Abstract—We present a method for learning neural representations of flow maps from time-varying vector field data. The flow map is pervasive within the area of flow visualization, as it is foundational to numerous visualization techniques, e.g., integral curve computation for pathlines or streaklines, as well as computing separation/attraction structures within the flow field. Yet bottlenecks in flow map computation, namely the numerical integration of vector fields, can easily inhibit their use within interactive visualization settings. In response, in our work we seek neural representations of flow maps that are efficient to evaluate, while remaining scalable to optimize, both in computation cost and data requirements. A key aspect of our approach is that we can frame the process of representation learning not in optimizing for samples of the flow map, but rather, a self-consistency criterion on flow map derivatives that eliminates the need for flow map samples, and thus numerical integration, altogether. Central to realizing this is a novel neural network design for flow maps, coupled with an optimization scheme, wherein our representation only requires the time-varying vector field for learning, encoded as instantaneous velocity. We show the benefits of our method over prior works in terms of accuracy and efficiency across a range of 2D and 3D time-varying vector fields, while showing how our neural representation of flow maps can benefit unsteady flow visualization techniques such as streaklines, and the finite-time Lyapunov exponent.

Index Terms—Flow Visualization, Visualization techniques and methodologies, Machine learning

1 Introduction

Visual analysis is central to gaining insight on the underlying behaviour of unsteady flow data. Numerous visualization techniques have been developed to extract meaningful information from flow data, all in support of analyzing a variety of flow features. Among these techniques, notable ones include the finite-time Lyapunov exponents (FTLE) [1], used to understand the rate of separation between nearby particles integrated over a finite time interval, and its resulting Lagrangian coherent structures (LCS) [1, 2], extracted as the ridges of the FTLE field. Streaklines are another visualization technique widely used by researchers, used to factor out background motion in flows, and identify underlying vortices that might be present. Other flow visualization techniques include line integral convolution (LIC) [3], almost invariant sets (AIS) [4], finite-size Lyapunov exponent (FSLE) [5, 6] and the coherent ergodic partitions [7].

A core component common to all the above techniques is the computation of the flow map. The flow map provides the position of a particle advected under a flow over a finite time span, and typically, this is computed by integrating a time-varying vector field. For large time spans, this integration process can become computationally expensive, and thus impede interactivity within visual analysis. For example, trajectories of a dense set of particles sufficiently covering the spatial domain must be computed in order to generate the FTLE field. If the user is interested in interactively exploring FTLE under varying time spans, the expense in computing the flow map can hinder this exploration. Techniques that can improve the flow map computation time are thus attractive for a wide-variety of downstream visualization tasks.

In the literature, numerous techniques have been proposed for fast FTLE computation [8], [9], [10], [11], [12], [13]. Most of these techniques fall into the general category of reducing the number of flow map evaluations required to accurately estimate the FTLE field. In a similar way, many other techniques have been proposed for fast LIC computation [14], [15], and fast streamline computation [16]. However, most of these techniques are targeted towards improving the computation time of a specific downstream visualization task. For the majority of flow visualization techniques, the flow map computation time acts as a bottleneck, and few techniques have focused on the core of the problem i.e. improving the flow map computation time.

Motivated by these problems, in this work we propose a novel technique for fast and accurate flow map computation. We propose a novel coordinate-based neural network that serves as a surrogate for a flow map. Specifically, given a particle identified by a spatiotemporal coordinate, and time span, the network predicts the spatial position corresponding to the particle’s integration under the flow field. Such neural representations of flow maps have been recently studied, both for 2D unsteady flows [17], and more broadly for learning latent space representations [18]. However, the learning of flow maps presents a number of challenges for existing methods. First, there is a steep training data requirement, where it is necessary to generate a large number of flow map samples on which to learn. Second, the input dimensionality varies over space, time, and time span, and thus in order to match the high dimensionality of the input space, the complexity (e.g. number of parameters) of the network often needs to be quite large. The complexity of the network can prove prohibitively expensive for training, provided the large dataset size, as well as prevent interactivity for use in downstream visual analysis.

Our approach aims to address, at once, these challenges through a novel network design that enables efficient inference, coupled with a novel optimization scheme for scalable training. A key aspect of our approach is that, through careful network design and optimization, we eliminate the
need to learn from ground-truth flow map samples altogether. Rather, we take advantage of a basic property of flow maps: the instantaneous velocity of the flow map should be equivalent to the vector field. By optimizing the flow map derivative to represent the vector field, this “primes” the flow map itself to give a good approximation of particle transport, under small time spans. We show how to leverage this in devising a self-consistency criterion for learning the flow map under a range of time spans. Moreover, building on recent hybrid grid-MLP models \cite{19}, our network is efficient to evaluate, which enables both scalable training, as well as efficient inference. Our implementation can be found here: https://github.com/SarojKumarSahoo/NIFM

Our main contributions can be summarized as follows:

1) We propose a novel network architecture for learning a neural representation of flow maps, that is fast, accurate and scalable.
2) We propose a novel way to optimize the network only using the vector field, without requiring access to flow map samples during optimization.
3) We show the advantage of using our technique by comparing against existing techniques both qualitatively and quantitatively. We demonstrate that, with modest training time, our method provides for a more accurate flow map approximation, and is more efficient at inference time, and hence applicable to numerous unsteady flow visualization techniques.

2 RELATED WORK

We introduce related work along four main directions: Lagrangian flow-based representations, methods for efficiently computing flow maps for FTLE, deep learning as applied to flow visualization, and more broadly relevant research in machine learning.

Lagrangian Particle Interpolation. The Lagrangian representation of unsteady flow fields stores data in the form of trajectories of massless particles. Agranovsky et al. \cite{20} showed that, for exploratory analysis, in-situ trajectory computation and post-hoc interpolation is more storage-efficient than compared to traditional Eulerian representations. Several works to improve the accuracy of post-hoc Lagrangian particle interpolation have been proposed since \cite{21}, \cite{22}, \cite{23}, \cite{24}, \cite{25}. Bujack and Joy \cite{23} proposed a method for representing trajectories as parametric curves for a more accurate post-hoc interpolation, and additionally, they performed an error estimation of the proposed Lagrangian representation. In a similar way, several works focused on theoretical/empirical error analysis of Lagrangian interpolation \cite{26}, \cite{27}, \cite{28}. Even though these techniques do not require expensive numerical integration during post-hoc analysis, they are still expensive because of the number of steps required to compute the full trajectory. In the recent work by Li et al. \cite{29} they represented the trajectories as B-spline curves and improve the computation time of new trajectories by interpolation between the B-spline control points. Lagrangian representations of flow are attractive as a kind of data reduction, and assuming a sufficiently-dense sampling of trajectories, can often be quite accurate. Nevertheless, this representation can come at a steep computational cost for interactive analysis, as a

common bottleneck is repeatedly performing spatial queries over irregularly-sampled particles in space-time.

