Algorithm 1 `sample_d1`

Input:
$\mathbf{v}$: velocity field
$\tilde{t}$: anchor time
$\mathcal{T} = (t_1, \dots, t_L), \mathcal{X} = (x_1, \dots, x_M), \mathcal{Y} = (y_1, \dots, y_N)$: regular grid

$\mathcal{D}_{1,pos} \leftarrow$ array of 2D vectors, size (L, M, N) ▷ saves destination of flow map
$\mathcal{D}_{1,bound} \leftarrow$ array of bytes, size (L, M, N) ▷ saves boundary flag

$bounds \leftarrow ((x_1, x_M), (y_1, y_N))$
$\mathbf{v}_c \leftarrow$ `constrain`$(\mathbf{v}, bounds)$ ▷ sets OutOfBounds outside grid

for $(i, j, k) \in (1, \dots, L) \times (1, \dots, M) \times (1, \dots, N)$ **in parallel do**
  $t \leftarrow \mathcal{T}[i]$
  $\mathbf{x} \leftarrow (\mathcal{X}[j], \mathcal{Y}[k])$
  $state \leftarrow$ `integrate`$(\mathbf{v}_c, \mathbf{x}, t, \tilde{t})$ ▷ integrates $\mathbf{v}$ from $(\mathbf{x}, t)$ to $\tilde{t}$, returns end state
  $(\mathbf{p}, b) \leftarrow$ `encode_d1`$(bounds, state)$ ▷ see Algorithm 4
  $\mathcal{D}_{1,pos}[i, j, k] \leftarrow \mathbf{p}$
  $\mathcal{D}_{1,bound}[i, j, k] \leftarrow b$
end for

return $\mathcal{D}_{1,pos}, \mathcal{D}_{1,bound}$

Algorithm 2 `sample_d2`

Input:
$\mathbf{v}$: velocity field
$\tilde{t}$: anchor time
$\mathcal{T} = (t_1, \dots, t_L), \mathcal{X} = (x_1, \dots, x_M), \mathcal{Y} = (y_1, \dots, y_N)$: regular grid

$\mathcal{D}_2 \leftarrow$ array of 2D vectors, size (L, M, N)

for $(i, j, k) \in (1, \dots, L) \times (1, \dots, M) \times (1, \dots, N)$ **in parallel do**
  $t \leftarrow \mathcal{T}[i]$
  $\mathbf{x} \leftarrow (\mathcal{X}[j], \mathcal{Y}[k])$
  $state \leftarrow$ `integrate`$(\mathbf{v}, \mathbf{x}, \tilde{t}, t)$ ▷ integrates from $(\mathbf{x}, t)$ to $\tilde{t}$, returns end state
  $\mathcal{D}_2[i, j, k] \leftarrow$ `pos`$(state)$ ▷ may be undefined vector if leaving domain
end for

return $\mathcal{D}_2$

Algorithm 3 `sample_d3`

Input:
$\mathbf{v}$: velocity field
$\tilde{t}$: anchor time
$\mathcal{T} = (t_1, \dots, t_L), \mathcal{X} = (x_1, \dots, x_M), \mathcal{Y} = (y_1, \dots, y_N)$: regular grid
b: boundary flag

$fixed \leftarrow (x_1, y_1, x_M, y_N)[b]$
$dim \leftarrow b \bmod 2$
$K \leftarrow dim = 1 \; ? \; N : M$ ▷ length of free dimension

$\mathcal{D}_{3,b} \leftarrow$ array of 2D vectors, size (L, L, K) ▷ grid with two time and one space dimensions

for $(i, j, k) \in (1, \dots, L) \times (1, \dots, L) \times (1, \dots, K)$ **in parallel do**
  $\mathbf{x} \leftarrow dim = 1 \; ? \; (fixed, \mathcal{Y}[k]) : (\mathcal{X}[k], fixed)$ ▷ construct position on the boundary
  $state \leftarrow$ `integrate`$(\mathbf{v}, \mathbf{x}, \mathcal{T}[i], \mathcal{T}[j])$
  $\mathcal{D}_{3,b}[i, j, k] \leftarrow$ `pos`$(state)$
end for

return $\mathcal{D}_{3,b}$

Algorithm 4 `encode_d1`

Input:
$bounds = ((x_s, x_e), (y_s, y_e))$: domain boundary
$state$: contains information of the end state of an integration

