DOI: 10.1111/cgf.14549
Eurographics Conference on Visualization (EuroVis) 2022
R. Borgo, G. E. Marai, and T. Schreck
(Guest Editors)

Volume 41 (2022), Number 3

Neural Flow Map Reconstruction

S. Sahoo, Y. Lu & M. Berger
Vanderbilt University

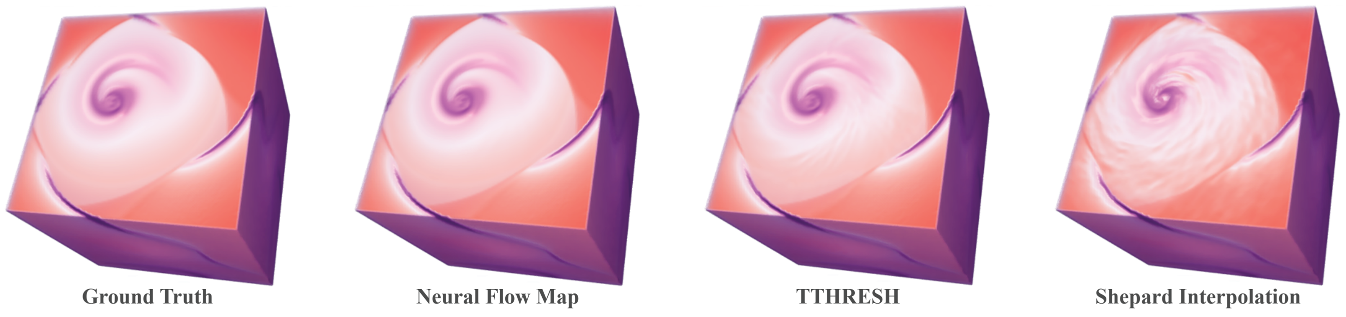

Figure 1: *We show a comparison of our method, Neural Flow Map, with existing time-varying vector field data reduction schemes for compression (TTHRESH) and scattered data interpolation (Shepard Interpolation). We show a volume rendering of the FTLE for the* Tornado *dataset for the recovered time-varying vector fields found by each method. Our proposed approach optimizes a neural network to recover flow map samples, leading to strong generalization in the flow map, and consequently, faithful recovery of derived quantities such as FTLE.*

Abstract

In this paper we present a reconstruction technique for the reduction of unsteady flow data based on neural representations of time-varying vector fields. Our approach is motivated by the large amount of data typically generated in numerical simulations, and in turn the types of data that domain scientists can generate in situ *that are compact, yet useful, for post hoc analysis. One type of data commonly acquired during simulation are samples of the flow map, where a single sample is the result of integrating the underlying vector field for a specified time duration. In our work, we treat a collection of flow map samples for a single dataset as a meaningful, compact, and yet incomplete, representation of unsteady flow, and our central objective is to find a representation that enables us to best recover arbitrary flow map samples. To this end, we introduce a technique for learning implicit neural representations of time-varying vector fields that are specifically optimized to reproduce flow map samples sparsely covering the spatiotemporal domain of the data. We show that, despite aggressive data reduction, our optimization problem – learning a function-space neural network to reproduce flow map samples under a fixed integration scheme – leads to representations that demonstrate strong generalization, both in the field itself, and using the field to approximate the flow map. Through quantitative and qualitative analysis across different datasets we show that our approach is an improvement across a variety of data reduction methods, and across a variety of measures ranging from improved vector fields, flow maps, and features derived from the flow map.*

CCS Concepts
• ***Computing methodologies*** $\rightarrow$ ***Neural networks; Reconstruction;*** • ***Human-centered computing*** $\rightarrow$ ***Scientific visualization;***

1. Introduction

In the field of scientific computing, scientists usually generate enormous amounts of data from large scale numerical simulations. Such simulations are often run in networked high performance computing (HPC) environments where disk storage and memory capacity is abundant. Traditionally, during the simulation run, time slices (simulation state) are periodically saved out to disk for post hoc analysis. In the ideal scenario where scientists have access to all of the time slices saved during the simulation, scientists usually use particle advection to derive various quantities such as pathlines, the finite-time Lyapunov exponent (FTLE) [Hal01], and Lagrangian coherent structure (LCS) [Hal00, HY00] to help make sense of the data. However in practice, scientists have limited access to the vector field data since most visualization systems are run on local workstation. Consequently, when transferring data from the HPC to local workstations, the network bandwidth, limited memory and storage space collectively act as a bottleneck to data access.

To mitigate this problem, numerous methods have been developed to reduce the size of time-varying data, e.g. compression methods[BRLP19] and superresolution techniques [GYH*20], with some methods specifically targeting vector field data [HW19]. These methods can alleviate the burden of network bandwidth limitations for more efficient data I/O transfer, e.g. transferring compressed/subsampled data from an HPC device to a local workstation. However, these techniques are designed to work with an Eulerian-based form of data and do not take into consideration how the vector fields are actually used in practice. On the other hand, Agranovsky et. al. [ACG*14], showed that Lagrangian flow map representations - where particles are integrated in-situ and only the start and the end positions of the particles are saved - gives a more accurate and efficient storage for performing post hoc analysis as compared to using discretized vector field time slices. Thus, developing techniques that can make use of these representation is an attractive option. Deep learning based techniques have proven to be a promising method for vector field superresolution [GYH*20], flow map superresolution [JGG20], compression [LJLB21] and a variety of other tasks. In this work, we wish to leverage these optimization based deep learning techniques along with a Lagrangian-based form of data to support a more accurate post hoc Lagrangian transport analysis.

To this end, we propose a novel method for representing and recovering unsteady flow that enables substantial data reduction in time-varying vector fields. Our approach takes on a hybrid Eulerian-Lagrangian viewpoint. Our representation of unsteady flow is Eulerian, in that we leverage implicit neural representations [SMB*20] to model time-varying vector fields. These are coordinate-based neural networks which, in our scenario, take as input spatiotemporal coordinates, and output vectors. The manner in which we optimize these representations, however, is Lagrangian. Our method assumes that a small set of flow map samples from the underlying field have been provided, and we optimize our implicit neural representation to best reproduce these flow map samples. This necessitates an optimization scheme that can perform integration with respect to our Eulerian representation. We show how this integration-based optimization can lead to neural flow-based representations that, both, faithfully recover the original time-varying vector field, as well as its flow map, when provided with just a sparse set of flow map samples. We show superior performance in comparison to Eulerian, and Lagrangian, unsteady flow data reduction approaches. Further, the Lagrangian-based optimization view of our method results in the faithful representation of derived unsteady flow quantities, such as FTLE, as demonstrated in Fig. 1. Additionally, our coordinate-based neural representation removes the need to maintain grid representations, allowing one to compute these quantities in a random-access manner.

Our main contributions can be summarized as follows:

1. We introduce a novel approach to learn a vector field representation only using samples from a flow map.
2. We study how different factors concerned with flow field integration affect the training process.
3. We evaluate our approach on a range of vector field datasets and demonstrate superior results compared to existing methods, both quantitatively and qualitatively.

2. Related Work

Our approach is closely related to existing works on data reduction for unsteady flow. Here we discuss the most pertinent works on compression, superresolution, and interpolation, in addition to recent work on learning over differential equations.

Compression-based data reduction. Given that 3D unsteady flow is often in an Eulerian representation, e.g. a time-varying vector field, a common method for data reduction is lossy compression. Although volumetric compression of scalar fields has a long history within scientific visualization [Mur93, GWGS02], with numerous approaches specifically targeting time-varying data [SW03, Ma03, JEG11], exploiting the redundancy between (1) space, (2) time, and (3) vector components presents challenges for effective reduction. One common strategy is to flatten the data as a 1D function, and approximate the function through different fits, e.g. B-splines [LSE*11] or more adaptive fitting [DC16]. Highly-compressive representations can also be obtained by treating the data as a tensor, e.g. a 5-tensor for space, time, and vector component, and performing a tensor-based decomposition of the data [SGM*11, ABK16]. TTHRESH [BRLP19] is a notable tensor compression method that exploits the fast decay in transform coefficients, achieved via adaptively thresholding and quantizing these coefficients for significant compression gains. Recent work considers fitting implicit neural representations [SMB*20, TSM*20] to time-varying volumetric scalar fields [LJLB21], with compression achieved by controlling the network size, and weight quantization. These existing methods seek a direct compression of the data, whereas our work decouples the compressive representation, e.g. neural represention of a time-varying vector field, from the provided data, e.g. samples of flow map.

