FlowPDE: Understanding and Justifying Flow Matching in PDE Solving

Tianxiao Li [*1] **Tianyu Huang** [*1 2] **Hang Zhou** [1] **Yuezhou Ma** [1] **Minghao Guo** [2] **Haixu Wu** [2] **Chuang Gan** [3] **Mingsheng Long** [1] **Wojciech Matusik** [2]

Abstract

Neural PDE solvers have recently begun to incorporate generative modeling. However, applying generative models to PDE solving raises a fundamental question: the uniqueness of the solution of a well-posed PDE seems to conflict with the distributional nature of generative learning, and raises concerns about whether such models may introduce non-physical or hallucinated solutions. In this work, we provide a theoretical justification for using Flow Matching in neural PDE solving. We show that, under full problem information, the target conditional distribution collapses to a one-point distribution at the exact solution, and that classical iterative solvers can be viewed as deterministic energy-descending flow trajectories. Based on this insight, we propose **FlowPDE**, a general framework that equips existing transformer-based neural operators with a conditional flow matching model through AdaLN-style injection of PDE conditions. We provide a unified formulation for both fully observed and partially observed PDE problems. Experiments on diverse PDE benchmarks with complex geometries demonstrate that FlowPDE consistently improves over strong deterministic baselines across both full and partial observation regimes.

1. Introduction

Partial differential equations (PDEs) are fundamental tools for modeling physical systems in science and engineering (Wang et al., 2023; Azizzadenesheli et al., 2024). Traditional numerical methods, such as finite element, finite difference, spectral, and Krylov-subspace solvers (Temam, 2001; Gottlieb and Orszag, 1977; Jasak, 2009), have been widely used for solving PDEs with high accuracy. However, they usually require repeated iterative computation for each new problem instance, which can be expensive in large-scale simulation and design optimization (Ashton et al., 2024; Bekemeyer et al., 2025). To accelerate scientific computing, neural PDE solvers and neural operators have recently emerged as a powerful alternative. Instead of solving a PDE iteratively, these methods (Li et al., 2021; 2020; Hao et al., 2023; Wu et al., 2024; 2026) propose to learn a non-linear mapping from the problem specification, such as geometry, boundary conditions, or initial conditions, to the corresponding solution field and have demonstrated promising performance across various benchmarks and industrial-level simulations (Luo et al., 2025; Alkin et al., 2025).

More recently, generative models have empowered significant progress in extensive domains (Achiam et al., 2023; Abramson et al., 2024; Price et al., 2025). Frontier neural solvers (Huang et al., 2024; Holzschuh et al., 2026) have started to incorporate generative modeling techniques, especially diffusion models (Ho et al., 2020; Lu et al., 2022) and Flow Matching (Lipman et al., 2023), into PDE solving. Rather than predicting a solution field in a single deterministic step, these methods model a transport or denoising process from an initial random field to the target physical solution. Such formulations have shown strong empirical potential, since they provide an iterative refinement mechanism and can represent uncertainty in partially observed settings (Ma et al., 2025). In particular, Flow Matching offers a simple and efficient framework for learning solution trajectories through a time-dependent velocity field.

Despite these empirical successes, applying generative modeling to PDE solving raises a fundamental concern. In standard generative modeling (Rombach et al., 2022; Lipman et al., 2023; Ho et al., 2020), the goal is to learn a target data distribution and generate diverse samples from it. Even given an extremely specialized instruction, the generative model may still produce results with randomness (Zhang et al., 2023). However, for a fixed well-posed PDE problem, the physical solution is unique (Evans, 2010; Wazwaz, 2002). That is, once the geometry, equation parameters, boundary conditions, and initial conditions are fully specified, the target distribution should collapse to a single point concentrated at the exact solution. This appears to con-

[*]Equal contribution [1]Tsinghua University, Beijing, China [2]Massachusetts Institute of Technology, Cambridge, USA [3]University of Massachusetts at Amherst, Amherst, USA. Correspondence to: Haixu Wu <wuhaixu98@gmail.com>.

Proceedings of the AI4Physics Workshop at the 43rd International Conference on Machine Learning (AI4Physics@ICML 2026), Seoul, South Korea. 2026. Copyright 2026 by the author(s).

flict with the distribution-modeling nature of diffusion or Flow-Matching models, and raises the question of whether generative neural solvers may introduce non-physical uncertainty or "hallucinated" solutions.

In this paper, we provide a theoretical justification for using Flow Matching in neural PDE solving. Our key observation is that, under full problem information, Flow Matching does not need to model an unconstrained *many-to-many distribution*. Instead, for a well-posed PDE, the final-time conditional distribution is a *one-point distribution*. Moreover, classical iterative solvers can be interpreted as energy-descending processes: starting from an initial guess, they progressively reduce an energy or residual functional until reaching the solution. By reparameterizing the solver time using the normalized energy decay, we show that such numerical solving trajectories can be written as deterministic conditional paths in Flow Matching. This perspective explains why Flow Matching can transport random initial fields to the unique physical solution without introducing unconstrained hallucination, encouraging future research embracing this advanced technique.

Motivated by this theoretical connection, we propose FlowPDE, a general Flow-Matching framework for neural PDE solvers. Unlike prior usages that mainly plug diffusion or Flow-Matching modules into existing neural operators, FlowPDE is presented from the solver-flow perspective and presents an elaborate architecture design. Specifically, we condition the learned velocity field on the full PDE problem specification, including geometry, boundary conditions, initial conditions, and equation parameters. And unlike prior designs, FlowPDE introduces all conditions (even geometry representation) in an AdaLN style (Peebles and Xie, 2023), which clearly separates the solution and condition representation. This allows the model to learn problem-dependent solution flows rather than unconditional generative trajectories. As a result, FlowPDE provides a unified formulation for both fully and partially observed PDE problems: under full information, it transports initial fields to the unique solution; under partial information, it captures the distribution of physically plausible solutions and mitigates the mean-field collapse of deterministic neural solvers. Experimentally, FlowPDE consistently improves various advanced PDE solvers across diverse industrial simulation tasks.

2. Preliminaries

2.1. Classical and Neural PDE Solvers

A general partial differential equation (PDE) can be written in operator form (Li et al., 2021; Kovachki et al., 2023) as $\mathcal{F}(\boldsymbol{w}) = \mathbf{0}$, where $\mathcal{F}$ denotes a differential operator that encodes the PDE problem P, *i.e.*, the governing equation together with associated boundary and initial conditions, acting on the unknown function $\boldsymbol{w}$. Under appropriate discretizations, *e.g.*, finite element and finite difference (Gottlieb and Orszag, 1977; Ŝolín, 2005; Jessica et al., 2023), this system can be reduced to a linear system of the form

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{1}$$

where $\mathbf{A}$ encodes the basis function and geometry discretization information, $\mathbf{b}$ denotes the boundary condition and $\mathbf{x}$ corresponds to the target solution coefficients. Both classical numerical methods and neural operator approaches are, in essence, built upon solving such linear systems (Jasak, 2009; Li et al., 2021; Wu et al., 2024).

Numerical methods These methods formulate solving Eq. (1) as an iterative optimization problem:

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} \Phi(\mathbf{x}), \text{where } \mathbf{x}_0 \to \mathbf{x}_1 \to \ldots, \mathbf{x}_\tau, \cdots \to \mathbf{x}^*. \tag{2}$$

Here $\mathbf{x}_0$ is the initialization, $\mathbf{x}_\tau$ represents the solution of intermediate steps and $\Phi(\mathbf{x})$ is an objective functional that measures the discrepancy between a candidate solution $\mathbf{x}$ and the exact solution. A representative example is the conjugate gradient (CG) method (Hestenes and Stiefel, 1952), which corresponds to minimizing the quadratic energy $\Phi(\mathbf{x}) := {}^1\!/_2\mathbf{x}^\top\mathbf{A}\mathbf{x} - \mathbf{b}^\top\mathbf{x}$, whereas the GMRES method (Saad and Schultz, 1986) instead minimizes the squared residual norm $\Phi(\mathbf{x}) := \|\mathbf{A}\mathbf{x} - \mathbf{b}\|^2$.

Neural operators Unlike numerical methods, neural operators attempt to directly learn an operator

$$\mathcal{G}_\theta : \mathbf{b} \mapsto \mathbf{x} = \mathbf{A}^{-1}\mathbf{b}, \tag{3}$$

where θ is the network parameter. A substantial body of work has explored this paradigm across different architectures and training strategies. Early efforts such as FNO (Li et al., 2021), which learns integral operators in the Fourier domain, demonstrate strong performance but are limited to regular domains. To address the complex geometries encountered in practical applications, subsequent works have introduced various extensions. Geo-FNO (Li et al., 2022) incorporates geometric deformations to handle irregular domains, while LSM (Wu et al., 2023) further compresses complex geometries into a compact latent representation. For problems defined on irregular structures, graph-based approaches, including Graph-UNet (Gao and Ji, 2019), GNO (Li et al., 2020), and GINO (Li et al., 2023a), have also been proposed. More recently, Transformers (Vaswani et al., 2017) have been introduced into neural operator frameworks. However, the standard attention mechanism incurs substantial computational overhead. As a result, subsequent works, such as OFormer (Li et al., 2023b), GNOT (Hao et al., 2023), ONO (Xiao et al., 2024), and FactFormer (Li et al., 2023c), have focused on designing more efficient attention mechanisms. In parallel, approaches including UPT (Alkin

et al., 2024), and Transolver (Wu et al., 2024) explore attention mechanisms defined in latent spaces. Among these, the Transolver family (Luo et al., 2025; Zhou et al., 2026) proposes a Physics-Attention mechanism, which groups mesh points into a set of physical states and performs attention across these states.

There also exist alternative approaches that follow different paradigms. Physics-Informed Neural Networks (PINNs) (Raissi et al., 2019) obtain the PDE solution by minimizing a physics-informed loss. Besides, Monte Carlo methods (Sawhney and Crane, 2020) exploit stochastic process representations of PDEs, expressing the solution in terms of expectations over random trajectories.

