FROM FIXED GRIDS TO MOVING PARTICLES: A TRANSFERABLE LATENT OPERATOR FOR FLUID DYNAMICS

Anonymous authors
Paper under double-blind review

ABSTRACT

Lagrangian modeling is vital to fluid dynamics, as it characterizes particle transport and complements the Eulerian description. However, Lagrangian trajectories are less commonly available than Eulerian fields, while most neural operators are trained and evaluated primarily in the Eulerian representation. This mismatch motivates a new learning problem: can a model trained solely on Eulerian observations generalize zero-shot from Eulerian field prediction to Lagrangian particle rollout, without Lagrangian supervision or task-specific adaptation? To address this problem, we propose the *Transferable Latent Operator* (TLO), which learns a unified flow representation shared by Eulerian field prediction and Lagrangian particle rollout. TLO decouples latent flow evolution from coordinate-dependent decoding: querying the evolving latent representation at fixed spatial coordinates yields Eulerian fields, whereas querying velocities at particle positions and recursively updating these positions enables Lagrangian rollout. Across five fluid-dynamics benchmarks, TLO achieves strong Eulerian forecasting accuracy and competitive zero-shot Lagrangian rollout, with further gains from decoder fine-tuning using Lagrangian supervision.

1 INTRODUCTION

Lagrangian particle trajectories provide a direct description of transport, mixing, and dispersion in fluid flows. However, most fluid-learning datasets are available as time-dependent fields on fixed Eulerian grids (Lu et al., 2021; Li et al., 2021; Takamoto et al., 2022; Ohana et al., 2024). Native Lagrangian data are less commonly available because they require specifying and tracking particle populations. At high particle counts, trajectory generation, storage, and particle-based training can also become computationally and memory intensive (Sanchez-Gonzalez et al., 2020; Toshev et al., 2023; Rochman-Sharabi et al., 2025). This mismatch motivates a practical question: can a model trained only on fixed-grid Eulerian data also support Lagrangian particle rollout of the same flow?

We formulate this problem as *zero-shot Eulerian-to-Lagrangian generalization*. A model is trained exclusively with Eulerian field supervision, without particle coordinates, particle velocities, or trajectory labels. At test time, the same model is queried at moving particle positions, and its predicted velocities are recursively integrated without parameter adaptation. This is more than ordinary arbitrary-coordinate evaluation: each velocity prediction changes the next particle position and therefore determines the next query location. Errors can consequently induce a history-dependent query shift and accumulate into trajectory drift.

Physics-based simulators, including differentiable ones, can already handle new initial conditions without retraining, but repeated high-fidelity forecasts still entail numerical solution of the governing dynamics (Hu et al., 2019; Holl et al., 2020). Learned surrogates aim to reduce this recurring computational cost by approximating flow evolution with trained operator evaluations, amortizing offline training across repeated predictions (Li et al., 2021). Existing methods have explored Eulerian–Lagrangian modeling; however, they do not directly address zero-shot Eulerian-to-Lagrangian rollout. Approaches based on latent particles, joint field–trajectory learning, or trajectory reconstruction (Wan et al., 2025; Thakur & Raissi, 2025; Alkin et al., 2024; Ma et al., 2024) typically require additional Lagrangian supervision or particle-specific modeling, preventing zero-shot transfer. Al-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

ternatively, Eulerian neural operators (Li et al., 2021; Tripura & Chakraborty, 2023; Rahman et al., 2022) can be combined with numerical interpolation to obtain particle velocities from predicted grid fields. However, this strategy remains tied to the resolution and finite support of the Eulerian discretization. Once particles move beyond the grid support, standard interpolation cannot continue without additional extrapolation rules.

These considerations motivate a neural operator that learns flow dynamics from Eulerian observations while supporting direct evaluation at recursively evolving particle locations. We therefore introduce the *Transferable Latent Operator* (TLO), which combines a shared latent flow representation with two query modes. An encoder maps fixed-grid Eulerian observations into spatially localized tokens augmented with global flow context, and a shared latent processor advances this representation independently of the output query set. A coordinate-conditioned decoder then evaluates the predicted field at requested locations. Querying on the fixed Eulerian grid produces an Eulerian field forecast, whereas querying the velocity channels at recursively advected particle positions produces a Lagrangian rollout. Particle coordinates are used only as decoder queries and are not required by the Eulerian training objective.

Our main contributions are summarized as follows:

- We formulate and evaluate *zero-shot Eulerian-to-Lagrangian generalization*, in which a model trained only with fixed-grid Eulerian supervision performs closed-loop particle rollout without particle supervision or parameter adaptation.
- We propose the *Transferable Latent Operator*, whose output-query-independent latent evolution and coordinate-conditioned decoder allow the same trained model to support both fixed-grid forecasting and moving particle queries.
- Across five Eulerian benchmarks and three particle-rollout benchmarks, TLO achieves strong performance, compares favorably with neural-operator and native particle-based baselines, and remains evaluable beyond the fixed Eulerian grid support.

2 Related Work

Neural operators. Neural operators learn mappings between function spaces for PDE modeling (Lu et al., 2021; Li et al., 2024; Wen et al., 2026). Representative architectures include spectral operators, coordinate-based models, and attention- or token-based operators (Li et al., 2021; Hao et al., 2023; Li et al., 2023; Wu et al., 2024; Alkin et al., 2024; Cao, 2021). Although several of these models support arbitrary-coordinate evaluation, they are generally evaluated on fixed or externally specified query locations. We instead consider moving queries whose positions are recursively generated by the model's predicted velocities.

Eulerian–Lagrangian modeling. Classical particle-grid methods couple Eulerian fields and Lagrangian particles through transfers between the two representations (Brackbill & Ruppel, 1986; Sulsky et al., 1994). Recent neural methods use latent particles to improve Eulerian forecasting (Ma et al., 2024), infer fields or trajectories from particle observations (Wan et al., 2025; Thakur & Raissi, 2025), or learn particle-based simulators (Rochman-Sharabi et al., 2025; Xiong et al., 2023). In contrast, the zero-shot version of TLO is trained only on fixed-grid Eulerian fields, with particle positions introduced at inference time solely as decoder queries. The same learned flow representation therefore supports both Eulerian field prediction and closed-loop Lagrangian rollout. Sparse decoder adaptation is studied separately as a non-zero-shot extension.

Differentiable and learned simulators. Learned particle simulators predict particle-state updates from data (Sanchez-Gonzalez et al., 2020), whereas differentiable physics-based simulators solve prescribed dynamics numerically and provide gradients through the simulation (Hu et al., 2019; Holl et al., 2020). TLO instead learns Eulerian flow evolution and numerically advects particles through the predicted velocities, requiring no Lagrangian supervision or adaptation in the zero-shot setting.

3 PROBLEM SETTING

We consider time-dependent flows on a physical domain $\Omega \subset \mathbb{R}^d$, with prescribed initial and boundary conditions. The Eulerian state $\boldsymbol{u}_t : \Omega \to \mathbb{R}^{d_u}$ describes physical quantities at fixed spatial locations, whereas Lagrangian trajectories follow particles transported by the velocity $\boldsymbol{v}_t = \Pi_v \boldsymbol{u}_t$, where Π_v extracts the d velocity channels. Eulerian observations $\mathbf{U}_t \in \mathbb{R}^{N_E \times d_u}$ sample the state at fixed coordinates $X_E = \{\boldsymbol{x}_i\}_{i=1}^{N_E}$. Here, t indexes snapshots separated by Δt, and hats denote predictions.

Figure 1 illustrates the shared workflow of TLO. The encoder and processor map the current Eulerian observation to a next-step latent flow representation. The coordinate-conditioned decoder then evaluates the predicted field at a requested query set $Q = \{\boldsymbol{q}_j\}_{j=1}^{N_Q} \subset \Omega$:

$$\widehat{\mathbf{U}}_{t+1}(Q) = \mathcal{G}_\theta(X_E, \mathbf{U}_t; Q). \tag{1}$$

The query coordinates need not coincide with X_E, allowing the same decoder to evaluate both fixed-grid fields and velocities at moving particle positions.

Eulerian-only training. We train on consecutive field pairs $(\mathbf{U}_t, \mathbf{U}_{t+1})$, setting $Q = X_E$ and minimizing the mean squared error between predicted and reference next-step grid fields. No particle positions, particle-velocity labels, or trajectories are used for training.

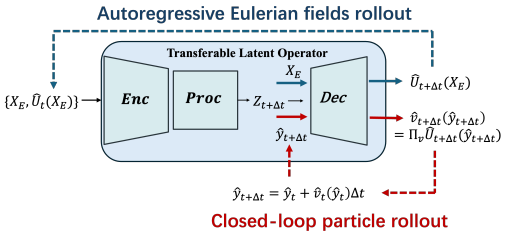

Figure 1: Dual-mode rollout of the Eulerian-trained TLO. The blue loop feeds predicted grid fields back as inputs. The red loop queries velocities at particle positions and updates subsequent queries through numerical integration. Both loops share the same flow representation.

