

1 **Appendix**

2 In Appendix A, we provide additional experiments results.

- 3 • In Appendix A.1, we further provide the full results for 3D vehicle aerodynamic design.
- 4 • In Appendix A.2, we provide more visualization of design results.
- 5 • In Appendix A.3, we provide more qualitative comparisons of topology-preserving refine-
- 6 ment.

7 Appendix B: The implementation details of baseline methods.

8

9 Appendix C: The dataset processing details.

10

11 Appendix D: The implementation details of 3DID.

12

13 Appendix E: The evaluation details of 3DID.

14

15 Appendix F: The broader impact of 3DID.

16

17 Appendix G: The limitations of 3DID.

18

19 Appendix H: The licenses of datasets, codes, and models used in this paper.

20 A Additional Results

21 A.1 Full Results for 3D Vehicle Aerodynamic Design

22 Here we present the full statistical results of our experiments, including 95% confidence intervals
 23 for all compared methods, shown in Table 1. A box plot of the simulation-derived drag coefficient
 24 (Sim-Drag) is shown in Figure 1, illustrating the distribution, variability, and outlier behavior across
 different approaches.

Table 1: Quantitative comparison for aerodynamic vehicle design.

Method	Pred-Drag↓	Sim-Drag↓	Novelty↑	Coverage↑
GP, Voxel	0.2997±0.0436	0.4254 ± 0.0351	1.0399±0.0572	0.5200±0.0675
GP, Voxel+PCA	0.3059±0.0490	0.4363 ± 0.0425	0.9734±0.0195	0.5850±0.0675
CEM, Voxel	0.2951±0.0421	0.4097 ± 0.0279	0.9792±0.0213	0.4350±0.0676
CEM, Voxel+PCA	0.3088±0.0478	0.4393 ± 0.0469	0.9864±0.0250	0.5100±0.0600
CEM, TripNet	0.3154±0.0476	0.4161 ± 0.0415	1.0399±0.0323	0.6050± 0.0725
Backprop, Voxel	0.2979±0.0314	0.4146 ± 0.0244	0.9860±0.0204	0.4750±0.0675
Backprop, Voxel+PCA	0.3061±0.0576	0.4614 ± 0.0316	0.9798±0.0208	0.4950±0.0675
Backprop, TripNet	0.3153±0.0472	0.4170 ± 0.0444	1.0294±0.0290	0.5900±0.0700
3DID–NoTopoRefine (ours)	0.2623±0.0373	0.3766 ± 0.0393	0.9195±0.0213	0.6950 ±0.0627
3DID (ours)	0.2607 ±0.0331	0.3536 ± 0.0313	1.1709 ±0.0282	0.4300±0.0650

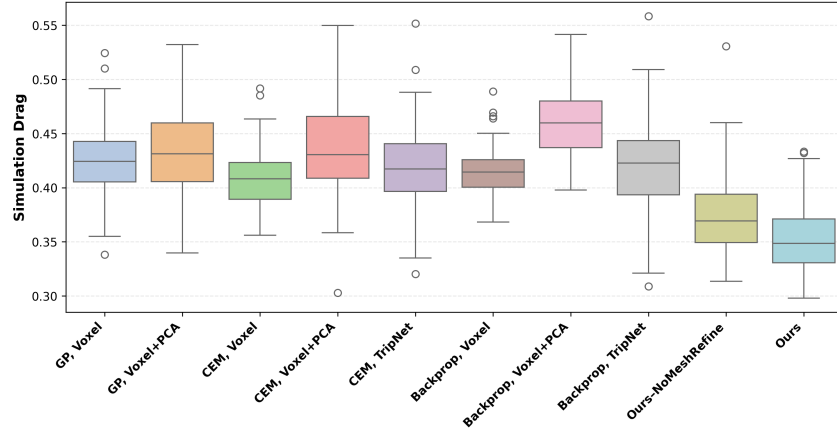


Figure 1: The box plot of the simulation-derived drag coefficient.

26 A.2 Visualization of 3DID Design

27 Additional visualizations of our designs are provided in Figure 2, where each design is shown
 28 alongside its geometry and corresponding physical fields.

29 A.3 Comparisons of Topology-Preserving Refinement

30 We provide additional qualitative comparisons in Figure 3 to demonstrate the effectiveness of our
 31 refinement stage. As shown, the design candidates consistently evolve toward a fastback profile
 32 after refinement, exhibiting reduced low-velocity recirculation regions and enhanced downward flow
 33 patterns, which indicate improved aerodynamic performance.

