Supplementary Material
Interactive Visualization of Time-Varying Flow Fields Using Particle Tracing Neural Networks

Mengjiao Han, Jixian Li, Sudhanshu Sane, Shubham Gupta, Bei Wang, Steve Petruzza, Chris R. Johnson

1 Overview of Supplementary Material

In Sec. 2, we offer additional details, including the equations used to simulate the Double Gyre and ABC datasets. Sec. 3 features a detailed table summarizing all the experimental setups used to identify the most effective model architecture. This section also includes additional evaluation results, such as using the mean closest point distance as an error metric to compare our deep-learning-based (DL) approach with the traditional barycentric coordinate (BC) approach, along with their corresponding visualizations. In addition, it demonstrates how the pathlines predicted by our DL approach effectively maintain temporal coherence. Furthermore, Sec. 4 provides an in-depth description of the functionalities incorporated in our web-based viewer and high-performance OSPRay-based viewer.

2 Details on Datasets: Simulation Equations

Double Gyre

$$\begin{aligned} \psi(x,y,t) &= A sin(\pi f(x,t)) sin(\pi y) \\ f(x,t) &= a(t)x^2 + b(t)x \\ a(t) &= \varepsilon sin(\omega t) \\ b(t) &= 1 - 2\varepsilon sin(\omega t) \\ \text{where } A &= 0.1,\ \omega = pi/5 \text{ and } \varepsilon = 0.25 \end{aligned} \tag{1}$$

ABC

$$\begin{aligned} f(x,t) &= A(t)sin(z) + Bcos(y) \\ f(y,t) &= Bsin(x) + Ccos(z) \\ f(z,t) &= Csin(y) + A(t)cos(x) \\ A(t) &= \sqrt{3} + 0.5t sin(\pi t) \\ \text{where } B &= \sqrt{2},\ C = 1 \end{aligned} \tag{2}$$

- *Mengjiao Han is with SCI Institute, University of Utah. E-mail: mengjiao@sci.utah.edu*
- *Jixian Li is with SCI Institute, University of Utah. E-mail: jixianli@sci.utah.edu*
- *Sudhanshu Sane is with SCI Institute, University of Utah. E-mail: ssane@sci.utah.edu.*
- *Shubham Gupta is with Utah State University. E-mail: shubhamg2404@gmail.com.*
- *Bei Wang is with SCI Institute, University of Utah. E-mail: beiwang@sci.utah.edu*
- *Steve Petruzza is with Utah State University. E-mail: steve.petruzza@usu.edu.*
- *Chris R. Johnson is with SCI Institute, University of Utah. E-mail: crj@sci.utah.edu*

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

3 More Details on Experiments and Results

3.1 Impact of Model Architecture

Sec. 3.1 displays the mean and median errors for each combination derived from our selection of the number of encoding layers, decoding layers, and the size of the hidden latent vector.

Our experiments reveal that the mean and median errors are similar when using four or six encoding or decoding layers. Additionally, models based on Multi-Layer Perceptron (MLP) generally show enhanced performance with shallower neural network architectures. Furthermore, in most instances, a larger dimension of the hidden latent vector is more effective than a smaller one. As presented in Sec. 3.1, the maximum error values are substantially higher than the median errors. This result is especially evident in scenarios involving hyperbolic Lagrangian Coherent Structures (LCS), where the errors are observed to increase with increasing flow turbulence. However, as illustrated in Fig. 1, our deep-learning-based approach consistently yields smaller maximum errors compared to the traditional interpolation method. Enhancing the model's capability to manage hyperbolic LCS effectively will be studied in future work. This advancement is key to significantly expanding the applicability and effectiveness of deep-learning-based approaches in this field.

