Neural methods for (nonlinear) advection-dominated equations

Victor Michel-Dansac*, joint work with
Martin Campos Pinto, Victor Fournet, Emmanuel Franck
Philipp Krah, Laurent Navoret, Vincent Vigon

Workshop on Machine Learning & Simulation, Nice, 12 October 2026

*MACARON project-team, Université de Strasbourg, CNRS, Inria, IRMA, France





1. Motivation: High-dimensional advection-dominated equations 1
2. Characteristic curves and the semi-Lagrangian method 3
3. The neural semi-Lagrangian method 5
4. Lagrangian PINNs based on flow maps 10
5. The neural δf particle-in-cell method 15
6. Conclusion and outlook 20

1. Motivation: High-dimensional advection-dominated equations

Plasma physics and the curse of dimensionality

High-dimensional advection-dominated equations appear for instance in **plasma physics & astrophysics**, through the Vlasov or Vlasov-Poisson equations.
For instance, high dimensionality is required to accurately model fusion in Tokamaks.

The unknown $f(t, x, v)$ represents a probability density in **phase space** ($d = 6$).

Plasma physics and the curse of dimensionality

High-dimensional advection-dominated equations appear for instance in **plasma physics & astrophysics**, through the Vlasov or Vlasov-Poisson equations.
For instance, high dimensionality is required to accurately model fusion in Tokamaks.

The unknown $f(t, x, v)$ represents a probability density in **phase space** ($d = 6$).

Main bottleneck: The curse of dimensionality
Grid sizes grow exponentially with the dimension: even a coarse 6D grid has $30^6 \approx 10^9$ points!

Goal: Use numerical methods based on **neural networks** in combination with specific features of advection-dominated equations to alleviate the curse of dimensionality.

Physics-informed neural networks (PINNs)

We consider a generic partial differential equation (PDE) of the form

$$\partial_t f(t,x) + \mathcal{D}[f](t,x) = 0, \qquad \text{for} \quad t \in [0,T] \text{ and } x \in \Omega.$$

PINNs [Raissi et al., 2019]: one network for the whole space-time domain.

The network $f_\theta : (t,x) \mapsto \mathcal{N}(t,x;\theta)$ has **dofs (trainable parameters)** θ **that are global constants** in time and space, minimizing the full residual, as

$$\theta \in \operatorname*{argmin}_{\vartheta \in \Theta} \int_0^T \int_\Omega \left| \partial_t f_\vartheta + \mathcal{D}[f_\vartheta] \right|^2 dx\, dt.$$

This is a **global optimization problem**, and a single network has to represent f at all times.

Physics-informed neural networks (PINNs)

We consider a generic partial differential equation (PDE) of the form

$$\partial_t f(t,x) + \mathcal{D}[f](t,x) = 0, \qquad \text{for} \quad t \in [0,T] \text{ and } x \in \Omega.$$

PINNs [Raissi et al., 2019]: one network for the whole space-time domain.

The network $f_\theta : (t,x) \mapsto \mathcal{N}(t,x;\theta)$ has **dofs (trainable parameters)** θ **that are global constants** in time and space, minimizing the full residual, as

$$\theta \in \operatorname*{argmin}_{\vartheta \in \Theta} \int_0^T \int_\Omega \left| \partial_t f_\vartheta + \mathcal{D}[f_\vartheta] \right|^2 dx\, dt.$$

This is a **global optimization problem**, and a single network has to represent f at all times.

Two drawbacks of PINNs:
- they are **not specialized** for advection-dominated problems;
- they struggle with solutions that **change a lot over time**.

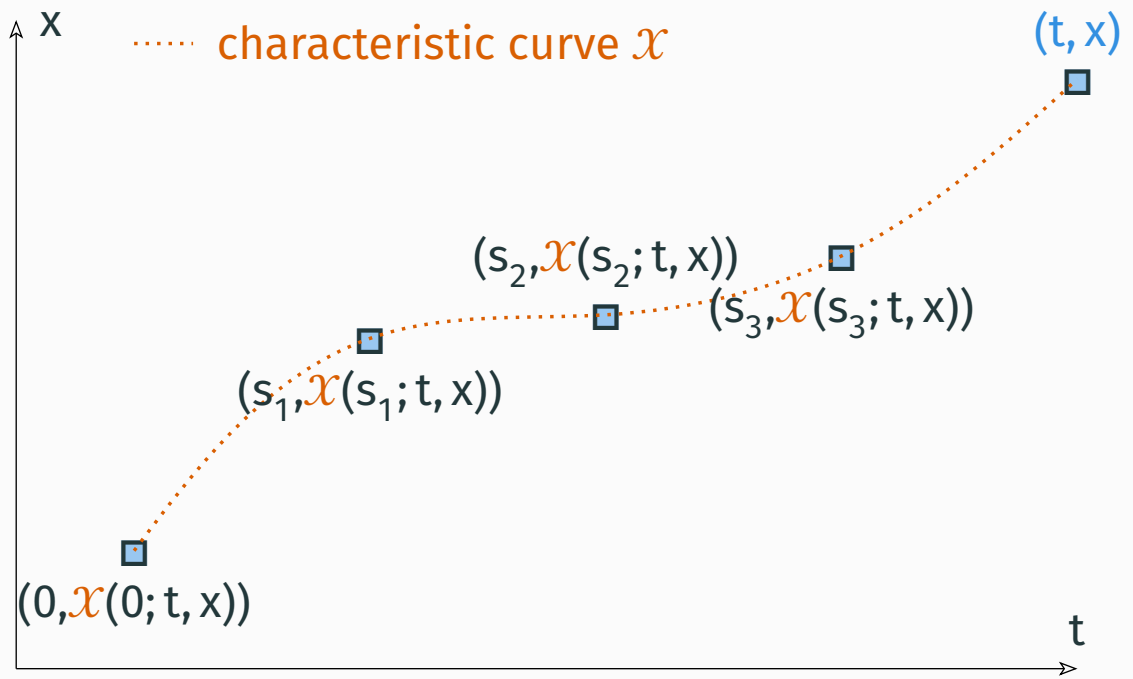
2. Characteristic curves and the semi-Lagrangian method

Characteristic curves and flow maps

Consider the linear advection equation $\partial_t f + a(t,x) \cdot \nabla_x f = 0$.

The characteristic curve $\mathcal{X}$ solves this backwards ODE for given t and x:

$$\begin{cases} \dfrac{d}{ds}\mathcal{X}(s;t,x) = a(s,\mathcal{X}(s;t,x)) , \quad \forall s \in [0,t] , \\ \mathcal{X}(t;t,x) = x. \end{cases}$$



Characteristic curves and flow maps

Consider the linear advection equation $\partial_t f + a(t,x) \cdot \nabla_x f = 0$.

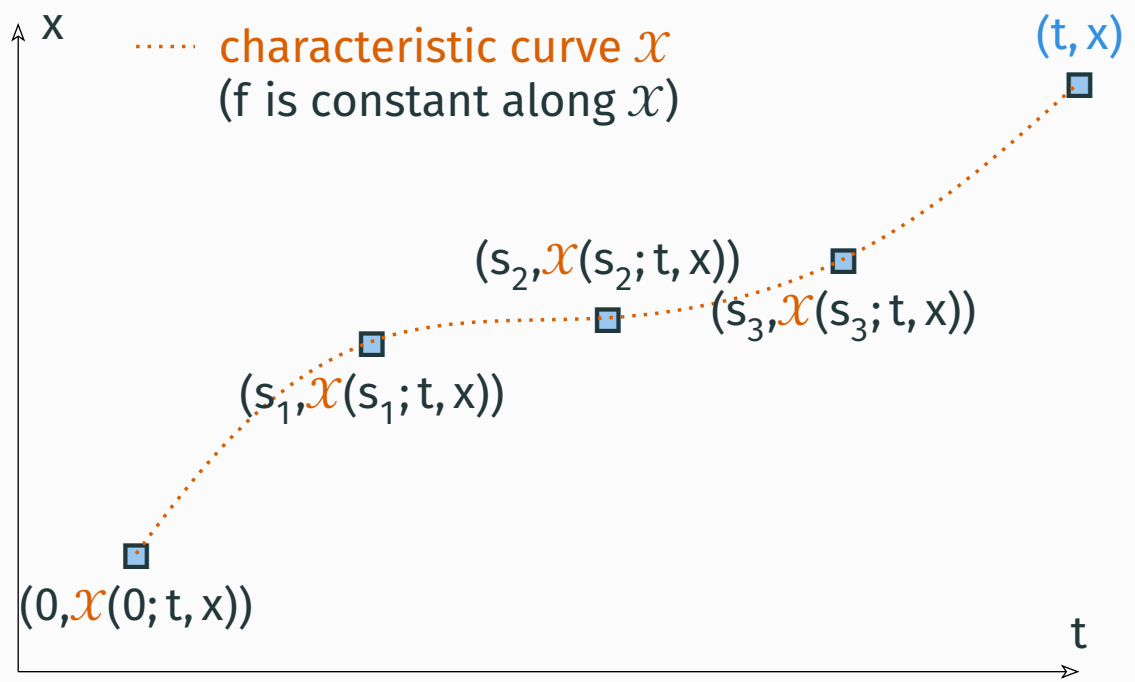
The characteristic curve $\mathcal{X}$ solves this backwards ODE for given t and x:

$$\begin{cases} \dfrac{d}{ds}\mathcal{X}(s;t,x) = a(s,\mathcal{X}(s;t,x)), \quad \forall s \in [0,t], \\ \mathcal{X}(t;t,x) = x. \end{cases}$$

The solution of the transport equation is

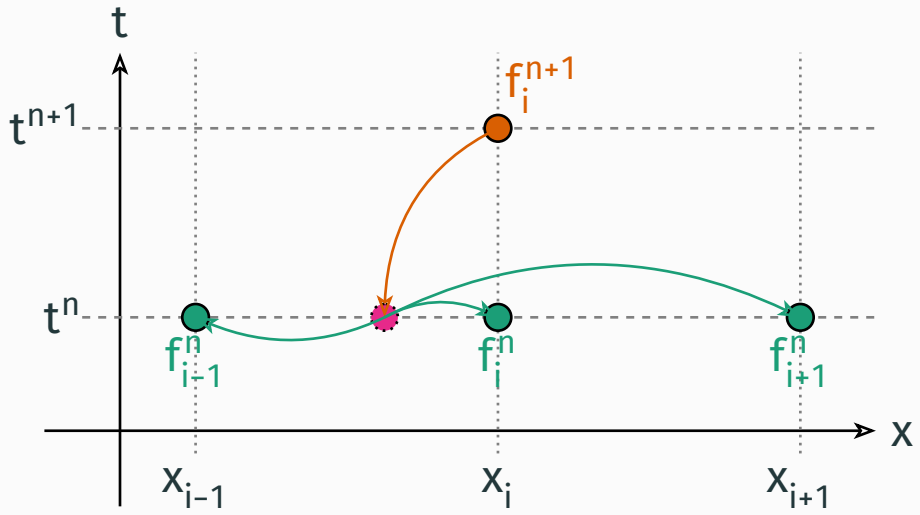
$$\forall x \in \Omega, \forall t \in [0,T], \quad \forall s \in [0,t],$$
$$f(t,x) = f(s,\mathcal{X}(s;t,x)).$$

For instance, if $f(0,\cdot) = f_0$, then $f(t,x) = f_0(\mathcal{X}(0;t,x))$.