Fast FTLE computation. Computation of FTLE and its applications have received significant attention in the literature. Haller et al. \cite{1}, \cite{2} showed that Lagrangian coherent structures can be extracted as the ridges of FTLE. Following this pioneering work many researchers focused on improving the computation time of FTLE. Garth et al. \cite{8} proposed an incremental flow map approximation technique for improving the computation time of FTLE. Sadlo et al. \cite{30} introduced a technique for FTLE ridge extraction using adaptive mesh refinement. Their proposed approach provides a speed-up in FTLE computation by avoiding integration of seed particles where no ridges are present. Kasten et al. \cite{9} constructed a localized FTLE and additionally, a faster way to compute it by reusing the separation values from previous time steps. Brunton et al. \cite{11} proposed a fast FTLE computation technique taking advantage of flow map composition for longer flow map approximations. Hlawatsch et al. \cite{31} introduced a hierarchical line integration scheme taking advantage of spatial and temporal coherence to improve the computation time of a dense set of particles. This work focuses on projection of particles based on the short pre-computed integral curves and thus has an accuracy trade-off. Sadlo et al. \cite{13} proposed a grid advection technique for efficient FTLE computation taking advantage of temporal coherence. All these techniques are specifically targeted towards improving the computation time of a specific downstream task i.e. either FTLE or extraction of LCS from FTLE. In this work, we focus on improving the computation time of flow map itself allowing for a improved computation time for a wide range of downstream tasks.

Deep learning for Flow Visualization. In the recent years, numerous deep learning based techniques have been proposed in relation to flow visualization. Han et al. \cite{32} introduced FlowNet an encoder-decoder deep learning framework for clustering, filtering and selection of streamlines and stream surfaces. Gu et al. \cite{33} proposed a two-stage deep learning framework for flow field reconstruction using selected streamlines. Guo et. al. \cite{34} and Sahoo et al. \cite{35} proposed a deep learning based vector field super resolution using novel loss functions. Most relevant to our method is the flow map super resolution technique proposed by Jakob et al. \cite{36}, wherein the proposed technique requires a low resolution flow map as input and outputs higher resolution (4x) flow map. Even though the inference of the high resolution flow map is fast, the technique still the requires computation of a low resolution flow map making it less efficient in terms of overall computation time. Moreover, the flow maps are limited only to the grid locations. Our technique on the other hand is able to generate arbitrary flow map samples at any given space-time location. Another relevant work to our method is the recent work by Han et al. \cite{17} where the authors proposed a deep learning technique which takes in a space-time coordinate and time span as input, and outputs the evaluation of the flow map. Our approach is similar to the technique proposed by Han et al. in scope, e.g.
learning the full flow map of a corresponding vector field, however we differ in network architecture, optimization scheme, and data requirements. Specifically, our technique does not require flow map samples for supervision, and instead can learn a flow map representation only using a provided vector field.

**Neural Differential Equations** Neural ordinary differential equation (Neural ODE), a technique to solve initial-value ODE problems proposed by Chen et al. [37] has been extended [38], [39] and applied to various different research domains [40], [41]. Theoretically, since, flow maps are solution to an initial-value ODE, neural ODE should naturally extend to solve the problem. However, learning a flow map representation using neural ODE has not been studied yet. Our work is closely related to the work by Biloš et al. [18], wherein they approximate the solution directly in a single step instead of integrating within a latent representation space. We draw inspiration from their work in the way we model the flow map prediction, however, the main distinction between our approach is in the network architecture design and the proposed optimization scheme.

Our method of optimizing for flow map derivatives further draws inspiration from gradient-based learning methods, e.g. modeling shapes with gradient fields [42], and accelerating the volume-rendering integral through learning antiderivative networks (AutoInt) [43]. A key difference between our method and AutoInt is that our novel network design allows us to forgo the requirement of computing integrals that are ultimately used for supervision in optimizing a gradient network, making our method of optimization more computationally efficient.

### 3 Methods

In this section we present our approach, where we first describe the objective we seek to optimize, followed by a network design suited for this objective, and last we describe our specific approach to optimization.

#### 3.1 Integration-free learning

The flow map is an important mathematical tool that is utilized by numerous visualization techniques. To mathematically represent the flow map, let us consider a time-dependent flow field $\nu : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n$, where $\nu(x(t), t)$ describes the vector of a particle at time $t$ with spatial position $x(t) \in \mathbb{R}^n$. The trajectory of a massless particle, advected under the influence of the flow field $\nu$, is governed by the following ordinary differential equation:

$$\frac{dx(t)}{dt} = \nu(x(t), t), \quad x(t_0) = x_0,$$  \hspace{1cm} (1)

where $x_0$ represents the initial position of the particle at starting time $t_0$. Integrating this differential equation under a specified time span $\tau$ gives us the flow map $\Phi$, which varies in initial position $x_0$, starting time $t_0$, and time span $\tau$:

$$\Phi(x_0, t_0, \tau) : \mathbb{R}^n \times \mathbb{R} \times \mathbb{R} \to \mathbb{R}^n = x_0 + \int_{t_0}^{t_0+\tau} \nu(x(t), t)dt.$$  \hspace{1cm} (2)

In practice, the computation of the flow map depends on (1) a means of interpolating the vector field at arbitrary space-time coordinates within the domain, and (2) a choice of numerical integration scheme, e.g. Euler or Runge-Kutta. Integrating for long time spans, however, can become a computational bottleneck when coupling the flow map with a particular visualization technique. This motivates the need for alternative flow map representations that can mitigate the expense of numerical integration.

In this work, we seek neural representations of flow maps, which we will denote $\hat{\Phi}$, that are (1) scalable to optimize, (2) efficient to evaluate, and (3) serve as accurate approximations. Satisfying all criteria, at once, is challenging with prior methods, as the choice of network design, objective(s) to be optimized, and data required for optimization, are all important considerations that interrelate. A standard approach [44] is to collect a dense set of samples of the flow map, and optimize a neural network to reproduce these samples, either directly as its output [17], or indirectly through integrating a learned vector field [44]. However, to ensure good generalization, the number of flow map samples to collect needs to be quite large – at least on the order of the vector field resolution – with each sample requiring expensive numerical integration. Further, coordinate-based networks need to be sufficiently large for accurate learning, and thus combined, the dataset size and network complexity can lead to expensive training, and inefficient evaluation.

Our work foregoes the need for numerically integrating the vector field altogether. Instead, we optimize for flow map derivatives, rather than the raw flow map output, taking advantage of the following basic property of a flow map:

$$\frac{\partial \Phi(x, t, \tau)}{\partial \tau} = \nu(\Phi(x, t, \tau), t + \tau).$$  \hspace{1cm} (3)

In other words, the derivative of the flow map taken with respect to time span $\tau$, at position $x$ and time $t$, can be found by (1) evaluating the flow map at the given inputs, and (2) accessing the vector field at the flow map’s positional output, at time $t + \tau$. A flow map representation whose derivative is satisfied at all positions and times will, by construction, produce valid integral curves. Specifically, upon fixing position and time, evaluating the representation in increasing time span $\tau$ will yield a curve whose tangent vectors match the vector field as defined in Eq. (3).

Of course, this approach assumes full access to the flow map itself, which is ultimately what we are trying to find. To help formulate a well-defined optimization problem, we identify two basic properties of a flow map that we expect any approximation should satisfy. Herein we refer to the neural flow map representation as $\hat{\Phi}$.

**P1. Identity mapping.** When we integrate a particle for a time span of $\tau = 0$, then the flow map $\hat{\Phi}$ should return the starting position, irrespective of the starting time:

$$\hat{\Phi}(x, t, 0) = x.$$  \hspace{1cm} (4)

We argue that any approximation $\hat{\Phi}$ should exactly satisfy identity preservation. Otherwise, a small perturbation $\delta \in \mathbb{R}^n$ yielding $\hat{\Phi}(x, t, 0) = x + \delta$ would lead to an accumulation in error for repeated evaluation of the flow map approximation $\hat{\Phi}$.