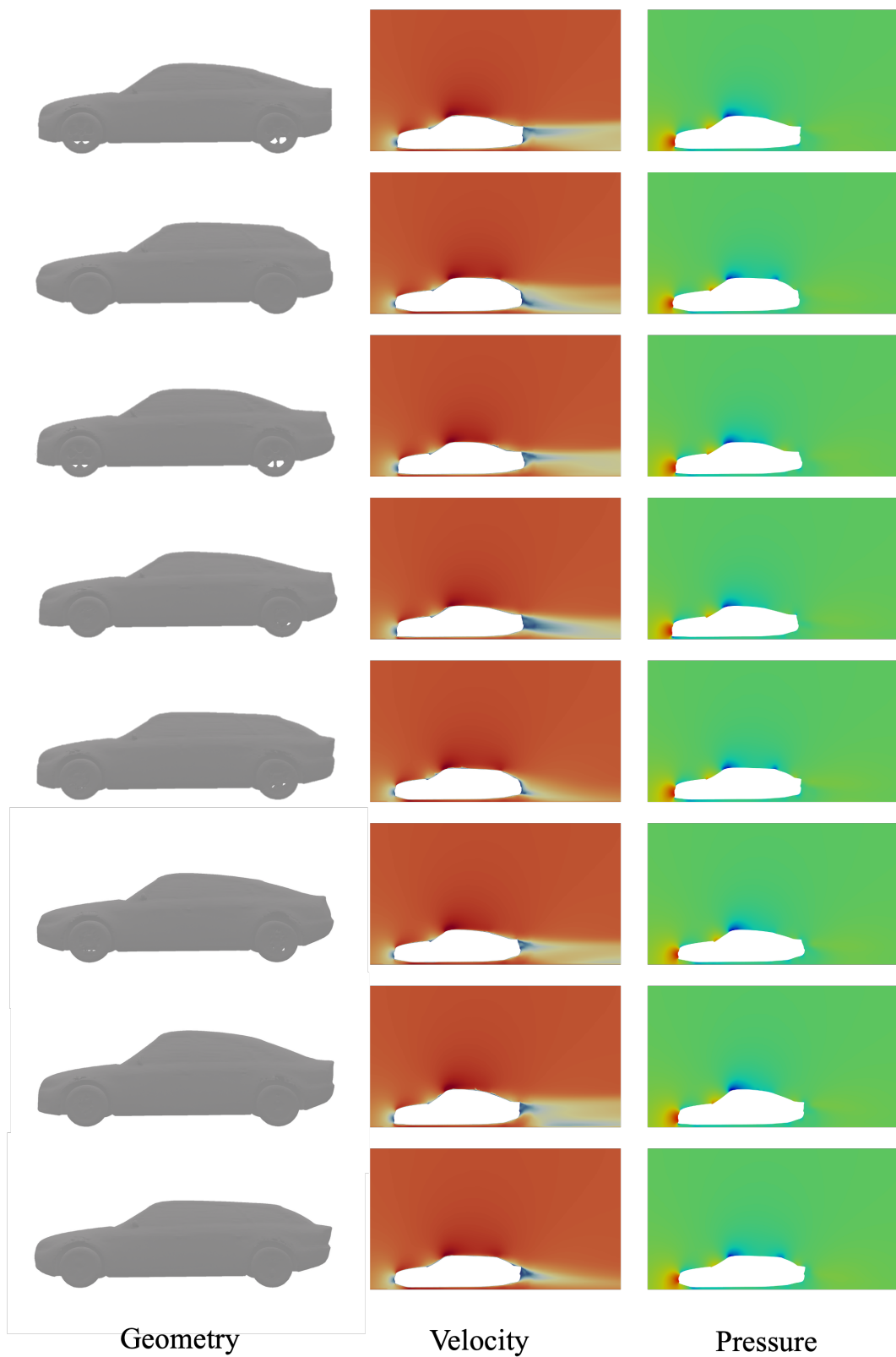


Figure 2: **Qualitative results of our 3DID.** Each row displays a design candidate along with its corresponding velocity and pressure field heatmaps.