2.2. Conditional Flow Matching

As one of the variants (Liu et al., 2022) of Flow Matching (FM), which has seen successful applications in domains such as robotics (Black et al., 2024) and video generation (Kong et al., 2024), Conditional Flow Matching (CFM) (Lipman et al., 2023) learns continuous normalizing flows by modeling a time-dependent vector field $\boldsymbol{v}_t(\boldsymbol{x})$ satisfying the ODE $\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = \boldsymbol{v}_t(\mathbf{x})$ that transports a prior distribution p_0 to the target data distribution p_1. Since directly learning the global transport in FM can be challenging, CFM makes training tractable by decomposing it into conditional probability paths $p_t(\mathbf{x} \mid \mathbf{x}_0, \mathbf{x}^*)$ connecting a source sample $\mathbf{x}_0 \sim p_0$ and a target sample $\mathbf{x}^* \sim p_1$. The model is trained by regressing the learned velocity field toward the induced conditional vector field through the standard CFM objective:

$$\mathcal{L}_{\mathrm{CFM}}(\theta) = \mathbb{E}_{\substack{t\sim\mathrm{Unif}[0,1] \\ (\mathbf{x}_0,\mathbf{x}^*)\sim q(\mathbf{x}_0,\mathbf{x}^*) \\ \mathbf{x}_t\sim p_t(\cdot|\mathbf{x}_0,\mathbf{x}^*)}} \left[\left\| \boldsymbol{v}_\theta(\mathbf{x}_t, t) - \boldsymbol{\nu}_t(\mathbf{x}_t \mid \mathbf{x}_0, \mathbf{x}^*) \right\|^2 \right], \tag{4}$$

where $\boldsymbol{\nu}_t(\mathbf{x} \mid \mathbf{x}_0, \mathbf{x}^*)$ denotes the conditional velocity field associated with the probability path $p_t(\mathbf{x} \mid \mathbf{x}_0, \mathbf{x}^*)$, and $q(\mathbf{x}_0, \mathbf{x}^*)$ is the joint coupling between source and target samples. A common instantiation of CFM adopts the Optimal Transport (OT) formulation, where the conditional probability path $\mathbf{x}_t$ and constant conditional velocity field $\boldsymbol{\nu}_t$ are defined through linear interpolation (Lipman et al., 2023):

$$\mathbf{x}_t = (1-t)\mathbf{x}_0 + t\mathbf{x}^*, \ \boldsymbol{\nu}_t(\mathbf{x} \mid \mathbf{x}_0, \mathbf{x}^*) = \mathbf{x}^* - \mathbf{x}_0. \tag{5}$$

This straight-line construction leads to a simple regression target and enables efficient sampling.

In the neural operator literature for partial differential equations, several advanced works (Holzschuh et al., 2026) have incorporated FM, achieving notable performance improvements in practice. However, to the best of our knowledge, no prior work has investigated the theoretical connection between classical iterative solvers and FM. Establishing this connection is the focus of the present work.

3. Method

As aforementioned, this paper attempts to establish a justification of Flow Matching in PDE solving. In this section, we first present the theoretical connection between numerical solvers and CFM, where the numerical method can be reformulated in the form of CFM, bringing interesting insights to PDE solving. Next, we introduce a theory-inspired architecture design to incorporate CFM to neural solvers for both fully and partially observable scenarios.

3.1. Classical PDE Solvers as Reparameterized Flow Matching

We first establish a connection between classical iterative solvers and Flow Matching through the lens of energy descent. As formalized in Eq. (1) and Eq. (2), classical solvers do not directly predict the solution $\mathbf{x}^*$ in one step. Instead, starting from an initial guess $\mathbf{x}_0$, they progressively reduce a problem-dependent energy or residual functional until convergence. Below, we show that this energy-descending process can be reparameterized as a Flow-Matching process in Eq. (5).

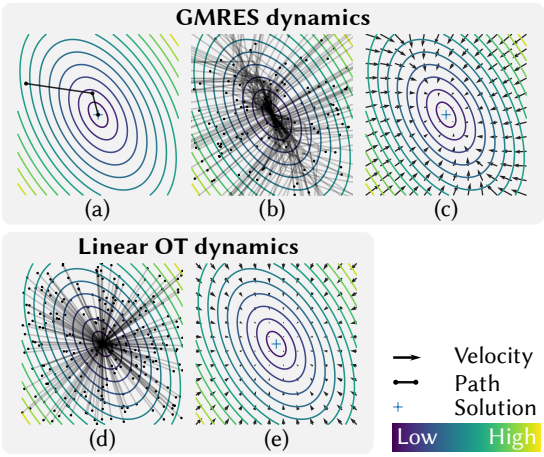

Figure 1. Comparison between classical solver dynamics (here we show GMRES dynamics) and straight-line Flow Matching dynamics. Classical PDE solvers follow equation-driven trajectories, whereas FM directly connects the initial field and the target solution.

Energy-descent formulation For a fixed PDE problem $P = [\mathbf{g}, \mathbf{u}]$, where $\mathbf{g}$ refers to the geometries and the PDE condition $\mathbf{u}$ consists of point-wise conditions $\mathbf{u}_{\mathrm{p}}$ and global conditions $\mathbf{u}_{\mathrm{g}}$, we define $\Phi_P(\mathbf{x}) \geq 0$ as the associated energy (or residual) functional, and assume $\Phi_P(\mathbf{x}^*) = 0$ at the exact solution. Many classical solvers can be viewed as producing a descent trajectory toward $\mathbf{x}^*$ by iteratively reducing Φ_P (see Figure 1(a, b)). A continuous idealization of this process is the gradient flow, which can be formalized as follows:

$$\frac{\mathrm{d}\mathbf{x}(s)}{\mathrm{d}s} = -\nabla_{\mathbf{x}} \Phi_P\left(\mathbf{x}(s)\right), \tag{6}$$

where s denotes the continuous solver time. Practical solvers may use algorithm-specific directions, preconditioners, and step sizes, but their updates are still governed by the decrease of the same underlying energy or residual. Therefore, for a fixed P and initialization $\mathbf{x}_0$, a classical solver induces an equation-driven trajectory from $\mathbf{x}_0$ to $\mathbf{x}^*$ as showcased in Figure 1(b).

Reparameterized timestep The iteration index does not naturally lie in the Flow-Matching interval $[0, 1]$. We therefore introduce a normalized time coordinate through the remaining energy ratio:

$$t(\mathbf{x}) = 1 - \frac{\Phi_P(\mathbf{x})}{\Phi_P(\mathbf{x}_0)}. \tag{7}$$

Since $\Phi_P(\mathbf{x}^*) = 0$, this gives $t(\mathbf{x}_0) = 0$ and $t(\mathbf{x}^*) = 1$. Equivalently, along the reparameterized trajectory $\mathbf{x}_t$, the energy satisfies

$$\Phi_P(\mathbf{x}_t) = (1 - t)\Phi_P(\mathbf{x}_0), \qquad t \in [0, 1]. \tag{8}$$

Thus, normalized time t increases monotonically whenever the solver monotonically decreases Φ_P.

Theorem 3.1 (Energy-normalized velocity field of numerical solvers)**.** *Given a fixed PDE problem P and associated energy $\Phi_P(\cdot)$ with $\Phi_P(\mathbf{x}^*) = 0$, its energy-normalized solving process can be reparameterized into a trajectory $\mathbf{x}_0 \to \mathbf{x}_t \to \mathbf{x}^*$. Let $\delta(\cdot)$ denote the Dirac measure. For $t \in [0, 1)$ with $\nabla_{\mathbf{x}}\Phi_P(\mathbf{x}_t) \neq \mathbf{0}$, the conditional path and one admissible velocity field can be formalized as*

$$\begin{aligned} p_t(\mathbf{x} \mid \mathbf{x}_0, P) &= \delta(\mathbf{x} - \mathbf{x}_t), \\ \frac{\mathrm{d}\mathbf{x}_t}{\mathrm{d}t} &= -\frac{\Phi_P(\mathbf{x}_0)}{\|\nabla_{\mathbf{x}}\Phi_P(\mathbf{x}_t)\|_2^2}\nabla_{\mathbf{x}}\Phi_P(\mathbf{x}_t). \end{aligned} \tag{9}$$

Proof. By the normalized time, the reparameterized trajectory satisfies Eq. (8). Taking the derivative *w.r.t.* t gives $\nabla_{\mathbf{x}}\Phi_P(\mathbf{x}_t)^\top \frac{\mathrm{d}\mathbf{x}_t}{\mathrm{d}t} = -\Phi_P(\mathbf{x}_0)$. The proposed velocity field satisfies this condition since

$$\nabla_{\mathbf{x}}\Phi_P(\mathbf{x}_t)^\top \left(-\frac{\Phi_P(\mathbf{x}_0)}{\|\nabla_{\mathbf{x}}\Phi_P(\mathbf{x}_t)\|_2^2}\nabla_{\mathbf{x}}\Phi_P(\mathbf{x}_t) \right) = -\Phi_P(\mathbf{x}_0). \tag{10}$$

Therefore, it realizes the desired energy-normalized descent. Since the trajectory is deterministic for fixed $\mathbf{x}_0$ and P, the conditional path is concentrated at $\mathbf{x}_t$, i.e., $p_t(\mathbf{x} \mid \mathbf{x}_0, P) = \delta(\mathbf{x} - \mathbf{x}_t)$. □

Corollary 3.2 (Well-posed PDEs induce one-point target distributions)**.** *For a fixed well-posed PDE problem P, the solution is unique. Therefore, the final-time conditional distribution of the Flow-Matching path is not a many-to-many target distribution, but a Dirac measure:*

$$p_{t=1}(\mathbf{x} \mid P) = \delta(\mathbf{x} - \mathbf{x}^*). \tag{11}$$

Thus, conditioned on P, different initial fields $\mathbf{x}_0 \sim p_0$ are transported to the same physical solution $\mathbf{x}^*$, rather than to arbitrary samples.

Theorem 3.1 shows that classical PDE solvers can be reparameterized as deterministic conditional paths in Flow Matching. Furthermore, Corollary 3.2 shows that, under full information, Flow Matching fits a one-point target distribution rather than an unconstrained generative distribution. The convergence behavior of the learned flow toward the one-point target is analyzed in Appendix D. This explains why the proposed formulation does not suffer from the non-physical "hallucinations" often associated with general-purpose generative models. Guided by the above perspective, in Figure 1(c), we successfully learn the dynamical field of the GMRES solver in a 2D parameter space using a small MLP.

3.2. FlowPDE

Motivated by the observation that the velocity field underlying a numerical solver, we present FlowPDE as a general framework for iterative PDE solving. As shown in Theorem 3.1, a classical iterative solver can be interpreted as constructing a problem-conditioned flow from an initial field to the PDE solution. FlowPDE replaces this solver-induced velocity field with a neural velocity model,

$$\frac{\mathrm{d}\mathbf{x}_t}{\mathrm{d}t} = \boldsymbol{v}_\theta(\mathbf{x}_t, t, P), \tag{12}$$

where P denotes the full PDE specification. This formulation preserves the iterative refinement nature of numerical solvers, while allowing each update to be represented by a nonlinear and nonlocal neural approximation. As a result, FlowPDE can learn problem-adaptive update directions that approximate the effect of multiple classical iterations within fewer neural steps.

Architecture inspiration from Theorem 3.1 As aforementioned, Theorem 3.1 suggests that classical solvers can be viewed as conditional PDE solving processes. Given a PDE problem P, each iteration advances the current state along a problem-dependent trajectory. This provides two design principles for neural solvers. First, instead of predicting the solution in one deterministic pass, neural PDE solvers can be formulated as iterative transport processes. Second, since the trajectory is determined by the PDE instance, all information, including geometry, boundary conditions, initial conditions, and equation parameters should be used as conditions of the velocity field.