**P2. Instantaneous velocity.** For a time span of $\tau = 0$, if we compute the derivative of the flow map $\hat{\Phi}$ with respect
to time span, then it should return the evaluation of the field \( \nu \) at the provided position \( x \) and time \( t \):

\[
\frac{\partial \hat{\Phi}(x, t, 0)}{\partial \tau} = \nu(x, t).
\]

A neural flow map representation \( \hat{\Phi} \) whose derivative poorly approximates the vector field, e.g., points in a different direction, can lead to particle trajectories that diverge from actual trajectories. Indeed, upon a simple first-order approximation, we have:

\[
\hat{\Phi}(x, t, \varepsilon) \approx \hat{\Phi}(x, t, 0) + \varepsilon \nu(x, t).
\]

and thus, if the derivative of \( \hat{\Phi} \) at \( \tau = 0 \) is poorly approximated, then this negatively impacts the action of the flow map itself. Also note that failing to preserve the identity mapping (P1) can further compound error.

Assuming the above properties hold, we propose the following criterion of self-consistency for learning flow maps:

\[
l_s(x, t, \tau) = \left\| \frac{\partial \hat{\Phi}(x, t, \tau)}{\partial \tau} - \frac{\partial \hat{\Phi}(\hat{\Phi}(x, t, \tau), t + \tau, 0)}{\partial \tau} \right\|.
\]

Here we have replaced the vector field in Eq. (3) with the flow map derivative. Hence, assuming property (P2) holds, the derivative of the flow map at \( \tau = 0 \) will faithfully represent the vector field. By minimizing this objective over the full domain via:

\[
L_s = \mathbb{E}_{(x, t) \in D, \tau \in \mathcal{T}} \left[ l_s(x, t, \tau) \right],
\]

where \( D \) is the spatiotemporal domain, and \( \mathcal{T} = [\tau_{\text{min}}, \tau_{\text{max}}] \) is an interval of time spans we aim to support in our approximation, we can ensure global self-consistency. Such a property is fundamental to any flow map model, but it is possible for \( \hat{\Phi} \) to minimize Eq. (8), while remaining a poor approximation of \( \Phi \). However, if instantaneous velocity is well-satisfied (P2), and remains fixed, if not minimally changed, during optimization, then this will limit the space of flow maps that satisfy Eq. (8).

To provide intuition for our approach, if a flow map initially satisfies properties (P1) and (P2) then this can give a simple linear approximation, as shown in Fig. 1 (left). However, the self-consistency criterion will naturally report a high loss for a sufficiently-large \( \tau > 0 \). By optimizing over a range of time spans \( \mathcal{T} \), we can incrementally improve on self-consistency: first for small time spans, given (P2) holds, and then for larger time spans, as notionally depicted in Fig. 1 (right). This idea of incrementally building the flow map has precedence in the literature [51], but in our approach we eliminate the need for numerical integration, and instead only require access to the original vector field. But critical to our approach, we require that the flow map approximation satisfy properties (P1) and (P2). We next turn to a novel network design suited for these ends.

### 3.2 A network design for flow maps

Coordinate-based neural networks, in particular ones based on sinusoidal positional encodings [45], [46], [47], are a natural choice for our flow map network design. Specifically, position \( n \) dimensions, time \( t \) dimension, and time span \( T \) dimension can collectively be treated as individual coordinates as input to a multi-layer perceptron (MLP) [45], [46], whose output corresponds to the flow map prediction, please see Fig. 2a. However, such an approach fails to guarantee property (P1) by design, and instead, the identity mapping must be learned. Moreover, the input-based derivatives of MLPs are themselves nontrivial neural representations, and instead, the approach we eliminate the need for numerical integration, and use a stable composition-based objective (c.f. Eq. (7)).

Rather than use a standard coordinate-based network we propose a 2-tiered network design, please see Fig. 2b for an overview. The first network, which we denote \( f_\nu : \mathbb{R}^n \times \mathbb{R} \rightarrow \mathbb{R}^n \) learns a \( d \)-dimensional spatiotemporal representation of the domain that is tasked with property (P2), learning a representation of instantaneous velocity. We condition the representation \( f_\nu \) with a given time span \( \tau \), via the following multiplicative scaling:

\[
z^{(0)} = \sigma_\nu(\tau m^{(0)}) \odot f_\nu(x, t),
\]

where \( m^{(0)} \in \mathbb{R}^d \) is a learnable vector aimed to reconcile the scaling of \( \tau \) – initially expressed in terms of the physical domain – for the neural representation. The function \( \sigma_\nu \) is a nonlinearity that serves to squash values into a predetermined range, and in practice this is set as a hyperbolic tangent, while \( \odot \) indicates element-wise multiplication. The second network, which we denote \( f_\tau : \mathbb{R}^n \times \mathbb{R} \rightarrow \mathbb{R}^d \), learns a \( d \)-dimensional spatiotemporal representation but one that is specific to the flow map for nonzero time spans. We combine the two representations, \( f_\nu \) and \( f_\tau \), through a residual connection:

\[
z^{(1)} = z^{(0)} + \sigma_\nu \left( \tau m^{(1)} \right) \odot \sigma_\tau \left( z^{(0)} \odot \left( W^{(1)} f_\tau(x, t) \right) \right),
\]

where \( m^{(1)} \in \mathbb{R}^d \) serves the same purpose as \( m^{(0)} \), and \( W^{(1)} \in \mathbb{R}^{d \times d} \) is a learnable linear transformation. A consequence of the above construction is that the derivative w.r.t \( \tau \) when \( \tau = 0 \) evaluates to

\[
\frac{dz^{(1)}}{d\tau} = \frac{dz^{(0)}}{d\tau} = \left( \sigma'_\nu m^{(0)} \right) \odot f_\nu(x, t).
\]

Subsequent representations are formed via residual connections, in order to preserve the above derivative:

\[
z^{(l)} = z^{(l-1)} + \sigma_\nu \left( \tau m^{(l)} \right) \odot \sigma_\tau \left( W^{(l)} z^{(l-1)} \right),
\]
and, finally the last layer \( L \) applies a single linear transformation to give us the output position, wherein we also include a skip connection for the input position:

\[
\hat{\Phi}(x, t, \tau) = x + W^{(L)} z^{(L-1)}.
\]  

(13)

Returning to our properties, we note that this network design, by construction, satisfies the identity mapping \((P1)\), so long as the chosen activation function \( \sigma_\nu \) satisfies \( \sigma_\nu(0) = 0 \). The multiplicative scaling performed at each layer ensures that all representations will be zero vectors throughout the network. Critically, we do not introduce bias vectors, in order to guarantee this identity mapping. More importantly, the designed residual connections lead to a particularly simple network for the flow map derivative at \( \tau = 0 \) \((P2)\):

\[
\frac{d\hat{\Phi}(x, t, 0)}{d\tau} = W^{(L)} (m^{(0)} \circ f_\nu(x, t)).
\]  

(14)

Please see the appendix for the supporting derivation. There are two implications of this result. First, instantaneous velocity of the flow map does not depend on the representation \( f_\tau \), as depicted in Fig. 2(c); in fact it is entirely decoupled from the rest of the network (c.f. Eqs. 19 and 20). Hence, we can directly optimize instantaneous velocity of the flow map for the vector field \( \nu \) using Eq. (14), without making reference to the remainder of the model. In turn, a flow map that satisfies instantaneous velocity helps “prime” the model in satisfying the self-consistency criterion, and ensures stability, e.g. we can choose to freeze the parameters associated with \( f_\nu \) and \( W^{(L)} \) when optimizing Eq. (8), and the network’s representation of instantaneous velocity will remain unchanged. Secondly, the simplicity of this derivative network ensures that we can easily optimize for the vector field. In contrast, for a standard MLP (c.f. Fig. 2) its instantaneous velocity would amount to an involved derivative network \((P5)\) to be optimized. This network is no different in structure for \( \tau > 0 \), and as a consequence, optimizing for both instantaneous velocity, and derivatives for \( \tau > 0 \), would require a careful balancing act.

