$\mathcal{M}$-PINNs: $\mathcal{M}$anifold - Physics-Informed Neural Networks for Solving PDEs on Curved and Changing Domains

Riccardo Valperga
VIS Lab
University of Amsterdam
r.valperga@uva.nl

Samuele Papa
VIS Lab
University of Amsterdam

Efstratios Gavves
VIS Lab
University of Amsterdam

Abstract

Solving partial differential equations on domains with complex geometries is a significant challenge. Physics-Informed Neural Networks (PINNs) would offer a promising mesh-free approach, but their application to manifolds is limited. This paper introduces Manifold Physics-Informed Neural Networks ($\mathcal{M}$-PINNs) that leverage domain decomposition for partitioning complex shapes into simpler charts, for which we learn local embeddings. The core of our method is a novel *universal autoencoder* architecture designed to generate these embeddings zero-shot for any given chart. By training this system on a diverse set of charts, the decoders for new, unseen charts can be obtained without retraining or adaptation. These decoders then serve as the local embeddings within which separate PINNs are trained to solve the PDE, effectively integrating the domain's geometry into the process. This approach results in a scalable and adaptable framework for solving PDEs on complex manifolds.

1 Introduction

Physics-Informed Learning [Karniadakis et al., 2021, Hao et al., 2022, Raissi et al., 2017] recently emerged as a new paradigm for leveraging the universal approximation capabilities of deep neural networks combined with automatic differentiation to incorporate knowledge of physical laws, such as partial differential equations (PDEs) with initial and boundary conditions, into the learning process. It offers an interesting alternative to numerical discretization algorithms that are often prohibitively expensive when it comes to incorporating noisy data from measurements to solve inverse problems. Physics-Informed Neural Networks (PINNs) [Raissi et al., 2019] are one instance of physics-informed learning consisting of various neural network architectures trained to approximate the solution of a PDE or, in general, an inverse problem, penalizing deviations from physical laws, such as the PDE itself or a conservation law. This approach provides several advantages over traditional numerical methods, such as the inherent ability to handle inverse problems where parameters or governing equations themselves are partially unknown.

Due to their high adaptability, PINNs have been applied extensively in various domains such as solid mechanics, fluid mechanics [Raissi et al., 2020], heat transfer [Cai et al., 2021], and biomedical engineering [Kissas et al., 2020, Ruiz Herrera et al., 2022].

Despite substantial progress in training techniques and architectures, a large body of PINN research focuses on relatively simple flat domains, while many practical problems involve domains with curved geometries and complex boundaries or topologies. Little work has been done to adapt PINNs to complex domains, particularly curved surfaces in 3D, maintaining their mesh-free nature, data integration capabilities and unified framework for forward and inverse problems. On top of that, the

Preprint. Under review.

practical application of standard PINN formulations has faced significant computational hurdles: training these networks, especially for the large and complex systems often encountered in real-world science and engineering, proved to be computationally expensive and faced fundamental limitations in scalability. Domain Decomposition Methods (DDM) [Jagtap and Karniadakis, 2020, Hu et al., 2021, Jagtap et al., 2020, Li et al., 2019] offers a valid strategy to address these challenges of applying PINNs to large-scale problems or those with complex geometries and multi-scale features. Rooted in classical numerical analysis, DDM provides a "divide and conquer" approach: decomposing the domain into subdomain and training separate PINNs independently in each, ensuring agreement at the boundaries. Domain decomposition techniques such as extended PINNs (XPINNs) [Jagtap and Karniadakis, 2020] not only have provable theoretical advantages over vanilla PINNs, but can also better exploit GPU parallelism [Meng et al., 2020].

In this work, we propose a geometry-aware formulation of PINNs to explicitly incorporate the domain's geometry into the PINN objective and solve forward and inverse problems *on* the manifold. We leverage domain decomposition and autoencoders to partition manifolds into *charts*, and learn *local* coordinates together with the Riemannian metric tensor field on the manifold. To do that, we propose a novel architecture that we refer to as *universal autoencoder* to obtain the local embeddings in a zero-shot manner. This results in a scalable and adaptable method that performs better than state-of-the-art PINN techniques, especially in the scarce data regime. Moreover, allowing PINNs to be trained on *time-varying* domains.

In this work, our contributions are the following:

- We introduce a domain decomposition strategy for training PINNs on arbitrary manifolds.
- We propose using neural network decoders as local embeddings of manifold partitions. To the best of our knowledge, this is the first mesh-free, fully differentiable approach for training PINNs on manifolds.
- We develop a universal autoencoder architecture that produces local embeddings zero-shot, allowing scalable and efficient training across arbitrary charts without retraining. Importantly, this allows PINNs to be trained on domains that can change over time.

Figure 1: Manifold is partitioned into charts. Then, charts are encoded into local charts.

2 Preliminaries

We begin by briefly describing the problem setting of physics-informed neural networks and domain decomposition to solve forward and inverse problems defined on smooth manifolds.

2.1 Physics-Informed Neural Networks

We consider a parametrized PDE system given by:

$$\begin{aligned} &\mathcal{N}[\boldsymbol{u};\lambda](x,t)=0, \quad x\in\mathcal{M},\ t\in[0,T],\\ &\boldsymbol{u}(x,t_0)=\boldsymbol{u}_0(x), \quad x\in\mathcal{M},\\ &\boldsymbol{u}(x,t)=\boldsymbol{u}_{bc}(t), \quad x\in\partial\mathcal{M},\ t\in[0,T], \end{aligned} \tag{1}$$

where x is the spatial coordinate and t is the time; $\boldsymbol{u}(x,t)$ is the solution of the PDE with initial condition $g_0(x)$ and boundary condition $g_\Gamma(t)$ (which can be Dirichlet, Neumann or mixed boundary condition); $\partial\mathcal{M}$ represent the boundary of the domain. Additionally, we might know the value of the solutions at some, possibly very sparse, locations:

$$\boldsymbol{u}(x^i,t^i)=\boldsymbol{u}^i, \quad (x^i,t^i)\in\mathcal{D}_{\text{data}}.$$

Physics-informed neural networks work by approximating the solution of the PDE with a neural network architecture $\boldsymbol{u}_\theta$, parameterized by θ, trained with gradient descent by minimizing the

following composite loss function that penalizes deviations from the solution

$$\mathcal{L}(\theta) = w_{ic}\mathcal{L}_{ic}(\theta) + w_{bc}\mathcal{L}_{bc}(\theta) + w_r\mathcal{L}_r(\theta), \tag{2}$$

where

$$\mathcal{L}_{ic}(\theta) = \frac{1}{N_{ic}} \sum_{i=1}^{N_{ic}} \left| \mathbf{u}_\theta(x_{ic}^i, 0) - \boldsymbol{u}_0(x_{ic}^i) \right|^2, \tag{3}$$

$$\mathcal{L}_{bc}(\theta) = \frac{1}{N_{bc}} \sum_{i=1}^{N_{bc}} \left| \mathbf{u}_\theta(x_{bc}^i, t_{bc}^i) - \boldsymbol{u}_{bc}(t_{bc}^i) \right|^2, \tag{4}$$

$$\mathcal{L}_r(\theta) = \frac{1}{N_r} \sum_{i=1}^{N_r} \left| \mathcal{N}[\boldsymbol{u}_\theta, \lambda](t_r^i, x_r^i) \right|^2. \tag{5}$$

Where $\left\{x_{ic}^i\right\}_{i=1}^{N_{ic}}$, $\left\{x_{bc}^i, t_{bc}^i\right\}_{i=1}^{N_{bc}}$ and $\left\{x_r^i, t_r^i\right\}_{i=1}^{N_r}$ are points sampled from $\mathcal{M}$, $\partial\mathcal{M}$, and $[0, T]$.

