

PUFM++: Point Cloud Upsampling via Enhanced Flow Matching

Zhi-Song Liu¹, Chenhang He², Roland Maier³, Andreas Rupp⁴

Received: XXX / Accepted: XXX

Abstract Recent advances in generative modeling have demonstrated strong promise for high-quality point cloud upsampling. In this work, we present PUFM++, an enhanced flow-matching framework for reconstructing dense and accurate point clouds from sparse, noisy, and partial observations. PUFM++ improves flow matching along three key axes: (i) geometric fidelity, (ii) robustness to imperfect input, and (iii) consistency with downstream surface-based tasks. We introduce a two-stage flow-matching strategy that first learns a direct, straight-path flow from sparse inputs to dense targets, and then refines it using noise-perturbed samples to approximate the terminal marginal distribution better. To accelerate and stabilize inference, we propose a data-driven adaptive time scheduler that improves sampling efficiency based on interpolation behavior. We further impose on-manifold constraints during sampling to ensure that generated points remain aligned with the underlying surface. Finally, we incorporate a recurrent interface network (RIN) to strengthen hierarchical feature interactions and boost reconstruction quality. Extensive experiments on synthetic benchmarks and real-world scans show that PUFM++ sets a new state of the art in point cloud upsampling, delivering superior visual fidelity and quantitative accuracy across a wide range of tasks. Code and pretrained models are publicly available at https://github.com/Holmes-Alan/Enhanced_PUFM.

¹ Department of Computational Engineering, Lappeenranta-Lahti University of Technology LUT, Finland

² The Hong Kong Polytechnic University

³ Institute for Applied and Numerical Mathematics at Karlsruhe Institute of Technology (KIT), Germany

⁴ Department of Mathematics at Saarland University, Germany

Keywords Point Clouds · Flow Matching · Upsampling · Testing-time Optimization

1 Introduction

In 3D data processing, point clouds are a fundamental yet versatile data structure, widely adopted in robotic navigation, industrial inspection, autonomous driving, and 3D scene reconstruction. Their unstructured nature enables efficient acquisition and processing; however, many downstream tasks, such as object recognition [17], surface reconstruction [11], and semantic scene understanding [48], require dense, uniformly distributed, and noise-free point clouds. Generating such high-quality data typically requires expensive, high-resolution 3D acquisition devices, including LiDAR systems and depth cameras.

To overcome the limitations of hardware-centric approaches, recent advances in deep learning for 3D geometry have spurred interest in learning-based point cloud upsampling [34, 1, 29], aiming to infer dense outputs from sparse inputs statistically. This task is conceptually related to image super-resolution [25, 26], yet introduces unique challenges due to the irregular, unordered nature of point sets, the presence of noise, and the prevalence of incomplete or topologically ambiguous shapes.

Traditional methods often formulate point cloud upsampling as an optimization problem, relying heavily on handcrafted geometric priors and deterministic rules. While effective to some extent, these approaches are generally limited in terms of generalization and scalability. The introduction of PUNet [50] marked a significant shift towards end-to-end learning, enabling the network to infer high-resolution point distributions di-

rectly from coarse inputs. Subsequent research further refined this line of work by modeling local geometric patches, thereby exploiting the manifold structure of point clouds and learning recurring patterns across spatial scales [49]. Despite these advances, the lack of expressive feature representations and globally consistent optimization has motivated the development of generative learning frameworks [18, 29, 37, 24], which seek to model complex point distributions more holistically.

Among these, diffusion probabilistic models [13] and flow-matching-based methods [20] have emerged as promising generative paradigms for 3D data synthesis. PUDM [37] and PUFM [24] are recent state-of-the-art techniques that leverage these principles for point cloud upsampling, showing strong performance across benchmarks. While diffusion models simulate a stochastic denoising process to generate data samples, flow matching provides a more direct and computationally efficient alternative by learning a continuous mapping from sparse to dense point sets.

In this work, we present PUFM++, a scalable flow-matching architecture designed for arbitrary point cloud upsampling across diverse object categories and geometric scales. Building upon our earlier framework PUFM [24], PUFM++ performs flow matching in the local patch space, enabling a more flexible and fine-grained modeling of geometric structures. Notably, PUFM++ achieves high-quality upsampling with as few as five inference steps, offering substantial improvements in efficiency while maintaining strong fidelity. Extensive experiments on multiple datasets demonstrate that PUFM++ achieves state-of-the-art performance. The analysis on different resolutions and real-world scenarios validates the superior performance, efficiency, and generalization to real-world examples.

Our contributions are summarized as follows:

- **Two-Stage Flow Matching Strategy.** We introduce a two-stage training framework for flow matching. The first stage optimizes the full transport trajectory between sparse and dense point distributions, while the second stage applies a dedicated refinement step that enhances initial inference quality, improving both fidelity and robustness.
- **Adaptive Time Sampling.** To accelerate inference, we propose an adaptive sampling scheme based on discretized ordinary differential equations (ODEs) that adjusts integration step sizes according to statistics derived from training trajectories, substantially reducing the number of sampling steps.
- **Test-Time Optimization and Post-Processing.** We incorporate a manifold-aware prior that en-

courages smooth transport trajectories, and apply a lightweight k NN-based post-processing module to improve local geometric consistency, leading to higher-quality downstream mesh reconstruction.

– Latent-State Recurrent Velocity Estimation.

To preserve long-range geometric consistency throughout the continuous reverse process, we extend the standard velocity network to a recurrent formulation with a compact evolving latent state. This memory mechanism provides persistent global shape context across noise levels, yielding marked improvements in fidelity, density uniformity, and structure preservation.

The basis of this work, PUFM, appeared at AAAI 2026 [24]. PUFM++ represents a major extension of PUFM in three primary aspects. (1) On the optimization side, PUFM++ integrates a two-stage flow matching scheme, statistics-guided time step sampling, and a manifold-aware prior, collectively addressing the convergence limitations of PUFM and substantially improving both efficiency and reconstruction fidelity (Sections 3.2.1–3.2.3). (2) Architecturally, PUFM++ replaces the original PointNet++ backbone with a recurrent latent self-conditioning mechanism inspired by Recurrent Interface Networks (RINs), enabling iterative refinement and improved spatial coherence, which are capabilities absent in the feedforward design of PUFM (Section 3.3). (3) Finally, we provide a significantly expanded empirical study, including more datasets, higher resolutions, and extensive real-world evaluations, demonstrating consistent improvements in fidelity, robustness, and generalization (Section 4).

2 Related Work

2.1 Point clouds and their analysis

Different from image data, point clouds contain irregular structures and are invariant to permutations. PointNet [3] is the pioneering work that applies shared MLPs to extract point-wise features. PointNet++ [32] significantly improves upon it by proposing a hierarchical architecture with set abstraction layers to extract multi-level features. This encoder-decoder framework has become a backbone for many subsequent tasks, including upsampling. Subsequent research has expanded in several directions. Graph-based methods like DGCNN [42] dynamically construct a graph to capture local structures. Convolution-based methods, such as PointConv [44] and KPConv [40], define continuous or discrete convolutions directly on point clouds. Voxel-based hybrids like PVCNN [27] combine

the efficiency of sparse voxel convolutions with the precision of point-based networks to improve computational efficiency. More recently, transformer-based architectures have demonstrated impressive performance. Models like Point Transformer [54] and its more efficient successor Point Transformer V2 [46] use vector self-attention to model long-range dependencies. Point Transformer V3 [45] further improves the efficiency for multiple downstream tasks by studying the trade-offs between accuracy and efficiency. Most recently, the DeepLA-Net [52] framework further popularized the use of deep neural networks for point cloud processing, surpassing SOTA performance on multiple tasks.

2.2 Learning-based point cloud upsampling

Point cloud upsampling aims to generate dense, uniform point sets that faithfully represent the underlying surface. Optimization-based and early learning-based methods often relied on handcrafted priors. The advent of deep learning led to pioneering data-driven works such as PU-Net [50], which uses a feature-expansion module followed by multi-layer perceptrons. EC-Net [51] introduced edge-aware training for better feature preservation. MPU [49] proposed a multi-patch progressive network to handle complex surfaces. To better model local geometry, PU-GCN [33] introduced a graph convolutional network with a NodeShuffle upsampling module. PUGAN [18] was the first to employ a generative adversarial network (GAN) [9] to encourage the generation of a uniform point distribution, mitigating the issue of clustered points. Dis-PU [19] later improved upon this by disentangling the upsampling and refinement stages. Another line of work explicitly models the underlying surface. PUGeo-Net [34] learns a geodesic mapping to project points onto a 2D parameterized domain for upsampling. Similarly, NePs [7] treats upsampling as a neural implicit surface reconstruction problem. SPU-Net [21] leverages self-attention and a learnable seed generation module to capture both global and local structures. PU-Transformer [36] expanded attention mechanisms to the transformer-based designs to capture long-range geometric dependencies. Recognizing that real-world point clouds are often noisy, recent methods have begun to integrate denoising. SUperPC [6] and PU-Dense [1] jointly perform upsampling and denoising, showing that the two tasks can be mutually beneficial. Despite these advances, a significant limitation of most prior work is the heavy reliance on the Chamfer Distance (CD) as the primary supervision signal. While computationally efficient, CD is known to be insensitive to the underlying distribution and can lead

to biased or non-uniform point distributions, failing to capture fine geometric details.

2.3 Diffusion model and flow matching

Generative models, particularly diffusion models [13] and flow matching methods [20], have recently achieved remarkable success in point cloud generation and processing. Diffusion model-based methods learn to denoise a Gaussian distribution into a data distribution. Diffusion Point Cloud [28] and PVD [56] are pioneering works that applied diffusion models to 3D point cloud generation and completion. GFNet [35] introduced flow matching for 3D shape synthesis, aligning source and target distributions through optimal transport trajectories learned from data. For upsampling, PUDM [37] frames the task as a conditional generation problem, where a sparse input guides the reverse diffusion process to generate a dense output. PDANS [53] further improves it by introducing the adaptive noise suppression (ANS) module to upsample point clouds against noise robustly. On the one hand, these methods often require 20 to 50 denoising steps, which slows inference. On the other hand, Normalizing Flows [31] learn an invertible mapping between a simple prior and a complex data distribution. PUFlow [29] is a seminal work that represents a 3D shape as an invertible flow over a latent space for point cloud upsampling. More recently, Flow Matching [20] has emerged as a simpler, more efficient alternative to diffusion models, offering a simulation-free training objective that achieves superior performance with fewer sampling steps. While applied to point cloud generation (PCFlow [43]), its potential for conditional tasks, such as upsampling, remains largely unexplored. One recent work is PUFM [24], which learn a straight flow between sparse and dense point clouds for upsampling. Due to its deterministic trajectory-based learning, it can efficiently upsample point clouds with as few as five steps.

3 Method

3.1 Preliminary

Flow matching (FM) [20] and its extension, conditional flow matching (CFM) [41], are effective frameworks for learning transport dynamics between probability distributions via ordinary differential equation (ODE) flows. The goal of FM is to learn a time-dependent vector field $\nu_\theta(\mathbf{x}, t)$, parameterized by a neural network with parameters θ , whose induced flow transports a source distribution ρ_0 to a target distribution ρ_1 . The resulting

probability density path $p_t(\mathbf{x})$ satisfies the continuity equation

$$\frac{\partial p_t(\mathbf{x})}{\partial t} + \nabla \cdot (p_t(\mathbf{x})\nu_\theta(\mathbf{x}, t)) = 0. \quad (1)$$

Directly learning the marginal vector field $\nu_\theta(\mathbf{x}, t)$ is generally intractable. CFM addresses this challenge by introducing a conditional probability path $p_t(\mathbf{x} | \mathbf{z})$ and a corresponding conditional vector field $u_t(\mathbf{x} | \mathbf{z})$, where \mathbf{z} denotes conditioning variables, typically the data endpoints. A commonly used conditional path is a Gaussian distribution whose mean follows a linear interpolation between the endpoints. That is,

$$p_t(\mathbf{x} | \mathbf{x}_0, \mathbf{x}_1) = \mathcal{N}(\mathbf{x} | \mu_t(\mathbf{x}_0, \mathbf{x}_1), \sigma_t^2 \mathbf{I}), \quad (2)$$

$$\mu_t(\mathbf{x}_0, \mathbf{x}_1) = t\mathbf{x}_1 + (1-t)\mathbf{x}_0, \quad (3)$$

where $\mathbf{x}_0 \sim \rho_0$ and $\mathbf{x}_1 \sim \rho_1$. The conditional vector field $u_t(\mathbf{x} | \mathbf{x}_0, \mathbf{x}_1)$ is defined such that its induced flow recovers this prescribed conditional path. The network parameters θ are learned by minimizing the CFM objective, which regresses the learned vector field ν_θ onto the conditional vector field. That is, we minimize

$$\mathbb{E}_{t, \mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_t} \left[\|\nu_\theta(\mathbf{x}_t, t) - u_t(\mathbf{x}_t | \mathbf{x}_0, \mathbf{x}_1)\|_2^2 \right], \quad (4)$$

where $t \sim \mathcal{U}[0, 1]$, the endpoint pairs $(\mathbf{x}_0, \mathbf{x}_1) \sim \pi(\mathbf{x}_0, \mathbf{x}_1)$ are sampled from a coupling π between ρ_0 and ρ_1 , and $\mathbf{x}_t \sim p_t(\cdot | \mathbf{x}_0, \mathbf{x}_1)$. In the special case of optimal transport conditional flow matching (OT-CFM), the conditional path is noise-free, corresponding to the limit $\sigma_t \rightarrow 0$. This results in a deterministic straight-line trajectory $\mathbf{x}_t = t\mathbf{x}_1 + (1-t)\mathbf{x}_0$, and the associated conditional vector field becomes time-independent for a fixed endpoint pair, i.e., $u_t(\mathbf{x}_t | \mathbf{x}_0, \mathbf{x}_1) = \mathbf{x}_1 - \mathbf{x}_0$.