The semi-Lagrangian method [Staniforth et al., 1991]

classical semi-Lagrangian scheme:
update for one mesh point



- step 1: project ● onto ● using the characteristic curve
- step 2: interpolate the value at ● from the neighbors ●

This method **does not have a stability restriction** on the time step Δt.

However, because of the interpolation, it is still subject to the **curse of dimensionality**.

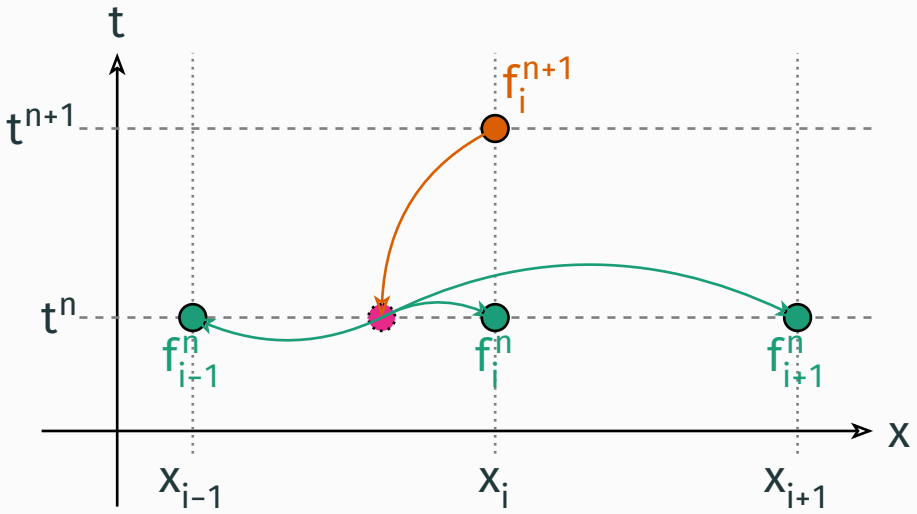
Could a neural method be used here?

3. The neural semi-Lagrangian method

[Franck, M.-D., Navoret & Vigon, CMAME, 2026]

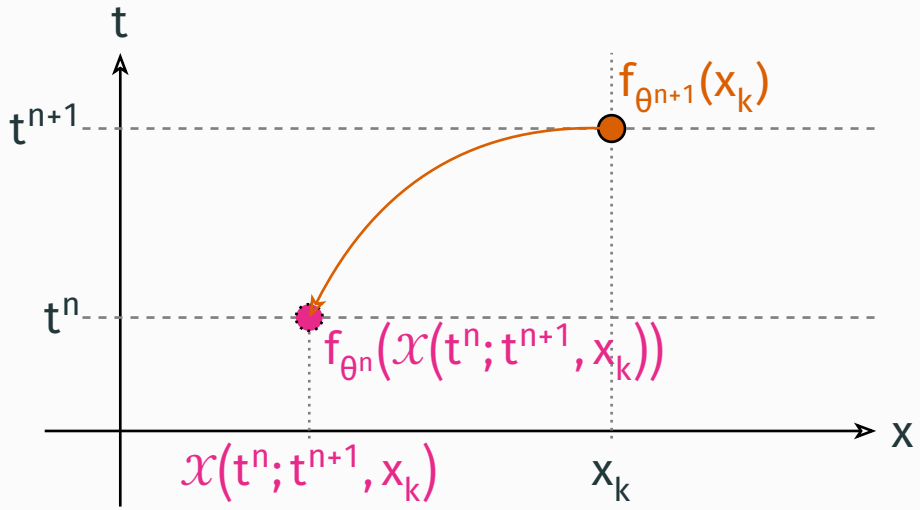
Classical vs neural semi-Lagrangian methods

classical semi-Lagrangian scheme:
update for one mesh point



- step 1: project ● onto ● using the characteristic curve
- step 2: interpolate the value at ● from the neighbors ●

neural semi-Lagrangian scheme:
update for one sampled point



- step 1: starting from ●, compute the foot ● of the characteristic curve
- step 2: fit $f_{\theta^{n+1}}(x_k)$ on $f_{\theta^n}(\mathcal{X}(t^n;t^{n+1},x_k))$

Optimization problem

The exact solution of the advection equation satisfies $f(t^{n+1}, x) = f(t^n, \mathcal{X}(t^n; t^{n+1}, x))$.

Thus, we seek the dofs θ such that

$$\forall n \geq 0, \quad \forall x \in \Omega, \qquad f_{\theta^{n+1}}(x) \approx f_{\theta^n}(\mathcal{X}(t^n; t^{n+1}, x)).$$

This leads us to considering the following optimization problem, linking the dofs at time t^n and t^{n+1}:

$$\theta^{n+1} \in \operatorname*{argmin}_{\vartheta \in \Theta} \int_\Omega \left| f_\vartheta(x) - f_{\theta^n}(\mathcal{X}(t^n; t^{n+1}, x)) \right|^2 dx.$$

This is simply a **function approximation problem**, no PDE! We have one nonlinear optimization per time step.

Optimization problem

The exact solution of the advection equation satisfies $f(t^{n+1}, x) = f(t^n, \mathcal{X}(t^n; t^{n+1}, x))$.

Thus, we seek the dofs θ such that

$$\forall n \geq 0, \quad \forall x \in \Omega, \qquad f_{\theta^{n+1}}(x) \approx f_{\theta^n}(\mathcal{X}(t^n; t^{n+1}, x)).$$

This leads us to considering the following optimization problem, linking the dofs at time t^n and t^{n+1}:

$$\theta^{n+1} \in \operatorname*{argmin}_{\vartheta \in \Theta} \int_\Omega \left| f_\vartheta(x) - f_{\theta^n}(\mathcal{X}(t^n; t^{n+1}, x)) \right|^2 dx.$$

This is simply a **function approximation problem**, no PDE! We have one nonlinear optimization per time step.

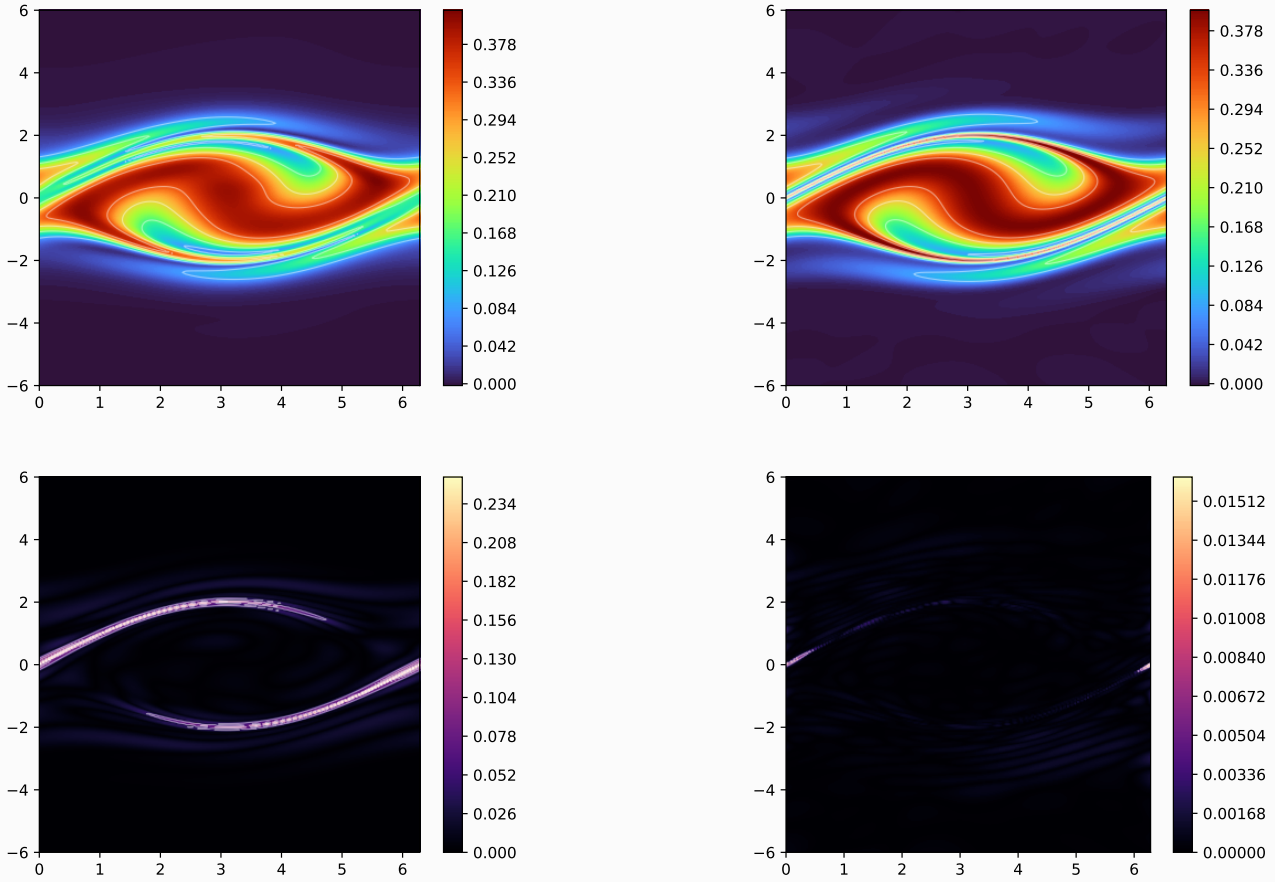
Remark: Extensive numerical experiments show that the error is **dominated by the optimization error**, which has to be reduced by using a quasi-second-order optimizer (e.g., Gauss-Newton or Levenberg-Marquardt).

