NEURAL REPRESENTATIONS OF FLOW DATA FOR VISUAL ANALYSIS

By

Saroj Kumar Sahoo

Dissertation

Submitted to the Faculty of the Graduate School of Vanderbilt University in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Computer Science

August 11, 2023

Nashville, Tennessee

Approved:

Dr. Matthew Berger, Ph.D.Dr. Gautam Biswas, Ph.D.Dr. Taylor Johnson, Ph.D.

Dr. David Hyde, Ph.D.

Dr. Joshua Levine, Ph.D.

Copyright © Saroj Sahoo All Rights Reserved

ACKNOWLEDGMENTS

First and foremost I would like to thank my parents. I am incredibly lucky to have been raised by them and I will be forever indebted to them for the sacrifices they have made for me. I wouldn't be here without their everlasting love and support.

I am incredibly thankful for my wonderful sisters and the meaningful conversations we shared about health, life, and philosophy. Those moments we spent together are truly invaluable. I cannot express enough gratitude for the constant presence and unwavering support I got whenever I needed it. They inspire me to be a better and kind person.

I want to express my heartfelt gratitude to my dear friends Sanket, Vikrant, Sarthak, and Ansuman. I am immensely grateful for the years of friendship that we share. Thank you for being there for me, patiently listening to my rants and frustrations about the setbacks and challenges I faced.

I would like to thank Suveni, for her selflessness and kindness in providing me with rides to and from the lab and the grocery store during times of need. Her generosity and willingness to help have made a significant impact on my life and I cannot thank her enough for her invaluable assistance.

Next, I would like to thank Azhar, for being an awesome and caring roommate. Thanks for all the times you cooked food for me while I was busy working and trying to meet a deadline. I am grateful to have a friend like you.

Thank you to my lab mates - Sangwon, Ava, and Bryan for their friendship, the shared meals, and countless discussions about research progress.

I want to express my deepest gratitude to Joslyn for her love and support. For being there with me during the good and bad times. Her care and dedication during times of sickness have been a true testament to her kindness and compassion. Her unwavering encouragement has uplifted me during times of doubt and uncertainty, pushing me to reach for my dreams. I consider myself incredibly fortunate to have crossed paths with Joslyn, and I cherish the moments we share together.

I would like to express my gratitude to my committee members Dr. David Hyde, Dr. Taylor Johnson, Dr. Joshua Levine, and Dr. Gautam Biswas for their technical guidance and insightful feedback that greatly impacted this work. A special thanks to Dr. Joshua Levine for hopping on Zoom calls with me when I was stuck and needed his expertise. I am deeply grateful for his willingness to share his knowledge and the time he dedicated to helping me.

Last and certainly not least I would like to thank my advisor, Matthew Berger. My Ph.D. journey would not have been possible without his invaluable guidance and mentorship. Since joining his lab, I have witnessed tremendous personal and professional growth in myself, and I owe it to his expertise and unwavering support. Matt is the epitome of hard work, and his dedication serves as a constant inspiration to all of us in the lab. I am truly grateful for the countless ways he has helped and supported me throughout this journey. His guidance has been instrumental in shaping me as a person and my research path, and I cannot stress enough how appreciative I am of his assistance.

TABLE OF CONTENTS

		Pa	age
AC	CKNO	WLEDGMENTS	iii
LI	ST O	TABLES	vi
LI	ST O	FIGURES	vii
1	Int	oduction	1
	1.1 1.2 1.3 1.4	Flow Specification	2 3 6 7
2	Ba	kground and Related Work	10
3	 2.1 2.2 2.3 2.4 2.5 	Flow Visualization Techniques 2.1.1 Direct Visualization 2.1.2 Texture-Based Visualization 2.1.3 Feature-Based Visualization 2.1.4 Geometry-Based Visualization 2.1.4 Geometry-Based Visualization Flow Field and Particle Trajectory Flow Map and Integral Curves 2.3.1 Flow Map 2.3.2 Integral Curves Numerical Integration Related Work	10 10 10 11 11 12 12 12 12 13 14
C	3.1 3.2 3.3 3.4 3.5 3.6 3.7	Approach Integration-Aware Upsampling Loss Implementation Details Dataset and Training Details Quantitative and Qualitative Results Hyperparameter Study Conclusion and Future Work Conclusion	19 20 21 22 23 24 25
4	Net	ral Flow Map Reconstruction	26
	4.1	Approach4.1.1Data Reduction: Flow Map4.1.2Neural Flow Map4.1.3Efficient Backpropagation4.1.3Fifting to Flow Map Samples, Fitting to Vectors4.2.1Fitting to Flow Map Samples, Fitting to Vectors4.2.2Effect of Integration Scheme4.2.3Effects of Sampling4.2.4Integration Duration	26 27 29 30 32 32 34 36 37

		4.2.5 Implementation Details
		4.2.6 2D Unsteady Flow
		4.2.7 3D Unsteady Flow 40
	4.3	Discussion
5	Sca	ale-reinforced Implicit Neural Representations
	5.1	REINFORCING INRs with SCALE
		5.1.1 Background: Implicit Neural Representation
		5.1.1.1 SIREN
		5.1.1.2 MFN
		5.1.1.3 Localized Implicit Neural Representation
		5.1.2 Fitting to a Field
		5.1.3 Learning scale space
	5.2	Analysis 52
	0.2	5.2.1 Weight Modulation as Frequency Filtering 52
		5.2.2 Proof of Theorem 5.1.1 53
		5.2.2 Frider of Theorem 5111
		5.2.2.1 MIR
	53	Distilling to a smaller network 58
	54	Results 50
	5.7	$54.1 \qquad \text{Implementation Details} \qquad \qquad$
		$5.4.1$ Implementation Details $\dots \dots \dots$
		5.4.2 Comparison
		5.4.2.1 Scale-as-Cooldinate
		5.4.2.2 Tensol-based Filtering
		5.4.5 Experimental Results
		5.4.4 Effect of Compression
		5.4.5 Network Distination
	~ ~	5.4.6 Persistence-based Scale Spaces
	5.5	Discussion
6	Int	egration-free learning of Flow Maps 72
	61	Approach 72
	0.1	611 Integration-free learning 72
		612 A network design for flow mans 76
		613 Ontimization Scheme 78
	62	Results 79
	0.2	6.2.1 Implementation details 81
		$6.2.1$ Implementation details \ldots 31
		6.2.2 2D unsteady flow
		6.2.4 Error analysis: numerical integration
		6.2.5 Ablation: compression and supervision
		6.2.6 Ablation: Model 01
		6.2.7 Eleve Man Instantaneous Valacity
	62	0.2.7 Flow Map Installatious velocity
	0.5	Discussion
7	Co	nclusion
R	eferei	nces

LIST OF TABLES

Table		Page
3.1 3.2	The dimensions and number of timesteps of each dataset	21 24
3.3	Average PSNR and ALP values for various α values for tornado dataset	24
3.4 3.5	Average PSNR and ALP values for different λ values for tornado dataset Average last position loss of streamlines (Eq. 3.6) for models trained and evaluated on	24
	different streamline lengths	25
4.1	We list the total training time in minutes, Inference time (time taken to integrate 10,000 particles for a duration of 100 grid time) in seconds, and the model size for different	
	reduction rate.	34
4.2	We list the datasets used for experimental comparisons, along with their size, the integra- tion duration we use for flow maps, and the sampling reduction rate.	36
4.3	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.	36
6.1	We list all datasets and their respective sizes used in experiments	81
6.2	We report the preprocessing times for different methods across 2D unsteady flows, along with corresponding timings for ETLE computation varying time span and image resolution	n 87
6.3	We report storage requirements, preprocessing time and inference time for computing streaklines on the Cylinder dataset, comparing our method against the streakline vector	11. 02
	field technique (143).	86
6.4	We report the processing times as well the FTLE computation times for different method across different 3D unsteady flow datasets.	87

LIST OF FIGURES

Figure		Page
1.1	Shown in the figure are graphical representations in the form of streaklines for two large- scale datasets (a) isotropic turbulence dataset - showcasing the chaotic nature of the dataset and (b) mantle dataset - illustrating the convection processes of the Earth's mantle	2
1.2 1.3	In the figure we show the space-time illustration of various integral curves	4 5
3.1 3.2	Comparison of streamline results between our technique (b), ground truth (a) and the base- line (c) for tornado dataset, differences highlighted in yellow	19
3.3	and outputs a super-resolution vector field. A content loss, alongside a streamline-based loss, between super-resolution and ground truth vector fields are used to optimize the network parameters. The use of 2d vector fields in the figure is for illustrative purpose only Comparision of the differences in streamlines of vector fields generated by different mod-	. 22
	els with respect to the ground truth highlighted in yellow. Top to bottom: square cylinder, tangaroa.	23
4.1	We show a comparison of our method, Neural Flow Map, with existing time-varying vec- tor field data reduction schemes for compression (TTHRESH) and scattered data interpo- lation (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 gen- eralization in the flow map, and consequently, faithful recovery of derived quantities such as ETLE	26
4.2	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	20
4.3	output of the flow map. The brightness of integral curves encodes time	28
44	features in this Heated Cylinder dataset compared to directly fitting to vectors (middle). We show quantitative results for our reduction rate experiment, where we measure the	33
7.7	RMSE of flow map error across different integration durations.	34
4.5	We qualitatively compare our method – Neural Flow Map to Neural Vector across different reduction rates. We show (top) the FTLE (generated by integration particles seeded at $t = 1100$ and for a duration of 80 in grid-time, and (bottom) difference images between the approximated FTLE and the ground truth FTLE. We find that optimizing for flow map samples yields better performance across all the reduction rates as compared to optimizing	
4.6	for the vectors themselves. We highlight the differences in purple	35
	for a simple, explicit Euler scheme for optimization.	35

4.7	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 canabilities of the models to integration durations that were not trained	
4.8	on as a heatmap (right)	36
	across different 2D datasets (rows). We show (1) difference images between the approxi- mated FTLE and the ground truth FTLE, and (2) the FTLE (computed by integrating for a duration of 300, 500, 500 for the fluid simulation, four rotating centers, and double gyre	
	dataset respectively). We find that our flow map method yields improved performance across all baselines.	39
4.9	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	
4.10	(longer) durations	40
	the simulation) for the ScalarFlow dataset between our method, neural vector fitting, and TTHRESH (6). 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	
4.11	center our method is able to better retain details of high repulsive behavior in the flow 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 (6), and Shepard interpolation. In general, our method obtains improved performance – note that	41
	at times, we obtain an improved performance in flow map, despite having a higher error in	
4.12	vector field (c.f. Fig. 4.12)	41 42
5.1	Our method seeks to incorporate scale spaces into implicit neural representations, specifi- cally those that model field-based data. A central advantage of our method is its generality, in the support of varying types of fields, as well as scale spaces (Fig. 1a –time-varying vector field with Gaussian smoothing; Fig. 1b – scalar field with bilateral filtering). Our method can be used in concert with field derivatives directly accessible from the model, shown on the left as the norm of a derived acceleration field under varying scale. On the right, we show how our method can learn nonlinear data-dependent filters, leading to a reduction in noise within the pelvic region, whereas the structure of bones and organs are	
5.2	enhanced.	46
5.2	we indistrate the basic idea behind our method for building scale into lives. On the top, we notionally depict an INR – namely its positional encoding, followed by a fully-connected layer – used in part to fit the 512^3 Hazelnuts field (top-right). Links encode network weights, where link thickness encodes a weight with large magnitude, and red/blue color encodes positive/negative. Our approach (bottom) learns to modulate the weights in an existing INR, in this case decreasing their magnitude in proportion to a provided scale, in order to accurately model a given scale-space transformation, here showing one based on	
5.3	a bilateral filter	49
2.5	mize models ranging from SIREN, two variants of MFNs (Fourier, Gabor), and a local INR (SIREN-Grid), to fit a Gaussian scale-space of a 2D scalar field (left), all of whose parameters lead to a compression ratio of 30x. As demonstrated quantitatively (right), and qualitatively for SIREN (top) all models are able to adequately learn a scale space for this	
	field, as indicated by the high PSNR values.	51

5.4	This plot demonstrates the relationship between the frequencies' L2 norm and their asso- ciated weights for a Fourier-based MFN, trained for a medical image endowed with scale	54
5.5	spaces for a Gaussian filter, and bilateral filter, respectively	54 59
56	Comparison of our method with an alternative model that explicitly conditions on scale as	57
5.0	a coordinate. Our method fares much better in generalization emphasizing the importance	
	of how to represent scale within an INR. Dataset: MHD-B (256 ³)	61
5.7	We compare our method against the tensor-based Tucker decomposition, which permits	
	compression-domain linear filtering. In the linear case (Gaussian), our method is compet-	
	itive, but for data-dependent filtering (Bilateral) our method outperforms Tucker decom-	
	position, and without needing to resample to a grid. On the bottom, we show a visual	
	comparison for Miranda under Gaussian filtering at scale $s = 2$, and Hazelnuts under bi-	
	lateral filtering at scale $s = 4$.	62
5.8	We show the results of our method applied to a diverse range of fields, across different	
	types of INRs. Each INR is compressive by design, while still enabling access to a scale-	
	parameterized family of transformations. By representing a field as a coordinate-based	
	neural network, we can perform backpropagation to compute spatial derivatives, thus en-	
	abling compressive, scale-dependent features on the field, shown here as gradient magni-	
	field for MHD (top right)	63
59	For grid-based SIRENs, we study the effect of the coarse grid resolution on learning a	05
5.7	bilateral filter scale space on a 1024^3 CT scan of a flower (compression: $400x$) As shown	
	a good compromise between quality (top-left) and training/inference time at a scale $s = 7$	
	(right) can be found (16^3), where "GT" corresponds to the time required to bilateral filter	
	the original volume at this scale.	65
5.10	We study the effect of compression on learning a bilateral scalespace for two different	
	INRs, namely, SIREN and Gabor (MFN), for the Lung dataset. In general we find that	
	our method is insensitive to the level of compression, maintaining a comparable level	
	of quality across the learned scale space. On the right we show that, qualitatively, the	
5 1 1	different levels of compression yield similar results (scale $s = 3$)	65
5.11	Our approach can distill a scale-based INR into a scale independent INR to achieve higher	
	field (in range [0, 1]) and compare distillation between SIREN and Grid SIREN (8 ³ grid)	
	We observe both can roughly meet the tolerance (left), while we observe SIREN is far	
	more compressive than Grid-SIREN (right)	66
5.12	For our network distillation results on the Flower dataset, and a SIREN-based INR, we	00
	show, for each scale-distilled model, the number of neurons retained at each layer after	
	performing pruning. We find a strong tendency to remove neurons in deeper portions of	
	the network.	68
5.13	We show the results of learning persistence based scale-space for the vorticity field of a	
	fluid simulation dataset.	69
5.14	We show the results of persistence based filtering quantitatively through PSNR and Wasser-	
	stein distance between the original and the predicted field for different scales applied to	
	the pressure field of the isotropic Turbulence dataset. We snow qualitative results of the fields generated from the ground truth (top row) and INP predictions (better row) for a	
	subset of scales	60
		0)
6.1	Our approach is based on a criterion of self-consistency, where the flow map derivative	
	under some time span τ should match the flow map's instantaneous velocity at this loca-	
	tion. A flow map whose instantaneous velocity initially matches the vector field can give	
	us a linear approximation (left), violating self-consistency. By optimizing over a range of	
	time spans, our approach aims to adjust the flow map such that this criterion is satisfied	
	(right)	/4

6.2	Standard coordinate-based networks (a) serve as simple models for flow maps, but are difficult to control and optimize. In contrast, our proposed network design (b) permits a clear delineation between the instantaneous velocity of the flow map, and integration for nonzero time spans (τ), achieved via τ -scaled residual connections. This leads to a simplified derivative network (c) at $\tau = 0$, one that is straightforward to fit to the vector	
6.3	field, and subsequently, stabilize flow map optimization	74
	course of optimization. Initially (40 steps), the flow map provides us with a linear approximation, owing to its instantaneous velocity well-representing the vector field. Over optimization, the flow map becomes more accurate in its predictions for longer time spans	79
6.4	We show the quantitative evaluation of flow map approximation methods across different datasets, and across different time spans, beginning at start times for which flow features	0.1
6.5	We compare FTLE (top row) and integration error (bottom row) for two Fluid Simulation datasets (Re 16 and Re 101.6) across different baselines. The left column corresponds to particles integrated beginning at $t_0 = 0$ for duration $\tau = 7$, while the right column corre-	81
6.6	sponds to particles integrated starting at $t_0 = 2$ and $\tau = 7$	83
6.7	the Double Gyre dataset. Particles are integrated from $t_0 = 0$ for a time-span $\tau = 10$ We show the FTLE (top of each pair) and error maps (bottom of each pair) for the flow over cylinder dataset generated by integrating particles starting at $t_0 = 18$ for a time-span	83
6.8	$\tau = 1$ We show the FTLE (top of each pair) and error maps (bottom of each pair) for the Boussi-	84
69	nesq dataset generated by integrating particles starting at $t_0 = 11.3$ for a time-span $\tau = 0.5$. We compare our method's ability to compute streaklines against the streakline vector field	85
0.9	technique (143), which only necessitates integrating a derived vector field. Qualitatively and quantitatively we find that our method produces comparable results, where we show	06
6.10	We compare, both qualitatively (volume rendering of FTLE field) and quantitatively (flow map evaluation), our method with standard coordinate-based networks (120) as well as pathline interpolation techniques (72) for modeling the flow map in 3D unsteady flows. We find our method is quantitatively an improvement over other methods, and qualitatively	80
6.11	our method contains fewer visual artifacts	87
	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	
6.12	We compare NIFM to Euler and RK4 integration schemes, showing FTLE (top), flow map error (middle), and quantitative evaluation (bottom). We find NIFM performs consistently	88
6.13	well across step sizes as compared to Euler and RK4 which can become numerically unstable. We study the effect of different schemes for taking steps in flow map optimization for the Boussinesq flow: full corresponds to taking one step per grid unit in time (and thus the most costly), sqrt corresponds to a square root scaling in (grid unit) time, log is a logarithmic scaling, while single corresponds to taking just a single step (thus the cheapest	88
6 1 /	in computation). We find little difference between these schemes upon evaluation We further study different schemes for taking stors in flow man optimization, here for	90
0.14	Double Gyre.	90
6.15	We study the effect of model size on accuracy for the Boussinesq flow. We find a small drop in accuracy as we decrease the model size.	92
6.16	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.	92
6.17	For the Boussinesq flow we compare the effects of τ_{max} on the overall performance of the model, finding that the self-consistency criterion when trained with large τ_{max} gracefully	-
	degrades in performance, indicating the stability of our optimization technique	92

6.18	We show the performance of NIFM under large integration durations for the double gyre	
	dataset. We show the FTLE and the corresponding flow map errors for $\tau = 20$ and $\tau = 30$.	
	We observe the technique is able to perform reasonably well even for very large integration	
	durations	93
6.19	We show the performance of different techniques under large integration durations for the	
	double gyre dataset. We show the FTLE and the corresponding flow map errors for $\tau = 20$.	
	We observe that our model is able to outperform all the baselines and incurs the least error.	93
6.20	We evaluate different techniques under varying integration duration for the double gyre	
	dataset	94

CHAPTER 1

Introduction

Flow simulations have emerged as a crucial tool for understanding and modeling complex flow structures that impact various aspects of our daily lives. In recent years, numerous disciplines in engineering and applied science, such as aerodynamics, oceanography, chemistry, medicine, and meteorology, have embraced numerical flow simulations. This growing adoption has been made possible by advancements in high-performance computing (HPC) environments, which offer advanced computational hardware, robust networking capabilities, and ample memory and bandwidth. These advancements have established a dependable framework for precisely simulating and generating flow datasets. This enables researchers to thoroughly analyze and acquire a deeper understanding of the underlying physical phenomena. As a result, scientists are consistently conducting numerous simulations using various configurations to extract valuable insights. The outcome of these simulations is the creation of dense, high-resolution, time-varying flow datasets. These datasets encompass crucial quantitative details such as velocity, pressure, and density. However, it is important to recognize that these quantities, whether primitive or derived, are essentially numerical values linked to specific space-time positions. While certain measures like average magnitude across the entire simulated space-time domain can be computed, they alone fail to provide a comprehensive understanding of the flow's evolution or the inherent flow structures within the data. Relying solely on numerical analysis to interpret raw numbers and draw conclusions may result in flawed analyses. Therefore, a more holistic approach is required to extract meaningful insights from these simulations.

Humans are inherently visual beings. Our brains are adept at processing visual data swiftly and with less cognitive strain compared to raw numerical data. As a result, data presented in a graphic format is considerably easier for us to comprehend. Our innate abilities to discern variations in size, shape, color, quantity, and relative positioning of data points aids in the identification of patterns within the data, which would otherwise be significantly more challenging to differentiate. This drives the necessity for techniques that graphically represent data for analytical purposes. Flow visualization is a particular branch of scientific visualization that tackles issues related to the graphical depiction of flow data. Over the years, it has proven to be an incredibly effective tool for the exploration and analysis of flows (144). By enhancing our ability to understand complex flows, flow visualization has considerably expanded our knowledge base.

Flow visualization is an active area of research and numerous techniques of varying complexities have been developed over the years to help scientists make sense of the data. Even though the basic principle of flow visualization is to generate graphical representations from which insights about the flow field can be



Figure 1.1: Shown in the figure are graphical representations in the form of streaklines for two large-scale datasets (a) isotropic turbulence dataset - showcasing the chaotic nature of the dataset and (b) mantle dataset - illustrating the convection processes of the Earth's mantle.

obtained, it is the nature of the flow field, its representation, and the visualization task at hand that dictate the diversity of existing techniques. Take for instance two distinct tasks: first, visualizing the separation structures present within the flow data, and second, visualizing the integral curves produced by advecting massless particles along the flow over a given time span. These tasks, by their very nature, require divergent visualization techniques. The latter task can be simply visualized by plotting graphical lines, whereas this technique is unsuitable for the former task, necessitating the adoption of alternative methods for effective visualization. Moreover, the representation of flow data also plays a decisive role in determining the type of visualization required for effective flow analysis. Thus, the complexity and variety of techniques in flow visualization are driven by these multifaceted requirements.

Next, we delve deeper and describe two distinct types of data representations commonly used in flow visualization. Subsequently, we provide a high-level description of diverse visualization techniques, thereby establishing a comprehensive understanding of their respective roles and applications within the broader context of flow visualization.

1.1 Flow Specification

Flow is usually described as the variation of a physical quantity (e.g velocity vector) as a function of time over a fixed spatial domain. In fluid dynamics, two different types of specifications are used to describe the flow, namely, (1) Eulerian and (2) Lagrangian.

Eulerian Specification Within the Eulerian framework, changes in flow properties are described from the perspective of fixed spatial and temporal locations. This can be likened to an observer monitoring the

flow from a stationary position external to the system. In practical applications, data utilizing Eulerian specification is stored as velocity fields. Here, each point in the space corresponds to the velocity of the flow at a specific moment in time.

Lagrangian Specification In contrast, the Lagrangian framework portrays variations by tracing the flow's path through space, analogous to an observer moving synchronously with the flow, becoming part of the flow system. Practically, Lagrangian specifications are embodied in flow maps. A flow map effectively outlines the final position of massless particles that have been advected along the flow for a certain duration of time, having commenced from a starting position at a specific time.

The choice between Eulerian and Lagrangian specifications for visualization purposes largely depends on the specific context and the nature of the data. Both approaches have their own advantages and can be more suitable in certain scenarios. Suppose we are interested in monitoring weather patterns at a specific geographic location, then Eulerian specification is advantageous in this case because it offers a static viewpoint and allows us to observe how the weather (e.g., temperature, humidity, wind speed, etc.) changes at a specific location over time. In contrast, a Lagrangian perspective would follow a moving air particle, which would not be as useful for understanding weather changes at a fixed point. However, if we were interested in understanding the life cycle of a storm, then in that case, Lagrangian specification would be more appropriate as it would allow us to follow the storm from its inception, through its development and dissipation. Additionally, we can track the changes in the storm's properties (like size, intensity, speed) and its path. Overall, Lagrangian particle tracking provides a very intuitive visualization of flow transport behavior. On the other hand, Eulerian perspective is useful for visualizing broader flow pattern evolution. Using a mix of both techniques can provide the most physical insight into complex flows.