We remark that our network bears similarity to prior work on flow map representations \([17, 18]\). In particular, the distinction between spatiotemporal coordinates and time span is considered by Han et al. \([17]\), yet the ability to distinguish properties of the flow map for \( \tau = 0 \) time span is not studied. Our network design is inspired by Bíloš et al. \([18]\), where they similarly consider residual connections. However, we make more precise the role of residual architectures in regards to flow map derivatives, and the relationship with the vector field, this being the only source of supervision in our work.

What remains is a specific instantiation of functions \( f_\nu \) and \( f_\tau \). Though in principle these could be arbitrary neural networks, in practice we adapt prior work on learnable feature grids \([19, 48, 49]\), where the parameters of the model are, in part, comprised of learnable spatiotemporal feature grids, each of varying resolution. For each feature grid we perform linear interpolation to obtain a feature vector, and concatenate the vectors obtained across all grids. We then apply a shallow MLP to the concatenated vector. For simplicity, grid cells are accessed exactly, rather than hashed as proposed in Müller et al. \([19]\). The bulk of the network parameters thus lie in the feature grids, rather than MLP weights, and so in practice the dimensionality \( d \) need not be too large – in practice we set \( d = 64 \). As a result, the cost of evaluating the network is inexpensive, requiring (1) interpolation of feature vectors from a set of grids, and (2) applying several matrix-vector multiplication operations (c.f. Eqs. 18–21).

### 3.3 Optimization Scheme

Our optimization scheme proceeds in two phases. In the first phase we optimize for the flow map’s instantaneous velocity, while in the second phase we optimize for the self-consistency criterion, in order to learn the flow map over the full spatiotemporal domain, and varying time spans.

**Vector field optimization.** To find the flow map’s instantaneous velocity, we minimize the following objective:

\[
\mathcal{L}_\nu = \mathbb{E}_{(x,t) \in \mathcal{D}} \left[ \| W^{(L)} (m^{(0)} \circ f_\nu(x, t)) - \nu(x, t) \| \right].
\]  

(15)

This amounts to optimizing over the parameters of \( f_\nu \), e.g. the multi-level feature grid, shallow MLP, vector \( m^{(0)} \), and final projection \( W^{(L)} \). We emphasize that it is only this phase of optimization that requires the vector field for supervision. The relevant portion of the model (c.f. Fig. 2(c)) can encode the vector field in a persistent manner, even as...
we optimize for the flow map in the subsequent phase, and thus we may discard the vector field post optimization. For large-scale vector fields that may not fit in memory, this gives us the opportunity to learn a compressed vector field representation, e.g. one that can fit in memory for use in the next optimization phase, as well as at inference time.

Flow map optimization. To learn the flow map, we are guided by the proposed self-consistency criterion of Eq. (7). Although we often find taking just a single step is sufficient for giving accurate flow maps, for certain datasets, we find it useful to instead take multiple steps in optimizing this loss. More specifically, we define $\bar{\Phi}(x, t, \tau) = (\Phi(x, t, \varepsilon), t + \varepsilon, \varepsilon)$ to be the resulting position, subsequent time step, and time span $\varepsilon$, from applying the flow map. Then for some target time span $\tau$, we can compose the flow map into multiple steps $k \in \mathbb{Z}^+$ as follows:

$$\tilde{\Phi}_k(x, t, \tau) = \tilde{\Phi} \circ \tilde{\Phi} \circ \cdots \circ \tilde{\Phi}(x, t, \frac{\tau}{k}).$$

We then redefine our self-consistency loss as:

$$l_\varepsilon(x, t, \tau) = \left\| \frac{\partial \tilde{\Phi}(x, t, \tau)}{\partial \tau} - \frac{\partial \tilde{\Phi}(\tilde{\Phi}_k(x, t, \tau), t, \tau + \varepsilon, 0)}{\partial \tau} \right\|.$$

4 Results

In this section we experimentally evaluate our method – herein termed NIFM for Neural Integration-free Flow Maps – for both 2D and 3D time-varying vector fields, comparing against various baselines that accelerate flow map computation in different ways. A requirement that is common to all baselines is access to samples of the flow map. Unless otherwise stated (c.f. Sec. 4.4), the methods against which we compare NIFM are based on flow maps generated via 4th order Runge-Kutta integration (RK4), with step size set to half of the temporal voxel size. We also use this very integration scheme to generate ground-truth flow map samples for the purposes of evaluation. In Table 1, we list the datasets used for comparison purposes. Further, all reported computational timings are based on a system with 12-core CPU AMD Ryzen 9 3900X, 16GB RAM, and GPU NVIDIA GeForce RTX 2080 Ti with 12GB memory.

We consider the flow map super resolution technique proposed by Jakob et al. [26], wherein we train a convolutional neural network (CNN) model using the 2D fluid flow dataset provided by the authors. To train the CNN we generate 16x downsampled flow maps along with their corresponding high-resolution ground truth flow maps, varying start times and time span of the integration, to permit model generalization for arbitrary start time/duration.

Additionally, we compare our method with the deep learning based Lagrangian interpolation technique proposed by Han et al. [17]. This technique uses an encoder-decoder network and is most similar to ours in terms of the input data the model expects, and the output of the model. We train the model on flow map samples computed by, first, generating seeds sampled uniformly at random in space and time, and secondly, integrating for varying small time spans. This flow map sampling technique is intended to resemble the Lagrangian short generation scheme proposed by the authors. We made a minor modification to the network by removing the ReLU activation function used in the output layer, allowing the model to output negative values. Further, we compare our method with a SIREN [45] that tacks time span on as an additional coordinate, along with particle space-time coordinates (c.f. Fig. 2(a)). We train the SIREN with the same data used to train the encoder-decoder model. Note that we could use a hybrid grid-MLP model [19, 49] in lieu of a standard coordinate-based MLP, but for 3D unsteady flows this would require storage of a 5D grid, which is not feasible.

We also compare our method against the recent work by Li et al. [29], where the authors showed an improvement over prior work in efficiently interpolating Lagrangian representation to obtain new trajectories. Note that the representation of flow in our datasets is Eulerian, whereas Li et al. works with particle-based data, thus, requiring a conversion from the former to the latter. For a fair comparison, we convert the Eulerian representation into a Lagrangian one by first placing $n_s$ number of seeds in the domain uniformly at random, where $n_s$ is the spatial resolution of the vector field data, and integrate these seed points via RK4. The temporal frequency with which we store particle positions is set as the temporal resolution of the field. Furthermore, the Lagrangian representation is limited to the temporal
duration on which we are evaluating, to have a better distribution of particles throughout the domain.