Linear Vlasov equation in 1D (space) + 1D (velocity)

linear Vlasov equation: $\partial_t f(t,x,v) + v\,\partial_x f(t,x,v) + \sin(x)\,\partial_v f(t,x,v) = 0, \quad t \in [0, 4.5],\ x \in [0, 2\pi],\ v \in [-6, 6]$

prediction at t = 4.5

error at t = 4.5



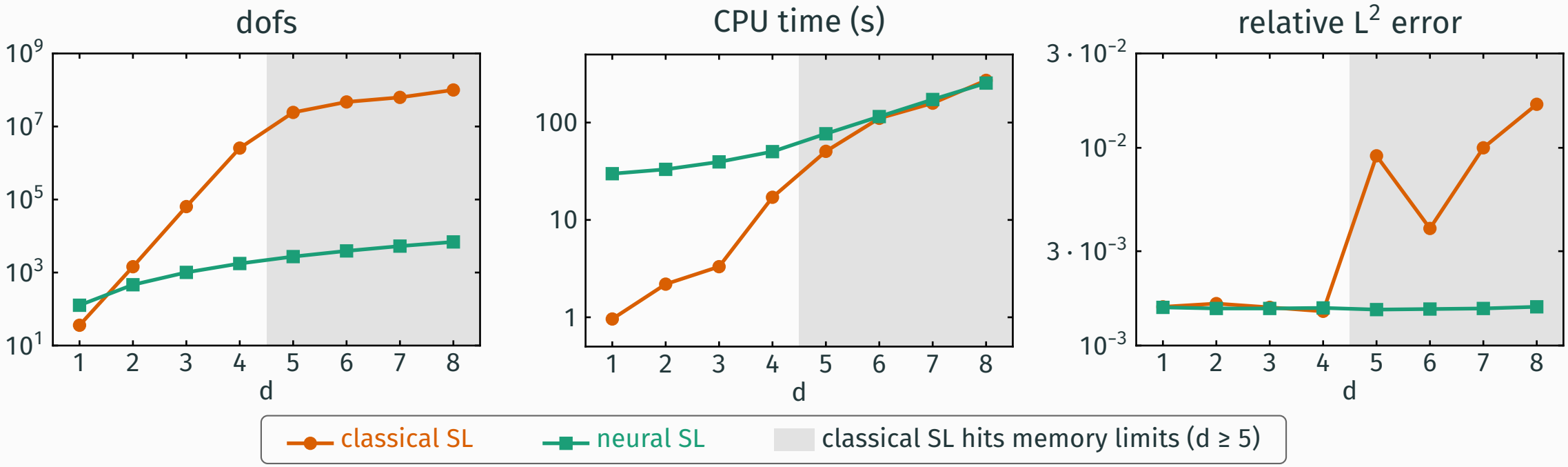
PINN neural semi-Lagrangian

	PINN	NSL
N	929	881
Δt	N/A	1.125
time	~30s	~30s
rel. L^2 error	1.0×10^{-1}	2.3×10^{-3}

Same number of dofs and similar CPU time: the NSL error is about 45 times smaller.

Linear advection-diffusion equation

$$\partial_t f + a \cdot \nabla f - \sigma \Delta f = 0 \quad \text{in} \quad \Omega = (-1, 1)^d, \quad \text{with} \quad a = (1, \dots, 1) \quad \text{and} \quad \sigma = 0.1$$



⇝ On this simple example, the classical SL error grows with d when memory issues hit (for d ≥ 5), but the NSL error remains stable and independent of d.

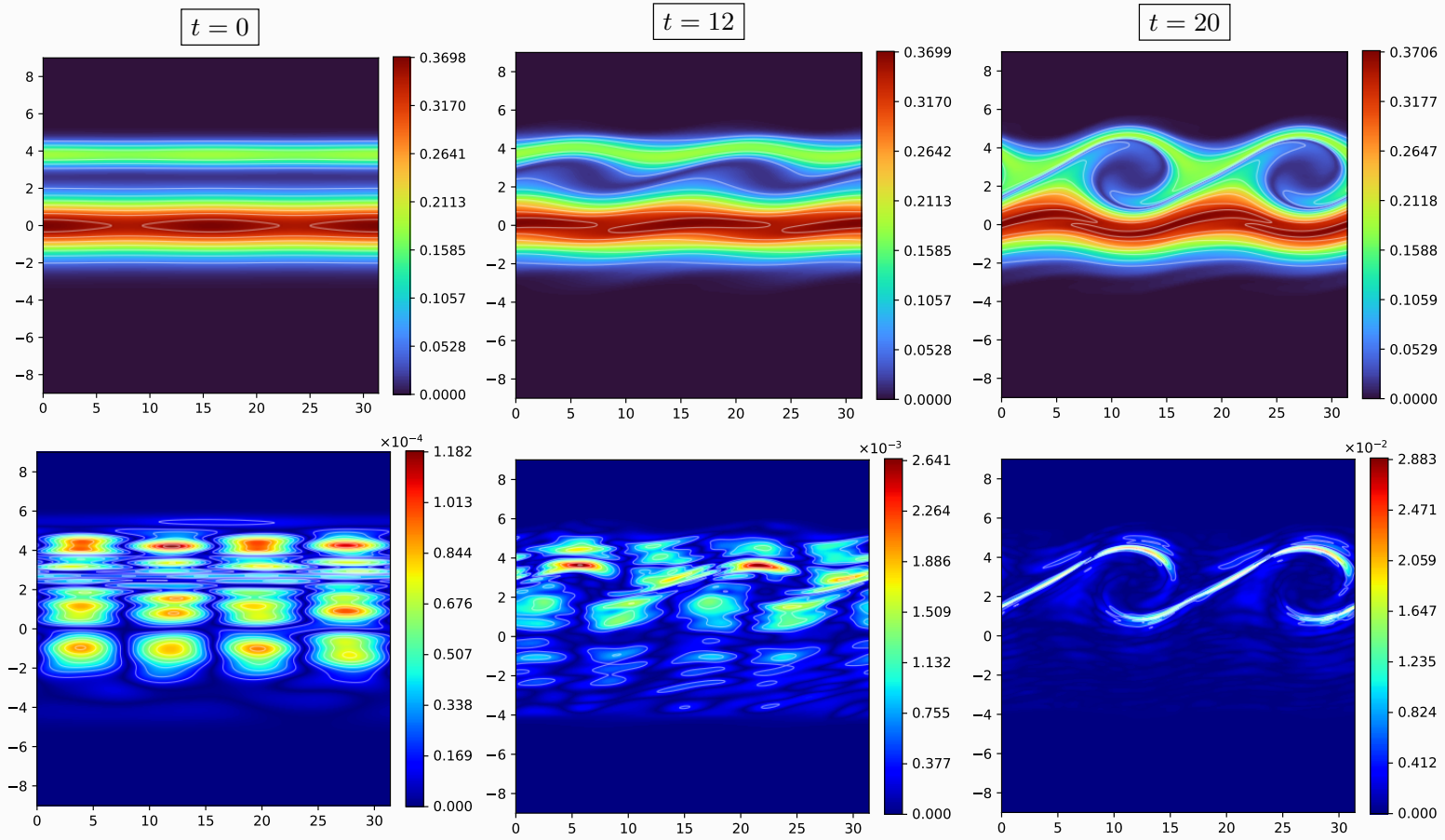
Extension to the Vlasov-Poisson equation: the bump-on-tail instability

Vlasov-Poisson equation: $\partial_t f + v\,\partial_x f + E[f]\,\partial_v f = 0$, where $E[f] = \partial_x \Psi[f]$ with $\partial_{xx}\Psi[f](t,x) = \int f(t,x,v)\,dv$

It is a nonlinear PDE: f is solved with the neural SL method, and E with a PINN. Errors increase in long time simulations.

prediction

error



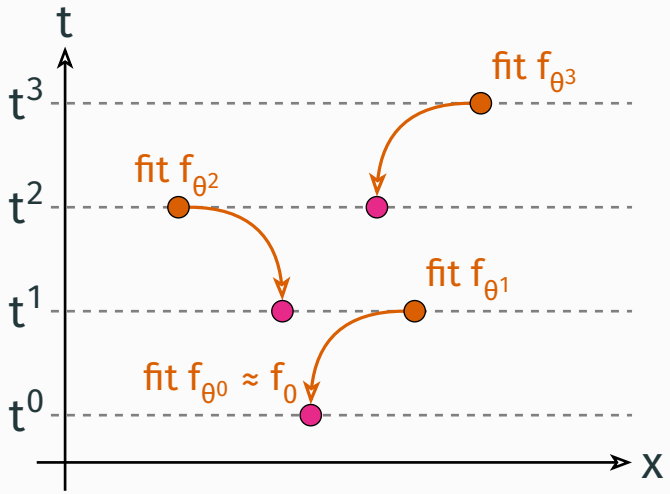
4. Lagrangian PINNs based on flow maps

[Campos Pinto, Fournet, Franck, Krah & M.-D., preprint, 2026]

Representing the flow map instead of the solution

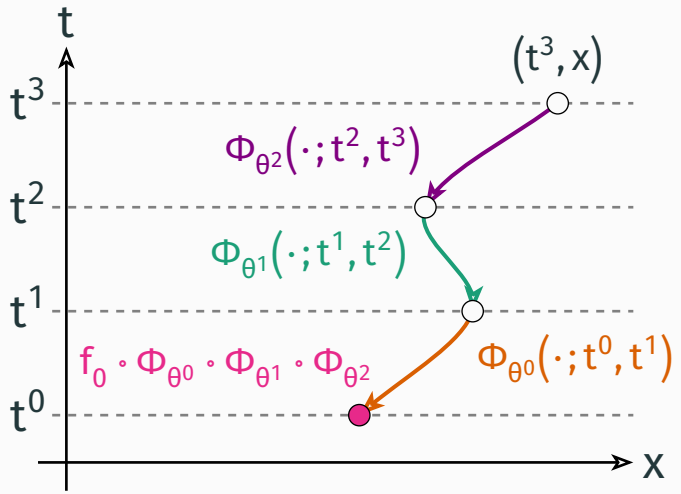
For fixed t, the map $\Phi(\cdot; 0, t) : x \mapsto \mathcal{X}(0; t, x)$ is the **backward flow map**; we approximate either f or this map.

neural SL: the network represents $f(t^n, \cdot)$



- one function fit per time step: $f_{\theta^{n+1}} \approx f_{\theta^n} \circ \tilde{\mathcal{X}}$
- the solution is re-approximated at each step