We adopt the straight-line Flow-Matching path

$$\mathbf{x}_t = (1 - t)\mathbf{x}_0 + t\mathbf{x}^*, \qquad \boldsymbol{\nu}_t = \mathbf{x}^* - \mathbf{x}_0, \tag{13}$$

and train the model by regressing the predicted velocity to $\boldsymbol{\nu}_t$. Compared with classical solvers whose updates

are restricted to residual, gradient, or Krylov directions, the learned velocity can represent nonlinear, nonlocal, and problem-adaptive updates.

Fully conditioned architecture As formalized in Eq. (12), a key design of FlowPDE is to inject the full PDE specification as conditioning information. Prior generative neural solvers often concatenate PDE conditions with the input state or encode them as ordinary token features (Zhou et al., 2025; Holzschuh et al., 2026; Huang et al., 2024). However, in Flow Matching, the current state $\mathbf{x}_t$ and the PDE condition P play different roles: $\mathbf{x}_t$ specifies the location along the transport path, while P determines which velocity field should be applied.

Unlike previous designs, motivated by the well-acknowledged conditional generative architecture DiT (Peebles and Xie, 2023), FlowPDE injects most PDE information through adaptive normalization (AdaLN). Specifically, we encode the PDE condition $\mathbf{u}$ and timestep t into a conditioning embedding $\mathbf{c} = \mathrm{Enc}_P(P, t)$, which is used to generate scale, shift, and residual-gating parameters:

$$(\boldsymbol{\gamma}, \boldsymbol{\beta}, \boldsymbol{\alpha}) = \mathrm{MLP}(\mathbf{c}). \tag{14}$$

For a hidden feature $\mathbf{h}$, a FlowPDE block is written as

$$\mathbf{h}' = \mathbf{h} + \boldsymbol{\alpha} \odot \mathcal{N}_\theta \left(\boldsymbol{\gamma} \odot \mathrm{LN}(\mathbf{h}) + \boldsymbol{\beta} \right), \tag{15}$$

where $\mathcal{N}_\theta$ can be instantiated as Physics-Attention (Wu et al., 2024), linear attention (Cao, 2021), cross-attention (Alkin et al., 2024), or other neural operator blocks. This design keeps the state and condition representations separated, while allowing the PDE condition to modulate the velocity field throughout the network. For geometries $\mathbf{g}$, we still feed it into the model via concatenation. Section 4.1 provides an ablation.

Full and partial observations FlowPDE naturally handles both *fully* and *partially* observable systems. **(i)** When the PDE problem is fully specified, the solution is unique. As discussed in Corollary 3.2, the target distribution degenerates to a Dirac measure at $\mathbf{x}^*$. Therefore, FlowPDE does not perform unconstrained generation in this setting. It learns a conditional transport from $\mathbf{x}_0 \sim p_0$ to the unique physical solution. During inference, we solve Eq. (12) from $t = 0$ to $t = 1$ conditioned on the full PDE specification. **(ii)** When only partial information $\tilde{P}$ is available, the solution may no longer be unique. Multiple physically plausible fields can be consistent with the same observation. Deterministic neural solvers tend to average these possibilities, resulting in mean-field collapse. FlowPDE instead models the conditional solution distribution, which is

$$p_{t=1}(\mathbf{x} \mid \tilde{P}), \tag{16}$$

where different initializations can be transported to different plausible solutions. Thus, FlowPDE behaves deterministically under full information and as a distributional solver under partial information.

4. Experiments

To verify the model's effectiveness, we test diverse Transformer-based neural solvers on four representative industrial simulation tasks that involve varying PDE conditions. Moreover, to examine the generality, we benchmark FlowPDE in both fully and partially observable settings.

Backbones and their FlowPDE variants We adopt three representative existing Transformer-based neural operators as the backbones: Transolver (Wu et al., 2024) that tackles complex geometries via Physics-Attention, Galerkin Transformer (Cao, 2021) that reformulates attention based on Galerkin projection, and GNOT (Hao et al., 2023) that designs a heterogeneous normalized attention layer. Based on these models, we develop their *FlowPDE variants*: Flow-Transolver, Flow-Galerkin, and Flow-GNOT. We refer to Figure 3(b) for the design details of these variants.

Table 1. Summary of experimental simulations. #Mesh records the size of the discretized meshes for each sample. #Variable records the varied simulation configurations among different samples.

Type	Benchmarks	#Mesh	#Variable	#Size
Aero-dynamics	DrivAerML	∼160M	Geometry	∼6TB
	NASA-CRM	∼450K	Geo, Speed, AoA	∼3GB
	AirCraft	∼330K	Geo, Speed, AoA, Slip	∼7GB
Crash	Car-Crash	∼1M	Impact Angle	∼8GB

Benchmarks As summarized in Table 1, we evaluate FlowPDE variants on four representative industrial-fidelity datasets spanning diverse physical regimes, boundary conditions, and mesh scales: DrivAerML (Ashton et al., 2024), NASA-CRM (Bekemeyer et al., 2025), AirCraft (Luo et al., 2025), and Car-Crash (Wu et al., 2026). Appendix B provides a detailed description of the datasets.

Implementation details We implement our experiments based on the Neural Solver Library (Wu et al., 2024). The models are configured with 8 layers, 8 attention heads, a hidden dimension of 256 (128 for GNOT due to excessive memory consumption), and an MLP ratio of 2. During training, we use a batch size of 1 for 200 epochs, following the configuration of GeoPT (Wu et al., 2026). The initial learning rate is set to 1×10^{-3}. We apply the 1cycle (Smith, 2018) scheduling policy to all backbones. For FlowPDE variants, we use 1cycle scheduling on the DrivAerML dataset, while employing a cosine annealing (Loshchilov and Hutter, 2017) schedule for all other datasets. We apply an Exponential Moving Average (EMA) (Morales-Brotons et al., 2024) with a decay rate of 0.999 to stabilize training. For the Flow Matching process, the time step t is sampled using a Logit-normal distribution. For the inference of the

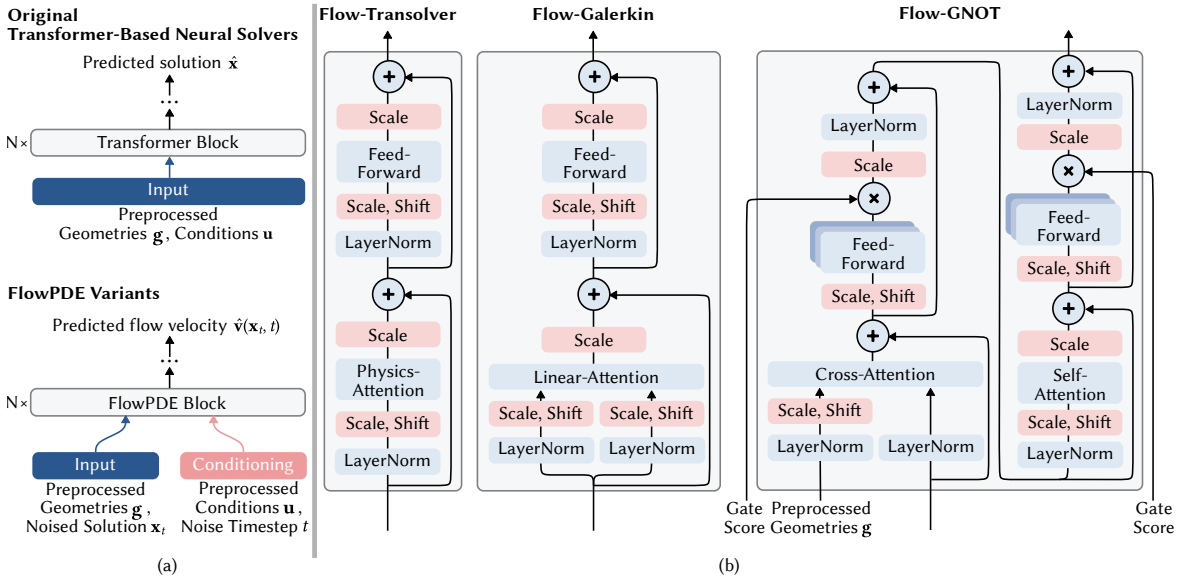

Figure 2. Architecture of FlowPDE, where the intermediate solution state $\mathbf{x}_t$ is processed as the main input, while the timestep t and PDE specification P are injected as conditioning signals through adaptive normalization. This fully conditioned design enables the learned velocity field to depend on the current state, time, geometry, boundary conditions, initial conditions, and equation parameters.

FlowPDE variants, we use the explicit Euler solver. Unless otherwise specified (see Section 4.3), we use 2 ODE steps for the full observation setting and 10 ODE steps for the partial observation setting. Additional implementation details are provided in Appendix C.

4.1. Full Observation

Setup We fully specify the PDE problem and compare the performance of existing backbones with their FlowPDE counterparts in the setting where the PDE solution is uniquely determined.

Performance improvements brought by FlowPDE As summarized in Table 2, the FlowPDE framework consistently and significantly enhances the performance of all tested backbones under the full observation setting. Specifically, FlowPDE achieves a lower relative L_2 error in 11 out of the 12 tested cases. The empirical gains are substantial: we observe a relative promotion of 21.16% for Flow-Transolver, 20.80% for Flow-Galerkin, and a remarkable 65.60% for Flow-GNOT. Notably, in the AirCraft task, Flow-GNOT achieves a 91.06% relative promotion compared to its deterministic backbone. To qualitatively illustrate these improvements, we provide detailed visualizations of the predicted fields and corresponding error maps for a representative AirCraft case in Appendix A.1. These results demonstrate that FlowPDE effectively pushes the accuracy boundaries of mainstream neural operators in complex scientific simulations.

Ablation study on conditioning mechanisms Building on the above experiments on AirCraft, we further investigate

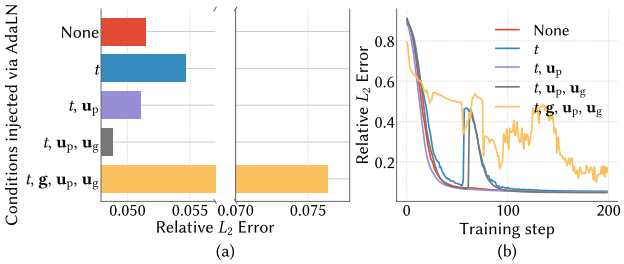

Figure 3. Results of the ablation study on conditioning mechanisms. Conditions that are not injected via AdaLN are incorporated through input concatenation.

the impact of how conditions are incorporated, either via direct input concatenation or AdaLN-based injection, on model performance. The results in Figure 3 show that injecting the timestep t, point-wise conditions $\mathbf{u}_p$, and global conditions $\mathbf{u}_g$ via AdaLN, while feeding the noise $\mathbf{x}_t$ and geometries $\mathbf{g}$ directly into the input, yields strong training and inference performance.