Last, we compare our method with the streakline vector field (SVF) work of Weinkauf et al. [16]. Specifically, the SVF is first precomputed by estimating flow map derivatives, computed via RK4, and then at runtime streaklines are generated by integrating the SVF. We view this as a fair comparison to our technique in that both approaches incur a precomputation cost, and thus we aim to compare the computation and storage requirement for the representations, as well as the accuracy and computation efficiency for generating streaklines.

4.1 Implementation details

We first describe the details of our network architecture, followed by details on optimization.

Network architecture settings

The design of \( f_{\nu} \) and \( f_{\tau} \) rely on parameter settings related to the multi-level feature grid, as well as the MLP. The feature grids for \( f_{\nu} \) and \( f_{\tau} \) are of identical design, where we use a 4-level feature grid, and each level is of a different spatial resolution. Specifically, for a given axis of resolution \( w \) at level \( l \), we set the resolution at the next level to be \( w^{\frac{1}{l}} \), with resolution scaling factor \( s \) set to 1.65, following the guidance of Müller et al. [19]. Each grid stores 8-dimensional feature vectors at its nodes, and thus the resulting concatenated feature is 32-dimensional.

We employ 2 and 1-layer MLPs for \( f_{\nu} \) and \( f_{\tau} \), respectively, along with activation \( \sigma \), chosen to be a Swish activation [50]. Experimentally we found Swish to outperform other more standard activations for INRs, e.g. ReLU, sin, consistent with findings in Autoint [43]. We control for the size of the network by a compression ratio, expressed as the ratio of the vector field size to the network size. We adjust the spatial network by a compression ratio, expressed as the ratio of the vector field size to the network size. We adjust the spatial

4.2 2D unsteady flow

We first conduct experimental comparisons for various 2D time-varying flow fields. Specifically, we evaluate different techniques by comparing the error in flow map approximations over varying seed points (spatial position and starting time) that have been integrated for varying time spans. We express error as the averaged Euclidean distance between the ground-truth flow map output, and the approximation scheme’s output, normalized by the domain’s bounding-box diagonal length. In Fig. 4, we present quantitative results comparing our method against different baselines, and in Table 2 we report inference and preprocessing times. Specifically, for the pathline interpolation approach of Li et al. [29], preprocessing refers to the time required to fit B-splines, while for Jakob et al. [36], this refers to the time required to optimize the CNN for super resolution. For all remaining methods, preprocessing refers to the time required for optimizing to an individual flow field.

In comparing the fluid simulation flows of varying Reynolds numbers, we find that our method sees consistent improvement in accuracy over SIREN and super resolution, while achieving faster inference times. We note that the super resolution approach requires optimizing a CNN over a collection of flow maps just once, and thus can find that both give results of comparable accuracy, in some occasions we found that fine-tuning can mitigate small grid-based artifacts in the output when leaving these weights frozen, and hence we fine-tune this portion of the network, using a learning rate of 0.0008.

Recall that our method supports a maximum time span \( \tau_{\max} \) on which to sample during optimization. Though in principle we could optimize for the full time span of a given dataset, we find that performance can suffer, especially for datasets exhibiting complex temporal dynamics. Thus, as a compromise we set a limit on \( \tau_{\max} \) during optimization, and at inference time, for any target \( \tau > \tau_{\max} \) we take multiple steps with our network until reaching the desired span \( \tau \). Specifically, for all 2D datasets, expressed in terms of grid units we set \( \tau_{\max} = 48 \) unless otherwise specified. For 3D datasets we customize \( \tau_{\max} \) based on grid resolution, and complexity of the flows.

| Dataset | FTLR res | \( \tau \) | Reference time(s) | Preprocessing time(min) | CR | Storage (MB) | method |
|---------|----------|----------|------------------|------------------------|----|-------------|-------|
| Fluid Sim | 512x512 | 7 | 22.185 | 10 | 48.05 | GT |
| Cylinder | 1200x150 | 1 | 0.985 | 10 | 16 | NFDM |
| Boussinesq | 450x150 | 0.5 | 0.59 | 10 | 97 | NFDM |
| Double Gyre | 1200x600 | 10 | 1.089 | 10 | 29 | NFDM |

| | | | | | |
| | | | | | |

| Dataset | Res [Lx,Y,Lx] | | | | |
|---------|----------------|------------------|------------------|----------------|-------------|
| Double Gyre | 500x400x200 | Cylinder | 1001x400x50 | Boussinesq | 2001x450x150 | Fluid Simulation | 1001x512x512 | Tornado | 50x128x128x128 | Scalar Flow | 151x100x178x100 | Thinflow | 151x640x240x80 | |

| Table 1
We list all datasets and their respective sizes used in experiments.

| Table 2
We report the preprocessing times for different methods across 2D unsteady flows, along with corresponding timings for FTLR computation, varying time span and image resolution.

| Dataset | FTLR res | \( \tau \) | Reference time(s) | Preprocessing time(min) | CR | Storage (MB) | method |
|---------|----------|----------|------------------|------------------------|----|-------------|-------|
| Fluid Sim | 512x512 | 7 | 22.185 | 10 | 48.05 | GT |
| Cylinder | 1200x150 | 1 | 0.985 | 10 | 16 | NFDM |
| Boussinesq | 450x150 | 0.5 | 0.59 | 10 | 97 | NFDM |
| Double Gyre | 1200x600 | 10 | 1.089 | 10 | 29 | NFDM |
generalize to low-resolution flow maps at inference time, albeit restricted to flows resembling those observed during training. Our method is limited to just a single dataset at a time, but nevertheless, our training times scale well in terms of standard INRs (e.g., SIREN), while exhibiting faster inference and more accurate flow map approximations. Qualitative results for the fluid simulation flows are shown in Fig. 3 in the form of the FTLE – computed using the method of Haller [7] – and color-encoded flow map errors. For high Reynolds number flows, we see that the super resolution method can fail to adapt to the rate at which particles separate, as indicated by the color shift, while also blurring out detailed ridges in the FTLE. Our method, however, excels in capturing FTLE ridges, while remaining efficient to compute, since the super resolution method still requires computing a low-resolution flow map as input to a (otherwise highly efficient) CNN. Recall that our method employs a compression ratio of 10 for all 2D experiments, which limits the grid resolution, and thus might limit the details we can reproduce in the flow map. However, from these results, we see that the coarser feature grid resolution does not limit the spatial resolution of the FTLE.

In comparing our method to other baselines (c.f. Fig. 4) for Double Gyre, Cylinder, and Boussinesq, we find that our method obtains higher accuracy in relation to other techniques. Prior INR methods such as the encoder-decoder...
We report storage requirements, preprocessing time and inference time for computing streaklines on the Cylinder dataset, comparing our method against the streakline vector field technique [16].

| Method               | Preprocessing Time (min) | Inference Time (sec) | Storage |
|----------------------|--------------------------|----------------------|---------|
| Ground Truth         | NA                       | 21.391               | 160.20 MB |
| SVF                  | 130.407                  | 1.204                | 160.36 GB |
| NIFM (16 grid steps) | 40.060                   | 0.952                | 77.20 MB  |

We report the processing times as well the FTLE computation times for different method across different 3D unsteady flow datasets.