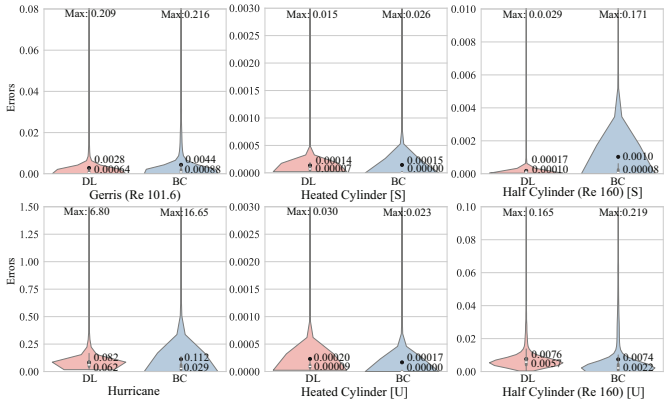


Fig. 1: The violin plot compares error rates between our deep learning-based method (DL) and the conventional barycentric coordinate interpolation (BC) method, using the mean-closest-point metric for error assessment. These plots are constructed from 5000 trajectory data points. In the plot, 'S' symbolizes structured data, whereas 'U' signifies unstructured data. The median of each distribution is shown by a white line on a gray background, and the mean is represented as a black dot. Each violin plot incorporates error limits to illustrate the range of errors, with the highest error value marked at the top of each plot. Our findings indicate that the DL method outperforms the BC method across all structured datasets with dense seedings and yields comparable outcomes for unstructured datasets with sparse seedings.

	Double Gyre								Gerris (Re 101.6)								Gerris (Re 445.7)							
	1024				2048				1024				2048				1024				2048			
[#E, #D]	Max	Max($< 3std$)	Mean	Med	Max	Max($< 3std$)	Mean	Med	Max	Max($< 3std$)	Mean	Med	Max	Max($< 3std$)	Mean	Med	Max	Max($< 3std$)	Mean	Med	Max	Max($< 3std$)	Mean	Med
[4, 4]	0.015040	0.002599	0.001198	0.001138	0.035495	0.002901	0.000360	0.000252	0.070006	0.008243	0.000864	0.000527	0.143171	0.012130	0.001125	0.000586	0.355923	0.066951	0.005818	0.001238	0.397267	0.071722	0.005599	0.000854
[4, 6]	0.022072	0.002170	0.000733	0.000664	0.041218	0.002686	0.000439	0.000339	0.097808	0.009787	0.000871	0.000521	0.082728	0.006718	0.000671	0.000410	0.360701	0.066855	0.005553	0.001140	0.401839	0.074412	0.005570	0.000829
[4, 8]	0.036979	0.002901	0.000508	0.000402	0.025446	0.001761	0.000146	0.000105	0.119044	0.010082	0.000909	0.000549	0.061141	0.005441	0.000659	0.000417	0.390928	0.069549	0.005615	0.001096	0.421647	0.071151	0.005549	0.001021
[6, 4]	0.004693	0.000351	0.000092	0.000082	0.007374	0.000554	0.000087	0.000075	0.075854	0.004795	0.000613	0.000413	0.153836	0.013044	0.001148	0.000589	0.394845	0.065593	0.005251	0.001063	0.410827	0.081046	0.006641	0.001028
[6, 6]	0.006525	0.000440	0.000097	0.000087	0.045319	0.003081	0.000496	0.000372	0.149521	0.010705	0.000987	0.000577	0.145615	0.012662	0.001156	0.000594	0.384426	0.064527	0.004981	0.000986	0.487556	0.085583	0.006971	0.001083
[6, 8]	0.032895	0.005353	0.001667	0.001477	0.046326	0.003171	0.000415	0.000266	0.136674	0.010987	0.000954	0.000574	0.103415	0.007892	0.000704	0.000421	0.389712	0.071249	0.005765	0.001128	0.403145	0.072066	0.005259	0.000785
[8, 4]	0.015751	0.010245	0.006276	0.006395	0.039029	0.003122	0.000689	0.000576	0.156254	0.012222	0.001410	0.000778	0.162986	0.013428	0.001265	0.000658	0.387796	0.078124	0.007043	0.001503	0.438808	0.085449	0.007408	0.001118
[8, 6]	0.032474	0.003925	0.001833	0.001812	0.078831	0.005179	0.000594	0.000417	0.137461	0.013975	0.001695	0.000885	0.153754	0.014227	0.001249	0.000643	0.405358	0.078671	0.008247	0.002640	0.407937	0.085184	0.006938	0.001134
[8, 8]	0.040973	0.005288	0.002546	0.002483	0.042387	0.003768	0.000879	0.000749	0.133435	0.012930	0.001616	0.000804	0.160815	0.014334	0.001225	0.000625	0.403534	0.078814	0.009005	0.001592	0.434742	0.084257	0.006894	0.001323

