ECOLE
Experience-based Computation:
Learning to Optimise

Project Number: 766186
Project Acronym: ECOLE
Project title: Experienced-based Computation: Learning to Optimise

Deliverable D1.3

Multi-domain Optimization based on Online and Offline Experience Exploitation for Dynamic Environments

(incl. Software Tools and Components)

Authors:
Thiago Rios, Sneha Saha, Stefan Menzel, Bernhard Sendhoff – HRI-EU

Project Coordinator: Professor Xin Yao, University of Birmingham
Beneficiaries: Leiden University, Honda Research Institute Europe, NEC Laboratories Europe

H2020 MSCA-ITN
Date of the report: 30.09.2021

Contents

Executive Summary

Engineers intuitively exploit their experience to address challenging optimization problems during the development of new products. However, new optimization instances may lack an intuitive synergy with similar problems, which other design teams encountered in the past, *e.g.*, current designs have different parameterizations or resolutions compared to past results, or different performance metrics had been used. Therefore, knowledge transfer in optimization is challenging and often performed in a non-systematic fashion. In work package WP1 of the ECOLE project, we research and develop techniques to learn novel representations of 3D designs that foster knowledge transfer in optimization and utilize these representations towards man-machine cooperative shape-generative tasks. Our proposed techniques learn domain-agnostic design representations directly from computer aided engineering (CAE) data, which allows us to transfer features between designs assigned to different problems and to sample the design population to address user preferences or increase the population diversity.

In this report, we present our methods and software in detail, which we developed for WP1, and propose a framework consisting of pre-processing and optimization stages. In the pre-processing stage, our framework includes methods for sampling 3D point clouds, training, benchmarking, and applying deep-generative models, where we learn a notion of experience from benchmark CAE data in an offline fashion. In the optimization stage, we apply the shape-generative models and explore the properties of the learned representations to train surrogate models, to initialize the populations of evolutionary algorithms online in a smart way, and to transfer knowledge in multi-task optimization scenarios. We finally provide details on the technical implementation of the algorithms, which includes requirements, usage, licensing, and deployment of our software components and tools.

Our code is available online at https://github.com/ECOLE-ITN/GDL4DesignApps.

Scientific Achievements

We summarize the scientific achievements described in this report in the following table of research questions and short answers.

Research questions	**Discussion**
What are the advantages of 3D point cloud autoencoder approaches with respect to other data-driven design representations?	Autoencoders allow more degrees of freedom to learn design features than data-driven techniques based on structured data. In addition, autoencoders also enable generative design with topology changes in contrast to traditional design modification methods applied in industry, like e.g., shape morphing techniques.
How do autoencoder-based geometric representations address knowledge-transfer challenges in design optimization?	Autoencoders learn a domain-agnostic representation, which performs as a mathematical basis to model and transfer properties for a wide range of domains.
How can geometric deep learning architectures exploit the engineering experience embedded in handcrafted geometric representations?	Deep learning architectures utilize historical design data for learning a compact representation. Through 3D point cloud pre-

	processing, existing design data is unified and therefore, allows to combine various shape source data, which usually has been generated with multiple tools, such as shape morphing methods, spline-based methods or shape boundary representations.
How is experience given as design and performance data from past optimizations captured and utilized in current engineering optimization tasks using (variational) autoencoders?	By learning surrogate models based on the latent representations of the (variational) autoencoders we build accurate multi-layer perceptron models offline that map 3D point clouds to geometric and engineering performance metrics and allow an efficient online utilization in engineering design optimization tasks.
How to increase the amount of information in the learned latent space of autoencoders to improve the quality of metamodels for surrogate-assisted design optimizations?	By enforcing the autoencoder to learn implicit information on the global point cloud structure, e.g., by ordering the points, the information content in the latent space is increased.
How to exploit data-driven variational autoencoder representations for improving the convergence in multi-criteria decision making?	We propose a “warm start” based on optimized designs for the objectives of interest and exploit the linear interpolation in the variational autoencoder latent space to address the user preferences.
What is the advantage of experience transfer through autoencoder representations in engineering design optimization?	Compact representations based on autoencoders offer efficient parameterizations for multiple tasks, which enable the transfer of latent features during optimization to foster commonality in the optimal designs for increased cost efficiency.
How can efficient learning-driven systems strengthen automated design optimization frameworks in the context of engineering development?	By combining learning point cloud data and deformable mesh prototypes in a novel deep neural network architecture (Point2FFD) we enable the generation of realistic designs that are directly suitable for engineering simulation in an automated design optimization framework reducing overall optimization runtime.

1. Introduction

Engineers intuitively utilize experience to set up new optimization problems targeting faster convergence and solutions with higher quality. However, due to the increasing complexity of optimization scenarios in the automotive industry, transferring settings between optimization problems is often a challenging process. Furthermore, considering the experience of a single engineer may over-constrain the range of optimum designs, hindering design exploration. Hence, in the Experience-based Computation: Learning to Optimise (ECOLE) project, we address these challenges by proposing a novel and smart engineering optimization framework, which learns a notion of experience from optimization data histories and exploits said experience in similar, yet more challenging, current optimization problems in the automotive domain (Figure 1).

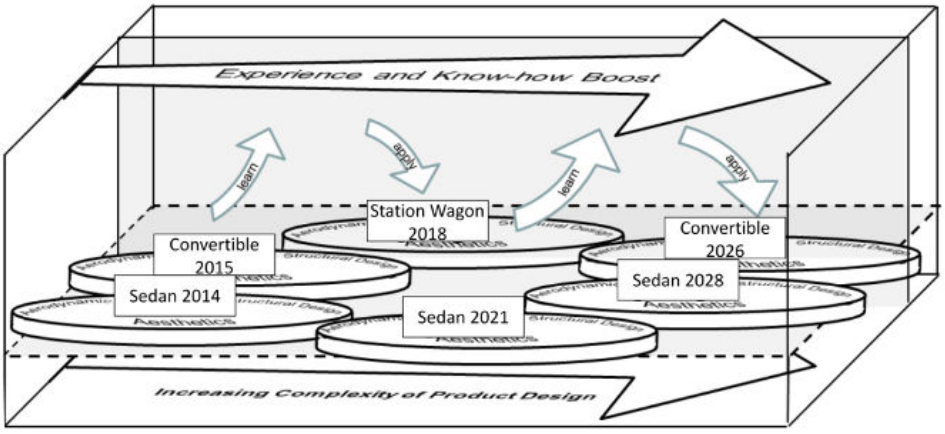

Figure 1– Vision for experience-based multi-criteria optimization in the automotive domain.

In this report, we summarize our research activities and results with respect to work package WP1, which focuses on multi-domain vehicle optimization improved by the integration and exploitation of experience, which is hidden in data from past product development cycles. Thus, we link scientific questions to industrial, often multi-domain problems that add a practical perspective to our research tasks. A systematic approach to model and transfer the experience in industrial optimization scenarios has the potential to improve the performance of new optimization instances in various aspects. For example, since the complexity of products increases over the development process, learning design features and simulated performance of designs allows engineers to sample, combine, and assess the performance of novel solutions faster and more accurately, which accelerates the design process. From a long-term perspective, computational experience-based models may cooperate with less experienced engineers to solve complex optimization problems and dial down the effects of team turnover by transferring knowledge from machine to humans more efficiently.

The parameterization of 3D geometries, also called design representation, is central for knowledge and experience transfer in optimization problems. The design features provide a mathematical basis to analyze different optimization results, to generate performance prediction models and to transfer solutions between different problems. Since engineers often manually select the design features based on their expertise and the domain of the optimization, design parameterizations

differ from one optimization to another, even between problems that belong to the same domain. In addition, representations change over time due to higher fidelities of models and adaptation of parameters based on product visions of design teams. Thus, transferring solutions and features from one optimization to another is not straightforward, particularly due to the lack of clear overlap and mapping between design spaces of different problems. However, since most of the design processes currently rely on virtual models and computer simulations developed with CAE tools, the design of multiple products generates sufficient digital data for machine learning algorithms to learn design features and novel shape-generative models that are transferrable to multiple problems.

Mining design features from CAE data is challenging due to the sparse, unstructured, and high-dimensional nature of the data samples. Particularly for computer simulation purposes, engineers discretize 3D shapes into polygonal meshes that comprise a permutation-invariant set of vertices connected by straight edges, which is an unsuitable input space for classical machine learning algorithms. With the advances on storage technology and increasing availability of powerful graphic processing units (GPUs), Bronstein *et al.* [1] introduced a novel class of algorithms called *geometric deep learning*, which addresses these challenges and extends the application of machine learning to unstructured data. Among the methods in this class, deep-generative models, like autoencoders, learn compact latent representations of geometric data, *e.g.,* 3D point clouds. Furthermore, since these architectures learn exclusively from geometric data, the latent space of the autoencoders is domain-agnostic, which allows the transfer of geometric features between shapes of different nature and to model performance metrics related to different domains. Hence, our approach (Figure 2) exploits geometric deep learning methods to learn design features from existing historical CAE data in an offline fashion and, thus, to also learn a notion of the users' expertise embedded in the data. Furthermore, the learned representations enable the knowledge transfer during the solution of concurrent optimization problems online, *e.g.*, by sharing solutions described in a common design space.

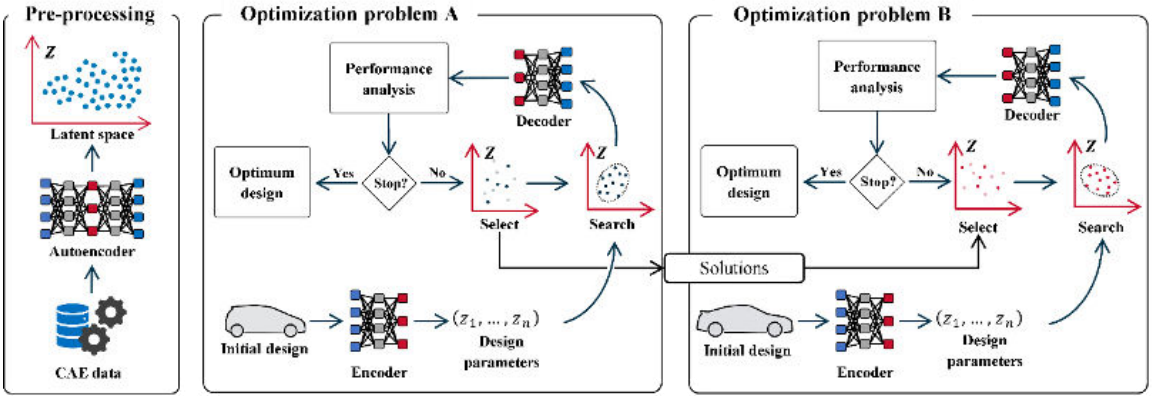

Figure 2 – Application scenario of design optimization with experience transfer.

For developing such a framework, we divided our approach into modules. For pre-processing (Section 2) the CAE data and learning the representations, we proposed three deep learning architectures, which learn compact design representations from 3D point cloud data in an offline and unsupervised fashion. Also in the pre-processing phase, we propose a feature visualization technique, which identifies the mapping between the learned variables and regions in the

geometries, such that the users can decide on which parameters to modify and during the optimizations and transfer between different problems. After the pre-processing we applied the networks in an online fashion in four real world-inspired vehicle design and optimization problems: The analysis of surrogate models based on latent space representations, a multi-objective optimization, a multi-task optimization and a single-objective optimization problem (Section 3). A comprehensive description of our developed software tools and components concludes the report (Section 4).

2. Pre-processing Stage

The main objectives of the pre-processing stage are to prepare the CAE data for the training and testing of our shape-generative models, and to enable the visualization of network features. For these algorithms, we assume that an existing set of geometries represented as polygonal meshes is available, in a data format that allows us to access, sample and modify the vertices of the mesh into 3D point clouds.