During inference, the learned flow is approximated using a discrete-time Euler integrator. Given a step size τ , the state is updated as

$$\mathbf{x}_{t+\tau} = \mathbf{x}_t + \tau \nu_\theta(\mathbf{x}_t, t). \quad (5)$$

Despite their success, FM and CFM share limitations similar to those discussed in [39, 2]. First, standard FM formulations yield deterministic ODE trajectories and therefore cannot recover the stochastic dynamics required for Schrödinger Bridge problems. Second, while linear CFM provides an approximation to optimal transport displacement, it does not strictly solve the entropic optimal transport problem between ρ_0 and ρ_1 without carefully designed couplings π .

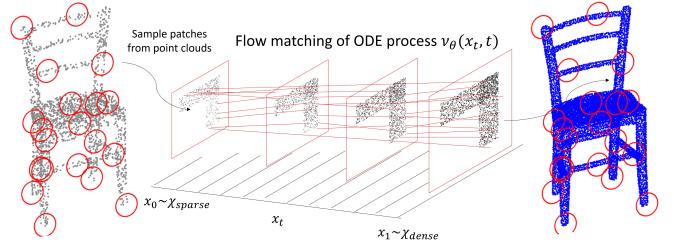


Fig. 1: Flow Matching for Point Cloud Upsampling. We extract paired patches from sparse and dense point clouds and learn the velocity field for point cloud upsampling.

3.2 Patch-based Point Cloud Upsampling via Flow Matching

We propose a patch-based point cloud upsampling via enhanced Flow Matching (PUFM++), mainly inspired by PUDM [37] and PUFM [24]. The former one learns a conditional diffusion model for point cloud upsampling, i.e., a mapping from noise to data. Thus, it is slow and thereby suboptimal. Contrarily, PUFM proposes to use flow matching to learn a sparse-to-dense distribution mapping and achieve an acceleration of more than factor 10. In this paper, we extend the work on PUFM and propose new techniques to improve it further, as explained in the following.

3.2.1 Two-stage flow matching optimization

Assume that we are given a dense point cloud $X_{dense} \in \mathbb{R}^{N \times 3}$ with N points and its sparse version $X_{sparse} \in \mathbb{R}^{M \times 3}$ with M points, where $M \ll N$. We initially densify the sparse point cloud via a midpoint interpolation [12] method and obtain the naively upsampled point cloud $X'_{dense} \in \mathbb{R}^{M \times 3}$. Then we extract corresponding patch pairs as $\mathbf{x}_0 \sim X'_{dense} \in \mathbb{R}^{Q \times 3}$ and $\mathbf{x}_1 \sim X_{dense} \in \mathbb{R}^{Q \times 3}$. As shown in Figure 1, our goal is to learn a flow matching model in the patch space by finding the optimal transport between \mathbf{x}_0 and \mathbf{x}_1 .

Stage 1: Pre-aligned Flow Matching Optimization. Based on (4), we learn a flow matching model to estimate the velocity field as $\nu_\theta(\mathbf{x}_t, t)$. In most cases, we assume that the underlying OT plan $\pi(\mathbf{x}_0, \mathbf{x}_1)$ is a pointwise correspondence between sparse and dense point clouds. Hence, we define the interpolants as in (3), where \mathbf{x}_1 and \mathbf{x}_0 are sampled from the training patches X'_{dense} and X_{dense} , respectively. However, point clouds are unordered in 3D space and are irregularly distributed. The true paired point patches should be sampled from the OT plan as pair $(\mathbf{x}_0, \mathbf{x}_1) \sim \pi^*$, which minimizes the expected cost of transporting mass from

the source to the target domain. That is,

$$\pi^* = \arg \min_{\pi \in \Pi(\mathbf{x}_0, \mathbf{x}_1)} \int \|\mathbf{x}_1 - \mathbf{x}_0\|^2 d\pi(\mathbf{x}_0, \mathbf{x}_1) \quad (6)$$

where $\Pi(\mathbf{x}_0, \mathbf{x}_1)$ is the set of all couplings (joint distributions) with margins of \mathbf{x}_0 and \mathbf{x}_1 .

To find the OT plan, we utilize the Wasserstein distance to measure the source and target distributions as $\rho_0 = \sum_{i=1}^N a_i \delta_{\mathbf{x}_0}$ and $\rho_1 = \sum_{i=1}^M b_i \delta_{\mathbf{x}_1}$, where $\delta_{\mathbf{x}_i}$ refers to the Dirac delta distribution centered at the point \mathbf{x}_i . The earth mover's distance (EMD), i.e., the Wasserstein 1-distance, is given by

$$\begin{aligned} \text{EMD}(\rho_0, \rho_1) \\ = \min_{\pi \in \Pi(\mathbf{x}_0, \mathbf{x}_1)} \sum_{i=1}^N \sum_{j=1}^M \pi_{ij} \|\mathbf{x}_0(i) - \mathbf{x}_1(j)\| \end{aligned} \quad (7)$$

where $x_0(i), x_1(j)$ represent the sparse and dense sampled points, $\sum_j \pi_{i,j} = a_i$ and $\sum_i \pi_{i,j} = b_j$. We then get a point-to-point correspondence between the sparse and dense point clouds. Notably, a major challenge of using EMD is its computational cost; PointMixup [4] proposes an EMD approximation via an auction algorithm. It aligns two point clouds by treating source points as bidders and target points as items, iteratively raising “prices” and reassigning matches so that each source gets its most cost-effective target, yielding an ϵ -approximate optimal transport plan π^ϵ in $\mathcal{O}(n^2)$ parallel time. That is, $\pi^\epsilon \in \Pi(\rho_0, \rho_1)$ with

$$\sum_{i,j} c_{i,j} \pi_{i,j}^\epsilon \leq \sum_{i,j} c_{i,j} \pi_{i,j} + \epsilon. \quad (8)$$

Here, $c_{i,j} = \|\mathbf{x}_1(i) - \mathbf{x}_0(j)\|^2$ is the cost of matching \mathbf{x}_1^i to \mathbf{x}_0^j . Utilizing this approximated EMD, we can find the matching function $\varphi: \{1, \dots, N\} \rightarrow \{1, \dots, M\}$ that assigns each dense point \mathbf{x}_1 to the sparse point \mathbf{x}_0 . Then we train a pre-aligned flow matching based on minimizing

$$\mathcal{L}_{\text{MSE}} = \mathbb{E}_{t \sim \mathcal{U}[0,1]} \|\nu_\theta(\mathbf{x}_t, t) - (\mathbf{x}_1 - \mathbf{x}_0)\|^2. \quad (9)$$

where t is sampled from the uniform distribution $\mathcal{U}[0, 1]$.

Stage 2: Endpoint Flow Matching Refinement. We propose a second-step refinement on the pre-aligned flow matching model to boost the quality of point cloud upsampling further. In (8), we perform OT-guided flow matching by regressing velocities along displacement paths built from an (approximate) optimal assignment π^ϵ , which teaches locally consistent transport directions. It does not guarantee that the global pushforward matches ρ_1 once we integrate from only \mathbf{x}_0 (no access to \mathbf{x}_1 at the sampling time), nor that the density is correct where OT pairings were noisy or sparse.

Algorithm 1 Two-Stage Flow Matching for Point Cloud Upsampling

Require: $\mathcal{X}_{\text{sparse}}, \mathcal{X}_{\text{dense}}$, noise scale σ

Stage 1: Pre-aligned Flow Matching

for each training iteration **do**

$$\begin{aligned} \mathbf{x}_0 &\sim \mathcal{X}_{\text{sparse}}, \quad \mathbf{x}_1 \sim \mathcal{X}_{\text{dense}} \\ t &\leftarrow 1 - \cos(s\pi/2), \quad s \sim \mathcal{U}[0, 1] \\ \tilde{\mathbf{x}}_0 &\leftarrow \text{mid}(\mathbf{x}_0, \eta) \quad \triangleright \text{midpoint interpolation} \\ \varphi^* &\leftarrow \pi(\tilde{\mathbf{x}}_0, \mathbf{x}_1) \quad \triangleright \text{OT pre-alignment (auction)} \\ \mathbf{x}_t &\leftarrow (1-t)\tilde{\mathbf{x}}_0 + t\mathbf{x}_1^{\varphi^*(i)} \\ \mathcal{L}_{\text{MSE}} &\leftarrow \|\nu_\theta(\mathbf{x}_t, t) - (\mathbf{x}_1^{\varphi^*(i)} - \tilde{\mathbf{x}}_0)\|_2^2 \\ \text{Update } \theta \text{ using } \mathcal{L}_{\text{FM}} \end{aligned}$$

end for

Stage 2: Endpoint Flow Matching Refinement

for each refinement iteration **do**

$$\begin{aligned} \mathbf{x}_0 &\sim \mathcal{X}_{\text{sparse}}, \quad \mathbf{x}_1 \sim \mathcal{X}_{\text{dense}} \\ \mathbf{x}'_0 &\leftarrow \mathbf{x}_0 + \sigma \xi, \quad \xi \sim \mathcal{N}(0, I) \quad \triangleright \text{additional noise} \\ \mathbf{x}_{\text{pred}} &\leftarrow \mathbf{x}'_0 + \nu_\theta(\mathbf{x}'_0, 0) \\ \mathcal{L}_{\text{CD}} &\leftarrow \text{CD}(\mathbf{x}_{\text{pred}}, \mathbf{x}_1) \\ \text{Update } \theta \text{ using } \mathcal{L}_{\text{CD}} \end{aligned}$$

end for

In the second step, we integrate trajectories from the source only and minimize a permutation-invariant chamfer distance (CD) loss between the sparse and dense point clouds. Furthermore, we add additional noise to the sparse point clouds to make the learned flow robust to input perturbations and encourage local coherence of trajectories. The desired value is then given by

$$\begin{aligned} \mathcal{L}_{\text{CD}} = \frac{1}{\|\tilde{\mathbf{x}}_1\|} \sum_{\tilde{\mathbf{x}}_1(i) \in \tilde{\mathbf{x}}_1} \min_{\mathbf{x}_1(i) \in \mathbf{x}_1} \|\tilde{\mathbf{x}}_1(i) - \mathbf{x}_1(i)\|^2 \\ + \frac{1}{\|\mathbf{x}_1\|} \sum_{\mathbf{x}_1(i) \in \mathbf{x}_1} \min_{\tilde{\mathbf{x}}_1(i) \in \tilde{\mathbf{x}}_1} \|\mathbf{x}_1(i) - \tilde{\mathbf{x}}_1(i)\|^2, \end{aligned} \quad (10)$$

where $\tilde{\mathbf{x}}_1 = \nu_\theta(\mathbf{x}_0 + \xi, 0)$ with $\xi \sim \mathcal{N}(0, \sigma^2 I)$. This enforces the terminal marginal constraints, reminiscent of the Schrödinger system, adjusting the potentials so that the learned distribution ρ_1^θ approximately equals the true ρ_1 . More importantly, the additional noise ξ introduces a smoothing regularizer into the flow-matching model. It reduces the gradient variance across nearby initial sparse points and pushes the learned field to transform the neighborhoods of $\mathbf{x}_0(i)$ toward $\mathbf{x}_1(i)$ consistently. The summary is shown in Algorithm 1.

3.2.2 Adaptive time scheduler

As shown in (5), we typically adopt a uniform time interval τ when solving the ODE for inference. However, the learned model dynamics are not uniform in time: in certain regions, the velocity field changes rapidly and requires a finer temporal resolution, while in other regions, it evolves smoothly and can be handled with

coarse steps. To address this, we propose an adaptive time scheduler (ATS), which dynamically allocates ODE sampling steps according to the temporal complexity of the learned dynamics. It assigns more steps where the model is difficult to predict and fewer steps where it is smooth.

To construct the ATS, we connect the time scheduler to the training loss in (9). Let the per-timestep MSE loss be denoted as a time-dependent function $\mathcal{L}_{\text{MSE}}(t)$. We convert this into a difficulty density, which reads

$$\omega(t_i) = (\mathcal{L}_{\text{MSE}}(t_i) + \psi)^\beta, \quad \beta > 0, \psi \geq 0, \quad (11)$$

where $\{t_i\}_{i=0}^K$ is a uniformly sampled training grid on $[0, 1]$. The hyperparameter β controls the sharpness of the density and ψ prevents degeneracy.

From this density, we form a discrete cumulative distribution function (CDF), given by

$$F_i = \frac{\sum_{j=0}^i \omega(t_j)}{\sum_{j=0}^K \omega(t_j)}, \quad i = 0, \dots, K. \quad (12)$$

It satisfies $F_0 = 0$ and $F_K = 1$. This CDF is monotonously increasing and reflects the relative difficulty of different time intervals.

Given a target sampling budget of S inference steps, we define uniformly spaced mass levels

$$o_s = \frac{s}{S}, \quad s = 0, \dots, S.$$

For each o_s , we find the interval $[F_{i-1}, F_i]$ such that $F_{i-1} \leq o_s < F_i$ and compute the corresponding adaptive time using a piecewise-linear inverse transform sampling. That is,

$$t_s^* = t_{i-1} + \frac{o_s - F_{i-1}}{F_i - F_{i-1}} (t_i - t_{i-1}). \quad (13)$$

The resulting sequence $0 = t_0^* < t_1^* < \dots < t_S^* = 1$, constitutes our ATS.