	ABC								Half Cylinder (Re 160)								Half Cylinder (Re 320)							
	1024				2048				1024				2048				1024				2048			
[#E, #D]	Max	Max($< 3std$)	Mean	Med	Max	Max($< 3std$)	Mean	Med	Max	Max($< 3std$)	Mean	Med	Max	Max($< 3std$)	Mean	Med	Max	Max($< 3std$)	Mean	Med	Max	Max($< 3std$)	Mean	Med
[4, 4]	0.036861	0.009468	0.001521	0.000412	0.099295	0.012862	0.003533	0.002726	0.107393	0.006192	0.000255	0.000111	0.070775	0.003762	0.000220	0.000124	0.199564	0.011533	0.000466	0.000154	0.201851	0.016832	0.000447	0.000115
[4, 6]	0.030883	0.007899	0.002667	0.002176	0.055693	0.012862	0.002835	0.001946	0.105311	0.005334	0.000265	0.000126	0.066850	0.003313	0.000208	0.000125	0.202506	0.014704	0.000487	0.000173	0.203822	0.012574	0.000450	0.000120
[4, 8]	0.081320	0.015790	0.004257	0.003406	0.045269	0.011415	0.002475	0.001914	0.082466	0.005549	0.000319	0.000186	0.069271	0.004303	0.000328	0.000228	0.212825	0.016497	0.000517	0.000200	0.205328	0.015798	0.000478	0.000152
[6, 4]	1.011904	0.069674	0.017979	0.015364	0.071155	0.018524	0.005582	0.004405	0.082571	0.005148	0.000287	0.000151	0.069917	0.003684	0.000209	0.000103	0.206473	0.016362	0.000495	0.000134	0.203245	0.015065	0.000478	0.000129
[6, 6]	0.235489	0.039737	0.012636	0.010582	0.104499	0.022884	0.005838	0.004300	0.086557	0.005517	0.000302	0.000170	0.072497	0.003869	0.000210	0.000105	0.205837	0.015488	0.000482	0.000187	0.201469	0.014333	0.000492	0.000159
[6, 8]	0.105150	0.024111	0.007958	0.006616	0.055382	0.013191	0.003861	0.003017	0.082592	0.005871	0.000342	0.000186	0.066915	0.004282	0.000244	0.000144	0.219285	0.015605	0.000618	0.000301	0.194538	0.017186	0.000518	0.000187
[8, 4]	10.579720	9.741066	3.665045	3.383233	9.065552	2.504865	0.262102	0.079539	0.068850	0.004751	0.000362	0.000228	0.066891	0.003938	0.000304	0.000207	0.206363	0.013851	0.000622	0.000246	0.205913	0.015392	0.000566	0.000213
[8, 6]	11.840756	8.123253	2.027380	1.131475	9.641712	1.966346	0.152597	0.040992	0.082675	0.004370	0.000341	0.000238	0.071471	0.004025	0.000260	0.000156	0.206732	0.013048	0.000637	0.000315	0.197799	0.011822	0.000516	0.000186
[8, 8]	8.073314	7.739241	4.040486	4.051986	7.983617	7.612440	4.040674	4.059320	0.100450	0.006299	0.000480	0.000333	0.073517	0.004194	0.000301	0.000197	0.204938	0.015042	0.000721	0.000350	0.196439	0.011611	0.000532	0.000206

Table 1: The performance evaluation of the model architecture on 2D and 3D datasets. Each row ($[E, D]$) represents the number of encoding layers (E) and decoding layers (D). The hidden vector dimension is either 1024 or 2048. The table presents the maximum (Max), maximum of within three standard deviations from the mean (Max($< 3std$), mean (Mean), and median (Med) errors, aggregated over 5,000 seeds and computed along the trajectories (Eqn. 3 in the manuscript). Most models achieve accurate results with errors smaller than one grid size. The optimal model architecture providing the lowest mean and median errors is highlighted. Increasing the dimension of the hidden vector leads to lower errors for all data sets. However, utilizing more encoding layers than decoding layers results in higher errors in our experiments. The maximum errors indicate that the model's performance is impacted by the hyperbolic of the flows. However, the deep-learning-based approach yields smaller maximum errors than the traditional interpolation method (refer to Fig. 1.

3.2 Comparison with Interpolation Methods

Our study contrasts our deep learning methodology with the traditional barycentric coordinate interpolation method. The performance evaluation uses Euclidean distance, comparing the interpolated and predicted outputs to the actual ground truth. As suggested by Isabelle et al. [2], the mean distance of closest distances provides a global similarity measure integrated along the whole curve, which can be used to identify the similarity of trajectories. We assess the discrepancies between our deep-learning approach and the conventional method by applying the mean-closest-point metric (refer to Fig. 1). Our deep-learning-based method demonstrates equivalent or enhanced accuracy across all structured datasets with dense seeding. When applied to unstructured datasets, its performance is comparable to the traditional method. However, to attain greater accuracy in unstructured datasets, dense seeds is necessary for training.