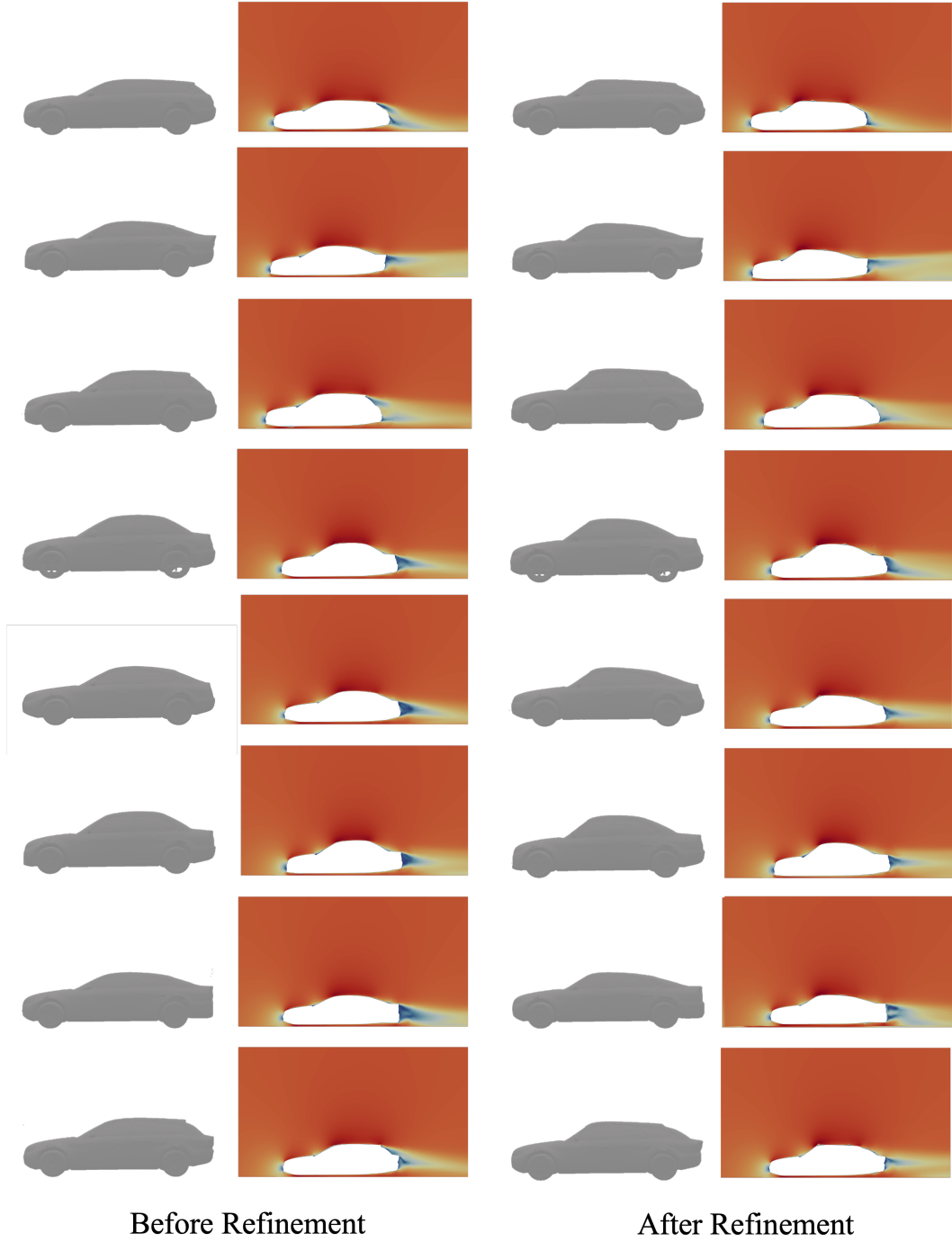


Figure 3: **Qualitative comparisons of topology-preserving refinement.** Each row presents two design candidates comparisons with their geometry and simulated velocity field heatmaps.

34 B Baseline implementation details

35 In our experiments, we compare our method against traditional sampling-based and backpropagation-
 36 based approaches using various design representations. As baselines, we include the Cross-Entropy
 37 Method(CEM) [1], the Gaussian-process surrogate with Bayesian optimization(GP) [2], and the
 38 gradient-based backpropagation method(Backprop) [3]. For representations, the optimizer is instanti-
 39 ated with three representations: a dense voxel grid [4], a PCA-compressed voxel grid (Voxel+PCA) [5],
 40 and a pure geometry triplane network (TripNet) [6]. For each representation, we train a VAE model [7]
 41 to compress the high-dimensional geometry into a compact latent code, which serves as the optimiza-
 42 tion space for inverse design.

43 B.1 Representation Baseline

44 **Voxel.** We train a voxel VAE [7] model directly on dense voxelized geometry to learn a latent
 45 embedding, as demonstrated in Figure 4. To train the model, we utilize the entire DrivAerNet++ [8]
 46 dataset, and voxelize the provided geometry with 256^3 resolution. For the Encoder, we leverage a
 47 sequence of 3D convolution layers followed by batch normalization and LeakyReLU to encode the
 48 voxel grid into a compact latent z_{voxel} . For voxel decoder, the latent vector z_{voxel} is first projected
 49 to a high-dimensional feature space and reshaped into a 3D tensor. A sequence of 3D transposed
 50 convolutional layers is then applied to reconstruct the voxel grid from this intermediate representation.
 51 Additionally, a separate drag prediction head, implemented as a multi-layer perceptron (MLP), is
 52 applied to estimate the target drag coefficient. We train the VAE model with reconstruction loss $\mathcal{L}_{\text{recon}}$
 53 , KL loss \mathcal{L}_{KL} , and the drag coefficient prediction loss $\mathcal{L}_{\text{drag}}$. The hyperparameters of the model and
 54 training are provided in Table 2

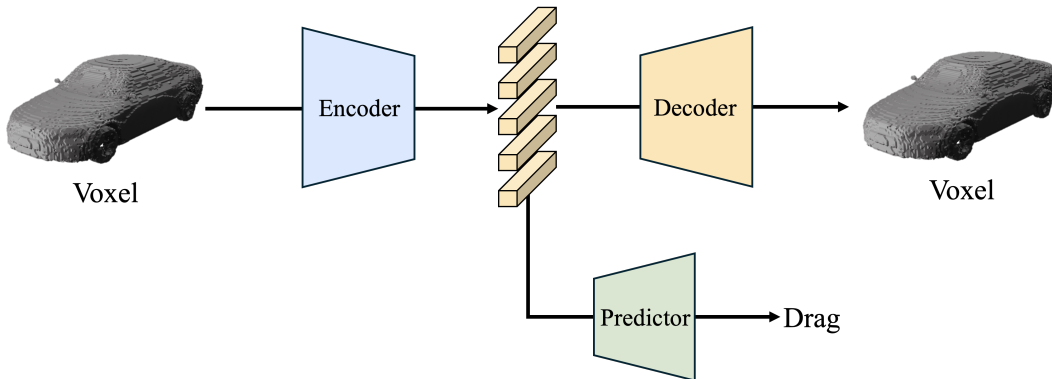


Figure 4: The overview of Voxel-VAE.