Dual-mode inference. Starting from the observed state $\mathbf{U}_0$, Eulerian forecasting repeatedly queries X_E and feeds each predicted grid field back as the next input (blue loop). For Lagrangian rollout, the same evolving representation is additionally queried at the current positions $\widehat{\boldsymbol{y}}_{i,t}$ of N_P particles. Numerical integration of the queried velocities, with the prescribed boundary handling, produces the next particle positions and hence the next queries (red loop). Particle positions affect only the decoder queries; the flow state continues to advance through the Eulerian loop. Both rollouts use the same Eulerian-trained parameters without adaptation. A notation summary and the detailed training and rollout formulations are provided in Appendix A.

Field and trajectory errors. Since Lagrangian rollout is driven by velocities queried from the predicted flow, fixed-grid field error alone does not fully characterize model performance. We therefore distinguish errors in Eulerian field prediction, velocity evaluation at particle positions, and the resulting closed-loop trajectories. We refer to these quantities as *Eul*, *Ref*, and *Path*, respectively, and report them throughout the experiments.

Let T denote the rollout horizon, N_E the number of Eulerian grid points, and N_P the number of particles. The state and spatial dimensions are d_u and d, respectively. First, the fixed-grid Eulerian error for autoregressive field prediction on X_E is defined as

$$\mathcal{E}_{\mathrm{Eul}} := \frac{1}{TN_E d_u} \sum_{t=1}^{T} \sum_{\boldsymbol{g} \in X_E} \|\widehat{\boldsymbol{u}}_t(\boldsymbol{g}) - \boldsymbol{u}_t(\boldsymbol{g})\|_2^2 . \tag{2}$$

To evaluate velocity without position feedback, we query at the reference positions $\boldsymbol{y}_{i,t}$:

$$\mathcal{E}_{\mathrm{Ref}} := \frac{1}{TN_P d} \sum_{t=1}^{T} \sum_{i=1}^{N_P} \|\widehat{\boldsymbol{v}}_t(\boldsymbol{y}_{i,t}) - \boldsymbol{v}_t(\boldsymbol{y}_{i,t})\|_2^2 . \tag{3}$$

The Eulerian field is still predicted autoregressively, but the query coordinates are reset to the reference pathline at every step. Hence, $\mathcal{E}_{\mathrm{Ref}}$ measures velocity accuracy along the correct trajectory without propagating particle-position errors.

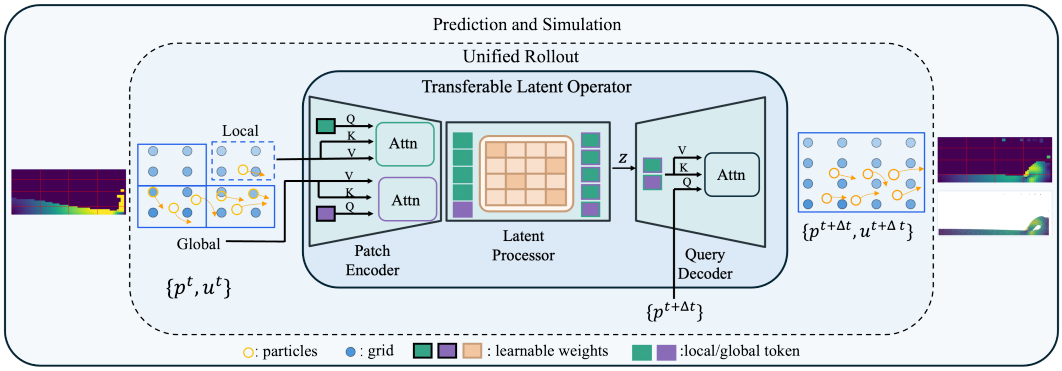

Figure 2: Overview of TLO: a local–global encoder, a query-independent latent processor, and a coordinate-conditioned decoder for Eulerian forecasting and Lagrangian rollout.

Finally, in the closed-loop rollout, the particle positions $\widehat{\boldsymbol{y}}_{i,t}$ are recursively generated using the model's queried velocities. The resulting trajectory error is

$$\mathcal{E}_{\text{Path}} := \frac{1}{TN_P d} \sum_{t=1}^{T} \sum_{i=1}^{N_P} d_\Omega^2(\widehat{\boldsymbol{y}}_{i,t}, \boldsymbol{y}_{i,t}) \,, \tag{4}$$

where d_Ω accounts for the domain geometry and uses the minimum-image distance on periodic domains.

4 Transferable Latent Operator

As illustrated in Figure 2, TLO follows an encode–process–decode architecture (Sanchez-Gonzalez et al., 2020) that separates latent flow modeling from coordinate-dependent field evaluation, with the one-step operator factorized as

$$\boldsymbol{Z}_t^0 = \mathcal{E}_\theta(X_E, \mathbf{U}_t), \qquad \boldsymbol{Z}_t^L = \mathcal{T}_\theta(\boldsymbol{Z}_t^0), \widehat{\mathbf{U}}_{t+1}(Q) \quad = \mathcal{D}_\theta(\boldsymbol{Z}_t^L, Q). \tag{5}$$

The encoder constructs a latent representation of the observed flow, the processor advances this representation, and the decoder evaluates the predicted state at the requested coordinates. Importantly, the encoder and latent processor are independent of Q; only the final readout depends on the output coordinates. Setting $Q = X_E$ produces the next Eulerian field, whereas setting Q to the current particle positions provides the velocities used for Lagrangian rollout. We now describe the architecture of TLO. For notational simplicity, we omit the time index in the processor equations.

Patch Encoder. To capture both localized transport dynamics and long-range flow dependencies, we construct a hierarchical domain representation consisting of local patches Ω_p and the global domain Ω_g:

$$\mathcal{D} = \{\underbrace{\Omega_1, \ldots, \Omega_P}_{\text{local patches}}, \underbrace{\Omega_g}_{\text{global context}}\}, \qquad \Omega_g = \bigcup_{p=1} \Omega_p.$$

Each observation is lifted into a coordinate–state feature:

$$\boldsymbol{r}_i = \phi[\text{PE}(\boldsymbol{x}_i), \text{PE}(\boldsymbol{x}_i^p), \boldsymbol{u}_t(\boldsymbol{x}_i), e_p] \,,$$

where ϕ is a learned lifting map, PE is a fixed Fourier encoding, $\boldsymbol{x}_i^p$ is the patch-relative coordinate, and e_p is a learnable patch embedding. For each $* \in \{1, \ldots, P, g\}$, cross-attention pools the feature set$R_* = \{\mathbf{r}_i \mid \mathbf{x}_i \in \Omega_*\}$ into a fixed number of latent tokens:

$$\boldsymbol{Z}_* = \text{Attn}(\boldsymbol{Q}^*, K(R_*), V(R_*)) \,,$$

using learnable queries $\boldsymbol{Q}^l$ for local patches and $\boldsymbol{Q}^g$ for global aggregation. The resulting local and global tokens are concatenated to form the initial latent state $\boldsymbol{Z}_t^0$.

By combining local coordinate encoding with global context aggregation, the hierarchical Patch Encoder maps coordinate-attached observations to a discretization-independent latent representation

that captures both local transport structures and global flow dependencies, enabling queries on fixed Eulerian grids or evolving Lagrangian trajectories.

Latent Processor. We update the encoded flow tokens using L attention-based blocks with slice-based aggregation inspired by Transolver (Wu et al., 2024). At layer ℓ, the latent tokens are flattened as $\boldsymbol{Z}^\ell \in \mathbb{R}^{N\times D}$, where N denotes the total number of latent tokens. The processor first assigns tokens into M adaptive latent slices:

$$\boldsymbol{W}^\ell = \mathrm{softmax}(\psi(\boldsymbol{Z}^\ell)) \in \mathbb{R}^{N\times M},$$

where softmax is applied over the M slices for each token and each row of $\boldsymbol{W}^\ell$ represents the contribution of a token to the latent slices. The tokens are pooled into slice representations by normalized weighted aggregation, followed by self-attention:

$$\boldsymbol{S}^\ell = \mathrm{Attn}(\hat{\boldsymbol{S}}^\ell),\ \hat{\boldsymbol{S}}^\ell = (\widetilde{\boldsymbol{W}}^\ell)^\top \boldsymbol{Z}^\ell,$$

where $\widetilde{W}^\ell_{i,j} = W^\ell_{i,j} / \sum_\alpha W^\ell_{\alpha,j}$. The updated slice information is distributed back to the latent tokens:

$$\boldsymbol{Z}^{\ell+1} = \boldsymbol{Z}^\ell + \mathrm{Proj}(\boldsymbol{W}^\ell \boldsymbol{S}^\ell) + \alpha\,\mathrm{FFN}(\boldsymbol{Z}^\ell).$$

Here, α controls the contribution of the feed-forward branch.