$t' \leftarrow$ `time`$(state)$
$\mathbf{x}' \leftarrow$ `pos`$(state)$
$\mathbf{v}' \leftarrow$ `velocity`$(state)$ ▷ important: vector is inverted if applying backward integration

if `succeeded`$(state)$ **then** ▷ default case: pathline stayed inside domain
  return $(\mathbf{x}', 0)$
end if

$b \leftarrow -1$ ▷ boundary flag
$d_{min} \leftarrow \infty$ ▷ current min distance to boundary

for $i \in \{1, 2\}$ **do** ▷ search for boundary flag b where the domain was left
  $d_s \leftarrow \mathbf{x}'[i] - bounds[i][1]$ ▷ distance to x_s or y_s
  $d_e \leftarrow bounds[i][2] - \mathbf{x}'[i]$ ▷ distance to x_e or y_e
  if $(\mathbf{v}[i] < 0) \wedge (d_s < d_{min})$ **then**
    $d_{min} \leftarrow d_s, \quad b \leftarrow i$
  else if $(\mathbf{v}[i] > 0) \wedge (d_e < d_{min})$ **then**
    $d_{min} \leftarrow d_e, \quad b \leftarrow 2 + i$
  end if
end for

$dim \leftarrow b \bmod 2$
$\mathbf{p} \leftarrow (dim = 1) \; ? \; (t', y') \; : \; (t', x')$ ▷ compose time-space vector

return $(\mathbf{p}, b)$

Algorithm 5 `reconstruct`

Input:
afm: anchor flow map with grid and pre-sampled data
$\mathbf{x}_0, t_0, t_1$: evaluation input

$(\mathcal{T}, \mathcal{X}, \mathcal{Y}) \leftarrow$ `grid_data`(afm)
$\mathcal{D}_{1,pos}, \mathcal{D}_{1,bound} \leftarrow$ `get_d1`(afm)
$\mathbf{x}_1 \leftarrow (0, 0)$ ▷ this will be the result
$(\mathcal{I}, \mathcal{W}) \leftarrow$ `interpolants`$((\mathcal{T}, \mathcal{X}, \mathcal{Y}), (t_0, \mathbf{x}_0))$ ▷ get indices and weights for interpolation

for $(i, w) \in (\mathcal{I}, \mathcal{W})$ **do**
  $(\mathbf{p}, b) \leftarrow (\mathcal{D}_{1,pos}[i], \; \mathcal{D}_{1,bound}[i])$
  if $b = 0$ **then**
    $\mathcal{D}_2 \leftarrow$ `get_d2`(afm)
    $\mathbf{x}_1 \leftarrow \mathbf{x}_1 + w \cdot$ `interpolate`$((\mathcal{T}, \mathcal{X}, \mathcal{Y}), \mathcal{D}_2, (t_1, \mathbf{p}))$
  else
    $\mathcal{D}_{3,b} \leftarrow$ `get_d3`(afm, b)
    $(t', s') = \mathbf{p}$ ▷ free time and space values
    $\mathbf{q} \leftarrow (t', t_1, s')$ ▷ insert target time t_1
    $dim \leftarrow b \bmod 2$
    $grid \leftarrow dim = 1 \; ? \; (\mathcal{T}, \mathcal{T}, \mathcal{Y}) \; : \; (\mathcal{T}, \mathcal{T}, \mathcal{X})$ ▷ determine corresponding grid
    $\mathbf{x}_1 \leftarrow \mathbf{x}_1 + w \cdot$ `interpolate`$(grid, \mathcal{D}_{3,b}, \mathbf{q})$
  end if
end for

return $\mathbf{x}_1$

1. Algorithms

Here we provide some information about the algorithms. Please see Sections 4.2 (Sampling and Reconstruction) and 4.4 (Implementation) of the main paper first. Again, we describe the 2D case for simplicity, but the 3D extension is straightforward.