Ablation study on network prediction outputs Under full observation, we investigate two distinct network prediction paradigms within the Flow Matching framework: v-prediction (v-pred), which directly estimates the flow velocity, and x-prediction (x-pred), which predicts the clean solution x while optimizing via the velocity-based loss, as inspired by JiT (Li and He, 2026). As demonstrated in Table 2, the optimal paradigm is highly dependent on the choice of the backbone. Specifically, Flow-Transolver and Flow-Galerkin consistently exhibit superior performance under the v-pred setting. In contrast, Flow-GNOT suffers from a significant performance degradation when using v-pred,

Table 2. Performance comparison in terms of Relative L_2 Error↓. Under appropriately chosen network prediction outputs for different backbone models, the FlowPDE variants significantly and consistently outperform deterministic baselines across various architectures and datasets. Here, relative promotion↑ is defined as $1 - {}^{\text{Our error } (x\text{-pred}/v\text{-pred})}/_{\text{The baseline error}}$ on each benchmark.

MODELS	AIRCRAFT	CAR-CRASH	NASA-CRM	DRIVAERML	AVG. PROMOTION
TRANSOLVER	0.1197	0.2007	0.0922	**0.0904**	**21.16%**
FLOW-TRANSOLVER (x-PRED)	0.0526	0.1883	0.0880	0.1000	
FLOW-TRANSOLVER (v-PRED)	**0.0487**	**0.1779**	**0.0761**	0.0936	
RELATIVE PROMOTION	59.33%	11.38%	17.46%	-3.54%	
GALERKIN	0.0768	0.2079	0.0998	0.1074	**20.80%**
FLOW-GALERKIN (x-PRED)	0.0509	0.1849	0.0845	0.1032	
FLOW-GALERKIN (v-PRED)	**0.0481**	**0.1778**	**0.0767**	**0.0986**	
RELATIVE PROMOTION	37.41%	14.45%	23.18%	8.16%	
GNOT	0.5750	0.4175	0.5480	0.2888	**65.60%**
FLOW-GNOT (x-PRED)	**0.0514**	**0.2172**	**0.0968**	**0.1704**	
FLOW-GNOT (v-PRED)	0.6085	0.7189	1.0380	0.8050	
RELATIVE PROMOTION	91.06%	47.98%	82.34%	41.01%	

yet it achieves its best results under the x-pred paradigm. These findings emphasize that the choice of prediction target should be carefully tailored to the specific representation capacity of the underlying neural operator.

4.2. Partial Observation

Setup Without loss of generality, and for ease of experimentation, here we restrict the term "partial observation" to the setting where the PDE condition $\mathbf{u}$ injected via AdaLN is unknown. Under this view, we train an unconditional Transolver model and an unconditional Flow-Transolver model to evaluate the performance of FlowPDE under partial observation settings. Specifically, for Transolver, we mask the PDE condition inputs with a constant dummy tensor. For Flow-Transolver, we replace the PDE conditions with a single global learnable parameter. To eliminate the influence of geometric variations, we use the Car-Crash dataset, which features consistent geometry across samples.

Evaluation metrics To quantitatively evaluate the generative performance under this partial observation setting, standard point-to-point metrics, such as the relative L_2 error, are inapplicable. Moreover, due to the absence of a universal encoder for PDE problems, commonly used metrics in generation tasks such as FID (Heusel et al., 2018) are no longer applicable. To address this, we introduce the *min relative L_2 error* to measure whether a generated solution closely approximates *any* authentic physical state. Let $\{\mathbf{x}_{\text{GT}}\}$ denote the set of all physically compliant solutions within the dataset. For a given generated sample $\hat{\mathbf{x}}$, its min relative L_2 error is defined as its relative distance to the nearest neighbor on the true solution manifold:

$$\text{MinRel}_{L_2}(\hat{\mathbf{x}}, \{\mathbf{x}_{\text{GT}}\}) = \min_{\mathbf{x} \in \{\mathbf{x}_{\text{GT}}\}} \frac{\|\hat{\mathbf{x}} - \mathbf{x}\|_2}{\|\mathbf{x}\|_2}. \tag{17}$$

Here, a lower value effectively guarantees that the generated field falls onto a valid state within the true physical manifold, distinguishing reasonable physical solutions from non-physical hallucinations.

Solution distribution modeling capability In Figure 4, we present the qualitative results of the models when the PDE conditions are absent. Locally, the results highlight that the deterministic Transolver suffers from mean-field collapse, inevitably producing a single smoothed average state that lacks physical realism. In contrast, our Flow-Transolver successfully captures the underlying diverse distribution. Globally, Flow-Transolver achieves a significantly lower min relative L_2 error compared to the deterministic baseline. This substantial reduction indicates that the independently sampled outputs from Flow-Transolver lie much closer to the true physical manifold, confirming that FlowPDE produces highly faithful physical solutions rather than unconstrained hallucinations.

Physics observation alignment Inspired by Diffusion-PDE (Huang et al., 2024), we demonstrate that FlowPDE can be applied to align the full solution field given partial observations of the PDE solution. We use a pre-trained Flow-Transolver checkpoint from the above experiment and adopt the lightweight PnP-Flow (Martin et al., 2025) approach to modify its inference procedure, enabling supervision from partially observed solution fields during inference. Specifically, we randomly select a data point from the test set and mask a subset of vertices in its solution field to generate partial observations for alignment. We report the relative L_2 error over the full solution field, the relative L_2 error over the unobserved regions, as well as their respective variances. The results are shown in Figure 5. As the observation ratio increases, we observe that the full field gradually aligns with the complete physical field, exhibiting a gradually decreasing mean and variance of the error.

4.3. Model Analysis

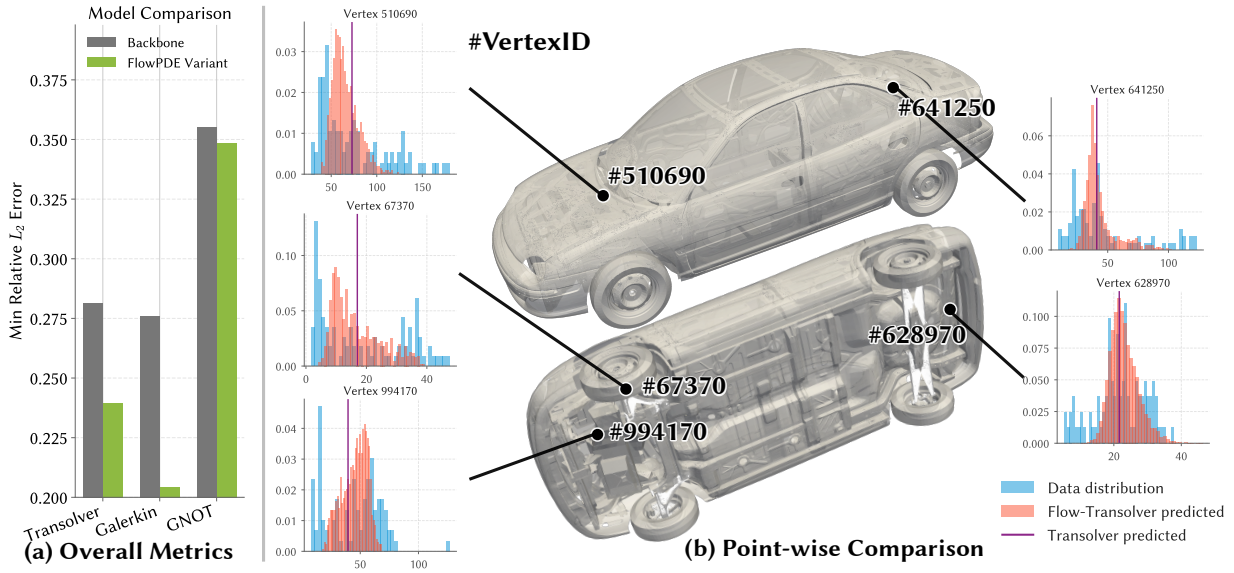

Figure 4. Main results under partial observation. In the unconditional setting, Flow-Transolver does not collapse to the mean solution, and significantly outperforms Transolver in both the global quantitative results shown in Figure (a) and the point-wise qualitative results shown in Figure (b).

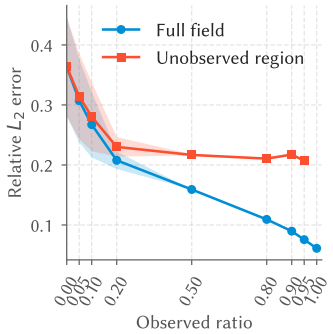

Figure 5. Results of physics observation alignment under various observed ratios.

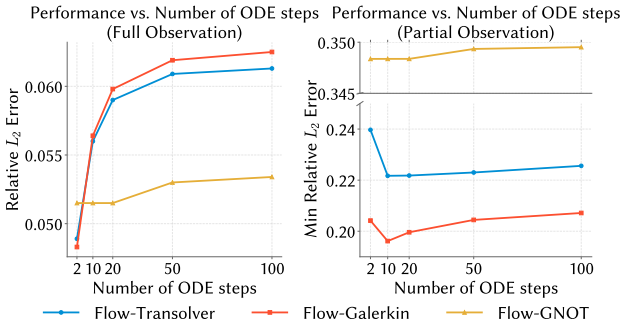

Figure 6. The performance curve changing *w.r.t.* the number of ODE steps under full and partial observation settings.

Performance curve *w.r.t.* inference ODE steps We study the relationship between model performance and the number of ODE steps during inference. Results on the AirCraft dataset are shown in Figure 6. We observe that, unlike standard Flow Matching models used in other tasks, FlowPDE achieves its best performance under full observation with a very *small* number of ODE steps. In contrast, under partial observation, Flow-Transolver and Flow-Galerkin exhibit non-monotonic behavior, where performance first improves and then degrades as the number of ODE steps increases, whereas Flow-GNOT still shows a monotonic trend.

5. Conclusion

In this work, we introduced FlowPDE, a Flow-Matching framework for neural PDE solving. Motivated by the connection between classical numerical solvers and continuous energy-descending dynamics, we showed that iterative PDE solvers can be reparameterized as deterministic conditional flow paths. This provides a theoretical justification for applying Flow Matching to PDE problems. Building on this insight, FlowPDE replaces hand-designed solver update rules with a neural velocity field conditioned on the full PDE specification, including geometry, boundary conditions, initial conditions, and equation parameters. This design preserves the refinement nature of numerical solvers while exploiting the nonlinear and nonlocal approximation ability of neural operators. Experiments across multiple industrial simulation tasks demonstrate that FlowPDE consistently improves deterministic prediction under full observation and effectively models physically plausible solution distributions under partial observation, mitigating the mean-field collapse of standard neural solvers. We hope this theoretical perspective can encourage broader exploration of frontier generative techniques for reliable and physically grounded neural PDE solving.

Acknowledgments

This work was supported by the Beijing Natural Science Foundation (QY26423) and the Tsinghua University Spark Scientific and Technological Innovation Fellowship. We thank Professor Feng Xu for providing computational resources.

Impact Statement

This work proposes FlowPDE, a framework that applies Flow Matching to neural operators for solving PDEs. From a positive perspective, we show that Flow Matching is well-suited for PDE solving tasks, and that under well-posed PDE settings the learned dynamics do not exhibit hallucination behavior. We also establish a connection between Conditional Flow Matching and classical numerical algorithms, providing a complementary viewpoint between learning-based and traditional PDE solvers. These results may benefit future research in scientific machine learning and operator learning.