Fig. 2 and Fig. 3 present a comparative analysis between trajectories computed by our method and the established ground truth, demonstrating a significant overlap. This overlap shows the accuracy of our prediction approach, surpassing traditional interpolation methods. Moreover, our method achieves this accuracy with minor visual artifacts, highlighting its effectiveness in precise trajectory computation.

3.3 Demonstration of Temporal Coherence

In our experiments, particle end locations are recorded at specific intervals. For instance, in the case of the Double Gyre, Gerris Flow, and ABC datasets, the end locations are saved at every fifth interval. The following approach is used to demonstrate the temporal coherence of our predicted trajectories. We train models using this five-interval approach, and then employ these trained models to make predictions at more frequent intervals, specifically at every single interval. This method allows us to represent the complete paths of the particles over time, providing a more detailed and continuous trajectory analysis.

Fig. 4 and Fig. 5 present both the visual comparison and error distribution when contrasting predicted pathlines with the ground truth. Our deep-learning-based method can predict end locations at untrained time steps accurately. This accuracy indicates the method's effectiveness in preserving the temporal coherence of the pathlines.

4 Details on Interactive Visualization

We use ONNX Runtime (ORT) [1] as the library to deploy our neural network. After the seeds are placed, the loaded model may trace and display the tracing results.

To render line primitives, however, the web-based renderer lacks visual effects such as transparency, global illumination, and ambient occlusion. To visualize high-fidelity results, we also develop and study the performance of a PC-based viewer using the OSPRay [6] rendering engine to support photorealistic effects (Sec. 4.2). Moreover, by altering the *Trace* function, users can easily apply neural networks with alternative architectures to both viewers.

4.1 The Web-Based Viewer

The web-based viewer is developed using the React framework [2] and implemented in JavaScript. The user interface elements of the application, including tabs, checkboxes, buttons, and selections, are created using Material-UI [3]. We utilize React-Grid-Layout [4] to design a responsive and flexible grid layout system. The renderings of seeds, trajectories, and interaction with rendered objects are managed by react-three-fiber [5] and its expanded libraries drei [6].

In the **Model Info** panel, users can begin by selecting the dataset for visualization (Sec. 4.1.1 and Panel 2 in Fig. 7). Upon model loading, the primary 3D display presents the bounding box, seed box, seeds, pathlines, and scalar field (Panel 1 from Fig. 7).

The bounding box, referred to as the *global domain*, confines seed placement and is depicted as a gray outline. Seeds reside within the global domain, whereas pathlines can extend beyond it, as demonstrated in Fig. 7.

The seed box, a subset of the global domain, offers users finer control over seed placement. It is manipulated using sliders in the **Seedbox Config** panel (Panel 3 from Fig. 6 and detailed in Sec. 4.1.2). Seeds appear as colored spheres, and pathlines as colored tubes, with styles adjustable through the **Line Style Config** panel (Panel 5 from Fig. 7 and discussed in Sec. 4.1.4).

The scalar field can be visualized as a volume or three axis-aligned slices, with rendering parameters modifiable in the **Scalars Config** and **Transfer Function** panels (Panel 4 from both Fig. 7 and Fig. 6, with further information in Sec. 4.1.3).

4.1.1 Model Conversion and Model Loading

Our approach employs the PyTorch framework for both model training and storage. We deploy the trained neural network in the browser using onnxruntime-web, which is a JavaScript API based on the ONNX Runtime (ORT)[7]. The onnxruntime-web accelerates model inference in