Algorithms 1, 2 and 3 show pseudocode for sampling the fields $\mathbf{d}_1, \mathbf{d}_2$ and $\mathbf{d}_3$. Algorithm 4 explains how we encode the result of $\mathbf{d}_1$. If the pathline stays in the domain, the result is $(\mathbf{x}', 0)$, i.e., the spatial end position and a zero. Latter encodes that we will use $\mathbf{d}_2$ in the reconstruction. Otherwise, the integration ended at a spatial boundary and we search the respective side (x or y, min or max). Each boundary side has a flag $b \in \{1, 2, 3, 4\}$. Additionally, we return the vector $\mathbf{p}$ which contains the free values, i.e., the end time and the remaining spatial value.

When computing $\mathbf{d}_3$ we only need to call `sample_d3` for a given boundary flag if $b \in \mathcal{D}_{1,bound}$, i.e., at lease one pathline left the domain on this side. This saves time and memory usage.

For Algorithm 5, note the difference between functions `interpolants` and `interpolate`. First returns vertex indices and weights for multilinear interpolations in the respective grid, latter directly applies interpolation with provided data $\mathcal{D}_2$ or $\mathcal{D}_{3,b}$.

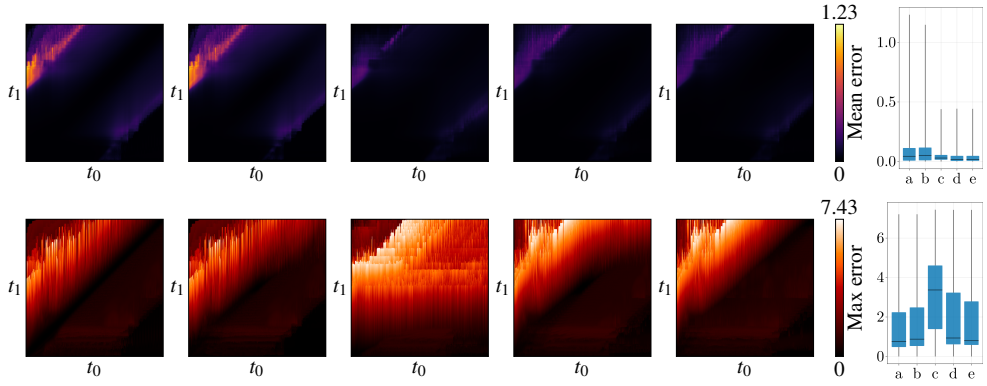
2. Error Plots

The main paper shows error plots of the Double Gyre dataset. The ABC flow (Figure 2) provides similar plots for mean errors. In case of maximum errors, however, the median and 25% quartile results of AFM$_c$ are worse than for FFM, even with res_A. The reason: all six boundary surfaces must be sampled for $\mathbf{d}_3$. This decreases the factor between res_F and res_A, leading to a smaller advantage.

The error plots for the Cylinder and Tangaroa datasets (Figures 1 and 3) look much less „regular". This stems from pathlines leaving the domain at some points which have a strong impact on the averages. This is especially prominent since the pathlines which stay in the domain longest show strong turbulence, and such pathlines are the hardest ones to approximate. For globally defined datasets like the Double Gyre and the ABC flow this does not apply.