| Dataset            | FTLE res | τ | Inference times (s) | Processing times(min) | CR | Method       |
|--------------------|----------|---|---------------------|-----------------------|----|--------------|
| Tornado            | 128x128x128 | 50 | 2.76                | -                     | -  | GT           |
|                    |          |    | 3.80                | 35.55                 | 10 | NIFM         |
|                    |          |    | 14.74               | 33.23                 | 10 | SIREN        |
|                    |          |    | 236.27              | 13.9                 | -  | Spline       |
|                    |          |    | 81.72               | -                     | -  | GT           |
|                    |          |    | 2.85                | 41.86                 | 10 | NIFM         |
|                    |          |    | 21.38               | 95.79                 | 10 | SIREN        |
|                    |          |    | 291.9               | 0.81                  | -  | Spline       |
|                    |          |    | 137.41              | -                     | -  | GT           |
|                    |          |    | 41.77               | 45.96                 | 40 | NIFM         |
|                    |          |    | 35.22               | 104.13                | 40 | SIREN        |
|                    |          |    | 29.6                 | 65.5                  | -  | Spline       |
|                    |          |    | 38.9                 | 103.14                | -  | GT           |
| Scalar Flow        | 100x178x100 | 2.5 | 137.41              | -                     | -  | GT           |
|                    |          |    | 41.77               | 45.96                 | 40 | NIFM         |
|                    |          |    | 35.22               | 104.13                | 40 | SIREN        |
| Half-Cylinder      | 640x240x80 | 2 | 137.41              | -                     | -  | GT           |
|                    |          |    | 41.77               | 45.96                 | 40 | NIFM         |
|                    |          |    | 35.22               | 104.13                | 40 | SIREN        |

architecture of Han et al. [17], or a pure coordinate-based approach [45] poorly generalize. We find that for small step sizes, the performance of these methods in fact steeply declines, as numerical error accumulates with the more steps taken. We attribute this to the basic limitations of the network architectures employed, failing to address the properties (identity mapping, instantaneous velocity) we target in our network design. The inability to generalize in these methods is further demonstrated qualitatively for Figs. 6 - 8. Pathline interpolation [29] is notable in its small precomputation cost. Nevertheless, the method is less accurate in preserving the flow map, while incurring a high computation cost at runtime.

We additionally evaluate our technique both quantitatively and qualitatively for the computation of streaklines. In Fig. 9 we show streaklines for the Cylinder dataset. We compare our method with SVF [16]. We can see that both the techniques are able to capture the vortices of the dataset faithfully, and are visually indistinguishable from the ground truth streaklines. Quantitatively both the techniques consistently incur low streakline error staying within the margin of $10^{-5}$ magnitude (relative to the bounding box diagonal). Interestingly, we find that both methods have comparable inference time as well, as reported in Table 5 despite the fact the streakline vector field evaluates its field fewer times than our neural flow map, since we must take multiple steps for sufficiently long time spans. However, an advantage of our method lies in data parallelism; we can evaluate the flow map over varying space/time/duration in a single batch, whereas integrating the streakline vector field is, by necessity, a sequential process. We further note that SVF precomputation is quite expensive, both in terms of speed and storage space. In Table 3 we can see that the computation of the entire 4D SVF has very large storage requirements (160GB), whereas our method is in proportion to the size of the vector field (77MB). We note that while our technique can be easily scaled to 3D datasets, SVF preprocessing for 3D unsteady flows is infeasible in practice, necessitating a 5D grid for storage.
Fig. 9. We compare our method’s ability to compute streaklines against the streakline vector field technique [16], which only necessitates integrating a derived vector field. Qualitatively and quantitatively we find that our method produces comparable results, where we show varying step sizes used for evaluating the flow map.

Fig. 10. We compare, both qualitatively (volume rendering of FTLE field) and quantitatively (flow map evaluation), our method with standard coordinate-based networks [45] as well as pathline interpolation techniques [29] for modeling the flow map in 3D unsteady flows. We find our method is quantitatively an improvement over other methods, and qualitatively our method contains fewer visual artifacts.

Fig. 11. In this figure, we compare our method both quantitatively and qualitatively against SIREN for the Half-Cylinder dataset. We find that our method is able to scale reasonably well to this large dataset, whereas, the SIREN fails to learn meaningful flow maps as can be seen from the FTLE.

4.3 3D unsteady flow

We next evaluate our method on a set of 3D unsteady flows, comparing our method with a SIREN-based flow map [45] as well as the B-spline pathline interpolation technique [29]. We first compare to the Tornado and Scalar Flow datasets, where we set the $\tau_{\text{max}}$ to 8 and 24, respectively, to match the temporal complexity in the flows. Fig. [1] shows qualitative results, via volume-rendering of the FTLE, as well as quantitative results. Our method is an improvement, if not comparable, to prior methods, but we obtain significant gains in inference time, as reported in Table [4]. We further compare to the Half Cylinder dataset, a large-scale unsteady flow dataset that cannot be readily stored in memory. We found the pathline interpolation method [29] failed to fit to the data, and thus we limit our comparison to SIREN, please see Fig. [11]. In this experiment we set $\tau_{\text{max}} = 8$ and the compression ratio to 40 to compensate for the larger data size. We find our method captures turbulent features in the wake of the half-cylinder object ($Re = 320$), whereas SIREN faces difficulties in accurately modeling the data. Notably, for this dataset we find our training scheme scales well (c.f. Table [4]) relative to the 2D unsteady flow datasets, whereas SIREN's increase in model size leads to slower training times.
4.4 Error analysis: numerical integration

Our method can be viewed as a novel technique for integrating a vector field, and thus, it is worth asking: how does our method compare to conventional numerical integration schemes? To help answer this question, we compare NIFM to existing numerical schemes, namely Euler and RK4, evaluated under varying step sizes. For the purpose of evaluation we use the Sine Ridge dataset provided by Kuhn et al. [?] - as this is a steady flow we adapt our method accordingly. The dataset has an analytically-defined flow map that allows us to compute the flow map error across different schemes. In Fig. 12, we show the FTLE (first row) and the flow map error (second row) for Euler, RK4, and NIFM. The FTLE is computed for a duration $\tau = 1.2$ with step size set to 30, where a single step amounts to 0.01 in the physical domain. We can see that NIFM best captures the FTLE, while maintaining low error in the flow map, in contrast with Euler and RK4. This provides evidence that our method is not merely a fixed linear (e.g. Euler), or higher-order (e.g. RK4) integration scheme, but rather adapts to the features of the data. We further show quantitative results for duration 0.6 and 1.2, again varying the step size. We can see that while NIFM has a consistent performance across all step sizes, the flow map error increases significantly for both RK4 and Euler with increasing step size.

4.5 Ablation: compression and supervision

Last, we run model ablations to study the effects of various design choices. Due to space limitations we limit ablation to compression, as well as the role of supervision in learning flow maps. Further experiments regarding the architecture choices (number of levels in the multiresolution grid) and optimization scheme (number of steps to take, c.f. Eq. 16) are detailed in the appendix.

Fig. 13. We qualitatively compare our model under varying compression ratios, showing the effect of compression on the step size taken by our model to produce the FTLE for the Boussinesq flow.

In Fig. 13, we show the results of our model, for the FTLE of the Boussinesq, optimized under varying compression ratios. In this experiment we specifically wish to study how compression might impart visual artifacts in derived quantities of the flow map approximation, as a higher level of compression results in coarser feature grids. Indeed, we find that lower levels of compression lead to fewer grid-like artifacts in the resulting FTLE when taking a smaller steps, e.g. in this setting, a step size of 48 grid units in time amounts to an evaluation of the model just 3 times per position. We further report inference times for the smallest and largest level of compressions, and as expected, a larger number of steps requires longer inference times (e.g. more feedforward passes with the network). Interestingly, we find the inference time is fairly consistent across these compression ratios, indicating that the increased resolution of the grid has a negligible impact on this matter. As detailed in the appendix, we also find that the flow map accuracy takes just a small hit in performance across compression ratios, indicating that flow map accuracy might not be predictive of visual artifacts in derived quantities. Nevertheless, as shown in the figure, training times come at a cost with smaller compression ratios. We thus see natural trade-offs in the (1) flow map quality, (2) inference time (hinging on step size), and (3) training time.