55 **Voxel-PCA.** Our Voxel-PCA representation is based on the representation proposed by [5] with
 56 modifications. In contrast to the Voxel-VAE, which directly uses voxel grids as input, the Voxel-PCA
 57 model first applies a dimensionality reduction step before downstream processing, as shown in
 58 Figure 5. Specifically, given the voxel data, we perform PCA [9] to obtain a compact representation
 59 of each geometry. Then, with this representation, we leverage a series of MLPs to encode the reduced
 60 features into a latent code $z_{\text{voxel-pca}}$. For reconstruction, an MLP decoder is first applied to reconstruct
 61 the PCA features from the latent code, which are then projected back to the voxel grid using the
 62 inverse PCA transformation. For drag prediction, similar to Voxel-VAE, a separate drag prediction
 63 head is applied to estimate the target drag coefficient. Our Voxel-PCA model is also trained with
 64 reconstruction loss $\mathcal{L}_{\text{recon}}$, KL loss \mathcal{L}_{KL} and drag prediction loss $\mathcal{L}_{\text{drag}}$. The hyperparameters of the
 65 model and training are provided in Table 3.

66 **TripNet.** Our TripNet representation is a pure geometry-based triplane representation, similar to the
 67 one proposed in [6], where it was used for forward prediction. The training procedure mirrors that
 68 of our unified physics-geometry framework, but excludes the physical field prediction branch, as
 69 illustrated in Figure 6. To obtain the representation, we utilize transformers with learnable tokens to
 70 extract features from the input point cloud. These features are then decoded using a transformer-based
 71 decoder and a geometry mapping network to predict the occupancy field of the design geometry. We

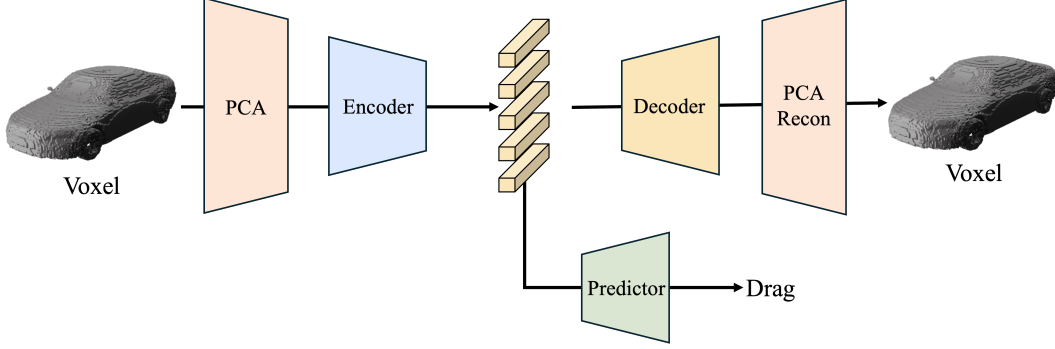


Figure 5: The overview of Voxel-PCA-VAE.

utilize the Binary Cross-Entropy loss \mathcal{L}_{BCE} and KL loss \mathcal{L}_{KL} to supervise the training of VAE. To further predict the drag coefficient, we adopt the same U-Net architecture used in our objective-guided diffusion model. The TripNet-VAE architecture adopts the same hyperparameter configuration as the geometry branch of our PG-VAE. More training hyperparameters are provided in Table 4.

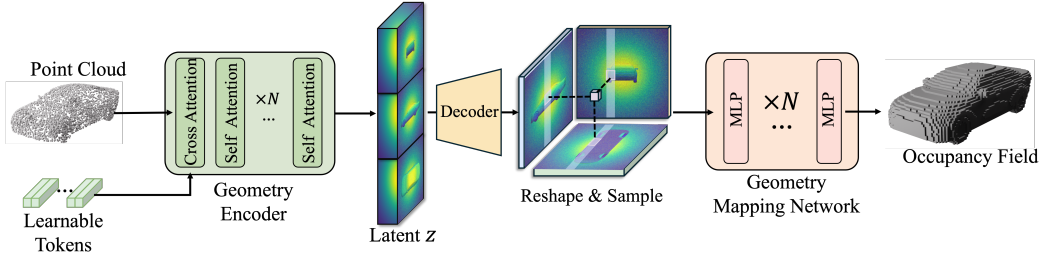


Figure 6: The overview of TripNet-VAE.