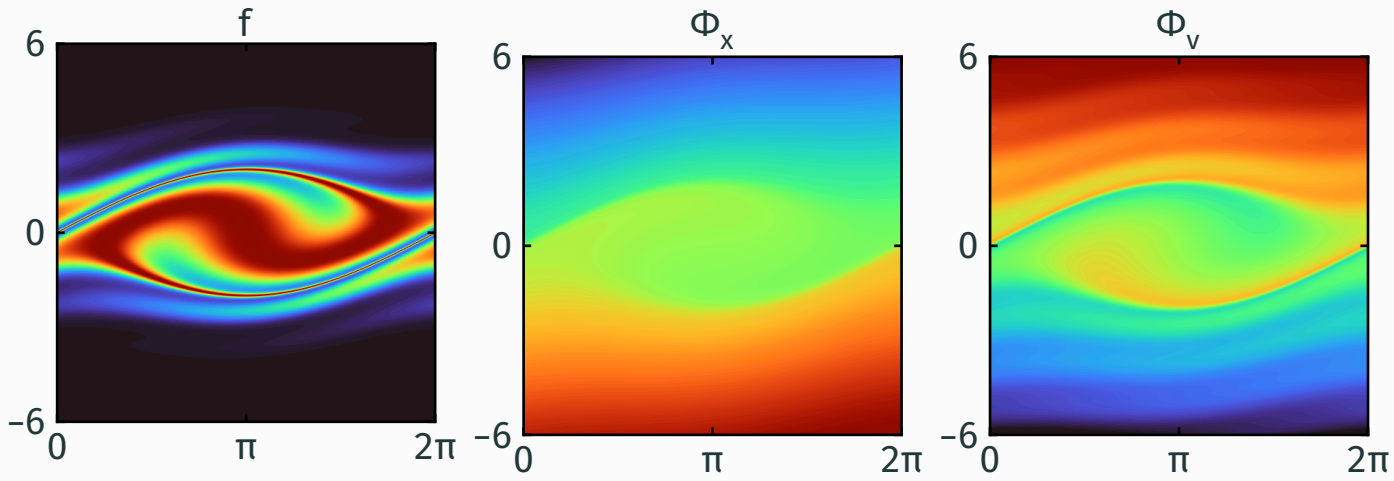
Lagrangian PINN: the networks represent the flow map Φ, which satisfies $f(t, x) = f_0(\Phi(x; 0, t))$



- one PINN per time step, approximating the flow map
- f_0 is evaluated exactly

Lagrangian PINNs modify the neural SL method: **instead of the solution, the networks represent the flow map.**

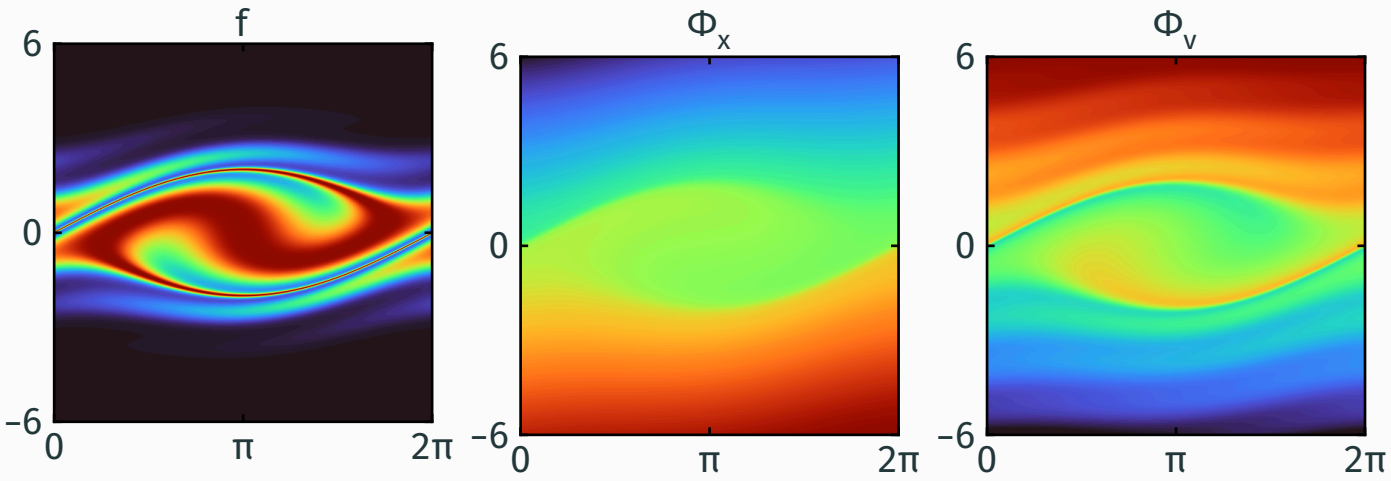
Why learn flow maps and submaps?



linear Vlasov, t = 4.5

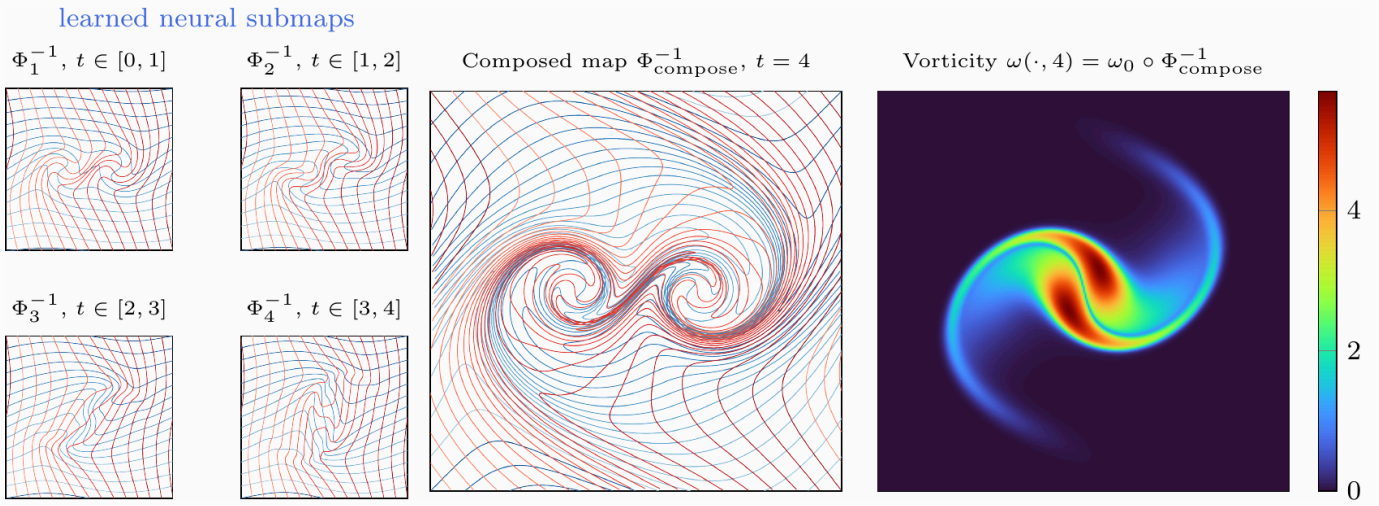
⇝ the flow map Φ is much smoother than $f = f_0 \circ \Phi$

Why learn flow maps and submaps?



linear Vlasov, t = 4.5

⇝ the flow map Φ is much smoother than $f = f_0 \circ \Phi$



vortex merger for the isentropic Eule equations, t = 4

⇝ each submap is much simpler than their composition

Lagrangian PINNs: training the flow map

If f satisfies the linear transport equation $\partial_t f + \nabla_x \cdot (a\, f) = 0$, then the flow map $\Phi_0^t : x \in \Omega \mapsto \mathcal{X}(0; t, x) \in \Omega$ also satisfies a linear transport equation:

$$\partial_t \Phi_0^t + \big(a \cdot \nabla_x\big)\Phi_0^t = 0, \quad \Phi_0^0 = \mathrm{Id}, \qquad \text{and we have} \quad f(t,x) = f_0\big(\Phi_0^t(x)\big)\left|\det \nabla_x \Phi_0^t(x)\right|.$$

Lagrangian PINNs: training the flow map

If f satisfies the linear transport equation $\partial_t f + \nabla_x \cdot (a f) = 0$, then the flow map $\Phi_0^t : x \in \Omega \mapsto \mathcal{X}(0; t, x) \in \Omega$ also satisfies a linear transport equation:

$$\partial_t \Phi_0^t + (a \cdot \nabla_x)\Phi_0^t = 0, \quad \Phi_0^0 = \mathrm{Id}, \qquad \text{and we have} \quad f(t,x) = f_0\big(\Phi_0^t(x)\big)\left|\det \nabla_x \Phi_0^t(x)\right|.$$

The **Lagrangian PINN** approximates Φ_0^t by a network $\Phi_\theta(t, \cdot)$, minimizing the residual of the above equation:

$$\theta \in \operatorname*{argmin}_\vartheta \int_0^T \int_\Omega \left\| \partial_t \Phi_\vartheta + (a \cdot \nabla_x)\Phi_\vartheta \right\|^2 dx\, dt + \lambda \int_\Omega \left\| \Phi_\vartheta(0,x) - x \right\|^2 dx.$$

Lagrangian PINNs: training the flow map

If f satisfies the linear transport equation $\partial_t f + \nabla_x \cdot (a f) = 0$, then the flow map $\Phi_0^t : x \in \Omega \mapsto \mathcal{X}(0; t, x) \in \Omega$ also satisfies a linear transport equation:

$$\partial_t \Phi_0^t + (a \cdot \nabla_x)\Phi_0^t = 0, \quad \Phi_0^0 = \mathrm{Id}, \qquad \text{and we have} \quad f(t,x) = f_0\left(\Phi_0^t(x)\right) \left|\det \nabla_x \Phi_0^t(x)\right|.$$

The **Lagrangian PINN** approximates $\Phi_{t^n}^{t^{n+1}}$ by a network $\Phi_{\theta^n}(t, \cdot)$, minimizing the residual of the above equation:

$$\theta^n \in \operatorname*{argmin}_{\vartheta} \int_{t^n}^{t^{n+1}} \int_\Omega \left\| \partial_t \Phi_\vartheta + (a \cdot \nabla_x)\Phi_\vartheta \right\|^2 dx\, dt + \lambda \int_\Omega \left\| \Phi_\vartheta(t^n, x) - x \right\|^2 dx.$$

Compared to a direct approximation of f, we have mass and volume conservation if Φ_{θ^n} is invertible.