To compute $\mathcal{L}_{\text{MSE}}(t_i)$ in practice, we freeze the pre-trained model and evaluate it over the training dataset on a fixed uniform time grid of $K = 50$ points (empirically chosen). We then construct the density $\omega(t_i)$, build the CDF in (12), and finally obtain the inference-time schedule $\{t_s^*\}_{s=0}^S$ using (13). This reparameterization allocates more ODE steps to regions with high training difficulty and fewer steps to smooth regions, improving reconstruction fidelity without additional training.

Algorithm 2 Sampling with Adaptive Time Scheduler

Require: $\mathbf{x}_0 \sim \mathcal{X}_{\text{sparse}}$
 $\tilde{\mathbf{x}}_0 = \text{mid}(\mathbf{x}_0, \eta)$
Compute per-timestep loss $\mathcal{L}_{\text{MSE}}(t_i)$ on the training grid $\{t_i\}_{i=0}^K$
Compute difficulty weights ω_i using (11)
Build the discrete CDF F_i using (12)
Define adaptive time schedule $\{t_i^*\}_{i=0}^K$ via inverse transform sampling using (13)
for $k = 0$ **to** $K - 1$ **do**
 $\delta = t_{k+1} - t_k$
 $\kappa = \text{CurvatureEstimate}(\mathbf{x}_{t_k})$ \triangleright per-point curvature
 $\mathbf{w} = 1 + \alpha_{\text{cur}} \kappa$ \triangleright curvature-based weights
 $\mathbf{x}_{t_{k+1}} = \mathbf{x}_{t_k} + \delta(\mathbf{w} \odot \nu_\theta(\mathbf{x}_{t_k}, t_k))$
end for

Post processing

$\mathbf{x}_{k\text{NN}} = k\text{NN}(\mathbf{x}_{t_{k+1}}, \tilde{\mathbf{x}}_0)$ \triangleright $k\text{NN}$ search
Compute manifold loss $\mathcal{L}_{\text{manifold}} = \|\mathbf{x}_{k\text{NN}} - \tilde{\mathbf{x}}_0\|_2$
Update $\mathbf{x}_{t_{k+1}} \leftarrow \mathbf{x}_{t_{k+1}} - \alpha \nabla_{\mathbf{x}_{t_{k+1}}} \mathcal{L}_{\text{manifold}}$

3.2.3 Testing time optimization and post processing

We also introduce additional manifold constraints into the sampling process to ensure surface smoothness after point cloud upsampling. Firstly, we estimate curvature on the updated point cloud and learn a weighted velocity for updating. Let the model-predicted point cloud be denoted by $\mathbf{x}_t = \{\mathbf{a}_{i=1}^N\}$. Then, we estimate the curvature score κ by computing

$$\begin{aligned} \mathbf{C}_i &= \frac{1}{k} \sum_{\mathbf{a}_j \in \mathcal{N}_k(\mathbf{a}_i)} (\mathbf{a}_j - \bar{\mathbf{a}}_i)(\mathbf{a}_j - \bar{\mathbf{a}}_i)^\top, \\ \mathbf{C}_i \mathbf{v}_j &= \lambda_j \mathbf{v}_j, \quad \text{with} \quad \lambda_1 \leq \lambda_2 \leq \lambda_3, \\ \kappa_i &= \frac{\lambda_1}{\lambda_1 + \lambda_2 + \lambda_3}, \end{aligned} \quad (14)$$

where $\mathcal{N}_k(\mathbf{a}_i)$ denotes the k -nearest neighbors ($k\text{NN}$) of point \mathbf{a}_i , and $\bar{\mathbf{a}}_i = \frac{1}{k} \sum_{\mathbf{a}_j \in \mathcal{N}_k(\mathbf{a}_i)} \mathbf{a}_j$ is the local mean of these neighbors. We compute the local covariance matrix \mathbf{C}_i and apply eigen decomposition to obtain the principal eigenvalues $\lambda_{\{1,2,3\}}$. The curvature score κ is now used to weight the estimated velocity $\nu_\theta(\mathbf{x}_{t_k}, t_k)$ by a small learning rate α_{cur} . Further details are presented in Algorithm 2, where the curvature-based weights ω is multiplied with the estimated velocity for point cloud updating (\odot denotes the element-wise multiplication).

In addition, given the t -th upsampled point cloud \mathbf{x}_t , we use $k\text{NN}$ again to find neighborhoods in the original sparse input $\tilde{\mathbf{x}}_0$, and we define the manifold constraint as the distance between them. We calculate the distance gradient with respect to the upsampled point cloud, then obtain the correct direction that moves the upsampled points closer to the underlying surface manifold. We update the point cloud with a small learning rate α (see post-processing in Algorithm 2).

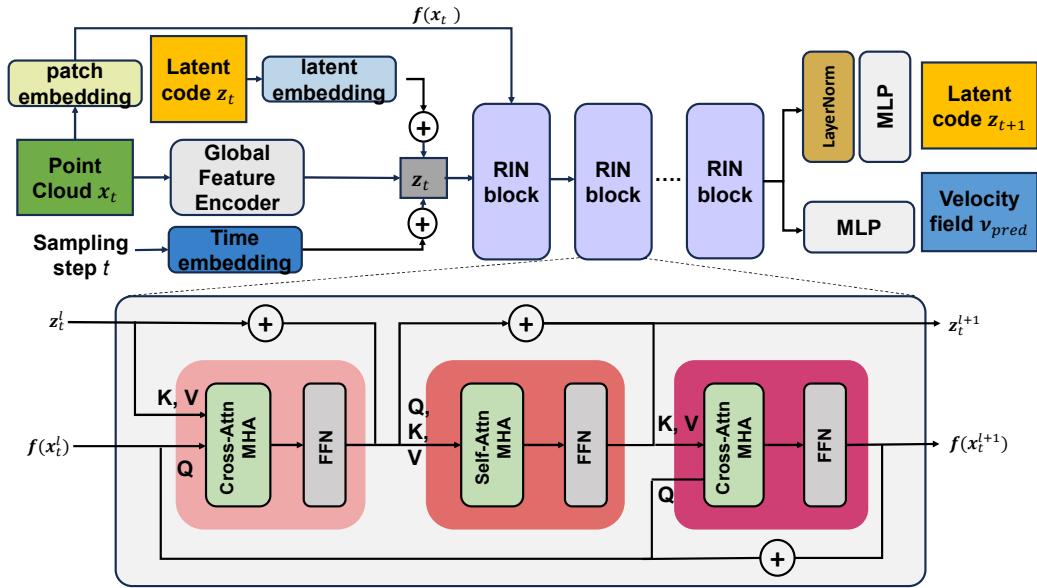


Fig. 2: Overview of the proposed iterative point cloud upsampling network. The network takes the current sampled point cloud \mathbf{x}_t , sampling step t , and previous latent code z as input. A point feature encoder extracts features $f(\mathbf{x}_t)$. The latent interface z_t is initialized by conditioning on time and global embedding. The core processing consists of stacked RIN blocks. The network outputs the estimated velocity field ν_θ and the updated latent code z_{t+1} for the next iteration.

3.3 Latent-state Recurrent Velocity Estimation

Standard flow matching models typically estimate the velocity field ν_θ as a stateless function of the current geometry \mathbf{x}_t and time t . However, point cloud generation is an inherently evolving process; relying solely on the instantaneous state \mathbf{x}_t forces the network to re-infer global structure from scratch at every step, often leading to temporal inconsistencies. To address this, we propose a state-dependent estimator that explicitly maintains a global context. We extend the formulation to have ν additionally depend on z_t , where the network predicts both the velocity and the updated latent state for the next step, i.e.,

$$[\nu_{pred}, z_{t+1}] = \nu_\theta(\mathbf{x}_t, z_t, t). \quad (15)$$

Here, z_t acts as an evolving memory and its initialization z_{init} comprises sinusoidal time embeddings and global features. This mechanism ensures that the velocity prediction is guided by the accumulated geometric history. To implement this state-dependent function, we adopt a recurrent interface network (RIN) [14] architecture, which utilizes a set of learnable latent tokens to represent z , and interacts with point features $f(\mathbf{x}_t^l)$ via a stacked Read-Compute-Write attention mechanism (where l denotes the l -th layer feature maps), which

reads

$$\begin{aligned} \text{Read: } & z_t^{l+1} = z_t^l + \text{MHA}(z_t, f(\mathbf{x}_t^l)) \\ & z_t^{l+1} = z_t^l + \text{MLP}(z_t^l) \\ \text{Compute: } & z_t^{l+1} = z_t^l + \text{MHA}(z_t^l, z_t^l) \\ & z_t^{l+1} = z_t^l + \text{MLP}(z_t^l) \\ \text{Write: } & f(\mathbf{x}_t^{l+1}) = f(\mathbf{x}_t^l) + \text{MHA}(f(\mathbf{x}_t^l), z_t^l) \\ & f(\mathbf{x}_t^{l+1}) = f(\mathbf{x}_t^l) + \text{MLP}(f(\mathbf{x}_t^l)), \end{aligned} \quad (16)$$

where MHA stands for multi-head attention. This structure allows the network to aggregate sparse local information into a dense global representation (Read), process it abstractly (Compute), and redistribute the refined context back to the points (Write).

Training with two-pass estimation. A challenge arises during training because time steps t are sampled independently, making the sequential history z_{t-1} unavailable. To simulate recurrence, we employ a two-pass strategy. In the first pass, we estimate a proxy latent state \tilde{z}_t using a null initialization. That is,

$$\tilde{v}_{pred}, \tilde{z}_t = \nu_\theta(\mathbf{x}_t, \mathbf{0}, t). \quad (17)$$

In the second pass, this proxy latent, acting as the “remembered” context, is used to condition the network for the final prediction, which reads

$$\nu_{pred}, z_{t+1} = \nu_\theta(\mathbf{x}_t, \text{sg}(z_t), t), \quad (18)$$

where $\text{sg}(\cdot)$ denotes the stop-gradient operation. This effectively trains the model to utilize its own latent predictions, bridging the gap between parallel training and sequential inference.

4 Experiments

4.1 Experimental Details

Dataset. We use two publicly available datasets for training and evaluation, PUGAN [18] and PU1K [33]. We follow the same procedure as in [49] to extract paired sparse and dense point clouds. Given the 3D meshes, we first use Poisson disk sampling to generate dense point clouds, then use farthest point sampling (FPS) to select centroids, and finally use k NN to form dense patches for all centroids. Each dense patch contains 1024 points. Then we randomly sample 256 points from each dense patch to obtain the corresponding sparse patch. For testing, we obtain 27 and 127 point clouds from PUGAN and PU1K, respectively. Moreover, we also use three real-world point clouds for further analysis using the datasets ScanNet [5], KITTI [8], and ABC [16].

Evaluation metrics. To measure point cloud upsampling quality, we use the Chamfer distance (CD), the Hausdorff distance (HD), and point-to-surface (P2F) as the evaluation metrics in our experiments. Additionally, we also use the Jensen-Shannon divergence (JSD) score to measure the point cloud distribution. We divide the 3D space into voxels using a k -d tree, then measure the average KL divergence between pairs of voxels. When JSD is close to zero, it indicates that the two point clouds are (almost) identical. We are also interested in mesh reconstruction from upsampled point clouds for downstream applications. Hence, we use normal consistency (NC), i.e., the mean absolute cosine of normals in one mesh and normals at nearest neighbors in the other mesh. The area-length ratio (ALR) [10] measures the shape quality of individual triangles within a mesh. This ratio attains its maximum value of 1 for equilateral triangles, indicating optimal shape quality. The manifoldness rate (MR) [10] quantifies the proportion of edges in a mesh that satisfy manifold conditions. An edge is considered a manifold if exactly two faces share it. A rate approaching 1 indicates a well-structured mesh suitable for operations like Boolean computations and 3D printing. Conversely, a lower rate highlights the presence of non-manifold edges, which may necessitate mesh repair or refinement.

Training settings. We modify the RIN [14] architecture and use it as our velocity field model ν_θ . As summarized in Algorithm 1, the training process comprises

two stages. For stage 1 of pre-aligned flow matching optimization, we mainly follow the setting in PUFM [24] and use a batch size of 32 and a learning rate of 10^{-4} . We train the model for 100 epochs with the Adam optimizer [15]. The average losses for different sampling steps are used for CDF computation (Equation 12) and saved for the inference stage. For state 2 of endpoint flow matching refinement, we fine-tune the model using a learning rate of 10^{-5} for 50 epochs. We define the noise scale $\sigma = 0.02$ for the additional Gaussian noise. For data augmentation, we randomly apply rotation and shifting to the sparse and dense point clouds. **Inference settings.** For the inference, we first apply FPS and k NN to the target sparse point clouds and obtain patches as model inputs. Based on Algorithm 2, we use the CDF and total number of inference steps to calculate the adaptive time steps for ODE sampling. In the last step, it is optional to apply manifold optimization for further improvement. We define the updating factor $\alpha = 0.01$ based on the empirical results. Finally, we assemble all the upsampled patches to form the final dense point clouds.

4.2 Comparison to the State of the Art

Here, we evaluate PUFM++ on two datasets, PUGAN and PU1K, and compare it to the state-of-the-art approaches PUBP [23], PUGCN [33], RepKPU [38], PUGAN [18], PUDM [37], Grad-PU [12], PUFM [24], SAPCU [55] and PDANS [53]. Note that the authors of PDANS do not release their pretrained models, so we reimplemented their model based on the official code and reported the results as PDANS[†]. Table 1 reports the results (including 4-times and 16-times upsampling) on the point cloud-based metrics CD, HD, P2F, and JSD. We can see that our approach achieves notable margins on all metrics, especially on P2F. Specifically, from CD and HD scores, we can see that our method outperforms PUFM by +0.07 and +0.13 on 4-times upsampling, and +0.05 and +0.12 on 16-times upsampling. These margins are even larger than those of other methods. For the P2F score, the most competitive method is Grad-PU. However, our method still achieves the best results, with improvements of 0.3–0.7 in two upsampling scenarios. We also add the JSD score as a distribution similarity measure, and we find that ours achieves the lowest value among all approaches. Specifically when we compare it with PUFM, PUFM++ reduces the JSD score by 20–45%.