In terms of applications, FlowPDE can improve the accuracy and robustness of neural operators under both fully and partially observed settings. This may be useful for a range of scientific and engineering problems involving PDEs, such as physical simulation and design tasks.

However, like other data-driven PDE solvers, the method may suffer from degraded performance under distribution shift or when applied outside the training regime, potentially leading to unreliable predictions if not properly validated. We therefore recommend that any deployment in safety-critical scenarios be accompanied by appropriate numerical verification.

References

J. Abramson, J. Adler, J. Dunger, R. Evans, T. Green, A. Pritzel, O. Ronneberger, L. Willmore, A. J. Ballard, J. Bambrick, et al. Accurate structure prediction of biomolecular interactions with alphafold 3. *Nature*, 2024.

J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

B. Alkin, A. Fürst, S. Schmid, L. Gruber, M. Holzleitner, and J. Brandstetter. Universal physics transformers: A framework for efficiently scaling neural operators. *NeurIPS*, 2024.

B. Alkin, M. Bleeker, R. Kurle, T. Kronlachner, R. Sonnleitner, M. Dorfer, and J. Brandstetter. AB-UPT: Scaling neural CFD surrogates for high- fidelity automotive aerodynamics simulations via anchored- branched universal physics transformers. *TMLR*, 2025.

N. Ashton, C. Mockett, M. Fuchs, L. Fliessbach, H. Hetmann, T. Knacke, N. Schonwald, V. Skaperdas, G. Fotiadis, A. Walle, et al. Drivaerml: High-fidelity computational fluid dynamics dataset for road-car external aerodynamics. *arXiv preprint arXiv:2408.11969*, 2024.

K. Azizzadenesheli, N. Kovachki, Z. Li, M. Liu-Schiaffini, J. Kossaifi, and A. Anandkumar. Neural operators for accelerating scientific simulations and design. *Nature Reviews Physics*, 2024.

P. Bekemeyer, N. Hariharan, A. M. Wissink, and J. Cornelius. Introduction of applied aerodynamics surrogate modeling benchmark cases. In *AIAA SCITECH 2025 Forum*, 2025.

K. Black, N. Brown, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai, L. Groom, K. Hausman, B. Ichter, S. Jakubczak, T. Jones, L. Ke, S. Levine, A. Li-Bell, M. Mothukuri, S. Nair, K. Pertsch, L. X. Shi, J. Tanner, Q. Vuong, A. Walling, H. Wang, and U. Zhilinsky. $\pi 0$: A vision-language-action flow model for general robot control. *ArXiv*, abs/2410.24164, 2024. URL https://api.semanticscholar.org/CorpusID:273811174.

S. Cao. Choose a transformer: Fourier or galerkin. In *NeurIPS*, 2021.

L. C. Evans. *Partial differential equations*. American Mathematical Soc., 2010.

H. Gao and S. Ji. Graph u-nets, 2019. URL https://arxiv.org/abs/1905.05178.

D. Gottlieb and S. A. Orszag. *Numerical analysis of spectral methods: theory and applications*. SIAM, 1977.

Z. Hao, Z. Wang, H. Su, C. Ying, Y. Dong, S. Liu, Z. Cheng, J. Song, and J. Zhu. Gnot: A general neural operator transformer for operator learning, 2023. URL https://arxiv.org/abs/2302.14376.

M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of research of the National Bureau of Standards*, 49:409–435, 1952. URL https://api.semanticscholar.org/CorpusID:2207234.

M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium, 2018. URL https://arxiv.org/abs/1706.08500.

J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. *NeurIPS*, 2020.

B. Holzschuh, G. Kohl, F. Redinger, and N. Thuerey. P3d: Highly scalable 3d neural surrogates for physics simulations with global context. In *ICLR*, 2026.

J. Huang, G. Yang, Z. Wang, and J. J. Park. Diffusionpde: Generative pde-solving under partial observation. *NeurIPS*, 2024.

A. Jaegle, F. Gimeno, A. Brock, A. Zisserman, O. Vinyals, and J. Carreira. Perceiver: General perception with iterative attention, 2021. URL https://arxiv.org/abs/2103.03206.

H. Jasak. Openfoam: Open source cfd in research and industry. *JNAOE*, 2009.

L. S. E. Jessica, N. A. Arafat, W. X. Lim, W. L. Chan, and A. W. K. Kong. Finite volume features, global geometry representations, and residual training for deep learning-based cfd simulation. *arXiv preprint arXiv:2311.14464*, 2023.

W. Kong, Q. Tian, Z. Zhang, R. Min, Z. Dai, J. Zhou, J.-L. Xiong, X. Li, B. Wu, J. Zhang, K. Wu, Q. Lin, J. Yuan, Y. Long, A. Wang, A. Wang, C. Li, D. Huang, F. Yang, H. Tan, H. Wang, J. Song, J. Bai, J. Wu, J. Xue, J. Wang, K. Wang, M. Liu, P.-Y. Li, S. Li, W. Wang, W. Yu, X. Deng, Y. Li, Y. Chen, Y. Cui, Y. Peng, Z. Yu, Z. He, Z. Xu, Z. Zhou, Z. Xu, Y.-D. Tao, Q. Lu, S. Liu, D. Zhou, H. Wang, Y. Yang, D. Wang, Y. Liu, J. Jiang, and C. Zhong. Hunyuanvideo: A systematic framework for large video generative models. *ArXiv*, abs/2412.03603, 2024. URL https://api.semanticscholar.org/CorpusID:274514554.

N. Kovachki, Z. Li, B. Liu, K. Azizzadenesheli, K. Bhattacharya, A. Stuart, and A. Anandkumar. Neural operator: Learning maps between function spaces with applications to pdes. *JMLR*, 2023.

T. Li and K. He. Back to basics: Let denoising generative models denoise, 2026. URL https://arxiv.org/abs/2511.13720.

Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Neural operator: Graph kernel network for partial differential equations, 2020. URL https://arxiv.org/abs/2003.03485.

Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Fourier neural operator for parametric partial differential equations, 2021. URL https://arxiv.org/abs/2010.08895.

Z. Li, N. B. Kovachki, C. Choy, B. Li, J. Kossaifi, S. P. Otta, M. A. Nabian, M. Stadler, C. Hundt, K. Azizzadenesheli, and A. Anandkumar. Geometry-informed neural operator for large-scale 3d PDEs. In *NeurIPS*, 2023a.

Z. Li, K. Meidani, and A. B. Farimani. Transformer for partial differential equations' operator learning, 2023b. URL https://arxiv.org/abs/2205.13671.

Z. Li, D. Shu, and A. B. Farimani. Scalable transformer for pde surrogate modeling, 2023c. URL https://arxiv.org/abs/2305.17560.

Z.-Y. Li, D. Z. Huang, B. Liu, and A. Anandkumar. Fourier neural operator with learned deformations for pdes on general geometries. *J. Mach. Learn. Res.*, 24:388:1–388:26, 2022. URL https://api.semanticscholar.org/CorpusID:250450893.

Y. Lipman, R. T. Q. Chen, H. Ben-Hamu, M. Nickel, and M. Le. Flow matching for generative modeling. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=PqvMRDCJT9t.

X. Liu, C. Gong, and Q. Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow, 2022. URL https://arxiv.org/abs/2209.03003.

I. Loshchilov and F. Hutter. Sgdr: Stochastic gradient descent with warm restarts, 2017. URL https://arxiv.org/abs/1608.03983.

C. Lu, Y. Zhou, F. Bao, J. Chen, C. Li, and J. Zhu. Dpm-solver: A fast ode solver for diffusion probabilistic model sampling in around 10 steps. *NeurIPS*, 2022.

H. Luo, H. Wu, H. Zhou, L. Xing, Y. Di, J. Wang, and M. Long. Transolver++: An accurate neural solver for pdes on million-scale geometries, 2025. URL https://arxiv.org/abs/2502.02414.

Y. Ma, H. Wu, H. Zhou, H. Weng, J. Wang, and M. Long. Physense: Sensor placement optimization for accurate physics sensing. In *NeurIPS*, 2025.

S. Martin, A. Gagneux, P. Hagemann, and G. Steidl. Pnp-flow: Plug-and-play image restoration with flow matching, 2025. URL https://arxiv.org/abs/2410.02423.

D. Morales-Brotons, T. Vogels, and H. Hendrikx. Exponential moving average of weights in deep learning: Dynamics and benefits, 2024. URL https://arxiv.org/abs/2411.18704.

W. Peebles and S. Xie. Scalable diffusion models with transformers. In *CVPR*, 2023.

I. Price, A. Sanchez-Gonzalez, F. Alet, T. R. Andersson, A. El-Kadi, D. Masters, T. Ewalds, J. Stott, S. Mohamed, P. Battaglia, et al. Probabilistic weather forecasting with machine learning. *Nature*, 2025.

M. Raissi, P. Perdikaris, and G. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2018.10.045. URL https://www.sciencedirect.com/science/article/pii/S0021999118307125.

R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer. High-resolution image synthesis with latent diffusion models. In *CVPR*, 2022.

Y. Saad and M. H. Schultz. Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869, 1986. doi: 10.1137/0907058. URL https://doi.org/10.1137/0907058.

R. Sawhney and K. Crane. Monte carlo geometry processing: A grid-free approach to pde-based methods on volumetric domains. *ACM Trans. Graph.*, 39(4), 2020.

L. N. Smith. A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay, 2018. URL https://arxiv.org/abs/1803.09820.

P. Ŝolín. *Partial differential equations and the finite element method*. John Wiley & Sons, 2005.

R. Temam. *Navier-Stokes equations: theory and numerical analysis*. American Mathematical Soc., 2001.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need, 2017. URL https://arxiv.org/abs/1706.03762.

H. Wang, T. Fu, Y. Du, W. Gao, K. Huang, Z. Liu, P. Chandak, S. Liu, P. Van Katwyk, A. Deac, et al. Scientific discovery in the age of artificial intelligence. *Nature*, 2023.

A. M. Wazwaz. Partial differential equations : methods and applications. 2002.

H. Wu, T. Hu, H. Luo, J. Wang, and M. Long. Solving high-dimensional pdes with latent spectral models, 2023. URL https://arxiv.org/abs/2301.12664.

H. Wu, H. Luo, H. Wang, J. Wang, and M. Long. Transolver: A fast transformer solver for pdes on general geometries, 2024. URL https://arxiv.org/abs/2402.02366.

H. Wu, M. Guo, Z. Li, Z. Dou, M. Long, K. He, and W. Matusik. Geopt: Scaling physics simulation via lifted geometric pre-training. 2026.

Z. Xiao, Z. Hao, B. Lin, Z. Deng, and H. Su. Improved operator learning by orthogonal attention. In R. Salakhutdinov, Z. Kolter, K. Heller, A. Weller, N. Oliver, J. Scarlett, and F. Berkenkamp, editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 54288–54299. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/xiao24c.html.