Superresolution methods for data reduction. Another common form of data reduction for vector fields is superresolution, where the objective is to upsample a low-resolution vector field to a high-resolution vector field. Different formulations have been considered, wherein downsampling is restricted to the spatial domain [GYH*20, SB21], as well as the spatiotemporal domain [HW19, HZCW21]. Deep learning-based superresolution schemes [DLHT15, DLT16] based on convolutional neural networks are commonly employed in these scenarios, where the assumption is that patterns in downsampled data can be exploited to appropriately resolve details in the upsampled data. Most relevant to our method is the approach of Jakob et al. [JGG20], where the focus is on flow map superresolution. In all such methods, there exists a reliance on training data – paired low-resolution and high-resolution vector field, or flow map, examples. As such, at inference time, if there exists a domain mismatch between train and test data then superresolution methods are less effective. Our approach does not rely on a collection of vector field examples, but rather, assumes just a single dataset. We show that we can obtain superior performance over superresolution schemes by shifting the focus on what data we collect, and optimize. Closely related to superresolution are works that aim to reconstruct the vector field from a sparse set of scattered data. Prior works [SFT*22, EMY*20, YWS*21] achieve flow field reconstruction by directly optimizing for the velocity vectors. Han et al. [HTZ*19] and Gu et al. [GHCW21] reconstruct vector fields from a set of representative streamlines. The work done by Han et al. [HSJ21] has similar goals to ours; however their approach uses a neural network that predicts flow maps directly. In our approach, we learn a function-space representation of the flow field. Further, we explicitly impose an integration scheme to ensure a more stable and guided optimization process.

Lagrangian methods for interpolation. Lagrangian representations of unsteady flow are a popular scheme for data reduction, due to their compact representation for post hoc analysis. One common type of representation is Lagrangian coherent structure (LCS), obtained as ridges in the largest finite time Lyapunov exponent (FTLE) [Hal00, HY00]. LCS is commonly used to identify geometric structures, as well as stable and unstable manifolds. Techniques like FTLE and LCS provide a powerful framework to the user for visualizing and analyzing the flow behavior of an unsteady vector field. Fundamental to these techniques are the computation of integral curves, as a means of obtaining flow map samples. However, since the computation of integral curves can be expensive, in the literature fast computation of integral curves, and efficient sampling of flow map has received significant attention. Burton et al. [BR10] introduced a fast FTLE computation by concatenating multiple flow maps. Garth et al. [GGTH07] introduced an incremental smooth approximation of flow map computation for fast FTLE computation. Similarly, Sadlo et. al [SP07] introduced an adaptive mesh refinement technique wherein seeding is avoided from regions where no ridges are present. Sadlo et al. [SRP11] introduced grid advection for efficient FTLE computation. Rapp et al. [RPD19] introduced a sampling strategy where the samples exhibit blue noise property and thus gives a better domain coverage as compared to random sampling. In the literature, post hoc Lagrangian reconstruction has also received signficant attention. Hlawatsch et al. [HSW10] introduced a technique for fast computation of integral curves by utilizing short integral curves in a hierarchical manner to construct longer curves.

The computation of FTLE and LCS relies on access to the underlying field's flow map, in order to numerically estimate the Cauchy-Green tensor in FTLE [SLM05a]. The underlying time-varying vector field is a natural representation from which to derive the flow map, but for large-scale numerical simulations, this might not be feasible to collect and store. On the other hand, the computation of flow map samples *in situ*, and post hoc storage, is more economical [ACG*14]. Hence, numerous methods for flow map interpolation have been developed, e.g. moving least squares interpolation [AGJ11] of densely-sampled integral curves, interpolation based on smoothed particle hydrodynamics [COJ14], barycentric interpolation [ACG*14], as well as varying duration and variable placement interpolation scheme [SCB]. The error in interpolation has been further studied [CBJ16, HBJG16], establishing a relationship between errors in interpolation with numerical integration of flow fields. In our approach, rather than impose an interpolant, we use coordinate-based neural networks whose derived integral curves fit to given flow map samples.

Neural differential equations. Our approach is closely related to work on neural ordinary differential equations (ODE) [CRBD18]. Indeed, the computation of the flow map is one of solving an ODE-based initial-value problem, and we utilize the adjoint sensitivity method [Pon87] for memory-efficient optimization. Nevertheless, the main distinction in our approach is that we do not solve an ODE in the learned representation space of a neural network [CRBD18, RCD19], but rather, directly in the spatiotemporal domain. Adjoint methods for scalable optimization have also seen recent adoption within differentiable direct volume rendering [WW21]. Viewed more broadly, our approach is similar in spirit to neural radiance fields (NERF) [MST*20, LGZL*20], in that we are attempting to recover field-based data from an otherwise unobserved process – for NERF this corresponds to density/radiance fields from volume-rendered images, while our method aims to recover unsteady flow fields from flow map samples.

3. Approach

We describe our approach in this section, organized by the type of data our approach assumes, a description of our neural field representation, and details on our approach for optimization.

3.1. Data Reduction: Flow Map

There are, fundamentally, two different ways in which to represent unsteady flow data: Eulerian representations, and Lagrangian representations. Within the context of fluid dynamics, the Eulerian representation of flow describes the time-variant instantaneous velocity of fluid particles at fixed spatial locations in the domain. Typically, an Eulerian specification of unsteady flow manifests as a time-varying vector field, which maps a given spatial position and (nonnegative) time value to a vector:

$$\nu : \mathbb{R}^d \times \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}^d, \tag{1}$$

and we assume $d \in \{2,3\}$. In practice, we are given a *sampling* of ν, usually sampled on a regular grid with spatial resolution $(s_x \times s_y \times s_z)$ and T time steps. As previously discussed, for the purposes of data reduction, Eulerian representations are typically (1) compressed, or (2) downsampled, and then upsampled on-demand via superresolution methods.

In contrast, in this work we use the Lagrangian representation of flow as a form of data reduction. The Lagrangian viewpoint describes the underlying motion of flow as a set of massless particles in the domain that travel through space and time. The mathematical object that represents how a particle is transported under the flow field, starting at a given spatial position, time step, and for a given duration, is known as the *flow map*. The flow map can be constructed by, first, defining how a particle $\mathbf{x}(t)$ traveling in the domain is advected by the flow field, governed by the following ordinary differential equation:

$$\frac{d\mathbf{x}(t)}{dt} = \nu(\mathbf{x}(t), t), \quad \mathbf{x}(t_0) = \mathbf{p}, \tag{2}$$

where the initial condition on the right-hand side specifies the particle's initial position $\mathbf{p}$ at time t_0. Secondly, the particle's advection for a duration δ can then be found via integration:

$$\Phi_{t_0}^{t_0+\delta}(\mathbf{p}) = \mathbf{x}(t_0) + \int_{t_0}^{t_0+\delta} \nu(\mathbf{x}(t), t)dt. \tag{3}$$

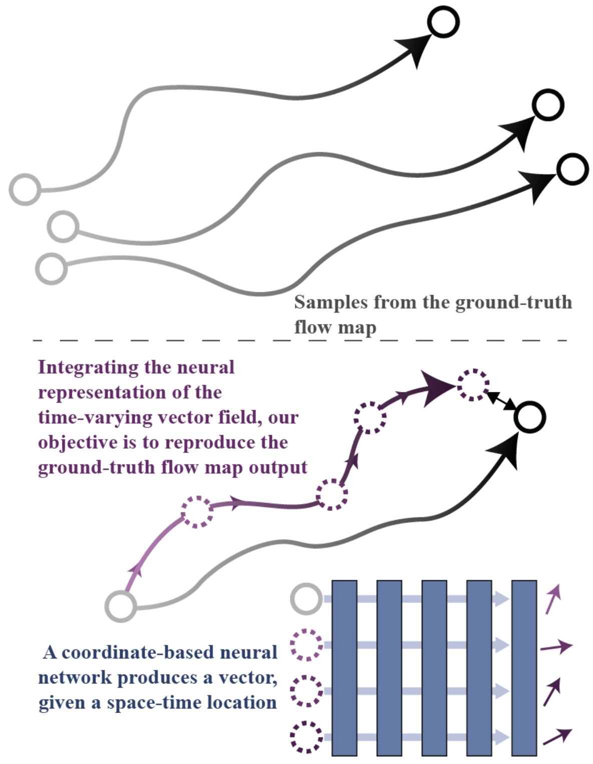
Samples from the ground-truth flow map
Integrating the neural representation of the time-varying vector field, our objective is to reproduce the ground-truth flow map output
A coordinate-based neural network produces a vector, given a space-time location

Figure 2: *We show an overview of our approach. Our method assumes samples of a flow map for optimization (top), namely the starting point, start time, and end point, being the result of integration. Given a single sample of a flow map (bottom), our method aims to learn a neural representation of a time-varying vector field that, upon integration, can recover the output of the flow map. The brightness of integral curves encodes time.*