2.2 Domain decomposition

The fundamental idea behind domain decomposition is to break down a large, computationally challenging problem defined on a global domain $\mathcal{M}$ into a collection of smaller, more manageable subproblems, each defined on a smaller subdomain $\mathcal{M}_i$. This partitioning is such that adjacent subdomains share a region of overlap. In DD techniques such as XPINNs Jagtap and Karniadakis [2020] or cPINNs Jagtap et al. [2020], a separate neural network $\boldsymbol{u}_{\theta_i}$ is assigned to each partition and trained imposing appropriate boundary conditions at the newly created artificial interfaces between subdomains to enforce consistency of the solutions. The objective for the i-th neural network becomes

$$\mathcal{L}_{DD}(\theta_i) = \mathcal{L}_{ic}(\theta_i) + w_I \sum_{ij:\in \mathcal{M}_i \cap \mathcal{M}_j \neq \emptyset} \mathcal{L}_I(\theta_i, \theta_j), \tag{6}$$

where

$$\mathcal{L}_I(\theta_i, \theta_j) = \frac{1}{n_{I,ij}} \sum_{k=1}^{n_{I,ij}} |u_{\theta_i}(x_{I,ij}^k, t_I^k) - \{\{u_{\theta\text{ avg}}\}\}(x_{I,ij}^k, t_I^k)|^2, \tag{7}$$

where $\{\{u_{\theta\text{ avg}}\}\}(x_{I,ij}^k) = (u_{\theta_i}(x_{I,ij}^k) + u_{\theta_j}(x_{I,ij}^k))/2$, $n_{I,ij}$ is the number of interface points between the i-th and j-th subdomains, while $x_{I,ij}^k$ is the k-th interface points between them.

3 Method

Our proposed method for solving PDEs on general manifolds consists of using autoencoders as local embeddings of appropriately partitioned manifolds. First, we outline the theory of PDEs on Riemannian madifold [Do Carmo, 2016] and then introduce the universal autoencoder for learning local embeddings zero-shot.

3.1 PDEs on Riemannian manifolds

We consider the problem of learning a partially observed function $\boldsymbol{u} : \mathcal{M} \longrightarrow \mathbb{R}$ defined on a d-dimensional smooth manifold $\mathcal{M}$. Without loss of generality $\mathcal{M}$ is a subset of $\mathbb{R}^n$ for some $n \geq d$. Practically, the input domain of $\boldsymbol{u}$ is a point-cloud $\mathcal{D} = \{x_i\}_{i=1}^N \subset \mathbb{R}^n$ of points lying on $\mathcal{M}$. We assume we know that $\boldsymbol{u}$ is the solution of some PDE of the form $\mathcal{N}[\boldsymbol{u}; \lambda] = 0$, where $\mathcal{N}[\cdot; \lambda]$ is a, generally nonlinear differential operator parameterized by λ. We will now endow $\mathcal{M}$ with a Riemannian metric and generalize differential operators to account for the intrinsic geometry of the manifold.

A Riemannian metric is a positive definite, symmetric 2-tensor field g that determines an inner product on each tangent space $T_x\mathcal{M}$, that is, for all $x \in \mathcal{M}$, and for all $X, Y \in T_x\mathcal{M}$, $g(X, Y) = g(Y, X)$ and $g(X, X) = 0$ if and only if $X = 0$. A Riemannian metric accounts for the intrinsic geometry, or *shape* of the manifold, when computing norms of tangent vectors or angles between them. In

local coordinates, g can be represented as a positive definite, symmetric, d-by-d matrix $\mathbf{g}$ such that $g(X,Y) = X^T \mathbf{g} Y$.

Let $\boldsymbol{u}$ be a smooth scalar function on $\mathcal{M}$, and X be a vector field on $\mathcal{M}$. We denote $\mathcal{X}(\mathcal{M})$ the space of vector fields on $\mathcal{M}$. The gradient of $\boldsymbol{u}$ is the *metrically equivalent* vector $\nabla \boldsymbol{u}$ defined by

$$g(\nabla \boldsymbol{u}, X) = d\boldsymbol{u}(X), \quad \forall X \in \mathcal{X}(\mathcal{M}). \tag{8}$$

One can show that Eq. (8) yields the following expression for the gradient vector field in local coordinates

$$\nabla \boldsymbol{u} = g^{ij} \frac{\partial \boldsymbol{u}}{\partial x^i} \frac{\partial}{\partial x^j}, \tag{9}$$

where $\frac{\partial}{\partial x^i}$ is the chart-induced basis of the tangent space. Similarly, one can derive the expression in local coordinates of the divergence of a vector field

$$\text{div}(X) = \frac{1}{\sqrt{|g|}} \partial_i (\sqrt{|g|} X^i), \tag{10}$$

and the Laplacian:

$$\Delta \boldsymbol{u} = \frac{1}{\sqrt{|g|}} \partial_j \left(\sqrt{|g|} g^{ij} \frac{\partial \boldsymbol{u}}{\partial x^i} \right), \tag{11}$$

where g^{ij} is the inverse metric ($g^{ij} = [g^{-1}]_{ij}$), and $\sqrt{|g|}$ is the square root determinant of the g. It is immediate to see that all the above equations result in the well-known forms of the corresponding differential operators in Euclidean space, where g is everywhere equal to the identity matrix.

Let $\phi : \mathcal{M} \hookrightarrow \mathbb{R}^n$ be a (*local*) embedding of the manifold $\mathcal{M}$ into Eucledian space $\mathbb{R}^n$, and ϕ_* the push-forward of ϕ. Given a metric g_e on the embedding manifold (the Euclidean space $\mathbb{R}^n$ in this case), we can define the pull-back of g_e through ϕ as the metric $\phi^* g$ such that

$$\phi^* g(X,Y) = g(\phi_* X, \phi_* Y) \quad \forall X, Y \in \mathcal{X}(\mathcal{M}). \tag{12}$$

The metric $\phi^* g(X,Y)$ is a Riemannian metric on $\mathcal{M}$. Locally, its expression is

$$[\phi^* g(X,Y)]_{ij} = g_{\alpha\beta} \frac{\partial \hat{\phi}^\alpha}{\partial x^i} \frac{\partial \hat{\phi}^\beta}{\partial x^j}, \tag{13}$$

where $\hat{\phi} : \mathbb{R}^d \longrightarrow \mathbb{R}^n$ is the local representation of the embedding ϕ. We will induce a Riemannian metric on $\mathcal{M}$ from the Eucledian metric in $\mathbb{R}^n$ $[g_e]_{ij} = \delta_{ij}$ (the identity matrix). In matrix form, eq. (13) is simply

$$\phi^* g = \mathbf{J}_\phi^T \mathbf{J}_\phi, \tag{14}$$

where $\mathbf{J}_\phi$ is the Jacobian of ϕ.