L. Zhang, A. Rao, and M. Agrawala. Adding conditional control to text-to-image diffusion models. In *CVPR*, 2023.

H. Zhou, Y. Ma, H. Wu, H. Wang, and M. Long. Unisolver: Pde-conditional transformers towards universal neural pde solvers. In *ICML*, 2025.

H. Zhou, H. Wu, H. Shangguan, Y. Ma, H. Weng, J. Wang, and M. Long. Transolver-3: Scaling up transformer solvers to industrial-scale geometries, 2026. URL https://arxiv.org/abs/2602.04940.

A. More Results

A.1. Showcase

To visually corroborate the substantial quantitative improvements reported in Section 4.1 (e.g., Table 2), we provide a qualitative comparison in this section. Figure 7 compares predicted fields and error maps on an AirCraft sample. Deterministic baselines often struggle with high-frequency details near complex geometries. By leveraging conditional priors, FlowPDE variants overcome this limitation, generating smoother and physically consistent predictions. Visualizations confirm that FlowPDE effectively suppresses local artifacts and significantly lowers error magnitudes, aligning with our prior quantitative findings.

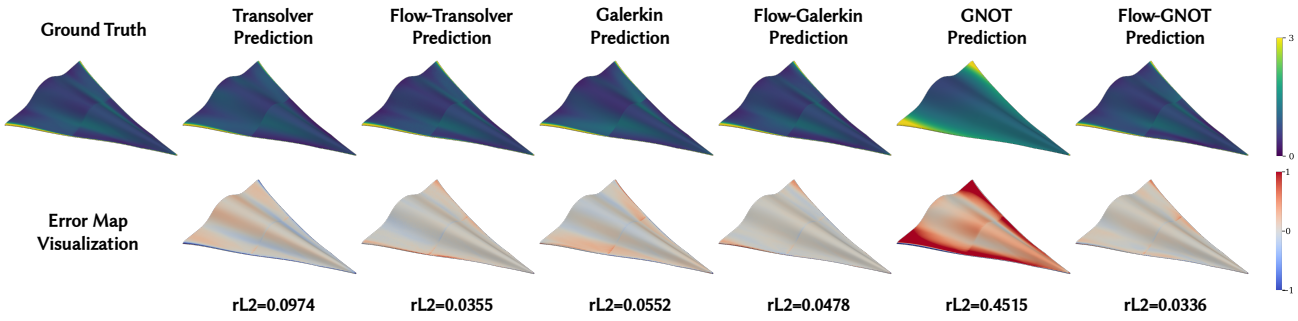

Figure 7. Visualization of predicted fields and error maps on an AirCraft case. Compared to the deterministic baselines, the FlowPDE variants effectively suppress local artifacts and achieve substantially lower error magnitudes across the computational domain.

A.2. Efficiency

Inference Efficiency Our proposed method requires solving an ODE trajectory through multiple integration steps, which inevitably introduces additional computational overhead compared to single-step deterministic backbones. To quantitatively assess this trade-off, we evaluate the average inference time per case on a single NVIDIA RTX 4090 GPU, as reported in Table 3.

As expected, integrating the FlowPDE framework (under both $\boldsymbol{x}$-pred and $\boldsymbol{v}$-pred paradigms) increases the inference time across all tested backbones. However, it is crucial to note that the absolute inference time remains highly efficient. Even on highly complex, industrial-scale meshes such as Car-Crash and NASA-CRM, the inference process only takes a fraction of a second to a few seconds per case. Given the substantial improvements in prediction accuracy and the ability to capture physical distributions, this modest increase in inference time is well within the acceptable range for practical scientific and engineering applications.

Table 3. Inference time comparison on the test set full-mesh. Values represent the average inference time (in seconds) per case measured on a single NVIDIA RTX 4090 GPU.

MODELS	AIRCRAFT	CAR-CRASH	NASA-CRM	DRIVAERML
TRANSOLVER	0.3636	1.1296	0.4900	0.0787
FLOW-TRANSOLVER ($\boldsymbol{x}$-PRED)	1.0350	3.3152	1.4175	0.2221
FLOW-TRANSOLVER ($\boldsymbol{v}$-PRED)	1.0357	3.3164	1.4158	0.2210
GALERKIN	0.7628	1.1924	0.4974	0.0788
FLOW-GALERKIN ($\boldsymbol{x}$-PRED)	1.1730	3.8141	1.5842	0.2497
FLOW-GALERKIN ($\boldsymbol{v}$-PRED)	1.1744	3.8176	1.5819	0.2504
GNOT	0.3671	2.4574	1.0290	0.1572
FLOW-GNOT ($\boldsymbol{x}$-PRED)	1.8174	5.9216	2.4643	0.3706
FLOW-GNOT ($\boldsymbol{v}$-PRED)	1.8154	5.9181	2.4694	0.3698

Performance under Comparable Memory Footprints To verify that our performance gains stem from the FlowPDE framework rather than mere parameter expansion, we evaluate all models under comparable memory footprints. As shown in Table 4, we increase the hidden dimension (h) of the deterministic baselines (denoted as *Scaled*) to match or slightly exceed the GPU memory consumption of their FlowPDE counterparts. The results indicate that naively scaling standard neural

operators yields marginal benefits and can even induce severe optimization instability. For instance, scaling Transolver on NASA-CRM drastically degrades the error from 0.0951 to 0.9074. Conversely, under equivalent memory budgets, FlowPDE variants consistently achieve substantial error reductions across all settings. This confirms that FlowPDE's superiority fundamentally arises from its continuous flow formulation, which explicitly refines solution trajectories, rather than brute-force capacity increases.

Table 4. Performance comparison under equivalent memory footprints. *Scaled* models denote deterministic baselines with expanded hidden dimensions to approximate the GPU memory (Mem., in MB) of their FlowPDE variants. Error is reported as Relative L_2 Error ↓.

Backbone	Models	AirCraft		NASA-CRM	
		Mem. (MB)	Error	Mem. (MB)	Error
Transolver	Baseline	4128	0.1194	6656	0.0951
	Scaled ($h: 256 \rightarrow 512$)	7428	0.1373	11162	0.9074
	Flow-Transolver (Ours)	6788	**0.0489**	11036	**0.0795**
Galerkin	Baseline	4140	0.0766	6320	0.1018
	Scaled ($h: 256 \rightarrow 512$)	7726	0.0703	11846	0.1039
	Flow-Galerkin (Ours)	7266	**0.0483**	11650	**0.0809**
GNOT	Baseline	7922	0.5736	12782	0.5503
	Scaled ($h: 128 \rightarrow 192$)	11352	0.5763	19174	0.5328
	Flow-GNOT (Ours)	10656	**0.0515**	17070	**0.0978**

A.3. Partial Observation

To evaluate the generative capability of our framework under partial observation, we conduct partial observation experiments on the Car-Crash dataset, where explicit PDE conditions are withheld. As shown in Table 5, deterministic baselines struggle in this ill-posed setting, inevitably suffering from mean-field collapse. In contrast, all FlowPDE variants achieve consistently lower Min Relative L_2 errors. This substantial reduction indicates that rather than predicting a non-physical blurred average, FlowPDE successfully captures the diverse posterior distribution, generating high-fidelity solutions that lie much closer to the true physical manifold.

Table 5. Quantitative comparison of partial observation tasks on the Car-Crash dataset. The evaluation metric is the Min Relative L_2 Error (MinRel_{L_2} ↓).

Backbone	MinRel_{L_2} Error ↓	
	Baseline	Flow (Ours)
Transolver	0.2815	**0.2397**
Galerkin	0.2759	**0.2042**
GNOT	0.3554	**0.3484**

A.4. Model Analysis

As discussed in Section 4.3 of the main text, we provide the complete quantitative results detailing the impact of the number of inference ODE steps on model performance. Table 6 shows that under full observation, FlowPDE optimally resolves the physical solution with minimal integration steps (typically 2), as additional steps slightly accumulate approximation errors. Conversely, Table 7 details the partial observation setting on the Car-Crash dataset, confirming the U-shaped trend discussed in the main text. Here, capturing the complex solution distribution benefits from moderate iterative refinement, achieving the lowest errors at 10 steps for Flow-Transolver and Flow-Galerkin.

A.5. Statistical Significance

We report the mean and standard deviation of relative L_2 error over multiple independent runs. Due to limited computational resources, all experiments reported here were repeated six times with different random seeds.

Table 8 summarizes the relative L_2 error results in the format of mean $\pm$ standard deviation. To assess statistical significance, we further construct a one-sided 99% confidence interval for the mean performance of each FlowPDE variant. Specifically, for a given benchmark with $n = 6$ runs, mean $\bar{x}$, and sample standard deviation s, we consider the upper confidence bound

Table 6. Performance comparison with different number of ODE steps under full observation. The evaluation metric is Relative L_2 Error ↓.

Models	Datasets	Number of ODE steps				
		2	10	20	50	100
Flow-Transolver	AirCraft	**0.0489**	0.0560	0.0590	0.0609	0.0613
	Car-Crash	**0.1789**	0.2044	0.2094	0.2129	0.2138
	NASA-CRM	**0.0795**	0.1116	0.1164	0.1193	0.1201
	DrivAerML	0.0935	**0.0933**	0.1000	0.1036	0.1066
Flow-Galerkin	AirCraft	**0.0483**	0.0564	0.0598	0.0619	0.0625
	Car-Crash	**0.1786**	0.1979	0.2016	0.2036	0.2045
	NASA-CRM	**0.0809**	0.0923	0.0981	0.1020	0.1016
	DrivAerML	**0.0985**	0.1151	0.1192	0.1224	0.1231
Flow-GNOT	AirCraft	**0.0515**	0.0515	0.0515	0.0530	0.0534
	Car-Crash	**0.2175**	0.2175	0.2175	0.2179	0.2180
	NASA-CRM	**0.0978**	0.0978	0.0978	0.1006	0.1014
	DrivAerML	**0.1704**	0.1704	0.1704	0.1714	0.1717

Table 7. Performance comparison on the Car-Crash dataset with different number of ODE steps under partial observation. The evaluation metric is the Min Relative L_2 Error ↓.

Models	Number of ODE steps				
	2	10	20	50	100
Flow-Transolver	0.2397	**0.2217**	0.2218	0.2230	0.2256
Flow-Galerkin	0.2042	**0.1961**	0.1996	0.2044	0.2071
Flow-GNOT	**0.3484**	0.3484	0.3484	0.3493	0.3495

$\bar{x} + t_{0.99,\, n-1} \frac{s}{\sqrt{n}}$. If this upper bound is still lower than the corresponding deterministic baseline, we regard the improvement as statistically significant at the 99% confidence level.

Consistent with table 2, FlowPDE variants outperform the deterministic baselines in 11 out of 12 settings. Moreover, all of these 11 improved cases satisfy the above 99% confidence criterion, indicating that the performance gains are not attributable to random seed fluctuations.