The flow map Φ is an important mathematical object for the analysis and visualization of unsteady flow phenomena, e.g. for extracting Lagrangian coherent structure [HY00] and for the visual analysis of attracting and repulsive behaviors [RGG19]. Prior work on data reduction typically takes a collection of integral curves, each curve being *densely sampled in time*, from which to then interpolate the flow map at arbitrary points in space-time. In contrast, in our work we do not perform such a dense sample; rather, we assume substantially less information for each item in our dataset: (1) an initial position $\mathbf{p}$, (2) the starting time of advection t, and (3) the result of applying the flow map $\Phi_t^{t+\delta}(\mathbf{p})$. For simplicity, in our work we assume that the duration δ is fixed as a constant, though this restriction can easily be relaxed. All told, our method assumes the following dataset as input:

$$\mathcal{T} = \left\{ \left(\mathbf{p}_1, t_1, \Phi_{t_1}^{t_1+\delta}(\mathbf{p}_1)\right), \dots, \left(\mathbf{p}_n, t_n, \Phi_{t_n}^{t_n+\delta}(\mathbf{p}_n)\right) \right\}, \quad (4)$$

namely, we assume n total flow map samples from Φ. As an example, in Fig. 2(top) this corresponds to the start and end points of integral curves.

Algorithm 1 Pseudocode for memory-efficient backpropagation under explicit Euler integration

```
1: Input: spatial position $\mathbf{p}$ , time $t_0$ , duration $\delta$, integration step size $\varepsilon$
2: Form a time-dependent particle $\tilde{\mathbf{x}}(t)$ under the neural vector field through solving Eq. 6 under explict Euler integration
3: Initialize position gradient $\mathbf{a} = \frac{dL}{d\tilde{\mathbf{x}}(t_0+\delta)}$
4: Initialize weight gradient $\frac{dL}{d\theta} = 0$
5: for $t = (t_0+\delta-\varepsilon)$ to $t_0$ in increments of $\varepsilon$ do
6:     Integrate weight gradient $\frac{dL}{d\theta} = \frac{dL}{d\theta} - \varepsilon \mathbf{a}^T \frac{\partial f_\theta(\tilde{\mathbf{x}}(t),t)}{\partial\theta}$
7:     Integrate position gradient $\mathbf{a} = \mathbf{a} - \varepsilon \mathbf{a}^T \frac{\partial f_\theta(\tilde{\mathbf{x}}(t),t)}{\partial\tilde{\mathbf{x}}(t)}$
8: end for
9: Return $\frac{dL}{d\theta}$
```

3.2. Neural Flow Map

The objective for our approach is to learn a model of unsteady flow that provides us with the following:

1. The model provides for an effective Eulerian representation of flow, e.g. it is a good approximation of the ground-truth vector field ν.
2. The model provides for an effective Lagrangian representation of flow, e.g. it allows us to reproduce the given dataset of flow map samples, $\mathcal{T}$, whilst generalizing to arbitrary samples of the flow map.

To this end, our model takes on an Eulerian reference frame, namely, we utilize neural representations of fields [SMB*20, TSM*20], in particular those that are time-varying [LJLB21, XHKK21]. In our problem, this amounts to a neural network that takes as input a spatiotemporal location, and outputs a vector. We denote this as a function, f, parameterized by a set of weights $\theta \in \mathbb{R}^D$, such that $f_\theta : \mathbb{R}^d \times \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}^d$. The function f_θ is a multi-layer perceptron with sinusoidal activation functions; we defer architecture details to Sec. 5.1. Ideally, we would like f_θ to be as close as possible to ν.

The manner in which we optimize for f_θ, however, takes on a Lagrangian frame of reference, please see Fig. 2 for an overview. Specifically, for a given spatial position $\mathbf{p}$, time t_0 and duration δ, we define our *neural flow map* as follows:

$$\tilde{\Phi}_{t_0}^{t_0+\delta}(\mathbf{p}) = \tilde{\mathbf{x}}(t_0) + \int_{t_0}^{t_0+\delta} f_\theta(\tilde{\mathbf{x}}(t), t)dt, \quad (5)$$

where a particle $\tilde{\mathbf{x}}(t)$ is governed by the following ODE:

$$\frac{d\tilde{\mathbf{x}}(t)}{dt} = f_\theta(\tilde{\mathbf{x}}(t), t), \quad \tilde{\mathbf{x}}(t_0) = \mathbf{p}. \quad (6)$$

Given samples of our flow map $\mathcal{T}$, we wish to find a neural representation of the time-varying vector field that minimizes the following equation:

$$\frac{1}{|\mathcal{T}|} \sum_{(\mathbf{p},t,\mathbf{q}) \in \mathcal{T}} \|\tilde{\Phi}_t^{t+\delta}(\mathbf{p}) - \mathbf{q}\|_2^2, \quad (7)$$

e.g. the output of the neural flow map is a good approximation of the *actual* flow map output $\mathbf{q} = \Phi_t^{t+\delta}(\mathbf{p})$, where "good" is measured in terms of their squared Euclidean distance, though in principle any differentiable loss function could be used. Note that our neural representation f_θ can be evaluated at *arbitrary* points in space and time. This obviates the need to interpolate from a sampled vector field, and further, permits us to optimize over a set of flow map samples $\mathcal{T}$ whose positions and times originate at arbitrary locations.

In practice, to approximate the neural flow map in Eq. 5, it is necessary to select a numerical integration scheme. For simplicity, we use an explicit Euler integrator with sufficiently small step size to mitigate global truncation error. Thus, we can write a particle advected under our neural field representation f_θ as follows:

$$\tilde{\mathbf{x}}(t+\varepsilon) \approx \tilde{\mathbf{x}}(t) + \varepsilon \cdot f_\theta(t), \tag{8}$$

for an appropriately-defined step size ε.

3.3. Efficient Backpropagation

An immediate computational problem arises from naively optimizing Eq. 7 under standard reverse-mode automatic differentiation. Namely, in order to compute gradients of the loss with respect to weights θ, we must record all activations produced by the neural network f_θ, for *each* step taken in our integration scheme (Eq. 8). For large-scale datasets, and batch-based optimization, this scheme quickly overwhelms the amount of memory necessary to perform backpropagation.

To address this challenge, we take advantage of the adjoint sensitivity method for ODEs, as proposed in Chen et al. [CRBD18]. In our setting, we are primarily concerned with efficiently computing the gradient over all examples from Eq. 7, which we express as:

$$\frac{1}{|\mathcal{T}|}\sum_{(\mathbf{p},t,\mathbf{q})\in\mathcal{T}} \nabla_\theta L(\tilde{\mathbf{x}}(t+\delta),\mathbf{q}), \tag{9}$$

where L is the loss for a single flow map sample as in Eq. 7, and $\tilde{\mathbf{x}}(t+\delta) = \tilde{\Phi}_t^{t+\delta}(\mathbf{p})$. We can formulate a memory-efficient gradient computation via "continuous backpropagation", considering the following for $\varepsilon > 0$:

$$\mathbf{a}(t) = \frac{dL}{d\tilde{\mathbf{x}}(t)} = \frac{dL}{d\tilde{\mathbf{x}}(t+\varepsilon)} \cdot \frac{d\tilde{\mathbf{x}}(t+\varepsilon)}{d\tilde{\mathbf{x}}(t)}, \tag{10}$$

that is, the derivative of the loss with respect to the particle's position at time t can be computed based on the flow map spatial Jacobian at the *subsequent* time step, $t+\varepsilon$. It is straightforward to show [CRBD18] that this condition, as well as Eq. 6 leads to the following *backward* ODE, starting from the end of integration:

$$\frac{d\mathbf{a}(t)}{dt} = -\mathbf{a}(t)^T \frac{\partial f_\theta(\tilde{\mathbf{x}}(t),t)}{\partial \tilde{\mathbf{x}}(t)}, \tag{11}$$

where the second term is the Jacobian of the neural vector field. By integrating forward to the full duration δ, we obtain $\mathbf{a}(t+\delta)$ which serves as our initial condition in this ODE, and consequently, can solve for $\mathbf{a}$ through numerical integration. Introducing an autonomous ODE by prescribing differential equations on θ and t:

$$\frac{d\theta}{dt} = 0, \quad \frac{dt}{dt} = 1, \tag{12}$$

and using the same reasoning above in Eq. 11, we have a second ODE to find our primary quantity of interest, the gradient of the loss:

$$\frac{dL}{d\theta(t)} = -\mathbf{a}(t)^T \frac{\partial f_\theta(\tilde{\mathbf{x}}(t),t)}{\partial \theta}. \tag{13}$$

In principle, the gradient computation can be applied to any numerical integration scheme. In our case, for explicit Euler integration, the formulation leads to a straightforward algorithm that we summarize in Listing 1 for a single flow map sample; batch computation is straightforward to extend. Note that this requires the computation of two types of gradients of the neural vector field per integration step: (1) weight gradient, and (2) positional gradient. This amounts to 2 applications of vector-Jacobian multiplication, a basic operation in modern deep learning libraries.