2.1.Geometric Data

2.1.1. Benchmark Data Set

As baseline data set, we utilize the ShapeNetCore repository [2], which comprises shapes represented in a standard 3D image format (OBJ), *i.e.*, polygonal meshes with texture, colors, and other rendering-related information. ShapeNet and ShapeNetCore are large 3D shape repositories, which are used as standard benchmark data sets for algorithm development and benchmarking. The dimensions of the shapes are rescaled to the space $[0,1]^3$ in a shape-independent fashion preserving the aspect ratio between axes. Furthermore, the meshes in the repository are non-isomorphic and have different quality levels, since the authors collected 3D models from multiple sources.

2.1.2. Point Cloud Sampling

We sample the point clouds from the polygonal meshes according to three techniques: Random uniform sampling (RUS), high-pass graph filter (HPF) and shrink-wrapping meshing (SWM). We explored the first two techniques in [3], where we assessed the scalability of point cloud autoencoder architectures to engineering CAE models, as also extensively reported in the ECOLE Deliverable 1.2 [4], which focused on our main findings and evaluations of our (variational) autoencoders. We introduced the SWM approach in [5], where we generated organized point clouds that have enabled us to quickly generate simplified mesh prototypes for engineering simulations.

In the RUS approach, we sample the vertices of a mesh assuming a uniform random probability. Hence, for a polygonal mesh with N vertices, any vertex has the probability $(N)^{-1}$ of being sampled. Also, the method is feature-agnostic, i.e., it neglects any information related to the distribution of points over the surface of the shape. Differently, the HPF approach assigns higher probability to points close to high-frequency shape transitions, *e.g.*, edges. To calculate the sampling probability, we apply the transition matrix A as graph shift operator (Eq. 1), which depends on the degree

matrix D and adjacency matrix W. The sampling probability of a vertex x_i is proportional to the magnitude of the response h_g of the operator (Eq. 2). Since the operator indicates the difference between the position of a point and the convex combination of its neighbors, sharper features, *e.g.*, edges, yield responses with higher magnitude and, thus, lead to higher sampling probability (Figure 3).

$$A = D^{-1}W \tag{1}$$

$$h_g(x_i, A) = x_i - \sum_{j \in \mathcal{N}} A_{i,j} x_j \tag{2}$$

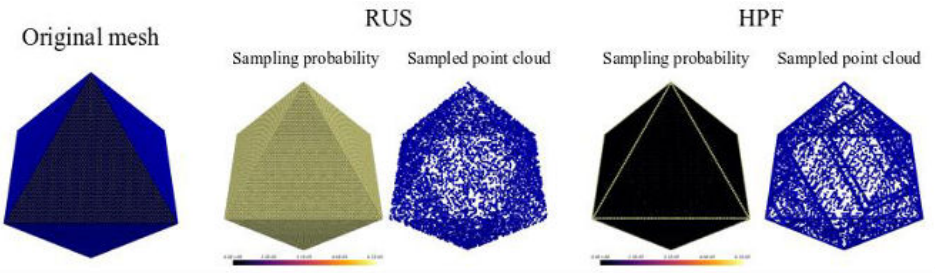

Figure 3 – Application of RUS and HPF sampling strategies on a regular octahedron mesh. Brighter colors indicate higher sampling probability.

The shrink-wrapping meshing (SWM) algorithm works according to a different principle. The algorithm first fits an initial mesh, which we in our experiments assume as a rectangular box, to the dimensions of the target geometry. In a second step, the initial mesh is iteratively shrunk by approximating each vertex x_i to its nearest vertex $x_{n,i}$ in the target mesh based on the distance between points and a step size α (Eq. 3). Since the shrinking may generate clusters of points in less dense regions of the target mesh, in a last step, the algorithm relaxes the generated mesh with a Laplacian algorithm [6], improving the distribution of the points (Figure 4).

$$x_i^{t+1} = x_i^t + \alpha\left(x_{n,i}^t - x_i^t\right) \tag{3}$$

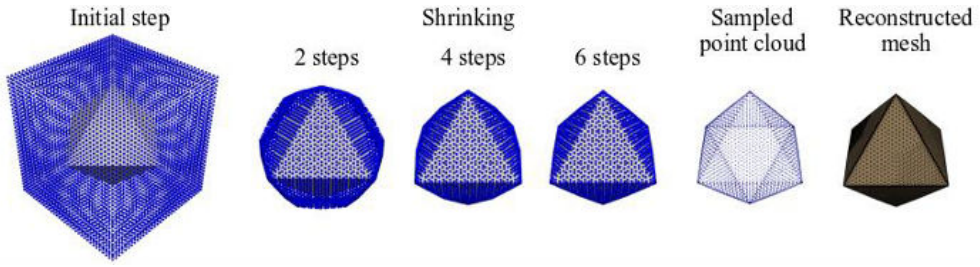

Figure 4 – Visualization of the point clouds obtained at intermediate and final steps with SWM applied to the mesh of a regular octahedron. On the right, the mesh reconstructed based on the initial mesh and point ordering.

By utilizing the same initial mesh for a set of shapes, SWM yields an organized set of 3D point clouds, where the points are ordered according to the vertex assignment of the mesh. Learning on sets of organized point clouds has two main advantages: First, since the point-correspondence between point clouds is known, it allows us to simplify the calculation of shape reconstruction

losses, which increases the computational efficiency of the training task. Second, organizing the points according to the graph vertex assignment allows us to quickly prototype meshes from the point clouds generated by our proposed models, since we can directly assign the predicted points to the corresponding vertices of the initial mesh [5], [7].

For our implementations, we integrated the three sampling techniques into a single algorithm (Algorithm 1). We assume that a data set of polygonal meshes is available, as well as the sampling probability p of the vertices for each geometry and the required size of the output point clouds N. We assume that any additional pre-processing was performed beforehand, *e.g.*, calculating the point sampling probability p and shrink-wrapping meshing, since these tasks are time-consuming and we only need to perform them once. For cases where meshes have different sizes, the algorithm verifies for each mesh whether the required point cloud size is larger than the number of available vertices. In this case, the algorithm enables the replacement (repetition) of samples, which performs as padding 2D images with black pixels for image processing tasks. If the meshes in the set are isomorphic and have size N, as obtained with SWM, the algorithm selects all the vertices according to the ordering assigned in the graph. For machine learning tasks, we group the point clouds in tensors $T(b, N, 3)$, where b is the batch size.

Algorithm 1 – Pseudo-algorithm of the random uniform sampling technique generalized to meshes with random number of vertices. In the algorithm, V and E are the mesh vertices and edges, respectively, N the number of required points and p the point sampling probability.

ALGORITHM: Point Cloud Sampling

INPUT: Polygonal mesh (V, E)**,** N, p
1: IF $N >$ size(V):
2: ind = sample N indices in $[0, size(V) - 1]$, with replacement and probability distribution p
3: ELSE IF $N <$ size(V):
4: ind = random N indices in $[0, size(V) - 1]$, without replacement and probability distribution p
5: ELSE
6: $ind = [0, \dots, N - 1]$
7: END
OUTPUT: 3D point cloud S = V[ind]

2.2. Deep-generative Models

For learning the representations from the sampled data, we proposed three deep-neural networks based on point cloud autoencoders.

2.2.1. 3D Point Cloud Autoencoder (PC-AE)

The architecture (Figure 5) is based on the autoencoder proposed in [8], which we utilized in design optimization problems in [5], [7], [9] and reported in our previous ECOLE Deliverable 1.2 [4]. The network comprises an encoder that compresses the input data into a latent space Z, and a decoder, which generates 3D point clouds from the representations in the latent space. The encoder

comprises five 1D convolutional layers, followed by a max-pooling operator, which yields the latent representations. The decoder comprises three fully connected layers and outputs the Cartesian coordinates of the points. All layers are activated with rectified linear units (ReLU), apart from the last convolutional layer, which is activated with a hyperbolic tangent function, and the output layer, which is activated with a sigmoid function. Hence, the latent variables are bound to the space $[-1,1]^{Lz}$, where L_Z is the dimensionality of the latent space, and the coordinates of the output point to the space $[0,1]^3$.

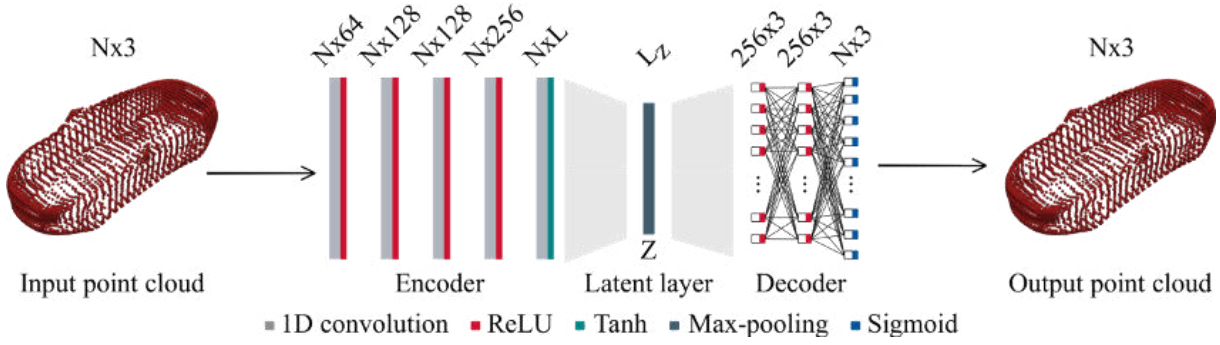

Figure 5 – Architecture of the 3D point cloud autoencoder.

We train our PC-AE by optimizing the network weights to minimize the shape reconstruction losses. In [8], [10], the authors utilize the Chamfer Distance (CD) [11] as loss function to overcome the lack of correspondence between points on different point clouds. Since shrink-wrapping organizes the data for training the autoencoder, it allows us to utilize the mean-squared distance (MSD) between input and predicted shapes (S_i and $\tilde{S}_i$, respectively) as loss function (Eq. 4), which is computationally more efficient and generates smoother shapes [7], [5].

$$MSD(S_i, \tilde{S}_i) = \frac{1}{N}\sum_{j=1}^{N}\left\|x_j - \tilde{x}_j\right\|^2 \tag{4}$$

2.2.2. 3D Point Cloud Variational Autoencoder (PC-VAE)

The architecture of the variational autoencoder (Figure 7) is analogous to the PC-AE architecture. The major difference is that, after the max-pooling, the latent layer is divided into two parts: a mean vector μ and a standard deviation vector σ with a sigmoid activation function. Hence, the latent representation z is stochastic in nature as it is sampled from the encoder's output distribution (μ, σ), from which the decoder generates a 3D point cloud. For training our PC-VAE, we minimize a loss function with two terms (Eq. 5): The first term is defined by the difference between the input and reconstructed point cloud, and the second is the Kullback-Leibler (KL)-divergence loss (D_{kl}) between the learned latent distribution and an assumed prior normal distribution. In [12], we utilize the Chamfer Distance (CD) to measure the reconstruction loss L_{recon}, where S_i and $\tilde{S}_i$ are the input and reconstructed output point clouds. The hyper-parameters α_1 and α_2 to balance the reconstruction and KL-divergence for PC-VAE-CD were tuned by grid search to α_1=250 and α_2=0.001.

$$L_{VAE}(S_i, \tilde{S}_i) = \alpha_1 L_{recon} + \alpha_2 D_{kl} \tag{5}$$

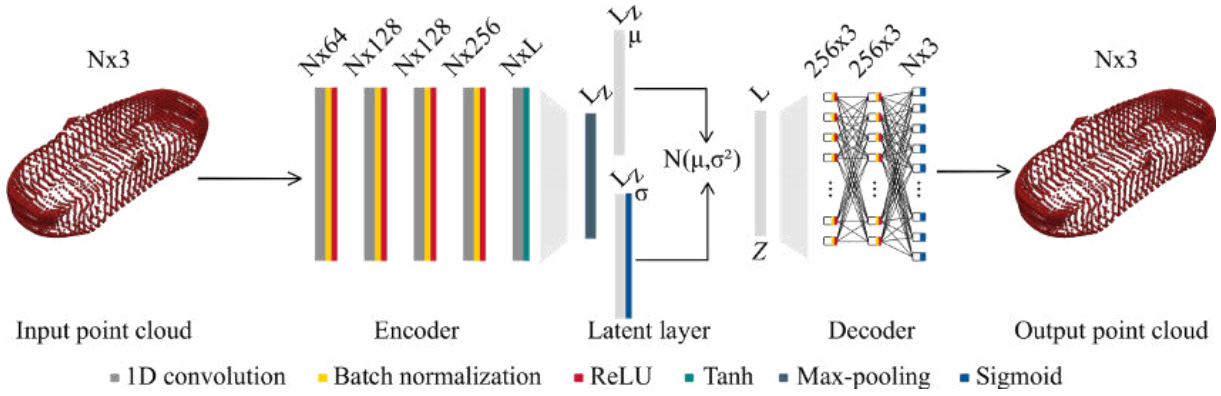

Figure 6 – Architecture of the 3D point cloud variational autoencoder.