Figure 3 shows the visual comparison of 4-times point cloud upsampling. To highlight the nuanced differences, we first calculate the point-to-point distance between the true data and the model prediction, then

Table 1: Point cloud upsampling comparison with state-of-the-art methods. We report the numbers of the Chamfer distance (CD, scaled by 10^4), the Hausdorff distance (HD, scaled by 10^3), point-to-surface (P2F, scaled by 10^3), and the Jensen-Shannon divergence (JSD without scaling) on the PUGAN and PU1K datasets. Red indicates the best results, and blue indicates the second-best results. Note that the comparisons are conducted on arbitrarily downsampled sparse point clouds; therefore, the results of different methods are not directly comparable to those reported in the literature.

Dataset	upsampling factor	Metrics	PUBP	PUGCN	RepKPU	PUGAN	PUDM	Grad-PU	PUFM	SAPCU	PDANS [†]	Ours
PUGAN	4	CD	1.649	2.774	1.067	1.541	1.221	1.132	1.049	1.522	1.201	0.980
		HD	1.476	3.831	1.139	1.391	1.174	1.186	0.876	1.471	1.165	0.747
		P2F	5.997	9.508	1.974	5.420	3.132	1.957	1.864	3.441	2.897	1.301
		JSD	0.208	0.205	0.196	0.196	0.147	0.111	0.121	0.135	0.148	0.098
	16	CD	0.982	1.102	0.384	0.869	0.533	0.415	0.353	0.717	0.498	0.286
		HD	2.071	1.785	1.245	1.746	1.185	1.142	0.844	5.371	1.175	0.687
		P2F	7.496	7.125	2.151	6.757	3.589	2.185	2.103	3.256	3.226	1.412
		JSD	0.149	0.250	0.077	0.143	0.108	0.056	0.074	0.070	0.089	0.040
PU1K	4	CD	0.694	1.241	0.566	0.682	0.706	0.626	0.545	0.605	0.669	0.491
		HD	0.593	1.504	0.619	0.632	0.605	0.583	0.556	0.587	0.600	0.432
		P2F	2.206	5.115	1.464	2.792	2.891	1.510	1.770	1.655	2.884	1.243
		JSD	0.286	0.359	0.265	0.315	0.351	0.262	0.304	0.361	0.352	0.256
	16	CD	0.343	2.393	0.290	0.361	0.421	0.316	0.220	0.379	0.415	0.176
		HD	0.712	4.214	0.592	0.773	0.602	0.552	0.578	2.006	0.589	0.429
		P2F	2.958	9.410	1.821	3.538	3.370	1.890	1.983	2.400	3.320	1.375
		JSD	0.282	0.362	0.229	0.225	0.287	0.211	0.228	0.262	0.285	0.135

Table 2: Mesh reconstruction comparison with state-of-the-art methods. We show the area-length ratio (ALR), the manifoldness rate (MR), and normal consistency (NC) on the PU1K datasets. Red indicates the best results, and blue indicates the second-best results.

Dataset	upsampling factor	Metrics	PUBP	PUGCN	RepKPU	PUGAN	PUDM	Grad-PU	PUFM	SAPCU	PDANS [†]	Ours
PU1K	4	ALR	0.231	0.202	0.251	0.237	0.239	0.229	0.243	0.243	0.240	0.250
		MR	0.990	0.989	0.990	0.991	0.989	0.990	0.991	0.991	0.988	0.992
		NC	0.946	0.932	0.949	0.944	0.930	0.939	0.949	0.941	0.931	0.957
	16	ALR	0.256	0.216	0.240	0.246	0.242	0.235	0.237	0.246	0.243	0.267
		MR	0.991	0.989	0.991	0.991	0.988	0.989	0.991	0.991	0.988	0.993
		NC	0.922	0.846	0.932	0.909	0.886	0.896	0.921	0.929	0.891	0.950

normalize the distances and encode them into different colors. We can conclude that 1) from the point distribution, our method has smooth and uniform distributions even when the input point clouds contain hole-like uneven distributions; cf., e.g., the motorcycle; 2) from the general color distribution on the point clouds, see, e.g., the chair in the first row, our approach achieves the lowest errors out of all approaches; 3) compared to PUFM, our approach can preserve sharp 3D shapes, cf., e.g., the plane, and remove outliers, as for the motorcycle; 4) SAPCU tends to oversmooth the regions without considering the sharp curvatures, see, e.g., the lamp, and PUDM preserves the uneven distributions which

causes the unsMOOTH surface, as for the chair and the motorcycle.

Besides, we are also interested in the mesh reconstruction from the upsampled point clouds. To generate the meshes, we first compute the normals from the point cloud, then generate the meshes using the Marching Cube method. We report the mesh comparison in Table 2 across 4-times and 16-times upsampling. For normal estimation (NC), our approach outperforms all other methods by significant margins, indicating its surface smoothness. For ALR, PUFM++ is the second-best one behind the RepKPU approach. For MR, PUFM++ achieves performance equal to or better than others. Figure 4 shows the visual comparison

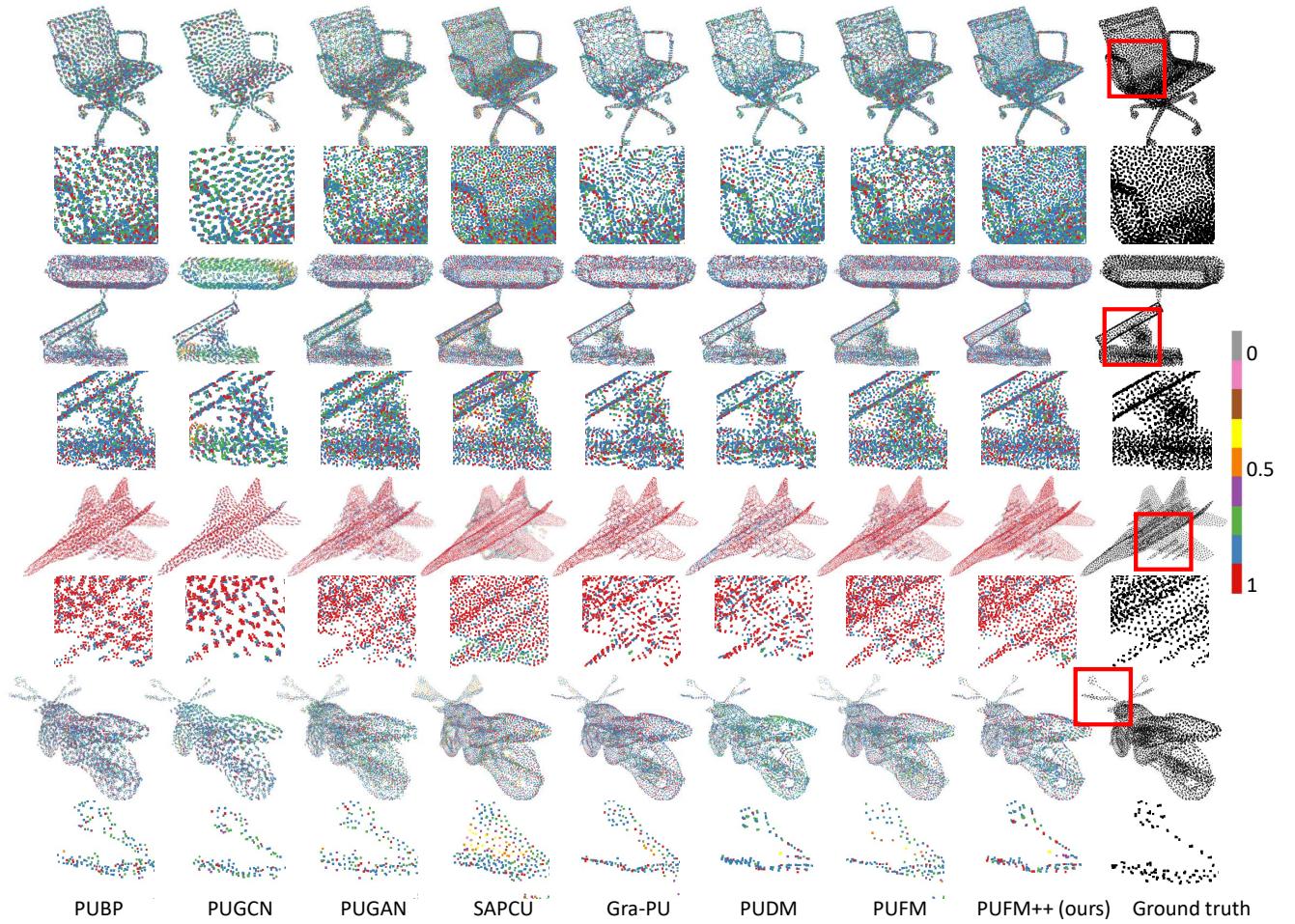


Fig. 3: Visual comparison of different methods on 4-times upsampling. We apply different methods to four examples from the PU1K dataset. Our method generates evenly distributed points and sharp edges, while others exhibit obvious holes and noisy surfaces (see the enlarged region in red boxes).

on mesh reconstruction. We can see that ours can produce meshes with smoother surfaces and fewer holes. On the contrary, other methods produce noisy, incomplete meshes (see the regions marked by the red boxes). For example, our method can estimate the fine details of the horse (first row) and bird (second row) around the legs. It can also produce smoother surfaces for the camera (third row), house (fourth row), and the plate (fifth row).

4.3 Ablation of PUFM++

In this section, we provide ablations of PUFM++. Specifically, we ablate the training schemes 4.3.1 and inference schemes 4.3.2.

4.3.1 Training schemes

For the training stage, we propose two significant improvements: 1) two-stage flow matching optimization with pre-alignment, and 2) replacing the PointNet++ with the Recurrent Interface Network (RIN).

Two-stage flow matching optimization. As introduced in Section 3, one of our key claims is the two-stage training approach. In stage 1, we perform pre-alignment to improve linear interpolation between sparse and dense point clouds. The pre-alignment is an approximation of the EMD, applied only during training with very little computational overhead. In the inference, we can skip pre-alignment and directly refine sparse point clouds to obtain dense output. In the second stage, we have the endpoint flow-matching refinement, which further improves upsampling quality. To make a comparison, we report the ablation results in Table 3.

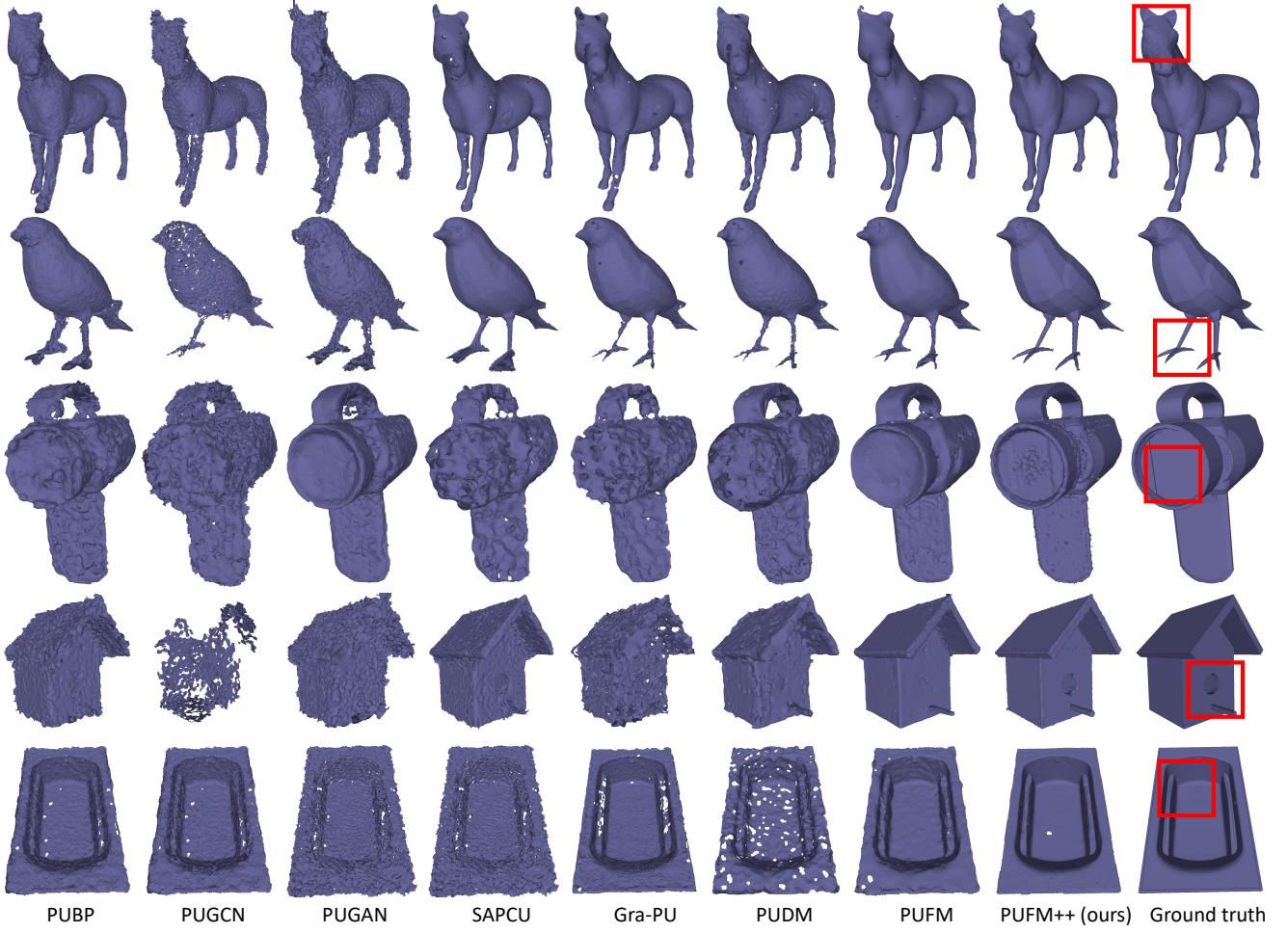


Fig. 4: Visual comparison of different methods on 16-times upsampling for mesh reconstruction. We apply different methods to five examples from the PUGAN and PU1K dataset. Then we use the Marching Cubes algorithm to reconstruct meshes from the upsampled point clouds for qualitative comparison.