Lagrangian PINNs: training the flow map

If f satisfies the linear transport equation $\partial_t f + \nabla_x \cdot (a f) = 0$, then the flow map $\Phi_0^t : x \in \Omega \mapsto \mathcal{X}(0; t, x) \in \Omega$ also satisfies a linear transport equation:

$$\partial_t \Phi_0^t + (a \cdot \nabla_x)\Phi_0^t = 0, \quad \Phi_0^0 = \mathrm{Id}, \qquad \text{and we have} \quad f(t,x) = f_0\left(\Phi_0^t(x)\right) \left|\det \nabla_x \Phi_0^t(x)\right|.$$

The **Lagrangian PINN** approximates $\Phi_{t^n}^{t^{n+1}}$ by a network $\Phi_{\theta^n}(t, \cdot)$, minimizing the residual of the above equation:

$$\theta^n \in \operatorname*{argmin}_{\vartheta} \int_{t^n}^{t^{n+1}} \int_\Omega \left\| \partial_t \Phi_\vartheta + (a \cdot \nabla_x)\Phi_\vartheta \right\|^2 dx\, dt + \lambda \int_\Omega \left\| \Phi_\vartheta(t^n, x) - x \right\|^2 dx.$$

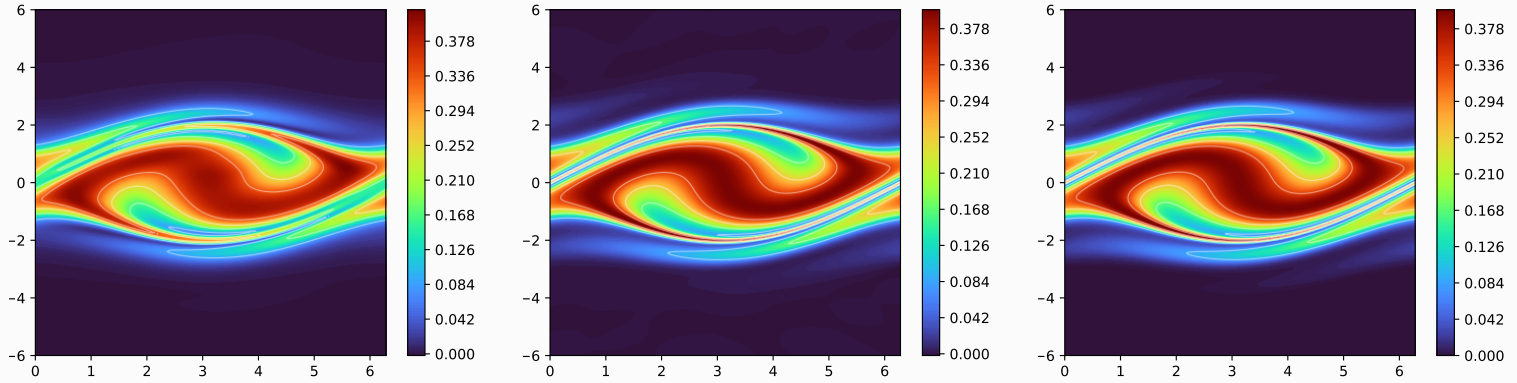
Compared to a direct approximation of f, we have mass and volume conservation if Φ_{θ^n} is invertible.

In the **nonlinear case** (a = a[f]), a **fixed-point iteration** recomputes $a[f_\theta]$ at each epoch, as the flow depends on f.

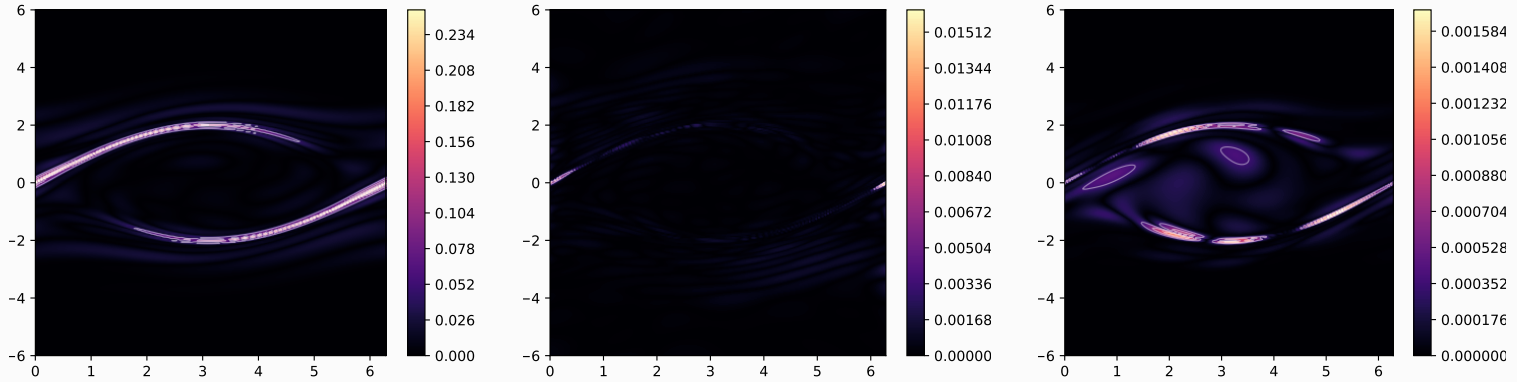
Back to the linear Vlasov equation

Same test as for the neural SL method: $\partial_t f + v\,\partial_x f + \sin(x)\,\partial_v f = 0, \quad t \in [0, 4.5],\ x \in [0, 2\pi],\ v \in [-6, 6]$

prediction at t = 4.5



error at t = 4.5



PINN | neural SL | Lagrangian PINN

⇝ 12 submaps, 550 dofs each, vs NSL with 884 dofs

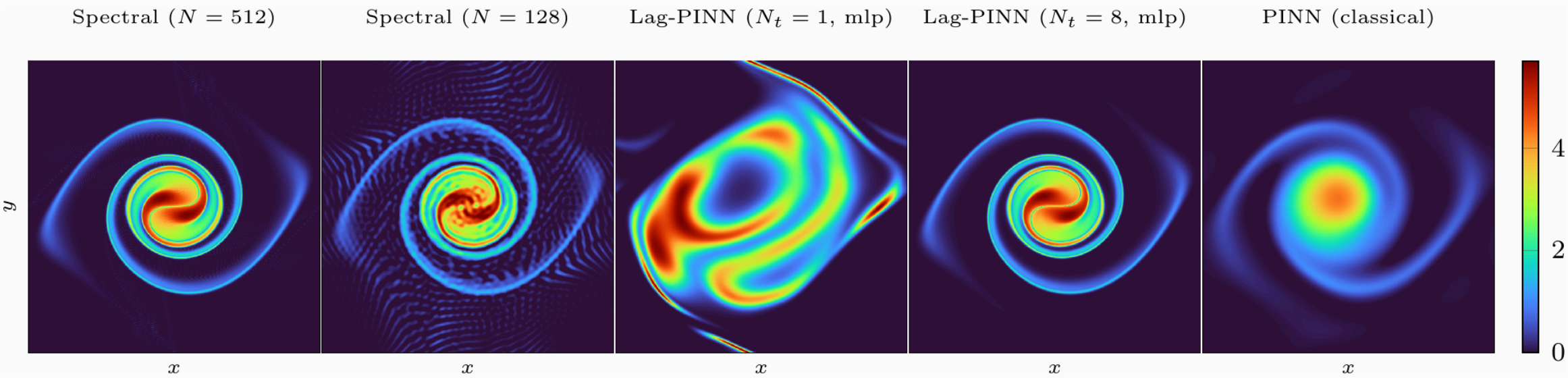
⇝ ~90s vs ~30s CPU time

⇝ but:

- L^2 error 3 times smaller
- L^∞ error 10 times smaller
- error less concentrated around the filaments

Incompressible Euler equations: vortex merger

MLP submaps (2370 parameters each); velocity obtained from an FFT Poisson solve on a 128^2 grid.



With $N_t = 8$ submaps, the Lagrangian PINN is very close to the 512^2 spectral reference, despite using only 2370 parameters per submap (MLPs) and a 128^2 FFT Poisson solve for the velocity.

⇝ We manage to reduce the required number of dofs.

5. The neural δf particle-in-cell method

[Fournet, Campos Pinto, Franck & M.-D., preprint, 2026]

Particle-in-cell (PIC) and δf PIC methods for the Vlasov-Poisson equations

In a **particle-in-cell (PIC)** method, f is represented by N_p numerical particles $z_k = (x_k, v_k)$, pushed along the characteristics and **deposited on a grid** to solve Poisson. The cost is linear in N_p, but there is **Monte Carlo noise**.

Particle-in-cell (PIC) and δf PIC methods for the Vlasov-Poisson equations

In a **particle-in-cell (PIC)** method, f is represented by N_p numerical particles $z_k = (x_k, v_k)$, pushed along the characteristics and **deposited on a grid** to solve Poisson. The cost is linear in N_p, but there is **Monte Carlo noise**.

The δf **method** splits f as $f = f_0 + \delta f$: the bulk f_0 is computed once, and only δf is carried by the particles.

Particle-in-cell (PIC) and δf PIC methods for the Vlasov-Poisson equations

In a **particle-in-cell (PIC)** method, f is represented by N_p numerical particles $z_k = (x_k, v_k)$, pushed along the characteristics and **deposited on a grid** to solve Poisson. The cost is linear in N_p, but there is **Monte Carlo noise**.

The δf **method** splits f as $f = f_0 + \delta f$: the bulk f_0 is computed once, and only δf is carried by the particles.

	full-f PIC	δf PIC
approximation of $f(t^n)$	particles only	f_0 + particles
bulk	none	static (e.g., an equilibrium)
Monte Carlo noise	$\|f\|/\sqrt{N_p}$	$\|f - f_0\|/\sqrt{N_p}$

⇝ A static bulk is not adequate when f changes a lot over time.

Particle-in-cell (PIC) and δf PIC methods for the Vlasov-Poisson equations

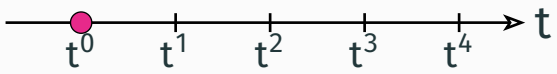
In a **particle-in-cell (PIC)** method, f is represented by N_p numerical particles $z_k = (x_k, v_k)$, pushed along the characteristics and **deposited on a grid** to solve Poisson. The cost is linear in N_p, but there is **Monte Carlo noise**.

The δf **method** splits f as $f = f_0 + \delta f$: the bulk f_0 is computed once, and only δf is carried by the particles.

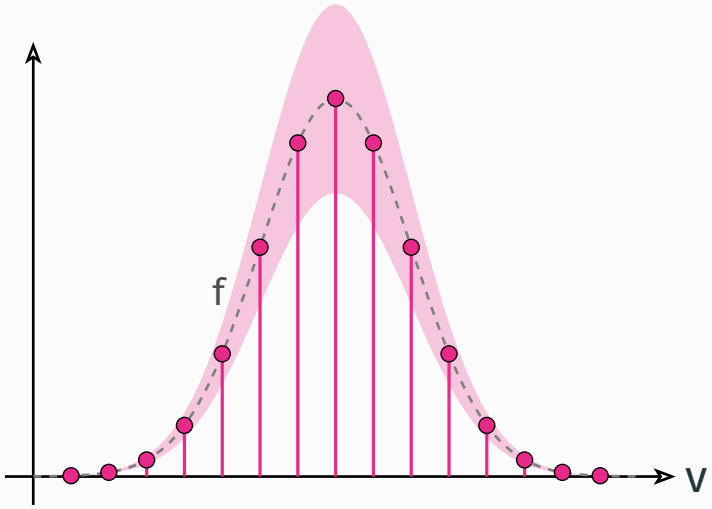
	full-f PIC	δf PIC	δf **PIC with evolving bulk**
approximation of $f(t^n)$	particles only	f_0 + particles	f_0^m + particles, $m = \lfloor n/L \rfloor$
bulk	none	static (e.g., an equilibrium)	**updated every L steps**
Monte Carlo noise	$\|f\|/\sqrt{N_p}$	$\|f - f_0\|/\sqrt{N_p}$	$\|f - f_0^m\|/\sqrt{N_p}$

⇝ A static bulk is not adequate when f changes a lot over time.