However, since the shrink-wrapping algorithm organizes the input data as explained above, in [13], we trained the PC-VAE with MSD (Eq. 4) as reconstruction loss function. Hence, we optimized the hyper-parameters α_1 and α_2 for PC-VAE-MSD and selected the values of α_1=1000 and α_2=0.001 for the hyper-parameters as they provide an acceptable trade-off between reconstruction accuracy and divergence in the latent space.

Both, PC-AE and PC-VAE result in 3D point cloud representations, which have shown promising results in shape-generative tasks [8], [9], [12]. However, engineering simulation algorithms often require 3D shapes represented as polygonal meshes, e.g., for computing car aerodynamic performance or structural stiffness. Since automated meshing on 3D point cloud data is still challenging, we have proposed a novel deep-learning architecture, which learns to compress 3D point clouds into latent representations but generates polygonal meshes based on a set of simulation-ready mesh templates.

2.2.3. Point2FFD: A Novel Deformation-based Deep-generative Model

Mesh reconstruction on 3D point clouds is an ill-posed problem and a well-known challenge in computer graphics. Current mesh reconstruction techniques based on 3D point cloud data still require manual tuning and verification, which is undesired in automated shape-generative pipelines, *e.g.,* shape optimization problems. In the literature, deep-generative networks address this challenge by learning to triangulate points [14], by recursive mesh deformations [15], or even by learning on different types of input data, such as 2D images [16] and segmented 3D shapes [17]. However, these models still require manual work for fine tuning post-processed models by augmenting the training data and handling the shape-generative process as an isolated task.

We have proposed *Point2FFD* to address these challenges and, more specifically, the state-of-readiness of the models generated by point cloud autoencoders. The network builds upon the PC-AE architecture and comprises two parts: An encoder and a shape-generative model (Figure 7). The encoder follows the architecture of the PC-AE and compresses the point cloud data to a low-dimensional latent space. From the latent representation, the network generates shapes in two steps. First, the network selects a mesh template from a batch of available simulation-ready models parameterized with free-form deformation (FFD) based on the similarity of the template to the design represented in the latent space. Second, the network predicts the deformation of the FFD

lattice and applies the deformation to the selected template. Thus, the network generates meshes suitable for simulations, as with FFD, but also with different topologies, as obtained with PC-AE.

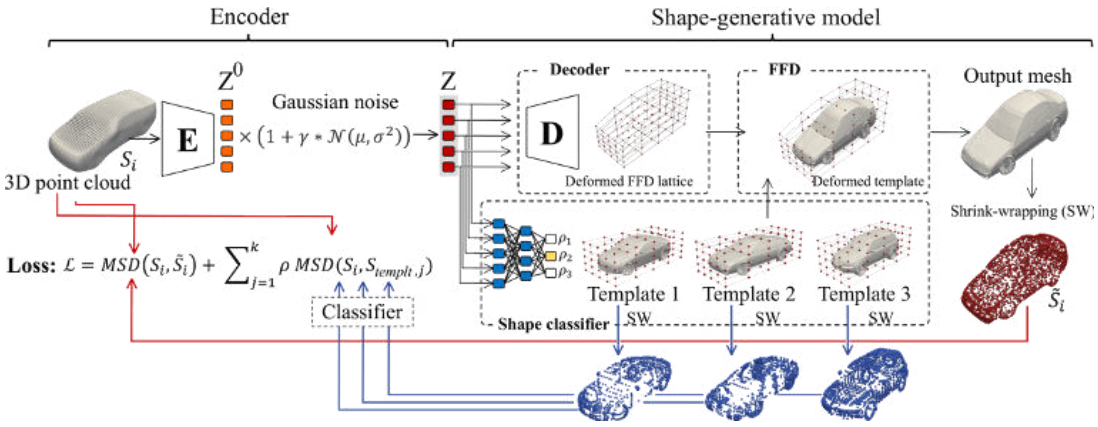

Figure 7 – Point2FFD architecture and flow of information for training the network.

The architecture of the encoder is identical to the encoder of the PC-AE. However, the network generates the latent representation Z by adding Gaussian noise to the output of the max-pooling operator Z^0 (Eq. (6). Our motivation is to increase the robustness of the shape-generative model without adding the Kullback-Leibler (KL) divergence to the loss function, as required for the PC-VAE. Hence, for training the model, we set $\gamma = 1$ and $\sigma^2 > 0$, where the standard deviation σ^2 depends on the desired level of noise.

$$Z = Z^0[1 + \gamma N(\mu, \sigma^2)] \tag{6}$$

Following the latent space, the network generates 3D shapes by selecting and deforming simulation-ready template models. We define a *template* as a 3D polygonal mesh, without artifacts and parameterized with standard FFD [18] (Figure 8). Mathematically, the templates are represented as pairs (B, V), where B is the matrix with the coefficients of the Bernstein tri-variate polynomials and V is the set of non-deformed control points. The coefficients $b_{a,p}$ in B are defined by the coordinates (x_a, y_a, z_a) of the mesh vertices and position $p = (i, j, k)$ of the control points in the lattice, which is assumed to have *l, m, n* control planes in *x*-, *y*- and *z*-direction, respectively. Hence, given a deformation ΔV of the control points, we obtain the deformed mesh nodes S_{def} through $S_{def} = B \times (V + \Delta V)$.

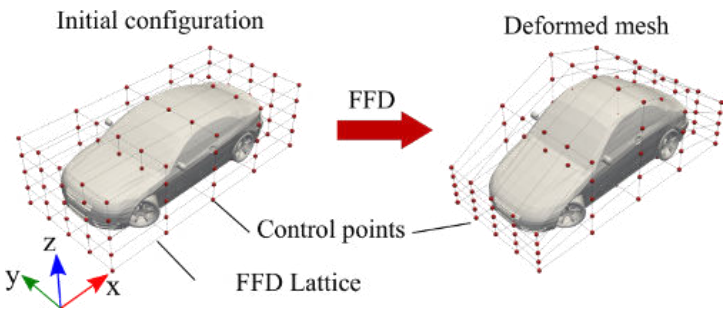

Figure 8 – Example of a polygonal mesh of a car shape parameterized and deformed using FFD.

For selecting the templates, we utilize a multi-layer perceptron (MLP) in one branch of the shape-generative model. The MLP comprises three layers, where the first two are activated by ReLU and the third by a *softmax* function, which outputs the probability of selecting each of the available templates. For the deformation, in a second branch, a decoder with three fully connected layers, where only the first two are activated by ReLU functions, predicts the displacement ΔV of the control points of the template FFD lattice. Then, in a last step, the network generates the output shape by deforming the selected template based on the predicted displacements using FFD.

The loss function for training Point2FFD comprises two terms (Eq. 7). The first term indicates the reconstruction losses between the input shape S_i and the reconstructed shape $\tilde{S}_i$, which we calculate using the mean-squared distance (MSD) between corresponding points $(x_j, \tilde{x}_j)$ (Eq. 4). Since the similarity between geometries improve the quality of the shape matching with FFD [19], we train the MLP-classifier to select the templates with highest similarity to the corresponding input shapes. The second term (MSD_{templt}) is the average of the MSD between the input shape S_i and each template $S_{templt,j}, j = 1 \dots k$ weighted by the corresponding selection probabilities $\rho_j \in [0,1]$ of each prototype. Thus, the second term is minimized by increasing the probability ρ for the prototypes that yield the lowest MSD values (highest similarity).

$$\mathcal{L} = MSD\left(S_i, \tilde{S}_i\right) + 1/k \sum_{j=1}^{k} \rho_j MSD(S_i, S_{templt,j}) \tag{7}$$

2.2.4. Benchmark analysis

We benchmarked our proposed networks against the point cloud autoencoder introduced in [8] named here PC-AE-Achlioptas, which we utilized as reference to design our architectures. We set up the networks considering similar hyperparameters and loss functions as reported in [8], [7], [4] (Table A 1, Appendix A). Particularly, for the PC-VAE, we set the weights α_1 =250 and α_2 =1.0E-03 to balance the Chamfer Distance (CD) [11] and KL-divergence in the loss function.

As benchmark scenarios, we considered four data sets that combine the car and airplane classes of ShapeNetCore [2] (Table 1). We sampled the shapes using the shrink-wrapping algorithm and we considered two point cloud sizes. Hence, the differences in performance from *Scenario A* to *Scenario B* allow us to assess the scalability of the networks. Also, by learning on data sets with multiple classes (*Scenario D*), we assess the performance of the MLP classifier and the capability of the networks to learn fundamentally different geometric features. Since Point2FFD is the only model that requires mesh templates, we utilized one template per object class and selected the shapes with highest similarity to the mean shape of each class (Figure 9). We parameterized the templates using lattices with same configuration: *l*=16, *m*=6 and *n*=6. Regarding the Gaussian noise, we set the standard deviation to σ^2 = 9.70E-03 to ensure 2.5% of noise with 99% confidence interval.

Table 1 – Scenarios considered for the benchmark analysis of the proposed deep-generative models. In the table, K corresponds to the number of utilized templates.

Scenario	Classes	Point cloud size	K
A	Car	6146	1
B	Car	24578	1
C	Airplane	24578	1
D	Car & Airplane	24578	2

We optimized the networks' weights using the Adam optimizer [20] for a maximum of 750 epochs. We set the algorithm learning rate to η=5.0E-04, and the moments to $\beta_1 = 0.900$ and $\beta_2 = 0.999$. Furthermore, we randomly divided the data set into 90%/10% partitions for training and testing the networks, respectively, and fed the networks gradually with batches of 50 shapes. We trained the networks on *Scenario A* on a machine with two CPUs Intel® Xeon® clocked at 2.10 GHz and four GPUs Nvidia® GeForce® RTX2080 Ti with 12 GB each. The experiments for the remaining scenarios were performed on a machine with similar CPU and two GPUs Nvidia® Quadro® RTX8000 with 48GB each. Each network was trained on a single GPU and with the machines under similar computational load.

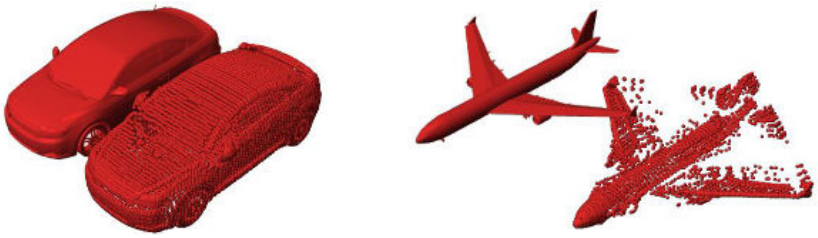

Figure 9 – Mesh and corresponding point clouds of the shapes utilized as templates for training our Point2FFD architecture.