Table 8. Relative L_2 error results reported as mean ± standard deviation over six independent runs.

Method	AirCraft	Car-Crash	NASA-CRM	DrivAerML
Flow-Transolver	0.0479 ± 0.0014	0.1789 ± 0.0009	0.0802 ± 0.0022	0.1015 ± 0.0049
Flow-Galerkin	0.0497 ± 0.0020	0.1784 ± 0.0003	0.0818 ± 0.0060	0.0975 ± 0.0059
Flow-GNOT	0.0519 ± 0.0010	0.2264 ± 0.0196	0.0960 ± 0.0027	0.1750 ± 0.0049

B. Benchmark Details

AirCraft (Luo et al., 2025) focuses on aerodynamic simulations governed by Navier-Stokes equations, requiring models to predict pressure and velocity fields over complex 3D aircraft surface meshes.

Car-Crash (Wu et al., 2026) is a highly nonlinear structural mechanics benchmark where the model must predict the maximum Von Mises stress fields for vehicle meshes undergoing complex contact dynamics and varying impact deformations.

NASA-CRM (Bekemeyer et al., 2025) evaluates aerodynamic generalization by tasking models to predict high-fidelity transonic flow fields over realistic wing-body configurations under different flight conditions.

DrivAerML (Ashton et al., 2024) tests industrial scalability through multi-million-cell CFD simulations, demanding accurate aerodynamic predictions over massive, realistic road-car geometries.

Table 9. Detailed summary of benchmarks. #Mesh records the size of the discretized meshes for each sample. #Variable records the varied simulation configurations among different samples. #Train and #Test represent the number of training and test samples.

Type	Benchmarks	#Mesh	#Variable	#Size	#Train	#Test	Output Physics
Aero-dynamics	DrivAerML	∼160M	Geometry	∼6TB	100	20	Pressure & Velocity
	NASA-CRM	∼450K	Geo, Speed, AoA	∼3GB	105	44	Pressure Coefficient
	AirCraft	∼330K	Geo, Speed, AoA, Sideslip	∼7GB	100	50	Aerodynamic Six Components
Crash	Car-Crash	∼1M	Impact Angle	∼8GB	100	30	Time-max 2D Von Mises stress

C. More Implementation Details

Usage of EMA As shown in Table 10, incorporating EMA yields negligible performance differences but significantly stabilizes the training process. Therefore, baselines with EMA are used by default in our main experiments.

Table 10. Performance comparison of backbones with and without EMA, indicating that, for the backbone models, the effect of EMA is not significant. Therefore, enabling or disabling EMA in the baselines does not affect the experimental conclusions.

	Transolver			Galerkin			GNOT		
Settings	AirCraft	Car-Crash	NASA-CRM	AirCraft	Car-Crash	NASA-CRM	AirCraft	Car-Crash	NASA-CRM
w/o EMA	0.1043	0.1991	0.0923	0.0764	0.2005	0.1100	0.5876	0.4165	0.5424
w/ EMA	0.1194	0.2022	0.0951	0.0766	0.2094	0.1018	0.5736	0.4177	0.5503

Experimental environment All experiments are conducted on a machine equipped with 512 GB of memory and an AMD EPYC 7763 64-Core Processor. For most experiments with GPU memory consumption below 24 GB, we use an RTX 4090. For experiments requiring more than 24 GB of GPU memory, we use an RTX A6000.

D. Proofs

D.1. Convergence Toward One-Point Target Distributions

This appendix complements the main text by analyzing the convergence of the proposed flow dynamics toward the unique PDE solution. Since well-posed PDEs induce a one-point target distribution under full information, we study a straight-process idealization and derive an explicit Wasserstein-2 bound for the discretized learned dynamics. The result shows that both time discretization and velocity approximation errors vanish in the limit of fine resolution and accurate field estimation.

Lemma D.1. *(Extended Grönwall's inequality) Let $f : [a, b] \to \mathbb{R}$ be absolutely continuous, and let $p, g : [a, b] \to \mathbb{R}$ be integrable functions. Suppose that*

$$\frac{\mathrm{d}f(t)}{\mathrm{d}t} \le p(t)f(t) + g(t) \qquad \textit{for a.e. } t \in [a, b]. \tag{18}$$

Then, for every $t \in [a, b]$, we have

$$f(t) \le f(a) \exp\left(\int_a^t p(s)\,\mathrm{d}s\right) + \int_a^t g(\tau) \exp\left(\int_\tau^t p(s)\,\mathrm{d}s\right) \mathrm{d}\tau. \tag{19}$$

Proof. Define the integrating factor

$$\mu(t) \coloneqq \exp\left(-\int_a^t p(s)\,\mathrm{d}s\right). \tag{20}$$

Then

$$\frac{\mathrm{d}}{\mathrm{d}t}\big(\mu(t)f(t)\big) = \mu(t)\frac{\mathrm{d}f(t)}{\mathrm{d}t} - \mu(t)p(t)f(t) \le \mu(t)g(t) \qquad \text{for a.e. } t \in [a, b]. \tag{21}$$

Integrating both sides over $[a, t]$ gives

$$\mu(t)f(t) - \mu(a)f(a) \le \int_a^t \mu(s)g(s)\,\mathrm{d}s. \tag{22}$$

Since $\mu(a) = 1$, we obtain

$$\mu(t)f(t) \leq f(a) + \int_a^t \mu(s)g(s)\,\mathrm{d}s. \tag{23}$$

Multiplying both sides by $\mu(t)^{-1}$ yields

$$f(t) \leq f(a)\exp\left(\int_a^t p(s)\,\mathrm{d}s\right) + \int_a^t g(\tau)\exp\left(\int_\tau^t p(s)\,\mathrm{d}s\right)\mathrm{d}\tau. \tag{24}$$

This completes the proof. □

Lemma D.2. *(Velocity field property) Under the straight-process parameterization used in this paper, the velocity field satisfies*

$$\boldsymbol{v}(\boldsymbol{x}, t) - \boldsymbol{v}(\boldsymbol{x}', t) = \frac{\boldsymbol{x} - \boldsymbol{x}'}{1 - t}, \qquad \forall\, \boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^d,\ t \in [0, 1). \tag{25}$$

In particular,

$$\left\|\boldsymbol{v}(\boldsymbol{x}, t) - \boldsymbol{v}(\boldsymbol{x}', t)\right\|^2 = \frac{\|\boldsymbol{x} - \boldsymbol{x}'\|^2}{(1 - t)^2}. \tag{26}$$

Proof. The claim follows directly from the explicit straight-process velocity field. Taking norms on both sides gives the second identity. □

Assumption D.3. (Velocity field approximation error) Let $\boldsymbol{v}_t(\boldsymbol{x})$ denote the ground-truth velocity field and let $\boldsymbol{v}_{\theta,t}(\boldsymbol{x})$ denote the learned velocity field. Assume that

$$\int_{\mathbb{R}^d} \left\|\boldsymbol{v}_t(\boldsymbol{x}_t(\boldsymbol{x}_0)) - \boldsymbol{v}_{\theta,t}(\boldsymbol{x}_t(\boldsymbol{x}_0))\right\|^2 \pi_0(\boldsymbol{x}_0)\,\mathrm{d}\boldsymbol{x}_0 \leq \epsilon_t^2, \tag{27}$$

and that

$$\int_0^T \epsilon_t^2\,\mathrm{d}t \leq \bar{\epsilon}_T^2, \tag{28}$$

where $\bar{\epsilon}_T > 0$. Moreover, $\boldsymbol{x}_t(\boldsymbol{x}_0)$ denotes the ground-truth trajectory starting from $\boldsymbol{x}_0 \sim \pi_0$.

Theorem D.4. *(Convergence of model approximation and time discretization) Given a time discretization*

$$0 = t_0 \leq t_1 \leq \cdots \leq t_n = T < 1, \tag{29}$$

let

$$\Delta_T \coloneqq \max_{0 \leq k \leq n-1} (t_{k+1} - t_k). \tag{30}$$

Then

$$\text{Wasserstein-2}\left(\pi_T, \widetilde{\pi}_{\theta,T}\right) = \mathcal{O}\left(\frac{\Delta_T + \bar{\epsilon}_T}{1 - T}\right), \tag{31}$$

where π_T is the ground-truth distribution at time T and $\widetilde{\pi}_{\theta,T}$ is the discretized approximation induced by the learned velocity field.

Proof. By the definition of the Wasserstein-2 distance,

$$\text{Wasserstein-2}\left(\pi_t, \widetilde{\pi}_{\theta,t}\right)^2 \leq \int_{\mathbb{R}^d} \left\|\boldsymbol{x}_t(\boldsymbol{x}_0) - \widetilde{\boldsymbol{x}}_{\theta,t}(\boldsymbol{x}_0)\right\|^2 \pi_0(\boldsymbol{x}_0)\,\mathrm{d}\boldsymbol{x}_0. \tag{32}$$

Define

$$\mathcal{L}_t \coloneqq \int_{\mathbb{R}^d} \left\|\boldsymbol{x}_t(\boldsymbol{x}_0) - \widetilde{\boldsymbol{x}}_{\theta,t}(\boldsymbol{x}_0)\right\|^2 \pi_0(\boldsymbol{x}_0)\,\mathrm{d}\boldsymbol{x}_0. \tag{33}$$

Since the numerical trajectory $\widetilde{\boldsymbol{x}}_{\theta,t}$ is generated by Euler integration, it is piecewise linear. For $t \in [t_k, t_{k+1}]$, we have

$$\frac{\mathrm{d}\mathcal{L}_t}{\mathrm{d}t} = 2\int_{\mathbb{R}^d} \left\langle \boldsymbol{v}_t(\boldsymbol{x}_t(\boldsymbol{x}_0)) - \boldsymbol{v}_{\theta,t_k}\left(\widetilde{\boldsymbol{x}}_{\theta,t_k}(\boldsymbol{x}_0)\right), \boldsymbol{x}_t(\boldsymbol{x}_0) - \widetilde{\boldsymbol{x}}_{\theta,t}(\boldsymbol{x}_0)\right\rangle \pi_0(\boldsymbol{x}_0)\,\mathrm{d}\boldsymbol{x}_0. \tag{34}$$