Query Decoder. For each query $\boldsymbol{q}_j$, the decoder identifies its patch Ω_p and constructs a coordinate feature:

$$\boldsymbol{g}_j = \varphi[\mathrm{PE}(\boldsymbol{q}_j), \mathrm{PE}(\boldsymbol{q}_j^p)],$$

using the previous embedding strategy. The physical quantity at the query location is recovered through cross-attention between the query feature and latent tokens from neighboring patches:

$$\hat{\boldsymbol{u}}(\boldsymbol{q}_j) = f_{out}\Big(\sum_{p_k\in\mathcal{N}(\boldsymbol{q}_j)} \alpha_k\,\mathrm{Attn}\left(Q(\boldsymbol{g}_j), K(\boldsymbol{Z}_{p_k}), V(\boldsymbol{Z}_{p_k})\right)\Big),$$

where $\mathcal{N}(\boldsymbol{q}_j)$ denotes the set of neighboring patches around the patch containing $\boldsymbol{q}_j$, and α_k represents the aggregation weight of each neighboring patch.

5 Error Analysis

TLO predicts flow states whose velocity channels drive Lagrangian particle rollout. Consider particle i over a rollout of T steps, indexed by $t = 0, \ldots, T$. Let Δt be the integration step, and let $\boldsymbol{v}_t$ and $\widehat{\boldsymbol{v}}_t$ denote the reference and predicted velocity fields at step t, respectively. The corresponding reference and predicted particle positions are denoted by $\boldsymbol{y}_{i,t}$ and $\widehat{\boldsymbol{y}}_{i,t}$.

Using integration scheme, their positions evolve as

$$\boldsymbol{y}_{i,t+1} = \mathcal{B}(\boldsymbol{y}_{i,t} + \Delta t\,\boldsymbol{v}_t(\boldsymbol{y}_{i,t}))\,,\quad \widehat{\boldsymbol{y}}_{i,t+1} = \mathcal{B}(\widehat{\boldsymbol{y}}_{i,t} + \Delta t\,\widehat{\boldsymbol{v}}_t(\widehat{\boldsymbol{y}}_{i,t}))\,,\ t = 0, \ldots, T-1. \tag{6}$$

where $\mathcal{B}$ is the bounce operator that handles particles that reach the boundary of the physical domain during rollout. Define the position error and the velocity error evaluated at the reference particle position as

$$\delta_{i,t} := d_\Omega(\widehat{\boldsymbol{y}}_{i,t}, \boldsymbol{y}_{i,t}),\ r_{i,t} := \|\widehat{\boldsymbol{v}}_t(\boldsymbol{y}_{i,t}) - \boldsymbol{v}_t(\boldsymbol{y}_{i,t})\|_2\,. \tag{7}$$

Proposition 1 (Velocity error to trajectory error)**.** *Assume that $\widehat{\boldsymbol{y}}_{i,0} = \boldsymbol{y}_{i,0}$, that $\mathcal{B}$ is non-expansive under d_Ω, and that $\widehat{\boldsymbol{v}}_t$ is $\widehat{L}_t$-Lipschitz over the region visited by the two particles. Then, for every $t = 0, \ldots, T-1$,*

$$\delta_{i,t+1} \le (1 + \Delta t\,\widehat{L}_t)\delta_{i,t} + \Delta t\,r_{i,t}. \tag{8}$$

Consequently, the position error at the rollout horizon T satisfies

$$\delta_{i,T} \le \Delta t \sum_{s=0}^{T-1} r_{i,s} \prod_{k=s+1}^{T-1} (1 + \Delta t\,\widehat{L}_k), \tag{9}$$

with the convention that an empty product equals one.

Table 1: Benchmark configurations. Eul/Lag: Eulerian resolution and particle count; Int: integration scheme; T: rollout horizon; Eva: Eul/Ref/Path denote field, reference-query, and trajectory evaluation.

Dataset	Native	Eul	Vars	Lag	Int	Split	BC	T	Eva
DAM2D	Lag SPH	32×64	2	5,000	EE	50/25/25	No-Slip	10	Eul/Ref/Path
NS2D	Eul Grid	64^2	3	1,024	RK4	960/120/120	Periodic	10	Eul
TGV3D	Lag SPH	20^3	3	8,000	EE	50/10/10	Periodic	5	Eul
Burgers3D	Eul Grid	32^3	3	4,096	RK4	70/15/15	Periodic	5	Eul/Ref/Path
SEA	Eul Grid	180×300	5	1,024	RK4	2700/300/600	Open	10	Eul/Ref/Path

Proposition 1 relates velocity prediction errors to trajectory errors in a closed-loop rollout. The term $\Delta t\, r_{i,t}$ bounds the contribution of velocity error at the reference position to the one-step position error. Once the predicted particle deviates from the reference path, subsequent velocities are queried at different locations. The factor $1+\Delta t\, \widehat{L}_t$ bounds the resulting amplification of the existing position error. Earlier velocity errors can affect more subsequent query locations, and their contributions may accumulate over a long rollout. Thus, a small *Ref* error does not by itself guarantee a small *Path* error, motivating the use of both metrics.

6 Experiments

Benchmarks. We evaluate TLO on five benchmarks covering particle simulations, grid-based PDEs, and real-world ocean currents. DAM2D and TGV3D are smoothed particle hydrodynamics (SPH) datasets from LagrangeBench (Toshev et al., 2023), converted to Eulerian fields by kernel splatting. NS2D, Burgers3D, and SEA provide grid-based velocity fields from Navier–Stokes dynamics, advection–diffusion, and ocean reanalysis, respectively (Kovachki et al., 2023; Koehler et al., 2024; Ma et al., 2024). DAM2D uses native trajectories, while Burgers3D and SEA use passive tracers generated from reference velocities with Parcels (Delandmeter & Van Sebille, 2019). All benchmarks use Eulerian evaluation; DAM2D, Burgers3D, and SEA additionally report Ref and Path. Table 1 summarizes the configurations.

Baselines. We compare with LSM (Wu et al., 2023), GINO (Li et al., 2023), GNOT (Hao et al., 2023), Transolver (Wu et al., 2024), UPT (Alkin et al., 2024), LNO (Wang & Wang, 2024), and DeepLag (Ma et al., 2024). All methods, including TLO, are trained with Eulerian-only supervision using the same one-step mean squared error objective:

$$\mathcal{L}_{\mathrm{Eul}} = \mathbb{E}_t\left[\frac{1}{N_E d_u}\left\|\mathcal{G}_\theta(X_E, \mathbf{U}_t; X_E) - \mathbf{U}_{t+1}\right\|_F^2\right].$$

At inference, predicted grid states are fed back autoregressively. We report Eul for field forecasting and Ref and Path for the particle-rollout comparisons, as defined in Section 3. For particle evaluation, models with coordinate-conditioned readouts query velocities directly at particle positions, whereas grid-output models use bilinear (2D) or trilinear (3D) interpolation of their predicted fields. To isolate the effect of the readout strategy, we additionally compare direct decoding and grid interpolation using the same TLO checkpoint. Decoder-only adaptation is evaluated separately from the Eulerian-only comparisons.

6.1 Eulerian prediction

Autoregressive Eulerian field rollout. We first examine whether TLO preserves predictive accuracy in the standard Eulerian setting. As shown in Table 2, TLO achieves strong Eulerian rollout performance across all five benchmarks. The improvements cover SPH-derived and grid-native data, two- and three-dimensional systems, and both simulated and reanalysis flows. These results show that TLO retains strong fixed-grid Eulerian forecasting accuracy while supporting moving particle queries through the same learned representation.

Table 2: Autoregressive Eulerian rollout error (Eul). Best results are bold; lower is better.