B.2 Optimization Baseline

CEM. Cross Entropy Method [1] is a traditional sampling-based optimization method widely used in classical inverse design problems. It starts with an initial distribution, and in each iteration, it samples multiple candidates from the current distribution. Then, these candidates are evaluated against the target objective function to select a subset of elite samples with the best performance. The distribution parameters are updated based on these elite samples. A smoothing coefficient controls the rate of distribution updates between iterations. This process continues until convergence or a maximum number of iterations is reached. In our experiment, we utilize a Gaussian distribution derived from encoded randomly selected samples as the initial distribution to provide a valid starting point.

GP. Gaussian-process surrogate with Bayesian optimization is a classical optimization method for black-box optimization [10, 11]. Bayesian optimization (BO) operates by constructing a probabilistic surrogate model, commonly a Gaussian-process model, to approximate the objective function based on past observations. At each iteration, an acquisition function is used to balance exploration of uncertain regions and exploitation of promising areas, guiding the selection of the next evaluation point. This strategy enables efficient optimization in high-cost or sample-limited scenarios by focusing evaluations on the most informative regions of the design space. While this method is effective in low-dimensional settings, constructing an accurate GP model becomes computationally expensive and challenging as the dimensionality of the design space increases. Therefore, GP-based Bayesian optimization is typically limited to small-scale or low-dimensional problems, where the surrogate can be reliably trained. In our experiment, our Gaussian process employs a Matérn kernel with constant and white noise components to model the objective function. At each iteration, Expected Improvement (EI) is used as the acquisition function.

98 **Backprop.** With the trained surrogate models, end-to-end backpropagation enables efficient gradient-
 99 based optimization of the design, leveraging the differentiability of the surrogate to guide updates [3,
 100 12]. In our experiments, we use the trained drag predictor as the surrogate and update the latent code
 101 using the Adam optimizer.

102 C Dataset processing details.

103 In this work, we conduct experiments on DrivAerNet++ [8], which is the largest aerodynamic car
 104 design dataset, comprising diverse car designs with corresponding CFD simulations. To train our
 105 model, we use the dataset with 8085 car designs to extract the point cloud and physical field. We
 106 first normalize each geometry of cars to fit within a unit cube, then uniformly sample 50,000 points
 107 with corresponding normals from the geometry surface. For the physical field, we apply the same
 108 scaling factor to ensure alignment with the normalized geometry. Subsequently, we randomly sample
 109 50,000 points within the unit cube and interpolate the physical field values at each location. These
 110 points serve as the input of our PG-VAE. For supervision, we additionally sample another 50,000
 111 points, each annotated with both occupancy values and physical field data. In this work, we focus on
 112 the pressure and velocity fields for the physical field representation, as wall shear stress is defined
 113 only on the surface of the geometry and is thus not suitable for volumetric sampling. During physical
 114 field interpolation, since some DrivAerNet++ samples are simulated using only half of the geometry,
 115 we map each sampled point to its symmetric counterpart when necessary. For the U-Net and GNN
 116 surrogate models used in guided diffusion sampling and topology-preserving optimization, we employ
 117 the drag coefficient values provided by the DrivAerNet++ dataset as ground truth supervision during
 118 training.

119 D Implementation details.

120 Our framework consists of three key components: the Physics–Geometry VAE (PG-VAE), Objective-
 121 Guided Diffusion, and Topology-Preserving Refinement. Below, we provide detailed implementation
 122 descriptions for each component.

123 **PG-VAE.** The PG-VAE serves to compress both the design geometry and the corresponding physical
 124 field into a unified latent representation. We sample $N_g = N_p = 50,000$ points for the geometry
 125 and physical field branches, respectively. The encoder consists of one cross-attention layer and
 126 eight self-attention layers, each with 12 attention heads and an embedding dimension of $d_z = 64$.
 127 We use $r = 64$ for learnable tokens, and each with a channel dimension of $d_e = 768$, to enhance
 128 representation expressiveness. The latent code dimension is set to $d_z = 32$. The decoder architecture
 129 consists of one self-attention layer followed by five ResNet blocks [13], which upsample the latent
 130 vector into a triplane representation with resolution $R = 256$ and channel dimension $d_t = 64$. The
 131 output triplane is then queried using a mapping network composed of five fully connected layers
 132 with a hidden size of 32 per branch. We adopt a semi-continuous occupancy formulation [14]
 133 and supervise both occupancy and physical field predictions using 50,000 sampled points within
 134 the normalized unit cube. We optimize the VAE using a combination of three loss terms: binary
 135 cross-entropy loss ($\lambda_{\text{BCE}} = 10^{-3}$), mean squared error for field regression ($\lambda_{\text{MSE}} = 10^{-5}$), and KL
 136 divergence ($\lambda_{\text{KL}} = 10^{-6}$). Training is performed using the AdamW optimizer [15] with a learning
 137 rate of 1×10^{-4} , batch size 8 per GPU, for 100,000 steps. We use four NVIDIA RTX A6000 GPUs
 138 to train the model.