1.2 Flow Visualization Techniques

There is a myriad of flow visualization techniques being used by scientists for the visual exploration of flow data. Traditionally, these techniques are sub-divided into direct, texture-based, geometry-based and feature-based categories. However, some techniques are more commonly used than others. First, we have a glyph-based visualization that operates directly on the raw data (data produced by the numerical simulation such as velocity vectors). Glyphs (cones, cylinders, or arrows) are placed at a given spatial position pointing in the direction of the flow. Optionally glyphs can encode the magnitude of the vector via its size. Due to its simplicity, glyph-based visualization is widely used in 2D flow visualization, however, its usage is limited in 3D due to excessive occlusion.

Another common technique used for the purpose of finding flow structures and analyzing global flow behavior are integral curves. These are obtained by integrating massless particles in the flow field to obtain



Figure 1.2: In the figure we show the space-time illustration of various integral curves.

the trajectories of the moving particles. These trajectories reveal structural patterns and the flow behaviour over the time-span they were integrated for. Commonly, these trajectories come in three forms: streamlines, pathlines, and streaklines. A streamline is characterized as a curve that is uniformly tangential to the immediate velocity of the flow field. It represents a snapshot of the flow field at a specific moment in time, demonstrating the direction of the flow at every point. The pathline, on the other hand, delineates the trajectory traced by a massless particle as it moves along the flow from a specific starting point. Unlike the streamline that represents a particular moment, the pathline is a historical record of the particle's journey and is tangential to the velocity of the flow over a period of time. Lastly, the streakline represents the locus of particles that have traversed a given spatial point at any previous time. It visualizes the history of particles that have passed through a certain point in space, serving as a unique tool to study the past behavior of flow patterns. In Fig. 1.2 we illustrate the different between the different types of integral curves and in Fig. 1.3, we provide an example showcasing the structural pattern created by different integral curves for the flow over cylinder dataset. For steady flow, all the aforementioned integral curves are the same. However, for unsteady flows these are quite different and reveal different structures of the flow field. Moreover, all the curves are tangential to a special derived (n+2)-dimensional flow field (143; 142).

Computationally, integral curves are approximated by using (1) an interpolant - required to access the velocity vectors at arbitrary particle positions and (2) a numerical integration scheme. In practice, both interpolation and approximation through numerical integration are computationally expensive processes. Furthermore, the computational expense of both interpolation and numerical integration is further compounded when dealing with large data sets that exceed the available memory. In such cases, the data needs to be paged into the main memory from a secondary source, introducing additional overhead and constraints. As a result, interactive exploration becomes impractical, hindering the ability to interactively analyze and visualize



(a) Streamlines



(b) Pathlines



Figure 1.3: We show the different types of structural pattern formed by various integral curves for the flow over cylinder dataset.

integral curves.

Lastly, a widely used visualization technique that focuses on the structural analysis of the flow field from the Lagrangian perspective is the finite-time Lyapunov exponent (FTLE) (38). It essentially determines the exponential separation rates among neighboring particles from a dense set of pathlines covering the spatial domain. The ridges obtained from the FTLE allow us to extract the Lagrangian Coherent Structure (LCS) that described the dynamic behaviour of the flow field in terms of its structural evolution.

1.3 Research Challenges

While numerous visualization techniques have been proposed in the literature, flow visualization comes with it's own challenges and several factors need to be addressed first. We list these factors below:-

- 1. Data Reduction : The conventional workflow for data analysis typically involves conducting extensive numerical simulations on networked high-performance computing (HPC) environments. Subsequently, the obtained high-resolution data is saved onto storage disks and then transferred to a local workstation for further visual analysis. However, the process of transferring data from the HPC system to the local workstation often becomes a bottleneck due to limitations in storage space and network bandwidth. Consequently, there arises a crucial requirement for a compact data representation capable of preserving the essential information necessary for accurate post-hoc visual analysis while minimizing storage and bandwidth demands.
- 2. Computation Time : In the literature, substantial advancement in the design of novel and informative visualization techniques has been made. Among these techniques, notable ones include the finite-time Lyapunov exponents (FTLE) (38), and its resulting Lagrangian coherent structures (LCS) (39; 38), 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. 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. This necessitates the need for approaches that can improve the computation times of flow maps and thus, interactivity of the underlying visualization process.

To address the challenges described above we take advantage of the recent advancements in the area of deep learning. In particular, two types of deep learning models show promise for addressing the identified challenges:

(1) Convolutional Neural Networks (CNNs) (70) for superresolution of sampled grid representation. By leveraging the inherent spatial relationships and patterns in grid-sampled flow data, CNNs can provide high-resolution representations, enabling detailed analysis and visualization of the flow behavior. Utilizing CNNs for superresolution can alleviate the storage and bandwidth limitations, allowing for more efficient transfer and analysis of flow data.

(2) Implicit neural representation (INRs) (120; 130) for compact data representation and reduced computation in flow visualization. INRs provide a learned parameterization of the flow field, enabling efficient and compact representation of the essential information. By endowing INRs with scales further data reduction can be achieved and at the same time improve the utility of INRs for downstream visual analysis tasks. Additionally, INRs can effectively reduce the computational burden associated with the computation of the flow map over large time spans, enabling interactive exploration and analysis of flow features.

By harnessing the power of deep learning, specifically, CNNs for superresolution of grid sampled data and INRs for data reduction and reduced computation, flow visualization can benefit from improved efficiency, scalability, and interactivity.

1.4 Contributions

In the first part of this dissertation, we primarily focus on the *data reduction* aspect of flow visualization. For the purpose of data reduction when the data specification is in Eulerian frame of reference several techniques like compression (6) and super-resolution (36) have been explored in the literature. Traditionally, simple interpolation techniques were used for super-resolution, however, these techniques lack the ability to capture the global flow behavior since upsampling is based on local information only. In recent years, deep learning based techniques have received significant attention and have shown to outperform these more traditional techniques. Yet a limitation common to existing methods is that no consideration is made to how upsampled flow fields are used in practice. Specifically, as part of a scientist's visual analysis workflow, it is extremely common to integrate the flow field to obtain streamlines, in order to discover more global, structural flow features. On the other hand, a key advantage to optimization-based techniques for super-resolution, e.g. deep learning methods, is that we may optimize for what we ultimately visualize, e.g. streamlines. To this end, we developed an integration-aware approach to vector field super-resolution technique. In this work, we showed how to augment more traditional super-resolution objectives with integration-aware losses, and studied the impact of various factors like seeding technique, integration length on the optimization process. This work has been published in Sahoo and Berger (105).

While techniques like compression and super-resolution have been explored in the literature for data reduction, these techniques only consider the Eulerian specification for flow fields. In practice, Lagrangian

specifications are also used widely. Agranovsky et al. (1) showed that Lagrangian specification for flow fields are more efficient and compact as compared to Eulerian specifications when the temporal resolution of the flow data is low. Thus, developing techniques that can make use of the Lagrangian specification for *data reduction* is an attractive option. To this end, we developed a novel method for representing and recovering unsteady flow that enables substantial data reduction in time-varying flow fields. Our approach takes on a hybrid Eulerian-Lagrangian viewpoint. Our representation of unsteady flow is Eulerian, in that we leverage INRs (120) to model time-varying flow fields. 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. 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 other Eulerian, and Lagrangian, unsteady flow data reduction approaches. This work has been published in Sahoo, Lu, and Berger (107).

INRs have received significant attention in recent years and have shown promising results in learning compressive representations of volumetric fields (80). However, existing methods for designing and optimizing INRs are not yet suitable for what is typically required of visual analysis techniques, e.g. if we want to build a progressive representation of a field for level-of-detail analysis, then it is necessary to (1) sample the INR to a grid, and (2) then build the relevant progressive representation. For the integration of INRs within visualization systems, we argue that INRs should be endowed with operations and properties that users wish to access. To address this, we propose a novel approach that incorporates a sense of scale into INRs, enabling user-oriented operations and fulfilling visual analysis needs. Our technique modulates the parameters of INRs with scale-dependent transformations, allowing them to accurately model scale-space field representations. This facilitates compressive, scale-dependent feature analysis using implicit frequency domain filtering, including both linear and non-linear filters. Furthermore, we demonstrate the possibility of distilling a scale-based INR into a more compact, scale-independent representation, leading to simplified fields. This work is currently being prepared for submission to the IEEE Pacific Vis Conference, authored by Berger, Sahoo, and Lu.

Lastly, we shift our focus towards *interactivity* aspect of flow visualization. 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 this work, we seek neu-

ral 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. This work is under review submitted as part of TVCG journal. Preprint can be found in Sahoo and Berger (106).

CHAPTER 2

Background and Related Work

2.1 Flow Visualization Techniques

In the field of flow visualization, over the years, countless visualization techniques have been proposed that present the flow data in ways allowing for easier visual exploration and analysis. Generally, flow visualization techniques are classified into the following widely accepted categories (98) :-

2.1.1 Direct Visualization

Direct visualization methods offer a straightforward approach to flow visualization by displaying the raw data points without any transformations or interpretations. This category of techniques is based on a one-to-one mapping between the graphical representation and the data point, making it very intuitive and easy to understand. An example of a direct visualization technique is the arrow plot (97), which utilizes vectors to show the direction and magnitude of flow data. However, while the simplicity of direct visualization can be an advantage, it often results in problems such as occlusion and aliasing (30). Occlusion is an issue when overlapping data points hide some information from the viewer, making it difficult to accurately interpret the data. Aliasing, on the other hand, occurs when the sampling rate of the data is not high enough, resulting in the visualization showing inaccurate representations of the flow patterns.

2.1.2 Texture-Based Visualization

Texture-based visualization techniques are more complex as they involve the dense encoding of data through the convolution of a noisy texture across the flow field. They provide a continuous and comprehensive view of the flow field, making them ideal for observing global flow patterns and structures. A well-known technique in this category is the Line Integral Convolution (LIC) method (13). LIC uses convolution along streamlines with a white noise texture to create a blurred image that reveals directional flow patterns. Other variants, like Fast LIC (122) and Accelerated Unsteady Flow LIC (141), aim to address some of the limitations of the original LIC method, such as computational efficiency and applicability to unsteady flows.

2.1.3 Feature-Based Visualization

Feature-based visualization techniques aim to extract and emphasize the most significant features of the data. These methods involve advanced computations and data analysis to identify and represent the primary characteristics of the flow field. For example, the finite time Lyapunov exponent method (114) identifies

chaotic regions in the flow, which are areas of high sensitivity to initial conditions. These techniques often involve the use of topological concepts, which are mathematical tools used to describe the shape and structure of data. An extensive survey by Laramee et al. (69) provides a detailed overview of various topology-based visualization techniques.

2.1.4 Geometry-Based Visualization

In geometry-based visualization, the flow dynamics are captured through integral curves and surfaces, such as streaklines, streamlines, and pathlines. These techniques are powerful tools for representing time-dependent flow fields and visualizing the trajectory of particles in a flow. McLoughlin et al. (85) provides a comprehensive review of various integration-based visualization techniques. Although they can be computationally intensive, the detailed insights that these methods provide about the flow dynamics make them invaluable tools in many research and application contexts.

In conclusion, each category of flow visualization techniques offers unique approaches for representing and interpreting complex flow data. By choosing the appropriate technique based on the specific requirements and constraints of a given scenario, researchers and practitioners can effectively visualize and understand complex flow dynamics. In this thesis, we focus mainly on Geometry-Based Visualization techniques. We provide additional background and introduce some important concepts relating to Geometry-based visualization techniques below.

2.2 Flow Field and Particle Trajectory

A flow field describes the motion of a fluid in space and time. It can be classified as either steady or unsteady. In steady flow, the velocity vector at any point in space does not change over time. Mathematically, it can be represented by the equation:

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{v}(\mathbf{x}(t)) \tag{2.1}$$

where $\mathbf{x} \in \mathbb{R}^n$ represents the spatial position and $v : \mathbb{R}^n \to \mathbb{R}^n$ is the time-independent flow field.

On the other hand, an unsteady flow is associated with a flow field whose instantaneous velocity changes over time. And the path traced by a massless particle under the influence exerted by the time-varying flow field gives us the particle trajectory. It can be determined by solving the following ordinary differential equation (ODE):

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{v}(\mathbf{x}(t), t), \quad \mathbf{x}(t_0) = \mathbf{x}_0$$
(2.2)

Here, $v : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n$ represents the time-dependent flow field and \mathbf{x}_0 is the initial spatial position of the particle at time t_0 . The solution to this ODE provides the particle trajectory.

To simplify the analysis, the ODE in Equation 2.2 can be treated as an autonomous ODE by introducing time as a state variable. This results in the following equation:

$$\frac{d}{dt} \begin{pmatrix} \mathbf{x}(t) \\ t \end{pmatrix} = \begin{pmatrix} \mathbf{v}(\mathbf{x}(t), t) \\ 1 \end{pmatrix}, \quad \begin{pmatrix} \mathbf{x} \\ t \end{pmatrix} (0) = \begin{pmatrix} \mathbf{x}_0 \\ t_0 \end{pmatrix}, \tag{2.3}$$

This autonomous ODE allows us to study the particle trajectory as a function of time.

2.3 Flow Map and Integral Curves

2.3.1 Flow Map

The flow map describes the final position of a massless particle after being integrated for a specified time duration (τ). It is defined as:

$$\Phi(\mathbf{x}_0, t_0, \tau) : \mathbb{R}^n \times \mathbb{R} \times \mathbb{R} \to \mathbb{R}^n = \mathbf{x} 0 + \int_{t_0}^{t_0 + \tau} \mathbf{v}(\mathbf{x}(t), t) dt$$
(2.4)

The flow map takes the initial position \mathbf{x}_0 , starting time t_0 , and integration time τ as inputs and gives the final position of the particle.

2.3.2 Integral Curves

Integral Curves are the particle trajectories computed by integration of massless particles in a flow fields. These curves are widely used as a visualization tool because they captures the flow dynamics and provide valuable insights regarding the flow behaviour. The three different types of integral curves used as part of this thesis are describe below:-

Streamlines: Streamlines are integral curves that are tangent to the flow field everywhere i.e. the mapping between the particle trajectory and the flow field is bijective.

Pathlines: Pathlines represent the trajectories of particles starting at a specific time t_0 and integrated for a finite time duration. They are obtained by solving the autonomous ODE in Equation 2.3.

Streaklines: Streaklines are formed by tracing the flow maps of particles originating from the same spatial position but at different times. To compute a streakline at time *t*, flow maps of particles seeded at position \mathbf{x}_0 at times $t_s \in [t_0, t]$ are computed. The curve connecting these flow maps represents the streakline.

In summary, the flow field describes the motion of a fluid, and the particle trajectory represents the path followed by a particle in that flow field. The flow map gives the final position of a particle after a

specified integration time, and integral curves (streamlines, pathlines, streaklines) provide insights into the flow behavior and are valuable for visualization.

2.4 Numerical Integration

In practice, most of the times analytical solution of autonomous ODEs are not readily available and thus the solution has to be approximated using numerical integration schemes. In this section, we describe the two most widely used numerical integration scheme to compute the solution of the autonomous ODE described in Eq. 2.3 and Eq. **??**.

Explicit Euler Integration Given a particle initial position (\mathbf{x}_0, t_0) , integral curve via numerical integration is computed by iteratively advecting the particle under the flow field. In case of Euler Integration the following set of operations are performed recursively until the desired duration is reached.

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \varepsilon \mathbf{v}(\mathbf{x}_i, t_i) \tag{2.5}$$

$$t_{i+1} = t_i + \varepsilon \tag{2.6}$$

The euler integration scheme is simple, and fast. However, the simplicity and fast computation speed comes at the cost of numerical accuracy. The accuracy of the integration is heavily dependent on the step size ε . There's a trade-off between computation speed and accuracy when it comes to explicit Euler integration scheme. Specifically, sufficiently smaller step sizes gives accurate integral curves, however, are more expensive to compute. And vice versa, with larger step size the errors accumulate resulting in inaccurate integral curves.

Fourth order Runge-Kutta Integration The fourth order Runge-Kutta (RK4) integration scheme is a more stable integration scheme as compared to explicit euler integration scheme. This integration scheme is more forgiving towards the choice of step-size. The flow map computation using Runge-Kutta is done using the following set of operations:-

$$k_1 = \mathbf{v}(\mathbf{x}_i, t_i) \tag{2.7}$$

$$k_2 = \mathbf{v} \left(\mathbf{x}_i + \frac{\varepsilon}{2} k_1, t_i + \frac{\varepsilon}{2} \right)$$
(2.8)

$$k_3 = \mathbf{v}(\mathbf{x}_i + \frac{\varepsilon}{2}k_2, t_i + \frac{\varepsilon}{2}) \tag{2.9}$$

$$k_4 = \mathbf{v}(\mathbf{x}_i + \varepsilon k_1, t_i + \varepsilon) \tag{2.10}$$

$$\mathbf{x}_{i+1} = \frac{\varepsilon}{6} (k_1 + 2k_2 + 2k_3 + k_4) \tag{2.11}$$

$$t_{i+1} = t_i + \varepsilon \tag{2.12}$$

The computation cost per step in RK4 mainly involves the evaluations of the derivative function at different points within the time step. Since RK4 evaluates the derivative function four times, it is generally more computationally expensive compared to simpler methods like the explicit Euler scheme. However, this increased cost is accompanied by improved accuracy and stability, making it a popular choice for many practical applications. It's worth noting that the computational cost per step can vary depending on the complexity of the derivative function and the dimensionality of the problem. In some cases, the cost can be further reduced by exploiting the problem's structure or employing adaptive step-size control techniques to dynamically adjust the step size based on error estimates.

Continuing from the brief overview of some of the fundamental concepts in the field of flow visualization, in the subsequent section we delve into the relevant literature and research closely related to the contribution made as part of this thesis.

2.5 Related Work

Image-Based Super resolution Super resolution methods for generating high resolution images have received significant attention in the literature. Conventional methods like interpolation-based (25; 61) and reconstruction-based techniques (83; 123; 20; 150) are simple but fail to generate perceptually accurate images. Dong et. al (23) introduced SRCNN - a pre-upsampling framework where the low-resolution(LR) images are upsampled first and then finetunes using CNN models to generate the high-resolution (HR) image. Similar to SRCNN many other techniques were introduced varying the network architecture and learning strategies (63; 125; 126; 117). To overcome the computational expense of a pre-upsampling networks, (24; 116) introduced post-upsampling networks where the LR input image is passed through a Deep CNN network with upsampling layers to generate HR images. Following the success of this framework, most of the recent work use this framework (75; 125; 136; 44; 153; 67). Regarding network architectures, ResNet (47) and DenseNet (53) has been widely used (71; 125; 75; 153). Lai et.al (67) introduced a Laplacian pyramid network to learn multi-scale super resolution. In practice, 11 loss between image pixels has shown to be a better choice as compared to 12 loss (154; 75; 3). However, pixel based loss are known to produce oversmooth textures. Many other loss functions like content loss (59), texture loss (108) and adversarial loss (71) has been explored to generate more photo-realistic image.

Super resolution methods for data reduction. A common form of data reduction for flow fields is super resolution, where the objective is to upsample a low-resolution flow field to a high-resolution flow field. Guo et. al (36) showed promising results for 2x and 4x up-sampling of flow field data using a deep learning based approach where they optimize for the magnitude and angle of the target vectors. The work by Guo et. al was only limited to the spatial domain of a dataset and did not account for the spatio-temporal coherence present in

unsteady flow fields. Han et al. (42) proposed SSR-TVD a method for flow field super-resolution that takes the spatio-temporal coherence present in the data into consideration while performing super-resolution and does not treat space and time domain independently. Other deep learning-based super resolution schemes (23; 24) based on convolutional neural networks (CNN) have been explored, where the assumption is that patterns in downsampled data can be exploited to appropriately resolve details in the upsampled data. Jakob et al. (56) used a CNN based neural network to perform flow map super resolution. In all such methods, there exists a reliance on training data – paired low-resolution and high-resolution flow field, or flow map, examples. As such, at inference time, if there exists a domain mismatch between train and test data then super resolution methods are less effective. Closely related to superresolution are works that aim to reconstruct the flow field from a sparse set of scattered data. Prior works (113; 28; 152) achieve flow field reconstruction by directly optimizing for the velocity vectors. Han et al. (41) and Gu et al. (32) reconstruct flow fields from a set of representative streamlines. The work done by Han et al. (43) has similar goals to ours; however their approach uses a neural network that predicts flow maps directly.

Compression-based data reduction. Given that 3D unsteady flow is often in an Eulerian representation, e.g. a time-varying flow field, a common method for data reduction is lossy compression. Although volumetric compression of scalar fields has a long history within scientific visualization (90; 37), with numerous approaches specifically targeting time-varying data (112; 82; 57), 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 (68) or more adaptive fitting (22). 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 (124; 4). TTHRESH (6) 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 (120; 130) to time-varying volumetric scalar fields (81) (neurcomp), with compression achieved by controlling the network size, and weight quantization. In an follow up work, Weiss et al. (145) improved neurcomp by taking advantage of the GPU tensor cores.

Implicit Neural Representation (INR) The development of neural representations of field-based data has grown at a rapid pace in the past 2 years, please see the following surveys (132; 147) for a comprehensive overview on the breadth of applications. The field has burgeoned, in large part, due to the development of neural radiance fields (NeRF) (87), or reconstruction of radiance fields from a collection of images. Their success has led to rapid development within so-called implicit neural representations, a class of coordinate-based neural networks for representing fields. Numerous methods have been developed that study how to

encode position as input to a neural network (48; 120; 130), and different network architecture designs and optimization schemes (29; 54). There has also been significant effort on methods that can handle large data sizes, e.g., high-resolution geometric shapes (84; 127) and images (84) that are scalable to train and permit reasonably fast inference. More broadly, localized representations (18; 86) have also been investigated to handle larger data sizes.

Scale-space and Multiresolution Representations The use of scale is predominant within visualization. For the purposes of preventing aliasing, scale-space representations are quite common to use in conjunction with subsampling, as seen in many visualization approaches that rely on multiresolution representations (51; 52; 118; 138). Multiresolution methods have further been studied within tensor-based modeling of volumetric fields (124). Multi-scale methods have further been employed for vector fields (91) and tensor fields (62), in order to extract features (e.g. coherent structures, expansion/contraction of flows) of multiple scales within complex vector/tensor-valued fields. More traditional scale-space representations, e.g. those based on successive applications of low-pass filters such as Gaussian filters, have been used for the design of transfer functions in volume rendering (19), as well as feature extraction within scalar fields for medical images (58), and for vector field feature extraction (139; 140). However, other types of nonlinear, datadependent filtering approaches have shown beneficial for visual analysis, e.g. bilateral filtering has been used in medical imaging (40), viewpoint selection (131), and feature-preserving temporal smoothing (134). Topological simplification (27) is another natural way to express scale as the thresholding of critical point pairings within a persistence diagram (33; 133). One of our proposed works SCALE-INR (Chapter 5) aims to support all such notions of scale in augmenting INRs, without necessarily tying a single scale to the compressive representation, as, for instance, pursued in Soler et al. (121) for topology-controlled compression.