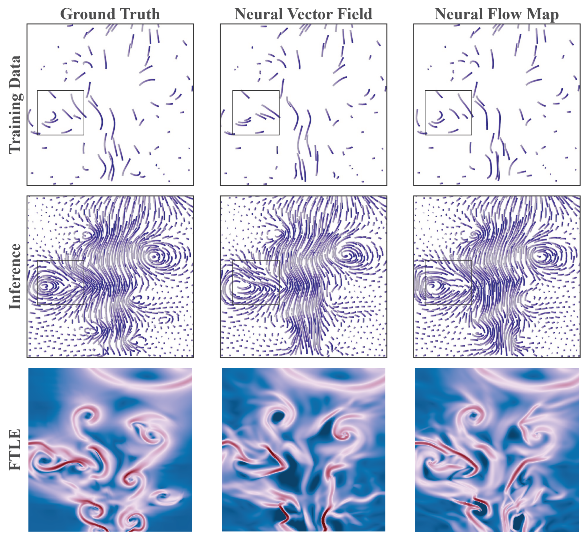

Figure 3: *We show the ability of our method to reproduce integral curves given a sparse set of flow map samples (top-left). The top row shows all integral curves within a particular time range from which flow map samples for training were derived, the middle row demonstrates generalization on withheld space-time seeds for integration, and the bottom row shows the FTLE. We find that optimization of the flow map (right) better captures swirling features in this Heated Cylinder dataset compared to directly fitting to vectors (middle).*

3.4. Fitting to Flow Map Samples, Fitting to Vectors

4. Analysis

In this section, we show through experimentation an analysis of different factors that effect flow map-based optimization. Specifically, our approach necessitates the use of an numerical integration scheme. Thus we are left with a number of questions, namely, is Euler integration sufficient, in practice? How do we sample seed points to generate the flow maps? For how long should we integrate? To answer these question, we perform a study to determine and justify the choices we make as part of the numerical integration scheme.

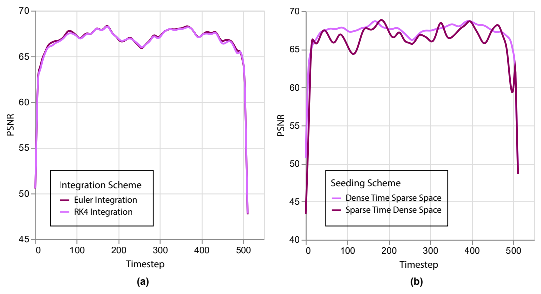

Figure 4: *We show the vector field PSNR across all the timesteps of the* Double Gyre *dataset reconstructed using flow map samples under Euler and RK4 integration scheme (a) and different seeding scheme (b). We find the performance to be comparable, thus motivating our choice for a simple, explicit Euler scheme for optimization.*

We first qualitatively highlight the benefits of learning a vector field via flow map-based optimization, compared to the more traditional baseline of directly fitting to vectors. Specifically, we have taken the Heated Cylinder dataset and generated 128K flow map samples, sampling uniformly at random over space-time, with integration duration $\delta = 0.15$. For comparison, we fit an implicit neural representation [SMB*20, TSM*20, LJLB21] to the vectors at the initial seeds of these flow map samples. Thus, both methods – directly fitting a vector field, optimizing for a flow map – receive different, but the same amount of, information.

Fig. 3 shows the results. The top row shows the integral curves from which the flow map samples were computed, namely, all such flow map samples that pass through time $t = 6.23$. Note the sparsity in the training data (top-left). We find that our method (top-right) is able to fit well to these training flow map samples, as one would expect assuming optimization is successful. For the vector field-based fit (top-middle), however, we find that integrating this vector field at these seeds results in sub-optimal integral curves. The ramifications of this on the rest of the data are presented in the middle row, where we take an arbitrary, dense set of seeds at the aforementioned time, and compute integral curves with the same duration $\delta = 0.15$. Here, we find that at inference, flow map-based optimization leads to vector fields whose integral curves better capture features in the dataset, e.g. swirling motion due to vortices shed throughout this simulation, in comparison to methods that just fit to the vectors directly. The bottom row of Fig. 3 compares the FTLE for an integration duration of $\delta = 0.3$. Given the extreme sparsity of samples we do not expect a high-quality FTLE approximation, but nevertheless, we can see, qualitatively, that flow map-based optimization tends to better capture the FTLE over fitting to vectors. We defer a more exhaustive comparison of varying levels of subsampling for this dataset as supplemental material.

4.1. Effect of Integration Scheme

In order to understand the impact of different integration schemes, we experiment with two of the most widely used integration schemes - namely Euler integration and 4th order Runge-Kutta (RK4) integration. Euler integration, albeit sensitive to the step size, is simple and fast. On the other hand 4th order Runge-Kutta integration is more robust and accurate. In our experimental setup, we use the Double Gyre dataset [SLM05b], and take flow map samples with fixed integration duration of $\delta = 10$ and a step size of 0.1, originating from every timestep excluding the last 10 timesteps for both integration schemes. Fig 4(a) shows the results of training using Euler and RK4 integration. Clearly, with sufficiently small step-size Euler integration performs on par with RK4 integration scheme and in the meanwhile being significantly faster to train. The model under Euler integration scheme was trained in about 97 minutes whereas the model under RK4 integration scheme took about 152 minutes. Thus for the remainder of the paper we use Euler integration scheme because of its simplicity, faster training speed and accuracy as compared to RK4 integration.

4.2. Effects of Sampling

The way that we sample flow maps is important to ensure that flow feature are captured by the pathlines, enabling efficient learning. Since we are working with a substantially reduced amount of data

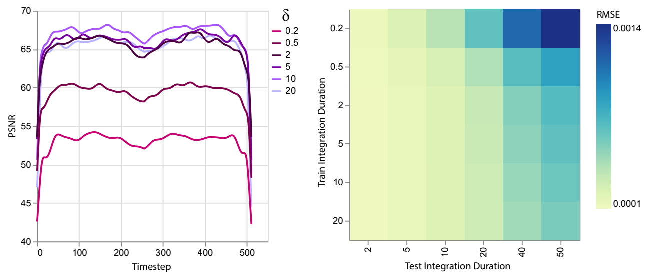

Figure 5: *We show the vector field PSNR (left) across all timesteps for the* Double Gyre *dataset for different models trained with flow map samples of varying integration duration. We show the generalization capabilities of the models to integration durations that were not trained on as a heatmap (right).*

Table 1: *We list the datasets used for experimental comparisons, along with their size, the integration duration we use for flow maps, and the sampling reduction rate.*

Dataset Name	Dimensions ($x \times y \times z \times t$)	δ	Reduction Rate
Four Rotating Centers (2D)[GGT17]	128x128x512	10	16x
Double Gyre (2D)[SLM05b]	256x128x512	10	16x
Fluid Simulation (2D)[JGG20]	512x512x1001	5	16x
Tornado	128x128x128x50	3	100x
Isabel	500x500x90x48	3	300x
ScalarFlow[EUT19]	100x178x100x150	5	300x
Half Cylinder[RG19]	640x240x80x151	3	300x

Table 2: *We list the total training time, inference time (time taken to integrate 10,000 particles for a duration of 20 in grid-time) and the model-size for all the 3D datasets.*

Dataset	Training Time (in minutes)	Inference Time (in seconds)	Model Size (in MB)
Tornado	226	5.939	2.1
ScalarFlow	128	5.740	1.7
Isabel	487	15.432	8.2
Half Cylinder	609	22.349	13.5