What do the numerical particles represent?



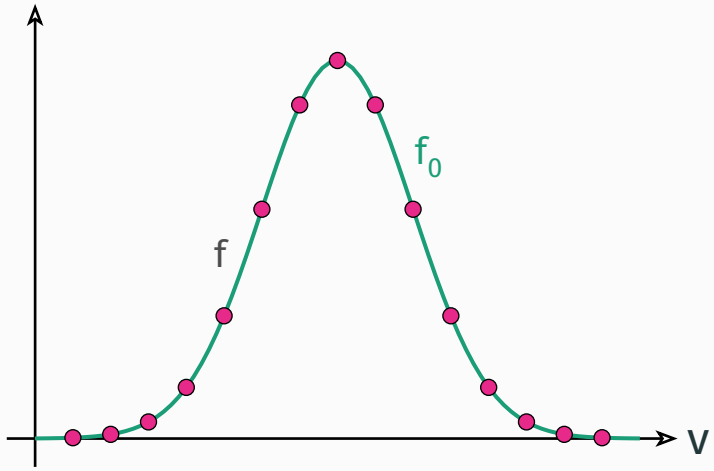
full-f PIC



the particles represent f

noise $\propto \|f\| / \sqrt{N_p}$

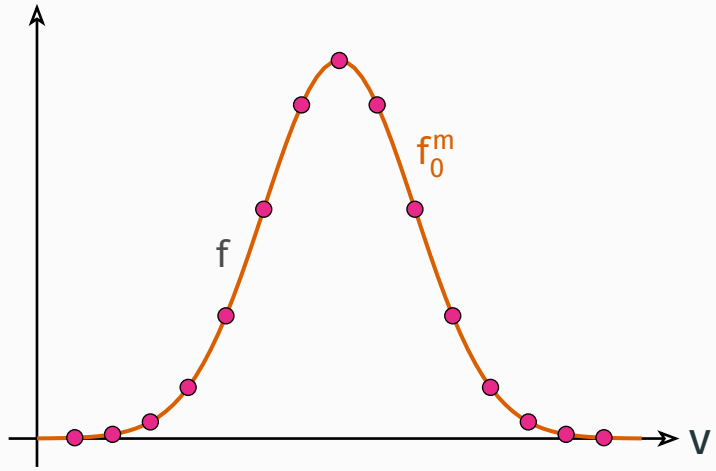
δf PIC: static bulk f_0



the particles represent $f - f_0$

noise $\propto \|f - f_0\| / \sqrt{N_p}$

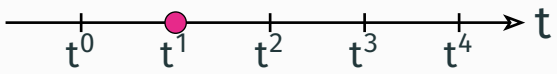
δf PIC: evolving bulk f_0^m



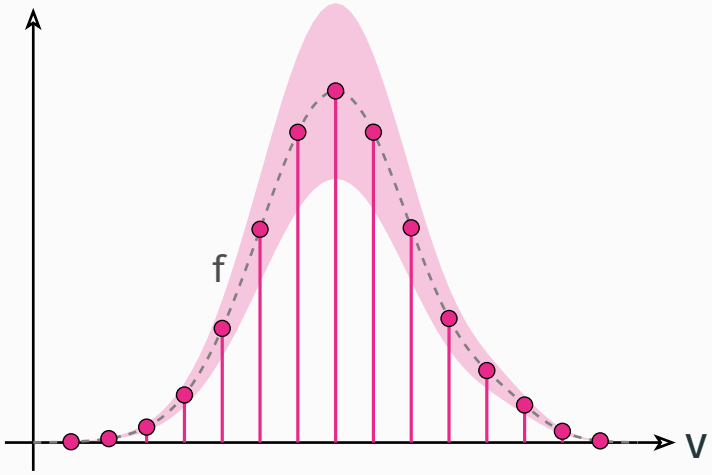
the particles represent $f - f_0^m$

noise $\propto \|f - f_0^m\| / \sqrt{N_p}$

What do the numerical particles represent?



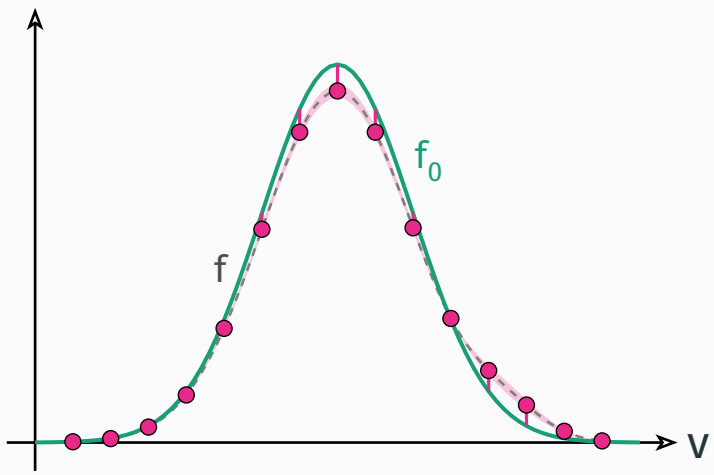
full-f PIC



the particles represent f

noise $\propto \|f\| / \sqrt{N_p}$

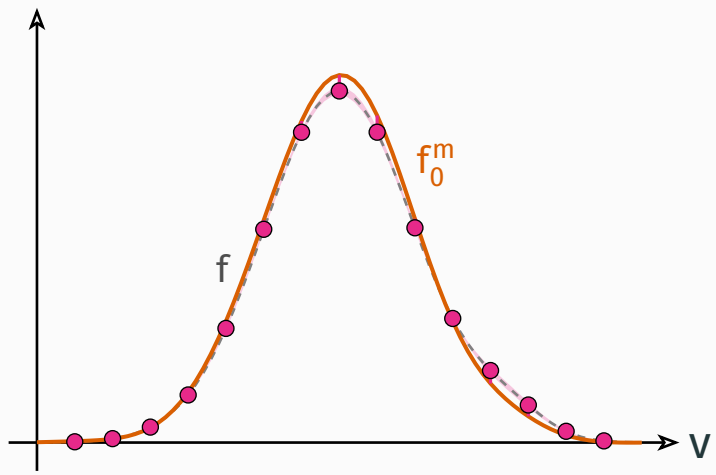
δf PIC: static bulk f_0



the particles represent $f - f_0$

noise $\propto \|f - f_0\| / \sqrt{N_p}$

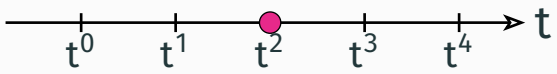
δf PIC: evolving bulk f_0^m



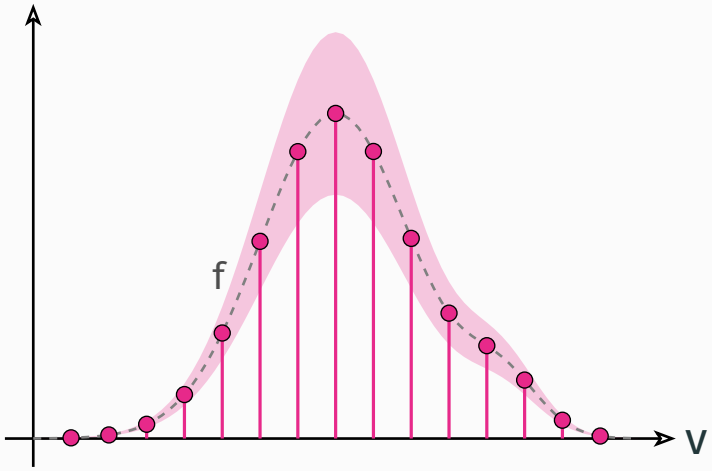
the particles represent $f - f_0^m$

noise $\propto \|f - f_0^m\| / \sqrt{N_p}$

What do the numerical particles represent?