Multiscale INR and INR Filtering Approaches that seek multiscale implicit neural representation are most relevant to SCALE-INR. The recent work of BACON (77) demonstrates how one can adapt a multiplicative filter network (29) to be band-limited by design. Unlike SCALE-INR, BACON does not require an explicit scale-space representation on which to optimize, but in turn, can only handle one kind of scale: a box filter applied in the frequency domain. The approach of mip-NeRF (8) proposes an integrated positional encoding, thus enabling a multiresolution representation akin to classic mipmaps in computer graphics. Similar to SCALE-INR, this method can learn fields of varying scales, but is limited to antialiasing. SCALE-INR considers a broader notion of scale, made possible by shifting how scale is incorporated within the network. In particular, SCALE-INR is influenced by recent methods that characterize the space of functions represented by INRs, both for precise characterizations of specialized models (29), and more broadly, multi-layer perceptrons (MLPs) that utilize positional encodings (151). Recently, Xu et al. (148) proposed a new approach called INSP-Net that allows for direct modification of INRs without explicit decoding. They achieve

this by using differential operators on the INRs to perform signal processing. Similar to INSP-Net, Nsampi et al. (93) proposed a method to convolve neural fields using piecewise polynomial kernel. Similar to SCALE-INR both the approaches operate on the neural fields directly, however, both approaches are limited to simple Gaussian filters while SCALE-INR can be reinforced with non-linear filters as well.

Lagrangian Particle Interpolation. The Lagrangian representation of unsteady flow fields stores data in the form of trajectories of massless particles. Agranovsky et al. (1) showed that, for exploratory analysis, insitu 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 (16; 2; 12; Sane et al.; 99). Bujack and Joy (12) 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 (15; 55; 109). 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. (72) they represent 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. (38?) 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. (31) proposed an incremental flow map approximation technique for improving the computation time of FTLE. Sadlo et al. (103) 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. (60) constructed a localized FTLE and additionally, a faster way to compute it by reusing the separation values from previous time steps. Lipinski et al. (78) proposed a ridge tracking algorithm that approximates ridges in the FTLE for each time step and then approximates the ridge location in subsequent times. Brunton et al. (11) proposed a fast FTLE computation technique taking advantage of flow map composition for longer flow map approximations. Hlawatsch et al. (50) 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. (104) 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.

Neural Differential Equations Neural ordinary differential equation (Neural ODE), a technique to solve initial-value ODE problems proposed by Chen et al. (17) has been extended (79; 92) and applied to various different research domains (45; 96). 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. Biloš et al. (10) proposed an alternative technique to neural ode, wherein they approximate the solution directly in a single step instead of integrating within a latent representation space (17; 102). Other methods have explored gradient-based learning, e.g. modeling shapes with gradient fields (14), and accelerating the volume-rendering integral through learning antiderivative networks (76). Li et al. (73) proposed a self-consistent approach for solving partial differential equations (PDEs). It is worth noting that their work closely aligns with one of our methodologies presented in this thesis (106) as concurrent work. However, the key distinguishing factor of our approach lies in the proposed novel network design.

CHAPTER 3

Integration-Aware Vector Field Super-Resolution

In this chapter, we delve into the problem of super-resolution as a means of reducing data for the Eulerian flow field representations. While previous research has explored super-resolution techniques for vector fields, there is a significant limitation shared by existing approaches: they overlook how vector fields are practically employed once an upsampled vector field is obtained. Streamlines, or integral curves of the vector field, play a crucial role in flow visualization, and their visual analysis is a cornerstone of this field. To this end, we propose an integration-aware super-resolution approach specifically designed for 3D vector fields, which incorporates streamlines into the optimization process.

Our approach goes beyond simply enhancing the resolution of vector fields; it takes into account the practical aspects of streamline analysis. By considering streamlines as an integral part of the optimization process, we ensure that the resulting upsampled vector field aligns with the requirements of flow visualization. We address important factors related to streamline integration, such as seeding and streamline length, and investigate how these factors impact the upsampled vector field. Additionally, to showcase the effectiveness of our proposed approach, we conduct a comprehensive evaluation, both quantitatively and qualitatively, on various flow field datasets to assess the performance of our model and highlight its advantages over existing techniques and emphasize its relevance in the context of flow visualization.

3.1 Approach

In this work, our goal is to estimate a high-resolution vector field, denoted $\mathbf{V}_{\mathbf{h}} \in \mathbb{R}^{w \times h \times d \times 3}$ of spatial resolution $(w \times h \times d)$, given its corresponding low-resolution counterpart, denoted $\mathbf{V}_{\mathbf{l}} \in \mathbb{R}^{w' \times h' \times d' \times 3}$. We assume



Figure 3.1: Comparison of streamline results between our technique (b), ground truth (a) and the baseline (c) for tornado dataset, differences highlighted in yellow.

a subsampling factor r, a positive integer such that $w = r \cdot w'$, $h = r \cdot h'$, and $d = r \cdot d'$. Our approach to superresolution is to learn a mapping that we denote as $f : \mathbb{R}^{w' \times h' \times d' \times 3} \to \mathbb{R}^{w \times h \times d \times 3}$, where f is parameterized as a volumetric convolutional neural network, following prior work (36). We depart from Guo et al. in what we optimize: we would like the mapping f to upsample vector fields in such a manner that integral curves of $\mathbf{V}_{\mathbf{h}}$ are preserved. Namely, integrating $f(\mathbf{V}_{\mathbf{l}})$ produces streamlines that are as close as possible to streamlines of $\mathbf{V}_{\mathbf{h}}$.

3.2 Integration-Aware Upsampling Loss

Loss functions used in super-resolution tend to focus on content given in the high-resolution target, e.g. pixels in a high-resolution image, or in our case, vectors in a high-resolution vector field (36). This can be expressed as follows:

$$\mathscr{L}_{M} = \frac{1}{|\mathscr{P}|} \sum_{\mathbf{p} \in \mathscr{P}} \|f(\mathbf{V}_{\mathbf{l}}))[\mathbf{p}] - \mathbf{V}_{\mathbf{h}}[\mathbf{p}]\|_{2}, \qquad (3.1)$$

where \mathscr{P} indexes over vertices of a $(w \times h \times d)$ grid.

To ensure that our network can preserve streamlines, we introduce a loss function that is based on integral curves of, both, $\mathbf{V_h}$ and upsampled vector field $f(\mathbf{V_l})$, please see Fig. 3.2. Specifically, we denote $S = \{s_1, s_2, ..., s_n\}$ as a set of integral curves derived from $\mathbf{V_h}$, where $s_i = (\mathbf{p}_{i,1}, \mathbf{p}_{i,2}, ..., \mathbf{p}_{i,m})$ is a set of points on curve s_i . Given the upsampled vector field $f(\mathbf{V_l})$, we also form integral curves, taking the seed points from ground truth for integration. Specifically for streamline s_i , we integrate $f(\mathbf{V_l})$ starting at position $\mathbf{p}_{i,1}$ to obtain streamline $s'_i = (\mathbf{p}'_{i,1}, \mathbf{p}'_{i,2}, ..., \mathbf{p}'_{i,m})$, where $\mathbf{p}'_{i,1} = \mathbf{p}_{i,1}$. Our loss function is designed to ensure that the two curves remain close at *all* integrated positions:

$$\mathscr{L}_{S} = \frac{1}{n \times m} \sum_{i=1}^{n} \sum_{j=1}^{m} \|\mathbf{p}_{i,j} - \mathbf{p}'_{i,j}\|_{2}.$$
(3.2)

In practice, we combine the two loss terms, ensuring a balance between vector content (\mathscr{L}_M) and flow structure (\mathscr{L}_S):

$$\mathscr{L}_T = \lambda \mathscr{L}_S + (1 - \lambda) \mathscr{L}_M \tag{3.3}$$

where λ is a hyper-parameter which determines the relative importance of integration-based loss – a high λ places large importance on integration. We optimize the loss via stochastic gradient descent, which requires backpropagating over the integration method of choice. However, integration schemes like Euler integration and Runge-Kutta, can be expressed as a differentiable function with respect to the vector field, assuming a differentiable form of interpolation for accessing vectors at arbitrary positions. In practice, we use trilinear interpolation, thus we may optimize the loss function \mathcal{L}_T end-to-end.

datasets	dimensions $(x \times y \times z)$	timesteps (t)
tornado	$128 \times 128 \times 128$	50
square cylinder	$192 \times 64 \times 48$	100
tangaroa	$300 \times 180 \times 120$	200

Table 3.1: The dimensions and number of timesteps of each dataset.

What remains is a way to form the streamline set \mathscr{P} . We would like to ensure the streamlines are representative of predominant flow features. To this end, we consider **seeding** and **integration length**, studied further in Sec. 3.6.

Seeding: The starting positions from which to integrate are important for ensuring flow features are preserved (110). To capture flow features, we use the entropy-based seeding technique of Xu et al. (149). We normalize the resulting entropy scalar field and treat it as a probability distribution from which to sample positions. In order to not starve low-entropy regions of the flow field, we modify the distribution to interpolate between a uniform distribution, one that is a function of the entropy field:

$$s(x) = x \frac{e^x + e^{-x}}{e^{-x} - \alpha e^x},$$
(3.4)

where $\alpha \in [-1,0]$ interpolates between distributions.

Integration Length: The length of the streamlines further has an impact on flow features. Specifically, we use Euler Integration with sufficiently small step size and consequently, identify streamline length with the number of steps *m* taken during integration. The streamline length with which we train can have an impact on the network's ability to generalize, e.g. by training on small-length streamlines, will the network produce vector fields that faithfully reflect long streamlines? Similarly, training on long streamlines may sacrifice the ability to preserve small streamlines.

3.3 Implementation Details

Our network architecture for f closely follows Guo et al. (36), the only exception that we replace their *voxel shuffle* layers with nearest-neighbor upsampling for simplicity. Through experimentation, we found training the model using Eq. 3.3 from scratch posed challenges for optimization. Hence, we first train the model using the content loss (Eq. 3.1) for 25k iterations, and then fine-tune the model using the total loss (Eq. 3.3) for another 10k iterations. We found Euler Integration with sufficiently small step size and 4^{th} order Runge-Kutta integration give similar results. However, the former trains significantly faster, making it our choice of integration scheme for all the experiments. We use the Adam optimizer (64), with a starting learning rate



Figure 3.2: Overview of our approach. The network takes in the low-resolution vector field as input and outputs a super-resolution vector field. A content loss, alongside a streamline-based loss, between super-resolution and ground truth vector fields are used to optimize the network parameters. The use of 2d vector fields in the figure is for illustrative purpose only.

of 10^{-4} , we reduce it by a factor of 0.8 every 1000 iterations until the model fails to improve on withheld validation data. While training, for all the experiments we used a subsampling factor r = 4, batch size of 1 and 2,000 streamlines within a batch to form the loss function (\mathscr{L}_S).

3.4 Dataset and Training Details

All our experiments were carried on the datasets listed in Table 4.2, where x, y, z represents the spatial dimensions and t represents the number of timesteps in the dataset. To assess generalization of IA-VFS, we include every 4^{th} timestep of a given dataset in the training set and randomly select $t = (\hat{t} \mod 10)$ timesteps for validation, where \hat{t} represents the timesteps not being used for training. All the remaining timesteps are then used for testing purposes. All the experiments were carried out on NVIDIA TESLA V100 GPU.

Baseline We use the following 2 baselines to compare with our technique. (1) **Trilinear Interpolation** (**TI**): We use trilinear interpolation to upsample the low-resolution vector field to high-resolution vector field. (2) **Content Loss (CL)**: Using the same network architecture described in Sec. 3.3, we optimize only for the content loss in Eq. 3.1. Note, this represents Guo et al. (36), without using an angle-based loss, which we experimentally found to produce similar streamline results.

Evaluation Metric We use two different evaluation metrics to quantitatively evaluate IA-VFS. We use



Figure 3.3: Comparision of the differences in streamlines of vector fields generated by different models with respect to the ground truth highlighted in yellow. Top to bottom: square cylinder, tangaroa.

PSNR to evaluate the quality of the super-resolution vector field. PSNR is defined as follows:

$$PSNR(\mathbf{V_h}, f(\mathbf{V_l})) = 20\log_{10}R - 10\log_{10}MSE(\mathbf{V_h}, f(\mathbf{V_l})),$$
(3.5)

R represents the difference between the minimum and maximum value of the vector fields across all the timesteps for a given dataset and $MSE(\mathbf{V_h}, f(\mathbf{V_l}))$ represents the mean square error between the vector fields $\mathbf{V_h}$ and $f(\mathbf{V_l})$.

Since errors accumulate quite easily when calculating streamlines, the position of the last point on a given streamline of $f(\mathbf{V_l})$ can indicate how much it deviated from the last point position of streamline of $\mathbf{V_h}$. To this end, we define ALP (Average last position loss) to evaluate the quality of streamlines as follows:

$$ALP = \frac{1}{n} \sum_{i=1}^{n} \|\mathbf{p}_{i,m} - \mathbf{p}'_{i,m}\|_{2}$$
(3.6)

where *n* represents the number of streamlines, $\mathbf{p}_{i,m}$ represents the last point's position of i^{th} streamline of $\mathbf{V}_{\mathbf{h}}$ and $\mathbf{p}'_{i,m}$ represents the last point's position of i^{th} streamline of the $f(\mathbf{V}_{\mathbf{l}})$.

3.5 Quantitative and Qualitative Results

In Table 3.2, we summarize the quantitative evaluation of IA-VFS against TI and CL by averaging the PSNR (Eq. 3.5) and ALP (Eq. 3.6) values across all the test timesteps. We observe that IA-VFS outperforms TI and CL in ALP, indicating that our method more faithfully preserves streamlines. In case of tornado dataset, CL has the highest PSNR but it comes at the cost of lower ALP - this can be seen in Figure **??** where, IA-VFS (b) produces more faithful streamlines as compared to CL (c) which fails to capture the helix like pattern at the

Dataset	Method	PSNR	ALP
	TI	47.33	0.233
Tornado	CL	51.69	0.092
	IA-VFS	50.84	0.058
	TI	32.53	0.791
Square cylinder	CL	48.85	0.268
	IA-VFS	48.86	0.245
	TI	49.68	1.222
Tangaroa	CL	51.96	1.025
	IA-VFS	52.21	0.914

 Table 3.2: Average last position loss (Eq. 3.6) and PSNR for all the datasets.

Table 3.3: Average PSNR and ALP values for various α values for tornado dataset.

α	PSNR	ALP
-1	50.93	0.0372
-0.01	50.88	0.0333
-0.001	50.78	0.0356
-0.0001	50.73	0.0378
0	51.47	0.0513

Table 3.4: Average PSNR and ALP values for different λ values for tornado dataset.

λ	PSNR	ALP
0	51.69	0.0521
0.001	52.04	0.0465
0.1	51.76	0.0379
0.3	51.22	0.0359
0.5	50.88	0.0333
0.7	50.65	0.0356
1	50.33	0.0374

eye of the tornado. In Figure 3.3 we can see the streamline errors made by CL and TI in all the datasets. In square cylinder dataset we can see that IA-VFS captures the highlighted streamline whereas CL and TI fails to do so. We can also observe that the spiral flow is more accurate in (b) as compared to (c) and (d).

3.6 Hyperparameter Study

In this section, we analyze how various streamline hyperparameters affect the training process and justify our choices. We experimented with the following hyperparameter settings: the choice of λ (c.f. Eq. 3.3), streamline seeding, and number of integration steps.

Study of α parameter From Eq 3.4, we may bias seeds towards high entropy or uniformly-distributed positions via the parameter α . Here we study the influence of α , where $\alpha = -1$ gives us the normalized entropy scalar field back, and increasing α increases the chances of high entropy regions to be selected as seeds.

[5em]TrainEval	150	200	250	300	350	400
150	0.0262	0.0357	0.0454	0.0552	0.0657	0.0764
250	0.0260	0.0347	0.0438	0.0531	0.0630	0.0732
300	0.0262	0.0343	0.0427	0.0514	0.0605	0.0699
350	0.0261	0.0337	0.0419	0.0501	0.0588	0.0678
400	0.0264	0.0342	0.0424	0.0508	0.0596	0.0686

Table 3.5: Average last position loss of streamlines (Eq. 3.6) for models trained and evaluated on different streamline lengths.

In Table 3.3, we observe that there is a trade-off between PSNR and ALP for the tornado dataset based on the value of α . Heavily biasing towards the high entropy regions ($\alpha = [-0.0001, 0]$) leads to high ALP values. Since the network receives few important streamlines it fails to capture them accurately during evaluation. Meanwhile, a more spread out selection of seed points in and around high entropy regions with $\alpha = -0.01$ gives us the best ALP value and with acceptable PSNR.

Study of λ hyperparameter We can see in Equation 3.3 that λ controls the weight on the content loss and streamline loss. From Table 3.4 we observe that as we increase the value of λ we see a decrease in both PSNR and ALP. We found that a $\lambda = 0.3$ provides a good balance between PSNR and ALP.

Study of streamline length Streamline length determines the maximum number of steps to be taken while integrating the streamline. From Table 3.5 we can see that models trained on longer streamlines e.g. 300, 350 and 400 outperform models trained on smaller streamlines e.g. 150, 250 in terms of average ALP values. We found that model trained on streamline length of 350 provides good generalization when evaluated across different streamline lengths.

3.7 Conclusion and Future Work

In this work, we proposed an integration-aware super resolution technique for 3d vector fields. We think our approach is an important steps towards incorporating visualization aspects of vector fields in the optimization process. We show the benefits of using our technique and how various factors of vector visualization via streamlines affects the training process. There are several directions we would like to explore for our future work. In this work we experimented with a downsampling factor of 4, and we intend to try our framework on larger scaling factor. We have thus far, only considered spatial super-resolution of vector fields, and we intend to take into account the temporal coherence of unsteady vector fields.
CHAPTER 4

Neural Flow Map Reconstruction

In this chapter, we shift our focus towards Lagrangian specification of data and address the problem of data reduction. 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.

4.1 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.



Figure 4.1: 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.

4.1.1 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:

$$\mathbf{v}: \mathbb{R}^d \times \mathbb{R}_{>0} \to \mathbb{R}^d, \tag{4.1}$$

and we assume $d \in \{2,3\}$. In practice, we are given a *sampling* of v, 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:

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{v}(\mathbf{x}(t), t), \quad \mathbf{x}(t_0) = \mathbf{p},$$
(4.2)

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:

$$\Phi_{t_0}^{t_0+\delta}(\mathbf{p}) = \mathbf{x}(t_0) + \int_{t_0}^{t_0+\delta} \mathbf{v}(\mathbf{x}(t), t) dt.$$
(4.3)

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 (?) and for the visual analysis of attracting and repulsive behaviors (100). 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 **p**, (2) the starting time of advection *t*, and



Figure 4.2: 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.

(3) the result of applying the flow map $\Phi_t^{I+\delta}(\mathbf{p})$. 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:

$$\mathscr{T} = \left\{ \left(\mathbf{p}_1, t_1, \Phi_{t_1}^{t_1 + \delta}(\mathbf{p}_1) \right), \dots, \left(\mathbf{p}_n, t_n, \Phi_{t_n}^{t_n + \delta}(\mathbf{p}_n) \right) \right\},$$
(4.4)

namely, we assume *n* total flow map samples from Φ . As an example, in Fig. 5.2(top) this corresponds to the start and end points of integral curves.

4.1.2 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 v.
- 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 (120; 130), in particular those that are time-varying (81; 146). 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 $f_{\theta} : \mathbb{R}^d \times \mathbb{R}_{\geq 0} \to \mathbb{R}^d$. The function f_{θ} is a multi-layer perceptron with sinusoidal activation functions; we defer architecture details to Sec. 4.2.5. Ideally, we would like f_{θ} to be as close as possible to v.

The manner in which we optimize for f_{θ} , however, takes on a Lagrangian frame of reference, please see Fig. 5.2 for an overview. Specifically, for a given spatial position **p**, time t_0 and duration δ , we define our *neural flow map* as follows:

$$\tilde{\Phi}_{t_0}^{t_0+\delta}(\mathbf{p}) = \tilde{\mathbf{x}}(t_0) + \int_{t_0}^{t_0+\delta} f_{\theta}(\tilde{\mathbf{x}}(t), t) dt, \qquad (4.5)$$

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

$$\frac{d\tilde{\mathbf{x}}(t)}{dt} = f_{\theta}(\tilde{\mathbf{x}}(t), t), \quad \tilde{\mathbf{x}}(t_0) = \mathbf{p}.$$
(4.6)

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:

$$\frac{1}{|\mathscr{T}|} \sum_{(\mathbf{p},t,\mathbf{q})\in\mathscr{T}} \|\tilde{\Phi}_t^{t+\delta}(\mathbf{p}) - \mathbf{q}\|_2^2, \tag{4.7}$$

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

- 1: **Input**: spatial position **p** , time t_0 , duration δ , integration step size ε
- 2: Form a time-dependent particle $\tilde{\mathbf{x}}(t)$ under the neural vector field through solving Eq. 4.6 under explicit Euler integration
- 3: Initialize position gradient $\mathbf{a} = \frac{dL}{d\mathbf{\tilde{x}}(t_0+\delta)}$
- 4: Initialize weight gradient $\frac{dL}{d\theta} = 0$ 5: for $t = (t_0 + \delta \varepsilon)$ to t_0 in increments of ε do
- Integrate weight gradient $\frac{dL}{d\theta} = \frac{dL}{d\theta} \varepsilon \mathbf{a}^T \frac{\partial f_{\theta}(\tilde{\mathbf{x}}(t),t)}{\partial \theta}$ Integrate position gradient $\mathbf{a} = \mathbf{a} \varepsilon \mathbf{a}^T \frac{\partial f_{\theta}(\tilde{\mathbf{x}}(t),t)}{\partial \tilde{\mathbf{x}}(t)}$ 6:
- 7:
- 8: end for
- 9: **Return** $\frac{dL}{d\theta}$

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. 4.5, 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:

$$\tilde{\mathbf{x}}(t+\varepsilon) \approx \tilde{\mathbf{x}}(t) + \varepsilon \cdot f_{\theta}(t),$$
(4.8)

for an appropriately-defined step size ε .

4.1.3 Efficient Backpropagation

An immediate computational problem arises from naively optimizing Eq. 4.7 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. 4.8). 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. (17). In our setting, we are primarily concerned with efficiently computing the gradient over all examples from Eq. 4.7, which we express as:

$$\frac{1}{|\mathscr{T}|} \sum_{(\mathbf{p},t,\mathbf{q})\in\mathscr{T}} \nabla_{\theta} L(\tilde{\mathbf{x}}(t+\delta),\mathbf{q}),$$
(4.9)

where L is the loss for a single flow map sample as in Eq. 4.7, 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 $\varepsilon > 0$:

$$\mathbf{a}(t) = \frac{dL}{d\mathbf{\tilde{x}}(t)} = \frac{dL}{d\mathbf{\tilde{x}}(t+\varepsilon)} \cdot \frac{d\mathbf{\tilde{x}}(t+\varepsilon)}{d\mathbf{\tilde{x}}(t)},$$
(4.10)

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 (17) that this condition, as well as Eq. 4.6 leads to the following *backward* ODE, starting from the end of integration:

$$\frac{d\mathbf{a}(t)}{dt} = -\mathbf{a}(t)^T \frac{\partial f_{\theta}(\tilde{\mathbf{x}}(t), t)}{\partial \tilde{\mathbf{x}}(t)}, \qquad (4.11)$$

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:

$$\frac{d\theta}{dt} = 0, \quad \frac{dt}{dt} = 1, \tag{4.12}$$

and using the same reasoning above in Eq. 4.11, we have a second ODE to find our primary quantity of interest, the gradient of the loss:

$$\frac{dL}{d\theta(t)} = -\mathbf{a}(t)^T \frac{\partial f_{\theta}(\tilde{\mathbf{x}}(t), t)}{\partial \theta}.$$
(4.13)

An autonomous ODE is an ordinary differential equation that does not explicitly depend on the independent variable, in this case, time *t*. To this end, we introduce an autonomous ODE by prescribing differential equations on θ and *t*:

$$\frac{d\theta}{dt} = 0, \quad \frac{dt}{dt} = 1, \tag{4.14}$$

and using the same reasoning above in Eq. 4.11, we have a second ODE to find our primary quantity of interest, the gradient of the loss:

$$\frac{dL}{d\theta(t)} = -\mathbf{a}(t)^T \frac{\partial f_{\theta}(\tilde{\mathbf{x}}(t), t)}{\partial \theta}.$$
(4.15)

The equation, $\frac{d\theta}{dt} = 0$, signifies that the weight parameter θ remains constant throughout the integration. This enables the computation of gradients with respect to this fixed-weight configuration, which can later be used for optimization. We are not actively changing θ while performing backpropagation; instead, we're investigating how changes to θ would affect the loss. The second equation, $\frac{dt}{dt} = 1$, indicates that the 'rate of change' of time is constant. By adding these equations, we create an augmented ODE system that allows us to capture the dynamics of all necessary quantities $(a(t), \theta, t)$ within a unified framework. We can then use an ODE solver to propagate this system in time and compute the quantities of interest (specifically, the gradients of the loss) in a computationally efficient manner.