3.2 Decoders as local embeddings

An *autoencoder* [Hinton and Salakhutdinov, 2006] is the composition of two neural networks $e_\psi : \mathbb{R}^n \longrightarrow \mathbb{R}^d$ and $d_\phi : \mathbb{R}^d \longrightarrow \mathbb{R}^n$ with $d < n$. The autoencoder is trained to approximate the identity map on the input domain, i.e., it is trained with the objective

$$\mathcal{L}_{ae}(\psi, \phi) = \frac{1}{N} \sum_i ||d_\phi(e_\psi(x_i)) - x_i||^2 \tag{15}$$

Suppose we are given a point cloud $\mathcal{P} = \{x_i\}_{i=1}^N \subset \mathbb{R}^3$ corresponding to points on a 2-dimensional manifold embedded in $\mathbb{R}^3$. If we train an autoencoder $d \circ e$ on $\mathcal{P}$ we can use the encoder e as a *chart map*, namely a diffeomorphism from the manifold to $\mathbb{R}^2$, and the decoder d as a *local embedding*, namely an injective map that is a homeomorphism onto its image. Note that we can also think of d as the inverse of the chart map e. We use a set of autoencoders, each one trained on subsets of the manifold. Once the decoder is trained we can obtain the Riemannian metric by pulling back the Eucledian metric as in eq. (14) using the Jacobian:

$$g = d^* g_e = \mathbf{J}_d^T \mathbf{J}_d. \tag{16}$$

This metric will be employed in the differential operators to compute the PINN residual objective in (5).

3.3 The universal autoencoder

Training a separate autoencoder for each chart in the manifold partition would be computationally demanding and difficult to scale. To address this limitation, we introduce the *universal autoencoder*, a single architecture capable of generating local embeddings for any chart in a zero-shot fashion. This model amortizes the cost of learning embeddings by training on a diverse collection of manifold charts, thereby generalizing to unseen charts without retraining. To understand the design of the universal autoencoder we outline its two fundamental requirements: (1) the model needs to produce local embeddings of manifold charts made of *any* number of points. (2) the local embedding needs to vary smoothly as the manifold charts changes shape. The universal autoencoder consists of three components:

- A chart encoder inspired by the Universal Physics Transformer (UPT) [Alkin et al., 2024a], which processes point clouds of arbitrary size and ordering, producing a compact latent representation of the chart;
- A conditional decoder, implemented as a SIREN-based conditional neural field [Dupont et al., 2022], from Sitzmann et al. [2020], with FiLM conditioning [Perez et al., 2018]. This decoder maps coordinates in the Euclidean space back to the ambient space;
- A Perceiver encoder [Jaegle et al., 2021] which maps point of the chart to their local representation in Eucledian space.

The first encoder produces two latent vectors: one for the encoder and one for the decoder. These vectors steer the Perceiver and the SIREN to produce a meaningful embedding-decoding pair tailored to the chart. In particular the Perceiver encoder is conditioned with cross-attention, and the SIREN is modulated with FiLM, which consists of multiplying its activations with the latent vector.

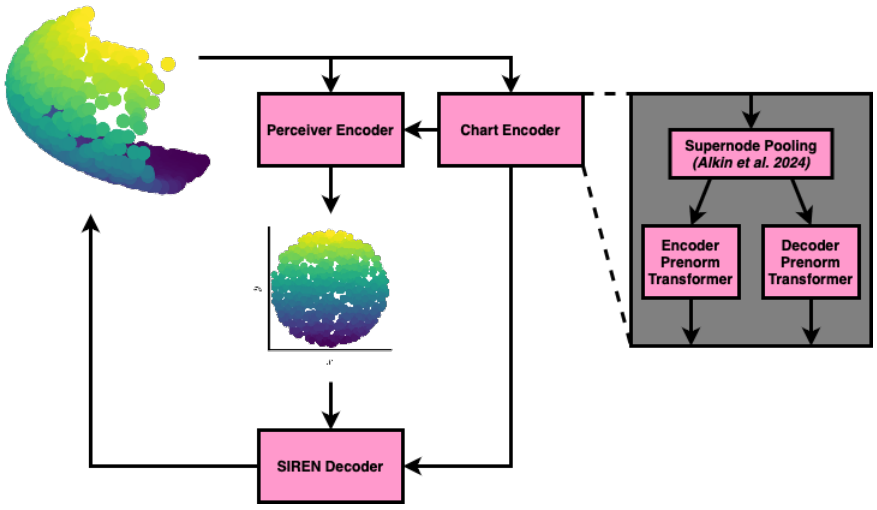


Figure 2: **Architecture diagram of the universal autoencoder.** The point-cloud goes through a Perceiver Encoder, that produces local coordinates, and the chart encoder that produces two latent representations. The latent representations are used to condition a SIREN decoder and the Perceiver encoder. The conditioned SIREN is the local embedding of the chart.

During training, the universal autoencoder is optimized end-to-end using a reconstruction loss across multiple charts:

$$\mathcal{L}_{\text{uae}} = \frac{1}{M} \sum_{i=1}^{M} \sum_{x \in \mathcal{P}_i} \left\| d_\phi \left(e_\psi(x; c^e(\mathcal{P}_i); c^d(\mathcal{P}_i)) \right) - x \right\|^2, \tag{17}$$

where d_ϕ is the conditional SIREN decoder, e_ϕ is the Perceiver encoder, and $c^e(\mathcal{P}_i), c^d(\mathcal{P}_i)$ are the two chart representations from the chart encoder.

From a functional perspective, the universal autoencoder can be viewed as a *neural operator* [Li et al., 2020, Alkin et al., 2024b, Wang et al., 2024, Li, 2021]: it learns a map between function spaces, taking as input point cloud representing a surface patch and returning its local coordinate embedding. If $\mathcal{X}$ denotes the space of discrete point clouds $\mathcal{P} \subset \mathbb{R}^n$ that approximate d-dimensional charts of a manifolds (e.g., elements of a shape space with mild regularity assumptions), and let $\mathcal{Y}$ be the space of smooth functions $d : \mathbb{R}^d \to \mathbb{R}^n$ representing local chart embeddings. Then, the universal autoencoder implements a map

$$\mathcal{C} : \mathcal{X} \to \mathcal{Y}, \quad \mathcal{C}(\mathcal{P})(z) = d_\phi(z; c^d(\mathcal{P})). \tag{18}$$

By amortizing this operator over a family of training patches $\mathcal{P}_i \in \mathcal{D} \subset \mathcal{X}$, the network generalizes to new unseen patches at inference time, producing their local embedding without retraining. This interpretation aligns our model with recent work on neural operators for scientific computing, while remaining grounded in geometric representation learning.

While our experiments focus on a limited family of patches obtained through continuous deformations, the universal autoencoder architecture is inherently scalable. In principle, it can be trained on a broad and diverse set of manifold charts, enabling generalization to a wide variety of unseen geometries. This scalability stems from the model's ability to amortize the embedding process across heterogeneous inputs through shared parameters and conditional adaptation. Furthermore the model is composed of transformer layers making it a potentially scalable architecture. Extending the model to operate over an extensive and heterogeneous patch distribution is a promising direction for future work and beyond the scope of the present study.