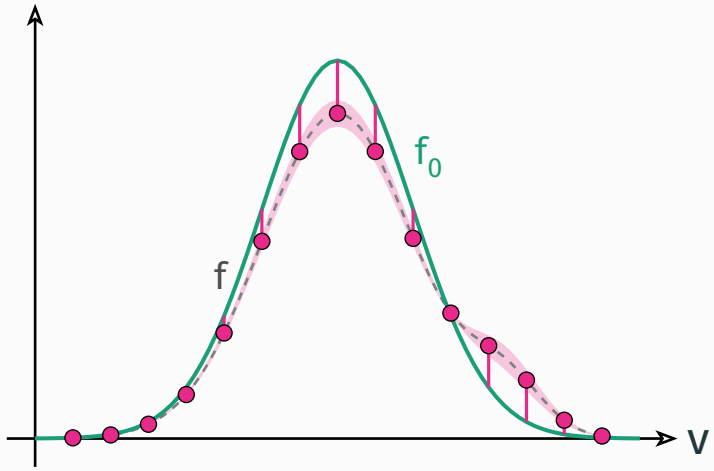
full-f PIC



the particles represent f

noise $\propto \|f\| / \sqrt{N_p}$

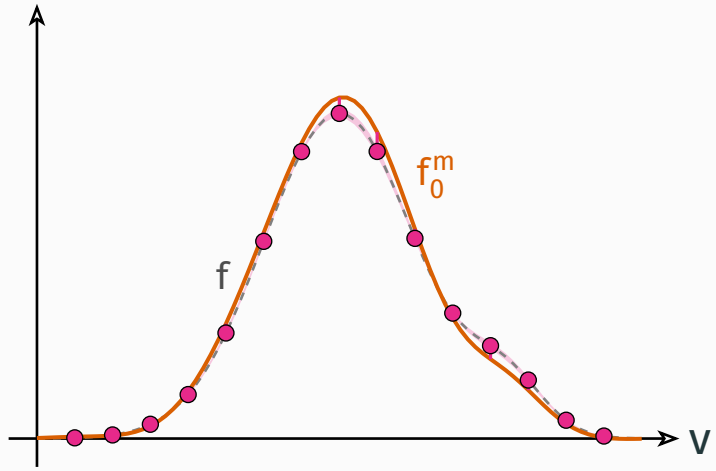
δf PIC: static bulk f_0



the particles represent $f - f_0$

noise $\propto \|f - f_0\| / \sqrt{N_p}$

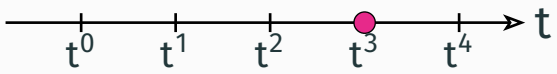
δf PIC: evolving bulk f_0^m



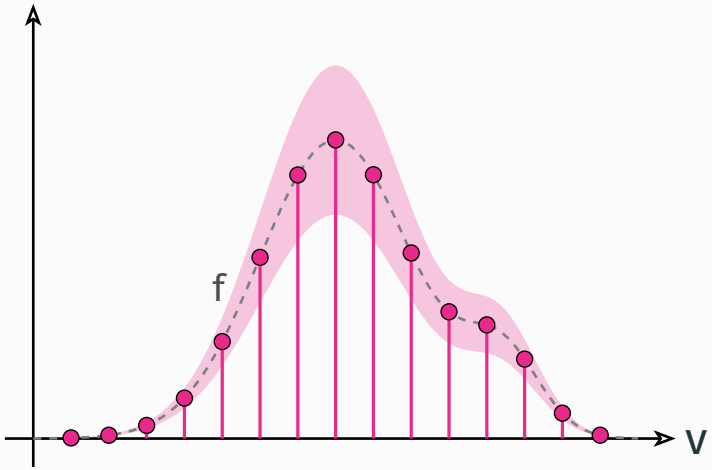
the particles represent $f - f_0^m$

noise $\propto \|f - f_0^m\| / \sqrt{N_p}$

What do the numerical particles represent?



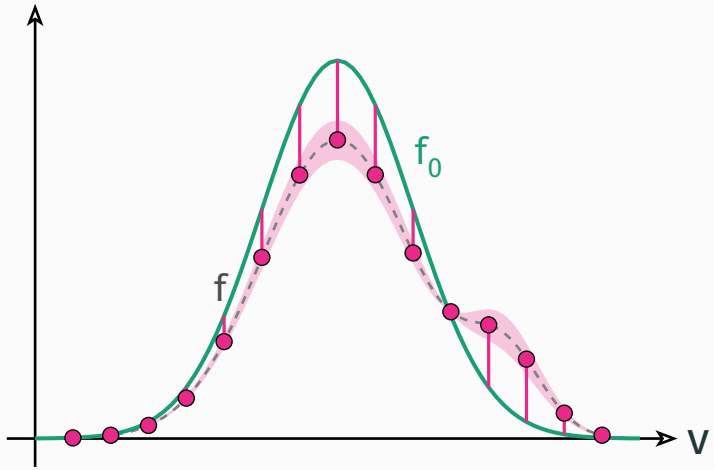
full-f PIC



the particles represent f

noise $\propto \|f\| / \sqrt{N_p}$

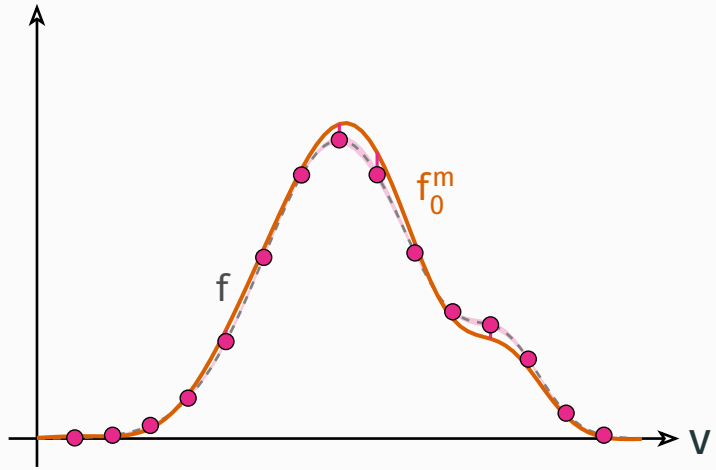
δf PIC: static bulk f_0



the particles represent $f - f_0$

noise $\propto \|f - f_0\| / \sqrt{N_p}$

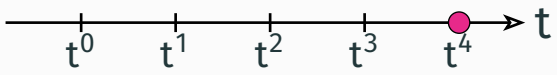
δf PIC: evolving bulk f_0^m



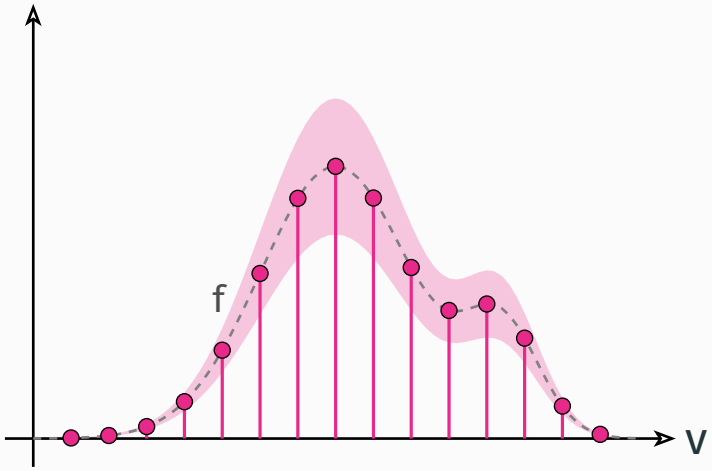
the particles represent $f - f_0^m$

noise $\propto \|f - f_0^m\| / \sqrt{N_p}$

What do the numerical particles represent?