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.

4.2 Analysis

4.2.1 Fitting to Flow Map Samples, Fitting to Vectors

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.

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 (120; 130; 81) 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. 4.3 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. 4.3 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



Figure 4.3: 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).

nevertheless, we can see, qualitatively, that flow map-based optimization tends to better capture the FTLE over fitting to vectors.

We perform an additional experiment wherein we vary the reduction rate for the heated cylinder dataset (95; 34), in order to study the differences between optimizing for flow map samples and fitting directly to vectors. For the experiment, we choose reduction rates of 50, 100, 200, 300, and 400, relative to the total size of the time-varying vector field. We set the network capacity to be 0.5 times the size of the total number of flow map samples. We generate flow map samples by integrating particle using Euler integration scheme with a step size of 0.1, measured in grid units, for a duration of 10. We train the models using Adam optimizer with a starting learning rate of 10^{-4} , decaying every 60 epochs by a factor of 0.2. We train the models for a total of 150 epochs.



Figure 4.4: We show quantitative results for our reduction rate experiment, where we measure the RMSE of flow map error across different integration durations.

In Figure 4.5 the top row shows the FTLE generated using flow map samples seeded at timestep 1100, and integrated for a duration of 80 (grid units). In the bottom row we show the FTLE error maps with respect to ground truth. We can see that optimizing for flow samples outperforms directly fitting to vectors across all reduction rates. In Figure 4.5 we show the flow map error for different durations. All the points are seeded on the grid at time step t=0 and integrated using Euler integration scheme. Consistent with the FTLE plots, we observe an improvement in performance with our method in representing the flow map. We find that when the reduction rate is comparatively low (e.g 50) both methods are able to learn a good representation of the underlying vector field and thus the flow map error is rather low. However, the difference between the two optimization techniques is more prominent under aggressive reduction rates (e.g 300 and 400). We can see that flow map-based optimization better preserves the features (e.g the swirling motions in the FTLE) as compared to optimizing for the vectors directly.

4.2.2 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 (115), 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. 4.6(a) shows the results of

Table 4.1: We list the total training time in minutes, Inference time (time taken to integrate 10,000 particles for a duration of 100 grid-time) in seconds, and the model size for different reduction rate.

Reduction Rate	Training Time	Inference Time	Model Size
50	209.68	53.81	5.4
100	92.75	40.05	2.7
200	40.3	23.06	1.3
300	26.11	18.72	0.90
400	20.40	14.23	0.67



Figure 4.5: We qualitatively compare our method – Neural Flow Map to Neural Vector across different reduction rates. We show (top) the FTLE (generated by integration particles seeded at t = 1100 and for a duration of 80 in grid-time, and (bottom) difference images between the approximated FTLE and the ground truth FTLE. We find that optimizing for flow map samples yields better performance across all the reduction rates as compared to optimizing for the vectors themselves. We highlight the differences in purple.



Figure 4.6: 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.

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.



Figure 4.7: 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).

4.2.3 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 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 4.6(b), we can see the results

Table 4.2: 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)(35)	$\frac{(x \times y \times z \times t)}{128 \times 128 \times 512}$	10	16x
Double Gyre (2D)(115)	256x128x512	10	16x
Fluid Simulation (2D)(56)	512x512x1001	5	16x
Tornado	128x128x128x50	3	100x
Isabel	500x500x90x48	3	300x
ScalarFlow(26)	100x178x100x150	5	300x
Half Cylinder(101)	640x240x80x151	3	300x

Table 4.3: 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

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.

4.2.4 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 vectorbased 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.

Fig 4.7 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. 4.7(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. 4.6 and Fig. 4.7 is most likely due to how the training data is generated (please see Sec. 4.2.5) where we generated fixed duration flow map samples using forward integra-

tion 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.

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. (56), where we train a CNNbased 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 (6), 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 (120; 130; 81), 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 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 4.2 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.

4.2.5 Implementation Details

Network Architecture Settings Our network architecture is adapted from prior work by Lu et al. (81) – comprised of fully-connected layers and sinusoidal activation function. We depart from Lu et al. in that we utilize Rezero (5) 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



Figure 4.8: 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 the fluid simulation, four rotating centers, and double gyre dataset respectively). We find that our flow map method yields improved performance across all baselines.

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 (65) 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.



Figure 4.9: 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.

4.2.6 2D Unsteady Flow

We first experimentally evaluate our method on 2D unsteady flows. In Fig. 4.8, 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. (56), 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 (56)), in addition to more standard 2D unsteady flow datasets,

In Fig. 4.9 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 4.2); 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.

4.2.7 3D Unsteady Flow

We next evaluate our method on a collection of 3D unsteady flows. First, we show in Fig. 4.10 a comparison of different methods for capturing details in the FTLE for the ScalarFlow dataset. Relative to TTHRESH (6),



Figure 4.10: 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 (6). 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 4.11: 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 (6), 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. 4.12).

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. 4.11 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. 4.12 we measure the performance of our method, in terms



Figure 4.12: 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.

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. 4.11, for ScalarFlow our method generally obtains large improvement in the flow map, particularly for longer integration duration, further verified qualitatively in Fig. 4.10. Thus, it's critical to note that the true depth of reconstruction quality extends beyond the PSNR metrics. Complementary measures like Finite-Time Lyapunov Exponents (FTLE) that are sensitive to topological differences can shed light on aspects of reconstruction accuracy that PSNR values may miss. Hence, by considering these measures alongside PSNR, we gain a more detailed insight into the degree to which a method succeeds in preserving the underlying flow structures.

4.3 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 4.3). Like neural ODEs (17), 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. (17), 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 (17; 102) 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 (137), 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 (119; 128), in particular, learned initializations for rapid training adaptation to novel signals (128). 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 (56), 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 (84) 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.

CHAPTER 5

Scale-reinforced Implicit Neural Representations

In this chapter, we present a method aimed at enhancing the effectiveness of implicit neural representations (INRs) as compressive representations, making them better suited for downstream visual analysis requirements. More specifically, we introduce the concept of scale within INRs. We achieve this by modulating the parameters of an INR, left fixed, with a corresponding set of learned scale-dependent parameters, optimized such that the resulting scale-reinforced INR accurately models a given scale-space field representation. We show how such a weight modulation can be viewed as a multiplicative weighting of coefficients within an exponentially-large linear combination of sinusoidal bases. As a consequence, for scale spaces based on low-pass filters, our scheme implicitly performs filtering in the frequency domain. Through leveraging other properties of INRs, this enables compressive, scale-dependent feature analysis, e.g. acceleration computed from a compressive Gaussian filtered time-varying vector field (c.f. Fig. 1a), directly accessible from the network. Yet our approach is not limited to linear filters: we can support other types of scale, e.g. those based on data-dependent, nonlinear filters, demonstrated in Fig. 1b for a bilateral filter (135) scale-space applied to the medical image of a lung. Furthermore, we show how one can distill a scale-based INR into a more compact, scale-independent INR, fixed upon a provided scale - a consequence of most scale-spaces yielding simplified fields. We show the generality of our method on a diverse set of scale spaces, ranging from Gaussian filtering, bilateral filtering, and topological persistence-based filtering.

5.1 REINFORCING INRs with SCALE

A broader aim of our work is to make INRs a more viable representation for storing, accessing, and processing field-based data, in turn addressing storage and memory limitations with more conventional, sampled, field representations. An INR for field data can be viewed as a reduced, model-based representation, whose storage footprint is determined by the number of model parameters and their precision. Controlling for model size at the parameter level can enable highly compressive representations while faithfully approximating the field (21; 80; 84; 151). Furthermore, accessing the field's value at a given input amounts to a single forward pass through the network, while field derivatives can be computed through backpropagation (120). Hence, INRs enable compression-domain access of the field, and its derivatives. One can view field access in an INR as a point-based query, independent of all other points in the field's domain. Scale-space processing, however, typically necessitates processing of spatial regions (or, optionally, spatiotemporal regions), not merely individual points. Thus, if we would like to perform scale-space processing of an INR, then by



(A) Gaussian Filtering

(B) Bilateral Filtering

Figure 5.1: Our method seeks to incorporate scale spaces into implicit neural representations, specifically those that model field-based data. A central advantage of our method is its generality, in the support of varying types of fields, as well as scale spaces (Fig. 1a –time-varying vector field with Gaussian smoothing; Fig. 1b – scalar field with bilateral filtering). Our method can be used in concert with field derivatives directly accessible from the model, shown on the left as the norm of a derived acceleration field under varying scale. On the right, we show how our method can learn nonlinear data-dependent filters, leading to a reduction in noise within the pelvic region, whereas the structure of bones and organs are enhanced.

convention, obtaining a scale-parameterized sequence of transformations would necessitate sampling the INR, e.g. onto a regular grid. As a consequence, we lose the benefits of random-access querying of the field, conditioned on a particular scale value. It is this limitation that we address in our work, namely, how to build a general notion of scale within an INR, retaining the benefits of compression-domain field access, both for the field directly and its derivative.

5.1.1 Background: Implicit Neural Representation

INRs are a class of neural networks that accept as input a position from the field's domain (e.g. x, y, z positional coordinates), and produce values in the field's range (a single value for a scalar field, multiple values for a vector field). As our approach targets a broad class of INRs in the literature, we first present a unifying view of existing INRs, structuring methods by (1) how they encode position , and (2) network architecture details.

5.1.1.1 SIREN

The approach of SIREN (120) employs a sinusoidal embedding as its positional encoding along with an MLP as its network architecture. More specifically, for a given d_i -dimensional point in the domain $x \in \mathbb{R}^{d_i}$, SIREN performs the following sequence of operations:

$$z^{(0)}(x) = \gamma(x) = \sin(\Omega x), \tag{5.1}$$

$$z^{(l)}(x) = \sin(W^{(l)}z^{(l-1)}(x)), \tag{5.2}$$

$$f(x) = W^L z^{(L-1)}.$$
(5.3)

In the above, we treat $z^{(l)}$ as the learned representation output at a given layer l, within an L-layer MLP, and $W^{(l)}$ are weight matrices that comprise the parameters of the network, $1 \le l \le L$. We assume a layer width of D used in all layers for simplicity. The very first learned representation, $z^{(0)}$, is SIREN's positional encoding, which applies a sine function, element-wise, to a linear projection of the input, denoted Ωx . All intermediate layers similarly use sinusoids as activation functions, while the final layer maps the representation $z^{(L-1)}$ to an element in the field's range, which we assume to be a d_o -dimensional vector, e.g. $d_o = 1$ for a scalar field. For simplified notation, we omit bias terms in the above; in practice each learned weight matrix $W^{(l)} \in \mathbb{R}^{D \times (D+1)}$ for l < L, and all representation vectors $z^{(l)}$ are appended with a value of 1. A similar process is done for $\Omega \in \mathbb{R}^{D \times (d_i+1)}$ along with the last output layer $W^{(L)} \in \mathbb{R}^{d_o \times (D+1)}$.

5.1.1.2 MFN

An MFN (29) uses a series of positional encodings, one for each layer (excluding the output), alongside an architecture that joins subsequent layers through a linear projection, in conjunction with an element-wise product:

$$z^{(0)}(x) = \gamma^{(0)}(x), \tag{5.4}$$

$$z^{(l)}(x) = [W^{(l)} z^{(l-1)}(x)] \circ \gamma^{(l)}(x),$$
(5.5)

where \circ is the Hadamard product. The output of the model is a simple linear projection, as in Eq. 5.3.

The main distinction in an MFN is the type of positional encoding $\gamma^{(l)}$ used at a given layer *l*. We consider the two flavors of MFNs proposed by Fathony et al.; first, the so-called Fourier encoding :

$$\gamma^{(l)}(x) = \sin(\Omega^{(l)}x), \tag{5.6}$$

where each $\Omega^l \in \mathbb{R}^{D \times (d_i+1)}$ as defined above. The second positional encoding is one that is spatiallysensitive, the so-called Gabor encoding that incorporates a per-element Gaussian weighting :

$$\gamma^{(l)}(x) = \exp\left(-\lambda^{(l)}d^2(x, X^{(l)})\right) \circ \sin(\Omega^{(l)}x),\tag{5.7}$$

where $X^{(l)}$ contains D total d_i -dimensional points, $d^2(x, X^{(l)})$ forms a D-dimensional vector of squared Euclidean distances between the given input x and each point in $X^{(l)}$, while $\lambda \in \mathbb{R}^D$ is a per-element scale of the Gaussian. The points $X^{(l)}$ and scales $\lambda^{(l)}$ are treated as learnable parameters, alongside weight matrices $W^{(l)}$.

5.1.1.3 Localized Implicit Neural Representation

Lastly, we consider INRs that are designed to scale to larger data sizes, both for optimization and inference, in contrast with using a single MLP or MFN as in the above. We utilize a variant of localized methods (18; 86; 127), and define a coarse spatial grid of learnable frequencies, which we denote Z, whose shape depends on the input dimensionality d_i . For instance, when $d_i = 2$ this amounts to $Z \in \mathbb{R}^{D_i \times w \times h}$, where the first dimension D_i denotes feature dimensions, e.g., a collection of learnable frequencies, while w and h denote, respectively, the width and height dimension. Similar grids can be formed for $d_i > 2$ domains.

Given an input $x \in \mathbb{R}^{d_i}$, we form its positional encoding by interpolating within the grid Z, elementwise (127):

$$z^{(0)}(x) = \gamma(x) = \sin(\operatorname{interp}(x, Z)), \tag{5.8}$$

where interp corresponds to an interpolation of the frequencies defined at the locations of *Z*. In practice, we use linear interpolation, performed element-wise, as considered in Takikawa et al. (127). Once the positional encoding is obtained, we adopt the remainder of SIREN's architecture, namely Eq. 5.2 and Eq. 5.3, to compute the field's output.

5.1.2 Fitting to a Field

In our approach, we assume that we have been provided a sampled field, whose domain is d_i -dimensional and range is d_o -dimensional. We further assume an arbitrary INR has been chosen, ostensibly one whose number of weights is much smaller than the field's resolution, to obtain a compressive representation. We then optimize the INR to fit the field:

$$\min_{\Theta} \sum_{x \in D} \|f_{\Theta}(x) - h(x)\|_{2}^{2},$$
(5.9)

where D contains all samples of the field, e.g., a regular grid, h denotes the ground-truth field, and f_{Θ} denotes the INR with Θ being the collection of parameters for a particular INR, e.g., parameters of its



Figure 5.2: We illustrate the basic idea behind our method for building scale into INRs. On the top, we notionally depict an INR – namely its positional encoding, followed by a fully-connected layer – used in part to fit the 512³ Hazelnuts field (top-right). Links encode network weights, where link thickness encodes a weight with large magnitude, and red/blue color encodes positive/negative. Our approach (bottom) learns to modulate the weights in an existing INR, in this case decreasing their magnitude in proportion to a provided scale, in order to accurately model a given scale-space transformation, here showing one based on a bilateral filter.

positional encoding and weight matrices. In practice, we optimize an INR using gradient descent, sampling uniformly at random from the field's domain in *D* at each step.

5.1.3 Learning scale space

Once an INR has been optimized, to endow the model with scale, our approach takes its positional encoding and architecture, and freezes all such parameters – they will remain fixed throughout learning a scale space. Instead, we introduce a new set of modulation matrices, denoted $M_{(s)}^{(l)}$, for $1 \le l \le L$, each of which parameterized by a single value *s* that represents the amount, or level, of scale within the scale space. Importantly, a value of s = 0 represents no scale, and should correspond to the originally fit field of an INR (c.f. Eq. 5.9). Each matrix $M_{(s)}^{(l)}$ is in direct correspondence with a weight matrix in an INR $W^{(l)}$, specifically of identical size, and is strictly nonnegative: $M_{(s)}^{(l)} > 0$. We derive a new, scale-dependent, weight matrix by modulating the weights, element-wise, via $M_{(s)}^{(l)}$:

$$W_{(s)}^{(l)} = M_{(s)}^{(l)} \circ W^{(l)}$$
(5.10)

We then replace the original weight matrices $W^{(l)}$ in an INR with their scale-dependent counterparts $W_{(s)}^{(l)}$, and retain the rest of the functionality of the INR for the purposes of both optimization and inference. Importantly, we do not modify the positional encoding, only weight matrices. Given the augmented INR, we next wish to optimize for the modulation matrices, such that it is capable of reproducing a full scale-space representation of the field. To this end, we assume that a scale space has been provided as a scale-parameterized series of function transformations, which we denote G_s , mapping from one field to another field. As one example, to realize a Gaussian scale space the transformation G_s boils down to a convolution with a Gaussian filter, and *s* corresponds to the Gaussian's scale parameter. We thus seek to optimize the following:

$$\min_{\bar{\Theta}} \sum_{s \ge 0} \sum_{x \in D} \|f_{\bar{\Theta}}(x,s) - (G_s \odot h)(x)\|_2^2.$$
(5.11)

In the above, the operation of transforming the field *h* by G_s is denoted \odot , while $\overline{\Theta}$ corresponds to, collectively, all modulation matrices $M_{(s)}^{(l)}$, and $f_{\overline{\Theta}}$ now conditions on (1) an input in the field domain, and (2) a scale value *s*. Intuitively, the intent of optimization is to learn a series of scale-dependent INRs that faithfully represent the scale-dependent transformations (G_s) of the original field (*h*).

What remains is a specific formulation of the modulation matrices. We introduce a single, scale-independent, matrix M_l , in correspondence with weight matrix W_l . We then form the scale-dependent matrix as follows:

$$\boldsymbol{M}_{(s)}^{(l)} = \left(\boldsymbol{\sigma}(\boldsymbol{M}^{(l)})\right)^{(\boldsymbol{\alpha}_l \cdot s)}.$$
(5.12)

where σ is an element-wise sigmoid function, and α_l is a layer-specific learnable scalar. In effect, the sigmoid ensures each entry in the matrix lies in the range [0,1]; we then element-wise exponentiate each entry by $(\alpha_l \cdot s)$. The α_l term can be viewed as a way to calibrate the domain of the scale, mapped into an appropriate range for exponentiating the modulation matrices. If $\alpha_l > 0$, this implies that as *s* increases, the entries of $M_{(s)}^{(l)}$ will monotonically decrease, and thus the magnitude of entries in $W_{(s)}^{(l)}$ will also monotonically decrease. The intuition behind this scheme is that certain entries within an INR's set of weight matrices are more predictive of the scale-induced transformation of the field than others. Through optimizing Eq. 5.11, we intend to find these very weights. Fig. 5.2 shows a graphical depiction of this process. Note that when s = 0,



Figure 5.3: Our approach is able to endow scale onto a broad class of INR models. Here we optimize models ranging from SIREN, two variants of MFNs (Fourier, Gabor), and a local INR (SIREN-Grid), to fit a Gaussian scale-space of a 2D scalar field (left), all of whose parameters lead to a compression ratio of 30x. As demonstrated quantitatively (right), and qualitatively for SIREN (top), all models are able to adequately learn a scale space for this field, as indicated by the high PSNR values.

each modulation matrix will become a matrix of exclusively 1's. Thus when modulating the weight matrices through Eq. 5.10, we obtain back the original weight matrices $W^{(l)}$, and the original INR remains unchanged.

A key feature of our approach is its generality: given the set of INRs discussed in Sec. 3.1, our method can readily be applied to all such models. Indeed, what these models have in common is the transformation of a representation at one layer ($z^{(l-1)}$), to a subsequent layer ($z^{(l)}$), and this is, in part, realized through a linear transformation $W^{(l)}$. To experimentally verify this claim, we have taken each of these INRs – SIREN, Fourer (MFN), Gabor (MFN), and the grid-based SIREN – and fit each to a Gaussian scale space of a 4096 × 4096 scalar field, originating from a turbulence simulation (74). We train each model on a sequence of equal-spaced scales, and evaluate the models on these scales, as well as all scales in between to assess generalization. Fig. 5.3 shows the results, demonstrating that all such models are capable of learning scale-space representations, namely that they retain high peak signal-to-noise ratio (PSNR) for scales both observed, and not observed, during training. The main difference between models largely lies in differences in positional encodings, and architectural differences. Of note, we find that for the grid-based approach (SIREN-Grid), the performance tends to decrease more quickly over scales, relative to other models. We anticipate a trade-off

in (1) the resolution of the coarse grid (here 8^3 , set in accordance with compression ratio) and (2) the MLP on which the scale space is learned. Namely, as the spatial support of the scale approaches the (coarse) grid resolution, we expect the localized positional encoding to have an effect on the learned scale space.

5.2 Analysis

Our choice to modulate the weights of an INR, for the purpose of realizing scale-space representations, is motivated by recent work that seeks to characterize the space of functions that can be represented by an INR (29; 151). More specifically, an INR that uses a sinusoidal positional encoding (e.g. of the form Eqs. 5.1 or 5.6), regardless of architecture, can be written as a linear combination of sinusoidal basis functions:

$$f_{\Theta}(x) = \sum_{\omega \in \Phi} \alpha_{\omega} \sin(x^T \omega), \qquad (5.13)$$

where Φ represents a collection of frequencies that are strictly a function of the learned positional encodings, e.g. for SIREN this boils down to matrix Ω (c.f. Eq. 5.1), while for a Fourier MFN the dependency is on all matrices $\Omega^{(l)}$ (c.f. Eq. 5.6). The coefficients α_{ω} are strictly a function of entries in the weight matrices, and independent of positional encoding. In particular, for an MFN, the size of the set, which we denote $|\Phi|_{MFN}$ can be precisely written as (77):

$$\Phi|_{\rm MFN} = \sum_{i=0}^{L-1} 2^{iD^{i+1}}.$$
(5.14)

We can thus see that the number of basis functions grows exponentially in the number of layers. For MLP architectures, Yuce et al. (151) shows that the frequency set Φ amounts to integer-valued linear combinations of frequencies as defined in Ω – the specific formation is a function of the activation function, e.g. the sine function in Eq. 5.2. Similar to MFNs, this results in sinusoidal basis functions that grow exponentially in layers. Last, for Gabor positional encodings (c.f. Eq. 5.7) a similar result holds for MFNs (29), while for grid-based SIRENs, similar conclusions can be drawn, replacing dot products in Eq. 5.13 with interpolation of frequencies.