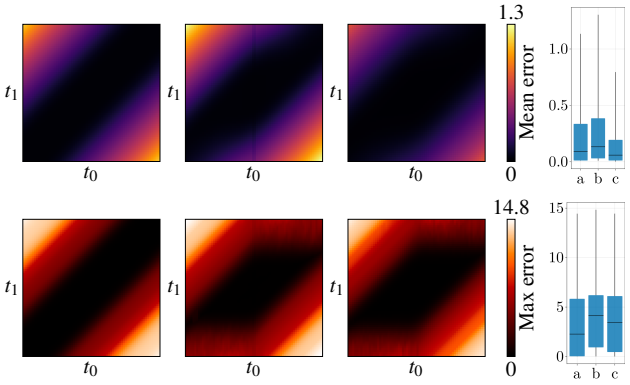
Dataset	Setup	Method	Mean errors	Max errors	FP	FN
Double Gyre	res_F	FFM	0.012 ± 0.017	0.58 ± 0.59	0%	0%
		AFM_c	0.014 ± 0.017	0.74 ± 0.52	0%	0%
	res_A	AFM_c	0.0028 ± 0.0042	0.54 ± 0.51	0%	0%
Cylinder	res_F	FFM	0.090 ± 0.14	1.48 ± 1.50	0.0064%	1.308%
		AFM_c	0.093 ± 0.14	1.59 ± 1.49	0.0029%	1.764%
	res_A	AFM_s	0.042 ± 0.046	3.18 ± 1.90	0.0034%	1.090%
		AFM_c	0.039 ± 0.057	1.96 ± 1.85	0.0031%	0.684%
		AFM_e	0.038 ± 0.053	1.80 ± 1.84	0.0021%	0.932%
ABC	res_F	FFM	0.21 ± 0.25	3.70 ± 4.20	0%	0%
		AFM_c	0.25 ± 0.28	4.45 ± 3.90	0%	0%
	res_A	AFM_c	0.13 ± 0.16	3.99 ± 3.70	0%	0%
Tangaroa	res_F	FFM	0.027 ± 0.046	0.29 ± 0.21	0.0035%	2.640%
		AFM_c	0.024 ± 0.042	0.30 ± 0.19	0.0017%	3.580%
	res_A	AFM_c	0.022 ± 0.038	0.31 ± 0.19	0.0014%	2.447%

Table 1: Error statistics of the AFM and FFM compared to the ground truth (i.e., pathline integration). For the AFM method, the subscript determines which anchor value $\tilde{t}$ is selected, with values t_s for AFM_s, t_e for AFM_e, and the center value for AFM_c. As explained in the main paper, we tracked the mean and maximum errors for each (t_0, t_1) sample. The values here show the respective means plus/minus standard deviations. Further, FP and NP note percentages of false positives and false negatives, i.e., falsely returning a flow map value or falsely returning none.



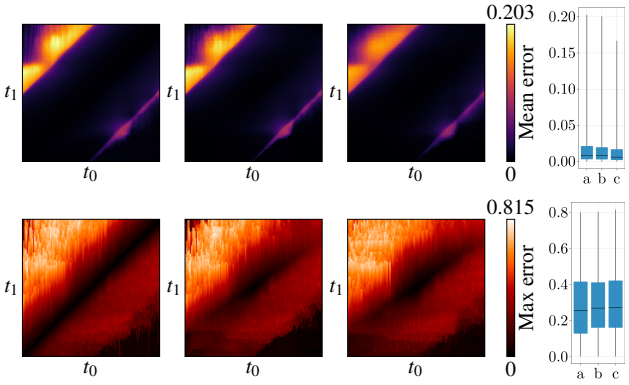
(a) FFM, res_F (b) AFM_c, res_F (c) AFM_s, res_A (d) AFM_c, res_A (e) AFM_e, res_A

Figure 1: Error plots for the Cylinder dataset.



(a) FFM, res_F (b) AFM_c, res_F (c) AFM_c, res_A

Figure 2: Error plots for the ABC flow dataset.



(a) FFM, res_F (b) AFM_c, res_F (c) AFM_c, res_A

Figure 3: Error plots for the Tangaroa dataset.