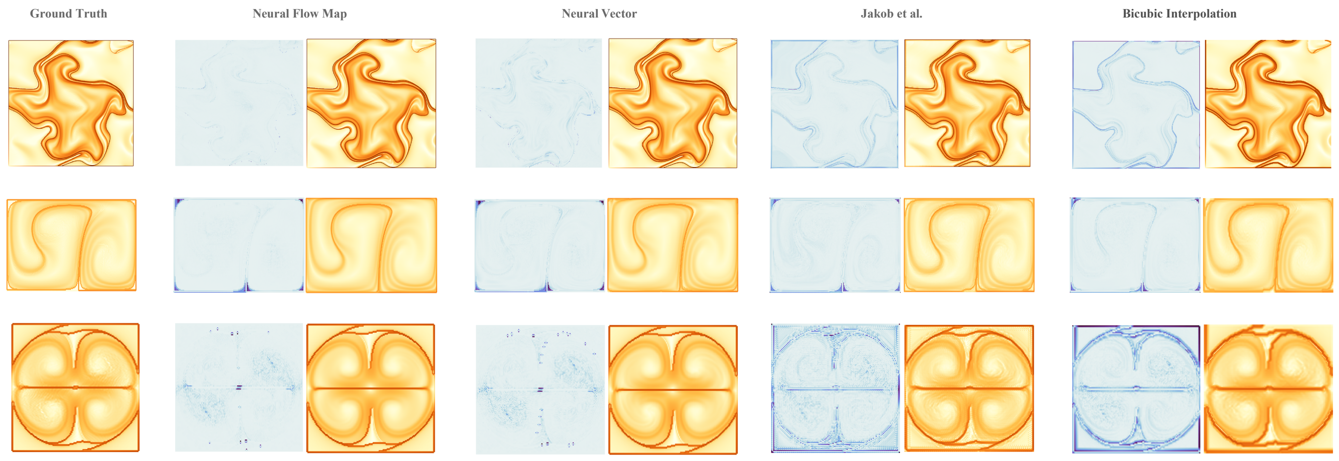

Figure 6: *We qualitatively compare our method – Neural Flow Map – to various baselines (columns) across different 2D datasets (rows). We show (1) difference images between the approximated FTLE and the ground truth FTLE, and (2) the FTLE (computed by integrating for a duration of 300, 500, 500 for fluid simulation, double gyre and four rotating centers respectively). We find that our flow map method yields improved performance across all baselines.*

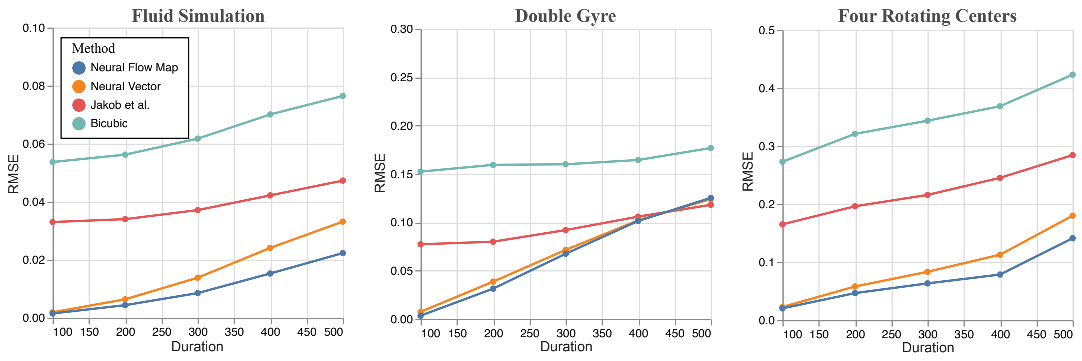

Figure 7: *We show quantitative results for our 2D experiments, where we measure the RMSE of flow map error across different integration durations. Across all baselines, we observe consistent improvements using our flow map-based method – note, despite the fact that our method optimizes on a single duration, it is nevertheless able to generalize to different (longer) durations.*

we can only cover so much of the dataset. To this end, we propose two different sampling schemes, namely sparse-time dense-space - wherein we sample densely in space for a given timestep and consider only a subset of the total timestep sparsely chosen, and dense-time sparse-space - here we give more importance to the temporal frequency over spatial frequency given the same budget of samples. In Fig 4(b), we can see the results of vector field PSNR. In this experiment, we used a total of 168K samples from the Double Gyre dataset - sampled as 336 spatial points across 500 timesteps and 1680 spatial points sampled across 100 timesteps. We observe that dense temporal sampling shows that temporal coverage is more important and gives better results as compared to sparse temporal sampling. Thus, in the remainder of the paper we use the dense-time sparse-space sampling scheme, to ensure better temporal coverage.

4.3. Integration Duration

The duration for which the particles are integrated to generate the flow map is an important factor for the training process. When integrated for extremely small duration, then the scenario resembles that of vector-based optimization, since the network need only predict the immediate vectors in close proximity of seed location to yield an accurate flow map. We hypothesize that in such cases the model would not be able to take full advantage of the flow map based optimization process and thus perform poorly. To confirm our hypothesis, we design an experiment where we train different models on flow map samples generated with increasing integration duration. For this experiment, we use the Double Gyre dataset, and we generate the flow map samples using Euler integration with a step size of 0.1. We also keep the initial weights and the seed locations from which flow map samples are generated for each of the models the same for fair comparison.

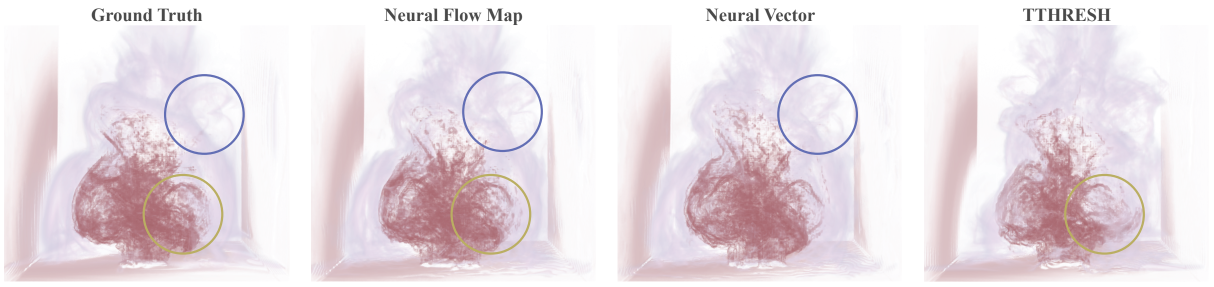

Figure 8: *We qualitatively compare the FTLE (computed by integrating for the entire duration of the simulation) for the* ScalarFlow *dataset between our method, neural vector fitting, and TTHRESH [BRLP19]. We find that our method does not inherit noisy artifacts away from the plume (blue circle), while in comparison to TTHRESH, we find that close to the plume center our method is able to better retain details of high repulsive behavior in the flow.*

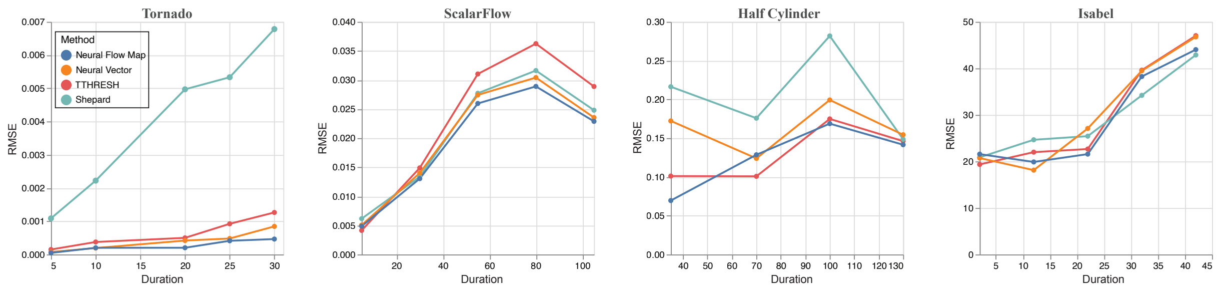

Figure 9: *We show quantitative results for the error incurred by flow maps obtained by different data reduction schemes for 3D flows: our method, neural fitting to vectors, TTHRESH [BRLP19], and Shepard interpolation. In general, our method obtains improved performance – note that at times, we obtain an improved performance in flow map, despite having a higher error in vector field (c.f. Fig. 10).*

Fig 5 shows the results of the comparison. Our hypothesis generally holds true, more specifically, models trained on smaller integration duration do not perform as well as models training on larger integration duration. Nevertheless, we see that at a certain point, we obtain diminishing returns, as the longest integration duration (20) is slightly worse than a duration of 10. We further test the generalization capabilities of the models on integration durations for which they were not trained on. In Fig. 5(right) we see that, models generalize well to integration durations for which they were trained. Importantly, though, for models trained on longer durations we do not sacrifice quality in preserving short-duration flow maps. Interestingly, we find that for integration duration of 20, we find that the model can generalize just as well, if not better, than smaller integration durations. This is despite the fact that its corresponding vector field quality is lower (left), suggesting that the flow map-based optimization can provide vector fields that take a small hit in performance with respect to the ground truth vector field, but nevertheless, faithfully capture the flow map.