[1] https://onnxruntime.ai/

[2] https://reactjs.org

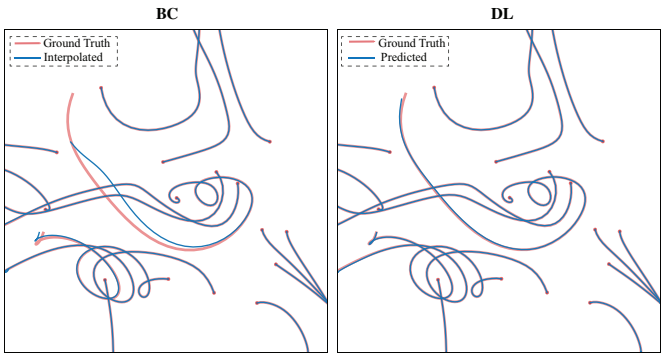
[3] https://mui.com

[4] https://github.com/react-grid-layout/react-grid-layout

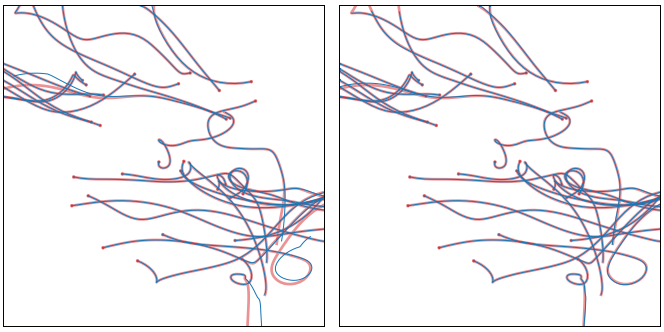
[5] https://github.com/pmndrs/react-three-fiber

[6] https://github.com/pmndrs/drei

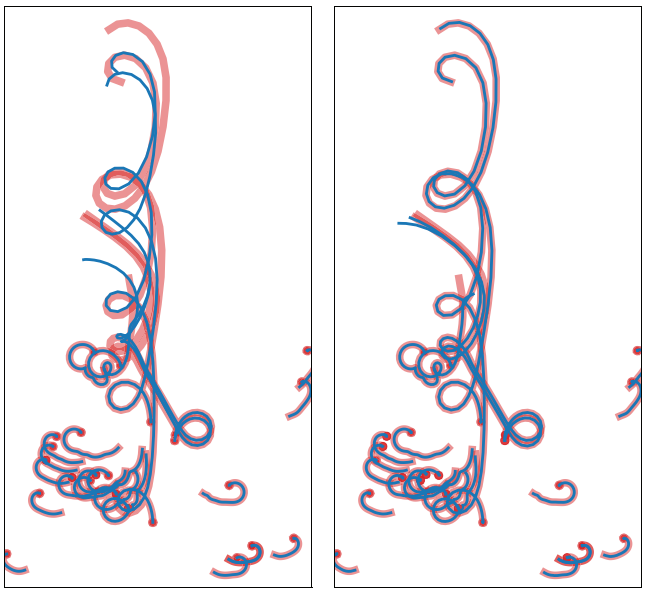
[7] https://onnxruntime.ai/



(a) Gerris (Re 101.6)



(b) Gerris (Re 445.7)



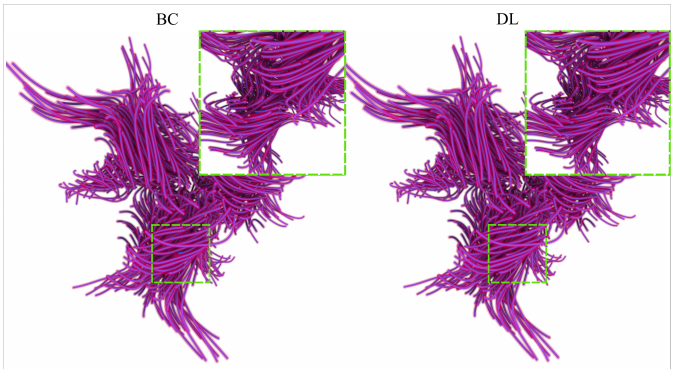
(c) Heated Cylinder

Fig. 2: Comparative visualization of trajectories. Left: results from barycentric coordinate interpolation (BC). Right: outcomes from our deep learning (DL) approach. The ground truth trajectories are marked in red for reference with the computed trajectories highlighted in blue. This figure demonstrates that our approach accurately traces pathlines compared to the traditional approach.

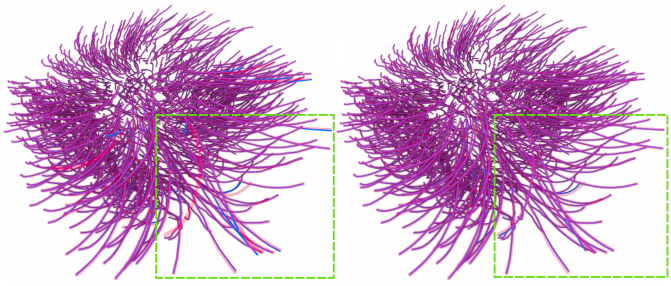
the browser by utilizing separate WebAssembly (Wasm)[8] and WebGL[9] backends on CPUs and GPUs. The ORT is compatible with popular deep learning frameworks, such as PyTorch [3], TensorFlow [1], and scikit-learn [4]. The conversion of a PyTorch model to an ONNX