139 **Objective-Guided Diffusion.** To explore the latent design space efficiently, we employ a latent-space
 140 diffusion model composed of 10 DiT blocks [16], each containing 16 attention heads with a head
 141 dimension of 72. The diffusion process includes 1,000 denoising steps. During inference, an auxiliary
 142 U-Net surrogate network is used to predict the task objective directly from the latent code z , thereby
 143 guiding the sampling process toward optimal designs. The diffusion model is trained using a learning
 144 rate of 5×10^{-5} , batch size of 4 per GPU, for 300,000 steps with the AdamW optimizer. We use four
 145 NVIDIA RTX A6000 GPUs to train the diffusion model.

146 **Topology-Preserving Refinement.** To refine the initial design candidates while maintaining mesh
 147 topology, we apply a Free-Form Deformation (FFD) grid with $20 \times 6 \times 6$ control points along the
 148 x, y, and z axes, respectively. The deformation is guided by a surrogate model based on Mesh-
 149 GraphNet [17], which comprises 8 message-passing blocks and operates on the surface mesh. The

MeshGraphNet is trained to predict the drag force from a deformed mesh, serving as a differentiable objective function during refinement. This model is trained with a learning rate of 1×10^{-5} , batch size of 8 per GPU, for 100,000 steps, using AdamW as the optimizer. We use two NVIDIA RTX A6000 GPUs to train the MeshGraphNet.

E Evaluation details.

In our experiments, we evaluate the design candidates using four metrics: predicted drag force (Pred-Drag), simulated drag force (Sim-Drag), novelty, and coverage.

Pred-Drag. We use the pretrained surrogate model to estimate the drag force of each candidate mesh. Given the mesh of designed candidates M^* , our surrogate model directly predict the objective drag force $\hat{\mathcal{J}}$ which can be formalized as:

$$\hat{\mathcal{J}} = \mathcal{F}_{\text{surrogate}}(M^*), \quad (1)$$

where $\mathcal{F}_{\text{surrogate}}$ denotes the learned mapping from 3D mesh geometry to the predicted drag coefficient. For our surrogate model, we adopt a MeshGraphNet [17] with 8 message passing blocks as the surrogate model. Unlike the model used in our topology-preserving refinement stage, this predictor operates solely on geometry, without requiring the associated physical field. To train the model, we use the entire DrivAerNet++ [8] dataset. Given that different representations may produce varying topological structures, we apply remeshing and simplification to all candidates for fair comparison.

Sim-Drag. To obtain an unbiased evaluation of the generated designs, we perform high-fidelity Computational Fluid Dynamics (CFD) simulations and compute the corresponding drag coefficients. Following DrivAerNet++ [8], we employ the OpenFOAM@V11 [18] to conduct steady-state incompressible simulation using the $k - \omega$ SST turbulence model, based on Menter’s formulation [19]. We performed a series of quality checks to ensure the generated geometries were simulation-ready and properly aligned within the CFD domain. During simulation, considering the computation cost, we set the maximum local cells to 10 million and the maximum global cells to 50 million in snappyHexMesh. The simulation iterates for 1000s, and we use the final 30% simulation data to calculate the average drag coefficient. The hyperparameters of our simulation are provided in Table 5.

Novelty. To quantitatively assess how different the generated designs are from the training data, we measure the novelty of each candidate. Specifically, novelty is computed as the average distance from each generated design to its nearest neighbor in the training set, reflecting how distinct the generated designs are from existing ones. Let $\{g_i\}_{i=1}^{N_g}$ denote the set of generated designs and $\{t_j\}_{j=1}^{N_t}$ denote the set of training designs in the feature space. The novelty is defined as:

$$\text{Novelty} = \frac{1}{N_g} \sum_{i=1}^{N_g} \min_j d(g_i, t_j), \quad (2)$$

where $d(\cdot, \cdot)$ denotes the distance between feature embeddings, computed using the pretrained PointNet encoder [20].

Coverage. The coverage metric (also known as recall) evaluates how well the generated designs cover the training distribution by measuring, for each training sample, the distance to its nearest generated design (using a k-nearest neighbor lookup) and reporting the fraction of training examples that fall within a predefined threshold. Let $\{g_i\}_{i=1}^{N_g}$ denote the set of generated designs and $\{t_j\}_{j=1}^{N_t}$ denote the set of training designs in the feature space. The coverage is defined as:

$$\text{Coverage} = \frac{1}{N_t} \sum_{j=1}^{N_t} \mathbf{1}[\min_i d(t_j, g_i) \leq \tau] \quad (3)$$

where $d(\cdot, \cdot)$ is a distance metric, τ is a predefined threshold, and $\mathbf{1}[\cdot]$ is the indicator function that equals 1 if the condition is true and 0 otherwise.

189 F Broader Impacts

190 **Academic Impact.** 3DID’s methodology, which enables direct navigation through 3D physics-
191 geometry space, simplifies the 3D inverse design process. With the unified physics-geometry
192 representation, the computation gap between 3D and lower-dimensional inverse design is narrowed,
193 allowing researchers to focus more on exploring cutting-edge inverse design strategies rather than
194 being constrained by computational limitations. With the two-stage optimization strategy, our method
195 balances between exploration and validity, offering researchers an effective approach for inverse
196 design involving 3D geometry.