Dataset	LSM	GINO	GNOT	Transolver	LNO	UPT	DeepLag	**TLO (Ours)**
DAM2D	2.97E-02	6.04E-03	2.24E-02	6.13E-03	3.75E-03	2.38E-02	1.65E-02	**3.12E-03**
NS2D	3.46E-03	3.68E-05	1.72E-03	1.40E-04	8.73E-05	1.31E-03	4.04E-03	**1.98E-05**
TGV3D	9.57E-04	2.47E-04	9.37E-04	7.67E-04	7.54E-03	7.54E-03	3.17E-04	**1.75E-04**
Burgers3D	2.17E-03	5.35E-03	4.21E-04	4.22E-04	4.42E-03	5.59E-03	3.38E-03	**9.88E-05**
SEA	5.72E-01	5.90E-01	4.19E-01	5.22E-01	5.21E-01	8.97E-01	8.17E-01	**1.77E-01**

Model ablations. Table 3(a) shows that the 12×12 patch lattice yields the lowest error, suggesting a trade-off between finer spatial partitioning and the amount of information available within each patch. Under the same total token budget, Local+Global consistently outperforms Local, showing that its gain cannot be explained by additional latent capacity alone. The Local+Global error exhibits an overall downward trend and reaches its lowest value at 160 tokens. These results support the complementary roles of local tokens for spatially resolved information and global tokens for domain-level context. In Table 3(c), we control model size to ensure a fair comparison. TLO has 150.8K parameters, compared with 146.7K for the variant without the FFN, while Local-only and Global-only use the same number of tokens as TLO. Thus, the gains cannot be explained solely by model capacity, but arise from the local–global representation and latent processing.

Table 3: NS2D ablations: test Eulerian rollout MSE (lower is better). (a) Patch lattice with 64 queries. (b) Matched query budgets at an 8×8 lattice; Local+Global uses 8 global queries. (c) Component ablations with the same 64-query budget.

(a) Patch lattice

Lattice	MSE ↓
8×8	1.98E-05
10×10	2.00E-05
12×12	**1.67E-05**
16×16	1.90E-05

(b) Query budgets

Queries	Local	Local+Global
48	2.31E-05	**2.13E-05**
64	2.22E-05	**1.98E-05**
96	2.23E-05	**2.05E-05**
160	2.22E-05	**1.82E-05**

(c) Components

Design	MSE ↓
TLO (L56+G8)	**1.98E-05**
Local-only (64)	2.22E-05
Global-only (64)	4.21E-05
w/o FFN	4.51E-05

6.2 Transfer from Eulerian to Lagrangian rollout

We next test whether dynamics learned from fixed-grid Eulerian supervision transfer to closed-loop rollout at moving particle coordinates. Figure 3 shows qualitative SEA rollouts, where all methods start from the same particles and recursively update their positions using predicted velocities.

At inference, the same checkpoint supports autoregressive Eulerian forecasting, while the decoder additionally evaluates velocities at recursively advected particle positions without parameter adaptation. As shown in Table 4, TLO achieves the lowest Eul error among the compared Eulerian-only (EO) methods on all three benchmarks and the lowest Ref and Path errors on DAM2D. On Burgers3D and SEA, Transolver obtains lower Ref and Path errors despite higher Eul errors, highlighting that better fixed-grid accuracy does not necessarily translate into better Lagrangian predictions. These results demonstrate zero-shot reuse of an Eulerian-trained model for moving-coordinate rollout, while revealing dataset-dependent differences in transfer performance.

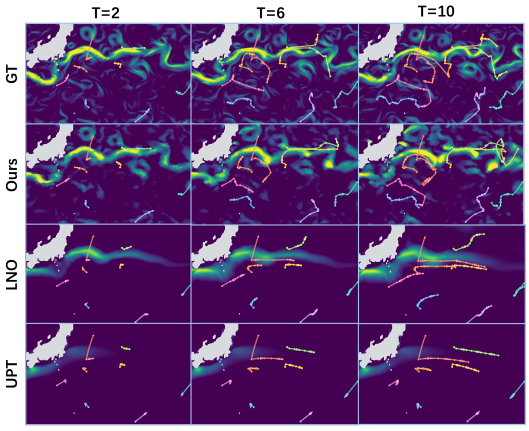

Figure 3: Closed-loop Lagrangian rollout on SEA.

We separately evaluate a *decoder-adapted* (DA) setting, where only the decoder is fine-tuned using sparse velocity samples at particle coordinates, with the encoder and latent processor frozen. Relative to EO, DA reduces Ref errors on all three benchmarks and Path errors on Burgers3D and SEA, while the DAM2D Path error remains unchanged at the reported precision. These gains show that sparse decoder calibration can further improve Lagrangian predictions while retaining the Eulerian-trained encoder and latent processor.

Table 4: Eulerian-to-Lagrangian transfer under Eulerian-only (EO) and decoder-adapted (DA) settings. Lower is better.

		DAM2D			Burgers3D			SEA		
Method	Setting	Eul ↓	Ref ↓	Path ↓	Eul ↓	Ref ↓	Path ↓	Eul ↓	Ref ↓	Path ↓
UPT		2.38E-02	7.62E-02	1.44E-01	5.59E-03	2.91E-03	7.73E-03	8.97E-01	4.92E-02	1.06E-02
LNO		3.75E-03	4.35E-03	5.30E-03	4.42E-03	3.36E-03	9.42E-03	5.21E-01	5.29E-02	1.10E-02
DeepLag	EO	1.65E-02	4.08E-02	4.65E-02	3.38E-03	2.16E-03	7.39E-03	8.17E-01	5.01E-02	1.08E-02
Transolver		6.13E-03	7.70E-03	9.60E-03	4.22E-04	**3.36E-04**	**2.75E-03**	5.22E-01	**2.34E-02**	**8.52E-03**
TLO		**3.12E-03**	**2.41E-03**	**3.72E-03**	**9.88E-05**	6.17E-04	3.01E-03	**1.77E-01**	3.00E-02	8.99E-03
TLO	DA	**3.08E-03**	**2.39E-03**	**3.72E-03**	**9.48E-05**	**2.25E-04**	**2.59E-03**	**1.72E-01**	2.95E-02	**8.29E-03**

Comparison with Lagrangian baseline. We compare TLO-DA with GNS (Sanchez-Gonzalez et al., 2020), a native Lagrangian model trained on particle trajectories, in Table 5. This comparison examines whether latent dynamics learned from Eulerian fields, together with lightweight decoder adaptation, can support competitive particle rollout. GNS learns particle interactions directly through trajectory supervision, whereas TLO-DA keeps its Eulerian-trained encoder and latent processor fixed and fine-tunes only the decoder using sparse velocity samples at particle coordinates, without a trajectory loss.

On DAM2D, TLO-DA achieves more accurate particle rollout with lower computational overhead than GNS. This advantage comes from the different representations used for dynamics modeling: GNS explicitly propagates information among particles through graph message passing, whose cost grows with particle interactions, while TLO evolves a compact latent flow representation and only decodes velocities at queried locations. These results show that an Eulerian latent representation can recover accurate particle dynamics without the cost of explicit particle interaction modeling.

Table 5: Comparison with a native Lagrangian model on DAM2D particle rollout. TLO and GNS contain 144K and 146K parameters, respectively. Memory and runtime are reported in MiB and ms/step, respectively.

Method	Ref ↓	Path ↓	Mem ↓	Time ↓
TLO-DA	**2.39E-03**	**3.72E-03**	**35.94**	**7.26**
GNS	4.19E-02	2.50E-02	47.99	8.68

6.3 Beyond-Grid Lagrangian Rollout

We further evaluate TLO on Lagrangian rollouts beyond the fixed Eulerian grid support. Starting from the same latent flow state, we compare two readout strategies: TLO-Interp obtains particle velocities by interpolating the predicted Eulerian field, whereas TLO-Direct evaluates the coordinate-conditioned decoder at particle locations.

Within the grid support, the two strategies achieve comparable accuracy. Once particles leave the grid domain, however, standard interpolation is no longer defined without an additional extrapolation rule, while TLO-Direct can still be evaluated at the off-grid particle coordinates. TLO-DA, which fine-tunes only the decoder using sparse particle-velocity supervision, further reduces the off-grid rollout error. These results highlight the advantage of decoupling latent flow evolution from coordinate-dependent readout: Lagrangian rollout is not restricted to the spatial support of the Eulerian output grid.

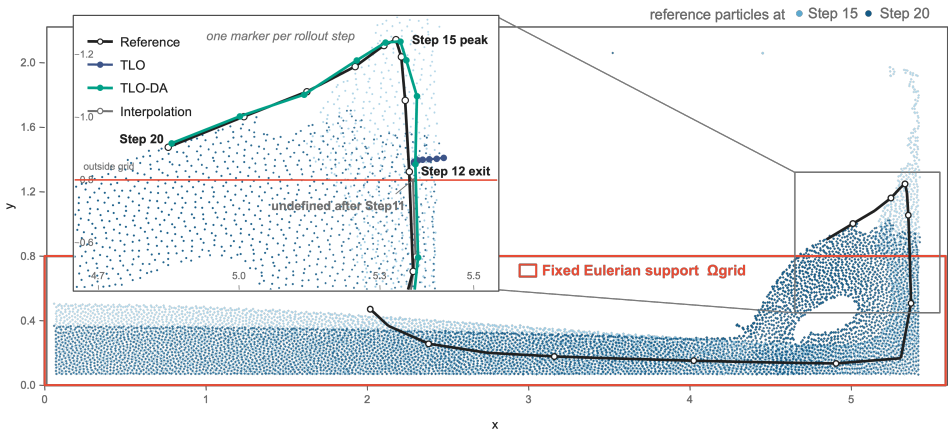

Figure 4: Closed-loop rollout beyond the fixed Eulerian grid support on DAM2D. Light- and dark-blue points show reference particles at Steps 15 and 20, respectively. The inset follows a particle crossing Ω_{grid}: grid interpolation becomes undefined after the exit, whereas direct decoding continues the rollout.