Table 3: Ablation on training strategies. CD, HD, and P2F scores on PUGAN and PU1K datasets.

Training strategy	PUGAN			PU1K		
	CD	HD	P2F	CD	HD	P2F
NoAlign	4.989	4.335	2.120	3.001	4.189	8.44
✓	1.135	1.200	1.895	0.648	0.756	1.766
✓	1.010	0.815	1.742	0.523	0.522	1.658
✓	1.012	0.801	1.754	0.526	0.516	1.716
✓ ✓	0.098	0.747	1.301	0.491	0.432	1.243

In Table 3, we compare the full model with or without the pre-alignment and stage two of refinement. For the pre-alignment, we compare the approximated EMD (EMD*) with CD for alignment. We can see that both of them achieve significant improvements over the approach without alignment. Furthermore, using the approximated EMD yields improvements of 0.1–0.4 in CD

Table 4: Ablation on network selection. CD, HD, and P2F scores on the PUGAN and PU1K datasets.

Network	PUGAN						PU1K		Params	Running time (s)
	CD	HD	P2F	CD	HD	P2F	CD	HD	P2F	
PointNet++ (single-stage)	1.049	0.876	1.864	0.545	0.556	1.770				30.36
PointNet++ (two-stage)	1.007	0.826	1.582	0.524	0.512	1.705				
Rin (single-stage)	1.012	0.801	1.754	0.526	0.516	1.716				115.26
RIN (two-stage)	0.098	0.747	1.301	0.491	0.432	1.243				1.231/6

and HD scores. Comparing row 3 and row 5, we can also see that using the additional endpoint flow-matching refinement reduces CD loss by 0.06 to 0.12 across two datasets. Visually, we show three examples on 4-times upsampling to compare the final PUFM++ model with or without these key modules. From Figure 5, we can see

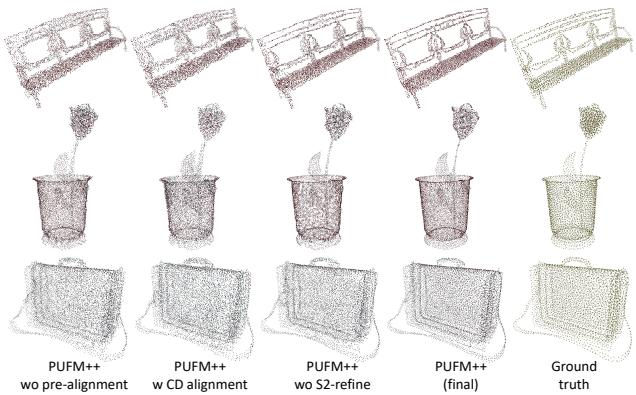


Fig. 5: Visual comparison of training with/without pre-alignment and refinement. The final PUFM++ (4th column) is compared to: PUFM++ without pre-alignment/refinement (1st), with CD pre-alignment only (2nd), and with EMD pre-alignment only (3rd).

that the model trained without pre-alignment generates noisy and blurry 3D surfaces around the bench (first row) and produces unstable edges (e.g., the flower and the bag in rows 2 and 3, respectively). Incorporating CD pre-alignment slightly reduces noise but does not fully resolve the artifacts. In contrast, comparing the third and fourth columns shows that adding the endpoint refinement stage significantly improves the point distribution along edges, as illustrated by the clearer bench patterns in the first example and the sharper vase contours in the second.

Recurrent Interface Network for PUFM++. We propose to replace the PointNet++ with the RIN network in Section 3.3. The main advantage is that it avoids the U-shaped feature extraction and aggregation, and refines the latent representation iteratively before projecting back onto the point space. Meanwhile, the latent self-conditioning stabilizes iterative upsampling, thereby avoiding artifacts such as point clustering or uneven density. In Table 4, we compare upsampling quality between PointNet++ and RIN. We can conclude that 1) the RIN uses 3 times more parameters but achieves 90% improvements on both PUGAN and PU1K datasets. The running time only increases to a 50% computation overhead (0.14s to 0.21s); 2) both PointNet++ and RIN can boost the performance using two-stage optimization, but using RIN with single-stage optimization achieves similar performance as the PointNet++ with two-stage optimization.

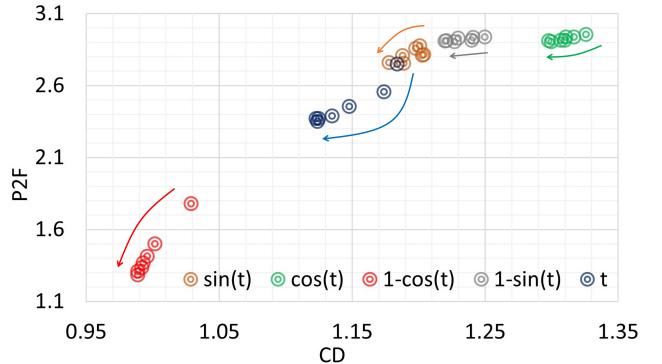


Fig. 6: Visual comparison of ODE sampling steps and time scheduler. We apply different time schedulers to train the proposed model, and then use different discretization steps of the ODE for inference. We plot the 4-times upsampling results on the PUGAN dataset, where the horizontal axis is the CD score and the vertical axis is the P2F distance.

4.3.2 Inference schemes

For the inference stage, we propose two novel approaches to boost the upsampling quality: 1) ODE inference steps, 2) the adaptive entropic time scheduler, and 3) the manifold constraint.

Inference steps. The number of inference steps and the time scheduler for linear interpolation used in training affect the point cloud upsampling. In our algorithm (Algorithm 1), we choose $1 - \cos(s\pi/2)$ as the time scheduler because we want to encourage the model to learn the target velocity from more challenging points that are closer to the sparse point clouds. During inference, the proposed PUFM++ is similar to other score-based generative models: the more iterations we use, the better the results are. From Figure 6, we can see that using $1 - \cos(s\pi/2)$ (red circles) lies on the left bottom corner, where it has both a low CD score and a P2F distance. On the other hand, we can see that t (blue), $\sin(s\pi/2)$ (orange), $1 - \sin(s\pi/2)$ (green) are all suboptimal. To balance upsampling quality and runtime, we apply 6-step inference.

Adaptive entropic time scheduler. As described in Section 3, we freeze the pre-trained model and test it on the training dataset, then record the average losses on the training sampling grid $\{t_0, t_1, t_2, \dots\}$. It is defined as a uniform time sequence sampled from $[0, 1]$ with a total of 50 steps. Then we compute the CDF of the recorded losses $\hat{F}(t)$. During the inference, we compute the time step as in 13 for ODE sampling. This approach leverages information from the training process to adjust step sizes, thereby improving inference quality dynamically. In Figure 8, we show results

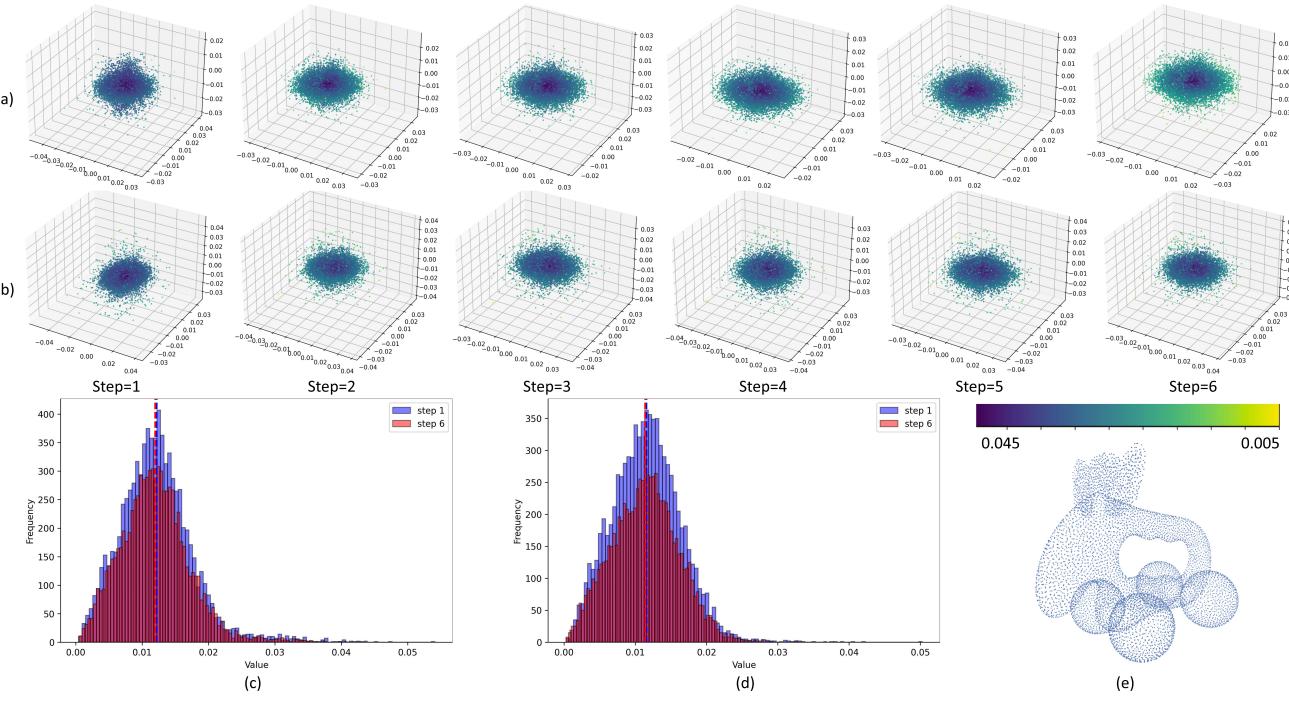


Fig. 7: Visualization of entropic time scheduler for inference. We show the residual scatter map on 4-times upsampling on the point cloud (e) by calculating the coordinate differences using (a) an entropic time scheduler and (b) uniform time scheduler. The darker the color, the bigger the errors. We also show the residual histograms for the two inference schemes in (c) and (d).

for PUFM, PUFM++ with a uniform time scheduler (PUFM++_uts) and the proposed PUFM++ with an entropic time scheduler (PUFM++_ets). We can see that 1) PUFM, as the baseline model, achieves the worst results at every inference step; 2) compared to PUFM++_uts, the proposed PUFM++_ets allocates larger step sizes in the early ODE iterations and progressively refines them in later stages. This behavior enables the model to focus more on complex refinements near the end of the trajectory. Quantitatively, on 4-times point cloud upsampling with the PUGAN dataset, our scheduler consistently improves both CD and P2F metrics over uniform time stepping, demonstrating its effectiveness.

We further provide a qualitative example of 4-times upsampling in Figure 7. In subfigures (a) and (b), we visualize the coordinate residuals between the upsampled point clouds and the ground-truth target. The residuals are normalized and mapped onto a 3D scatter plot, color-coded by their Euclidean distances. Darker color indicates bigger errors, and tigher clusters indicate smaller errors. We can see from subfigures (a) and (b) that using the entropic time scheduler provides overall smaller errors across all inference steps. Subfigures (c) and (d) show the corresponding histograms of the

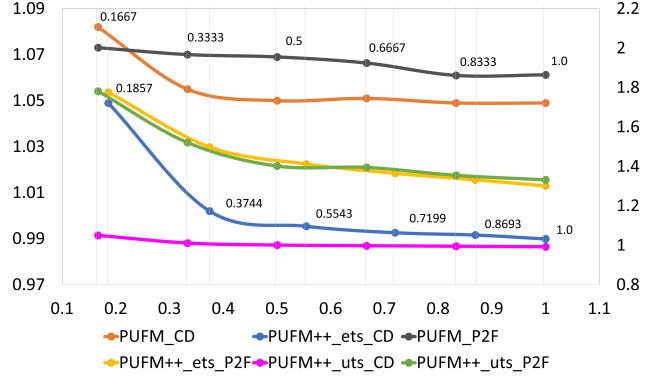


Fig. 8: Compare different time schedulers for inference. We show the per-step inference using the uniform time scheduler and the proposed adaptive time scheduler on PUFM and PUFM++.

residuals. Results in (a) and (c) are obtained with the uniform time scheduler, while (b) and (d) correspond to our proposed entropic time scheduler. Compared with uniform scheduling, the entropic scheduler shifts the mean residuals closer to zero. It significantly compresses the overall error distribution (see the reduced maximum

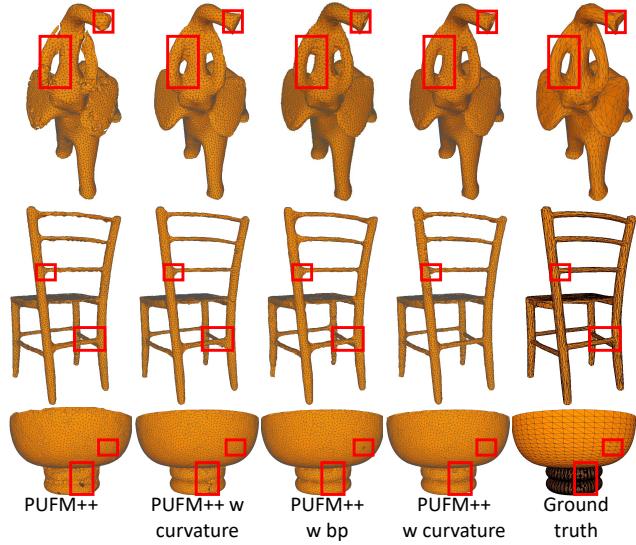


Fig. 9: Visual comparison of manifold constraints for mesh reconstruction. We apply manifold constraints to the PUFM++ inference stage (4-times upsampling) and compare the upsampling quality based on the mesh reconstruction.