We compared the networks based on four criteria: Number of variables, training runtime, and the reconstruction losses (CD) on training and test sets (Table 2). The number of variables and runtime indicate the computational effort to learn the models. Here, we observe some of the advantages of our proposed network. First, comparing the runtime and number of variables required by PC-AE and PC-AE-Achlioptas, we conclude that MSD is computationally more efficient than the CD, used for the PC-AE-Achlioptas. Second, the number of free (trainable) parameters of Point2FFD is invariant to the point cloud size. However, the total size of the network depends on the size of the point clouds and number of templates. Since the runtime obtained with Point2FFD and PC-AE-Achlioptas are comparable, we conclude that our proposed network efficiently processes higher-dimensional point clouds, even though it allocates more memory. Finally, we also notice that PC-VAE required longer runtime than the other networks, which we expected since PC-VAE processes 3D point clouds with more operations.

Table 2 - Results of the benchmark analysis performed on the four scenarios and selected networks.

Scenario A: Car class, point cloud size: 6146 points				
Network	PC-AE-Achlioptas	PC-AE	PC-VAE	Point2FFD
Loss function	CD	MSD	$\alpha_1 CD + \alpha_2 MSD$	$MSD_S + MSD_{templt}$
Free variables	1.83E+06	1.83E+06	1.89E+06	**3.92E+05**
Total variables	**1.83E+06**	**1.83E+06**	1.89E+06	3.93E+06
CD training	(1.34 ± 0.03)E-04	(9.37 ± 0.57)E-05	(5.59 ± 0.82)E-04	**(5.67 ± 0.39)E-05**
CD test	(1.34 ± 0.09)E-04	(8.93 ± 1.41)E-05	(5.03 ± 2.62)E-04	**(5.52 ± 1.07)E-05**
Runtime	1 h 45 min 19 s	**1 h 42 min 24 s**	2 h 34 min 2 s	2 h 7 min 21 s
Scenario B: Car class, point cloud size: 24578 points				
Free variables	6.60E+06	6.60E+06	6.67E+06	**3.93E+05**
Total variables	**6.60E+06**	**6.60E+06**	6.67E+06	1.45E+07
CD training	(1.00 ± 0.03)E-04	(1.05 ± 0.06)E-04	(4.10 ± 0.79)E-04	**(4.04 ± 0.09)E-05**
CD test	(1.05 ± 0.11)E-04	(1.03 ± 0.17)E-04	(4.99 ± 3.03)E-04	**(4.95 ± 0.99)E-05**
Runtime	4 h 32 min 40 s	**3 h 42 min 13 s**	6 h 39 min 58 s	4 h 43 min 23 s
Scenario C: Airplane class, point cloud size: 24578 points				
CD training	**(8.90 ± 0.27)E-05**	(1.47 ± 0.06)E-04	(4.58 ± 0.28)E-04	(1.69 ± 0.06)E-04
CD test	**(9.61 ± 1.05)E-05**	(1.48 ± 0.19)E-04	(4.62 ± 0.79)E-04	(1.80 ± 0.24)E-04
Runtime	4 h 29 min 17 s	**3 h 42 min 22 s**	6 h 31 min 50 s	4 h 35 min 39 s
Scenario D: Car and airplane classes, point cloud size: 24578 points				
Total variables	**6.60E+06**	**6.60E+06**	6.67E+06	2.87E+07
CD training	**(1.34 ± 0.05)E-04**	**(1.43 ± 0.06)E-04**	(5.48 ± 0.52)E-04	**(1.36 ± 0.11)E-04**
CD test	**(1.31 ± 0.10)E-04**	**(1.44 ± 0.18)E-04**	(4.26 ± 1.08)E-04	**(1.43 ± 0.35)E-04**
Runtime	4 h 31 min 5 s	**3 h 43 min 3s**	6 h 38 min 27 s	4 h 39 min 53 s

The reconstruction losses indicate the shape-generative capability of the networks. In most of the scenarios, Point2FFD achieved at least comparable results to the other models. Particularly, in *Scenario C*, the PC-AE-Achlioptas yielded the lowest reconstruction losses. To verify the results, we visually inspected the reconstruction of shapes randomly sampled from the data set (Figure 10). We observed in the reconstructions that the networks trained with MSD generate smoother shapes, however, with less topological differentiation. Yet, since Point2FFD generate shapes based on prototypes, the shapes of the airplanes obtained with Point2FFD are more realistic than obtained with PC-AE. Also, despite the higher CD values, PC-VAE distinguished the topology of the shapes more clearly and yielded point clouds with less noise than the PC-AE-Achlioptas. Therefore, we conclude that the networks proposed in the framework of ECOLE improved the shape-generative capabilities of the baseline model (PC-AE-Achlioptas), which was used as reference for our research.

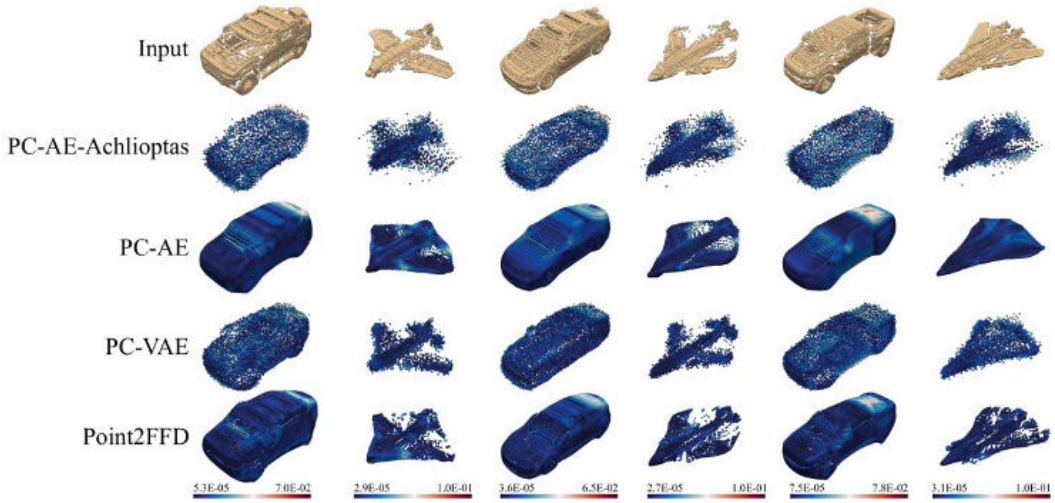

Figure 10 – Visualization of shapes randomly selected from the data set and the corresponding reconstructions yielded by the benchmarked networks.

2.3.Feature Visualization

Our feature visualization technique [10] projects the activations obtained at the layers of the encoder back onto the corresponding points in the 3D point cloud representation as color maps. Since the latent variables are obtained through a max-pooling operator, the visualization of the features obtained in the layer immediately prior to the latent space reveals the geometric characteristics encoded in the latent variables. As reported in [10], [4], the latent variables indicate the occupancy of specific regions in the input Cartesian space, which optimally describe the variations between the shapes. Since the visualization technique assumes an encoder with only 1D convolutions (point-wise operators), the same technique is also suitable for the PC-VAE and Point2FFD networks.

Although unexplored in [10], the feature visualization also allows us to select sub-sets of latent variables to constrain shape modifications to specific regions of interest in the shapes. In [7], we utilize the feature visualization scheme to select and transfer features between latent representations in order to generate crossover shapes, *e.g.*, by transferring a spoiler from a sports car to a sedan car (Figure 11). Furthermore, in a multi-task design optimization problem, we also show that by enabling the transfer of the same features in the representations of designs assigned to different tasks, the optimizer fosters commonality in the optimized shapes, which also indicates that the representation learned by the PC-AE is suitable for transferring knowledge between different optimization problems.

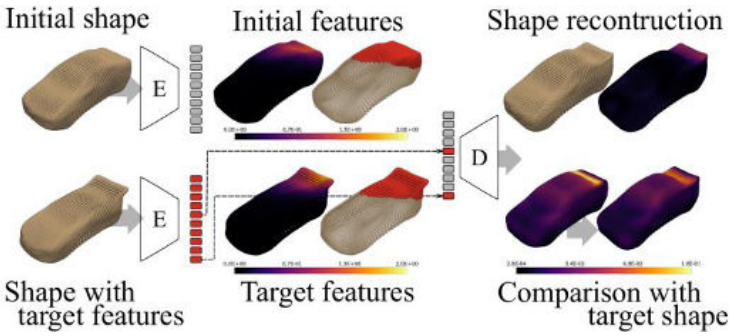

Figure 11 – Example of crossover car shapes generated by transferring features between different latent representations.

3. Applications of Deep-generative Models in Engineering Optimization

In the previous section, we reviewed the methods utilized to process geometric data by training shape-generative models and visualizing the learned features. In this section, we demonstrate the application of the proposed models in real world-inspired surrogate modelling and design optimization problems. For four use cases, we demonstrate the applicability of our (V)AEs (1) for surrogate modelling of performance data, which is important for reducing the runtime of real-world applications, such as vehicle design optimization, (2) for improving multi-objective optimization frameworks by exploiting the interpolation in the latent space of the VAEs, (3) for improving multi-task optimization with online exploitation of experience, and (4) for generating simulation-ready triangulated meshes for improved vehicle design optimization.

3.1. Surrogate Models for Optimization

In early stages of automotive development, engineers estimate the performance of a design based on computer simulations, *e.g.*, calculating aerodynamic forces using computational fluid dynamics (CFD) simulations. However, in order to achieve a target level of accuracy, these simulations require considerable computational effort, which limits the freedom of designers in design exploration tasks. To replace expensive function evaluations, it is a common approach to utilize a surrogate model to predict the performance of the designs based on the values of the design variables. Hence, our objective is to utilize the latent space of our proposed autoencoders (PC-AE and PC-VAE) to learn performance metrics calculated on designs utilized to train the architectures.

Previous work on autoencoder-based surrogate models in the chemistry and biology domains showed promising results in assessing the characteristics of chemical compounds before committing to expensive synthesis processes [21]. However, generating surrogate models for automotive design is a more challenging task due to various reasons foremost due to the limited availability of training data. Furthermore, since the proposed deep-generative models learn exclusively on geometric data, the training algorithm neglects any explicit information related to the performance of the designs. Therefore, the relation between the learned latent variables and design performance is also unknown *a priori*, thus configuring the surrogate models to map highly nonlinear data, such as aerodynamic performance, becomes also very challenging. Hence, in our research in [13], we utilize information-theoretic measures to quantify the information stored in

the latent space of different configurations of the PC-AE and PC-VAE with respect to the volume V and aerodynamic drag force F_x of the shapes. Furthermore, we exploit the generative capability of the PC-VAE to augment the training data for a surrogate model to predict both V and F_x (Figure 12). We selected the volume due to the low computational cost and due to the similarity to the interpretation of the latent variables, and the aerodynamic drag, on which we have higher interest due to the capability of the surrogate model to accelerate time-consuming aerodynamic design optimization problems.

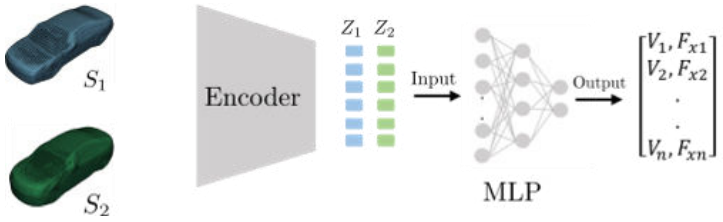

Figure 12 – Schematic of the data processing for predicting the performance of 3D designs based on the latent variables and using a multi-layer perceptron (MLP).