We emphasize that these results do not answer the question of *what duration* to select, given a dataset. We believe that answering such a question is domain-dependent, e.g. for certain analyses, flow maps of longer duration are more relevant than those of shorter duration. Rather, our results show the robustness of our method to varying duration, capable of optimizing over a range of durations. We further hypothesize the drop in PSNR near the start and end timesteps in Fig. 4 and Fig. 5 is most likely due to how the training data is generated (please see Sec. 5.1) where we generated fixed duration flow map samples using forward integration only, thus biasing the network more towards the intermediate timesteps. We believe generating training data using a combination of forward and backward integration along with variable duration of integration can help alleviate this problem.

5. Results

We experimentally evaluate our method, both in 2D and 3D, by comparing across a range of baselines. Specifically, in 2D we compare to the superresolution method of Jakob et al. [JGG20], where we train a CNN-based superresolution model of flow map upsampling via their provided dataset of 2D flow simulations, under a 16x data reduction. Further, we compare to a standard baseline of bicubic upsampling, also 16x reduction. In 3D, we compare to the state-of-the-art compression method of TTHRESH [BRLP19], where the compression ratio is set to approximately the reduction rate that we use to train our model – this is an approximation, as TTHRESH is error-controlled, so we choose the error that leads to a compression ratio that is approximately our reduction rate. In addition, we compare to implicit neural representations [SMB*20, TSM*20, LJLB21], trained on the initiating seed positions that we take for our flow map samples, namely, the vectors at those positions. We also compare to two standard interpolation baselines: (1) Shepard interpolation, using the same aforementioned collection of points, and (2) cubic upsampling, where

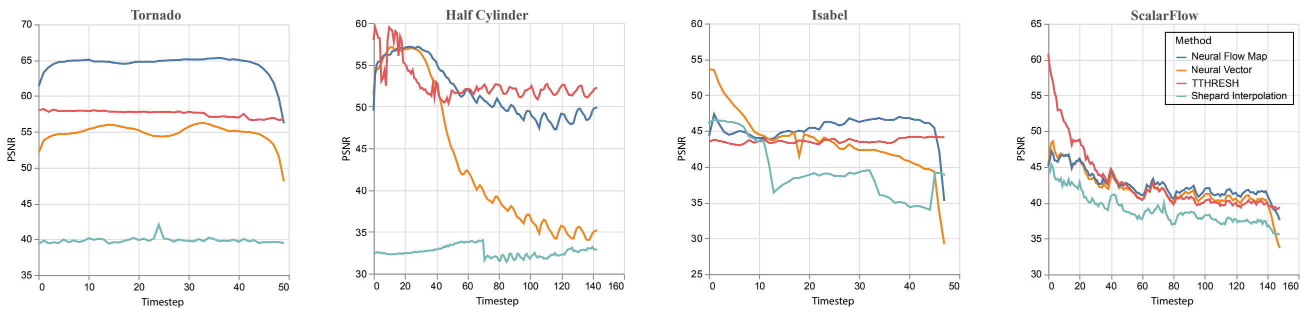

Figure 10: *We show quantitative results for 3D unsteady flows measured in terms of their vector field PSNR. We find, in general, that our method leads to an improved, if not competitive, performance across existing methods. We emphasize that our method does not explicitly optimize for a vector field, but nevertheless, the vector field that is found is a faithful approximation.*

we perform a 64x downsampling of the field. Though other interpolation schemes exist, e.g. ones based on barycentric coordinates, these methods can be quite expensive to compute due to the requirement of a triangulation / tetrahedralization, and thus we omit them from our study. Table 1 lists all of the datasets that we use in the paper, along with their spatial resolution, the reduction rate chosen for our experimental comparisons, as well as the integration duration. Since each dataset lives on a different physical domain, and thus time is not comparable, in the table we list duration in terms of the number of grid time steps taken.

5.1. Implementation Details

Network Architecture Settings Our network architecture is adapted from prior work by Lu et al. [LJLB21] – comprised of fully-connected layers and sinusoidal activation function. We depart from Lu et al. in that we utilize Rezero [BMM*20] wherein layers with skip connections are weighed by a learnable scalar value initialized to 0 at the start of training process. This modification to the architecture helped in stabilizing the training process. We use a total of 6 hidden layers in all of our experiments. The number of neurons in each of the layers is computed based on the size of the flow map samples used for training and the hyperparameter $\eta \in (0,1]$ which is set by the user. The hyperparameter η controls the number of network parameters with respect to the number of training samples. Setting η to 1 will result in a network with roughly equal number of parameters as the number of training samples. In practice, we use $\eta = .5$ in most of our experiments, unless otherwise specified.

Training Data Generation For a given duration δ we choose seed points in the spatial domain uniformly at random for all timesteps in the temporal domain $t \in [t_s, t_e - \delta]$, where t_s is the first timestep of the flow field data and t_e is the last timestep. Based on the total budget (i.e. the total size of flow field data divided by the reduction rate) - we allocate same budget of spatial seed points for each of the timesteps. We integrate these seeds points using RK4 integration scheme with a step size of 0.1.

Training Hyperparameters We use the ADAM optimizer [KB15] with a starting learning rate of 10^{-4} and decay it by a factor of 0.2 every 40 epochs. We train for a total of 100 epochs unless otherwise specified. We use explicit Euler integration, where the step size is set to $\frac{1}{10}$ of the grid-based time, in all of our experiments to train our model.

5.2. 2D Unsteady Flow

We first experimentally evaluate our method on 2D unsteady flows. In Fig. 6, we show a qualitative comparison between our method and baseline schemes, where we show (1) FTLE error maps as absolute difference between the ground truth FTLE and the predicted FTLE, and (2) the FTLE. Notably, we find that our method has significantly less visual artifacts relative to Jakob et al. [JGG20], despite their method being supervised on a large collection of 2D fluid flows. This suggests that obtaining a quality flow map reconstruction, given reduced data, can be addressed *without* the need of learning over a collection of fluid flows. We in fact obtain improved visual results for in-domain examples as well (fluid flow dataset [JGG20]), in addition to more standard 2D unsteady flow datasets,

In Fig. 7 we show the quantitative performance of our method over different datasets, where we evaluate the different methods under varying integration duration. Note that our method was trained on just *one* integration duration, for each of these datasets (c.f. Table 1); nevertheless, our method is not merely overfitting to the durations that it was trained on, and is able to generalize to arbitrary durations. As one would expect, we obtain similar flow map errors by a direct fit to the vectors (Neural Vector) for small durations, since optimizing just for vectors should lead to good local (in small duration) approximations in the flow map. However, for larger durations, we can see how our method, generally, improves over the neural vector fit baseline.

5.3. 3D Unsteady Flow

We next evaluate our method on a collection of 3D unsteady flows. First, we show in Fig. 8 a comparison of different methods for capturing details in the FTLE for the ScalarFlow dataset. Relative to TTHRESH [BRLP19], we find that our method is able to better capture intricate repulsive features near the central portion of the plume (yellow circle) for this smoke simulation. We find that a direct fit to the vectors yields comparable results, but as highlighted (blue circle) our method does not reproduce noisy features away from the plume center, as does the direct vector-based fit.

In Fig. 9 we show the quantitative performance of our method over different baselines. All in all, we find that our method yields improved performance over existing methods, though in certain instances we find TTHRESH leads to lower error. Note, however, that a significant advantage of our method over TTHRESH is that the computation of integral curves, and consequently the approximation of the flow map, does not require any such resampling to a regular grid. By representing the time-varying vector field as a coordinate-based neural network, we can compute the flow map *on demand*, in a random-access manner, whereas compression-based techniques, such as TTHRESH, require decompressing to the full, sampled regular grid in order to compute integral curves.

Although our method is not designed to optimize for a vector field, we find that, in general, it is capable of producing good vector field approximations. In Fig. 10 we measure the performance of our method, in terms of the vector field PSNR, relative to other techniques. Overall we find that our method is an improvement, if not competitive, with other techniques. Interestingly, we find that a good approximation to the flow map need not imply that the found vector field is faithful to the ground truth field. As an example, for the ScalarFlow dataset, TTHRESH obtains an improved vector field relative to our method early in the simulation; later our method sees an improvement, though the margin of improvement is modest. Nevertheless, as shown in Fig. 9, for ScalarFlow our method generally obtains large improvement in the flow map, particularly for longer integration duration, further verified qualitatively in Fig. 8.