Our choice to learn flow maps via a self-supervisory signal is in contrast with how numerous visualization techniques interpolate [21, 29], or build models [17] given samples of the flow map, e.g. typically as densely-sampled
pathlines. Therefore we ask: is our self-consistency criterion an inferior objective to directly supervising on flow map samples? To this end, we have gathered a large collection of flow map samples, and modified our objective (Eq. 7) to accept the ground-truth flow map, and its corresponding derivative at the output position. We optimize for Boussinesq, using 20M and 50M flow map samples, and compare with our proposed objective, please see Fig. 14 for the results. We find that our method is able to learn comparable, if not better, flow map approximations, without ever observing flow map samples. In particular, at 50M samples we find that flow map supervision starts to become competitive with our method. Although supervising an on even larger number of samples might be more beneficial, clearly the data requirement starts to become prohibitively expensive, both for integrating the flow field, as well as storage requirements. In contrast, our method avoids these issues by requiring the vector field as the only supervision.

5 Discussion

In this paper we have presented an approach for integration free learning of flow maps, where we use coordinate-based neural networks as surrogates for fast and accurate flow map computation. We achieve this through our novel network design and optimization scheme that takes advantage of the basic properties of flow maps, in order to learn only using the provided vector field. We demonstrate the strength of our technique experimentally by comparing our method with various baselines and across multiple datasets.

There are several research directions we intend to pursue for future work. First, we acknowledge that although our method is scalable to optimize relative to other methods, optimization remains the computational bottleneck. We expect that porting our optimization scheme to the GPU, using fully-fused CUDA kernels for both the grid and MLPs, will alleviate this cost, as studied in prior works [19], [49]. Additionally, our self-consistency scheme is only an approximation, whereas other approaches have studied the design of invertible neural networks for computing discrete [52] or continuous [18], [37] mappings of learned representations. We believe that adopting such approaches for representing flow maps in 2D and 3D unsteady flows is a fruitful research avenue.

References

[1] G. Haller, “Finding finite-time invariant manifolds in two-dimensional velocity fields,” Chaos: An Interdisciplinary Journal of Nonlinear Science, vol. 10, no. 1, pp. 99–108, 2000.
[2] G. Haller and G. Yuan, “Lagrangian coherent structures and mixing in two-dimensional turbulence,” Physica D: Nonlinear Phenomena, vol. 147, no. 3–4, pp. 352–370, 2000.
[3] B. Cabral and L. C. Leedom, “Imaging vector fields using line integral convolution,” in Proceedings of the 20th annual conference on Computer graphics and interactive techniques, 1993, pp. 263–270.
[4] G. Froyland and K. Padberg, “Almost-invariant sets and invariant manifolds—connecting probabilistic and geometric descriptions of coherent structures in flows,” Physica D: Nonlinear Phenomena, vol. 238, no. 16, pp. 1507–1523, 2009.
[5] V. Artale, G. Botti, A. Celani, M. Cencini, and A. Vulpiani, “Dispersion of passive tracers in closed basins: Beyond the diffusion coefficient,” Physics of Fluids, vol. 9, no. 11, pp. 3162–3171, 1997.
[6] E. Aurell, G. Botti, A. Crisanti, G. Paladin, and A. Vulpiani, “Predictability in the large: an extension of the concept of lyapunov exponent,” Journal of Physics A: Mathematical and General, vol. 30, no. 1, p. 1, 1997.
[7] G. You and S. Leung, “An eulerian method for computing the coherent ergodic partition of continuous dynamical systems,” Journal of Computational Physics, vol. 264, pp. 112–132, 2014.
[8] C. Garth, F. Gerhardt, X. Tricoche, and H. Hans, “Efficient computation and visualization of coherent structures in fluid flow applications,” IEEE Transactions on Visualization and Computer Graphics, vol. 13, no. 6, pp. 1464–1471, 2007.
[9] J. Kasten, C. Petz, I. Hotz, B. R. Noack, and H.-C. Hege, “Localized finite-time lyapunov exponent for unsteady flow analysis.” in IMAV, 2009, pp. 265–270.
[10] F. Sadlo and R. Peikert, “Visualizing lagrangian coherent structures and comparison to vector field topology,” in Topology-Based Methods in Visualization II. Springer, 2009, pp. 15–29.
[11] S. L. Brunton and C. W. Rowley, “Fast computation of finite-time lyapunov exponent fields for unsteady flows,” Chaos: An Interdisciplinary Journal of Nonlinear Science, vol. 20, no. 1, p. 017503, 2010.
[12] D. Lipinski and K. Mobseni, “A ridge tracking algorithm and error estimate for efficient computation of lagrangian coherent structures,” Chaos: An Interdisciplinary Journal of Nonlinear Science, vol. 20, no. 1, p. 017504, 2010.
[13] F. Sadlo, A. Rigazzi, and R. Peikert, “Time-dependent visualization of lagrangian coherent structures by grid advection,” in Topological Methods in Data Analysis and Visualization. Springer, 2011, pp. 151–165.
[14] D. Stalling and H.-C. Hege, “Fast and resolution independent line integral convolution,” in Proceedings of the 22nd annual conference on Computer graphics and interactive techniques, 1995, pp. 249–256.
[15] H. Battke, D. Stalling, and H.-C. Hege, “Fast line integral convolution for arbitrary surfaces in 3d,” in Visualization and Mathematics. Springer, 1997, pp. 183–193.
[16] T. Weinkauf and H. Theisel, “Streak lines as tangent curves of a derived vector field,” IEEE Transactions on Visualization and Computer Graphics, vol. 16, no. 6, pp. 1225–1234, 2010.
[17] M. Han, S. Sane, and C. R. Johnson, “Exploratory lagrangian-based particle tracing using deep learning,” arXiv preprint arXiv:2110.08338, 2021.
[18] M. Blash, J. Sommer, S. S. Rangapuram, T. Januschowski, and S. Günnemann, “Neural flows: Efficient alternative to neural odes,” Advances in Neural Information Processing Systems, vol. 34, pp. 21 325–21 337, 2021.
[19] T. Müller, A. Evans, C. Schied, and A. Keller, “Instant neural graphics primitives with a multiresolution hash encoding,” arXiv preprint arXiv:2201.05989, 2022.
[20] A. Agranovskv, D. Camp, C. Garth, E. W. Bethel, K. I. Joy, and H. Childs, “Improved post hoc flow analysis via lagrangian representations,” in 2014 IEEE 4th Symposium on Large Data Analysis and Visualization (LDAV). IEEE, 2014, pp. 67–75.
[21] J. Chandler, H. Obermaier, and K. I. Joy, “Interpolation-based pathline tracing in particle-based flow visualization,” IEEE transactions on visualization and computer graphics, vol. 21, no. 1, pp. 68–80, 2014.
[22] A. Agranovskv, H. Obermaier, C. Garth, and K. I. Joy. “A multi-resolution interpolation scheme for pathline based lagrangian flow representations,” in Visualization and Data Analysis 2015, vol. 9397. SPIE, 2015, pp. 221–232.
**APPENDIX A**

**FLOW MAP INSTANTANEOUS VELOCITY**

In this appendix we show how our network design leads to a simple form for the flow map’s instantaneous velocity.