Table 5: Ablation on network selection. CD, HD, and P2F scores on PUGAN and PU1K datasets.

Curvature Est.	Back Proj.	PUGAN				PU1K			
		CD	HD	ALR	NC	CD	HD	ALR	NC
✓		0.286	0.687	0.250	0.957	0.176	0.429	0.267	0.950
		0.291	0.648	0.242	0.988	0.181	0.424	0.246	0.986
	✓	0.284	0.651	0.280	0.989	0.176	0.423	0.289	0.988
✓	✓	0.281	0.650	0.282	0.990	0.175	0.422	0.290	0.990

values in (b) vs. (a)), indicating more stable and accurate reconstructions.

Manifold constraints for mesh reconstruction. In Section 3, we introduce the curvature estimation and back projection-based post-processing to adjust the upsampled point clouds. The goal is to use these two technologies to explore a point distribution that better approximates the underlying 3D manifold. They can be useful for downstream applications like mesh reconstruction. Hence, we compare the proposed PUFM++ to its curvature-estimation and back-projection variants and present the results in Figure 9.

We reconstructed meshes from the upsampled point clouds using the Ball Pivoting algorithm in MeshLab and visualized the differences. As shown, using PUFM++ alone does not yield satisfactory mesh reconstruction results. In contrast, incorporating back projection or curvature estimation significantly improves mesh quality. For example, in the first row,

the nose and teeth of the elephant are much more clearly restored compared to the baseline PUFM++ output. In the highlighted red box regions of the chair and bowl, both curvature estimation and back projection enhance the sharp edges, although some noise is introduced. The combination of curvature estimation and back projection achieves the best balance between smoothness and sharpness in mesh reconstruction.

Meanwhile, the quantitative comparisons on the PUGAN and PU1K datasets are presented in Table 5. The results demonstrate that incorporating curvature estimation and back projection not only improves point cloud upsampling performance (as reflected by lower CD and HD scores) but also enhances mesh reconstruction quality, evident by higher ALR and NC scores.

5 Analysis of PUFM++

In this section, we provide an extended analysis on the robustness and generalization of the proposed method.

Multiscale point cloud upsampling. The proposed PUFM++ can also be used for multiscale point cloud upsampling. As illustrated in other methods [33, 37, 18, 49], we can iteratively apply the model to the sparse point clouds for 4× upsampling, followed by FPS to downsample the points to the desired size. This iterative procedure enables arbitrary-scale upsampling. The visualization of multiscale upsampling is shown in Figure 10. We compare our approach with RepKPU, PUDM and PUFM at factors 5, 7, 8, 12, 16, 20, 24 and 32. As shown in the first example, PUFM++ consistently reconstructs objects with smoother surfaces and fewer holes. In the second example, our method accurately preserves the thin and straight structures of the legs of the device without introducing noise.

Noisy point cloud upsampling. We consider that real-world point clouds suffer from various noise. To test the robustness of our method, we quantitatively compare it with others on noisy point cloud upsampling. We randomly add Gaussian noise to the sparse point clouds with different levels η , then we use different methods to upsample noisy sparse point clouds to simulate real-world point clouds. We can see from Table 6 that our method achieves the lowest CD, HD, and P2F scores among all methods, with the only exception of HD at noise level $\eta = 0.01$ for PUDM. Figure 11 further illustrates that our upsampled point clouds are visually much smoother than those produced by other methods, effectively mitigating the noise perturbations.

Real-world point cloud upsampling. To demonstrate the generalization of the proposed method, we

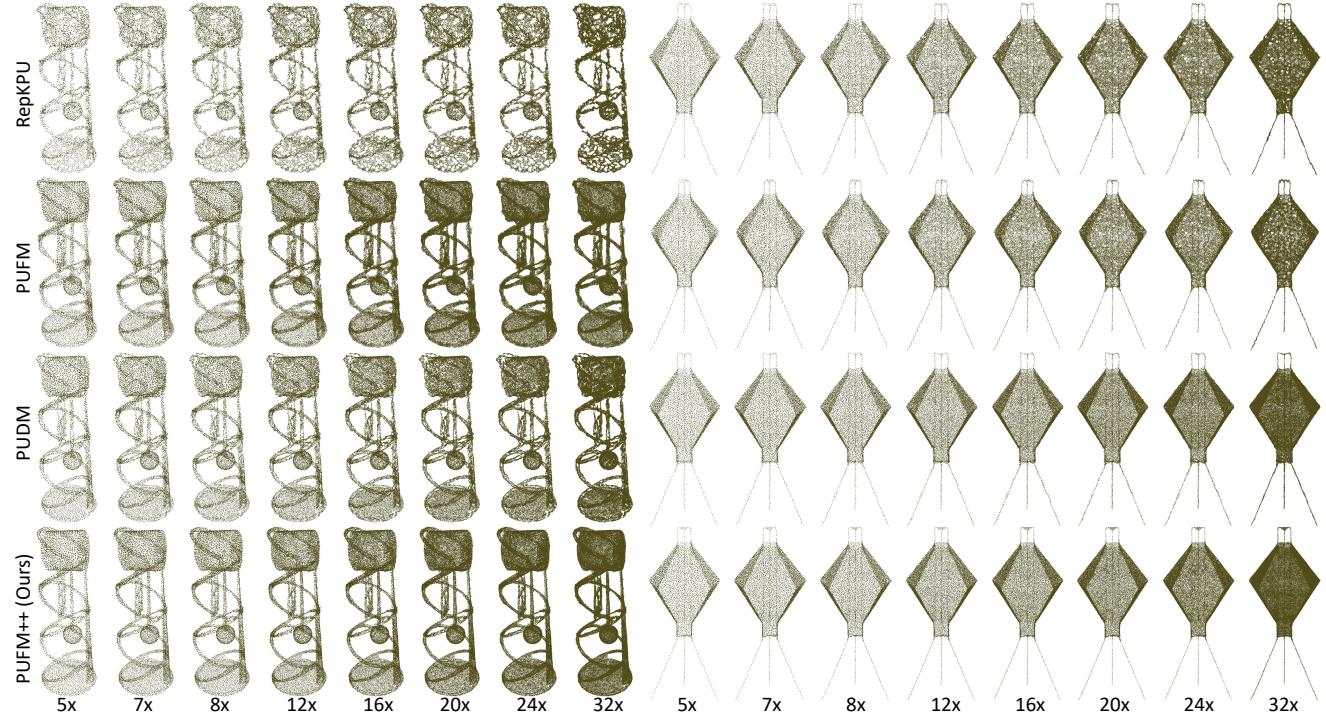


Fig. 10: Visual comparison of multiscale point cloud upsampling. We apply multiscale upsampling on two examples and compare the upsampling quality using different methods.

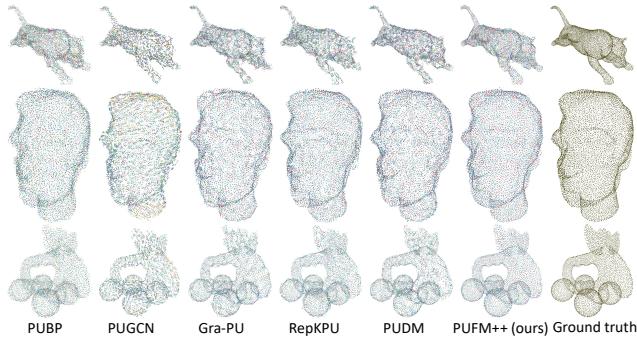


Fig. 11: Visual comparison of noisy point cloud upsampling. We apply point cloud upsampling on PU-GAN datasets with noise level $\eta = 0.01$ and show three examples for comparison.

apply it to real-world point clouds, including ABC [16] (computer-aided design (CAD) dataset), ScanNet [5] (RGB-D video dataset) and KITTI [8] (LiDAR driving dataset). We consider both upsampling quality and performance on downstream tasks, including object detection and segmentation.

For real-world point cloud upsampling, we directly apply pretrained models to ScanNet and KITTI. For ScanNet, we select 20 samples and randomly downsample each by 4-times to obtain sparse point clouds.

Noise level	$\eta = 0.01$			$\eta = 0.02$		
	CD	HD	P2F	CD	HD	P2F
PUDM	0.983	1.069	5.188	2.175	2.737	1.146
RepKPU	1.232	1.212	6.654	2.888	3.638	1.336
Grad-PU	1.125	1.258	6.214	2.475	3.375	1.247
PUFM	0.890	1.081	5.166	1.761	2.287	1.081
Ours	0.706	1.082	5.142	1.662	2.231	1.041

Table 6: 4-times point cloud upsampling on noisy PU1K. We test on 4-times upsampling and we can see that ours outperforms others, especially on noise level $\eta = 0.02$.

bbox AP@0.7,0.7,0.7	Sparse PC	Gra-PU	PUDM	PDANS [†]	PUGAN	PUFM	PUFM++	Ground truth
	Car	20.35	20.50	20.18	20.25	16.78		
Pedestrian	17.05	16.67	15.79	16.04	14.68	13.51	16.88	18.18

Table 7: Quantitative metric of the LiDAR data detection. We test on 2-times upsampling using different methods, and use the upsampled LiDAR data to estimate the object detection performance.

We then apply different upsampling methods to recover dense point clouds for comparison.

For KITTI, we simulate 32-beam LiDAR scans from the original 64-beam point clouds using angular sampling. Each sparse point cloud is then upsampled by 4-

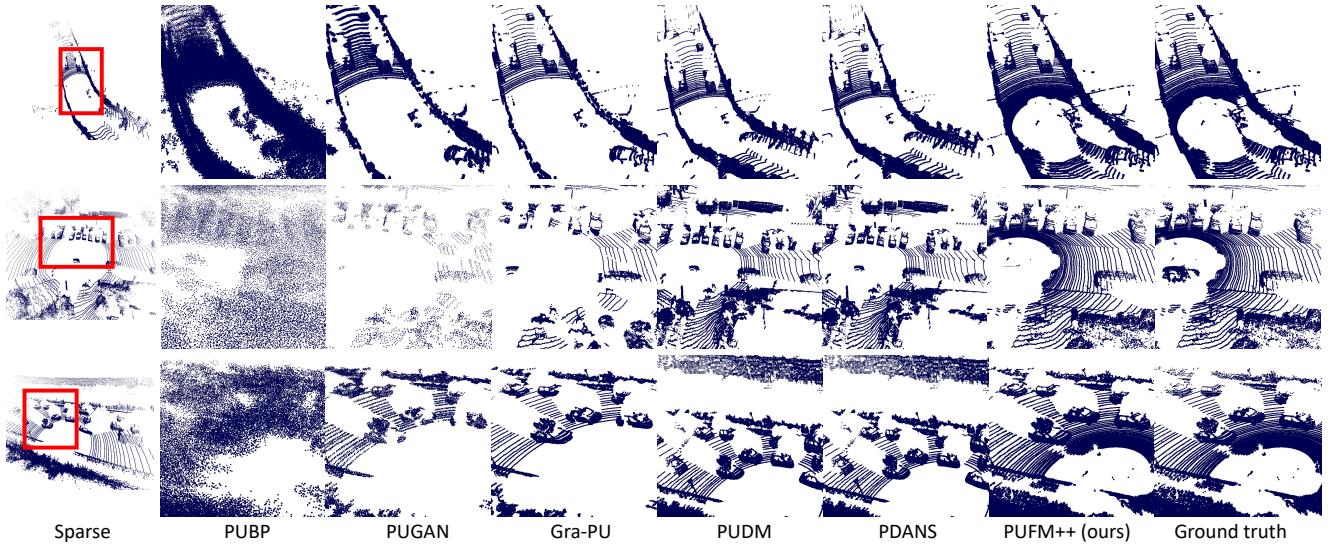


Fig. 12: Visual comparison of KITTI LiDAR point cloud upsampling. We apply different upsampling methods to sparse point clouds for 2 \times upsampling and zoom in the red box for visual comparison.

