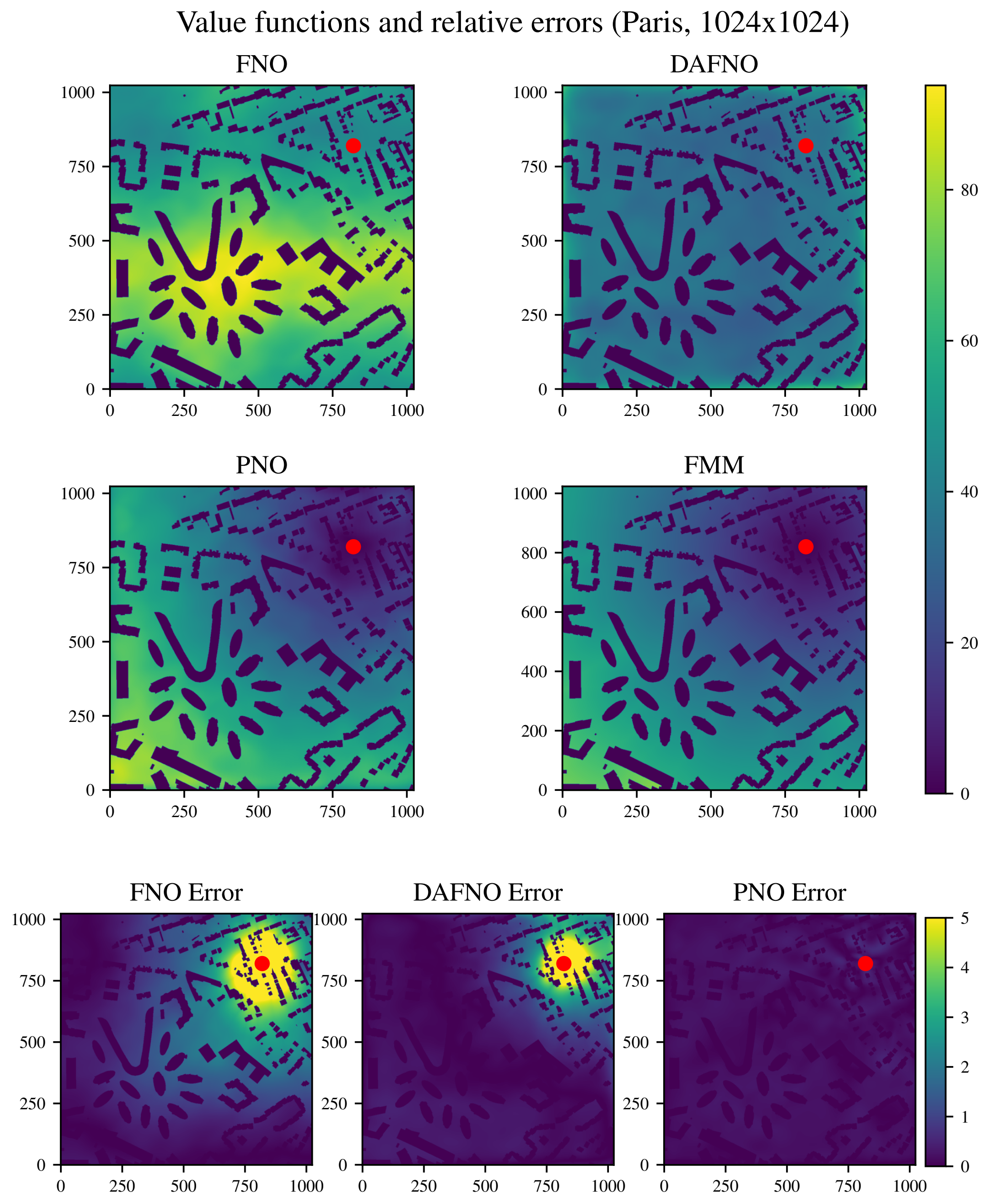


Figure 6: Example of various operator architectures on a Paris 1024×1024 map. The red dot indicates the target goal position.

E Employing operator learned value functions as neural heuristics: Experiment details and additional results.

For this experiment, we trained directly on the city maps at 256^2, 512^2 and 1024^2 resolutions. All of the training and testing in this section was completed using an NVIDIA $A100$. To build our dataset, we considered the 30 city maps provided by the MovingAI city dataset along with 10 goals for each map. Additionally, we randomly erode each of the maps between 1 and 30 layers to generate more data and to help the operator infer the effect of erosion. Again, for each eroded map, we use 10 goals.