6. Discussion

We have presented an approach for data reduction of unsteady flow, where we aim to learn Eulerian representations, e.g. time-varying vector fields, through explicitly optimizing for Lagrangian representations, e.g. samples of a flow map. Our experimental results demonstrate improvements in performance, both with respect to the underlying vector field, as well as the ground truth flow map. By learning a neural representation of a time-varying vector field, we further allow for the random-access computation of the flow map, obviating the need to explicitly sample the vector field to a regular grid, and thus providing a low-friction, convenient form of post-hoc analysis of unsteady flow. Although the main focus of our work has been for unsteady flows we believe that our method is applicable to steady flows as well. However, from the perspective of data reduction, we expect marginal gains with our method for steady flows. For large reduction rates in unsteady flows, our method encourages spatio-temporal consistency in the learned time-varying vector fields, a property that would be lacking in steady flows.

We acknowledge several limitations with our approach. Perhaps the main limitation is the time required for optimization (c.f. Table 2). Like neural ODEs [CRBD18], each step of optimization, in effect, necessitates the integration of a (learned) time-varying vector field. In practice, the adjoint sensitivity scheme for gradient computation ends up dominating the computation time; in the case of explicit Euler integration, this method requires performing backpropagation at each integration step. On the other hand, as studied in Chen et al. [CRBD18], an advantage of framing optimization-via-integration is that we can employ adaptive integration schemes. This has the potential to reduce the number of steps required for producing good gradient estimates, e.g. only requiring finely-resolved integration when gradients for particular integral curves are important.

We further acknowledge that our results are competitive with state-of-the-art data reduction methods, but in some instances our method is inferior. The restriction to optimizing *only* over fixed-duration flow map samples is largely for simplicity, and we believe that the incorporation of a richer set of information for optimization, e.g. a sparse sampling of vectors in addition to flow map samples, flow maps of varying integration duration, would lead to more effective reconstruction.

For future work, we plan on extending our method for flow map *extrapolation*, rather than just *interpolation*. We will investigate how to extend latent space integration [CRBD18, RCD19] rather than just integrating over the spatial domain, in order to enable our models to extrapolate flow. We expect that such a dynamics-based regularization on the latent space should prove useful for generalization, based on prior work in manifold mixup [VLB*19], and we anticipate these advantages will transfer to coordinate-based MLPs.

We also plan on investigating schemes to facilitate the time required for optimization. We are encouraged by recent works in meta-learning for coordinate-based MLPs [SCT*20, TMW*21], in particular, learned initializations for rapid training adaptation to novel signals [TMW*21]. Such schemes should transfer well to the rapid learning of unsteady flow. Moreover, thanks to recently-created datasets for building machine learning models on flow datasets [JGG20], we now have the opportunity to learn over a rich set of fluid flows. We plan on investigating coordinate-based neural networks that can scale better in space-time, rather than the use of a simple MLP. Since the size of the network grows quickly with the complexity of the input data, learning a good representation of a large dataset would require a larger network, thereby increasing the training as well as inference time. Methods such as ACORN [MLL*21] should prove useful in this regard, and we intend to adapt such architectures to time-varying flows.

Last, we plan on extending our method to optimize not just for flow maps, but more general properties of flows. In principle, it should be possible to optimize for quantities derived from flow maps, e.g. FTLE – indeed, similar types of memory-efficient schemes that we developed can be adapted to this setting. Moreover, other flow properties, be it steady (vorticity) or unsteady (acceleration) should also be possible to gather as part of data reduction *in situ*, and directly optimize post hoc. More broadly, we believe that coordinate-based neural networks have significant utility for data reduction in scientific visualization, and we are excited to pursue such directions as part of future work.

7. Acknowledgements

This work is supported by the National Science Foundation under grant number IIS-2007444.

References

[ABK16] Austin W., Ballard G., Kolda T. G.: Parallel tensor compression for large-scale scientific data. In *2016 IEEE international parallel and distributed processing symposium (IPDPS)* (2016), IEEE, pp. 912–922. 2

[ACG*14] Agranovsky A., Camp D., Garth C., Bethel E. W., Joy K. I., Childs H.: Improved post hoc flow analysis via lagrangian representations. In *2014 IEEE 4th Symposium on Large Data Analysis and Visualization (LDAV)* (2014), IEEE, pp. 67–75. 2, 3

[AGJ11] Agranovsky A., Garth C., Joy K.: Extracting flow structures using sparse particles. 3

[BMM*20] Bachlechner T., Majumder B. P., Mao H. H., Cottrell G. W., McAuley J.: Rezero is all you need: Fast convergence at large depth. *arXiv preprint arXiv:2003.04887* (2020). 9

[BR10] Brunton S. L., Rowley C. W.: Fast computation of finite-time lyapunov exponent fields for unsteady flows. *Chaos: An Interdisciplinary Journal of Nonlinear Science 20*, 1 (2010), 017503. 3

[BRLP19] Ballester-Ripoll R., Lindstrom P., Pajarola R.: Tthresh: Tensor compression for multidimensional visual data. *IEEE transactions on visualization and computer graphics 26*, 9 (2019), 2891–2903. 2, 8, 9

[CBJ16] Chandler J., Bujack R., Joy K. I.: Analysis of error in interpolation-based pathline tracing. In *EuroVis (Short Papers)* (2016), pp. 1–5. 3

[COJ14] Chandler J., Obermaier H., Joy K. I.: Interpolation-based pathline tracing in particle-based flow visualization. *IEEE transactions on visualization and computer graphics 21*, 1 (2014), 68–80. 3

[CRBD18] Chen R. T. Q., Rubanova Y., Bettencourt J., Duvenaud D. K.: Neural ordinary differential equations. In *Advances in Neural Information Processing Systems* (2018), vol. 31, Curran Associates, Inc. 3, 5, 10

[DC16] Di S., Cappello F.: Fast error-bounded lossy hpc data compression with sz. In *2016 ieee international parallel and distributed processing symposium (ipdps)* (2016), IEEE, pp. 730–739. 2

[DLHT15] Dong C., Loy C. C., He K., Tang X.: Image super-resolution using deep convolutional networks. *IEEE transactions on pattern analysis and machine intelligence 38*, 2 (2015), 295–307. 2

[DLT16] Dong C., Loy C. C., Tang X.: Accelerating the super-resolution convolutional neural network. In *European conference on computer vision* (2016), Springer, pp. 391–407. 2

[EMY*20] Erichson N. B., Mathelin L., Yao Z., Brunton S. L., Mahoney M. W., Kutz J. N.: Shallow neural networks for fluid flow reconstruction with limited sensors. *Proceedings of the Royal Society A 476*, 2238 (2020), 20200097. 3

[EUT19] Eckert M.-L., Um K., Thuerey N.: Scalarflow: a large-scale volumetric data set of real-world scalar transport flows for computer animation and machine learning. *ACM Transactions on Graphics (TOG) 38*, 6 (2019), 1–16. 6

[GGT17] Günther T., Gross M., Theisel H.: Generic objective vortices for flow visualization. *ACM Transactions on Graphics (TOG) 36*, 4 (2017), 1–11. 6

[GGTH07] Garth C., Gerhardt F., Tricoche X., Hans H.: Efficient computation and visualization of coherent structures in fluid flow applications. *IEEE Transactions on Visualization and Computer Graphics 13*, 6 (2007), 1464–1471. 3

[GHCW21] Gu P., Han J., Chen D. Z., Wang C.: Reconstructing unsteady flow data from representative streamlines via diffusion and deep-learning-based denoising. *IEEE Computer Graphics and Applications 41*, 6 (2021), 111–121. 3

[GWGS02] Guthe S., Wand M., Gonser J., Strasser W.: Interactive rendering of large volume data sets. In *IEEE Visualization, 2002. VIS 2002.* (2002), IEEE, pp. 53–60. 2

[GYH*20] Guo L., Ye S., Han J., Zheng H., Gao H., Chen D. Z., Wang J.-X., Wang C.: Ssr-vfd: Spatial super-resolution for vector field data analysis and visualization. In *2020 IEEE Pacific Visualization Symposium (PacificVis)* (2020), IEEE Computer Society, pp. 71–80. 2

[Hal00] Haller G.: Finding finite-time invariant manifolds in two-dimensional velocity fields. *Chaos: An Interdisciplinary Journal of Nonlinear Science 10*, 1 (2000), 99–108. 1, 3

[Hal01] Haller G.: Distinguished material surfaces and coherent structures in three-dimensional fluid flows. *Physica D: Nonlinear Phenomena 149*, 4 (2001), 248–277. 1

[HBJG16] Hummel M., Bujack R., Joy K. I., Garth C.: Error estimates for lagrangian flow field representations. In *EuroVis (Short Papers)* (2016), pp. 7–11. 3