3.1.1. Information-theoretic Assessment of the Latent Space

The information-based analysis of the latent representations indicates the expected quality of surrogate models trained on the latent space to map specific target metrics. In our study, we compared the latent representation based on the mutual information (MI), which indicates the total information stored by a random variable x about another random variable y, regardless of the degree of nonlinearity of the relation between x and y. Therefore, the higher the MI, the higher is the expected accuracy of the surrogate model trained to predict y based on x.

For the analysis of the latent space, we considered the PC-AE and PC-VAE trained with both, Chamfer Distance CD (PC-AE-CD and PC-VAE-CD) and Mean Squared Distance MSD (PC-AE-MSD and PC-VAE-MSD). As data set, we utilized the car class from ShapeNetCore [2], sampled with shrink-wrapping into point clouds with 24578 points. We trained the models with the Adam optimizer [20] and the same settings as discussed in the benchmark analysis (Section 2.2.4). We calculated the volume and aerodynamic drag force using the shrink-wrapped meshes and CFD simulations using the open source solver OpenFOAM® (www.openfoam.com). From the 3500 shapes, the simulations of 600 samples converged to an expected range of forces, which we used for the MI analyses and training the surrogate model.

By analyzing the mutual information (Figure 13), we observe two main differences between the models. First, the PC-AE architectures yielded a more even distribution of the MI among the latent variables compared to the PC-VAE. Second, the architectures trained with MSD yielded higher values of MI, indicating that the loss function also has an influence on the information in the latent space. Our main conclusion from the experiment was that the regularization reduces the information content in the latent space since it adds a stochastic component to the latent space. On the other hand, MSD increases the MI values by forcing the autoencoders to implicitly learn global shape information, which is defined by the organization of the point clouds.

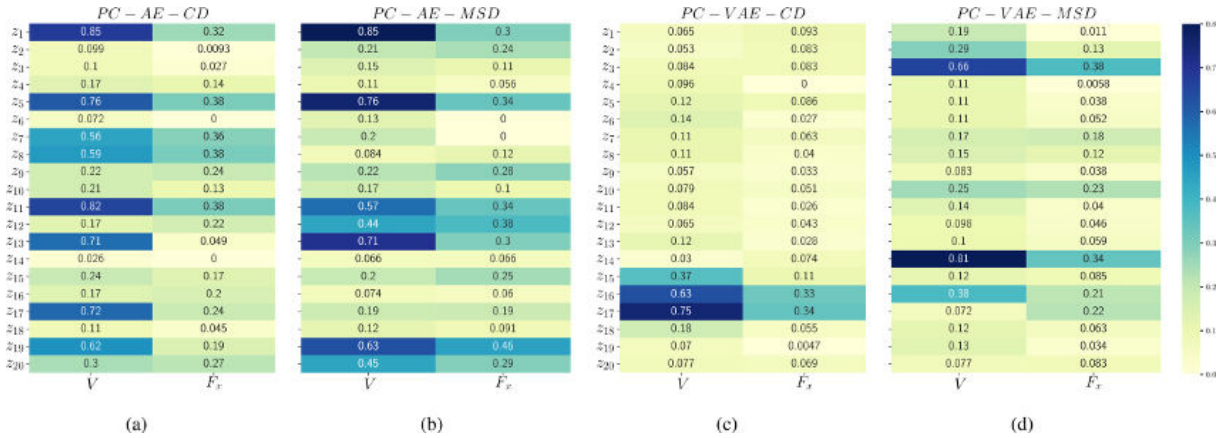

Figure 13 – Mutual information measured based on the latent variables learned with variations of the PC-AE and PC-VAE. The variables V *and* F_x *indicate volume and aerodynamic drag, respectively.*

3.1.2. Generating Surrogate Models for Vehicle Aerodynamics

Based on the information-theoretic analysis, we generated surrogate models using the latent spaces learned by the architectures trained with MSD, which preserve more information in the latent space. As surrogate model, we utilized a two-output multi-layer perceptron (MLP) to predict simultaneously the volume and aerodynamic drag of the car shapes. The MLP comprises a single hidden layer with 10 hidden neurons, which are activated by ReLU and defined based on a five-fold cross-validation test, and an output layer with two neurons activated by sigmoid functions.

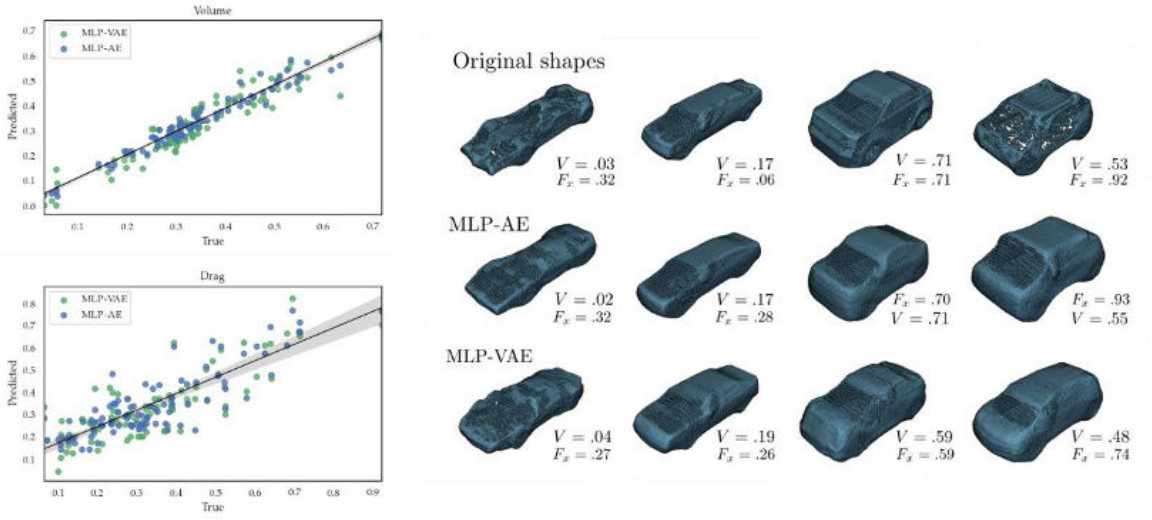

Figure 14 – True vs prediction values of the MLPs trained on the latent space of PC-AE and PC-VAE.

In a first stage, we train the MLP using the Adam optimizer [20] with a learning ratio η=1E-03 for a maximum of 1000 epochs. Also, we divide the data set into 85%/15% partitions for training and testing, respectively, which we feed to the networks in batches of 50 samples. By analyzing the prediction performance (Figure 14), we observe that the MLPs predicted the shape volume with more accuracy than the aerodynamic drag, which is expected since the latent variables in both, PC-AE and PC-VAE, hold more information about the volume as geometric descriptor. For the

aerodynamic drag, the MLP trained on the PC-AE latent space (MLP-AE) achieved accuracies approximately 13% higher (RMSE = 0.048), but both models present disperse predictions, in particular outside the range [0.2, 0.6], which would negatively impact the performance of downstream tasks based on the surrogate model.

In order to overcome the limitation of data in the range with the poorest performance, we utilized the shape-generative capabilities of the PC-VAE to augment the training data. As the PC-VAE learns a shape-generative model that smoothly interpolate and combine design features [12], it allows us to sample the latent space in the sparser regions of the data set and generate new realistic car shapes to augment the data. This generative process is more efficient than manually modifying the 3D meshes and exploits the knowledge stored in the latent space of the PC-VAE, which is one of the main advantages of our proposed architecture. Hence, for the purposes of our experiments, we augmented the data with additional 300 samples, for which we calculated the latent representations with both autoencoders, as well as shape volume and aerodynamic drag.

For assessing the improvement of the MLP accuracy due to the data augmentation, we re-trained the MLPs considering data set increments of 20% (Table 3). Based on the root mean squared error (RMSE), we observe that a data augmentation up to 60% was beneficial for both models. For values higher than 60%, we concluded that the augmentation introduced redundancies in the data, which increased the difficulty for learning the aerodynamic performance of the designs.

Table 3 – Root mean squared error (RMSE) obtained for the MLPs trained on the latent spaces of the PC-AE (MLP-AE) and PC-VAE (MLP-VAE) considering different ratios of data augmentation.

Ratio of augmented data	MLP-AE (RMSE)	MLP-VAE (RMSE)
0	0.0480 ± 0.0006	0.0550 ± 0.0023
0.2	**0.0454 ± 0.0011**	**0.0488 ± 0.0014**
0.4	**0.0447 ± 0.0011**	**0.0492 ± 0.0015**
0.6	**0.0452 ± 0.0013**	**0.0491 ± 0.0011**
0.8	0.0474 ± 0.0012	0.0557 ± 0.0018
1	0.0492 ± 0.0017	0.0570 ± 0.0020

3.2. Multi-objective Design Optimization

Although meta-models accelerate the calculation to estimate the design performance in multiple domains, the engineers still have to handle the results to take multi-criteria decisions (MCDM), such as balancing between structural efficiency, aesthetics, and manufacturing costs. Thus, providing an efficient approach to optimize designs considering design preferences in MCDM potentially leads to higher chances of design innovation at early product development stages. Therefore, in [12], we formulated a real-world inspired design optimization scenarios with and without prior preference knowledge and explored the feasibility of the learned latent representation of our PC-VAE (Section 2.2.2) as a geometric representation for multi-objective optimization.

In our experiments, we formulate two scenarios of multi-objective 3D target shape matching optimization problems. Hence, we select target shapes that address specific design problems, such as aesthetics (S_1) and aerodynamic performance (S_2). In our first scenario (in Figure 15.a), we assume that the designer aims to generate a wide range of designs, as in an initial design exploration phase in product development. In the second scenario (in Figure 15.b), we assume the designer/decision maker (DM) aims to generate a solution of interest such that the shape is an intermediate between the two reference shapes (S_1 and S_2).

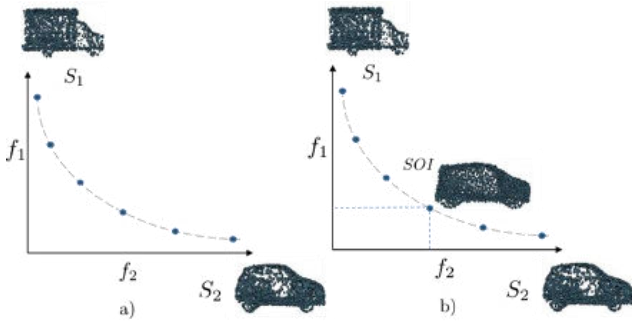

Figure 15 – Two scenarios for multi-criteria decision analysis. S1 and S2 are two reference shapes for the optimization task.

Hence, we formulate our multi-objective design problem as:

$$\min_{x \in D} f_1(z),\ f_2(z), \qquad z \sim S \tag{8}$$

$$f_i(S, S_i) = \sum_{x \in S} \min_{x_i \in S_i} \|x - x_i\|_2^2 + \sum_{x_i \in S_i} \min_{x \in S} \|x_i - x\|_2^2 \tag{9}$$

where f_1, f_2 are the two objective functions evaluated using Chamfer Distance (CD) and z is an n-dimensional latent vector of our PC-VAE, where each parameter is a decision variable in the optimization for generating the currently deformed shape S, and which are modified by the optimization algorithm. S_i represents the two reference shapes S_1 and S_2 and x, x_i are points in the currently deformed shape S and the reference shapes respectively. The objective functions of the optimization receive an n-dimensional latent vector of each 3D shape from the latent space of our trained PC-VAE. Then, the decoder of the PC-VAE converts the latent vector to the 3D domain where the objective function computes the difference between the reference shapes S_i to the currently deformed shape S using CD.

For the second scenario in Figure 12.b), in order to generate a single solution point that reflects preferences, the DM needs to specify comparative preference states for each objective function value. However, since our objective functions are distance functions, we assume the DM can specify the desired distance values $d_1 and\ d_2$ for each objective. Thus, the DM's preference can be defined as a distance ratio (α) function, where $\alpha = \frac{d_1}{d_2}$.