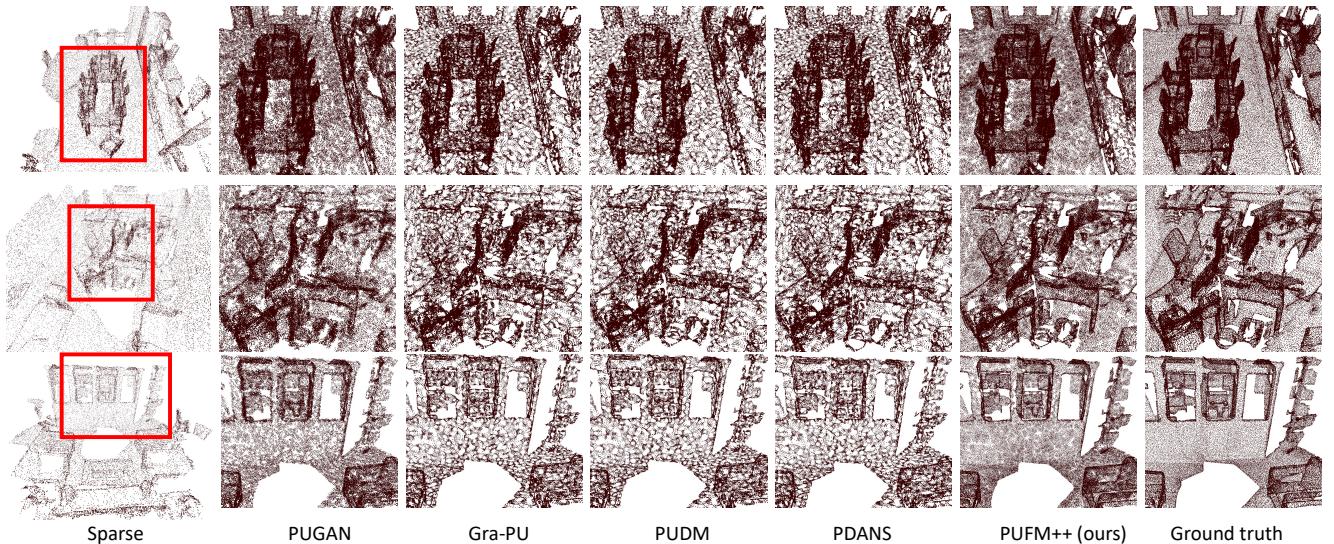


Fig. 13: Visual comparison of ScanNet RGB-D point cloud upsampling. We apply different upsampling methods to sparse point clouds for 4 \times upsampling and zoom in the red box for visual comparison.

Metric	PUBP	PUGAN	Gra-PU	PUDM	PDANS [†]	PUFM	PUFM++
CD	1.514	1.516	1.039	1.038	1.041	1.020	0.899
HD	0.385	0.367	0.248	0.258	0.256	0.223	0.156

Table 8: Quantitative metric of the ScanNet point cloud upsampling. We randomly downsample 4-times \times upsampling task.

times, followed by a 2-times downsampling using FPS. The resulting point clouds are fed into SECOND [47] for 3D object detection. We report the average precision

of the bird’s-eye view bounding boxes at the easy difficulty level in Table 7. Our method achieves the highest AP among all competing methods. Remarkably, it even outperforms the original LiDAR data on the car class. Figure 12 provides qualitative comparisons on KITTI, with red boxes highlighting regions of interest. Our method preserves beam-like LiDAR patterns similar to the original data. In contrast, PUBP produces noticeable blurring across the scene, while PUGAN and Gra-PU fail to generate consistent beam structures. PUDM and PDANS yield comparable results but introduce misalignment in fine details.

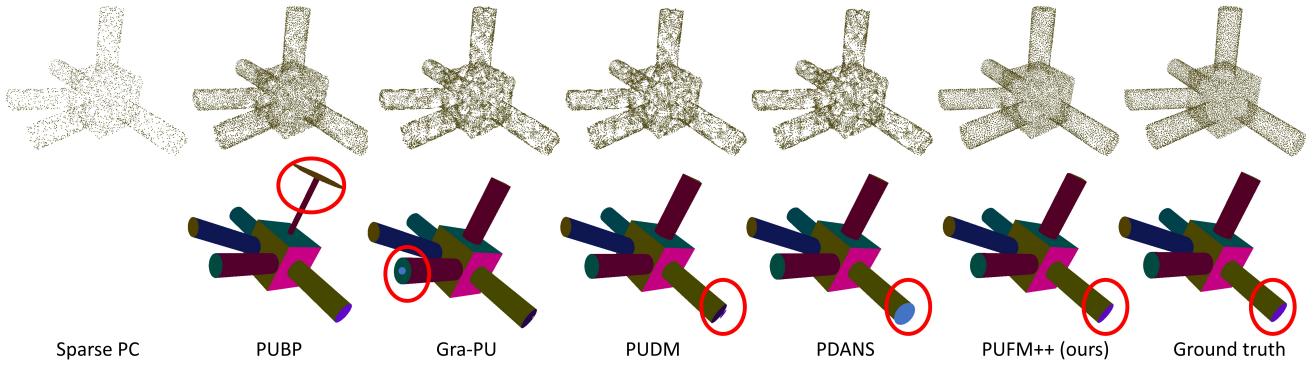


Fig. 14: Visual comparison of converting point clouds to CAD models. We apply different upsampling methods to sparse point clouds for 4 \times upsampling and then use Point2CAD to convert them to CAD models. Different colors denote different topological surfaces.

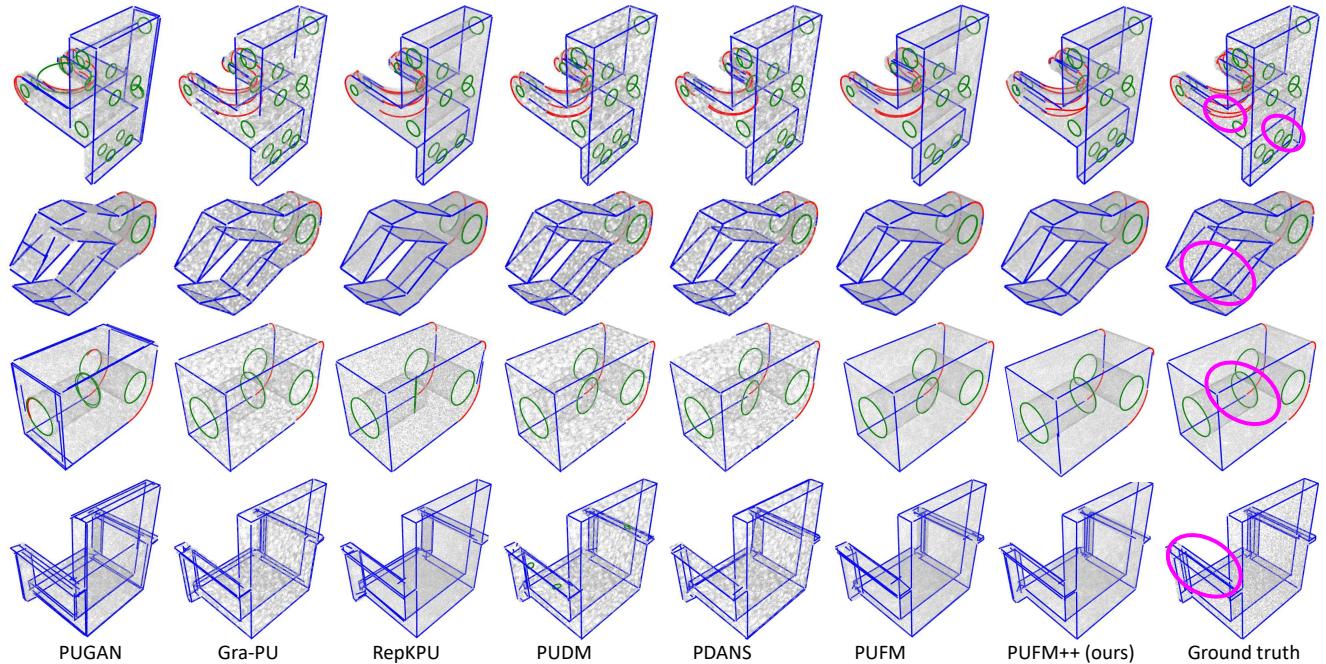


Fig. 15: Visual comparison of point clouds curve estimation. We apply different upsampling methods to sparse point clouds for 4-times upsampling and then use PI3DETR to estimate the 3D curves from point clouds. Different colors denote different edges: red for arcs, blue for lines, green for circles.

For the ScanNet dataset, we randomly sample 10 scenes and downsample each point cloud by a factor of 4-times to generate sparse inputs. We then apply various upsampling methods to recover the dense point clouds with the same 4-times upsampling ratio. To quantitatively assess the reconstruction quality, we report the CD and HD scores in Table 8. We can see that our method achieves the best CD and HD scores. We also provide qualitative comparisons in Figure 13. As highlighted in the zoom-in regions (red boxes), our method more accurately restores fine-grained 3D surfaces, producing smooth and uniformly distributed

points, like the floor in all three examples, and the cupboard in the third example. In contrast, competing approaches often struggle to reconstruct the underlying geometry and tend to generate point clouds with noticeable holes or irregularities.

To validate the robustness of our point cloud upsampling approach on industrial manufacturing, we apply our model to CAD object recognition, including 1) point to CAD conversion, and 2) 3D curve estimation.

For task 1), we randomly select 10 point clouds from the ABC datasets with their corresponding label information. We randomly downsample the original point

Point cloud to CAD								
Metric	PUBP	Gra-PU	PUDM	PDANS	RepKPU	PUFM	PUFM++	
Res-err↓	0.022	0.006	0.008	0.008	0.010	0.006	0.005	0.004
P-cover↑	0.934	0.993	0.989	0.990	0.988	0.994	0.995	0.997
3D curve estimation								
Metric	PUGAN	Gra-PU	RepKPU	PUDM	PDANS	PUFM	PUFM++	
CD↓	0.0189	0.0040	0.0041	0.0049	0.0045	0.0042	0.0038	0.0032
HD↓	0.156	0.0674	0.0675	0.0678	0.0676	0.0674	0.0670	0.0665

Table 9: Quantitative metric of the point cloud for CAD analysis. We test on 4× upsampling on two point clouds to CAD applications: 1) point cloud to CAD conversion and 2) 3D curve estimation.

clouds by 4-times and then upsample them by using different upsampling methods. Finally, we use kNN tree search to align the upsampled points with the original labels. Next, we use Point2CAD [22], a recent work on reconstructing CAD models from raw point clouds, to convert upsampled point clouds to CAD models. We visualize one example in Figure 14. The first row includes the upsampled point clouds, and the second row the reconstructed CAD models, with different colors denoting different topological surfaces. From the marked red circles, we can see that our proposed method produces the closest CAD model to the ground truth.

For task 2), we use the same set of 10 point clouds from the ABC dataset to produce sparse and original dense point clouds. Our goal is to estimate the 3D curves of the point clouds to see if the upsampling method can preserve the 3D curvatures. We apply the recent parametric instance detection of 3D point cloud edges, PI3DETR [30], to the point clouds and visualize the estimated curves. In Figure 15, we show the edge estimation with different colors. To highlight the differences, we circle them in pink. We can see that 1) our method correctly produces the green circles in the first and third example, as well as the straight lines in the second and fourth examples; 2) Visually, PUGAN and Grad-PU produce more extra or incomplete edges; 3) PUDM produces extra circles in the fourth example; 4) PDANS and PUFM are closer to our approach, but they also fail at fine details.

For both tasks (1) and (2), we quantitatively compare different upsampling methods using several evaluation metrics. For task (1), we adopt Res-Err to measure the discrepancy between the reconstructed surface and the ground-truth surface, where the correspondence is established using Hungarian matching. We additionally report P-Cover, which quantifies the proportion of the input point cloud that is covered by the generated surface (see Point2CAD [22] for metric def-

initions). Our method achieves the second-best performance, only slightly lower than that obtained using the ground-truth point clouds. For task (2), we evaluate reconstruction quality in edge regions using Chamfer Distance (CD) and Hausdorff Distance (HD), where lower values indicate better geometric consistency. Except for the ground truth data, our approach achieves the best performance among all competing upsampling methods.

6 Ethical Discussion

Practical Impact. PUFM++ provides a unified and highly efficient framework for point cloud upsampling with applications in 3D vision, robotics, AR/VR, reverse engineering, and digital twins. First, PUFM++ can help practitioners reconstruct high-fidelity 3D geometry from sparse, noisy, or incomplete scans, enabling more accessible digitization of cultural heritage, industrial assets, and environmental data. Second, the method can support robotics and autonomous systems by improving scene understanding, particularly for manipulation and navigation in cluttered or partially observed environments. Finally, PUFM++ may facilitate content creation for film, gaming, and 3D design tools, reducing manual labor for artists and engineers.

However, the ability to enhance sparse 3D data can introduce risks. PUFM++ could be misused to artificially densify or restore 3D scans in ways that obscure provenance or authenticity. This may enable sophisticated forms of 3D spoofing, for example, producing deceptive digital replicas of individuals, objects, or scenes. As with any generative AI system, responsible use requires transparency about whether 3D content has been algorithmically modified or reconstructed.

Societal Impact. PUFM++ is trained on benchmark datasets that primarily contain synthetic or curated 3D models, which do not comprehensively represent global object distributions, environmental diversity, or scanning conditions. As a result, the model may generalize unevenly to real-world scans that contain domain shifts such as culturally specific artifacts, region-specific object designs, or sensor-specific distortions. Moreover, because upsampling methods implicitly learn priors over shape structures, they may unintentionally impose biases, e.g., over-smoothing culturally distinctive geometry or failing on underrepresented categories. Such biases can propagate into downstream tasks like mesh reconstruction, robotics perception, or 3D mapping, potentially affecting reliability in safety-critical systems.

PUFM++ does not explicitly model personal identity or biometric information, but high-fidelity reconstruction of human-related geometry (e.g., faces, bodies) could raise privacy considerations if used on sensitive scans without consent. Users should carefully evaluate the appropriateness of applying generative reconstruction models in domains involving personal or proprietary 3D data.