3.4 Partitioning manifolds

It is crucial to partition the manifold $\mathcal{M}$ into multiple subdomains $\mathcal{M}_i$ in such a way that autoencoders don't have a hard training. Furthermore, the subdomains must cover the manifold, namely $\bigcup_i \mathcal{M}_i = \mathcal{M}$, and have some overlapping region to allow information exchange between subdomains. We experimented multiple solutions based on existing algorithms borrowed from other disciplines, and found that a combination of greedy Poisson sampling and a region-growing algorithm based on the graph of nearest neighbours provides very good partitions very fast. We refer to the Appendix for further details on the partitioning algorithm.

3.5 $\mathcal{M}$-PINNs: Putting it All Together

Our full method, Manifold Physics-Informed Neural Networks ($\mathcal{M}$-PINNs), combines manifold-aware domain decomposition with a universal autoencoder to solve PDEs on complex geometries in a fully differentiable and mesh-free manner. The overall pipeline consists of three key stages:

1. **Partitioning the manifold:** The input point cloud $P \subset \mathbb{R}^n$ is partitioned into overlapping charts, each representing a local patch of the manifold. The partitions are constructed to cover the domain and allow information flow via overlaps.
2. **Zero-shot local embedding via the universal autoencoder:** For each chart P_i, the universal autoencoder generates a decoder d_i that serves as a local embedding $\mathbb{R}^d \to \mathbb{R}^n$. These decoders define the chart-specific Riemannian metrics $g_i = \mathbf{J}_{d_i}^\top \mathbf{J}_{d_i}$ via pullback of the Euclidean metric. The chart domain $d_i^{-1}(P_i)$ then becomes the coordinate space on which a PINN can be trained.
3. **Training local PINNs:** Separate PINNs u_{θ_i} are trained over the local chart domains $d_i^{-1}(P_i)$ using the Riemannian metric g_i to define intrinsic differential operators. Domain decomposition loss terms enforce agreement between neighboring charts across their overlap regions.

This framework enables scalable and parallel training on complex geometries without requiring mesh generation or retraining of embeddings. Crucially, the universal autoencoder amortizes the cost of learning chart embeddings by generalizing to unseen patches, making $\mathcal{M}$-PINNs adaptable to novel domains at inference time.

4 Related work

Given the rapid expansion and breadth of the literature on PINNs, this section does not provide an exhaustive survey. Instead, we focus our review on prior work most pertinent to the contributions herein, specifically examining PINN-based approaches developed for or applied to problems featuring complex geometric domains. Readers seeking a more general overview of PINN theory, methodologies, and applications are directed to several existing comprehensive survey Hao et al. [2022]. Regarding domain decomposition see Meng et al. [2020]for a detailed explanation of the techniques and parallel algorithm for cPINNs and XPINNs.

The challenge of applying PINNs to problems defined on domains with complex shapes and topologies led to the development of various techniques most of which focus on representing the domain boundary or transforming the coordinate system in simple 2D domains, rather than embedding the intrinsic geometry of the domain into the network's learning process. One approach involves a Signed Distance Function (SDF) to implicitly represent the domain's boundary [Kraus and Tatsis, 2024]. The SDF measures the shortest distance from any point to the boundary, with the sign indicating whether the point is inside or outside the domain. Incorporating SDFs allowed for the exact enforcement of Dirichlet boundary conditions by constructing the solution ansatz such that it inherently satisfied the boundary values, avoiding the need for potentially less accurate penalty terms in the loss function. While useful for boundary treatment, this application of SDFs was mainly demonstrated in 2D contexts and did not fundamentally alter the network's input to make it aware of the overall domain geometry, especially for complex 3D surfaces. Another strategy involved defining a mapping between the complex physical domain and a simpler, regular computational domain (e.g., a unit square or cube). This could be achieved through various techniques, including analytical transformations [Burbulla, 2023], elliptic coordinate mapping, or leveraging the inverse isoparametric maps [Sunil and Sills, 2024] from finite element methods. Once the problem was transformed to the computational domain, standard numerical techniques (like finite differences or convolutional neural networks operating on the regular grid) could be applied more easily. For problems involving vector fields defined on surfaces embedded in 3D space (e.g., fluid flow on a surface), researchers introduced additional penalty terms into the PINN loss function [Fang et al., 2021]. These terms explicitly enforced the constraint that the learned vector field must remain tangent to the surface at all points. This approach directly addresses a physical requirement on the solution. However, its effectiveness was noted to be higher for relatively simple and smooth surfaces where the Euclidean distance between points in the embedding space closely approximates the intrinsic geodesic distance along the manifold. For surfaces with high curvature or complex topology, where these distances differ significantly, this method could struggle.

Δ-PINNs Costabal et al. [2024] represent a distinct approach to handling complex geometries by fundamentally changing how the domain's structure is presented to the neural network. The central innovation of Δ-PINNs is a novel positional encoding scheme based on the eigenfunctions of the Laplace-Beltrami Operator (LBO) associated with the domain manifold (or the standard Laplacian operator for bounded domains in Euclidean space). Instead of feeding the standard Cartesian coordinates directly into the neural network, the input layer receives the values of the first k LBO eigenfunctions evaluated at the input point. The LBO is intrinsically linked to the geometry and topology of the manifold. Its eigenfunctions form a basis that naturally reflects these properties. Points that are close to each other in terms of geodesic distance on the manifold will tend to have similar values for the lower-frequency eigenfunctions. To the best of our knowledge Δ-PINNs are the best existing adaptation of PINNs to complex geometries, outperforming by far all the others. Therefore, we will compare our proposed method against it in the following section.

5 Experiments

To the best of our knowledge the Δ-PINN is the only method for adapting PINNs to complex geometries. In their paper Costabal et al. [2024], showed that Δ-PINNs outperform any other method that they compared with, such as GCNs Gao et al. [2022] and vanilla PINNs. Therefore, we compare $\mathcal{M}$-PINNs against Δ-PINNs only.

Scarce-data regime. We first investigate how $\mathcal{M}$-PINNs perform in the scarce-data regime, namely, when the problem is ill-posed and necessitates data points. We test with the Eikonal equation on a