Table 6: Direct decoding and grid interpolation on DAM2D. A dash indicates that interpolation cannot continue after grid exit.

	In-grid (10 steps)		Outside (20 steps)		Full (20 steps)
Method	Ref ↓	Path ↓	Ref ↓	Path ↓	Path ↓
TLO-Interp	**2.89E-04**	**1.63E-04**	–	–	–
TLO-Direct	3.22E-04	1.89E-04	1.35E-01	6.05E-02	4.48E-03
TLO-DA	3.22E-04	1.89E-04	**3.82E-02**	**3.51E-02**	**2.51E-03**

7 Conclusion

TLO achieves strong Eulerian forecasting accuracy and competitive zero-shot Lagrangian rollout,

To answer whether flow dynamics learned only from fixed-grid Eulerian data can support closed-loop Lagrangian particle rollout, we formulated zero-shot Eulerian-to-Lagrangian generalization and developed TLO. By separating query-independent latent dynamics from coordinate-conditioned decoding, TLO supports both fixed-grid field prediction and rollout at recursively evolving particle locations. Across five field-prediction and three particle-rollout benchmarks, TLO achieves strong Eulerian accuracy and competitive rollout errors; sparse decoder adaptation further improves accuracy and outperforms GNS on DAM2D. The decoder can also be queried after particles leave the fixed-grid domain, whereas interpolation requires an additional extrapolation rule. Overall, these results show that models trained on fixed-grid supervision can learn transferable flow dynamics beyond fixed-grid inference, while closed-loop moving queries provide a stringent test of this transferability.

AI use statement

Generative AI tools were used for language polishing and grammar checking. The authors reviewed all AI-assisted edits and take full responsibility for the final manuscript and its claims.

References

Benedikt Alkin, Andreas Fürst, Simon Schmid, Lukas Gruber, Markus Holzleitner, and Johannes Brandstetter. Universal physics transformers: A framework for efficiently scaling neural operators. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://arxiv.org/abs/2402.12365.

Jeremiah U Brackbill and Hans M Ruppel. Flip: A method for adaptively zoned, particle-in-cell calculations of fluid flows in two dimensions. *Journal of Computational physics*, 65(2):314–343, 1986.

Shuhao Cao. Choose a transformer: Fourier or galerkin. *Advances in neural information processing systems*, 34:24924–24940, 2021.

CMEMS and MDS. Global ocean physics reanalysis. DOI: 10.48670/moi-00021 (Accessed on 23 September 2023), 2023.

Philippe Delandmeter and Erik Van Sebille. The parcels v2. 0 lagrangian framework: new field interpolation schemes. *Geoscientific Model Development*, 12(8):3571–3584, 2019.

Zhongkai Hao, Zhengyi Wang, Hang Su, Chengyang Ying, Yinpeng Dong, Songming Liu, Ze Cheng, Jian Song, and Jun Zhu. Gnot: A general neural operator transformer for operator learning. In *International conference on machine learning*, pp. 12556–12569. PMLR, 2023.

Philipp Holl, Vladlen Koltun, Kiwon Um, and Nils Thuerey. phiflow: A differentiable pde solving framework for deep learning via physical simulations. In *NeurIPS workshop*, volume 2. NeurIPS, 2020.

Yuanming Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley, and Frédo Durand. Difftaichi: Differentiable programming for physical simulation. *arXiv preprint arXiv:1910.00935*, 2019.

Felix Koehler, Simon Niedermayr, Rüdiger Westermann, and Nils Thuerey. APEBench: A benchmark for autoregressive neural emulators of pdes. *Advances in Neural Information Processing Systems*, 37:120252–120310, 2024.

Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces with applications to pdes. *Journal of Machine Learning Research*, 24(89):1–97, 2023.

Zhihao Li, Haoze Song, Di Xiao, Zhilu Lai, and Wei Wang. Harnessing scale and physics: A multi-graph neural operator framework for pdes on arbitrary geometries. *arXiv preprint arXiv:2411.15178*, 2024.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=c8P9NQVtmnO.

Zongyi Li, Nikola Kovachki, Chris Choy, Boyi Li, Jean Kossaifi, Shourya Otta, Mohammad Amin Nabian, Maximilian Stadler, Christian Hundt, Kamyar Azizzadenesheli, et al. Geometry-informed neural operator for large-scale 3d pdes. *Advances in Neural Information Processing Systems*, 36:35836–35854, 2023.

Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, 2021. doi: 10.1038/s42256-021-00302-5. URL https://doi.org/10.1038/s42256-021-00302-5.

Qilong Ma, Haixu Wu, Lanxiang Xing, Shangchen Miao, and Mingsheng Long. Deeplag: Discovering deep lagrangian dynamics for intuitive fluid prediction. *Advances in Neural Information Processing Systems*, 37:122833–122866, 2024.

Ruben Ohana, Michael McCabe, Lucas Meyer, Rudy Morel, et al. The well: A large-scale collection of diverse physics simulations for machine learning. *Advances in Neural Information Processing Systems*, 37:44989–45037, 2024.

Md Ashiqur Rahman, Zachary E Ross, and Kamyar Azizzadenesheli. U-no: U-shaped neural operators. *arXiv preprint arXiv:2204.11127*, 2022.

Omer Rochman-Sharabi, Sacha Lewin, and Gilles Louppe. A neural material point method for particle-based emulation. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=zSK81A2hxQ.

Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. Learning to simulate complex physics with graph networks. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 8459–8468. PMLR, 2020. URL https://proceedings.mlr.press/v119/sanchez-gonzalez20a.html.

Deborah Sulsky, Zhen Chen, and Howard L Schreyer. A particle method for history-dependent materials. *Computer methods in applied mechanics and engineering*, 118(1-2):179–196, 1994.

Makoto Takamoto, Timothy Praditia, Raphael Leiteritz, Dan MacKinlay, Francesco Alesiani, Dirk Pflüger, and Mathias Niepert. PDEBench: An extensive benchmark for scientific machine learning. In *Advances in Neural Information Processing Systems*, volume 35, 2022. URL https://arxiv.org/abs/2210.07182.

Sukirt Thakur and Maziar Raissi. Elpinn: eulerian lagrangian physics-informed neural network. *arXiv preprint arXiv:2504.09053*, 2025.

Artur Toshev, Gianluca Galletti, Fabian Fritz, Stefan Adami, and Nikolaus Adams. Lagrangebench: A lagrangian fluid mechanics benchmarking suite. *Advances in Neural Information Processing Systems*, 36:64857–64884, 2023.

Tapas Tripura and Souvik Chakraborty. Wavelet neural operator for solving parametric partial differential equations in computational mechanics problems. *Computer Methods in Applied Mechanics and Engineering*, 404:115783, 2023.

Jingdi Wan, Hongping Wang, Bo Liu, Xiaolei Yang, Xiaodong Hu, Shengze Cai, Guowei He, and Yang Liu. TrajectoryFlowNet: Lagrangian–eulerian learning of flow field and trajectories, 2025. URL https://arxiv.org/abs/2507.09621.

Tian Wang and Chuang Wang. Latent neural operator for solving forward and inverse pde problems. *Advances in Neural Information Processing Systems*, 37:33085–33107, 2024.

Shizheng Wen, Arsh Kumbhat, Levi Lingsch, Sepehr Mousavi, Yizhou Zhao, Praveen Chandrashekar, and Siddhartha Mishra. Geometry aware operator transformer as an efficient and accurate neural surrogate for pdes on arbitrary domains. *Advances in Neural Information Processing Systems*, 38:155423–155501, 2026.

Haixu Wu, Tengge Hu, Huakun Luo, Jianmin Wang, and Mingsheng Long. Solving high-dimensional pdes with latent spectral models. In *International Conference on Machine Learning*, pp. 37417–37438. PMLR, 2023.

Haixu Wu, Huakun Luo, Haowen Wang, Jianmin Wang, and Mingsheng Long. Transolver: A fast transformer solver for PDEs on general geometries. In *International Conference on Machine Learning*, 2024. URL https://arxiv.org/abs/2402.02366.