197 **Social Impact.** The proposed 3D Inverse Design (3DID) framework extends the scope of geometry-
198 driven design by enabling direct optimization of full 3D structures from scratch. By combining unified
199 physics-geometry representations with physics-aware optimization, our method opens the door to
200 more efficient, automated design workflows in fields such as aerospace engineering, biomedicine,
201 additive manufacturing, and nanophotonics. In particular, 3DID can be applied to complex design
202 tasks that traditionally rely on expert-crafted initial geometries and time-consuming simulation-
203 based evaluations. In mechanical engineering, it can be used to optimize structural components for
204 strength, weight, and thermal performance without manual trial-and-error. In the medical field, 3DID
205 enables the fabrication of patient-specific implants by automatically generating geometries tailored to
206 individual physiological and functional requirements.

207 G Limitations

208 **Limited to static physical fields.** Despite the fact that 3DID achieves impressive results, a significant
209 limitation is its focus on static fields. The current framework does not support inverse design
210 involving time-dependent or dynamic physical fields. Time-dependent physical systems often
211 involve solid geometries coupled with evolving physical properties over time. This would pose
212 challenges for representation and optimization within our framework. Enhancing 3DID with time-
213 aware representations and models may address these limitations, which we leave as an important
214 direction for future work.

215 **Limited to single objective optimization.** In 3DID, we address the inverse problem with a single
216 objective, which may limit its applicability for broader scenarios. Although it is straightforward to
217 aggregate multiple objectives into a single composite loss, this approach may overlook potential con-
218 flicts and trade-offs between objectives. Extending 3DID to support true multi-objective optimization
219 is a promising direction for future research.

H License

The code will be publicly accessible. We use standard licenses from the community. We include the following licenses for the codes, datasets, and models we used in this paper.

1. Dataset

- DrivAerNet++ [8]: CC BY-NC 4.0

2. Codes

- NVIDIA PhysicsNeMo: Apache License 2.0

3. Evaluation

- OpenFOAM [18]: GNU General Public License

Table 2: **Hyperparameters for Voxel-VAE**

Hyperparameter name	Value
Hyperparameters for Voxel-VAE architecture:	
Input shape	[8, 256, 256, 256]
Output shape	[8, 256, 256, 256]
Number of 3D convolution layer	5
Dimension of latent z_{voxel}	512
Number of 3D transposed convolutional layer	5
Number of MLPs in drag predictor	5
Batch size	8
Dimension of encoder	(1, 32, 64, 128, 256, 512)
Dimension of voxel decoder	(512, 256, 128, 64, 32, 1)
Dimension of drag predictor	(512, 256, 128, 64, 32, 1)
Hyperparameters for Voxel-VAE training:	
Optimizer	AdamW
Learning rate	$1e-4$
Learning steps	100K
Learning rate adjustment strategy	Cosine
Warm-up steps	5K
$\mathcal{L}_{\text{recon}}$ weight	10^{-3}
\mathcal{L}_{KL} weight	10^{-4}
$\mathcal{L}_{\text{drag}}$ weight	10^{-3}

Table 3: **Hyperparameters for Voxel-PCA-VAE**

Hyperparameter name	Value
Hyperparameters for Voxel-PCA-VAE architecture:	
PCA output dimension	400
Number of MLP layers in encoder	4
Dimension of latent $z_{\text{voxel-pca}}$	64
Number of MLP layers in decoder	4
Number of MLPs in drag predictor	2
Batch size	32
Dimension of encoder	(400, 256, 128, 64, 64)
Dimension of PCA decoder	(64, 64, 128, 256, 400)
Dimension of drag predictor	(64, 32, 1)
Hyperparameters for Voxel-PCA-VAE training:	
Optimizer	AdamW
Learning rate	$5e-4$
Learning steps	100K
Learning rate adjustment strategy	Cosine
Warm-up steps	5K
$\mathcal{L}_{\text{recon}}$ weight	10^{-2}
\mathcal{L}_{KL} weight	10^{-4}
$\mathcal{L}_{\text{drag}}$ weight	10^{-3}

Table 4: **Hyperparameters for TripNet VAE**

Hyperparameter name	Value
Hyperparameters for TripNet-VAE training:	
Batch size	8
Optimizer	AdamW
Learning rate	$1e-4$
Learning steps	100K
Learning rate adjustment strategy	Cosine
Warm-up steps	5K
\mathcal{L}_{BCE} weight	10^{-3}
\mathcal{L}_{KL} weight	10^{-6}