5.2.1 Weight Modulation as Frequency Filtering

In light of the representational space of INRs, we can now characterize our weight modulation scheme: The modulation of weights manifests as the following equivalent linear combination:

$$f_{\overline{\Theta}}(x) = \sum_{\omega \in \Phi} (m_{\omega}(s) \cdot \alpha_{\omega}) \sin(x^T \omega), \qquad (5.15)$$

where $m_{\omega}(s)$ corresponds to a multiplicative combination of entries found in modulation matrices $M^{(l)}(s)$. Please see Sec. 5.2.2 for the proof. Assuming $\alpha_l > 0$, we thus observe that weight modulation has the effect of filtering in the frequency domain, e.g. as the scale s increases, we are decreasing the influence of the weight α_{ω} for frequency vector ω . If our scale space is based on a low-pass filter (e.g. a Gaussian), our approach will implicitly reduce the importance of coefficients that correspond to high frequencies. This stands in contrast to mip-NeRF (8) and BACON (77), where scale is controlled at the frequency level (ω). Although these methods can handle low-pass filters to a certain extent, controlling for scale at the coefficient level (α_{0}) permits us to build a broader family of scale-parameterized transformations into an INR. The expansion in Eq. 5.15 is not feasible to compute (see: Eq. 5.14); the formation of MLPs and MFNs in computing a network output is far more economical. Nevertheless, it is instructive to explicitly form the expansion, in order to verify the claims made above, namely, that optimizing Eq. 5.11 for a Gaussian scale space should filter out high frequencies - here we treat "frequency" as the norm of the 2-vector of frequencies resulting from the expansion. We have taken a 300×300 medical image, and trained an MFN with Fourier positional encoding consisting of 2 hidden layers, and layer width D = 209 – its equivalent expansion amounts to $|\Phi| \approx 10^8$. We have trained both Gaussian and bilateral filtering scale spaces for this image. The sheer number of frequencies, and wide range of weights presents challenges for depicting the contributions of certain frequency ranges. To this end, we uniformly split frequencies into a small number of bins, and within each separate frequency bin, we normalize counts over a (log-spaced) binning of weight magnitudes. Fig. 5.4 shows the results, where we observe that for a Gaussian scale space, the contribution of high frequencies are substantially diminished as we increase in scale – the effects of this are further emphasized qualitatively in the smoothed images. Yet a key feature of INRs is that they do not merely learn a grid-aligned frequency representation, the set of frequencies formed by Φ are far more expressive. Thus we can learn more general scale spaces, as demonstrated by the bilateral filter scale space, wherein, as one would expect, we diminish some high frequencies while keeping those that contribute to the features preserved in the image (e.g. the bone structure in this image).

5.2.2 Proof of Theorem 5.1.1

In this section we provide the proof of Theorem 5.15 for the 2 forms of INRs covered in this work: multiplicative filter networks (MFN), and multi-layered perceptrons (MLP) with sinusoidal positional encodings. In the following we omit bias terms for simplicity, but they are straightforward to incorporate, only adding a negligible amount of terms to the expansion.



Figure 5.4: This plot demonstrates the relationship between the frequencies' L2 norm and their associated weights for a Fourier-based MFN, trained for a medical image endowed with scale spaces for a Gaussian filter, and bilateral filter, respectively.

5.2.2.1 MFN

Following Fathony et al. (29), whether considering a Fourier positional encoding or a Gabor positional encoding, the equivalent expansion of an MFN (without introducing our modulation scheme) gives:

$$f_{\Theta}(x) = \sum_{\omega \in \Phi} \alpha_{\omega} \sin(x^T \omega).$$
(5.16)

The specific set of frequency vectors Φ can be found by considering the combination of components in positional encodings, across all layers:

$$\theta_{iL-1,iL-2,\dots,i1,i0}(x) = \gamma_{iL-1}^{(L-1)}(x) \cdot \gamma_{iL-2}^{(L-2)}(x) \dots \gamma_{i1}^{(1)}(x) \cdot \gamma_{i0}^{(0)}(x),$$
(5.17)

where each index i_l varies over [1,D]. This product of positional encodings can be expressed as a weighted sum of the individual frequencies [1, 2], and the specific form of this expansion depends on the type of positional encoding (Fourier vs. Gabor). Nevertheless, for a single configuration of indices (iL - 1, iL - 2, ..., i1, i0), we can form a weight that is in direct correspondence, where for simplicity, we will assume the field's output dimension is 1 (e.g., a scalar field):

$$W_{iL-1,iL-2,\dots,i1,i0} = W_{1,iL-1}^{(L)} \cdot W_{iL-1,iL-2}^{(L-1)} \dots W_{i2,i1}^{(2)} \cdot W_{i1,i0}^{(1)},$$
(5.18)

Recall that our modulation scheme performs the following, at every layer *l*:

$$W^{(l)}(s) = M^{(l)}(s) \circ W^{(l)}.$$
(5.19)

Thus, following Eq. 5.18, we can form an analogous weight over all combinations of indices:

$$W_{iL-1,iL-2,\dots,i1,i0}(s) = W_{1,iL-1}^{(L)}(s) \cdot W_{iL-1,iL-2}^{(L-1)}(s) \dots W_{i2,i1}^{(2)}(s) \cdot W_{i1,i0}^{(1)}(s).$$
(5.20)

The Hadamard product of matrices formed by our modulation scheme implies that we can separate terms (1) exclusive to the modulation and (2) exclusive to the (already optimized) INR. For the former, we have:

$$M_{iL-1,iL-2,\dots,i1,i0}(s) = M_{1,iL-1}^{(L)}(s) \cdot M_{iL-1,iL-2}^{(L-1)}(s) \dots M_{i2,i1}^{(2)}(s) \cdot M_{i1,i0}^{(1)}(s),$$
(5.21)

and thus we can express each weight in the explicit expansion of Eq. 5.15 as:

$$W_{iL-1,iL-2,\dots,i1,i0}(s) = M_{iL-1,iL-2,\dots,i1,i0}(s) \cdot W_{iL-1,iL-2,\dots,i1,i0}.$$
(5.22)

The product in Eq. 5.17 can be expressed as a sum of 2^{L-1} terms; consequently, a weight $W_{iL-1,iL-2,...,i1,i0}(s)$ is in direct correspondence with each of these individual basis functions, derived from the product-to-sum rule for sinusoids (and exponentials for the Gabor encoding). This set of basis functions forms Φ , and thus we can identify an individual modulation weight $m_{\omega}(s)$ and (frozen) INR weight α_{ω} with each frequency vector $\omega \in \Phi$.

Before we cover the case of MLPs, we remark that, by design, the parameters for the positional encodings across layers are not modulated. This implies that our proposed scheme leaves frequencies unchanged. Only the weight coefficients associated with frequencies are adjusted.

5.2.2.2 MLP

Following Yüce et al. (151), for an MLP, we will assume that its nonlinear activation can be well-approximated by a K-degree polynomial. For a sine activation, we have:

$$\sin(x) \approx \sum_{k=0}^{K} \beta_k x^k, \tag{5.23}$$

where the coefficients β_k are specific to the sine activation but otherwise constant. While Yüce et al. [3] studied the formation of sinusoidal bases resulting from an MLP with a sinusoidal positional encoding, here we will consider the coefficients formed, those in correspondence with these bases.

We will start by considering the post-activation vector formed after the first nonlinearity:

$$z^{(1)}(x) = \sin\left(W^{(1)}\sin(\Omega x)\right),$$
 (5.24)

and denote the positional encoding by $\gamma^{(0)}(x) = \sin(\Omega x)$. Considering our polynomial expansion (Eq. 5.23), we have:

$$z^{(1)} \approx \sum_{k=0}^{K} \beta_k \left(W^{(1)} \gamma^{(0)}(x) \right)^k.$$
(5.25)

Isolating a single component *j*:

$$z_j^{(1)} \approx \sum_{k=0}^{K} \beta_k \left(\sum_{t=1}^{D} W_{j,t}^{(1)} \gamma_t^{(0)}(x) \right)^k.$$
(5.26)

This expansion takes on a similar form to MFNs, in that we are computing products over frequencies as well as products over weights—specifically, weights within a single row of $W^{(1)}$. Thus, for a given k with $1 \le k \le K$, we can explicitly form these weight combinations:

$$W_{j,t_k,t_{k-1},\dots,t_2,t_1}^{(1)} = W_{j,t_k}^{(1)} \cdot W_{j,t_{k-1}}^{(1)} \cdot \dots \cdot W_{j,t_2}^{(1)} \cdot W_{j,t_1}^{(1)},$$
(5.27)

where each $t_i \in [1, D]$. Note that for k = 1, this would be all weights in row j of the matrix $W^{(1)}$. Correspondingly, we have an expansion in our positional encodings:

$$\gamma_{t_k,t_{k-1},\dots,t_2,t_1}^{(0)}(x) = \gamma_{t_k}^{(0)}(x) \cdot \gamma_{t_{k-1}}^{(0)}(x) \cdot \dots \cdot \gamma_{t_2}^{(0)}(x) \cdot \gamma_{t_1}^{(0)}(x).$$
(5.28)

Note that this can be written as an integer-valued linear combination of sinusoidal bases, again due to the product-to-sum rule of sinusoids, similar to MFNs.

Incorporating our modulation scheme, we can form an analogous set of modulation products:

$$M_{j,t_k,t_{k-1},\ldots,t_2,t_1}^{(1)}(s) = M_{j,t_k}^{(1)}(s) \cdot M_{j,t_{k-1}}^{(1)}(s) \cdot \ldots \cdot M_{j,t_2}^{(1)}(s) \cdot M_{j,t_1}^{(1)}(s),$$
(5.29)

and thus form a single modulated weight in the expansion:

$$W_{j,t_k,t_{k-1},\dots,t_2,t_1}^{(1)}(s) = M_{j,t_k,t_{k-1},\dots,t_2,t_1}^{(1)}(s) \cdot W_{j,t_k,t_{k-1},\dots,t_2,t_1}^{(1)}.$$
(5.30)

We form a set of frequencies at this layer $\Phi^{(1)}$ corresponding to all combinations of indices $(t_k, t_{k-1}, \dots, t_2, t_1)$, and thus express the output at the first layer as:

$$z_j^{(1)}(x) \approx \sum_{\boldsymbol{\omega} \in \Phi^{(1)}} (\boldsymbol{M}_{j,\boldsymbol{\omega}}^{(1)}(s) \cdot \boldsymbol{W}_{j,\boldsymbol{\omega}}^{(1)}) \sin(x^T \boldsymbol{\omega}),$$
(5.31)

At this stage, one can see the major distinction from an MFN: for an MLP, a single weight coefficient α_{ω} will be comprised of products of weights spread across input components (t_i indices), and likewise for the modulation weights. Yet importantly, like the MFN case, we see that the modulation terms $M_{j,t}^{(1)}(s)$ remain paired with their corresponding weights $W_{i,t}^{(1)}$.

We proceed by induction, assuming that at a given layer l, we may represent its output at some component t as:

$$z_t^{(l)}(x) \approx \sum_{\boldsymbol{\omega} \in \Phi^{(l)}} (\boldsymbol{M}_{t,\boldsymbol{\omega}}^{(l)}(s) \cdot \boldsymbol{W}_{t,\boldsymbol{\omega}}^{(l)}) \sin(x^T \boldsymbol{\omega}),$$
(5.32)

In considering the subsequent layer l + 1, we obtain:

$$z_{j}^{(l+1)}(x) \approx \sum_{k=0}^{K} \beta_{k} \left(\sum_{t=1}^{D} \sum_{\boldsymbol{\omega} \in \Phi^{(l)}} (M_{j,t}^{(l+1)}(s) \cdot M_{t,\boldsymbol{\omega}}^{(l)}(s)) \cdot (W_{j,t}^{(l+1)} \cdot W_{t,\boldsymbol{\omega}}^{(l)}) \sin(x^{T} \boldsymbol{\omega}) \right)^{k},$$
(5.33)

where we have grouped together the modulation terms and the weight terms, and we recognize that they individually spread across rows, as in Eq. 5.26. The frequency (ω) expansion follows, again, from repeated

application of the product-to-sum rule (151), and thus the resulting set of frequency terms, denoted $\Phi^{(l+1)}$, remain independent of the weights.

Considering the result formed by the multiplicative combination of modulation terms and weight terms, note that the number of terms in the new expansion at layer l + 1 will be of the order $O(DK^{(l+1)})$, since the expansion $\Phi^{(l)}$ from the previous layer l has an order of $O(DK^{(l)})$ terms (e.g., see Eq. 5.27 for layer l = 1), and at layer l + 1, the *k*-th degree polynomial will result in an order of O(DK) possible combinations of terms from $\Phi^{(l)}$. The modulation/weight terms in $\Phi^{(l)}$ will become mixed across all entries in all matrices, due to the matrix-vector product in Eq. 5.34.

Due to our modulation scheme being a Hadamard product of matrices, we can further separate weight terms and modulation terms in the products. Writing $\omega^{(l)}$ as a multi-index over different terms in layer *l*, in correspondence with Eqs. 5.27 and 5.29, we have:

$$W_{j,\boldsymbol{\omega}^{(l+1)}}^{(l+1)} = \left(W_{j,t_k}^{(l+1)} \cdot W_{t_k,\boldsymbol{\omega}^{(l)}}^{(l)}\right) \cdot \ldots \cdot \left(W_{j,t_1}^{(l+1)} \cdot W_{t_1,\boldsymbol{\omega}^{(l)}}^{(l)}\right).$$
(5.34)

and

$$M_{j,\boldsymbol{\omega}^{(l+1)}}^{(l+1)}(s) = \left(M_{j,t_k}^{(l+1)}(s) \cdot M_{t_k,\boldsymbol{\omega}^{(l)}}^{(l)}(s)\right) \cdot \ldots \cdot \left(M_{j,t_1}^{(l+1)}(s) \cdot M_{t_1,\boldsymbol{\omega}^{(l)}}^{(l)}(s)\right).$$
(5.35)

given a combination of indices $(t_k, t_{k-1}, \dots, t_2, t_1)$ in correspondence with the polynomial expansion. Thus, the weight products and modulation products can be separately expressed, each of which is in correspondence with one another in the expansion (Eq. 5.32). We remark that the frequency sets in the case of an MLP Φ differ from those in an MFN, in that for MLPs, a resulting frequency is expressed as an integer-valued linear combination of the input frequencies found in Ω . Thus, it is possible for low frequencies to be represented in the expansion for an MLP, e.g., choosing just one frequency in Ω (when k = 1 in polynomial expansion across all layers), whereas an MFN will always combine all frequencies across positional encodings. This gives a potential high-frequency bias towards MFNs compared to MLPs. A consequence is that our weight modulation scheme cannot correct for this high-frequency bias, leading to the prevalence of high-frequency artifacts. Experimentally, we have verified this in simple 2D fields, see, e.g., Fig. 5.5.

5.3 Distilling to a smaller network

Most scale spaces intend to simplify the fields on which they operate, e.g. reducing the contribution of high frequencies. This implies that our scale-dependent INR encodes fields of varying complexity, and thus if we are only interested in a single scale, we can distill (49) the model into a scale-independent INR with fewer parameters, directly from the original scale-dependent INR without requiring access to the original field.

To this end, we have adapted the neuron pruning approach of Molchanov et al. (88) for the case of



Figure 5.5: A comparison of SIREN with the Fourier positional encoding MFN, for learning a bilateral filter with scale s = 4.

coordinate-based MLPs. Specifically, we treat the scale-dependent INR as a "teacher" network (49), providing direct supervision for a "student" network. The student network is created by iteratively pruning rows and columns from the weight matrices (and corresponding bias values) of the teacher network. We prioritize rows by computing the magnitude of their weight derivatives, scaled by the magnitude of the weights themselves, as a proxy to measure the sensitivity of pruning a weight entry (88) – this is summed over all rows, and used in a greedy manner to prune rows that have low sensitivity. A row pruned at a given layer implies that its corresponding column in the next layer's weight matrix may also be pruned. We interleave this pruning procedure with fine-tuning the resulting reduced model, and iterate these steps until a user-supplied error tolerance is exceeded. We emphasize that weight sensitivity and fine-tuning are all carried out under the original scale-based INR; however, we experimentally verify that the discrepancy from ground-truth is not too large.

5.4 Results

To demonstrate the effectiveness of our approach we compare to a number of baseline techniques, and evaluate our approach across a number of datasets, flavors of INRs, and scale spaces. Please see Table 1 for an overview of datasets used in our experiments. We employ three types of scale spaces in our experiments: Gaussian filtering, bilateral filtering, and topological persistence-based filtering. For the Gaussian scale spaces, we associate its variance parameter as scale, expressed in units of the grid (e.g. a value of "1" would be one voxel unit in a volume). For bilateral filtering, we keep the variance for the data term fixed, and instead vary the spatial scale. Last, for topological filtering we treat the threshold at which critical pairs are filtered as our notion of scale, e.g. as scale increases, critical points are merged and a smoothed field of simplified persistence is produced – we use TTK's implementation (133). We note that unlike Gaussian/bilateral filtering, topological filtering is not a continuous scale space. We provide further implementation details on model settings and optimization in the following section.

5.4.1 Implementation Details

There are two optimization schemes with our approach: (1) fitting an INR to a field, (2) and then building scale into the INR. For field fitting, for simplicity and fair comparison we optimize each INR in the same manner: we perform 80K optimization steps, using a batch size of 40K samples per step. We take samples uniformly-at-random from the ground truth field. Unless otherwise specified in the main text, each model has 5 hidden layers. In each model, we express a compression ratio, and subsequently compute the number of parameters that will lead to a network size with, approximately, the desired compression. For SIREN, Fourier, and Gabor, this amounts to setting the layer width to, with Fourier and Gabor considering the per-layer positional encodings. For the grid-based SIREN, unless otherwise stated, to gain benefits of locality we fix the layer width to 256 and then derive the coarse grid resolution that would achieve the provided compression ratio. We use the ADAM optimizer, setting the initial learning rate to be inversely proportional to the model's layer width for stability in optimization (ranging from 10^{-5} to 10^{-4}). During training, we linearly decay the learning rate to 0 upon concluding optimization.

In practice, we find that building scale into an already optimized INR is significantly faster to train than the original fit, and more stable. Specifically, we take 4K optimization steps, using the ADAM optimizer with an initial learning rate of 10^{-1} , linearly decaying the learning rate to 0. We assume a fixed number of scales on which we can access on demand, the scale-transformed field, and in each optimization step, we take a number of samples over the field domain, at different scales, to learn the modulation weights. In practice, we take 80K samples distributed equally over all training scales. The modulation matrices M(1) are initialized to a value of 5, and the 1 parameters set to 1. This setting ensures that the sigmoid function produces values close to 1, thus resembling the original INR prediction and serving as a reasonable initialization. This helps to prevent saturated activations. We found that initializing to a value of 0, and thus having the sigmoid produce a value of 0.5, led to poor local minima in optimization.

For our bilateral filter, we treat the spatial bandwidth in the Gaussian as the primary scale in the scale space, whereas the data term is fixed and set to a constant – this constant is ultimately data-dependent and subjective. We have strived to set this to ensure sharp features are preserved through the filter. For topological persistence, to generate a scale space, we define scale with respect to the threshold at which critical pairings



Figure 5.6: Comparison of our method with an alternative model that explicitly conditions on scale as a coordinate. Our method fares much better in generalization, emphasizing the importance of how to represent scale within an INR. Dataset: MHD-B (256³)

are removed. We manually set a maximum threshold, say, and equally sample over the interval [0,] to produce a series of scalar fields that serve as supervision.

5.4.2 Comparision

5.4.2.1 Scale-as-Coordinate

It is natural to ask whether a simpler, alternative scheme can be used to build scale into an INR. One option is to explicitly condition on the scale value as an additional coordinate input to the network; this scheme has been used in prior work, see e.g. ACORN (84), but not for explicit scale-space learning. We compare to this baseline, in order to assess generalization: for the 256³ MHD-B scalar field we learn a SIREN. based Gaussian scale space from scales $s \in [1,9]$. We optimize our scale-reinforced model, and said baseline, under differing number of scales observed during training: $\{1,9\}$, $\{1,5,9\}$, and $\{1,3,5,7,9\}$. We evaluate the methods on scales trained, and withheld scales, inclusive of the original field (e.g. a scale of s = 0).

Fig. 5.6 shows the results of this experiment. We first note that the baseline – treating scale as a coordinate – clearly fails to generalize across scales that it did not observe during training. Our method sees good performance once provided with 3 unique scales at training, but still performs reasonably well when given just scales $\{1,9\}$, suggesting that our proposed method is learning a meaningful representation of scale. The baseline method overfits, or "memorizes", to scales provided during training, which might be acceptable if an end user requires just a small number of scales for processing. However, this comes at a cost: the fit to


Figure 5.7: We compare our method against the tensor-based Tucker decomposition, which permits compression-domain linear filtering. In the linear case (Gaussian), our method is competitive, but for datadependent filtering (Bilateral) our method outperforms Tucker decomposition, and without needing to resample to a grid. On the bottom, we show a visual comparison for Miranda under Gaussian filtering at scale s = 2, and Hazelnuts under bilateral filtering at scale s = 4.

the original field worsens as the number of scales for training increases. This is a central advantage of our approach being retrospective, and instead reinforcing scale within an existing INR.

5.4.2.2 Tensor-based Filtering

We compare our method to Ballester-Ripoll et al. (6), where linear filtering can be exactly performed in the compression domain of a Tucker-based tensor decomposition. We consider a Gaussian scale which exactly fits this assumption, and a bilateral filtering scale, wherein it is necessary to resample to a grid, and sub-sequently perform bilateral filtering, due to the data-dependent filtering. We have set both methods (ours – SIREN) to have approximately equivalent compression ratios, where data reduction is strictly in terms of



Figure 5.8: We show the results of our method applied to a diverse range of fields, across different types of INRs. Each INR is compressive by design, while still enabling access to a scale-parameterized family of transformations. By representing a field as a coordinate-based neural network, we can perform back-propagation to compute spatial derivatives, thus enabling compressive, scale-dependent features on the field, shown here as gradient magnitude for the pressure-based Isotropic Turbulence (bottom left), and vorticity of a vector field for MHD (top right).

model reduction, and no other postprocessing (e.g., no quantization is performed (7; 80)). Fig. 5.7 shows the results for two different volumes, across a set of scales, where we evaluate on the scales that we have trained on. For Gaussian filtering, we observe that our method is mostly an improvement. As shown qualitatively, for smaller scales, our method can excel in preserving features, highlighted for Miranda under Gaussian filtering at a scale of s = 2. In certain cases, however, filtering the compressed Tucker-based volume can lead to an improvement, where we hypothesize that the noise introduced by the low-rank tensor decomposition is of high frequency, and thus smoothed out under Gaussian filtering. For bilateral filtering, however, our method consistently outperforms the low-rank tensor decomposition, and without necessitating decompression to a regular grid. Here, the compression artifacts introduced by the Tucker decomposition cannot be overcome through bilateral filtering, which attempts to retain data-dependent features (e.g., sharp edges) whilst removing noise. On the bottom, we show a visual comparison for Hazelnut within a bilateral filtered scale space, further conveying our method's ability to learn a meaningful representation of scale; in contrast, bilateral filtering a decompressed volume leaves artifacts intact.

5.4.3 Experimental Results

We have augmented existing INRs across a variety of models, and different forms of datasets. Fig. 5.8 shows the quantitative and qualitative results for our method applied to four different types of INRs (c.f. Sec 3.1),

and four volumes that range from (1) a CT scan, (2) scalar fields of turbulence simulations (pressure and temperature fields), and (3) a vector field resulting from a turbulence simulation. In these experiments, we use a Gaussian scale space, where each model is trained on odd-valued integer scales ranging from 1 to 9, and evaluated on all integer-valued scales within [1,9]. Hence, even-valued integers correspond to scales not observed during training. From the plots depicting PSNR over scale, we find that all INRs successfully generalize over these scales, as evidenced by no noticeable drops in performance in PSNR at withheld scales. A notable advantage of our approach is that we may access scale-dependent derivatives from an INR, simply through backpropagation, conditioned on a given scale. We show this qualitatively via the gradient magnitude of the pressure field (lower left), leading to well known tube-like structures in pressure gradients within isotropic turbulence that are gradually diminished over larger spatial scales of smoothing (of the pressure field, not its gradient). Likewise, we show the vorticity of the (scale-space) vector field in this magnetohydrodynamics simulation.