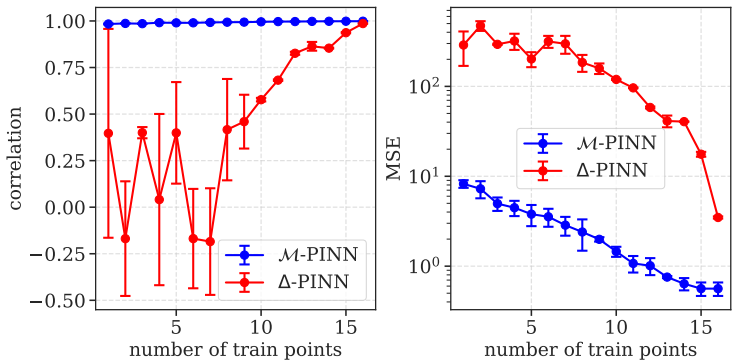
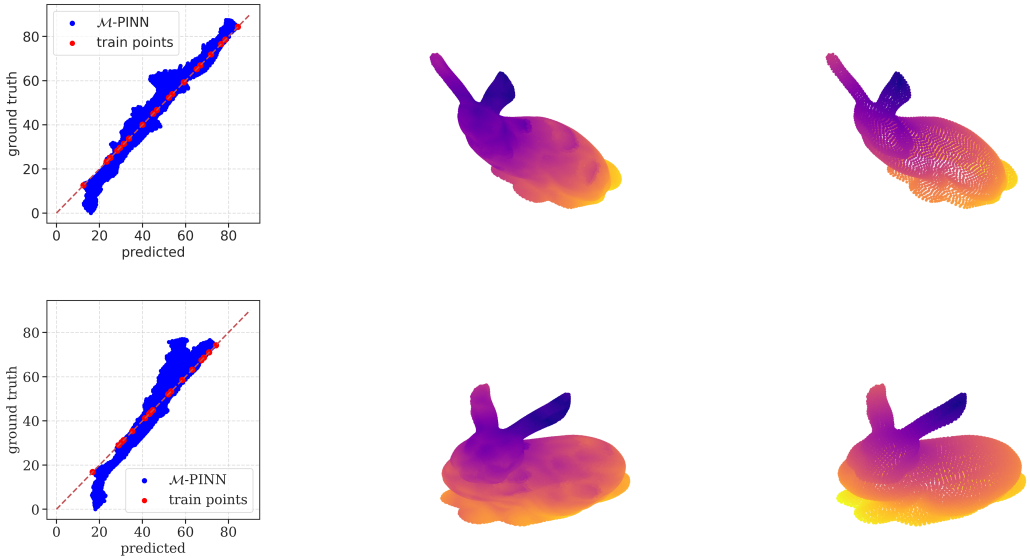


Figure 3: **Ablation study in the scarce data regime.** A Δ-PINN and a $\mathcal{M}$-PINN are trained with a varying number of data points. The two figures report MSE and correlation with the ground truth solution obtained with an exact geodesic algorithm.



(a) Correlation between groundtruth and predicted solution.

(b) Solution on the deformed manifold

(c) Ground truth solution on the deformed manifold.

Figure 4: **Eikonal equation on a deformed Stanford Bunny.** The local embeddings used to solve the PDE are obtained zero-shot from a new, unseen deformation.

coil. The Eikonal euation reads

$$\begin{cases} \sqrt{\nabla \boldsymbol{u} \cdot \nabla \boldsymbol{u}} = 1, \\ \boldsymbol{u}(x_b) = 0, \end{cases} \tag{19}$$

and the solution can be interpreted as the geodesic distance from the base point x_b to any point on the manifold. Since we do not impose the initial condition in the loss function, the problem is ill-posed and necessitates of data points. We use 40 charts and a single-layer MLP with 16 hidden units and tanh activation for the PINN in each chart. We compare with a Δ-PINN with the same hyperparameters described in Costabal et al. [2024] for the coil experiment. Figure 3 shows the correlation of the PINN solution with the ground-truth solution obtained with an exact geodesic algorithm from libigl. Jacobson et al. [2013]. From these experiments, it appears that Δ-PINNs struggle to generalize over such a large domain with few data-points.

Zero-shot generalization to unseen shapes and geometries. As a second experiment we test the ability of the universal autoencoder to generalize to unseen shapes zero-shot. To this end we train a

universal autoencoder on a dataset of charts from a complex shape. Before feeding the charts to the model we act on it with a random transformation from the family

$$\mathcal{T} = \{f : \mathbb{R}^3 \longrightarrow \mathbb{R}^3, p \mapsto \exp(t\mathbf{M})p\} \tag{20}$$

where $\mathbf{M}$ is a random 3-by-3 matrix obtained by sampling each entry from a normal distribution and then subtracting $\frac{1}{3}\text{Tr}(M)$ so that the final transformation is invertible and preserves volume. This is done to prevent the surface from self intersecting. At test time a new transformation is sampled and use to deform the shape. Figure 4 shows how we can obtain solutions in accordance with the ground truth by using the local embeddings obtained zero-shot with the universal autoencoder. See Appendix for further details.

Solving PDEs on time-varying surfaces. Last, we test the ability to solve a PDE on a domain that *change* in time. To do that we train a universal autoencoder on random deformations of a 2D plane embedded in 3D space, obtained by adding and subtracting sine waves of random frequencies and amplitude. The frequencies are such that the boundaries are kept fixed. We then solve the diffusion equation, $\boldsymbol{u}_t - D\Delta\boldsymbol{u}$, where Δ is the Laplacian from equation (3.1), while the shape is continuously deformed by one such deformation. See Appendix for further details. To the best of our knowledge this is the first algorithm that allows PDEs, or inverse problem solving with PINNs, on shapes that change in time.

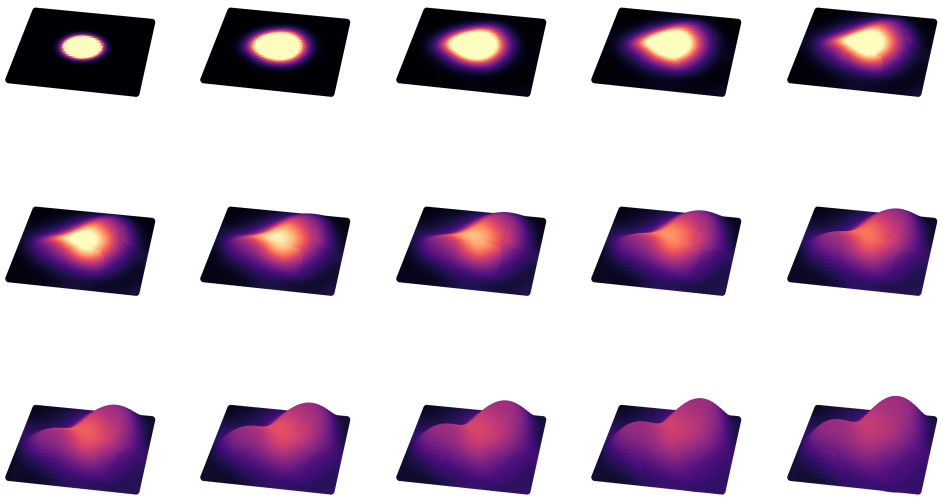


Table 1: **Solution of the diffusion equation with a domain that changes in time.** A Universal Autoencoder is trained on random deformations and then used to obtain local embeddings.

6 Conclusion

This paper introduced $\mathcal{M}$anifold Physics-Informed Neural Networks ($\mathcal{M}$-PINNs), a novel framework designed to address the problem of solving PDEs on domains with complex, curved, and time-varying, geometries. Traditional physics-informed neural networks are limited in their direct applicability to such manifolds. Our approach overcomes these limitations by integrating domain decomposition with a unique universal autoencoder architecture. Our experiments demonstrated the effectiveness of $\mathcal{M}$-PINNs, showing superior performance compared to state-of-the-art methods like Δ-PINNs in scenarios with scarce data. While we focused on relatively small function spaces for the chart embeddings, a promising direction for future work involves extending the universal autoencoder to generalize across more extensive, diverse, and heterogeneous collections of manifold charts. $\mathcal{M}$-PINNs offer a scalable, adaptable solution for tackling PDEs on complex manifolds. For the inherent differentiability of the universal autoencoder, in future work we will also explore shape optimization problems involving PDEs.

References

George Em Karniadakis, Ioannis G Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021.