Table 5: CFD Simulation Parameters for OpenFOAM

Parameter name	Value
Solver Configuration:	
OpenFOAM version	v11
Solver	incompressibleFluid
Algorithm	SIMPLE
Turbulence model	k- ω -SST
Simulation type	Steady-state RANS
Flow Conditions:	
Flow velocity (u_∞)	30 m/s
Kinematic viscosity (ν)	1.56×10^{-5} m ² /s
Air density (ρ)	1.184 kg/m ³
Turbulent kinetic energy (k)	0.375 m ² /s ²
Specific dissipation rate (ω)	1.78 s ⁻¹
Computational Domain:	
Domain dimensions	44×8×6.4 m
Inlet distance	12 m upstream
Outlet distance	32 m downstream
Solver Tolerances:	
Pressure absolute tolerance	1×10^{-6}
Pressure relative tolerance	3×10^{-2}
Velocity absolute tolerance	1×10^{-8}
Velocity relative tolerance	5×10^{-3}
Turbulence absolute tolerance	1×10^{-8}
Turbulence relative tolerance	1×10^{-3}
Potential solver absolute tolerance	1×10^{-7}
Potential solver relative tolerance	1×10^{-2}
Mesh Refinement:	
Surface refinement level	3-4
Feature refinement level	4
Regional refinement level	2
Wake refinement level	2
Boundary layers	5 layers
Layer expansion ratio	1.2
Final layer thickness	0.5
Force Calculation:	
Reference length (l_{ref})	4.777 m
Reference area (A_{ref})	2.0 m ²
Reference center	(0, 0, 0)
Drag direction	(1, 0, 0)
Lift direction	(0, 0, 1)
Simulation Control:	
End time	1000 s
Time step	1 s
Write interval	100 steps
Force coeffs write interval	10 steps

References

- [1] Reuven Y Rubinstein and Dirk P Kroese. *The cross-entropy method: a unified approach to combinatorial optimization, Monte-Carlo simulation and machine learning*. Springer Science & Business Media, 2004.
- [2] Peter ZG Qian and CF Jeff Wu. Bayesian hierarchical modeling for integrating low-accuracy and high-accuracy experiments. *Technometrics*, 50(2):192–204, 2008.
- [3] Kelsey Allen, Tatiana Lopez-Guevara, Kimberly L Stachenfeld, Alvaro Sanchez Gonzalez, Peter Battaglia, Jessica B Hamrick, and Tobias Pfaff. Inverse design for fluid-structure interactions using graph network simulators. *Advances in Neural Information Processing Systems*, 35:13759–13774, 2022.
- [4] Premith Kumar Chilukuri, Binyang Song, SungKu Kang, and Ran Jin. Generating optimized 3d designs for manufacturing using a guided voxel diffusion model. In *International Manufacturing Science and Engineering Conference*, volume 88117, page V002T07A006. American Society of Mechanical Engineers, 2024.
- [5] Jonathan Tran, Kai Fukami, Kenta Inada, Daisuke Umehara, Yoshimichi Ono, Kenta Ogawa, and Kunihiko Taira. Aerodynamics-guided machine learning for design optimization of electric vehicles. *Communications Engineering*, 3(1):174, 2024.
- [6] Qian Chen, Mohamed Elrefaie, Angela Dai, and Faez Ahmed. Tripnet: Learning large-scale high-fidelity 3d car aerodynamics with triplane networks. *arXiv preprint arXiv:2503.17400*, 2025.
- [7] Diederik P Kingma, Max Welling, et al. Auto-encoding variational bayes, 2013.
- [8] Mohamed Elrefaie, Florin Morar, Angela Dai, and Faez Ahmed. Drivaernet++: A large-scale multimodal car dataset with computational fluid dynamics simulations and deep learning benchmarks. *Advances in Neural Information Processing Systems (NeurIPS)*, 37:499–536, 2024.
- [9] Hervé Abdi and Lynne J Williams. Principal component analysis. *Wiley interdisciplinary reviews: computational statistics*, 2(4):433–459, 2010.
- [10] Timothy MS Jim, Ghifari A Faza, Pramudita S Palar, and Koji Shimoyama. Bayesian optimization of a low-boom supersonic wing planform. *AIAA journal*, 59(11):4514–4529, 2021.
- [11] Grégoire Mariethoz, Philippe Renard, and Jef Caers. Bayesian inverse problem and optimization with iterative spatial resampling. *Water Resources Research*, 46(11), 2010.
- [12] Tailin Wu, Willie Neiswanger, Hongtao Zheng, Stefano Ermon, and Jure Leskovec. Uncertainty quantification for forward and inverse problems of pdes via latent global evolution. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 320–328, 2024.
- [13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.
- [14] Shuang Wu, Youtian Lin, Feihu Zhang, Yifei Zeng, Jingxi Xu, Philip Torr, Xun Cao, and Yao Yao. Direct3d: Scalable image-to-3d generation via 3d latent diffusion transformer. *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.
- [15] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.
- [16] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 4195–4205, 2023.
- [17] Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter Battaglia. Learning mesh-based simulation with graph networks. In *International Conference on Learning Representations (ICLR)*, 2020.

- 277 [18] Christopher Greenshields. *OpenFOAM v11 User Guide*. The OpenFOAM Foundation, London,
278 UK, 2023.
- 279 [19] Florian R Menter, Martin Kuntz, Robin Langtry, et al. Ten years of industrial experience with
280 the sst turbulence model. *Turbulence, heat and mass transfer*, 4(1):625–632, 2003.
- 281 [20] Alex Nichol, Heewoo Jun, Prafulla Dhariwal, Pamela Mishkin, and Mark Chen. Point-e: A
282 system for generating 3d point clouds from complex prompts. *arXiv preprint arXiv:2212.08751*,
283 2022.