3.2.1. Generating diverse design solutions (Scenario 1)

For scenario 1 (Figure 12.a)), to generate a diverse range of design solutions between shape S_1 and S_2, we utilize a multi-objective evolutionary algorithm (MOEA) to solve our multi-objective

optimization problem. The MOEAs for solving multi-objective optimizations are generally initialized with a random initial population, which may lead to infeasible solutions. Thus, to improve the convergence of MOEAs, we propose an approach that generates a target-oriented initial population (Figure 16). We first decompose our multi-objective optimization problem into 2 single-objective sub-problems. Every single optimization is performed using the covariance matrix adaptation evolution strategy (CMA-ES) method [22], which we selected due to its suitability for small populations, high convergence ratio and a low number of hyper-parameters. Then, we interpolate between the optimized solution of each problem, which are represented in the latent space, and enumerate the interpolated samples based on the contribution of each solution. The latent vectors generated through interpolation (Lerp-seed) are used for the initialization of the initial population of an MOEA.

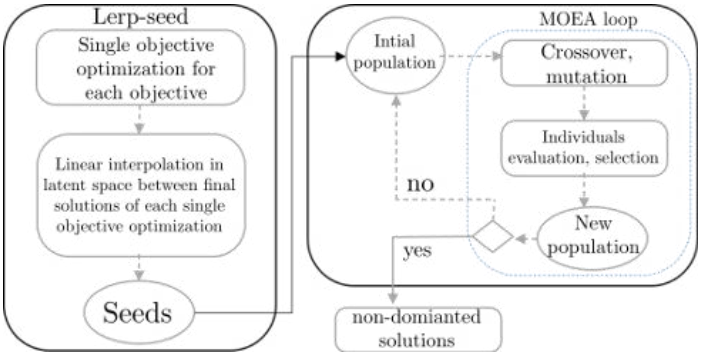

Figure 16 – Proposed seeding mechanism in an MOEA.

We applied our proposed seeding strategy in a vehicle multi-objective design optimization problem. For the individual optimizations with CMA-ES, we set the initial step size to 0.01 and the maximum number of generations to 100. As seed population to the MOEA, we generated 100 designs by interpolating the optimized solutions in the latent space. We analyzed the effect of our seeding strategy on a multi-objective particle swarm optimization algorithm (OMOPSO) and compared our results to analogous cases performed with no seed MOEAs (NSGA-II and OMOPSO with random initialization). To compare the performance of the algorithms, we utilized the following metrics: hypervolume coverage (HV), inverted generational distance (IGD), spacing (SP) and number of function evaluation (NFEs) (Table 4).

Table 4 – Performance measures of the optimization result for NSGA-II, OMOPSO, and Lerp-seed OMOPSO.

Methods	NFEs	HV (mean)	SP (mean)	IGD (mean)
NSGA-II	10000	0.473	0.186	0.06
OMOPSO	10000	0.484	0.229	0.059
Lerp-seed OMOPSO	10000	**0.49**	0.211	**0.055**

Considering that an MOEA that yields a set of solutions with high HV and low IGD values provides better distribution of the generated non-dominated solutions, we observe that the *Lerp-seed* OMOPSO achieves the best results with respect to mean HV and IGD values for our design optimization task. Furthermore, by comparing the values of HV and IGD over the generations after

the CMA-ES initialization for 30 different runs (Figure 17), we conclude that the OMOPSO with *Lerp-seed* for initial population construction outperformed a random based approach on average and, thus, that spending some additional computational effort in constructing a target oriented initial population of an MOEA helps to improve the convergence in the optimization.

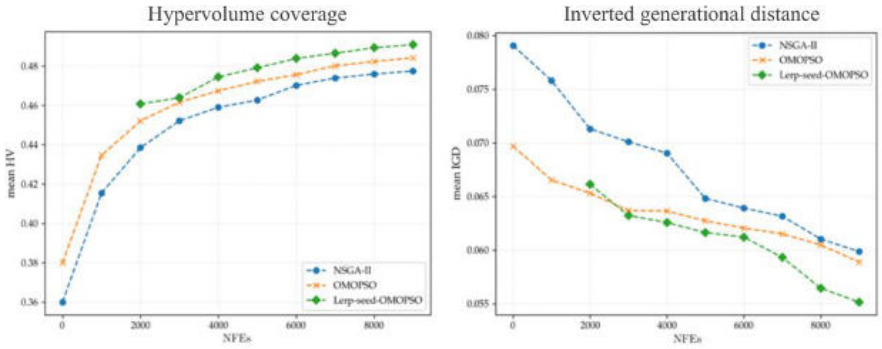

Figure 17 – Change of mean HV and IGD (over 10 runs) with respect to the number of function evaluations (NFEs).

3.2.2. Generating a solution of interest (Scenario 2)

For scenario 2, in order to generate a single solution that reflects the user preference (α), the decision maker (DM) needs to specify a reference value for each objective. Often this additional knowledge of the DM preference is integrated as an additional objective within the optimization algorithm. Here, we add the preference information to the MOP using the weighted sum method (WSM). The WSM combines the objectives of the MOP into a single-objective problem, which consists of the weighted sum of the objective functions (Eq. 10), where w_i is the weight assigned to the objective function f_i.

$$F(\vec{x}, w_1) = w_1 f_1(\vec{x}) + w_2 f_2(\vec{x}) \tag{10}$$

The weights are chosen between 0 and 1, and it is a common practice to choose weights in a way that their sum equals to 1, i.e., $w_2 = 1 - w_1$. Since the literature lacks work on adapting the weights in WSM to match the designer's preference, we propose a method in [12] to adapt the WSM weights in order to obtain optimized solutions that reflect the DM's design preference with a low computational cost.

For our experiments, we considered a two-objective design optimization problem and formulated according to the weighted sum approach. To search an optimal w_1 within a bound, we propose a two-step method (Figure 18) involving a global optimization of the weighted-sum function (Eq. 11) to minimize a cost function $C(\vec{x}, w_1)$.

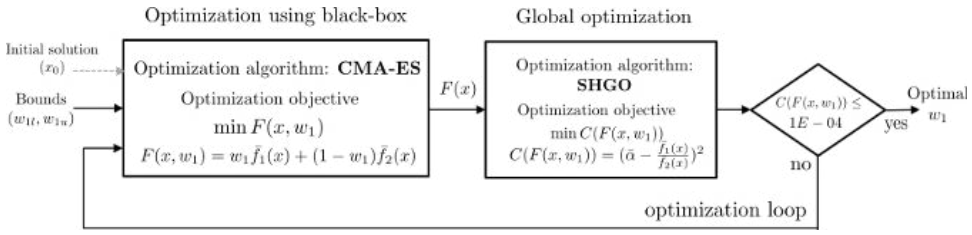

Figure 18 – Optimization strategy to determine weight w1 in a weighted -sum method to match the DM's design preference.

$$C(\vec{x}, w_1) = (\bar{\alpha} - \frac{f_1(\vec{x})}{f_2(\vec{x})})^2 \tag{11}$$

In the first step of our proposed strategy (Figure 18), we considered CMA-ES for our WSM. CMA-ES often requires providing a feasible solution as a starting point as the initial candidate solution. We utilized our proposed seeding strategy and selected a solution that is the closest to the preferred distance function $\overline{d_1}$ and $\overline{d_2}$ as the initial solution vector (x_o) for CMA-ES [22]. This way of selecting the initial solution vector targets to reduce the number of generations required for convergence to the optimum. In the second step, we utilize the simplicial homology global optimization (SHGO) algorithm, which is a promising, derivative-free global optimization algorithm, and it also returns all other local and global minima. The optimizer determines the global minimum of weight w_1 by minimizing the cost function in Eq. 11. The final optimal weight w_1 helps the DM to understand the relation between the weights in WSM and the final solution.

In our experiments, we considered three preference cases (α = 2, 1 and 0.5) in Scenario 2 (Figure 15) and utilized our proposed method for determining the weights to generate a final solution to match the DM's design preference. As baseline performance, we considered the results obtained with MOEA with the designer's preference added as a third objective function. From our experiments, we conclude that the quality of the results is better, if the generated solution is closer to the normalized preference ratio ($\bar{\alpha}$) with low number of function evaluations (NFEs) and lower quality. Therefore, we defined as quality metric the Euclidean distance between the generated sample and preferred solution ($\overline{d_1}$, $\overline{d_2}$). By comparing the approaches based on the quality of the results (Table 5), we observed that the optimization applying our method achieved designs that matched the designer's preference with less deviations.

Thus, through our research we confirmed the feasibility of the latent representation for multi-objective design optimization tasks. Through quantitative evaluations we demonstrate that our proposed strategy based on the utilization of the PC-VAE's latent space knowledge contributes to an improved convergence of multi-objective optimization algorithms in different scenarios of preference-driven optimization.

Table 5 – Quality of the final solution comparison between WSM method and 3-objective MOEA.

Methods	Metrics	Preference ratio $\alpha = 2$ $\bar{\alpha} = 7.94$	$\alpha = 1$ $\bar{\alpha} = 1.60$	$\alpha = 0.5$ $\bar{\alpha} = 0.56$
Search w_1 with global optimizer in WSM	NFEs	**4000**	**5000**	**5500**
	Normalized preference ($\bar{\alpha}$)	**7.96**	1.1	**0.48**
	Quality	**0.019**	0.085	**0.089**
3-objective MOEA	NFEs	10000	10000	10000
	Normalized preference ($\bar{\alpha}$)	6.3	1.1	0.56
	Quality	0.11	0.12	0.28

3.3. Multi-task Design Optimization using 3D Point Cloud Autoencoders

In the previous optimization scenarios, we explored the notion of experience given as information learned by the autoencoders from a data set of CAE models in an offline fashion. In multi-task optimization problems, the algorithms map multiple design representations into a single search space, where the solutions are combined throughout the generations and reassigned to different tasks (Figure 19). The underlying assumption is that the individuals that excel in a particular task potentially carry useful information to drive designs in synergetic tasks faster towards optimality. Therefore, the optimization algorithms that follow this paradigm rely on an online implicit knowledge transfer between sub-sets of the population that address different tasks.

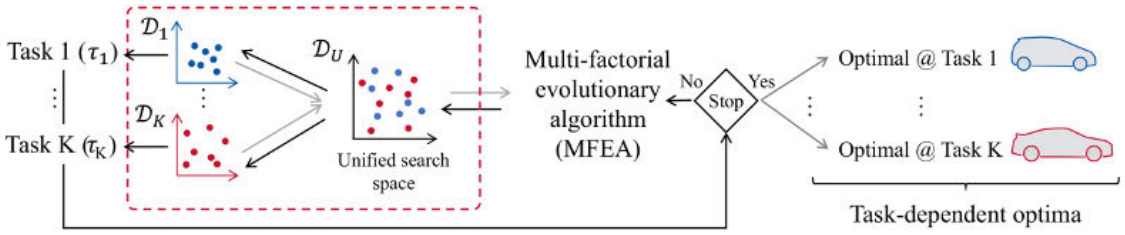

Figure 19 – Example of a generic multi-task design optimization framework.

However, mapping multiple design features without a clear semantic overlap to a single design space is a challenging task. Current solutions, *e.g.*, binary key encoding, address the problem technically, but generate representations that miss synergies between tasks and have lower scalability to higher dimensional spaces. We address this challenge by utilizing the latent space learned by our proposed 3D point cloud autoencoder (PC-AE) as unified search space for multi-task design optimization. Since the latent space is domain-agnostic, the representation allows us to combine geometries that are fundamentally different and address different tasks. Furthermore, the autoencoder learns the overlap between and mapping between different design features. Thus, our approach reduces (if not eliminates) the manual work to combine design features from different design spaces.