Zhongkai Hao, Songming Liu, Yichi Zhang, Chengyang Ying, Yao Feng, Hang Su, and Jun Zhu. Physics-informed machine learning: A survey on problems, methods and applications. *arXiv preprint arXiv:2211.08064*, 2022.

Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations. *arXiv preprint arXiv:1711.10561*, 2017.

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.

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.

Shengze Cai, Zhiping Mao, Zhicheng Wang, Minglang Yin, and George Em Karniadakis. Physics-informed neural networks (pinns) for fluid mechanics: A review. *Acta Mechanica Sinica*, 37(12): 1727–1738, 2021.

Georgios Kissas, Yibo Yang, Eileen Hwuang, Walter R Witschey, John A Detre, and Paris Perdikaris. Machine learning in cardiovascular flows modeling: Predicting arterial blood pressure from non-invasive 4d flow mri data using physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering*, 358:112623, 2020.

Carlos Ruiz Herrera, Thomas Grandits, Gernot Plank, Paris Perdikaris, Francisco Sahli Costabal, and Simone Pezzuto. Physics-informed neural networks to learn cardiac fiber orientation from multiple electroanatomical maps. *Engineering with Computers*, 38(5):3957–3973, 2022.

Ameya D Jagtap and George Em Karniadakis. Extended physics-informed neural networks (xpinns): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations. *Communications in Computational Physics*, 28(5), 2020.

Zheyuan Hu, Ameya D Jagtap, George Em Karniadakis, and Kenji Kawaguchi. When do extended physics-informed neural networks (xpinns) improve generalization? *arXiv preprint arXiv:2109.09444*, 2021.

Ameya D Jagtap, Ehsan Kharazmi, and George Em Karniadakis. Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems. *Computer Methods in Applied Mechanics and Engineering*, 365:113028, 2020.

Ke Li, Kejun Tang, Tianfan Wu, and Qifeng Liao. D3m: A deep domain decomposition method for partial differential equations. *Ieee Access*, 8:5283–5294, 2019.

Xuhui Meng, Zhen Li, Dongkun Zhang, and George Em Karniadakis. Ppinn: Parareal physics-informed neural network for time-dependent pdes. *Computer Methods in Applied Mechanics and Engineering*, 370:113250, 2020.

Manfredo P Do Carmo. *Differential geometry of curves and surfaces: revised and updated second edition*. Courier Dover Publications, 2016.

Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

Benedikt Alkin, Andreas Fürst, Simon Schmid, Lukas Gruber, Markus Holzleitner, and Johannes Brandstetter. Universal physics transformers: A framework for efficiently scaling neural operators. *Advances in Neural Information Processing Systems*, 37:25152–25194, 2024a.

Emilien Dupont, Hyunjik Kim, SM Eslami, Danilo Rezende, and Dan Rosenbaum. From data to functa: Your data point is a function and you can treat it like one. *arXiv preprint arXiv:2201.12204*, 2022.

Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *Advances in neural information processing systems*, 33:7462–7473, 2020.

Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Andrew Jaegle, Sebastian Borgeaud, Jean-Baptiste Alayrac, Carl Doersch, Catalin Ionescu, David Ding, Skanda Koppula, Daniel Zoran, Andrew Brock, Evan Shelhamer, et al. Perceiver io: A general architecture for structured inputs & outputs. *arXiv preprint arXiv:2107.14795*, 2021.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020.

Benedikt Alkin, Tobias Kronlachner, Samuele Papa, Stefan Pirker, Thomas Lichtenegger, and Johannes Brandstetter. Neuraldem-real-time simulation of industrial particulate flows. *arXiv preprint arXiv:2411.09678*, 2024b.

Sifan Wang, Jacob H Seidman, Shyam Sankaran, Hanwen Wang, George J Pappas, and Paris Perdikaris. Bridging operator learning and conditioned neural fields: A unifying perspective. *arXiv preprint arXiv:2405.13998*, 2024.

Zongyi Li. Neural operator: Learning maps between function spaces. In *2021 Fall Western Sectional Meeting*. AMS, 2021.

Michael A Kraus and Konstantinos E Tatsis. Sdf-pinns: Joining physics-informed neural networks with neural implicit geometry representation. In *GNI Symposium & Expo on Artificial Intelligence for the Built World 2024*, 2024.

Samuel Burbulla. Physics-informed neural networks for transformed geometries and manifolds. *arXiv preprint arXiv:2311.15940*, 2023.

Pranav Sunil and Ryan B Sills. Fe-pinns: finite-element-based physics-informed neural networks for surrogate modeling. *arXiv preprint arXiv:2412.07126*, 2024.

Zhiwei Fang, Justin Zhang, and Xiu Yang. A physics-informed neural network framework for partial differential equations on 3d surfaces: Time-dependent problems. *arXiv preprint arXiv:2103.13878*, 2021.

Francisco Sahli Costabal, Simone Pezzuto, and Paris Perdikaris. δ-pinns: Physics-informed neural networks on complex geometries. *Engineering Applications of Artificial Intelligence*, 127:107324, 2024.

Han Gao, Matthew J Zahr, and Jian-Xun Wang. Physics-informed graph neural galerkin networks: A unified framework for solving pde-governed forward and inverse problems. *Computer Methods in Applied Mechanics and Engineering*, 390:114502, 2022.

Alec Jacobson, Daniele Panozzo, Christian Schüller, Olga Diamanti, Qingnan Zhou, N Pietroni, et al. libigl: A simple c++ geometry processing library. *Google Scholar*, 2013.

A The Partitioning Algorithm

In order to partition the manifold into subdomains to facilitate the universal autoencoder, and such that the manifold is covered by their union, we use a combination of Poisson sampling and region growing. Given a point cloud, Poisson sampling is a greedy algorithm that iteratively chooses a point and removes from the set of points that can be chosen all the points that are less than a prespecified distance from it. The algorithm stops when there are no more points that can be sampled. We then use a simple region growing algorithm using the sampled point as *seed points*. Region growing is a family of classical and widely used algorithms in computer vision and image processing for segmentation and clustering. From a set of seed points, region growing progressively assigns to a chart all the points in the neighborhood of points that are already in that chart, unless they already belong to some other chart. The algorithm iterates until all points belong to a chart. This way, charts *grow* from seed points. To ensure charts overlap, we do one additional iteration at the end.

B Universal Autoencoder: Architecture Details

The Universal Autoencoder architecture combines transformer-based point cloud encoding with a conditional neural field. The encoder component processes input point clouds by first selecting representative points and applying a graph neural network for local feature aggregation. This is inspired by the supernode pooling architecture proposed by Alkin et al. [2024a]. These features are projected to a higher dimension and refined through a series of prenorm transformer blocks. Last, we employ a perceiver-based pooling mechanism to extract latent tokens from the point cloud representation. Coordinates are encoded with a perceiver-based model [Jaegle et al., 2021]. A dedicated latent token is concatenated with the perceiver output and processed through additional transformer layers to extract a global latent code. The decoder component utilizes a modulated SIREN network [Sitzmann et al., 2020], which implements sinusoidal activation functions modulated by the extracted latent code, enabling high-fidelity reconstruction of complex signals across the coordinate space. This hybrid architecture effectively combines the strengths of transformer-based feature extraction with the continuous representation capabilities of implicit neural representations. Because the embedding of a manifold does not change if the manifold is translated, every chart is centered at the origin before feeding it to the universal autoencoder.