In general, we find that the SIREN INR tends to perform best. As discussed in detail in the Sec. 5.2.2, MLP architectures are, in general, more expressive than MFNs, thus they have a higher capacity for learning a better representation of scale. On the other hand, the grid-based model also uses an MLP akin to SIREN, but whose positional encoding is based on a linear interpolation of learned features at grid vertices. In these experiments, the grid resolution is left as a constant, whereas its layer width adjusts to compensate for a target compression. For smaller data spatial resolution (e.g. MHD), we find that the grid-based approach can deteriorate in performance, particularly when the spatial support of a chosen scale approaches the size of a cell within the coarse grid. As we believe that localized implicit neural representations are a practical representation for the visualization of large-scale field-based data (127; 77; 145), we study the effect of the coarse grid resolution on model performance, here for the 1024³ Flower CT scan. In contrast with Fig. 5.8 (top-left), here we study bilateral filtering. We vary the coarse grid resolution over 8^3 , 16^3 , and 32^3 , while adjusting the layer width of the models (D) to attain a compression ratio of 400x in each. Fig. 5.9 shows the performance of each model, along with timings pertaining to (1) fitting to the field, (2) learning the bilateral filtering scale space, and (3) time required to perform inference, at a scale of s = 7, in reconstructing the bilateral filtered volume. We observe that as the grid resolution becomes too large (32^3) , performance begins to suffer. However, at 16^3 , we only observe a small drop in performance, while training and inference time both strike a reasonable compromise. In particular, we also show the time required to perform bilateral filtering on the original volume for scale s = 7, under a GPU-optimized PyTorch implementation. Here we see that it is slower than all such methods, approximately 3x slower for 16^3 coarse grid resolution. This example demonstrates the inference-time benefit of using localized INRs as a more efficient means of reconstructing filtered representations if so desired, while still permitting random-access evaluation within the volume.



Figure 5.9: For grid-based SIRENs, we study the effect of the coarse grid resolution on learning a bilateral filter scale space on a 1024^3 CT scan of a flower (compression: 400x). As shown, a good compromise between quality (top-left) and training/inference time at a scale s = 7 (right) can be found (16^3), where "GT" corresponds to the time required to bilateral filter the original volume at this scale.



Figure 5.10: We study the effect of compression on learning a bilateral scalespace for two different INRs, namely, SIREN and Gabor (MFN), for the Lung dataset. In general we find that our method is insensitive to the level of compression, maintaining a comparable level of quality across the learned scale space. On the right we show that, qualitatively, the different levels of compression yield similar results (scale s = 3).



Figure 5.11: Our approach can distill a scale-based INR into a scale independent INR to achieve higher compression. We set an error tolerance of 10^{-5} for a bilateral filter scale-space Flower field (in range [0,1], and compare distillation between SIREN and Grid-SIREN (8³ grid). We observe both can roughly meet the tolerance (left), while we observe SIREN is far more compressive than Grid-SIREN (right).

5.4.4 Effect of Compression

In Fig. 5.10, we show the effect of different compression ratios on learning a bilateral scale-space for SIREN and Gabor (MFN) INR using a Medical CT scan image. We can see that with increasing compression ratio the ability of both the INRs to model the data accurately decreases. This is due to the noise that is inherently present in medical data and thus makes the learning task for the INRs with smaller capacity network harder. We find that, consistently across different compression ratios irrespective of the choice of INR, the PSNR is comparatively higher for larger scales. This suggests that the INRs are able to successfully reduce the noise present in the modeled data as a consequence of bilateral filtering. We further find that the ability of an INR to reproduce a scale-space is dependent on how well the INR models the data i.e., an INR that models the data poorly will yield a lower quality scale space as compared to an INR that models the data more accurately.

5.4.5 Network Distillation

In Fig. 5.11, we show the results of network distillation, applied to the Flower dataset (c.f. Fig. 5.9) under bilateral filtering. Here we prune weights until an error tolerance of 10^{-5} (PSNR 50) is met, and compare SIREN and SIREN-Grid to assess how compressible these models are. We note that both models start from INRs of a compression ratio of 400x, and consequently of lower PSNR, so it is only at scale s = 4 where the error tolerance can be satisfied. It is anticipated that both methods would have an error slightly above the tolerance (below the PSNR threshold). This is because we are using a "teacher" INR for supervision, leading to a small discrepancy, observed to be between 1 and 1.5 PSNR units. Both INR models achieve similar PSNR, but interestingly, we find that SIREN is far more compressive across all scales. This suggests that

network distillation for otherwise expensive SIRENs can effectively lead to reduced models.

The analysis presented in Sec. 5.2.2, for SIREN-based INRs, suggests that weights in deeper layers will largely tend to contribute mostly towards higher frequencies. Our network distillation method, wherein we weight-prune to meet a prescribed error tolerance, enables us to study this claim by counting the number of neurons retained, at each layer, after pruning has been achieved. Fig. 5.12 depicts these results. We observe a clear trend towards pruning weights in deeper layers as we increase scale. In this example we use a bilateral filter scale space, where we should expect some high frequencies to be removed, while others (e.g. sharp edges) should remain.

5.4.6 Persistence-based Scale Spaces

Finally, we demonstrate the results of our technique for learning a scale space based on persistence-based topological filtering. We begin with results on a 2D scalar field example, specifically, the vorticity of a velocity field from a fluid simulation, provided by the dataset of Jakob et al. (56). We sample persistence threshold values at equal spacing, producing a series of scalar fields on which our model is trained. We train on every other field in the sequence, and test on all scales. Fig. 5.13 presents the results, where we plot the PSNR, as well as Wasserstein distance (66) between the persistence diagrams resulting from our predicted scalar field at each scale, and the ground truth. For those scales at which our model is trained, we find that our method performs quite well along both measures. However, for withheld scales, we observe that occasionally our method may not capture the persistence-based features that are removed – we find this to be especially the case for persistence-based thresholds at which a significant number of features are removed. Persistence-based scale spaces are more challenging to capture than other nonlinear scale spaces (e.g. bilateral filtering), due to the highly nonuniform change in the field.

For persistence-based scale spaces wherein the features do not change so abruptly, we find that our method can generalize reasonably well. We demonstrate this for a 3D pressure-based scalar field of Isotropic Turbulence. In general, we observe that the PSNR remains consistently high throughout, while qualitatively one can observe that features removed – at withheld scales – are consistent between our method and ground truth. The Wasserstein distance can be somewhat high in this setting, but we emphasize that our method does not explicitly optimize for topology, only a mean-squared error to the filtered field. On the other hand, it should be possible to extend our method to support topology-based objectives during optimization, similar to Soler et al. (121), but we leave this for future work.















Neurons

Scale 8

3 4 Layer





Figure 5.12: For our network distillation results on the Flower dataset, and a SIREN-based INR, we show, for each scale-distilled model, the number of neurons retained at each layer after performing pruning. We find a strong tendency to remove neurons in deeper portions of the network.



Figure 5.13: We show the results of learning persistence based scale-space for the vorticity field of a fluid simulation dataset.



Figure 5.14: We show the results of persistence based filtering quantitatively through PSNR and Wasserstein distance between the original and the predicted field for different scales applied to the pressure field of the Isotropic Turbulence dataset. We show qualitative results of the fields generated from the ground truth (toprow) and INR predictions (bottom-row) for a subset of scales.

5.5 Discussion

We have presented a technique for endowing implicit neural representations with scale, enabling multiscale compressive analysis and visualization of field-based data. We have demonstrated the effectiveness of our method across a diverse set of data, INR models, and scale spaces, and improvements over existing compression-domain filtering methods. A key takeaway from our work is the specific way in which one should augment coordinate-based neural networks to learn scale. Naive approaches, e.g. tacking on scale as an input coordinate, clearly suffer in generalization (c.f. Fig. 5.6). Our technique of weight modulation leads to INRs that faithfully represent scale-space transformations, and we make this more precise in terms of the space of functions that INRs can represent.

We acknowledge several limitations with our current approach. First, the time required to optimize INRs remains quite expensive in the literature, even for methods that target larger-scale data (127; 84), and our method is no exception. The computation/inference timings reported in Fig. 5.9 are, generally, reflective of computational complexity across other models, as the time required to perform optimization is proportional to the data size, and the model size. Nevertheless, in practice we find that building scale into an existing INR converges significantly faster than the original fit to the data. We thus view our method as complementary to existing INR techniques that seek to optimize for large data (89), and we envision our method can be applied to future INR models as well.

In comparison to recent work that learns multi-scale INRs, namely BACON (77), our method necessitates access to a scale space on which to optimize. It would be more ideal to build the INR with the scale space, by design, rather than a post hoc process. However for scale spaces that do not have a clearly defined form known a priori, e.g. a frequency domain characterization, it is challenging to instrument an INR to represent a target scale space. We further acknowledge that our method has a bias towards learning continuous scale spaces, where fields change continuously in response to increasing scale. The characterization of functions represented by our method suggests this to be the case (e.g. frequency filtering, albeit in an extremely large frequency decomposition) – thus for scale spaces that vary in a discrete manner, e.g. topological persistence, our method may not capture such discrete phenomena. However, we view these as two distinct regimes, and thus different inductive biases within an INR: scale spaces that vary discretely, or continuously.

We see several avenues for future work. First, although our method can support progressive representations through varying scale, our results on distillation suggest that INRs can be purposed to have a progressive computational structure, in support of more efficient inference. As discussed in the Sec. 5.4.5, network pruning over increasing scales leads to a structure wherein the larger the scale, the more likely we can prune neurons in later layers. This observation suggests that 4the computation of an INR can be arranged to incrementally introduce neurons, in accordance with scale, e.g. computation with few neurons to give a coarse scale, with the introduction of additional neurons for finer scale. A naive approach, independent of a scale space, would lead to artifacts, where as our scale-based approach can potentially lead to a progressive representation where intermediate results take on properties of the scale space itself (e.g. few neurons used leads to smoothed results in the case of a Gaussian scale space).

We also plan on integrating our methods within neural rendering systems (89; 145) for interactive visual exploration of field-based data. Upon a proper port to a GPU-based rendering system, we believe that the methods we have introduced can open up a large number of possibilities for rendering systems that would otherwise be complex, or too expensive, to realize. For instance, the ability to access scalebased derivatives, directly from an INR would eliminate the need to perform 2 separate passes on a field for (1) filtering, and (2) numerically estimating derivatives. More broadly, we believe that our method has a number of benefits in the scale-oriented analysis of field-based data. We hypothesize that a continuous representation of scale can enable new methods for scale selection, e.g. exactly which scales highlight important features in field-based data, and where in the data. In this sense, we think that our method can support other methods that rely on scale-based analysis (19; 94; 134), but done in a manner that permits compressive analysis. Further, our specific approach to learning weight modulation is just one way to augment INRs. Like BACON (77), we believe that a better understanding of the representation space of INRs can permit new methods for designing models with built-in, theoretically-motivated, properties. We think this perspective can inform a large class of new machine learning models for representing field-based data, designed for visualization.

CHAPTER 6

Integration-free learning of Flow Maps

In the previous chapters, we focused on the data reduction aspect of flow visualization for different flow field specifications. We showed that substantial data reduction can be achieved through deep learning based techniques by incorporating what we visualize e.g. integral curves in the optimization process. Although data reduction is an important factor, another equally important factor essential for effective visual analysis is the computation time of visualization techniques. 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. To this end, we propose a technique for an alternative flow map representation that is efficient to evaluate and 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. 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.

6.1 Approach

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.

6.1.1 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 $v : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n$, where $v(\mathbf{x}(t),t)$ describes the vector of a particle at time *t* with spatial position $\mathbf{x}(t) \in \mathbb{R}^n$. The trajectory of a mass-free particle, advected under the influence of the flow field *v*, is governed by the following ordinary differential equation:

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{v}(\mathbf{x}(t), t), \quad \mathbf{x}(t_0) = \mathbf{x}_0, \tag{6.1}$$

where \mathbf{x}_0 represents the initial position of the particle at starting time t_0 . Integrating this differential equation under a specified time span τ gives us the flow map Φ , which varies in initial position \mathbf{x}_0 , starting time t_0 , and time span τ :

$$\Phi(\mathbf{x}_0, t_0, \tau) : \mathbb{R}^n \times \mathbb{R} \times \mathbb{R} \to \mathbb{R}^n = \mathbf{x}_0 + \int_{t_0}^{t_0 + \tau} \mathbf{v}(\mathbf{x}(t), t) dt.$$
(6.2)

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 (43; 107) 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 (43), or indirectly through integrating a learned vector field (107). 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:

$$\frac{\partial \Phi(\mathbf{x},t,\tau)}{\partial \tau} = \nu(\Phi(\mathbf{x},t,\tau),t+\tau).$$
(6.3)

In other words, the derivative of the flow map taken with respect to time span τ , 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 τ will yield a curve whose tangent vectors match the vector field as defined in Eq. (6.3).

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



Figure 6.1: Our approach is based on a criterion of self-consistency, where the flow map derivative under some time span τ should match the flow map's instantaneous velocity at this location. A flow map whose instantaneous velocity initially matches the vector field can give us a linear approximation (left), violating self-consistency. By optimizing over a range of time spans, our approach aims to adjust the flow map such that this criterion is satisfied (right).



Figure 6.2: Standard coordinate-based networks (a) serve as simple models for flow maps, but are difficult to control and optimize. In contrast, our proposed network design (b) permits a clear delineation between the instantaneous velocity of the flow map, and integration for nonzero time spans (τ), achieved via τ -scaled residual connections. This leads to a simplified derivative network (c) at $\tau = 0$, one that is straightforward to fit to the vector field, and subsequently, stabilize flow map optimization.

as Φ.

(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:

$$\hat{\Phi}(\mathbf{x},t,0) = \mathbf{x},\tag{6.4}$$

We argue that any approximation $\hat{\Phi}$ should *exactly* satisfy identity preservation. Otherwise, a small perturbation $\delta \in \mathbb{R}^n$ yielding $\hat{\Phi}(\mathbf{x}, t, 0) = \mathbf{x} + \delta$ 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 v at the provided position x and time t:

$$\frac{\partial \hat{\Phi}(\mathbf{x},t,0)}{\partial \tau} = \mathbf{v}(\mathbf{x},t). \tag{6.5}$$

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:

$$\hat{\Phi}(\mathbf{x},t,\varepsilon) \approx \hat{\Phi}(\mathbf{x},t,0) + \varepsilon v(\mathbf{x},t).$$
(6.6)

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:

$$l_{s}(\mathbf{x},t,\tau) = \left\| \frac{\partial \hat{\Phi}(\mathbf{x},t,\tau)}{\partial \tau} - \frac{\partial \hat{\Phi}(\hat{\Phi}(\mathbf{x},t,\tau),t+\tau,0)}{\partial \tau} \right\|.$$
(6.7)

Here we have replaced the vector field in Eq. (6.3) 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:

$$\mathscr{L}_{s} = \mathbb{E}_{(\mathbf{x},t)\in\mathscr{D}, \tau\in\mathscr{T}}\left[l_{s}(\mathbf{x},t,\tau)\right],\tag{6.8}$$

where \mathscr{D} is the spatiotemporal domain, and $\mathscr{T} = [\tau_{min}, \tau_{max}]$ 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, but it is possible for $\hat{\Phi}$ to minimize Eq. (6.8), while remaining a poor approximation of Φ . 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. (6.8).

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. 6.1 (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. 6.1 (right). This idea of incrementally building the flow map has precedence in the literature (50), 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.

6.1.2 A network design for flow maps

Coordinate-based neural networks, in particular ones based on sinusoidal positional encodings (120; 129; 29), are a natural choice for our flow map network design. Specifically, position (*n* dimensions), time (1 dimension), and time span (1 dimension) can collectively be treated as individual coordinates as input to a multi-layer perceptron (MLP) (120; 129), whose output corresponds to the flow map prediction, please see Fig. 6.2(a). 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 networks (120), and do not permit a distinction between instantaneous velocity (**P2**) and flow map derivatives of nonzero time span. This presents complications for ensuring a stable composition-based objective (c.f. Eq. (6.7)).

Rather than use a standard coordinate-based network we propose a 2-tiered network design, please see Fig. 6.2(b) for an overview. The first network, which we denote a $f_v : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^d$ 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_v with a given time span τ , via the following multiplicative scaling:

$$\mathbf{z}^{(0)} = \boldsymbol{\sigma}_{\boldsymbol{\nu}}(\boldsymbol{\tau}\mathbf{m}^{(0)}) \odot f_{\boldsymbol{\nu}}(\mathbf{x},t), \tag{6.9}$$

where $\mathbf{m}^{(0)} \in \mathbb{R}^d$ is a learnable vector aimed to reconcile the scaling of τ – initially expressed in terms of the physical domain – for the neural representation. The function σ_V is a nonlinearity that serves to squash values into a predetermined range, in practice this is set as a hyperbolic tangent, while \odot indicates element-wise multiplication. The second network, which we denote $f_\tau : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^d$, similarly 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_V and f_τ , through a residual connection:

$$\mathbf{z}^{(1)} = \mathbf{z}^{(0)} + \sigma_{\mathbf{v}} \left(\tau \mathbf{m}^{(1)} \right) \odot \sigma_{\tau} \left(\mathbf{z}^{(0)} \odot \left(W^{(1)} f_{\tau}(\mathbf{x}, t) \right) \right), \tag{6.10}$$

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. A consequence of the above construction is that the derivative w.r.t τ when $\tau = 0$ evaluates to

$$\frac{d\mathbf{z}^{(1)}}{d\tau} = \frac{d\mathbf{z}^{(0)}}{d\tau} = \left(\boldsymbol{\sigma}_{\nu}'\mathbf{m}^{(0)}\right) \odot f_{\nu}(\mathbf{x},t).$$
(6.11)

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

$$\mathbf{z}^{(l)} = \mathbf{z}^{(l-1)} + \sigma_{\nu} \left(\tau \mathbf{m}^{(l)} \right) \odot \sigma_{\tau} \left(W^{(l)} \mathbf{z}^{(l-1)} \right), \tag{6.12}$$

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:

$$\hat{\Phi}(\mathbf{x},t,\tau) = \mathbf{x} + W^{(L)} \mathbf{z}^{(L-1)}.$$
(6.13)

Returning to our properties, we note that this network design, by construction, satisfies the identity mapping (**P1**), so long as the chosen activation function σ_v satisfies $\sigma_v(0) = 0$. 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**):

$$\frac{d\hat{\Phi}(\mathbf{x},t,0)}{d\tau} = W^{(L)}(\mathbf{m}^{(0)} \odot f_{\mathbf{v}}(\mathbf{x},t)).$$
(6.14)

Please see the Sec. 6.2.7 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_{τ} , as depicted in Fig. 6.2(c); in fact it is entirely decoupled from the rest of the network (c.f. Eqs. (6.19) and (6.20)). Hence, we can directly optimize instantaneous velocity of the flow map for the vector field v using Eq. (6.14), 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_v , and $W^{(L)}$ when optimizing Eq. (6.8), 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. 6.2) its instantaneous velocity would amount to an involved derivative network (120) 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 (10; 43). In particular, the distinction between spatiotemporal coordinates and time span is considered by Han et al. (43), yet the ability to distinguish properties of the flow map for $\tau = 0$ time span is not studied. Our network design is inspired by Biloš et al. (10), 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_v and f_{τ} . Though in principle these could be arbitrary neural networks, in practice we adapt prior work on learnable feature grids (127; 145; 89), 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. (89). 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. (6.18)–(6.21)).

6.1.3 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:

$$\mathscr{L}_{\mathbf{v}} = \mathbb{E}_{(\mathbf{x},t)\in\mathscr{D}}\left[\|W^{(L)}(\mathbf{m}^{(0)} \odot f_{\mathbf{v}}(\mathbf{x},t)) - \mathbf{v}(\mathbf{x},t)\| \right].$$
(6.15)

This amounts to optimizing over the parameters of f_v , e.g. the multi-level feature grid, shallow MLP, vector $\mathbf{m}^{(0)}$, 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. 6.2(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. (6.7). 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 $\overline{\Phi}(\mathbf{x},t,\varepsilon) = (\widehat{\Phi}(\mathbf{x},t,\varepsilon),t+\varepsilon,\varepsilon)$ to be the resulting position, subsequent time step, and time span ε , from applying the flow map. Then for some target time span τ , we can compose the flow map into multiple steps $k \in \mathbb{Z}^+$ as follows:

$$\hat{\Phi}_k(\mathbf{x},t,\tau) = \underbrace{\bar{\Phi} \circ \bar{\Phi} \circ \cdots \circ}_{k-1} \bar{\Phi}(\mathbf{x},t,\frac{\tau}{k}).$$
(6.16)



Figure 6.3: We illustrate how flow map optimization incrementally learns longer time spans over the course of optimization. Initially (40 steps), the flow map provides us with a linear approximation, owing to its instantaneous velocity well-representing the vector field. Over optimization, the flow map becomes more accurate in its predictions for longer time spans.

We then redefine our self-consistency loss as:

$$l_{s}(\mathbf{x},t,\tau) = \left\| \frac{\partial \hat{\Phi}(\mathbf{x},t,\tau)}{\partial \tau} - \frac{\partial \hat{\Phi}(\hat{\Phi}_{k}(\mathbf{x},t,\tau),t+\tau,0)}{\partial \tau} \right\|.$$
(6.17)

Experimentally, we find that the number of steps k to take can be set in proportion to the time span. In particular, for a given time span τ if we let τ_g be this value's expression in grid units of the field's time domain, then we find it sufficient to set $k(\tau) = \lceil \sqrt{\tau_g} \rceil$. Given that our network architecture permits efficient evaluation, this is reasonably cheap to compute, especially in relation to full numerical integration. For further efficiency, we find that the right-hand side of Eq. (6.17) can be frozen during optimization, and hence we only optimize for the flow map derivative under nonzero time span τ .

During flow map optimization, we may freeze the network dedicated to instantaneous velocity, if not fine-tune it with a learning rate much smaller than the flow map portion of the network, e.g. approximately 2 orders of magnitude less. Moreover, τ -scaled residual connections ensure that the derivative network for nonzero τ is not overly complex, e.g. for small τ it will remain close to the derivative at $\tau = 0$. We find the ability of the flow map to well-represent small time spans (c.f. Eq. (6.6)) significantly helps stabilize optimization, and in practice, we find that the flow map incrementally improves on larger time spans over the course of optimization, please see Fig. 6.3 for an illustration.

6.2 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. 6.2.4), 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 6.1 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. (56), 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. (43). 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 (120) that tacks time span on as an additional coordinate, along with particle space-time coordinates (c.f. Fig. 6.2(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 (89; 145) 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. (72), 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. (143). Specifically, the SVF is first precomputed by estimating flow map derivatives, computed via RK4, and then at run-



Table 6.1: We list all datasets and their respective sizes used in experiments.

Figure 6.4: We show the quantitative evaluation of flow map approximation methods across different datasets, and across different time spans, beginning at start times for which flow features have largely resolved.

time 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.

6.2.1 Implementation details

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

Network architecture settings The design of f_v and f_τ rely on parameter settings related to the multilevel feature grid, as well as the MLP. The feature grids for f_v and f_τ 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 $w^{s \cdot l}$, with resolution scaling factor *s* set to 1.65, following the guidance of Müller et al. (89). 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_v and f_τ , respectively, along with activation σ_τ chosen to be a Swish activation (46). Experimentally we found Swish to outperform other more standard activations for INRs, e.g. ReLU, sin, consistent with findings in AutoInt (76). 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 resolution of the feature grids to best match a provided compression ratio, but leave the MLPs unchanged as they comprise a tiny portion of the model. Last, we use a 3-layer MLP with 64 layer width for the residual network. Unless otherwise specified, we use a compression ratio of 10 for all 2D datsets, and customize compression ratios for 3D as appropriate.

Dataset	FTLE res	τ	Inference	Preprocessing	CR	Storage	method
			time(s)	time(min)		(MB)	minud
Fluid Sim	512x512	7	21.161	-	-	2003	GT
			0.585	48.01	10	189	NIFM
			2.010	63.33	1		Siren
			4.701	1104.60	-		FSR
Cylinder	1200x150	1	1.853	-	-	153	GT
			0.055	33.50	10	16	NIFM
			0.554	74.26	-		ED
			0.324	41.76	1		Siren
			29.94	0.04	-		Spline
Boussinesq	450x1350	0.5	2.220	-	-	1030	GT
			0.079	37.25	10	97	NIFM
			0.938	122.89	-		ED
			0.621	63.28	1		Siren
			91.57	0.15	-		Spline
Double Gyre	1200x600	10	34.453	-	-	611	GT
			1.020	34.80	10	29	NIFM
			6.278	40.70	-		ED
			1.689	19.84	1		Siren
			252.63	0.29	-		Spline

Table 6.2: We report the preprocessing times for different methods across 2D unsteady flows, along with corresponding timings for FTLE computation, varying time span and image resolution.