In [7], we propose a multi-task optimization framework with two steps (Figure 20): Pre-processing and multi-task optimization. In the pre-processing step, the objective is to learn the design representations and select the latent features that will perform as optimization degrees of freedom based on the feature visualization technique. In the multi-task phase, the optimization algorithm searches for solutions in the latent space, and we utilize the trained decoder as a shape-generative model, which maps back the solutions from the unified search space to the domain of each task. As a demonstration of our framework, we performed a multi-task vehicle aerodynamic optimization, where we minimize the aerodynamic drag of three car shapes. Furthermore, we exploit the transfer of features in the latent space to foster common underbody design in the optimized shapes, which would allow the vehicles to have a similar platform design. Platform designs refer in the automotive domain to having a portfolio of different car types, which have a high share of similar components and, thus, reduce the overall manufacturing costs.

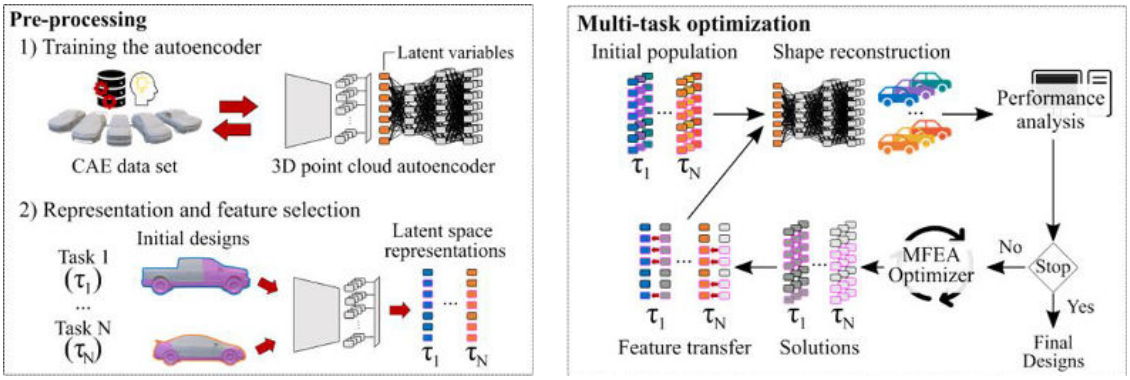

Figure 20 – Proposed multi-task framework for the optimization of 3D designs utilizing the latent space of the 3D point cloud autoencoder (PC-AE) as unified search space.

3.3.1. Pre-processing

For learning the design representations, we train a PC-AE on the shapes of the car class of ShapeNetCore [2], sampled with the shrink-wrapping method into 3D point clouds with 6146 points. We set the autoencoder hyperparameters as described in Table A 1 (Appendix A), but reduced the dimensionality of the latent space to 20 dimensions since interpreting the features of latent spaces with more than 20 dimensions is challenging [7]. Also, we utilize the same algorithm and corresponding settings to optimize the network weights as described for *Scenario A* of the benchmark analysis (Section 2.2.4). For the proposed settings, PC-AE yields a mean Chamfer Distance of CD = (3.03±0.07)E-04 on the test data, which is a higher value than in the benchmark experiment, as expected, since we reduced the dimensionality of the latent space.

Then, for a set of initial designs, we utilize the feature visualization to select the degrees of freedom for the design optimization. The motivation to utilize a sub-set of latent features is two-fold: First, we constrain the design modifications to areas of interest, *e.g.*, optimizing only the passenger cabin of a pick-up truck to keep the volume of transportable cargo (functionality) of the vehicle. Second, it allows the multi-task optimization to combine specific common features of the shapes and potentially generate optimized shapes with common design characteristics, such as the underbody shape. In our experiments [7], we utilized as initial designs a pick-up truck (S1), a sedan (S2) and a hatchback (S3). For each shape, we enabled different sets of degrees of freedom, which have as

intersection the features that map the underbody of the car shapes (Figure 21). Hence, in our aerodynamic optimization, we aim to improve the performance of the vehicles but generate shapes with similar underbody to simplify the underbody design and reduce manufacturing costs.

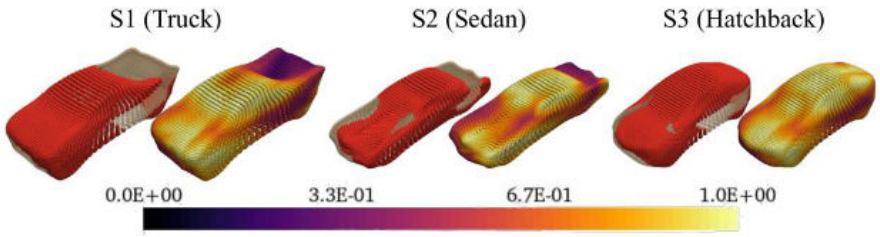

Figure 21 – Visualization of the regions mapped by the selected features Z' for each of the shapes (in red) and combined activations of the selected features (color maps). The brighter colors indicate higher values of combined activations.

3.3.2. Multi-factorial Evolutionary Algorithm

The algorithm implemented in our framework is an adaptation of the multi-factorial evolutionary algorithm (MFEA) proposed in [23]. The MFEA is a genetic algorithm where the individuals are labelled with a skill factor τ_i and a scalar fitness φ_i, which indicate the task on which the i-th individual excels the fitness of the individual compared to the remaining set of the population assigned to the same task. Hence, to generate an offspring population, the algorithm iteratively selects pairs of parents from the population with a probability based on the scalar fitness, combines the selected individuals through assortative mating and assigns the new skill factors using vertical cultural transmission.

In MFEA, the assortative mating and vertical cultural transmission are the mechanisms that allow to exchange knowledge between individuals assigned to different tasks (Figure 22). Given two parents (P_A, P_B) and a random mating probability rmp, the assortative mating generates two offspring designs through simulated binary crossover (SBX), if the parents have the same skill factor or $\varepsilon < rmp$, where ε is a random number sampled at every mating and $\varepsilon \in [0,1]$. Otherwise, the offspring designs are generated through mutation of the parent designs. Regardless of the case, the algorithm outputs the offspring design S^* and skill factors τ_P of the corresponding parents, which are forwarded to the vertical cultural transmission. Then, given S^* and τ_P, the vertical cultural transmission assigns the skill factor to S^* as the same as the parents, if the parents were assigned to the same task. Otherwise, the skill factor is assigned a random value.

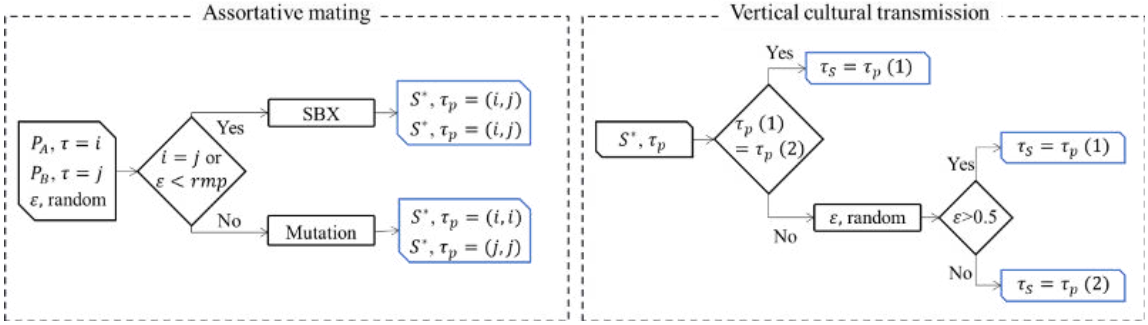

Figure 22 – Block diagram of the pseudo-code for the assortative mating and vertical cultural transmission algorithms.

Compared to the proposal in [23], our framework has two main differences. First, instead of initializing the population with random individuals, we generate λ individuals in K equally sized partitions by mutating the latent representation of the initial designs. Hence, we avoid the expensive evaluation of all individuals across all tasks since the skill factor is known beforehand. Second, after the vertical cultural transmission, we map the offspring back to the task-specific populations by transferring the features selected with the feature visualization to the initial design representations. The complete algorithm (Algorithm 2) is presented in Appendix B.

We verified the modified algorithm with a set of experiments using four standard 20D benchmark functions used in optimization algorithm research: Sphere, Rastrigin, Ackley and Rosenbrock defined in the interval $[-50,50]^{20}$. In terms of scenarios, we optimized the functions individually, in pairs, trios and all together, which covers all possible combinations for multi-task optimization with the proposed functions. Furthermore, we varied the *rmp* parameter between (0.0, 0.3, 0.5, 0.7, 1.0) to assess the effects of the knowledge transfer on performance of the optimizations. For each case (combination and *rmp* value), we performed 30 runs with different initializations. By analysing the results with respect to the function combinations and *rmp* (Figure 23), we conclude that the MFEA accelerated the optimizations in all cases, even though the quality of the results varied depending on the combination of functions. Furthermore, we obtain the highest convergence ratio by using intermediate values of *rmp* (0.3), which indicates that increasing the probability to combine individuals from different tasks also increases the chances of negative knowledge transfer, which slows down the optimization and lead to suboptimal results.

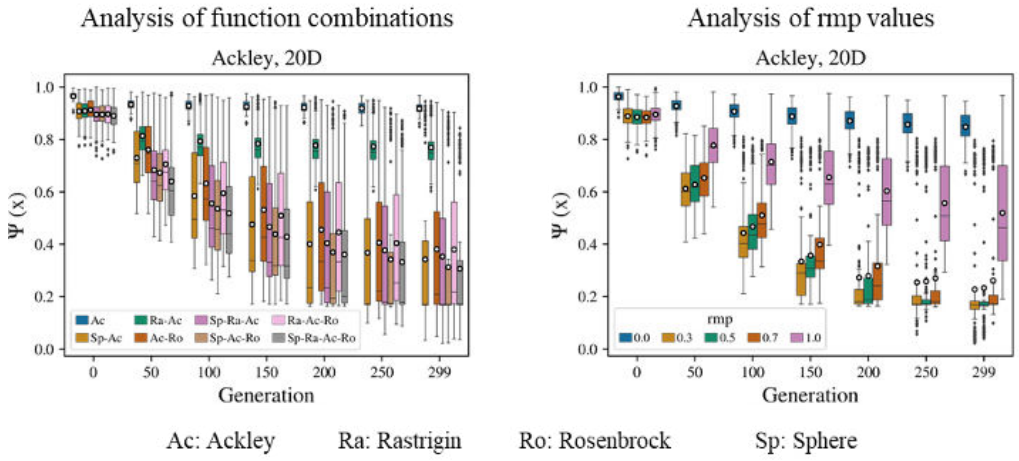

Figure 23 – Analysis of the mean factorial cost Ψ_i (function value) obtained for the Ackley function in all optimization cases over 30 runs.

Since the optimization results were in line with the work presented in [23], [24], we considered our algorithm successfully validated. Therefore, we applied our framework in a real world-inspired scenario of vehicle aerodynamic optimization, which we discuss in the following section.

3.3.3. Multi-task Vehicle Aerodynamic Optimization

Our aerodynamic optimization problem consists of minimizing the drag force of the three car shapes selected in the pre-processing phase (Section 3.3.1). We calculate the aerodynamic drag

with CFD simulations using OpenFOAM®, where we consider the vehicles driving in a straight line with a constant speed of U =110 km/h. The factorial cost of each task is defined as:

$$\Psi_j = F_x + \rho \mathrm{MSD}(S_j^*, S_{j,near}^*) \tag{12}$$

Where F_x is the aerodynamic drag, ρ a penalty factor, S_j^* the proposed solution and $S_{j,near}^*$ the most similar design in the autoencoder training data to the proposed solution. Hence, the penalty forces the MFEA to search for solutions near the learned region in the latent space, which increases the chances of generating realistic car shapes as optimum designs.