Shiying Xiong, Xingzhe He, Yunjin Tong, Yitong Deng, and Bo Zhu. Neural vortex method: From finite lagrangian particles to infinite dimensional eulerian dynamics. *Computers & Fluids*, 258: 105811, 2023.

A Notation and Detailed Problem Formulation

This appendix summarizes the notation in Table 7 and provides the detailed Eulerian training objective, autoregressive field prediction, and numerical particle rollout corresponding to the workflow in Figure 1.

B Additional Error Analysis

This appendix provides the proof of Proposition 1, connects fixed-grid velocity error to velocity error at moving coordinates, and clarifies the relation between direct coordinate decoding and grid interpolation. For a fixed particle i, we omit the particle index throughout the proof.

Table 7: Main notation. Hats denote predictions.

Symbol	Definition
Ω	Physical flow domain.
$T,\ \Delta t$	Rollout horizon and time interval between snapshots.
$X_E,\ N_E$	Fixed Eulerian coordinates and number of grid points.
$\boldsymbol{u}_t,\ \mathbf{U}_t$	Eulerian state field and its values sampled at X_E.
$\boldsymbol{v}_t$	Velocity field extracted from $\boldsymbol{u}_t$.
$Q,\ N_Q$	Output query coordinates and number of queries.
$\boldsymbol{y}_{i,t},\ N_P$	Position of particle i at step t and number of particles.
$\mathcal{G}_\theta$	Learned one-step flow operator.
$\boldsymbol{Z}_t^\ell$	Latent tokens at processor layer ℓ for input step t.

B.1 Proof of **Proposition 1**

Proof. By the non-expansiveness of $\mathcal{B}$ and the updates in Equation 6,

$$\delta_{t+1} \le \delta_t + \Delta t \left\| \widehat{\boldsymbol{v}}_t(\widehat{\boldsymbol{y}}_t) - \boldsymbol{v}_t(\boldsymbol{y}_t) \right\|_2 . \tag{10}$$

Adding and subtracting $\widehat{\boldsymbol{v}}_t(\boldsymbol{y}_t)$ gives

$$\begin{aligned} &\left\| \widehat{\boldsymbol{v}}_t(\widehat{\boldsymbol{y}}_t) - \boldsymbol{v}_t(\boldsymbol{y}_t) \right\|_2 \\ &\quad \le \left\| \widehat{\boldsymbol{v}}_t(\widehat{\boldsymbol{y}}_t) - \widehat{\boldsymbol{v}}_t(\boldsymbol{y}_t) \right\|_2 + \left\| \widehat{\boldsymbol{v}}_t(\boldsymbol{y}_t) - \boldsymbol{v}_t(\boldsymbol{y}_t) \right\|_2 \\ &\quad \le \widehat{L}_t \delta_t + r_t. \end{aligned} \tag{11}$$

Substitution into Equation 10 yields

$$\delta_{t+1} \le (1 + \Delta t\, \widehat{L}_t)\delta_t + \Delta t\, r_t, \tag{12}$$

which proves Equation 8 in the main paper.

Starting from $\delta_0 = 0$ and recursively applying the one-step inequality gives

$$\delta_T \le \Delta t \sum_{t=0}^{T-1} r_t \prod_{k=t+1}^{T-1} (1 + \Delta t\, \widehat{L}_k), \tag{13}$$

where an empty product equals one. This is Equation 9 in the main paper. □

If $\widehat{L}_t \le \widehat{L}$ for all t, the same result admits the more readable bound

$$\delta_T \le \Delta t \sum_{t=0}^{T-1} \exp\Big(\widehat{L}(T-1-t)\Delta t\Big)\, r_t. \tag{14}$$

Thus, velocity errors introduced earlier in the rollout can have a larger effect because they alter all subsequent query positions.

B.2 From Grid Error to Trajectory Error

Let

$$X_E = \{\boldsymbol{g}_j\}_{j=1}^{N_E} \tag{15}$$

be the fixed Eulerian grid, and define the maximum error of the velocity components on this grid by

$$\epsilon_t^{\mathrm{grid}} := \max_{\boldsymbol{g} \in X_E} \left\| \widehat{\boldsymbol{v}}_t(\boldsymbol{g}) - \boldsymbol{v}_t(\boldsymbol{g}) \right\|_2 . \tag{16}$$

This quantity concerns only velocity and is therefore distinct from an Eulerian metric that may average over space or include additional state variables.

For any query coordinate $\boldsymbol{q}$, let

$$\rho_E(\boldsymbol{q}) := d_\Omega(\boldsymbol{q}, X_E) = \min_{\boldsymbol{g} \in X_E} d_\Omega(\boldsymbol{q}, \boldsymbol{g}). \tag{17}$$

Assume that $\boldsymbol{v}_t$ and $\widehat{\boldsymbol{v}}_t$ are respectively L_t- and $\widehat{L}_t$-Lipschitz on the region containing X_E and $\boldsymbol{q}$. For a nearest grid point $\boldsymbol{g}_{\boldsymbol{q}}$, the triangle inequality gives

$$\begin{aligned}
&\|\widehat{\boldsymbol{v}}_t(\boldsymbol{q}) - \boldsymbol{v}_t(\boldsymbol{q})\|_2 \\
&\quad \le \|\widehat{\boldsymbol{v}}_t(\boldsymbol{q}) - \widehat{\boldsymbol{v}}_t(\boldsymbol{g}_{\boldsymbol{q}})\|_2 + \|\widehat{\boldsymbol{v}}_t(\boldsymbol{g}_{\boldsymbol{q}}) - \boldsymbol{v}_t(\boldsymbol{g}_{\boldsymbol{q}})\|_2 \\
&\qquad + \|\boldsymbol{v}_t(\boldsymbol{g}_{\boldsymbol{q}}) - \boldsymbol{v}_t(\boldsymbol{q})\|_2 \\
&\quad \le \epsilon_t^{\mathrm{grid}} + (L_t + \widehat{L}_t)\rho_E(\boldsymbol{q}).
\end{aligned} \tag{18}$$

Evaluating this result at the reference particle position gives

$$r_{i,t} \le \epsilon_t^{\mathrm{grid}} + (L_t + \widehat{L}_t)\rho_E(\boldsymbol{y}_{i,t}). \tag{19}$$

Combining Equation 19 with **Proposition 1** yields

$$\delta_{i,t+1} \le (1 + \Delta t \widehat{L}_t)\delta_{i,t} + \Delta t \left[\epsilon_t^{\mathrm{grid}} + (L_t + \widehat{L}_t)\rho_E(\boldsymbol{y}_{i,t})\right]. \tag{20}$$

For a query inside a sufficiently resolved grid, $\rho_E(\boldsymbol{q})$ is bounded by the grid covering radius and is typically $O(h)$ for grid spacing h. Once a particle leaves the fixed Eulerian grid support, $\rho_E(\boldsymbol{q})$ also contains its distance beyond the support. The coordinate decoder remains evaluable there, but the bound becomes weaker as that distance increases. Hence, being defined outside the grid and being accurate outside the grid are distinct properties.

B.3 Direct Querying and Grid Interpolation

Inside the region Ω_{grid} covered by valid grid cells, interpolation of the predicted grid field is a valid alternative to direct coordinate decoding. Let $\mathcal{I}_h$ be a stable interpolation operator of order p. Applying it to the predicted grid values gives

$$\begin{aligned}
&\left\|\mathcal{I}_h\big(\widehat{\boldsymbol{v}}_t|_{X_E}\big)(\boldsymbol{q}) - \boldsymbol{v}_t(\boldsymbol{q})\right\|_2 \\
&\quad \le \left\|\mathcal{I}_h\big((\widehat{\boldsymbol{v}}_t - \boldsymbol{v}_t)|_{X_E}\big)(\boldsymbol{q})\right\|_2 + \left\|\mathcal{I}_h\big(\boldsymbol{v}_t|_{X_E}\big)(\boldsymbol{q}) - \boldsymbol{v}_t(\boldsymbol{q})\right\|_2 \\
&\quad \le C_{\mathrm{stab}}\epsilon_t^{\mathrm{grid}} + C_{\mathrm{int}} h^p |\boldsymbol{v}_t|_{W^{p,\infty}(\Omega_{\mathrm{grid}})}, \qquad \boldsymbol{q} \in \Omega_{\mathrm{grid}}
\end{aligned} \tag{21}$$

Therefore, direct decoding is not theoretically guaranteed to outperform interpolation for in-grid queries. Both depend on the accuracy and spatial resolution of the predicted Eulerian field.

The structural distinction arises outside Ω_{grid}. Standard cell-based interpolation requires a valid enclosing cell and is therefore undefined once a particle leaves the grid support, unless an additional extrapolation rule is introduced. A coordinate-conditioned decoder can still be evaluated at such a coordinate. Equation 18, however, shows that this extended interface does not by itself guarantee accurate spatial extrapolation.