Optimization details For both phases of optimization we use Adam (65), where we take a total of 40,000 optimization steps and decay the learning rate every 8,000 steps. Specific to optimization phase, in fitting to the vector field we use a learning rate of 0.02, while for flow map optimization we use a learning rate of 0.01 – fitting the flow map derivative to the vector field is quite stable, and benefits from larger learning rates. In optimizing for the flow map, we have the choice of leaving the instantaneous velocity portion of the network frozen, or fine-tuning its weights to compensate for the remainder of the network. Although we 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 τ_{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 τ_{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 τ . Specifically, for all 2D datasets, expressed in terms of grid units we set $\tau_{max} = 48$ unless otherwise specified. For 3D datasets we customize τ_{max} based on grid resolution, and complexity of the flows.

6.2.2 2D unsteady flow

We first conduct experimental comparisons for various 2D time-varying flow fields. Specifically, we evaluate different techniques by computing 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



Figure 6.5: We compare FTLE (top row) and integration error (bottom row) for two Fluid Simulation datasets (Re 16 and Re 101.6) across different baselines. The left column corresponds to particles integrated beginning at $t_0 = 0$ for duration $\tau = 7$, while the right column corresponds to particles integrated starting at $t_0 = 2$ and $\tau = 7$.



Figure 6.6: We compare FTLE (top row) and integration error (bottom row) for different baselines for the Double Gyre dataset. Particles are integrated from $t_0 = 0$ for a time-span $\tau = 10$.

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. 6.4 we present quantitative results comparing our method against different baselines, and in Table 6.2 we report inference and preprocessing times. Specifically, for the pathline interpolation approach of Li et al. (72), preprocessing refers to the time required to fit B-splines, while for Jakob et al. (56) 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 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. Qualititative results for the fluid simulation flows are shown in Fig. 6.5 in



Figure 6.7: We show the FTLE (top of each pair) and error maps (bottom of each pair) for the flow over cylinder dataset generated by integrating particles starting at $t_0 = 18$ for a time-span $\tau = 1$.

the form of the FTLE – computed using the method of Haller (?) – 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



Figure 6.8: We show the FTLE (top of each pair) and error maps (bottom of each pair) for the Boussinesq dataset generated by integrating particles starting at $t_0 = 11.3$ for a time-span $\tau = 0.5$.

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. 6.4) 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 architecture of Han et al. (43), or a pure coordinate-based approach (120) 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 net-



Figure 6.9: We compare our method's ability to compute streaklines against the streakline vector field technique (143), 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.

Table 6.3: 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 (143).

	Preprocessing	Inference	Storage	
Method	Time	Time		
	(min)	(sec)		
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	
NIFM (24 grid steps)	+0.000	0.671	77.20 MD	

work design. The inability to generalize in these methods is further demonstrated qualitatively for Figs. 6.6 - 6.8. Pathline interpolation (72) 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. 6.9 we show streaklines for the Cylinder dataset. We compare our method with SVF (143). 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^{-3} magnitude (relative to the bounding box diagonal). Interestingly, we find that both methods have comparable inference time as well, as reported in Table 6.3, 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 6.3 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.

Datacet	ETI E res	τ	Inference	Processing	CR	Method
Dataset	I ILL ICS		times (s)	times(m)	CK	
Tornado		50	27.16	-	-	GT
	128x128x128		3.60	35.55	10	NIFM
			14.14	93.21	10	SIREN
			286.29	0.87	-	Spline
Scalar Flow	100x178x100	2.5	81.72	-	-	GT
			2.55	41.66	10	NIFM
			21.48	95.57	10	SIREN
			291.39	0.81	-	Spline
Half-Cylinder		2	137.41	-	-	GT
	640x240x80		3.82	45.56	40	NIFM
			53.52	103.13	40	SIREN

Table 6.4: We report the processing times as well the FTLE computation times for different method across different 3D unsteady flow datasets.



Figure 6.10: We compare, both qualitatively (volume rendering of FTLE field) and quantitatively (flow map evaluation), our method with standard coordinate-based networks (120) as well as pathline interpolation techniques (72) 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.

6.2.3 3D unsteady flow

We next evaluate our method on a set of 3D unsteady flows, comparing our method with a SIREN-based flow map (120) as well as the B-spline pathline interpolation technique (72). We first compare to the Tornado and Scalar Flow datasets, where we set the τ_{max} to 8 and 24, respectively, to match the temporal complexity in the flows. Fig. 6.10 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 6.4. 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 (72) failed to fit to the data, and thus we limit our comparison to SIREN, please see Fig. 6.11. In this experiment we set $\tau_{max} = 8$ and the compression ratio to 40 to compensate for the larger data size. We find our method captures



Figure 6.11: 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.



Figure 6.12: We compare NIFM to Euler and RK4 integration schemes, showing FTLE (top), flow map error (middle), and quantitative evaluation (bottom). We find NIFM performs consistently well across step sizes as compared to Euler and RK4 which can become numerically unstable.

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 6.4) relative to the 2D unsteady flow datasets, whereas SIREN's increase in model size leads to slower training times.

6.2.4 Error analysis: numerical integration

Two key contributions of NIFM are (1) the reduction in computation time to compute flow maps, and (2) the reduction in storage requirements, relative to the original time-varying vector field. We study these factors in relation to existing numerical integration schemes, namely Euler and RK4 integration, for the 2D time-varying vector field Boussinesq (c.f. Table 6.1). We compare methods by varying the step size taken for integration, and report the computation time and accuracy of the approximation. To ensure fair comparisons in timing, all methods were implemented in PyTorch using CUDA support for fast grid access. Moreover, for fair comparison in data size, Euler and RK4 utilize the vector field subsampled by a factor of 2x in spatial and temporal resolution, and NIFM uses the same level of data reduction (e.g. $\frac{1}{8}$ the original data size). We take as ground truth RK4 integration applied to the original data at a step size set to $\frac{1}{10}$ the temporal resolution.

Fig. ?? shows the results, where we qualitatively compare the approximations using the FTLE (first row time span $\tau = 1.5$) and the flow map error (second row). Qualitatively 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. On the right, we show quantitative results for a time span of $\tau = 4$, 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.

6.2.5 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. 6.16) are detailed in the appendix.

In Fig. ?? 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

Grid steps



Figure 6.13: We study the effect of different schemes for taking steps in flow map optimization for the Boussinesq flow: full corresponds to taking one step per grid unit in time (and thus the most costly), sqrt corresponds to a square root scaling in (grid unit) time, log is a logarithmic scaling, while single corresponds to taking just a single step (thus the cheapest in computation). We find little difference between these schemes upon evaluation.



Figure 6.14: We further study different schemes for taking steps in flow map optimization, here for Double Gyre.

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 (16; 72), or build models (43) 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. 6.7) 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. **??** 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.

6.2.6 Ablation : Model

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 τ_{max} is set to 48 (in grid units), and we use default parameters as originally specified in the paper, unless otherwise mentioned.

Figs. 6.13 and 6.14 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. 6.15 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. 6.16 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. 6.17 we study the impact of τ_{max} on the performance of the model. We evaluate the model by taking the minimum between the integration duration and the τ_{max} the model was trained on as the step size. We found that when trained with large values of τ_{max} the overall 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. Lastly, in Fig. 6.19 and Fig. 6.20 we study the performance of different techniques across all integration durations. Additionally, we also observe that our method incurs the least amount of error and visual artifacts in the FTLE when integrated for a duration of $\tau = 20$.

Grid steps



Figure 6.15: We study the effect of model size on accuracy for the Boussinesq flow. We find a small drop in accuracy as we decrease the model size.



Figure 6.16: 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.



Figure 6.17: For the Boussinesq flow we compare the effects of τ_{max} on the overall performance of the model, finding that the self-consistency criterion when trained with large τ_{max} gracefully degrades in performance, indicating the stability of our optimization technique.



Figure 6.18: We show the performance of NIFM under large integration durations for the double gyre dataset. We show the FTLE and the corresponding flow map errors for $\tau = 20$ and $\tau = 30$. We observe the technique is able to perform reasonably well even for very large integration durations.



Figure 6.19: We show the performance of different techniques under large integration durations for the double gyre dataset. We show the FTLE and the corresponding flow map errors for $\tau = 20$. We observe that our model is able to outperform all the baselines and incurs the least error.

6.2.7 Flow Map Instantaneous Velocity

In this section 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 τ , and omit spatial position **x** and starting time *t* where necessary. The first layer



Figure 6.20: We evaluate different techniques under varying integration duration for the double gyre dataset.

produces the following:

$$\mathbf{z}^{(0)}(\tau) = \sigma_{\mathbf{v}}(\tau \mathbf{m}^{(0)}) \odot f_{\mathbf{v}}(\mathbf{x}, t).$$
(6.18)

The function $f_{v} : \mathbb{R}^{n} \times \mathbb{R} \to \mathbb{R}^{d}$ 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, σ_{v} 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:

$$\mathbf{z}^{(1)}(\tau) = \mathbf{z}^{(0)}(\tau)$$

$$+ \sigma_{\mathbf{v}}\left(\tau \mathbf{m}^{(1)}\right) \odot \sigma_{\tau}\left(\mathbf{z}^{(0)}(\tau) \odot \left(W^{(1)} f_{\tau}(\mathbf{x}, t)\right)\right),$$
(6.19)

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:

$$\mathbf{z}^{(l)}(\tau) = \mathbf{z}^{(l-1)}(\tau) + \sigma_{\nu}\left(\tau\mathbf{m}^{(l)}\right) \odot \sigma_{\tau}\left(W^{(l)}\mathbf{z}^{(l-1)}(\tau)\right),\tag{6.20}$$

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:

$$\hat{\Phi}(\mathbf{x},t,\tau) = \mathbf{x} + W^{(L)} \mathbf{z}^{(L-1)}(\tau).$$
(6.21)

We further make the following assumptions on activation functions σ_v and σ_τ :

$$\sigma_{\nu}(0) = 0$$
$$\sigma_{\nu}'(0) \neq 0$$
$$\sigma_{\tau}(0) = 0$$

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

$$\frac{d\hat{\Phi}(\mathbf{x},t,0)}{d\tau} = \frac{d\hat{\Phi}}{d\mathbf{z}^{(L-1)}} \frac{d\mathbf{z}^{(L-1)}(0)}{d\tau}.$$
(6.22)

The term on the left is simply:

$$\frac{d\hat{\Phi}}{d\mathbf{z}^{(L-1)}} = W^{(L)} \tag{6.23}$$

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

$$\frac{d\mathbf{z}^{(l)}}{d\tau} = \frac{d\mathbf{z}^{(l-1)}(0)}{d\tau}$$

$$+ \sigma'_{\nu}(0)\mathbf{m}^{(l)} \odot \sigma_{\tau} \left(W^{(l)}\mathbf{z}^{(l-1)}\right)$$

$$+ \sigma_{\nu}(0) \odot \frac{d\sigma_{\tau} \left(W^{(l)}\mathbf{z}^{(l-1)}\right)}{d\tau}.$$
(6.24)

The term in the second line will evaluate to zero, since activation vectors $\mathbf{z}^{(l)}(0)$, 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 $\sigma_{v}(0) = 0$. A similar reasoning can be applied for layer l = 1 (c.f. Eq. 6.19), due to the multiplicative scaling of $\mathbf{z}^{(0)}(0)$ with $(W^{(1)}f_{\tau}(\mathbf{x},t))$. Thus, we have the following:

$$\frac{d\mathbf{z}^{(L-1)}(0)}{d\tau} = \frac{d\mathbf{z}^{(0)}(0)}{d\tau} = \left(\sigma'_{\nu}(0)\mathbf{m}^{(0)}\right) \odot f_{\nu}(\mathbf{x},t).$$
(6.25)

As we choose σ_v to be a hyperbolic tangent function, we obtain $\sigma'_v(0) = 1$. Thus, plugging Eqs. 6.23 and 6.25 into Eq. 6.22, we arrive at:

$$\frac{d\hat{\Phi}(\mathbf{x},t,0)}{d\tau} = W^{(L)}(\mathbf{m}^{(0)} \odot f_{\nu}(\mathbf{x},t)).$$
(6.26)

Note that in Eq. 6.19, the multiplicative scaling within the activation σ_{τ} is essential for this result - a different way of combining the representations, e.g. adding them together, would introduce dependencies on f_{τ} , and weight matrices $W^{(l)}$, l > 0, in computing the derivative.

6.3 Discussion

In this work, 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 (145; 89). Additionally, our self-consistency scheme is only an approximation, whereas other approaches have studied the design of invertible neural networks for computing discrete (9) or continuous (17; 10) 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.

CHAPTER 7

Conclusion

This thesis addresses various challenges in flow visualization, with a particular focus on developing techniques that utilize neural representations to enhance the fidelity and efficiency of flow visualization workflows.

In Chapter 3, we introduced an integration-aware vector field super-resolution technique that incorporates streamlines into the optimization process. Our method has demonstrated improved performance over existing approaches through quantitative and qualitative evaluations on diverse datasets. We have also conducted a comprehensive study of essential factors related to streamline integration, such as seeding and streamline length, to investigate their impact on the upsampled flow field. We believe our approach is an important step towards incorporating visualization aspects of vector fields into the optimization process and opens the door for further exploration and investigation in this area.

In Chapter 4, we presented an approach for data reduction of unsteady flow fields by learning an Eulerian representation through Lagrangian optimization. Our technique eliminates the need for sampling the vector field onto a regular grid, providing a seamless and convenient method for post-hoc analysis. While we acknowledge that our method has limitations and there are areas where it may not outperform state-of-the-art techniques, we want to emphasize that our approach has shown substantial improvements in various aspects of flow visualization. These improvements have been demonstrated through extensive experimentation and rigorous evaluation.

In Chapter 5, we proposed a technique to incorporate scale into implicit neural representations (INRs), enabling multi-scale compressive analysis and visualization of field-based data. Our method surpasses existing compression-domain filtering methods and successfully captures scale-space transformations. We recognize the time-consuming optimization process for INRs and acknowledge the challenges of instrumenting scale spaces without prior knowledge. However, we believe that integrating scale into existing models opens up possibilities for rendering systems and scale-oriented analysis, facilitating new methods for scale selection and compressive analysis.

Finally, in Chapter 6, we presented an integration-free learning approach for flow maps using coordinatebased neural networks. Our technique leverages the properties of flow maps to enable efficient and accurate computation solely from the given vector field. Through extensive experimental comparisons, we have demonstrated the effectiveness of our method against various baselines and across multiple datasets.

While Implicit Neural Representations (INRs) are not the central focus of this dissertation, they play a
significant role in numerous chapters. Consequently, it is essential to discuss their robustness when dealing with noisy data. Broadly, INRs are generally robust to small amounts of random noise. However, the type and level of noise can impact learning and generalization capabilities. The over-parameterization and smooth function approximation of INRs provides an inherent denoising effect during training. This is evident from the results of modeling image gradients as demonstrated by Sitzmann et al. (120). However, we hypothesize that extremely high noise levels can overwhelm the network, preventing it from discerning the underlying signal and resulting in poor training. Thus, in such cases, the robustness can be improved by using the dropout mechanism to reduce reliance on individual inputs.

Looking ahead, the integration of neural representations within flow visualization frameworks presented in this dissertation opens up numerous avenues for future research and development. Although the proposed techniques show promising results, there is room for further enhancing the accuracy and efficiency of neural representations for flow visualization. Investigating other deep learning architectures, such as generative adversarial networks (GANs) or transformer models, could possibly result in more sophisticated and expressive representations. Additionally, exploring techniques for adaptive neural representations that can dynamically adjust the level of detail and fidelity based on the characteristics of the flow data would be valuable.

Uncertainty is inherent in many flow simulation datasets due to various sources of error and imprecision. Incorporating uncertainty estimation and propagation within neural representations can provide valuable insights for flow visualization. Future work can focus on developing techniques to quantify and visualize uncertainty in neural representations, enabling scientists and researchers to make informed decisions and interpretations based on the confidence levels associated with the flow data.

Interactive visualization plays a vital role in exploring and analyzing complex flow data. Expanding upon the proposed neural representations, future work can investigate the development of interactive visualization tools that leverage the power of these representations. This can involve techniques for real-time manipulation, exploration, and interaction with high-resolution flow fields, enabling researchers to gain immediate insights and facilitate hypothesis testing.

As numerical simulations continue to grow in complexity and scale, the management and analysis of large-scale flow datasets become increasingly challenging. Future research can focus on developing techniques for distributed neural representations that allow efficient storage, transmission, and processing of massive flow data across distributed computing systems. This could involve exploring parallel computing architectures, compression techniques, and scalable algorithms to handle datasets that are significantly larger in size than the ones considered in this dissertation.

The techniques proposed in this dissertation have the potential to be applied to a wide range of real-world flow problems, such as weather forecasting, aerodynamics, and fluid dynamics. Future work can focus on adapting and validating the proposed neural representations within these domains, addressing specific challenges and requirements unique to each application. This can involve collaborating with domain experts and conducting extensive empirical evaluations to demonstrate the effectiveness and practicality of the developed techniques.

Overall, through our research efforts, we have contributed novel techniques and insights to address the challenges of flow visualization. We are confident that our findings pave the way for improved data reduction, interactivity, and analysis of flow fields, and we look forward to the continued exploration of these ideas in the field.

References

- Agranovsky, A., Camp, D., Garth, C., Bethel, E. W., Joy, K. I., and Childs, H. (2014). Improved post hoc flow analysis via lagrangian representations. In 2014 IEEE 4th Symposium on Large Data Analysis and Visualization (LDAV), pages 67–75. IEEE.
- [2] Agranovsky, A., Obermaier, H., Garth, C., and Joy, K. I. (2015). A multi-resolution interpolation scheme for pathline based lagrangian flow representations. In *Visualization and Data Analysis 2015*, volume 9397, pages 221–232. SPIE.
- [3] Ahn, N., Kang, B., and Sohn, K.-A. (2018). Fast, accurate, and lightweight super-resolution with cascading residual network. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 252–268.
- [4] Austin, W., Ballard, G., and Kolda, T. G. (2016). Parallel tensor compression for large-scale scientific data. In 2016 IEEE international parallel and distributed processing symposium (IPDPS), pages 912–922. IEEE.
- [5] Bachlechner, T., Majumder, B. P., Mao, H. H., Cottrell, G. W., and McAuley, J. (2020). Rezero is all you need: Fast convergence at large depth. arXiv preprint arXiv:2003.04887.
- [6] Ballester-Ripoll, R., Lindstrom, P., and Pajarola, R. (2019). Thresh: Tensor compression for multidimensional visual data. *IEEE transactions on visualization and computer graphics*, 26(9):2891–2903.
- [7] Ballester-Ripoll, R., Steiner, D., and Pajarola, R. (2017). Multiresolution volume filtering in the tensor compressed domain. *IEEE transactions on visualization and computer graphics*, 24(10):2714–2727.
- [8] Barron, J. T., Mildenhall, B., Tancik, M., Hedman, P., Martin-Brualla, R., and Srinivasan, P. P. (2021). Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5855–5864.
- [9] Behrmann, J., Grathwohl, W., Chen, R. T., Duvenaud, D., and Jacobsen, J.-H. (2019). Invertible residual networks. In *International Conference on Machine Learning*, pages 573–582. PMLR.
- [10] Biloš, M., Sommer, J., Rangapuram, S. S., Januschowski, T., and Günnemann, S. (2021). Neural flows: Efficient alternative to neural odes. *Advances in Neural Information Processing Systems*, 34:21325–21337.
- [11] Brunton, S. L. and Rowley, C. W. (2010). Fast computation of finite-time lyapunov exponent fields for unsteady flows. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 20(1):017503.
- [12] Bujack, R. and Joy, K. I. (2015). Lagrangian representations of flow fields with parameter curves. In 2015 IEEE 5th Symposium on Large Data Analysis and Visualization (LDAV), pages 41–48. IEEE.
- [13] Cabral, B. and Leedom, L. C. (1993). Imaging vector fields using line integral convolution. In Proceedings of the 20th annual conference on Computer graphics and interactive techniques, pages 263–270.
- [14] Cai, R., Yang, G., Averbuch-Elor, H., Hao, Z., Belongie, S., Snavely, N., and Hariharan, B. (2020). Learning gradient fields for shape generation. In *European Conference on Computer Vision*, pages 364–381. Springer.
- [15] Chandler, J., Bujack, R., and Joy, K. I. (2016). Analysis of error in interpolation-based pathline tracing. In *EuroVis (Short Papers)*, pages 1–5.
- [16] Chandler, J., Obermaier, H., and Joy, K. I. (2014). Interpolation-based pathline tracing in particle-based flow visualization. *IEEE transactions on visualization and computer graphics*, 21(1):68–80.
- [17] Chen, R. T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. (2018). Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

- [18] Chen, Y., Liu, S., and Wang, X. (2021). Learning continuous image representation with local implicit image function. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8628–8638.
- [19] Correa, C. and Ma, K.-L. (2008). Size-based transfer functions: A new volume exploration technique. *IEEE transactions on visualization and computer graphics*, 14(6):1380–1387.
- [20] Dai, S., Han, M., Xu, W., Wu, Y., Gong, Y., and Katsaggelos, A. K. (2009). Softcuts: a soft edge smoothness prior for color image super-resolution. *IEEE transactions on image processing*, 18(5):969– 981.
- [21] Davies, T., Nowrouzezahrai, D., and Jacobson, A. (2020). On the effectiveness of weight-encoded neural implicit 3d shapes. arXiv preprint arXiv:2009.09808.
- [22] Di, S. and Cappello, F. (2016). Fast error-bounded lossy hpc data compression with sz. In 2016 ieee international parallel and distributed processing symposium (ipdps), pages 730–739. IEEE.
- [23] Dong, C., Loy, C. C., He, K., and Tang, X. (2015). Image super-resolution using deep convolutional networks. *IEEE transactions on pattern analysis and machine intelligence*, 38(2):295–307.
- [24] Dong, C., Loy, C. C., and Tang, X. (2016). Accelerating the super-resolution convolutional neural network. In *European conference on computer vision*, pages 391–407. Springer.
- [25] Duchon, C. E. (1979). Lanczos filtering in one and two dimensions. *Journal of Applied Meteorology and Climatology*, 18(8):1016–1022.
- [26] Eckert, M.-L., Um, K., and Thuerey, N. (2019). Scalarflow: a large-scale volumetric data set of realworld scalar transport flows for computer animation and machine learning. ACM Transactions on Graphics (TOG), 38(6):1–16.
- [27] Edelsbrunner, H. and Harer, J. L. (2022). Computational topology: an introduction. American Mathematical Society.
- [28] Erichson, N. B., Mathelin, L., Yao, Z., Brunton, S. L., Mahoney, M. W., and Kutz, J. N. (2020). Shallow neural networks for fluid flow reconstruction with limited sensors. *Proceedings of the Royal Society A*, 476(2238):20200097.
- [29] Fathony, R., Sahu, A. K., Willmott, D., and Kolter, J. Z. (2021). Multiplicative filter networks. In International Conference on Learning Representations.
- [30] Fuhrmann, A. and Groller, E. (1998). Real-time techniques for 3d flow visualization. In Proceedings Visualization'98 (Cat. No. 98CB36276), pages 305–312. IEEE.
- [31] Garth, C., Gerhardt, F., Tricoche, X., and Hans, H. (2007). Efficient computation and visualization of coherent structures in fluid flow applications. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1464–1471.
- [32] Gu, P., Han, J., Chen, D. Z., and Wang, C. (2021). Reconstructing unsteady flow data from representative streamlines via diffusion and deep-learning-based denoising. *IEEE Computer Graphics and Applications*, 41(6):111–121.
- [33] Günther, D., Jacobson, A., Reininghaus, J., Seidel, H.-P., Sorkine-Hornung, O., and Weinkauf, T. (2014). Fast and memory-efficienty topological denoising of 2d and 3d scalar fields. *IEEE transactions* on visualization and computer graphics, 20(12):2585–2594.
- [34] Günther, T., Gross, M., and Theisel, H. (2017a). Generic objective vortices for flow visualization. ACM Transactions on Graphics (Proc. SIGGRAPH), 36(4):141:1–141:11.
- [35] Günther, T., Gross, M., and Theisel, H. (2017b). Generic objective vortices for flow visualization. ACM Transactions on Graphics (TOG), 36(4):1–11.