We split the right-hand side into three parts:

$$\begin{aligned}\frac{\mathrm{d}\mathcal{L}_t}{\mathrm{d}t} &= 2\int_{\mathbb{R}^d} \left\langle \boldsymbol{v}_t(\boldsymbol{x}_t(\boldsymbol{x}_0)) - \boldsymbol{v}_t\big(\widetilde{\boldsymbol{x}}_{\theta,t}(\boldsymbol{x}_0)\big), \boldsymbol{x}_t(\boldsymbol{x}_0) - \widetilde{\boldsymbol{x}}_{\theta,t}(\boldsymbol{x}_0)\right\rangle \pi_0(\boldsymbol{x}_0)\,\mathrm{d}\boldsymbol{x}_0 \\ &\quad + 2\int_{\mathbb{R}^d} \left\langle \boldsymbol{v}_t\big(\widetilde{\boldsymbol{x}}_{\theta,t}(\boldsymbol{x}_0)\big) - \boldsymbol{v}_t\big(\widetilde{\boldsymbol{x}}_{\theta,t_k}(\boldsymbol{x}_0)\big), \boldsymbol{x}_t(\boldsymbol{x}_0) - \widetilde{\boldsymbol{x}}_{\theta,t}(\boldsymbol{x}_0)\right\rangle \pi_0(\boldsymbol{x}_0)\,\mathrm{d}\boldsymbol{x}_0 \\ &\quad + 2\int_{\mathbb{R}^d} \left\langle \boldsymbol{v}_t\big(\widetilde{\boldsymbol{x}}_{\theta,t_k}(\boldsymbol{x}_0)\big) - \boldsymbol{v}_{\theta,t_k}\big(\widetilde{\boldsymbol{x}}_{\theta,t_k}(\boldsymbol{x}_0)\big), \boldsymbol{x}_t(\boldsymbol{x}_0) - \widetilde{\boldsymbol{x}}_{\theta,t}(\boldsymbol{x}_0)\right\rangle \pi_0(\boldsymbol{x}_0)\,\mathrm{d}\boldsymbol{x}_0.\end{aligned} \tag{35}$$

For the first part, Lemma D.2 implies

$$\begin{aligned}&2\int_{\mathbb{R}^d} \left\langle \boldsymbol{v}_t(\boldsymbol{x}_t(\boldsymbol{x}_0)) - \boldsymbol{v}_t\big(\widetilde{\boldsymbol{x}}_{\theta,t}(\boldsymbol{x}_0)\big), \boldsymbol{x}_t(\boldsymbol{x}_0) - \widetilde{\boldsymbol{x}}_{\theta,t}(\boldsymbol{x}_0)\right\rangle \pi_0(\boldsymbol{x}_0)\,\mathrm{d}\boldsymbol{x}_0 \\ &= \frac{2}{1-t}\mathcal{L}_t.\end{aligned} \tag{36}$$

For the second part, since

$$\widetilde{\boldsymbol{x}}_{\theta,t}(\boldsymbol{x}_0) = \widetilde{\boldsymbol{x}}_{\theta,t_k}(\boldsymbol{x}_0) + (t-t_k)\boldsymbol{v}_{\theta,t_k}\big(\widetilde{\boldsymbol{x}}_{\theta,t_k}(\boldsymbol{x}_0)\big), \tag{37}$$

again using Lemma D.2 and Young's inequality, we obtain

$$\begin{aligned}&2\int_{\mathbb{R}^d} \left\langle \boldsymbol{v}_t\big(\widetilde{\boldsymbol{x}}_{\theta,t}(\boldsymbol{x}_0)\big) - \boldsymbol{v}_t\big(\widetilde{\boldsymbol{x}}_{\theta,t_k}(\boldsymbol{x}_0)\big), \boldsymbol{x}_t(\boldsymbol{x}_0) - \widetilde{\boldsymbol{x}}_{\theta,t}(\boldsymbol{x}_0)\right\rangle \pi_0(\boldsymbol{x}_0)\,\mathrm{d}\boldsymbol{x}_0 \\ &\le \int_{\mathbb{R}^d} \left\|\boldsymbol{v}_t\big(\widetilde{\boldsymbol{x}}_{\theta,t}(\boldsymbol{x}_0)\big) - \boldsymbol{v}_t\big(\widetilde{\boldsymbol{x}}_{\theta,t_k}(\boldsymbol{x}_0)\big)\right\|^2 \pi_0(\boldsymbol{x}_0)\,\mathrm{d}\boldsymbol{x}_0 + \mathcal{L}_t \\ &\le \left(\frac{t-t_k}{1-t}\right)^2 V + \mathcal{L}_t,\end{aligned} \tag{38}$$

where $V \coloneqq \max \|\boldsymbol{v}_t\|^2$ along the considered trajectory range.

For the third part, Assumption D.3 and Young's inequality give

$$\begin{aligned}&2\int_{\mathbb{R}^d} \left\langle \boldsymbol{v}_t\big(\widetilde{\boldsymbol{x}}_{\theta,t_k}(\boldsymbol{x}_0)\big) - \boldsymbol{v}_{\theta,t_k}\big(\widetilde{\boldsymbol{x}}_{\theta,t_k}(\boldsymbol{x}_0)\big), \boldsymbol{x}_t(\boldsymbol{x}_0) - \widetilde{\boldsymbol{x}}_{\theta,t}(\boldsymbol{x}_0)\right\rangle \pi_0(\boldsymbol{x}_0)\,\mathrm{d}\boldsymbol{x}_0 \\ &\le \int_{\mathbb{R}^d} \left\|\boldsymbol{v}_t\big(\widetilde{\boldsymbol{x}}_{\theta,t_k}(\boldsymbol{x}_0)\big) - \boldsymbol{v}_{\theta,t_k}\big(\widetilde{\boldsymbol{x}}_{\theta,t_k}(\boldsymbol{x}_0)\big)\right\|^2 \pi_0(\boldsymbol{x}_0)\,\mathrm{d}\boldsymbol{x}_0 + \mathcal{L}_t \\ &\le \epsilon_t^2 + \mathcal{L}_t.\end{aligned} \tag{39}$$

Therefore,

$$\frac{\mathrm{d}\mathcal{L}_t}{\mathrm{d}t} \le \left(2 + \frac{2}{1-t}\right)\mathcal{L}_t + \left(\frac{t-t_k}{1-t}\right)^2 V + \epsilon_t^2, \qquad t \in [t_k, t_{k+1}]. \tag{40}$$

Applying Lemma D.1 on each interval $[t_k, t_{k+1}]$ and recursively summing over $k = 0, \ldots, n-1$, we obtain

$$\mathcal{L}_{t_n} \le C_1 \frac{\Delta_T^2}{(1-T)^2} + C_2 \frac{\overline{\epsilon}_T^2}{(1-T)^2}, \tag{41}$$

for some constants $C_1, C_2 > 0$ independent of T. Consequently,

$$\text{Wasserstein-2}\left(\pi_T, \widetilde{\pi}_{\theta,T}\right) \le \mathcal{O}\left(\frac{\Delta_T + \overline{\epsilon}_T}{1-T}\right). \tag{42}$$

This proves the theorem. □

Lemma D.5. *(Convergence of the ground-truth process) Given the straight process from π_0 to the terminal Dirac distribution π_1, we have*

$$\text{Wasserstein-2}(\pi_T, \pi_1) \le \mathcal{O}(1-T). \tag{43}$$

Proof. By the straight-process construction, for every trajectory starting from $\boldsymbol{x}_0$ we have

$$\|\boldsymbol{x}_T(\boldsymbol{x}_0) - \boldsymbol{x}_1\| \le (1-T)\|\boldsymbol{x}_0 - \boldsymbol{x}_1\|. \tag{44}$$

Therefore,

$$\begin{aligned} \text{Wasserstein-2}(\pi_T, \pi_1)^2 &\le \int_{\mathbb{R}^d} \|\boldsymbol{x}_T(\boldsymbol{x}_0) - \boldsymbol{x}_1\|^2 \pi_0(\boldsymbol{x}_0)\,\mathrm{d}\boldsymbol{x}_0 \\ &\le (1-T)^2 \int_{\mathbb{R}^d} \|\boldsymbol{x}_0 - \boldsymbol{x}_1\|^2 \pi_0(\boldsymbol{x}_0)\,\mathrm{d}\boldsymbol{x}_0 \\ &= \mathcal{O}\big((1-T)^2\big). \end{aligned} \tag{45}$$

Taking square roots yields the claim. □

Theorem D.6 (Overall convergence to the one-point target)**.** *Under Assumption D.3, we have*

$$\text{Wasserstein-2}\left(\widetilde{\pi}_{\theta,T}, \pi_1\right) \le \mathcal{O}\left(1 - T + \frac{\Delta_T + \bar{\epsilon}_T}{1-T}\right). \tag{46}$$

In particular, if

$$T \to 1 \qquad \textit{and} \qquad \Delta_T + \bar{\epsilon}_T = o(1-T), \tag{47}$$

then

$$\text{Wasserstein-2}\left(\widetilde{\pi}_{\theta,T}, \pi_1\right) \to 0. \tag{48}$$

Equivalently, choosing $1 - T \asymp \sqrt{\Delta_T + \bar{\epsilon}_T}$ *yields the balanced rate*

$$\text{Wasserstein-2}\left(\widetilde{\pi}_{\theta,T}, \pi_1\right) = \mathcal{O}\left(\sqrt{\Delta_T + \bar{\epsilon}_T}\right). \tag{49}$$

Proof. By the triangle inequality,

$$\text{Wasserstein-2}\left(\widetilde{\pi}_{\theta,T}, \pi_1\right) \le \text{Wasserstein-2}\left(\widetilde{\pi}_{\theta,T}, \pi_T\right) + \text{Wasserstein-2}(\pi_T, \pi_1). \tag{50}$$

Combining Theorem D.4 and the previous lemma yields

$$\text{Wasserstein-2}\left(\widetilde{\pi}_{\theta,T}, \pi_1\right) \le \mathcal{O}\left(\frac{\Delta_T + \bar{\epsilon}_T}{1-T}\right) + \mathcal{O}(1-T), \tag{51}$$

which proves the first claim. The convergence condition follows immediately from

$$1 - T \to 0, \qquad \frac{\Delta_T + \bar{\epsilon}_T}{1-T} \to 0. \tag{52}$$

Finally, if $1 - T \asymp \sqrt{\Delta_T + \bar{\epsilon}_T}$, then the two terms are of the same order and the bound becomes

$$\mathcal{O}\left(\sqrt{\Delta_T + \bar{\epsilon}_T}\right). \tag{53}$$

This completes the proof. □

E. Limitations

From a theoretical perspective, although our work highlights the connection between classical numerical methods and CFM, we do not explicitly study the velocity fields induced by classical solvers, nor do we train neural networks to learn these velocity fields. Instead, we adopt the commonly used straight-line Flow Matching path. Investigating whether learning such solver-induced velocity fields could further guide the design of neural operators is an interesting direction for future work.

On the empirical side, we observe several intriguing phenomena that remain not yet fully understood. For example, the differing behaviors of x-pred (Li and He, 2026) and v-pred across backbones, as well as the counterintuitive relationship

between performance and ODE steps compared to standard generative tasks, are particularly noteworthy. However, we have not yet provided a satisfactory explanation for these observations, which we leave for future investigation. In addition, our partial-observation experiments are still in a preliminary stage, and we expect that more accurate metrics and improved approaches for modeling the posterior distribution of FlowPDE will be needed.

Regarding backbone selection, we do not include UPT (Alkin et al., 2024) in our study, primarily because its Perceiver-based architecture (Jaegle et al., 2021) makes it difficult to inject point-wise conditions into the latent space. More effective conditioning mechanisms, as well as more general FlowPDE designs, remain promising directions for future research.