[HSJ21] Han M., Sane S., Johnson C. R.: Exploratory lagrangian-based particle tracing using deep learning. *arXiv preprint arXiv:2110.08338* (2021). 3

[HSW10] Hlawatsch M., Sadlo F., Weiskopf D.: Hierarchical line integration. *IEEE transactions on visualization and computer graphics 17*, 8 (2010), 1148–1163. 3

[HTZ*19] Han J., Tao J., Zheng H., Guo H., Chen D. Z., Wang C.: Flow field reduction via reconstructing vector data from 3-d streamlines using deep learning. *IEEE computer graphics and applications 39*, 4 (2019), 54–67. 3

[HW19] Han J., Wang C.: Tsr-tvd: Temporal super-resolution for time-varying data analysis and visualization. *IEEE transactions on visualization and computer graphics 26*, 1 (2019), 205–215. 2

[HY00] Haller G., Yuan G.: Lagrangian coherent structures and mixing in two-dimensional turbulence. *Physica D: Nonlinear Phenomena 147*, 3-4 (2000), 352–370. 1, 3, 4

[HZCW21] Han J., Zheng H., Chen D. Z., Wang C.: Stnet: An end-to-end generative framework for synthesizing spatiotemporal super-resolution volumes. *IEEE Transactions on Visualization and Computer Graphics* (2021). 2

[JEG11] Jang Y., Ebert D. S., Gaither K.: Time-varying data visualization using functional representations. *IEEE Transactions on Visualization and Computer Graphics 18*, 3 (2011), 421–433. 2

[JGG20] Jakob J., Gross M., Günther T.: A fluid flow data set for machine learning and its application to neural flow map interpolation. *IEEE Transactions on Visualization and Computer Graphics 27*, 2 (2020), 1279–1289. 2, 6, 8, 9, 10

[KB15] Kingma D. P., Ba J.: Adam: A method for stochastic optimization. In *ICLR (Poster)* (2015). 9

[LGZL*20] Liu L., Gu J., Zaw Lin K., Chua T.-S., Theobalt C.: Neural sparse voxel fields. *Advances in Neural Information Processing Systems 33* (2020). 3

[LJLB21] Lu Y., Jiang K., Levine J. A., Berger M.: Compressive neural representations of volumetric scalar fields. *Comput. Graph. Forum 40*, 3 (2021), 135–146. 2, 4, 6, 8, 9

[LSE*11] Lakshminarasimhan S., Shah N., Ethier S., Klasky S., Latham R., Ross R., Samatova N. F.: Compressing the incompressible with isabela: In-situ reduction of spatio-temporal data. In *European Conference on Parallel Processing* (2011), Springer, pp. 366–379. 2

[Ma03] Ma K.-L.: Visualizing time-varying volume data. *Computing in Science & Engineering 5*, 2 (2003), 34–42. 2

[MLL*21] Martel J. N. P., Lindell D. B., Lin C. Z., Chan E. R., Monteiro M., Wetzstein G.: Acorn: Adaptive coordinate networks for neural scene representation. *ACM Trans. Graph. (SIGGRAPH) 40*, 4 (2021). 10

[MST*20] Mildenhall B., Srinivasan P. P., Tancik M., Barron J. T., Ramamoorthi R., Ng R.: Nerf: Representing scenes as neural radiance fields for view synthesis. In *European conference on computer vision* (2020), Springer, pp. 405–421. 3

[Mur93] MURAKI S.: Volume data and wavelet transforms. *IEEE Computer Graphics and applications 13*, 4 (1993), 50–56. 2

[Pon87] PONTRYAGIN L. S.: *Mathematical theory of optimal processes*. CRC press, 1987. 3

[RCD19] RUBANOVA Y., CHEN R. T., DUVENAUD D. K.: Latent ordinary differential equations for irregularly-sampled time series. *Advances in Neural Information Processing Systems 32* (2019), 5320–5330. 3, 10

[RG19] ROJO I. B., GÜNTHER T.: Vector field topology of time-dependent flows in a steady reference frame. *IEEE transactions on visualization and computer graphics 26*, 1 (2019), 280–290. 6

[RGG19] ROJO I. B., GROSS M., GÜNTHER T.: Accelerated monte carlo rendering of finite-time lyapunov exponents. *IEEE transactions on visualization and computer graphics 26*, 1 (2019), 708–718. 4

[RPD19] RAPP T., PETERS C., DACHSBACHER C.: Void-and-cluster sampling of large scattered data and trajectories. *IEEE transactions on visualization and computer graphics 26*, 1 (2019), 780–789. 3

[SB21] SAHOO S., BERGER M.: Integration-aware vector field super resolution. 2

[SCB] SANE S., CHILDS H., BUJACK R.: An interpolation scheme for vdvp lagrangian basis flows. 3

[SCT*20] SITZMANN V., CHAN E. R., TUCKER R., SNAVELY N., WETZSTEIN G.: Metasdf: Meta-learning signed distance functions. In *Proc. NeurIPS* (2020). 10

[SFT*22] SCHWERI L., FOUCHER S., TANG J., AZEVEDO V. C., GÜNTHER T., SOLENTHALER B.: A physics-aware neural network approach for flow data reconstruction from satellite observations. *New techniques for improving climate models, predictions and projections* (2022). 3

[SGM*11] SUTER S. K., GUITIAN J. A. I., MARTON F., AGUS M., ELSENER A., ZOLLIKOFER C. P., GOPI M., GOBBETTI E., PAJAROLA R.: Interactive multiscale tensor reconstruction for multiresolution volume visualization. *IEEE Transactions on Visualization and Computer Graphics 17*, 12 (2011), 2135–2143. 2

[SLM05a] SHADDEN S. C., LEKIEN F., MARSDEN J. E.: Definition and properties of lagrangian coherent structures from finite-time lyapunov exponents in two-dimensional aperiodic flows. *Physica D: Nonlinear Phenomena 212*, 3-4 (2005), 271–304. 3

[SLM05b] SHADDEN S. C., LEKIEN F., MARSDEN J. E.: Definition and properties of Lagrangian coherent structures from finite-time Lyapunov exponents in two-dimensional aperiodic flows. *Physica D: Nonlinear Phenomena 212*, 3-4 (2005), 271–304. doi:10.1016/j.physd.2005.10.007. 6

[SMB*20] SITZMANN V., MARTEL J., BERGMAN A., LINDELL D., WETZSTEIN G.: Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems 33* (2020). 2, 4, 6, 8

[SP07] SADLO F., PEIKERT R.: Efficient visualization of lagrangian coherent structures by filtered amr ridge extraction. *IEEE Transactions on Visualization and Computer Graphics 13*, 6 (2007), 1456–1463. 3

[SRP11] SADLO F., RIGAZZI A., PEIKERT R.: Time-dependent visualization of lagrangian coherent structures by grid advection. In *Topological Methods in Data Analysis and Visualization*. Springer, 2011, pp. 151–165. 3

[SW03] SCHNEIDER J., WESTERMANN R.: Compression domain volume rendering. In *IEEE Visualization, 2003. VIS 2003.* (2003), IEEE, pp. 293–300. 2

[TMW*21] TANCIK M., MILDENHALL B., WANG T., SCHMIDT D., SRINIVASAN P. P., BARRON J. T., NG R.: Learned initializations for optimizing coordinate-based neural representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2021), pp. 2846–2855. 10

[TSM*20] TANCIK M., SRINIVASAN P. P., MILDENHALL B., FRIDOVICH-KEIL S., RAGHAVAN N., SINGHAL U., RAMAMOORTHI R., BARRON J. T., NG R.: Fourier features let networks learn high frequency functions in low dimensional domains. *NeurIPS* (2020). 2, 4, 6, 8

[VLB*19] VERMA V., LAMB A., BECKHAM C., NAJAFI A., MITLIAGKAS I., LOPEZ-PAZ D., BENGIO Y.: Manifold mixup: Better representations by interpolating hidden states. In *International Conference on Machine Learning* (2019), PMLR, pp. 6438–6447. 10

[WW21] WEISS S., WESTERMANN R.: Differentiable direct volume rendering. *IEEE Transactions on Visualization Computer Graphics*, 01 (2021), 1–1. 3

[XHKK21] XIAN W., HUANG J.-B., KOPF J., KIM C.: Space-time neural irradiance fields for free-viewpoint video. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2021), pp. 9421–9431. 4

[YWS*21] YUQI W., WU Y., SHAN L., JIAN Z., HUIYING R., TIECHUI Y., MENGHAI K.: Flow field reconstruction method based on array neural network. *The Aeronautical Journal 125*, 1283 (2021), 223–243. 3