B.4 Reference Trajectories from a Different Integrator

Proposition 1 assumes that the reference and predicted particles use the same forward-Euler update. When the reference trajectory is produced by a native simulator or a different numerical integrator, define its one-step consistency error as

$$\zeta_{i,t} := d_\Omega(\boldsymbol{y}_{i,t+1}, \mathcal{B}(\boldsymbol{y}_{i,t} + \Delta t\, \boldsymbol{v}_t(\boldsymbol{y}_{i,t}))). \tag{22}$$

Repeating the proof with one additional triangle inequality gives

$$\delta_{i,t+1} \le (1 + \Delta t\, \widehat{L}_t)\delta_{i,t} + \Delta t\, r_{i,t} + \zeta_{i,t}, \tag{23}$$

and consequently

$$\delta_{i,T} \leq \sum_{t=0}^{T-1} (\Delta t\, r_{i,t} + \zeta_{i,t}) \prod_{k=t+1}^{T-1} (1 + \Delta t\, \widehat{L}_k). \tag{24}$$

The additional term separates disagreement caused by velocity prediction from that caused by using different trajectory-generation rules. It vanishes when the reference and predicted pathlines use the same integrator.

B.5 Relation to the Reported Metrics

The bounds above are pointwise, whereas the experiments report errors pooled over particles, time steps, and test trajectories. The reported *Ref* metric averages the squared counterpart of $r_{i,t}$ while holding query positions on the reference pathlines. The *Path* metric averages the squared counterpart of $\delta_{i,t}$ under recursively predicted particle positions. The former measures velocity accuracy without particle-position feedback; the latter additionally contains the accumulation and amplification.

C Data Curation

Here, we describe the data-curation procedures used to convert between two physical representations: Eulerian grid fields and Lagrangian particles.

Native Particles to Eulerian Fields. Given a Lagrangian snapshot with particle positions and physical quantities $\{(\boldsymbol{y}_i^t, \boldsymbol{v}_i^t)\}_{i=1}^{N_P}$ at time t, we construct the Eulerian grid with cell centers $\{\boldsymbol{x}_j\}_{j=1}^{N_g}$ over the physical domain. Each particle is splatted to nearby grid points using linear interpolation. The Eulerian quantity $\boldsymbol{u}_j^t$ at $\boldsymbol{x}_j$ is given by

$$\boldsymbol{u}_j^t(\boldsymbol{x}_j) = \mathcal{I}\big[\{\boldsymbol{y}_i^t, \boldsymbol{v}_i^t\}\big](\boldsymbol{x}_j) = \frac{\sum_{i=1}^{N_P} w_{ij}^t \boldsymbol{v}_i^t}{\sum_{i=1}^{N_P} w_{ij}^t + \epsilon}, \tag{25}$$

where $\mathcal{I}$ denotes the linear interpolation operator, w_{ij}^t is the corresponding linear interpolation weight between particle $\boldsymbol{y}_i^t$ and grid point $\boldsymbol{x}_j$, and $\epsilon > 0$ prevents division by zero.

Eulerian Fields to Passive Tracers. Given the reference Eulerian field and initial particle positions $\{\boldsymbol{x}_i^0\}$, the reference trajectories are obtained by solving the transport ODE:

$$\frac{d\boldsymbol{y}_i(\tau)}{d\tau} = \boldsymbol{v}(\boldsymbol{y}_i(\tau), \tau),\ \boldsymbol{y}_i(0) = \boldsymbol{y}_i^0 \tag{26}$$

where particle velocity is sampled from

$$\boldsymbol{v}_i^t = \mathcal{I}\big[\{\boldsymbol{g}_j, \boldsymbol{u}_j^t\}\big](\boldsymbol{y}_i^t)\,. \tag{27}$$

Unless otherwise noted, we solve the transport Equation 26 with RK4.

D Benchmark Details

In this section, we provide details of the benchmarks.

D.1 Native Lagrangian Data

DAM2D and TGV3D are adopted from LagrangeBench (Toshev et al., 2023) and are generated using a JAX-based Smoothed Particle Hydrodynamics (SPH) solver for the weakly compressible Navier–Stokes equations. Since the particle snapshots provide positions but not velocities, we estimate the velocity of particle i using finite differences:

$$\boldsymbol{v}_i^t = \frac{\boldsymbol{y}_i^{t+1} - \boldsymbol{y}_i^t}{\Delta t}. \tag{28}$$

We temporally subsample both datasets before differencing to obtain smoother velocity.

Table 8: Numbers of trainable parameters across the five benchmark datasets.

Method	DAM2D	NS2D	TGV3D	Burgers3D	SEA
LSM	146,088	146,088	120,663	120,663	146,139
GINO	151,769	113,396	147,077	147,077	152,351
GNOT	148,804	148,804	149,141	149,141	157,047
Transolver	131,730	131,730	132,051	132,051	132,309
LNO	152,658	152,658	152,851	152,851	152,949
UPT	148,442	148,442	148,483	148,483	148,565
DeepLag	121,516	133,804	153,038	162,146	166,597
Ours	144,155	150,811	186,544	149,139	152,753

DAM2D features a rapidly evolving free surface and strong fluid–solid wall interactions. We temporally subsample the particle snapshots to a coarse interval of $\Delta t = 0.3$, yielding 41 position frames and 40 finite-difference velocity states per trajectory. Only the $5{,}000$ moving fluid particles are used for Eulerian-field construction and trajectory evaluation, while the 740 boundary particles are excluded. The Eulerian velocity field is deposited onto a 32×64 cell-centered grid over physical domain $[0, 5.586] \times [0, 2.22]$.

TGV3D describes an unforced, decaying vortical flow initialized from the three-dimensional Taylor–Green vortex, featuring nonlinear vortex interactions and viscous dissipation. Temporal subsampling gives a coarse interval of $\Delta t = 1.5$, yielding 21 particle-position frames and 20 finite-difference velocity states per trajectory. We retain $8{,}000$ fluid particles per trajectory and use a 50/10/10 train/validation/test split. The Eulerian velocity field is deposited onto a 20^3 cell-centered grid over $[0, 2\pi]^3$.

D.2 Native Eulerian Grid Data

NS2D describes two-dimensional incompressible flow governed by the Navier–Stokes equations. The dataset is generated by solving the equations with viscosity 10^{-5} on a 64×64 grid under periodic boundary conditions. Given the initial field, the task is to predict the next 10 frames. For particle diagnostics, $1{,}024$ initial grid cells are sampled with replacement with probability proportional to the initial vorticity magnitude $|\omega_0|$, followed by uniform jitter within each selected cell.

Burgers3D describes the interplay between nonlinear advection and viscous diffusion, providing a challenging test of a model's ability to resolve steep gradients and complex spatiotemporal dynamics. We generate this dataset using the `3d_diff_burgers` configuration in APEBench (Koehler et al., 2024). Each trajectory contains 41 frames of the three velocity components (u, v, w) on a cell-centered 32^3 grid over the periodic domain $[0, 1]^3$. For Lagrangian evaluation, $4{,}096$ initial markers are placed on a shared cell-centered 16^3 lattice, with coordinates $(i + 0.5)/16$ along each spatial dimension. Marker velocities are obtained by periodic trilinear interpolation of the Eulerian velocity field.

SEA contains daily ocean reanalysis data from 2011 to 2020, cropped to a 180×300 region in the North Pacific covering approximately $375\,\mathrm{km} \times 625\,\mathrm{km}$ (Ma et al., 2024; CMEMS & MDS, 2023). Each frame contains potential temperature, salinity, zonal and meridional velocities, and sea-surface height with shape $1 \times 180 \times 300 \times 5$. $1{,}024$ particles are sampled uniformly and trajectories are generated using bilinear spatial interpolation and temporal RK4 integration. Velocities are converted to normalized one-day displacements, and tracers entering land or leaving the domain are deactivated and held fixed.

E Implementation Details

Baseline implementations. We adapt the baselines from the Neural-Solver-Library and their official repositories. For a fair comparison, we adjust the model configurations to target approximately 150K trainable parameters, as shown in Table 8. The architecture of TLO is listed in Table 9.

Table 9: TLO configurations used for the main results. Q_l and Q_g denote the numbers of local and global learnable queries. D is the hidden dimension. "Heads" denotes the number of attention heads, and "Mixing layer" denotes the number of latent mixing layers.

Dataset	Patch	Q_ℓ	Q_g	D	Heads	Mixing layers
DAM2D	4^2	16	8	48	4	4
NS2D	8^2	56	8	48	4	4
TGV3D	4^3	48	16	52	4	3
Burgers3D	4^3	32	32	48	4	4
SEA	6^2	128	32	48	4	3

Table 10: Dataset-specific training settings and autoregressive validation rollout horizons. Eul@T denotes Eulerian validation over an T-step autoregressive rollout.