[8]https://webassembly.org/

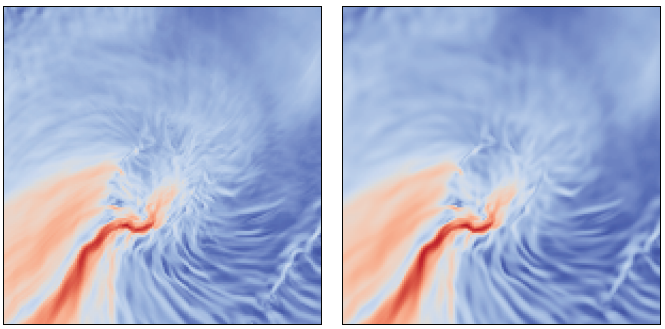
[9]https://www.khronos.org/webgl/



(a) ABC



(b) Hurricane

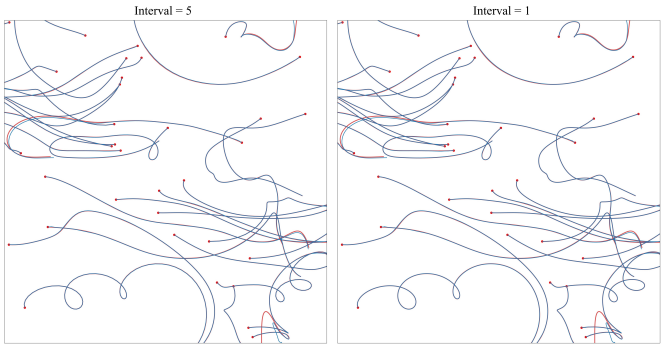


(c) Hurricane (FTLE)

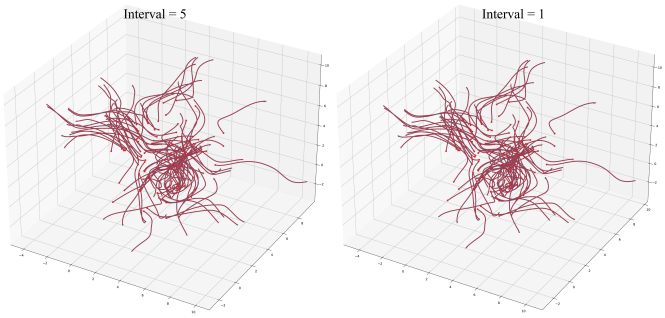
Fig. 3: (a) and (b) show the comparative visualization of trajectories rendered by OSPRay. This figure contrasts trajectory visualizations from barycentric coordinate interpolation (BC) on the left and deep learning (DL) on the right. Ground truth trajectories are marked in red for reference, and computed trajectories are highlighted in blue. In the ABC dataset, there is a notable overlap of computed trajectories with the ground truth in both methods. However, our approach yields more accurate trajectories for the Hurricane dataset. Moreover, (c) compares the ground truth FTLE on the left with the results computed using our method on the right. These visualizations underscore the proficiency of our method in accurately tracing pathlines.

model is facilitated by the *torch.onnx.export* method, and the size of the ONNX model is identical to that of the original PyTorch model. The conversion technique is available on our project's Github website.

We use a JSON file to store the necessary dataset information to facilitate data processing, including the bounding box, the directory containing the models, the start and stop file cycles, the interval, and the step size. When users select a dataset from the drop-down list, the viewer automatically loads the models by parsing the information in the corresponding JSON file. After successfully loading the model, the viewer displays the dataset's bounding box, and information about the bounding box and flow maps appears in the **Model Info** panel (Panel 2 in Fig. 6).

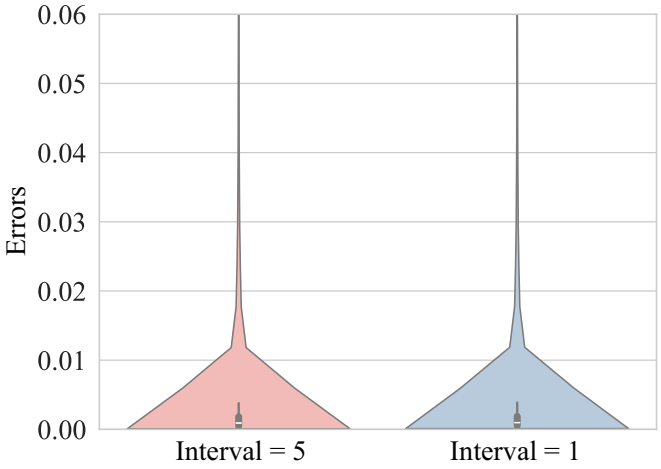


(a) Gerris (Re 445.7)