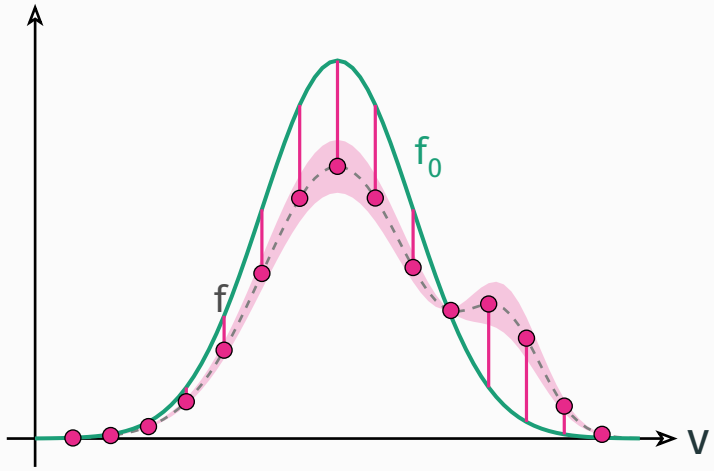
full-f PIC



the particles represent f

noise $\propto \|f\| / \sqrt{N_p}$

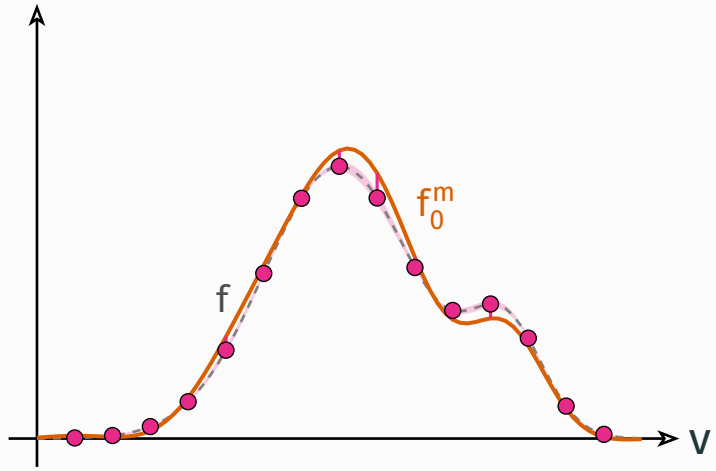
δf PIC: static bulk f_0



the particles represent $f - f_0$

noise $\propto \|f - f_0\| / \sqrt{N_p}$

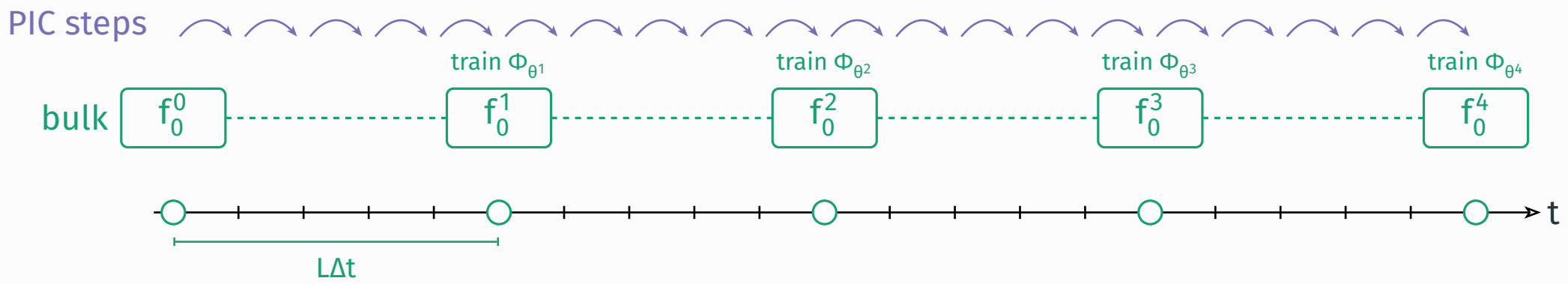
δf PIC: evolving bulk f_0^m



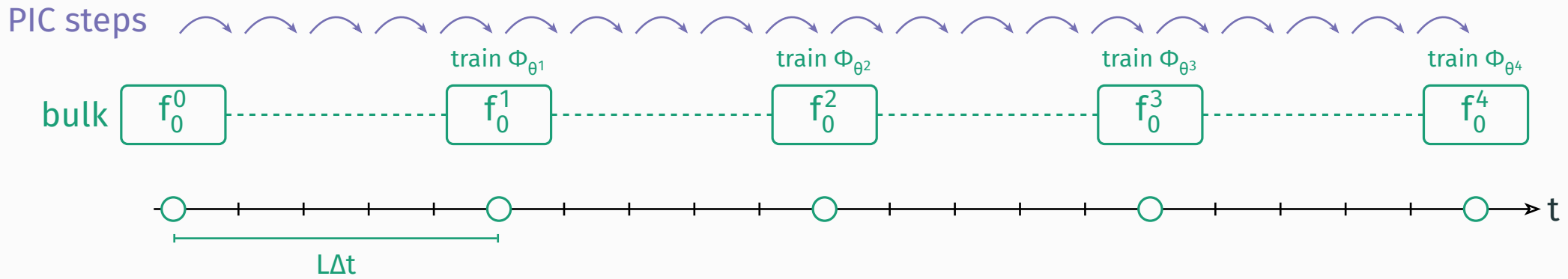
the particles represent $f - f_0^m$

noise $\propto \|f - f_0^m\| / \sqrt{N_p}$

Representing the evolving bulk with a neural network



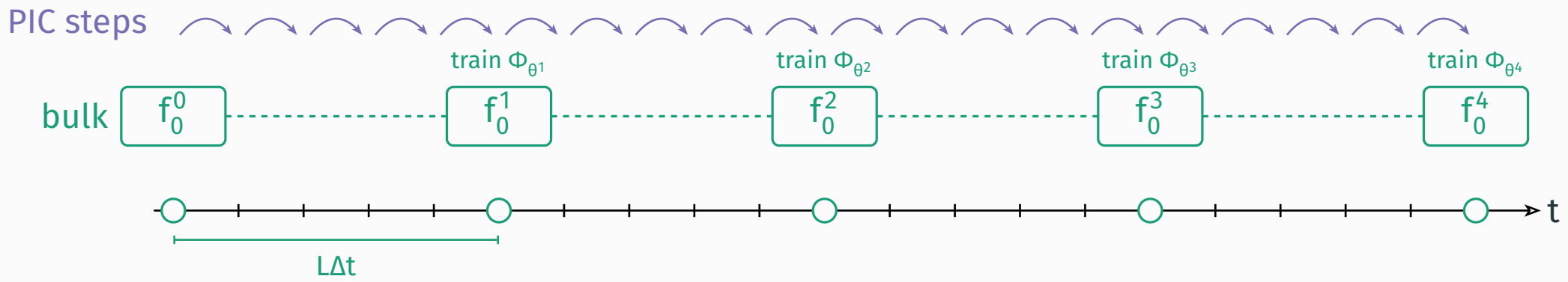
Representing the evolving bulk with a neural network



The particles follow the characteristics, so the **backward flow** maps them back in time: $z_k^{n-L} = \Phi\left(z_k^n; t^{n-L}, t^n\right)$.
We fit this flow with a neural network every L time steps, and then use it to transport the bulk:

$$\theta^m \in \operatorname*{argmin}_\theta \frac{1}{N_p} \sum_{k=1}^{N_p} \left\| \Phi_\theta\left(z_k^{mL}\right) - z_k^{(m-1)L} \right\|^2, \qquad f_0^m = f_0^{m-1} \circ \Phi_{\theta^m} = f_0^0 \circ \Phi_{\theta^1} \circ \cdots \circ \Phi_{\theta^m}.$$

Representing the evolving bulk with a neural network



The particles follow the characteristics, so the **backward flow** maps them back in time: $z_k^{n-L} = \Phi\left(z_k^n; t^{n-L}, t^n\right)$.
We fit this flow with a neural network every L time steps, and then use it to transport the bulk:

$$\theta^m \in \operatorname*{argmin}_\theta \frac{1}{N_p} \sum_{k=1}^{N_p} \left\| \Phi_\theta\left(z_k^{mL}\right) - z_k^{(m-1)L} \right\|^2, \qquad f_0^m = f_0^{m-1} \circ \Phi_{\theta^m} = f_0^0 \circ \Phi_{\theta^1} \circ \cdots \circ \Phi_{\theta^m}.$$

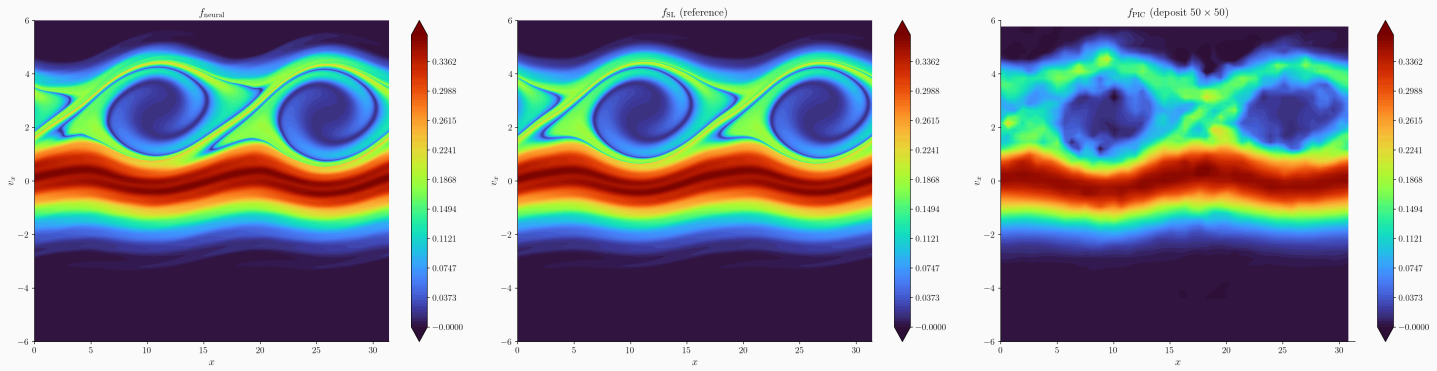
In practice, each Φ_θ is a small **periodic SympNet** [Jin et al., 2020].
Composing with f_0^0 keeps the bulk **nonnegative**, and since each Φ_θ is **symplectic**, it **conserves mass**, for any θ.

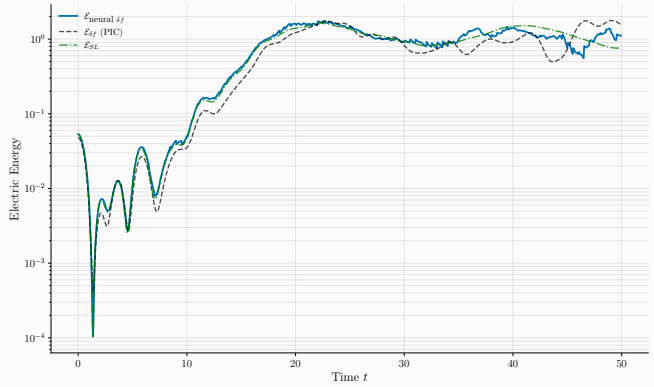
1D1V bump-on-tail instability

neural δf PIC: learned bulk f_0

SL reference (1024^2)

standard δf PIC: particle deposit





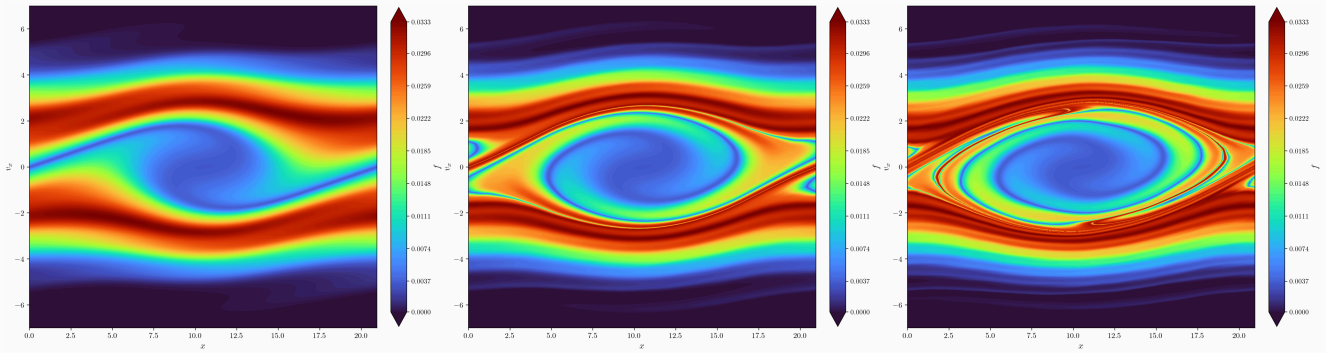
electric energy: neural δf PIC, standard δf PIC, reference

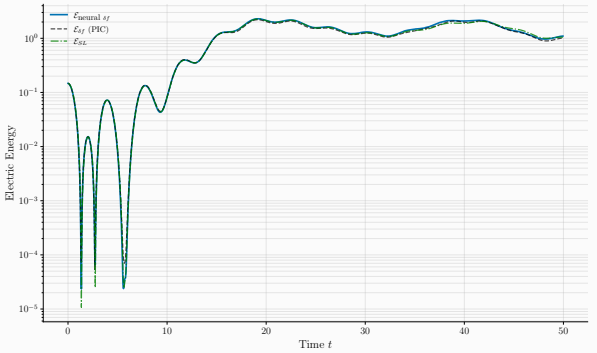
particles	$N_p = 10^4$
bulk updates	every L = 10 steps
composed networks	22 for 100 bulk updates
network size	480 parameters
MC error	**divided by about 3** (like 10 × more particles)

3D3V two-stream instability

$f(x, v_x)$ (cut at $y = z = 0$, $v_y = v_z = 0$)

t = 15 t = 30 t = 45





particles	$N_p = 10^6$, on a single GPU
bulk updates	every L = 10 steps
composed networks	13 for 99 bulk updates
network size	960 parameters
MC error	**divided by about 3** (like 10 × more particles)

electric energy:
neural δf PIC,
standard δf PIC,
BSL6D [Schild et al., 2024]

6. Conclusion and outlook

Three neural methods for advection-dominated problems

	neural semi-Lagrangian	Lagrangian PINNs based on flow maps	neural δf particle-in-cell
the network approximates	the solution $f(t^n, \cdot)$, with new dofs θ^n at each step	the flow map $\mathcal{X}(s; t, \cdot)$ on each time sub-interval	the backward flow between two bulk updates
optimization	fitting $f_{\theta^n} \circ \tilde{\mathcal{X}}$ (no PDE residual)	minimizing the PDE residual of the flow map	regression on the trajectories of PIC particles
nonlinear problems	electric field given by a PINN	fixed-point iteration on the field	standard PIC field solver
reference	CMAME, 2026	preprint, 2026	preprint, 2026

The last two methods learn a flow and recover the solution by composing the initial condition with this flow.

Conclusion and outlook

Conclusion:

neural SL:
- fully meshless
- stable without a time step condition

Lagrangian PINNs:
- learn the flow instead of the solution
- more accurate than PINNs

neural δf PIC:
- bulk evolved by SympNets trained on the PIC particles
- MC error divided by 2 to 3

Limitations and perspectives:
- in low dimension, these methods are **more expensive** than grid-based or standard PIC solvers;
- the cost of composed flows grows linearly with their number;
- next models to tackle: **gyrokinetic models** (5D phase space).

These methods are implemented in **Scimba**: **https://www.scimba.org**.

Thank you for your attention!