Dataset	Batch size	Initial LR	Val rollout
DAM2D	16	1×10^{-3}	Eul@10
NS2D	32	1×10^{-3}	Eul@10
TGV3D	16	5×10^{-4}	Eul@5
Burgers3D	2	5×10^{-4}	Eul@5
SEA	1	5×10^{-4}	Eul@10

Training Details. All models are trained using the mean squared error (MSE) between predicted and reference next-step states. Model selection is based on the Eulerian MSE accumulated over an autoregressive validation rollout, whose horizon is specified in Table 10. The same validation metric is used for checkpoint selection, learning-rate scheduling, and early stopping.

We use AdamW with zero weight decay and clip the gradient norm to 1. The learning rate is reduced by a factor of 0.5 after 10 epochs without validation improvement, with a minimum learning rate of 10^{-5}. Training runs for at most 100 epochs. Early stopping uses a patience of 30 epochs and is enabled only after epoch 50.

E.1 Aggregation of Local and Global Tokens

The encoder partitions the normalized input coordinates into a $P_1 \times \cdots \times P_d$ patch lattice. For each patch, Q_ℓ local queries cross-attend only to the input points assigned to that patch, producing

$$Z_{\text{local}} \in \mathbb{R}^{B \times P_1 \times \cdots \times P_d \times Q_\ell \times D}.$$

In parallel, Q_g global queries cross-attend to all input points, producing

$$\widetilde{Z}_{\text{global}} \in \mathbb{R}^{B \times 1 \times \cdots \times 1 \times Q_g \times D}.$$

The global tokens are then broadcast across the patch lattice and the local and global tokens are concatenated along the query dimension:

$$Z = \text{Concat}\left(Z_{\text{local}}, Z_{\text{global}}\right) \in \mathbb{R}^{B \times P_1 \times \cdots \times P_d \times (Q_\ell + Q_g) \times D}.$$

Learned patch and query positional embeddings are incorporated before latent mixing to preserve patch identity and token identity.

E.2 Decoder Adaptation on DAM2D

During decoder adaptation, the encoder and latent processor are frozen, and only the decoder parameters are optimized. We fine-tune the decoder for 10 epochs using AdamW with a learning rate of 5×10^{-5}, a batch size of 16, and random seed 0. The adaptation objective is

$$\mathcal{L}_{\text{DA}} = 0.9\,\mathcal{L}_{\text{Eul}} + 0.1\,\mathcal{L}_{\text{Lag}}. \tag{29}$$

No particle-position or trajectory loss is used. The best adapted checkpoint is selected using the same validation Eulerian metric as in the original training stage. Thus, decoder adaptation calibrates the coordinate-conditioned readout without modifying the learned latent dynamics.

For completeness, Table 11 reports additional decoder-adaptation results for the coordinate-query baselines, complementing Table 4 in the main paper. Decoder adaptation generally improves or preserves Lagrangian evaluation performance across the reported settings.

Table 11: Effect of decoder adaptation on Lagrangian rollout for coordinate-query operators. EO denotes Eulerian-only training; DA fine-tunes only the decoder using particle-coordinate supervision. Lower is better.

		DAM2D			Burgers3D			SEA		
Method	Setting	Eul ↓	Ref ↓	Path ↓	Eul ↓	Ref ↓	Path ↓	Eul ↓	Ref ↓	Path ↓
UPT	EO	2.38E-02	7.62E-02	1.44E-01	5.59E-03	2.91E-03	7.73E-03	8.97E-01	4.92E-02	1.06E-02
LNO	EO	3.75E-03	4.35E-03	5.30E-03	4.42E-03	3.36E-03	9.42E-03	5.21E-01	5.29E-02	1.10E-02
UPT	DA	2.38E-02	7.62E-02	1.44E-01	5.58E-03	2.91E-03	7.71E-03	7.97E-01	4.85E-02	9.78E-03
LNO	DA	3.65E-03	4.22E-03	5.26E-03	4.29E-03	3.23E-03	9.27E-03	5.21E-01	5.29E-02	1.10E-02

E.3 Beyond-Grid Lagrangian Rollout

For this experiment, the full physical domain of DAM2D is

$$\Omega = [0, 5.586] \times [0, 2.22],$$

which is also used to normalize all particle and decoder-query coordinates. However, Eulerian supervision and grid-based prediction are restricted to the lower subdomain

$$\Omega_{\text{grid}} = [0, 5.586] \times [0, 0.8] \subset \Omega.$$

Therefore, coordinates in $\Omega \setminus \Omega_{\text{grid}}$ remain inside the physical domain but lie outside the support of the Eulerian output grid. The term *beyond-grid* refers to this region, rather than to coordinates outside the physical domain.

During the first 10 rollout steps, all evaluated particles remain within Ω_{grid}. Extending the rollout to 20 steps causes some particles to enter $\Omega \setminus \Omega_{\text{grid}}$, where grid interpolation is unavailable without an explicit extrapolation rule. For TLO-DA, we initialize from the same Eulerian-trained checkpoint, freeze the encoder and latent processor, and fine-tune only the coordinate decoder using particle-velocity supervision from the training split. No particle position or trajectory loss is used.

F Ablation Study

F.1 Direct Query versus Interpolation

We compare direct coordinate queries with bilinear and nearest-neighbor interpolation of the predicted Eulerian field. As shown in Table 12, interpolation-based evaluation depends on both the Eulerian grid resolution and the interpolation scheme. Coarsening the grid from 32×64 to 8×16 substantially increases both Ref and Path errors, while bilinear interpolation consistently outperforms nearest-neighbor interpolation at the same resolution.

Direct Query achieves accuracy comparable to high-resolution bilinear interpolation without relying on an auxiliary output grid or a manually chosen interpolation rule. Bilinear interpolation requires decoder outputs at the four grid points surrounding each particle, whereas nearest-neighbor interpolation uses one grid point. In contrast, Direct Query evaluates the decoder once at the particle coordinate. It therefore provides a resolution-independent and efficient interface for Lagrangian rollout, although bilinear interpolation at 32×64 achieves a slightly lower Ref error.

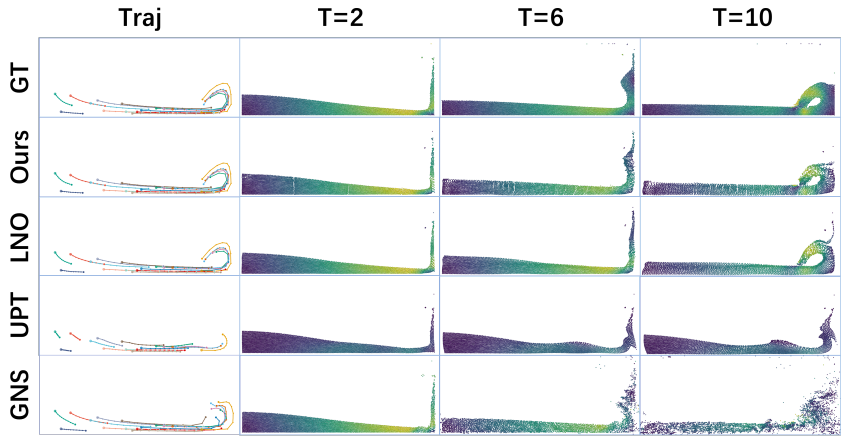

Figure 5: Qualitative closed-loop Lagrangian rollout on DAM2D. The first column (Traj) shows representative particle trajectories, while the remaining columns show all-particle snapshots at different rollout steps ($t = 2, 6, 10$).

Table 12: Comparison of interpolation-based and direct coordinate readouts on DAM2D. Direct Query is independent of the Eulerian output resolution and requires only one decoder query per particle. Lower is better.

	Ref ↓			Path ↓		
Resolution	Bilinear	Nearest	Direct query	Bilinear	Nearest	Direct query
32×64	**2.34E-03**	2.90E-03		3.75E-03	4.76E-03	
16×32	3.29E-03	4.85E-03	2.39E-03	5.79E-03	8.10E-03	**3.72E-03**
8×16	1.13E-02	1.39E-02		2.45E-02	2.90E-02	
Decoder evals.	4	1	1	4	1	1

G Additional Visualizations

As shown in Figure 5, the Eulerian-trained models preserve the overall fluid shape and free-surface structure more faithfully than GNS at later rollout steps. GNS explicitly models neighborhood interactions among thousands of particles, which is more computationally demanding under the same training budget. Consequently, particle-level errors accumulate during closed-loop rollout, leading to a dispersed and less coherent particle distribution at $T = 10$.