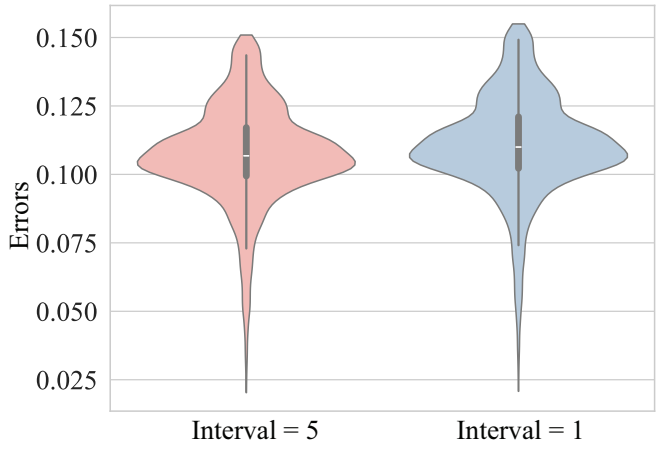


(b) ABC

Fig. 4: This visualization compares the predicted pathlines with the ground truth. We train our models using data at intervals of five and then apply these models for predictions at both five-interval (left) and one-interval (right) frequencies. The ability of our deep-learning-based approach to accurately predict pathlines at every time step demonstrate its effectiveness in preserving the temporal coherence of the pathlines.



(a) Gerris (Re 445.7)



(b) ABC

Fig. 5: The violin plots illustrate error comparisons between the predicted pathlines (marked as blue) and the ground truth (marked as red). Models are trained using data at five-interval frequencies and subsequently utilized to predict at both five-interval (left) and one-interval (right) frequencies. Our approach maintains similar accuracy levels in predicting end locations at untrained time steps, demonstrating its capability to preserve temporal coherence effectively.

4.1.2 Seeding Approaches

Our viewer allows users to add seeds in the global domain or a seed box to display trajectories in their desired regions. In the web-based viewer, the seed box is defined by size and position. Users can specify a seeding-plane or seeding-line by setting the sizes of one or two dimensions to zeros. The **Seedbox Config** panel contains two triggers: *Display* to control the seed box visibility, and *Active* to set the seed box area as the active seeding area (Panel 3 in Fig. 6). The seeds are scattered in the global domain by default if the seed box is not active. For seed placement, users can distribute new seeds uniformly, randomly, or manually add individual seeds in the **Seed Placement** panel (Panel 3 in Fig. 7). When the *Add Seeds* button clicks, the viewer will visualize the seeds in real time.

4.1.3 Visualization of Scalar Fields

Scalar fields such as the FTLE or velocity magnitude can provide more context for users to identify interesting regions to explore. In our viewer, scalar field data in VTI format can be uploaded through the **Scalars Config** panel (Panel 4 in Fig. 7). Once the data is uploaded, users can choose to render the volume or slices (Panel 3 in Fig. 7), and apply transfer functions with support for multiple colormaps (Panel 3 in Fig. 6). Visualizing scalar fields, in addition to the pathlines, helps users understand the spatial context of the flow features. It can also help user identify important features and patterns in the flow data.

4.1.4 Particle Tracing and Line Style

After placing seeds, users can initiate particle tracing by clicking the *Trace Particles* button located in the **Model Info** panel (Panel 2 in Fig. 7). Our viewer also provides users with the option to apply a constant color to the seeds and pathlines or to use the same color mapping specified by the color transfer function in the **Scalars Config** panel for enhanced visualization (Panel 4 in Fig. 7 and Fig. 6). Users

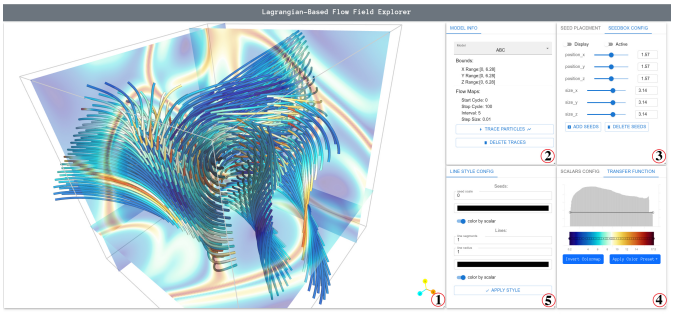


Fig. 6: Illustration of our web-based viewer for visualizing inferred pathlines using our pre-trained model in the **ABC** dataset. The interface includes panels for (1) main display, (2) model loading, data information and particle tracing, (3) seed box configuration, (4) transfer function for scalar field data visualization, and (5) seed and line style configuration.