First, we recall the specific form of our network. For clarity in derivations, we explicitly denote the dependency on time span \( \tau \), and omit spatial position \( \mathbf{x} \) and starting time \( t \) where necessary. The first layer produces the following:

\[
\mathbf{z}^{(0)}(\tau) = \sigma_\nu(\tau \mathbf{m}^{(0)}) \odot f_\nu(\mathbf{x}, t).
\]

The function \( f_\nu : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^d \) is a \( d \)-dimensional spatiotemporal representation, for our purposes this is an arbitrary neural network. Vector \( \mathbf{m}^{(0)} \in \mathbb{R}^d \) aims to reconcile domain-specific scaling, \( \sigma_\nu \) is an activation function, and \( \odot \) indicates element-wise multiplication. The second function \( f_\tau : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^d \) also learns a \( d \)-dimensional spatiotemporal representation, one specific to the flow map for nonzero time spans. These two representations are combined to give us the next layer’s output:

\[
\mathbf{z}^{(1)}(\tau) = \mathbf{z}^{(0)}(\tau) + \sigma_\nu(\tau \mathbf{m}^{(1)}) \odot \sigma_\tau\left(\mathbf{z}^{(0)}(\tau) \odot (W^{(1)} f_\tau(\mathbf{x}, t))\right),
\]

where \( \mathbf{m}^{(1)} \in \mathbb{R}^d \) serves the same purpose as \( \mathbf{m}^{(0)} \), and \( W^{(1)} \in \mathbb{R}^{d \times d} \) is a learnable linear transformation. Subsequent representations are formed via residual connections:

\[
\mathbf{z}^{(l)}(\tau) = \mathbf{z}^{(l-1)}(\tau) + \sigma_\nu(\tau \mathbf{m}^{(l)}) \odot \sigma_\tau\left(W^{(l)} \mathbf{z}^{(l-1)}(\tau)\right),
\]

while the last layer \( L \) applies a single linear transformation to give us the output position, wherein we also include a skip connection for the input position:

\[
\hat{\Phi}(\mathbf{x}, t, \tau) = \mathbf{x} + W^{(L)} \mathbf{z}^{(L-1)}(\tau).
\]

We further make the following assumptions on activation functions \( \sigma_\nu \) and \( \sigma_\tau \):

\[
\begin{align*}
\sigma_\nu(0) &= 0 \\
\sigma_\nu'(0) &\neq 0 \\
\sigma_\tau(0) &= 0
\end{align*}
\]

We aim to compute the derivative of the neural flow map at time span \( \tau = 0 \):

\[
\frac{d\hat{\Phi}(\mathbf{x}, t, 0)}{d\tau} = \frac{d\hat{\Phi}}{dz^{(L-1)}} \frac{dz^{(L-1)}(0)}{d\tau}.
\]

The term on the left is simply:

\[
\frac{d\hat{\Phi}}{dz^{(L-1)}} = W^{(L)}
\]

As for the term on the right, we have the following recurrence for \( l > 1 \):

\[
\frac{dz^{(l)}(0)}{d\tau} = \frac{dz^{(l-1)}(0)}{d\tau} + \sigma_\nu'(0) \mathbf{m}^{(l)} \odot \sigma_\tau(W^{(l)} z^{(l-1)})
\]

\[
+ \sigma_\nu(0) \frac{d\sigma_\tau(W^{(l)} z^{(l-1)})}{d\tau}.
\]

The term in the second line will evaluate to zero, since activation vectors \( z^{(l)}(0) \), for any layer \( l \), will be zero due to the multiplicative scaling with \( \tau = 0 \), and the fact that \( \sigma_\tau(0) = 0 \). Likewise, the third line will vanish due to our assumption on the activation function evaluating to \( \sigma_\nu(0) = 0 \). A similar reasoning can be applied for layer \( l = 1 \) (cf. Eq. 19), due to the multiplicative scaling of \( z^{(0)}(0) \) with \( W^{(1)} f_\tau(\mathbf{x}, t) \). Thus, we have the following:

\[
\frac{dz^{(L-1)}(0)}{d\tau} = \frac{dz^{(0)}(0)}{d\tau} = \left(\sigma_\nu(0) \mathbf{m}^{(0)}\right) \odot f_\nu(\mathbf{x}, t).
\]

As we choose \( \sigma_\nu \) to be a hyperbolic tangent function, we obtain \( \sigma_\nu'(0) = 1 \). Thus, plugging Eqs. 23 and 25 into Eq. 22 we arrive at:

\[
\frac{d\hat{\Phi}(\mathbf{x}, t, 0)}{d\tau} = W^{(L)}(\mathbf{m}^{(0)} \odot f_\nu(\mathbf{x}, t)).
\]

Note that in Eq. 19 the multiplicative scaling within the activation \( \sigma_\tau \) is essential for this result - a different way of combining the representations, e.g. adding them together, would introduce dependencies on \( f_\tau \) and weight matrices \( W^{(l)}, l > 0 \), in computing the derivative.
Fig. 18. We vary the number of grid levels used in the multiresolution feature grid for Boussinesq, and find that our method is robust to this particular hyperparameter setting.

Fig. 19. For the Boussinesq flow we compare the effects of $\tau_{\text{max}}$ on the overall performance of the model, finding that the self-consistency criterion when trained with large $\tau_{\text{max}}$ gracefully degrades in performance, indicating the stability of our optimization technique.

APPENDIX B

ABLATION RESULTS

We include model/optimization ablation results to demonstrate the robustness of our method across a variety of parameter settings. In all experiments the maximum time span $\tau_{\text{max}}$ is set to 48 (in grid units), and we use default parameters as originally specified in the paper, unless otherwise mentioned.

Figs. 15 and 16 compare different strategies for the number of steps taken by our method in optimizing the self-consistency criterion: one step per grid unit, a square root scaling (the default choice used throughout all results in the paper), a log scaling, as well as taking just a single step. As shown in the figures, across datasets we find little difference in the results, evaluated across varying step sizes. As a compromise, we set the square root scaling as it adds little computation cost, while ensuring additional stability in optimization.

In Fig. 17 we study the effect of model size, e.g. compression ratio, on accuracy for the Boussinesq flow. In general we find a small drop in accuracy, suggesting that our model can generalize well even when utilizing a smaller number of parameters. In Fig. 18 we study the impact on the number of grid levels used for our multiresolution feature grid. Although we default the number of levels to 4 in the main paper, in general, we find little difference in quality as we adjust the number of levels. In Fig. 19 we study the impact of $\tau_{\text{max}}$ on the performance of the model. We evaluate the model by taking the minimum between the integration duration and the $\tau_{\text{max}}$ the model was trained on as the step size. We found that when trained with large values of $\tau_{\text{max}}$ the overall performance of the model degrades affecting both smaller and larger timespans. Last, in Fig. 20 we study the performance of NIFM when the integration duration is large. We perform this experiment on the double gyre dataset and evaluate the model for $\tau = 20$ and $\tau = 30$ respectively. From the FTLE and its corresponding flow map errors, we can see that even for long integration duration the model performs reasonably well.