C Ablation Study

The ablation study involved the coil point cloud made of 117k points. 16 points were sampled randomly and ordered with respect to distance from the base point. We used increasingly more points in the training dataset following this order. For reference, the farthest point is 400 units from the base point. For more than 16 points, the performance of Δ-PINNS quickly converges to that of $\mathcal{M}$-PINNs, but in the scarce data regime, we observe that Δ-PINNs find solutions that achieve low PDE residual loss, but don't agree with the ground truth. For every number of data points we average the result over three different random seeds. We think that this difference in performance is mostly due to domain decomposition and the use of independent PINNs in every chart.

D Deformed Manifold Experiment

For the deformed manifold experiment, we trained a 2-million-parameter universal autoencoder on random charts, taken from partitioning the Stanford Bunny with region growing, and deforming them with random transformations from the family of transformations described in Eq. (20). At test time, a new deformation was sampled, and the Eikonal equation was trained using the predicted local embeddings for the new set of charts. Figure 2 shows the same experiment performed on the same coil shape used in [Costabal et al., 2024]. Note that our solution is sampled much more densely than the ground truth. This is because the continuous local embedding allows for sampling off the mesh. This is a clear advantage of the method we propose: the solution does not rely on any mesh structure, which can be costly to obtain, and can be computed using only the point cloud. For reference, both the coil and the Stanford Bunny were partitioned into 90 charts, and a single-layer neural MLP with tanh activation was used as the PINN on each chart. Domain decomposition allows for using small PINNs on each partition which compensates for the computational overhead given by the additional

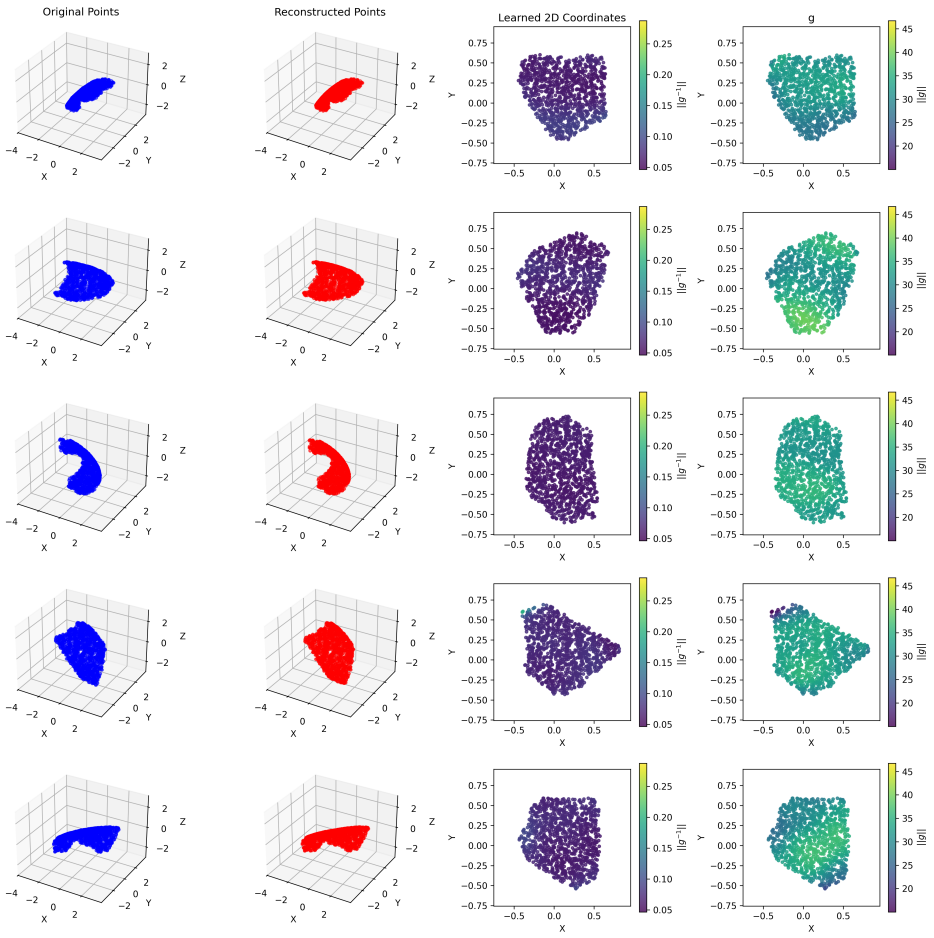


Figure 5: **Examples of charts**. First column: charts. Second column: charts reconstructed by the universal autoencoder with the predicted embedding. Third column: the 2D charts predicted by the encoder with the norm of the inverse Riemannian metric computed by pulling back the Euclidean metric through the SIREN decoder. Fourth column: the same coordinates with the norm of the Riemannian metric.

boundary term in the loss. Overall, we found that our implementation was comparable to Δ-PINN in terms of speed.

E Changing Domain Experiment

The changing domain experiments were performed on a 64-by-64 grid embedded in 3D space and deformed by a random transformation from the family

$$\mathcal{T} = \{f : \mathbb{R}^3 \longrightarrow \mathbb{R}^3, (x, y, z) \mapsto (x, y, \sum_{i=1}^{N} t_i \sin(n_i \pi x) \sin(m_i \pi y))\}, \tag{21}$$

where t_i is uniformly sampled between 0 and 1 and n_i, m_i are random integers sampled between 0 and 4. We trained a universal autoencoder to produce embeddings of this manifolds and use them to solve the diffusion equation $\mathbf{u}_t = D\Delta\mathbf{u}$ and the wave equation $\mathbf{u}_{tt} = c^2\Delta\mathbf{u}$. A roll-out of the diffusion equation is reported in Fig. 1 while in Fig. 3 and 4 we show roll-outs of the wave equation. For the wave equation, we use a Gaussian-like initial condition at the center of the grid with derivative zero everywhere. The PDE is solved while the domain is being continuously deformed by a transformation from the same family: a n_i and m_i are kept fixed and t is continuously varied from 0 to 1. Crucially, Δ-PINN would require re-computing the eigenfunctions of the Laplace-Beltrami operator every