Environmental Impact. Our experiments rely on 2 NVIDIA V100 GPUs, each with an approximate power draw of 300W (V100) during training. Across all experiments, we required roughly 800 GPU hours. Training the full PUFM++ model requires around 10 GPU hours on a V100 GPU, corresponding to approximately 3 kWh of electricity and an estimated 1.2kg of CO₂ emissions, depending on local energy sources. While the training cost is modest compared to large-scale generative models, we encourage the community to adopt energy-efficient practices such as mixed-precision training, model reuse through fine-tuning, and sharing pretrained models to reduce redundant computations.

7 Conclusions

We introduce PUFM++, a versatile point cloud upsampling framework built upon a novel two-stage flow-matching strategy, combined with inference-time optimization, to achieve state-of-the-art performance for arbitrary upsampling factors. Our results show that the proposed two-stage flow-matching process significantly enhances geometric restoration and provides strong robustness against noise. The inference-time optimization further improves alignment with the underlying 3D manifold. To the best of our knowledge, we are also the first to adopt a recurrent inference network for point cloud processing, which consistently outperforms PointNet++ across all evaluation metrics. Extensive analyses and visualizations validate our findings and demonstrate the superior performance of PUFM++. We also compare our model with existing approaches across multiple downstream tasks and various real-world datasets, including KITTI, ScanNet, and ABC, highlighting the robustness and efficiency of our method. Future work includes one-step flow-matching distillation and joint upsampling-completion models, aiming to further improve efficiency and generalization.

Funding information. Z. S. Liu has been supported by the Research Council of Finland’s decision number

359633. Chenhang He acknowledges support from the National Natural Science Foundation of China (No.62406268). R. Maier acknowledges support from the German Academic Exchange Service (DAAD), project ID 57711336.

Conflict of interest statement. The authors declare that they have no conflict of interest.

Data availability. Code and pretrained models are publicly available at https://github.com/Holmes-Alan/Enhanced_PUFM

References

1. Akhtar, A., Li, Z., Auwera, G.V.d., Li, L., Chen, J.: Pu-dense: Sparse tensor-based point cloud geometry upsampling. *IEEE Transactions on Image Processing* **31**, 4133–4148 (2022) [1](#), [3](#)
2. Albergo, M.S., Boffi, N.M., Vanden-Eijnden, E.: Stochastic interpolants: A unifying framework for flows and diffusions. *ArXiv Preprint*, 2303.08797 (2023) [4](#)
3. Charles, R., Su, H., Kaichun, M., Guibas, L.J.: Pointnet: Deep learning on point sets for 3d classification and segmentation. In: *IEEE Conf. Comput. Vis. Pattern Recog.* pp. 77–85. Los Alamitos, CA, USA (2017) [2](#)
4. Chen, Y., et al.: Pointmixup: Augmentation for point clouds. In: *Eur. Conf. Comput. Vis.* (2020) [5](#)
5. Dai, A., Chang, A.X., Savva, M., Halber, M., Funkhouser, T., Nießner, M.: Scannet: Richly-annotated 3d reconstructions of indoor scenes. In: *Proc. Computer Vision and Pattern Recognition (CVPR)*, IEEE (2017) [8](#), [15](#)
6. Du, Y., Zhao, Z., Su, S., Golluri, S., Zheng, H., Yao, R., Wang, C.: SuperPC: A single diffusion model for point cloud completion, upsampling, denoising, and colorization. In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2025) [3](#)
7. Feng, W., et al.: Neural points: Point cloud representation with neural fields for arbitrary upsampling. In: *IEEE Conf. Comput. Vis. Pattern Recog.* (2022) [3](#)
8. Geiger, A., et al.: Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)* (2013) [8](#), [15](#)
9. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial networks. *Commun. ACM* **63**(11), 139–144 (2020) [3](#)
10. Guo, M., Wang, B., He, K., Matusik, W.: Tetsphere splatting: Representing high-quality geometry with lagrangian volumetric meshes. In: *The Thirteenth International Conference on Learning Representations* (2025) [8](#)
11. Hanocka, R., et al.: Point2mesh: A self-prior for deformable meshes. *ACM Trans. Graph.* **39**(4) (2020) [1](#)
12. He, Y., et al.: Grad-pu: Arbitrary-scale point cloud upsampling via gradient descent with learned distance functions. In: *IEEE Conf. Comput. Vis. Pattern Recog.* pp. 5354–5363 (2023) [4](#), [8](#)
13. Ho, J., et al.: Denoising diffusion probabilistic models. In: *Adv. Neural Inform. Process. Syst.* (2020) [2](#), [3](#)
14. Jabri, A., Fleet, D.J., Chen, T.: Scalable adaptive computation for iterative generation. In: *Proceedings of the 40th International Conference on Machine Learning* (2023) [7](#), [8](#)

15. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. *Int. Conf. Learn. Represent.* (2014) **8**
16. Koch, S., Matveev, A., Jiang, Z., Williams, F., Artemov, A., Burnaev, E., Alexa, M., Zorin, D., Panozzo, D.: Abc: A big cad model dataset for geometric deep learning. In: *IEEE Conf. Comput. Vis. Pattern Recog.* pp. 9593–9603 (2019) **8**, **15**
17. Lemke, O., et al.: Spot-compose: A framework for open-vocabulary object retrieval and drawer manipulation in point clouds. In: *Int. Conf. on Robotics and Automation* (2024) **1**
18. Li, R., Li, X., Fu, C., Cohen-Or, D., Heng, P.: Pu-gan: A point cloud upsampling adversarial network. *Int. Conf. Comput. Vis.* pp. 7202–7211 (2019) **2**, **3**, **8**, **14**
19. Li, R., et al.: Point cloud upsampling via disentangled refinement. In: *IEEE Conf. Comput. Vis. Pattern Recog.* (2021) **3**
20. Lipman, Y., et al.: Flow matching for generative modeling. In: *The Eleventh International Conference on Learning Representations* (2023) **2**, **3**
21. Liu, X., Liu, X., Liu, Y.S., Han, Z.: Spu-net: Self-supervised point cloud upsampling by coarse-to-fine reconstruction with self-projection optimization. *Trans. Img. Proc.* **31**, 4213–4226 (2022) **3**
22. Liu, Y., Obukhov, A., Wegner, J.D., Schindler, K.: Point2cad: Reverse engineering cad models from 3d point clouds. In: *IEEE Conf. Comput. Vis. Pattern Recog.* pp. 3763–3772 (2024) **18**
23. Liu, Z.S., et al.: Arbitrary point cloud upsampling via dual back-projection network. In: *IEEE Int. Conf. Image Process.* pp. 1470–1474 (2023) **8**
24. Liu, Z.S., He, C., Ju, Y., Li, L.: PUFM: Efficient point cloud upsampling via flow matching. *AAAI* (2026) **2**, **3**, **4**, **8**
25. Liu, Z.S., Siu, W.C., Chan, Y.L.: Photo-realistic image super-resolution via variational autoencoders. *IEEE Transactions on Circuits and Systems for Video Technology* **31**(4), 1351–1365 (2021) **1**
26. Liu, Z.S., Wang, L.W., Li, C.T., Siu, W.C.: Image super-resolution via attention based back projection networks. In: *IEEE International Conference on Computer Vision Workshop(ICCVW)* (2019) **1**
27. Lu, X., et al.: Low rank matrix approximation for 3d geometry filtering. *IEEE Trans. Vis. Comput. Graph.* **28**(4), 1835–1847 (2022) **2**
28. Luo, S., Hu, W.: Diffusion probabilistic models for 3d point cloud generation. In: *IEEE Conf. Comput. Vis. Pattern Recog.* pp. 2836–2844 (2021) **3**
29. Mao, A., Du, Z., Hou, J., Duan, Y., Liu, Y.J., He, Y.: Pu-flow: A point cloud upsampling network with normalizing flows. *IEEE Trans. Vis. Comput. Graph.* **29**(12), 4964–4977 (2023) **1**, **2**, **3**
30. Oberweger, F.F., Schwingshakl, M., Staderini, V.: Pi3detr: Parametric instance detection of 3d point cloud edges with a geometry-aware 3detr. *ArXiv Preprint*, 2509.03262 (2025) **18**
31. Papamakarios, G., Nalisnick, E., Rezende, D.J., Mohamed, S., Lakshminarayanan, B.: Normalizing flows for probabilistic modeling and inference. *J. Mach. Learn. Res.* **22**(1) (2021) **3**
32. Qi, C.R., et al.: Pointnet++: deep hierarchical feature learning on point sets in a metric space. In: *Adv. Neural Inform. Process. Syst.* p. 5105–5114. *NIPS’17* (2017) **2**
33. Qian, G., et al.: Pu-gcn: Point cloud upsampling using graph convolutional networks. In: *IEEE Conf. Comput. Vis. Pattern Recog.* pp. 11683–11692 (2021) **3**, **8**, **14**
34. Qian, Y., Hou, J., Kwong, S., He, Y.: Pugeo-net: A geometry-centric network for 3d point cloud upsampling. In: *Computer Vision – ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XIX*. p. 752–769 (2020) **1**, **3**
35. Qiu, H., Yu, B., Tao, D.: GFNet: Geometric flow network for 3d point cloud semantic segmentation. *Trans. Mac. Lear. Res.* (2022) **3**
36. Qiu, S., Anwar, S., Barnes, N.: Pu-transformer: Point cloud upsampling transformer. In: *ACCV*. pp. 2475–2493 (2022) **3**
37. Qu, W., et al.: A conditional denoising diffusion probabilistic model for point cloud upsampling. In: *IEEE Conf. Comput. Vis. Pattern Recog.* pp. 20786–20795 (2024) **2**, **3**, **4**, **8**, **14**
38. Rong, Y., et al.: Repkpu: Point cloud upsampling with kernel point representation and deformation. In: *IEEE Conf. Comput. Vis. Pattern Recog.* pp. 21050–21060 (2024) **8**
39. Shi, Y., Bortoli, V.D., Campbell, A., Doucet, A.: Diffusion schrödinger bridge matching. In: *Thirty-seventh Conference on Neural Information Processing Systems* (2023) **4**
40. Thomas, H., et al.: Kpconv: Flexible and deformable convolution for point clouds. *Int. Conf. Comput. Vis.* (2019) **2**
41. Tong, A., FATRAS, K., Malkin, N., Huguet, G., Zhang, Y., Rector-Brooks, J., Wolf, G., Bengio, Y.: Improving and generalizing flow-based generative models with mini-batch optimal transport. *Transactions on Machine Learning Research* (2024) **3**
42. Wang, Y., et al.: Dynamic graph cnn for learning on point clouds. *ACM Trans. Graph.* **38**(5) (2019) **2**
43. Wu, L., Wang, D., Gong, C., Liu, X., Xiong, Y., Ranjan, R., Krishnamoorthi, R., Chandra, V., Liu, Q.: Fast point cloud generation with straight flows. In: *IEEE Conf. Comput. Vis. Pattern Recog.* pp. 9445–9454 (2023) **3**
44. Wu, W., Qi, Z., Fuxin, L.: Deep convolutional networks on 3d point clouds. In: *IEEE Conf. Comput. Vis. Pattern Recog.* pp. 9613–9622 (2019) **2**
45. Wu, X., Jiang, L., Wang, P.S., Liu, Z., Liu, X., Qiao, Y., Ouyang, W., He, T., Zhao, H.: Point transformer v3: Simpler, faster, stronger. In: *IEEE Conf. Comput. Vis. Pattern Recog.* (2024) **3**
46. Wu, X., Lao, Y., Jiang, L., Liu, X., Zhao, H.: Point transformer v2: grouped vector attention and partition-based pooling. In: *Adv. Neural Inform. Process. Syst.* (2022) **3**
47. Yan, Y., Mao, Y., Li, B.: Second: Sparsely embedded convolutional detection. *Sensors* **18**(10), 3337 (2018) **16**
48. Yang, Z., et al.: Visual point cloud forecasting enables scalable autonomous driving. In: *IEEE Conf. Comput. Vis. Pattern Recog.* (2024) **1**
49. Yifan, W., et al.: Patch-based progressive 3d point set upsampling. *IEEE Conf. Comput. Vis. Pattern Recog.* pp. 5951–5960 (2019) **2**, **3**, **8**, **14**
50. Yu, L., et al.: Pu-net: Point cloud upsampling network. *IEEE Conf. Comput. Vis. Pattern Recog.* pp. 2790–2799 (2018) **1**, **3**
51. Yu, L., et al.: Ec-net: an edge-aware point set consolidation network. *Eur. Conf. Comput. Vis.* (2018) **3**
52. Zeng, Z., Dong, M., Zhou, J., Qiu, H., Dong, Z., Luo, M., Li, B.: Deepla-net: Very deep local aggregation networks for point cloud analysis. In: *IEEE Conf. Comput. Vis. Pattern Recog.* pp. 1330–1341 (2025) **3**
53. Zhang, B., Yang, S., Chen, H., Yang, C., Jia, J., Jiang, G.: Point cloud upsampling using conditional diffusion module with adaptive noise suppression. In: *IEEE Conf.*

- Comput. Vis. Pattern Recog. pp. 16987–16996 (2025) 3,
8
- 54. Zhao, H., Jiang, L., Jia, J., Torr, P.H., Koltun, V.: Point transformer. In: Int. Conf. Comput. Vis. pp. 16259–16268 (2021) 3
 - 55. Zhao, W., Liu, X., Zhong, Z., Jian, J., Gao, W., Li, G., Ji, X.: Self-supervised arbitrary-scale point clouds upsampling via implicit neural representation. In: Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR). pp. 1999–2007 (2022) 8
 - 56. Zhou, L., Du, Y., Wu, J.: 3d shape generation and completion through point-voxel diffusion. In: Int. Conf. Comput. Vis. pp. 5826–5835 (2021) 3