- [36] Guo, L., Ye, S., Han, J., Zheng, H., Gao, H., Chen, D. Z., Wang, J.-X., and Wang, C. (2020). Ssr-vfd: Spatial super-resolution for vector field data analysis and visualization. In 2020 IEEE Pacific Visualization Symposium (PacificVis), pages 71–80. IEEE Computer Society.
- [37] Guthe, S., Wand, M., Gonser, J., and Straßer, W. (2002). Interactive rendering of large volume data sets. In *IEEE Visualization*, 2002. VIS 2002., pages 53–60. IEEE.
- [38] Haller, G. (2000). Finding finite-time invariant manifolds in two-dimensional velocity fields. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 10(1):99–108.
- [39] Haller, G. and Yuan, G. (2000). Lagrangian coherent structures and mixing in two-dimensional turbulence. *Physica D: Nonlinear Phenomena*, 147(3-4):352–370.
- [40] Hamarneh, G. and Hradsky, J. (2007). Bilateral filtering of diffusion tensor magnetic resonance images. *IEEE Transactions on Image Processing*, 16(10):2463–2475.
- [41] Han, J., Tao, J., Zheng, H., Guo, H., Chen, D. Z., and Wang, C. (2019). Flow field reduction via reconstructing vector data from 3-d streamlines using deep learning. *IEEE computer graphics and applications*, 39(4):54–67.
- [42] Han, J. and Wang, C. (2020). Ssr-tvd: Spatial super-resolution for time-varying data analysis and visualization. *IEEE Transactions on Visualization and Computer Graphics*.
- [43] Han, M., Sane, S., and Johnson, C. R. (2021a). Exploratory lagrangian-based particle tracing using deep learning. arXiv preprint arXiv:2110.08338.
- [44] Han, W., Chang, S., Liu, D., Yu, M., Witbrock, M., and Huang, T. S. (2018). Image super-resolution via dual-state recurrent networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1654–1663.
- [45] Han, Z., Ding, Z., Ma, Y., Gu, Y., and Tresp, V. (2021b). Temporal knowledge graph forecasting with neural ode. arXiv preprint arXiv:2101.05151.
- [46] Hayou, S., Doucet, A., and Rousseau, J. (2018). On the selection of initialization and activation function for deep neural networks. arXiv preprint arXiv:1805.08266.
- [47] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 770–778.
- [48] Hertz, A., Perel, O., Giryes, R., Sorkine-Hornung, O., and Cohen-Or, D. (2021). Sape: Spatiallyadaptive progressive encoding for neural optimization. *Advances in Neural Information Processing Systems*, 34:8820–8832.
- [49] Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the knowledge in a neural network. *arXiv* preprint arXiv:1503.02531.
- [50] Hlawatsch, M., Sadlo, F., and Weiskopf, D. (2010). Hierarchical line integration. *IEEE transactions on visualization and computer graphics*, 17(8):1148–1163.
- [51] Hoang, D., Klacansky, P., Bhatia, H., Bremer, P.-T., Lindstrom, P., and Pascucci, V. (2018). A study of the trade-off between reducing precision and reducing resolution for data analysis and visualization. *IEEE transactions on visualization and computer graphics*, 25(1):1193–1203.
- [52] Hoang, D., Summa, B., Bhatia, H., Lindstrom, P., Klacansky, P., Usher, W., Bremer, P.-T., and Pascucci, V. (2020). Efficient and flexible hierarchical data layouts for a unified encoding of scalar field precision and resolution. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):603–613.
- [53] Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. (2017). Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700– 4708.

- [54] Huang, Z., Bai, S., and Kolter, J. Z. (2021). (\textrm {Implicit})^ 2: Implicitlayers for implicit representations. Advances in Neural Information Processing Systems, 34 : 9639 -9650.
- [55] Hummel, M., Bujack, R., Joy, K. I., and Garth, C. (2016). Error estimates for lagrangian flow field representations. In *EuroVis (Short Papers)*, pages 7–11.
- [56] Jakob, J., Gross, M., and Günther, T. (2020). 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):1279– 1289.
- [57] Jang, Y., Ebert, D. S., and Gaither, K. (2011). Time-varying data visualization using functional representations. *IEEE Transactions on Visualization and Computer Graphics*, 18(3):421–433.
- [58] Jerman, T., Pernuš, F., Likar, B., and Špiclin, Ž. (2015). Blob enhancement and visualization for improved intracranial aneurysm detection. *IEEE Transactions on Visualization and Computer Graphics*, 22(6):1705– 1717.
- [59] Johnson, J., Alahi, A., and Fei-Fei, L. (2016). Perceptual losses for real-time style transfer and superresolution. In *European conference on computer vision*, pages 694–711. Springer.
- [60] Kasten, J., Petz, C., Hotz, I., Noack, B. R., and Hege, H.-C. (2009). Localized finite-time lyapunov exponent for unsteady flow analysis. In VMV, pages 265–276.
- [61] Keys, R. (1981). Cubic convolution interpolation for digital image processing. *IEEE transactions on acoustics, speech, and signal processing*, 29(6):1153–1160.
- [62] Khan, F., Roy, L., Zhang, E., Qu, B., Hung, S.-H., Yeh, H., Laramee, R. S., and Zhang, Y. (2019). Multiscale topological analysis of asymmetric tensor fields on surfaces. *IEEE transactions on visualization and computer graphics*, 26(1):270–279.
- [63] Kim, J., Kwon Lee, J., and Mu Lee, K. (2016). Deeply-recursive convolutional network for image superresolution. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1637– 1645.
- [64] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.
- [65] Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In ICLR (Poster).
- [66] Lacombe, T., Cuturi, M., and Oudot, S. (2018). Large scale computation of means and clusters for persistence diagrams using optimal transport. *Advances in Neural Information Processing Systems*, 31.
- [67] Lai, W.-S., Huang, J.-B., Ahuja, N., and Yang, M.-H. (2017). Deep laplacian pyramid networks for fast and accurate super-resolution. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 624–632.
- [68] Lakshminarasimhan, S., Shah, N., Ethier, S., Klasky, S., Latham, R., Ross, R., and Samatova, N. F. (2011). Compressing the incompressible with isabela: In-situ reduction of spatio-temporal data. In *European Conference on Parallel Processing*, pages 366–379. Springer.
- [69] Laramee, R. S., Hauser, H., Zhao, L., and Post, F. H. (2007). Topology-based flow visualization, the state of the art. *Topology-based methods in visualization*, pages 1–19.
- [70] LeCun, Y., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W., and Jackel, L. (1989). Handwritten digit recognition with a back-propagation network. *Advances in neural information processing sys*tems, 2.
- [71] Ledig, C., Theis, L., Huszár, F., Caballero, J., Cunningham, A., Acosta, A., Aitken, A., Tejani, A., Totz, J., Wang, Z., et al. (2017). Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4681–4690.

- [72] Li, H., Xiong, T., and Shen, H.-W. (2022). Efficient interpolation-based pathline tracing with b-spline curves in particle dataset. *arXiv preprint arXiv:2207.07224*.
- [73] Li, L., Hurault, S., and Solomon, J. (2023). Self-consistent velocity matching of probability flows. arXiv preprint arXiv:2301.13737.
- [74] Li, Y., Perlman, E., Wan, M., Yang, Y., Meneveau, C., Burns, R., Chen, S., Szalay, A., and Eyink, G. (2008). A public turbulence database cluster and applications to study lagrangian evolution of velocity increments in turbulence. *Journal of Turbulence*, (9):N31.
- [75] Lim, B., Son, S., Kim, H., Nah, S., and Mu Lee, K. (2017). Enhanced deep residual networks for single image super-resolution. In *Proceedings of the IEEE conference on computer vision and pattern recognition* workshops, pages 136–144.
- [76] Lindell, D. B., Martel, J. N., and Wetzstein, G. (2021). Autoint: Automatic integration for fast neural volume rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14556–14565.
- [77] Lindell, D. B., Van Veen, D., Park, J. J., and Wetzstein, G. (2022). Bacon: Band-limited coordinate networks for multiscale scene representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16252–16262.
- [78] Lipinski, D. and Mohseni, K. (2010). A ridge tracking algorithm and error estimate for efficient computation of lagrangian coherent structures. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 20(1):017504.
- [79] Liu, G.-H., Chen, T., and Theodorou, E. (2021). Second-order neural ode optimizer. *Advances in Neural Information Processing Systems*, 34:25267–25279.
- [80] Lu, Y., Jiang, K., Levine, J. A., and Berger, M. (2021a). Compressive neural representations of volumetric scalar fields. In *Computer Graphics Forum*, volume 40, pages 135–146. Wiley Online Library.
- [81] Lu, Y., Jiang, K., Levine, J. A., and Berger, M. (2021b). Compressive neural representations of volumetric scalar fields. *Comput. Graph. Forum*, 40(3):135–146.
- [82] Ma, K.-L. (2003). Visualizing time-varying volume data. Computing in Science & Engineering, 5(2):34– 42.
- [83] Marquina, A. and Osher, S. J. (2008). Image super-resolution by tv-regularization and bregman iteration. *Journal of Scientific Computing*, 37(3):367–382.
- [84] Martel, J. N. P., Lindell, D. B., Lin, C. Z., Chan, E. R., Monteiro, M., and Wetzstein, G. (2021). Acorn: Adaptive coordinate networks for neural scene representation. *ACM Trans. Graph. (SIGGRAPH)*, 40(4).
- [85] McLoughlin, T., Laramee, R. S., Peikert, R., Post, F. H., and Chen, M. (2010). Over two decades of integration-based, geometric flow visualization. In *Computer Graphics Forum*, volume 29, pages 1807–1829. Wiley Online Library.
- [86] Mehta, I., Gharbi, M., Barnes, C., Shechtman, E., Ramamoorthi, R., and Chandraker, M. (2021). Modulated periodic activations for generalizable local functional representations. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14214–14223.
- [87] Mildenhall, B., Srinivasan, P. P., Tancik, M., Barron, J. T., Ramamoorthi, R., and Ng, R. (2020). Nerf: Representing scenes as neural radiance fields for view synthesis. In *European conference on computer vision*, pages 405–421. Springer.
- [88] Molchanov, P., Mallya, A., Tyree, S., Frosio, I., and Kautz, J. (2019). Importance estimation for neural network pruning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11264–11272.

- [89] Müller, T., Evans, A., Schied, C., and Keller, A. (2022). Instant neural graphics primitives with a multiresolution hash encoding. arXiv preprint arXiv:2201.05989.
- [90] Muraki, S. (1993). Volume data and wavelet transforms. *IEEE Computer Graphics and applications*, 13(4):50–56.
- [91] Nguyen, D. B., Monico, R. O., and Chen, G. (2020). A visualization framework for multi-scale coherent structures in taylor-couette turbulence. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):902–912.
- [92] Norcliffe, A., Bodnar, C., Day, B., Moss, J., and Liò, P. (2021). Neural ode processes. arXiv preprint arXiv:2103.12413.
- [93] Nsampi, N. E., Djeacoumar, A., Seidel, H.-P., Ritschel, T., and Leimkühler, T. (2023). Neural field convolutions by repeated differentiation. arXiv preprint arXiv:2304.01834.
- [94] Pálenik, J., Byška, J., Bruckner, S., and Hauser, H. (2019). Scale-space splatting: Reforming spacetime for cross-scale exploration of integral measures in molecular dynamics. *IEEE Transactions on Visualization* and Computer Graphics, 26(1):643–653.
- [95] Popinet, S. (2004). Free computational fluid dynamics. ClusterWorld, 2(6).
- [96] Portwood, G. D., Mitra, P. P., Ribeiro, M. D., Nguyen, T. M., Nadiga, B. T., Saenz, J. A., Chertkov, M., Garg, A., Anandkumar, A., Dengel, A., et al. (2019). Turbulence forecasting via neural ode. arXiv preprint arXiv:1911.05180.
- [97] Post, F. H. and Van Walsum, T. (1993). Fluid flow visualization. Springer.
- [98] Post, F. H., Vrolijk, B., Hauser, H., Laramee, R. S., and Doleisch, H. (2002). Feature extraction and visualisation of flow fields. In *Eurographics (State of the Art Reports)*.
- [99] Rapp, T., Peters, C., and Dachsbacher, C. (2019). Void-and-cluster sampling of large scattered data and trajectories. *IEEE transactions on visualization and computer graphics*, 26(1):780–789.
- [100] Rojo, I. B., Gross, M., and Günther, T. (2019). Accelerated monte carlo rendering of finite-time lyapunov exponents. *IEEE transactions on visualization and computer graphics*, 26(1):708–718.
- [101] Rojo, I. B. and Günther, T. (2019). Vector field topology of time-dependent flows in a steady reference frame. *IEEE transactions on visualization and computer graphics*, 26(1):280–290.
- [102] Rubanova, Y., Chen, R. T., and Duvenaud, D. K. (2019). Latent ordinary differential equations for irregularly-sampled time series. Advances in Neural Information Processing Systems, 32:5320–5330.
- [103] Sadlo, F. and Peikert, R. (2007). Efficient visualization of lagrangian coherent structures by filtered amr ridge extraction. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1456–1463.
- [104] Sadlo, F., Rigazzi, A., and Peikert, R. (2011). Time-dependent visualization of lagrangian coherent structures by grid advection. In *Topological Methods in Data Analysis and Visualization*, pages 151–165. Springer.
- [105] Sahoo, S. and Berger, M. (2021). Integration-Aware Vector Field Super Resolution. In Agus, M., Garth, C., and Kerren, A., editors, *EuroVis 2021 - Short Papers*. The Eurographics Association.
- [106] Sahoo, S. and Berger, M. (2022). Integration-free learning of flow maps. *arXiv preprint arXiv:2211.03192*.
- [107] Sahoo, S., Lu, Y., and Berger, M. (2022). Neural flow map reconstruction. In *Computer Graphics Forum*, volume 41, pages 391–402. Wiley Online Library.

- [108] Sajjadi, M. S., Scholkopf, B., and Hirsch, M. (2017). Enhancenet: Single image super-resolution through automated texture synthesis. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4491–4500.
- [109] Sane, S., Bujack, R., and Childs, H. (2018). Revisiting the evaluation of in situ lagrangian analysis. In EGPGV@ EuroVis, pages 63–67.
- [110] Sane, S., Bujack, R., Garth, C., and Childs, H. (2020). A survey of seed placement and streamline selection techniques. *STAR*, 39(3).
- [Sane et al.] Sane, S., Childs, H., and Bujack, R. An interpolation scheme for vdvp lagrangian basis flows.
- [112] Schneider, J. and Westermann, R. (2003). Compression domain volume rendering. In *IEEE Visualization*, 2003. VIS 2003., pages 293–300. IEEE.
- [113] Schweri, L., Foucher, S., Tang, J., Azevedo, V. C., Günther, T., and Solenthaler, B. (2022). A physicsaware neural network approach for flow data reconstruction from satellite observations. *New techniques for improving climate models, predictions and projections.*
- [114] Shadden, S. C., Lekien, F., and Marsden, J. E. (2005a). Definition and properties of lagrangian coherent structures from finite-time lyapunov exponents in two-dimensional aperiodic flows. *Physica D: Nonlinear Phenomena*, 212(3-4):271–304.
- [115] Shadden, S. C., Lekien, F., and Marsden, J. E. (2005b). Definition and properties of Lagrangian coherent structures from finite-time Lyapunov exponents in two-dimensional aperiodic flows. *Physica D: Nonlinear Phenomena*, 212(3-4):271–304.
- [116] Shi, W., Caballero, J., Huszár, F., Totz, J., Aitken, A. P., Bishop, R., Rueckert, D., and Wang, Z. (2016). Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1874–1883.
- [117] Shocher, A., Cohen, N., and Irani, M. (2018). "zero-shot" super-resolution using deep internal learning. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 3118–3126.
- [118] Sicat, R., Krüger, J., Möller, T., and Hadwiger, M. (2014). Sparse pdf volumes for consistent multiresolution volume rendering. *IEEE transactions on visualization and computer graphics*, 20(12):2417–2426.
- [119] Sitzmann, V., Chan, E. R., Tucker, R., Snavely, N., and Wetzstein, G. (2020a). Metasdf: Meta-learning signed distance functions. In *Proc. NeurIPS*.
- [120] Sitzmann, V., Martel, J., Bergman, A., Lindell, D., and Wetzstein, G. (2020b). Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33:7462–7473.
- [121] Soler, M., Plainchault, M., Conche, B., and Tierny, J. (2018). Topologically controlled lossy compression. In 2018 IEEE Pacific Visualization Symposium (PacificVis), pages 46–55. IEEE.
- [122] Stalling, D. and Hege, H.-C. (1995). Fast and resolution independent line integral convolution. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 249–256.
- [123] Sun, J., Xu, Z., and Shum, H.-Y. (2008). Image super-resolution using gradient profile prior. In 2008 *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE.
- [124] Suter, S. K., Guitian, J. A. I., Marton, F., Agus, M., Elsener, A., Zollikofer, C. P., Gopi, M., Gobbetti, E., and Pajarola, R. (2011). Interactive multiscale tensor reconstruction for multiresolution volume visualization. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2135–2143.
- [125] Tai, Y., Yang, J., and Liu, X. (2017a). Image super-resolution via deep recursive residual network. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 3147–3155.
- [126] Tai, Y., Yang, J., Liu, X., and Xu, C. (2017b). Memnet: A persistent memory network for image restoration. In *Proceedings of the IEEE international conference on computer vision*, pages 4539–4547.

- [127] Takikawa, T., Litalien, J., Yin, K., Kreis, K., Loop, C., Nowrouzezahrai, D., Jacobson, A., McGuire, M., and Fidler, S. (2021). Neural geometric level of detail: Real-time rendering with implicit 3d shapes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11358–11367.
- [128] Tancik, M., Mildenhall, B., Wang, T., Schmidt, D., Srinivasan, P. P., Barron, J. T., and Ng, R. (2021). Learned initializations for optimizing coordinate-based neural representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2846–2855.
- [129] Tancik, M., Srinivasan, P., Mildenhall, B., Fridovich-Keil, S., Raghavan, N., Singhal, U., Ramamoorthi, R., Barron, J., and Ng, R. (2020a). Fourier features let networks learn high frequency functions in low dimensional domains. *Advances in Neural Information Processing Systems*, 33:7537–7547.
- [130] Tancik, M., Srinivasan, P. P., Mildenhall, B., Fridovich-Keil, S., Raghavan, N., Singhal, U., Ramamoorthi, R., Barron, J. T., and Ng, R. (2020b). Fourier features let networks learn high frequency functions in low dimensional domains. *NeurIPS*.
- [131] Tao, Y., Lin, H., Bao, H., Dong, F., and Clapworthy, G. (2009). Structure-aware viewpoint selection for volume visualization. In 2009 IEEE Pacific Visualization Symposium, pages 193–200. IEEE.
- [132] Tewari, A., Thies, J., Mildenhall, B., Srinivasan, P., Tretschk, E., Yifan, W., Lassner, C., Sitzmann, V., Martin-Brualla, R., Lombardi, S., et al. (2022). Advances in neural rendering. In *Computer Graphics Forum*, volume 41, pages 703–735. Wiley Online Library.
- [133] Tierny, J., Favelier, G., Levine, J. A., Gueunet, C., and Michaux, M. (2017). The topology toolkit. *IEEE transactions on visualization and computer graphics*, 24(1):832–842.
- [134] Tkachev, G., Frey, S., and Ertl, T. (2019). Local prediction models for spatiotemporal volume visualization. *IEEE Transactions on Visualization and Computer Graphics*, 27(7):3091–3108.
- [135] Tomasi, C. and Manduchi, R. (1998). Bilateral filtering for gray and color images. In *Sixth international conference on computer vision (IEEE Cat. No. 98CH36271)*, pages 839–846. IEEE.
- [136] Tong, T., Li, G., Liu, X., and Gao, Q. (2017). Image super-resolution using dense skip connections. In Proceedings of the IEEE International Conference on Computer Vision, pages 4799–4807.
- [137] Verma, V., Lamb, A., Beckham, C., Najafi, A., Mitliagkas, I., Lopez-Paz, D., and Bengio, Y. (2019). Manifold mixup: Better representations by interpolating hidden states. In *International Conference on Machine Learning*, pages 6438–6447. PMLR.
- [138] Wang, C. and Shen, H.-W. (2006). Lod map-a visual interface for navigating multiresolution volume visualization. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1029–1036.
- [139] Wang, Z., Seidel, H.-P., and Weinkauf, T. (2015a). Hierarchical hashing for pattern search in 3d vector fields. In VMV, pages 41–48.
- [140] Wang, Z., Seidel, H.-P., and Weinkauf, T. (2015b). Multi-field pattern matching based on sparse feature sampling. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):807–816.
- [141] Wegenkittl, R. and Groller, E. (1997). Fast oriented line integral convolution for vector field visualization via the internet. In *Proceedings. Visualization*'97 (*Cat. No.* 97CB36155), pages 309–316. IEEE.
- [142] Weinkauf, T., Hege, H.-C., and Theisel, H. (2012). Advected tangent curves: A general scheme for characteristic curves of flow fields. In *Computer Graphics Forum*, volume 31, pages 825–834. Wiley Online Library.
- [143] Weinkauf, T. and Theisel, H. (2010). Streak lines as tangent curves of a derived vector field. IEEE Transactions on Visualization and Computer Graphics, 16(6):1225–1234.
- [144] Weiskopf, D. et al. (2007). GPU-based interactive visualization techniques. Springer.

- [145] Weiss, S., Hermüller, P., and Westermann, R. (2021). Fast neural representations for direct volume rendering. *arXiv preprint arXiv:2112.01579*.
- [146] Xian, W., Huang, J.-B., Kopf, J., and Kim, C. (2021). Space-time neural irradiance fields for freeviewpoint video. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9421–9431.
- [147] Xie, Y., Takikawa, T., Saito, S., Litany, O., Yan, S., Khan, N., Tombari, F., Tompkin, J., Sitzmann, V., and Sridhar, S. (2022). Neural fields in visual computing and beyond. In *Computer Graphics Forum*, volume 41, pages 641–676. Wiley Online Library.
- [148] Xu, D., Wang, P., Jiang, Y., Fan, Z., and Wang, Z. (2022). Signal processing for implicit neural representations. In Advances in Neural Information Processing Systems.
- [149] Xu, L., Lee, T.-Y., and Shen, H.-W. (2010). An information-theoretic framework for flow visualization. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1216–1224.
- [150] Yan, Q., Xu, Y., Yang, X., and Nguyen, T. Q. (2015). Single image superresolution based on gradient profile sharpness. *IEEE Transactions on Image Processing*, 24(10):3187–3202.
- [151] Yüce, G., Ortiz-Jiménez, G., Besbinar, B., and Frossard, P. (2022). A structured dictionary perspective on implicit neural representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 19228–19238.
- [152] Yuqi, W., Wu, Y., Shan, L., Jian, Z., Huiying, R., Tiechui, Y., and Menghai, K. (2021). Flow field reconstruction method based on array neural network. *The Aeronautical Journal*, 125(1283):223–243.
- [153] Zhang, Y., Tian, Y., Kong, Y., Zhong, B., and Fu, Y. (2018). Residual dense network for image superresolution. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2472– 2481.
- [154] Zhao, H., Gallo, O., Frosio, I., and Kautz, J. (2016). Loss functions for image restoration with neural networks. *IEEE Transactions on computational imaging*, 3(1):47–57.