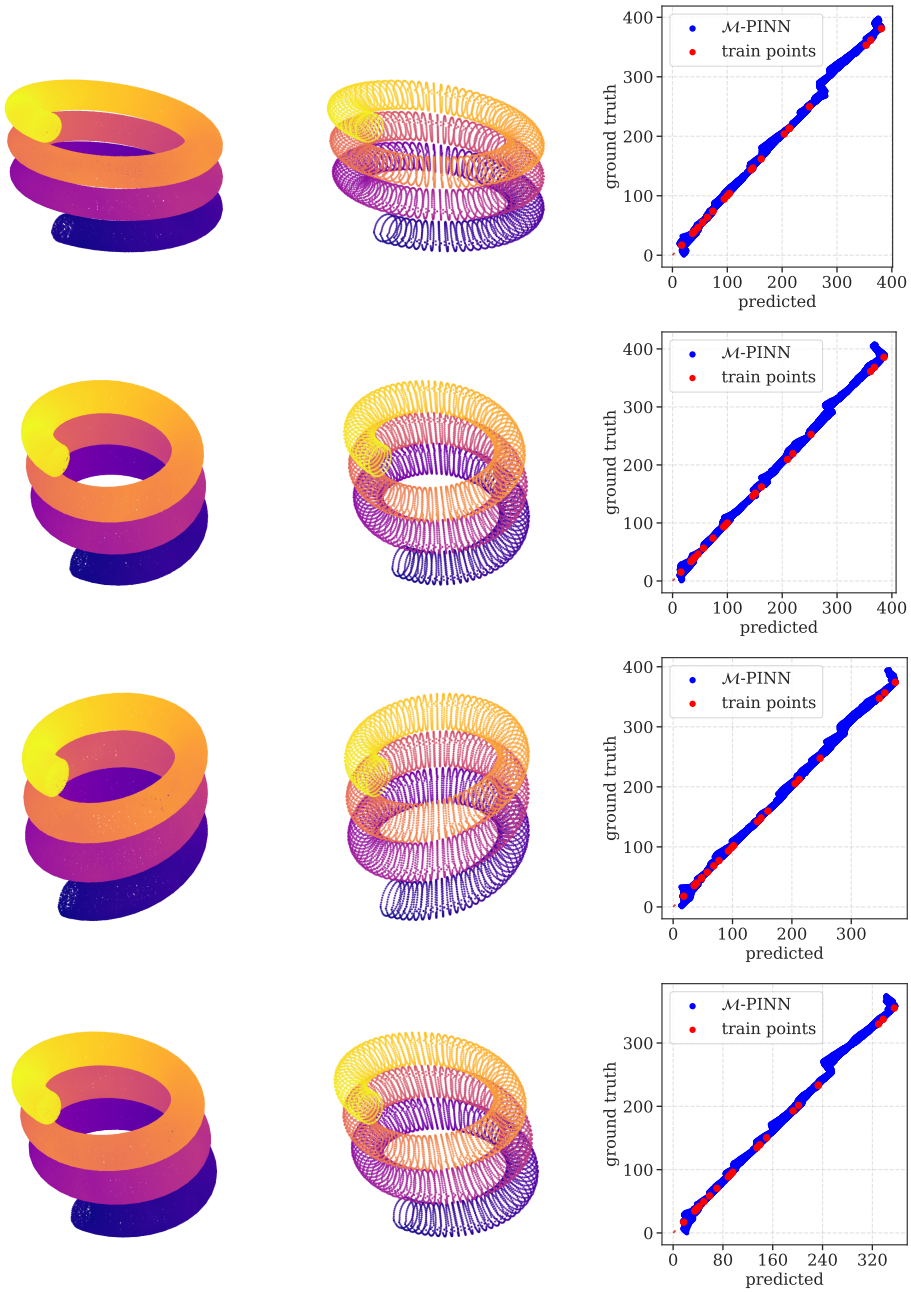


Table 2: **Solution of the Eikonal equation on random deformations of a coil.** First column: the solution obtained using the PINNs trained with local embedding obtained with the universal autoencoder. The solution is sampled much more densely than the original point cloud. Second column: the ground truth obtained with an exact geodesic algorithm using the original mesh structure. Third column: the correlation between the ground truth solution and the PINN solution obtained by using the local embeddings from the universal autoencoder.

iteration, which is very costly. The universal autoencoder requires a single, differentiable forward pass to obtain the local embeddings.

All experiments were performed on a single H100 GPU, using Adam optimizer with linear learning rate warm-up followed by cosine decay. Training took 2 hours for the square domain and 10 hours for the experiments with the Stanford Bunny and the coil. The code for reproducing the experiments can be found at https://anonymous.4open.science/r/manifold-pinns-universal-autoencoders-7057.

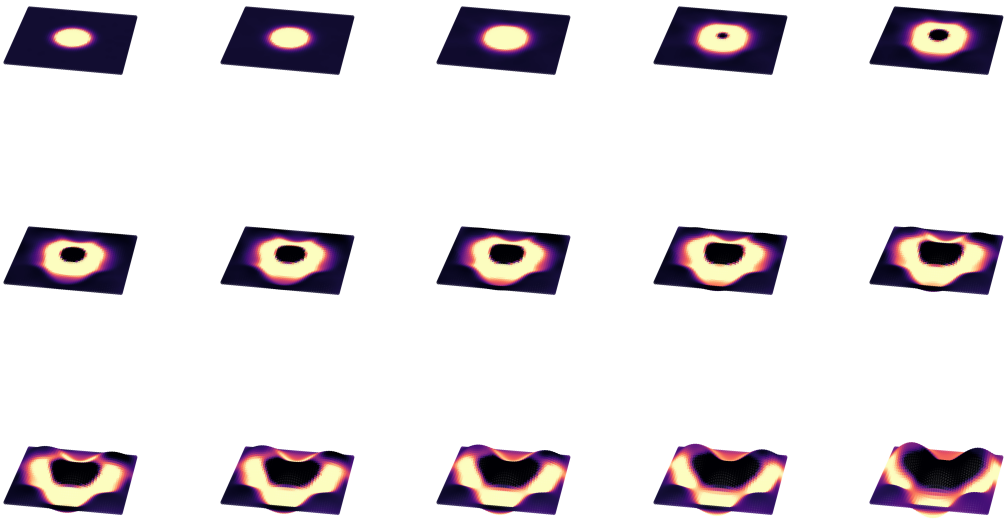


Table 3: **Solution of the wave equation on a domain that changes in time.**

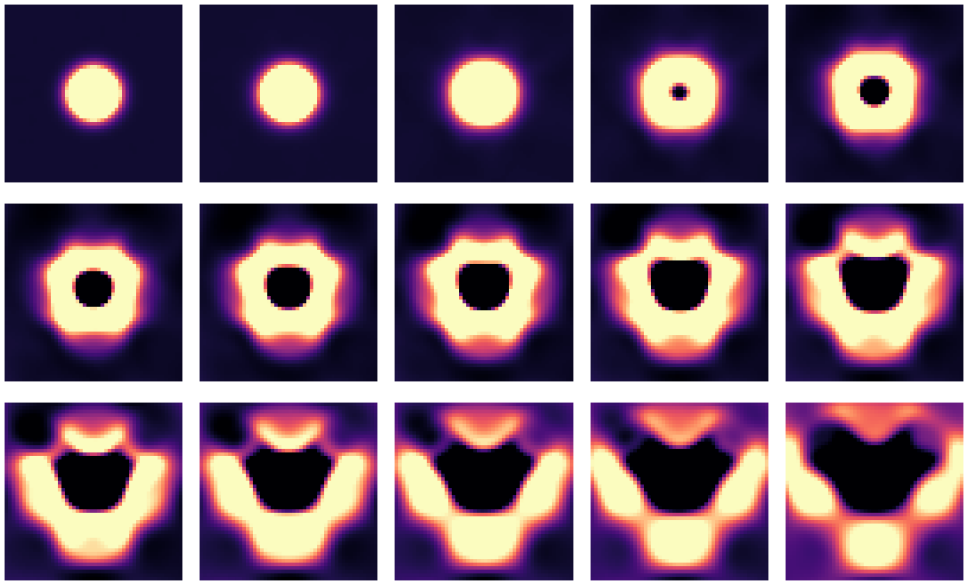


Table 4: **Projection of the solution of the wave equation on a domain that changes in time.**