can also modify the radius of seeds and the line width for the pathlines. Additionally, our viewer supports line smoothing by increasing the

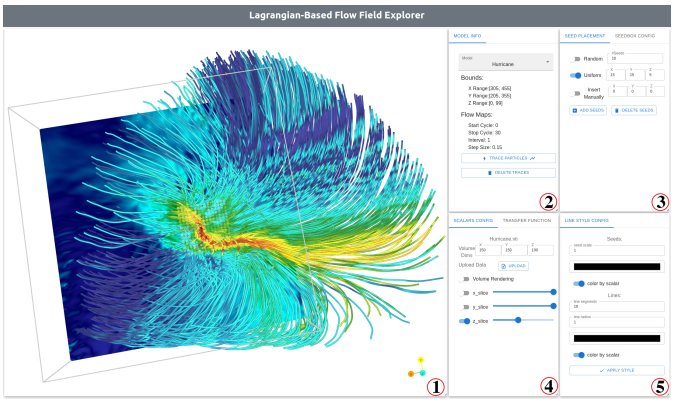


Fig. 7: Illustration of our web-based viewer for visualizing inferred pathlines using our pre-trained model in the **Hurricane** dataset. The interface includes panels for (1) main display, (2) model loading, dataset information and particle tracing, (3) seed placement, (4) scalar field data visualization, and (5) seed and line style configuration.

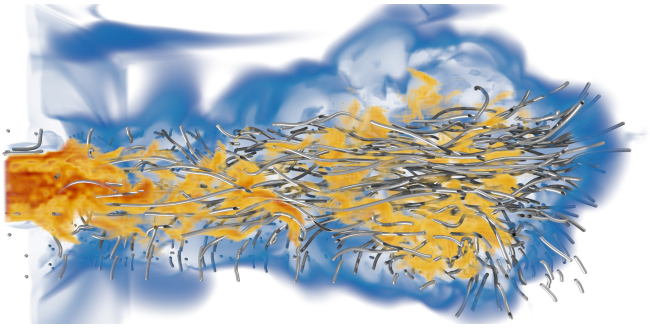


Fig. 8: Multiworkflow visualization of the **ScalarFlow** dataset using our OSPRay-based viewer, which integrates our neural network with the OSPRay renderer. The visualization displays the FTLE as a volume and the pathlines inferred using our neural network. A clipping plane is aligned along the y-axis. The model is trained with the $Lagrangian_{hybrid}$ approach. Each pathline encompasses 15 time steps, ranging from time step 135 to time step 150.

number of line segments in the **Line Style Config** panel (Panel 5 in Fig. 7 and Fig. 6. The number of line segments equals one providing a piece-wise linear path. A larger number of line segments will sample more points along the Catmull-Rom spline [5] to provide a smoother path for each pathline.

4.2 Integrating with OSPRay

Although using a web-based viewer is easier to access for nonpowerful computers, integrating the viewer with rendering engines such as OSPRay [6] is essential to leverage fast and high-fidelity rendering. For this purpose, we use the ORT C++ API to deploy the trained model. As shown in Fig. 8, we visualize pathlines inferred by our trained model and render them using OSPRay. While a user interface for the OSPRay viewer is scheduled for future development, we demonstrate the effectiveness of our approach for post hoc exploration of Lagrangian-based flow data.

REFERENCES

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. {TensorFlow}: A System for {Large-Scale} Machine Learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pages 265–283, 2016.

[2] I. Corouge, S. Gouttard, and G. Gerig. Towards a Shape Model of White Matter Fiber Bundles Using Diffusion Tensor MRI. In *2004 2nd IEEE international symposium on biomedical imaging: nano to macro (IEEE Cat No. 04EX821)*, pages 344–347. IEEE, 2004.

[3] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An Imperative Style, High-Performance Deep Learning Library. *Advances in neural information processing systems*, 32, 2019.

[4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine Learning in Python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[5] C. Twigg. Catmull-Rom splines. *Computer*, 41(6):4–6, 2003.

[6] I. Wald, G. P. Johnson, J. Amstutz, C. Brownlee, A. Knoll, J. Jeffers, J. Günther, and P. Navrátil. OSPRay-A CPU Ray Tracing Framework for Scientific Visualization. *IEEE transactions on visualization and computer graphics*, 23(1):931–940, 2016.