For employment as a heuristic, FMM was insufficient for generating training data. This is due the fact A* algorithm works using a set of discrete control inputs to find a path with the shortest distance. However, the value function generated by the Eikonal equation (FMM) represents a value function for a continuous control input space. Using this directly as a heuristic does not yield an accurate value function since such a function largely under approximates the cost-to-go function for the shortest distance problem with discrete control space. In order to alleviate this issue we train our neural operator on value functions generated using the Dijkstra Algorithm that gives the value function at every node for the same set of control inputs. This required a data generation time of 8 hours for the 1024 × 1024 city maps (3 hours and 25 minutes for the 512 × 512 and 256 × 256 maps), but only needed to be complete once, offline. We provde the full model sizes as well as the relative errors in Table 6.

Table 6: Model parameters and performance metrics for the PNO models trained and deployed as neural heuristics.

Map size	**PNO model number of parameters**	**Avg. L_2 relative error training data**	**Avg. L_2 relative error test data**
256 × 256	26029	0.051	0.065
512 × 512	102048	0.049	0.055
1024 × 1024	161920	0.058	0.053

In addition to our results in Section 5, we also conducted an analysis on the effect of erosion. Fig. 7 shows that as the amount of layers eroded increases, the path improves at the cost of expanding more nodes as the heuristic which is as expected given that a fully eroded map yields the Euclidean norm. Perhaps future work can explore different eroding methods for improving the admissibility of the value function.

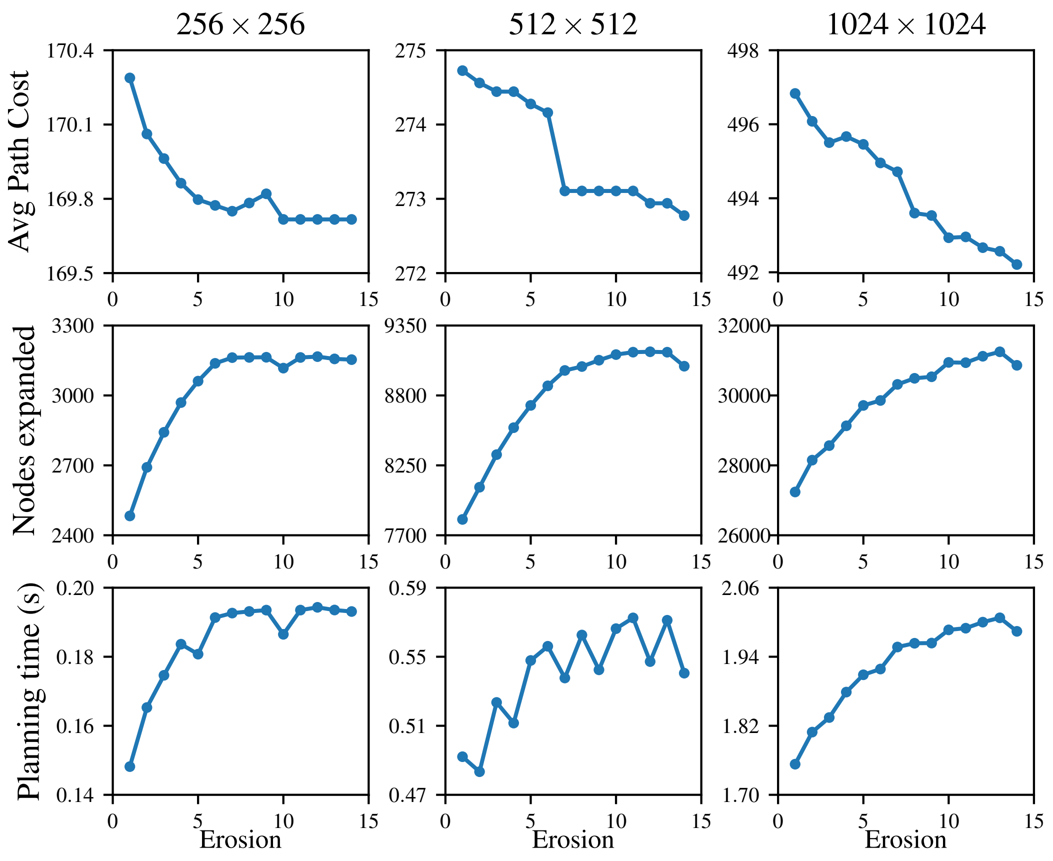


Figure 7: Effect of erosion on various parameters