We considered as scenarios the optimization of the shapes in all possible pairs and the three shapes together, similar to the verification experiments. As baseline performance, we considered the optimization of the shapes with a genetic algorithm initialized with identical hyperparameters as in the MFEA. We set the population size to $\lambda = 15$, maximum number of generations to 20 and rmp to 0.3. We implemented our framework on a cluster setup to perform the CFD simulations in parallel with 16 processors each. The runtime of each batch of simulations took up to 75 minutes and a complete optimization 23h, on average.

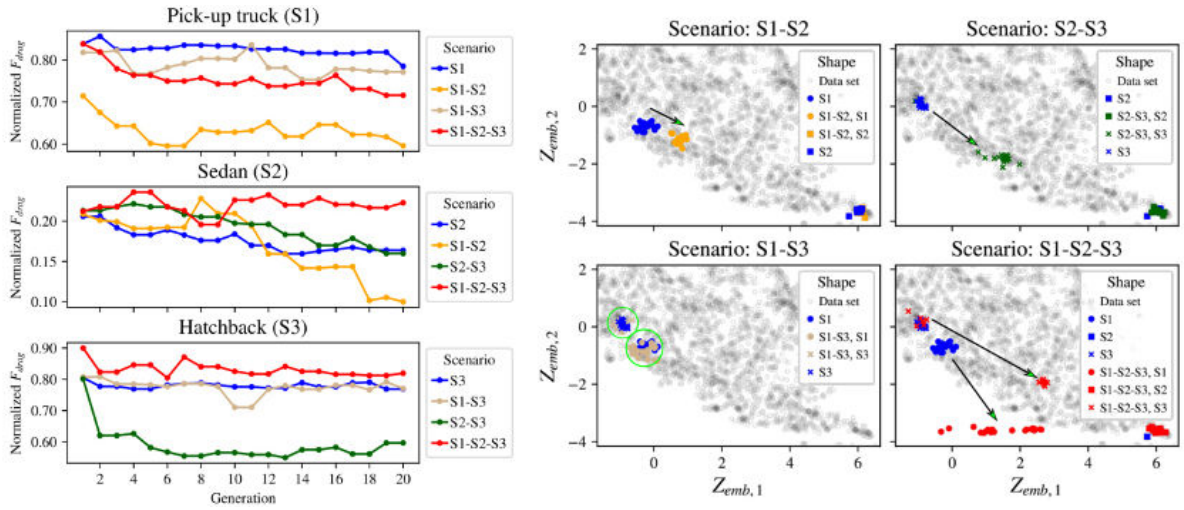

Figure 24 – Analysis of the convergence ratio of the optimizations with MFEA (left) and visualization of the optimized populations in a 2d embedding of the latent space for different optimization cases (right).

By analyzing the fitness convergence results (Figure 24, left), we observed that the MFEA accelerated most of the optimization cases. The optimizations in pairs were more successful when the sedan (S2) was one of the designs in the pair. The optimizations with the pick-up and hatchback (S1,S3) and the optimization with all shapes were comparable or worse with respect to the baseline. We verified these results by visualizing the optimized populations in a 2D projection of the latent space (Figure 24, right). By comparing the latent representations, we conclude that our MFEA framework searches for designs that approximate the shape with best performance, which in these scenarios was the sedan design. Hence, when the shapes had similar performance (*e.g.*, S1 and S3), the optimization was slower, since the shapes shared similar features. Also, in the optimization with all designs, the MFEA potentially caused negative knowledge transfer from S1 and S3 to S2,

such that the sedan design remained nearly unchanged, and the other shapes presented minor improvements.

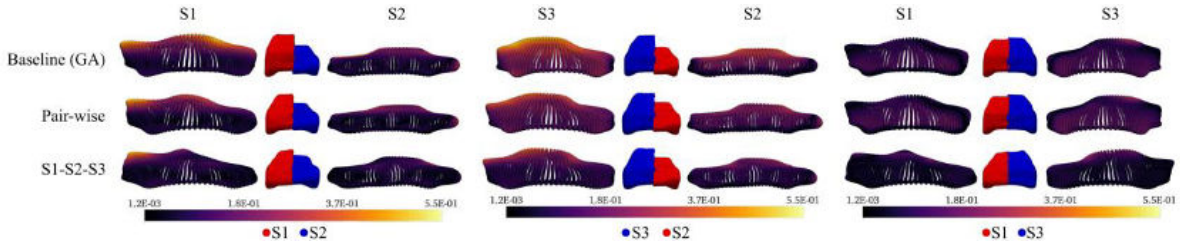

Figure 25 – Analysis of the distance between corresponding points for pairs of shapes obtained in different optimization scenarios. The brighter colors indicate greater distance (less similarity) between points.

Finally, we visualized the optimal designs obtained with MFEA and compared shape similarities for different optimization scenarios (Figure 25). Comparatively, the MFEA increased the similarity between shapes (darker regions in Figure 25), particularly in the underbody region. Therefore, we conclude that our proposed MFEA framework also enables to foster common design features in optimized shapes, which can be used to address simultaneously manufacturing-related problems without explicit constraints. Furthermore, our proposed representation adds a semantic interpretation to the knowledge transfer in MFEA, which is the transfer of geometric features between shapes that excel on different tasks.

3.4. Vehicle Aerodynamic Optimization Using Point2FFD

In the application scenario presented in this section, we propose to address the state-of-readiness of the models generated by the autoencoders using our Point2FFD architecture. In the previous cases, we prototyped simplified meshes (water-tight, genus-0) to perform the aerodynamic simulations. Since Point2FFD generates simulation-ready CAE meshes based on detailed template models, we compared the performance of Point2FFD with respect to PC-AE in a vehicle aerodynamic optimization scenario to assess the potential benefits of the proposed architecture with respect to our previous approach.

3.4.1. Experimental Set-up

For this set of experiments, we proposed the minimization of the aerodynamic drag of three car shapes: a sport utility vehicle (SUV), a coupé, and a sedan (Figure 26), which we assume to be known designs from previous product development processes. Therefore, we train Point2FFD considering these shapes as mesh templates, such that the network can exploit design features learned from a benchmark data set by deforming the prototypes.

Figure 26 – Initial designs for the aerodynamic optimization and utilized as templates to train Point2FFD.

As benchmark data set, we utilized the car class from ShapeNetCore, sampled with the shrink-wrapping method into 3D point clouds with 6146 points. We trained Point2FFD with the hyperparameters listed in Table A 1 (Appendix A), except the size of the latent layer, which we set to 50, and the number of prototypes, which we set to K=3. We set the training algorithm and remaining hyperparameters as proposed for the *Scenario A* of the benchmark analysis. For comparing the optimization performance, we trained our PC-AE architecture considering the same settings and conditions.

To calculate the aerodynamic forces, we utilized the same simulation framework considered in the multi-task optimizations: Vehicles moving in a straight line with a constant speed of $U = 110$ km/h. We performed unconstrained optimizations and, thus, the objective function only comprised the aerodynamic drag force. For optimizing the shapes, we utilized the CMA-ES algorithm [22], which we initialized with a population size of $\lambda = 16$, number of parents of $\mu = 5$ and maximum number of generations of 20. We implemented our framework in the same cluster set up as used in the multi-task optimizations, where we performed each simulation in parallel using 16 processors. Each batch of simulations based on the simplified meshes generated on the output point clouds of the PC-AE took at most 75 minutes, while the batches of simulations based on the Point2FFD output meshes took 120 minutes each.

3.4.2. Results and Discussion

Comparing the mean fitness of the population over the generations (Figure 27), we observe that Point2FFD achieved results significantly better (>50%) than PC-AE. The difference in performance is justified by two main reasons. The first is that Point2FFD ultimately generates shapes by deforming high-quality polygonal meshes and the shape-generative model is trained with noise. Hence, the shape-generative model of Point2FFD is more robust and when the optimization algorithm visits unknown regions in the latent space, the FFD potentially smooths the shape modifications while the PC-AE yields fuzzy point clouds.

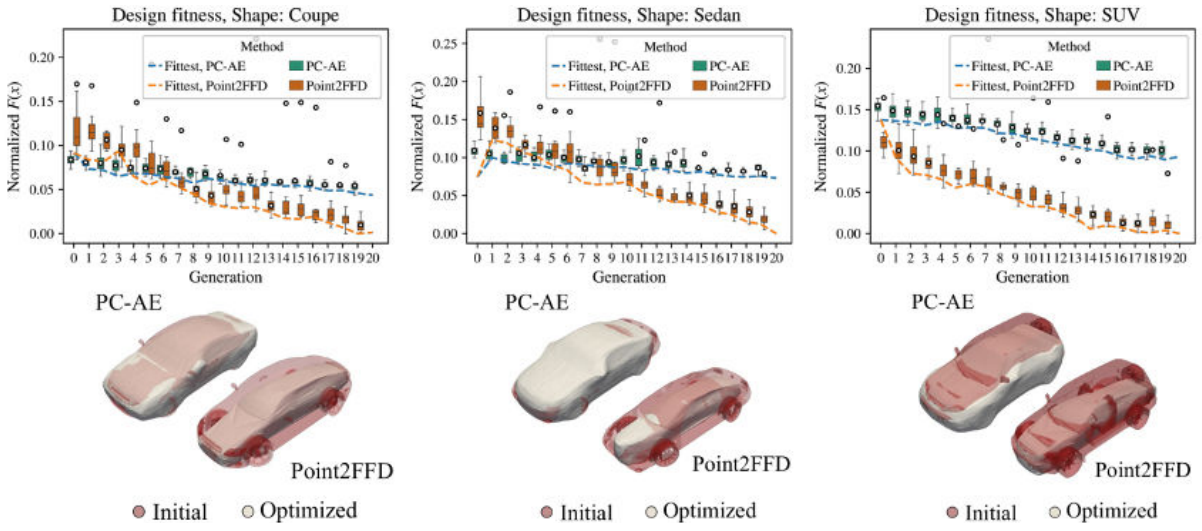

Figure 27 – Analysis of the population fitness over the generations (top) and comparison of the optimized shapes (bottom) obtained with the PC-AE and Point2FFD representations.

The second reason is the level of detail in the generated shapes, which reflects the fidelity of the simulation results and, thus, the evolution path. Since Point2FFD recovers more realistic shapes, the representation allows the optimizer to modify local geometric characteristics more efficiently, since in the mesh reconstruction on the PC-AE point clouds eliminates small scale changes. Hence, the optimization algorithm is able to achieve better solutions with Point2FFD within a smaller number of generations.

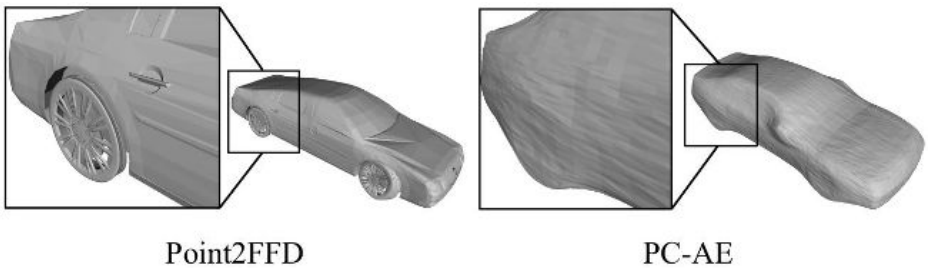

Figure 28 – Comparison of the wheels and wheelhouse of the fittest individuals at the first generation of the sedan optimization.

4. GDL4DesignApps: Geometric Deep Learning for Design Applications

In order to efficiently utilize the proposed methods in engineering design optimization, we implemented the deep-generative models in a single software package called *GDL4DesignApps*: Geometric Deep Learning for Design Applications. The package comprises the algorithms for training and applying our neural network architectures, for the visualization of the network features, as well as examples of their applications in design optimization as described in this report.

4.1. Requisites and Testing Conditions

We implemented the software using the Python language and open-source libraries (Table 6). For

the applications that require the calculation of the Chamfer Distance implemented in [11], we adapted the source code used in [8] to Python 3 and to the specific hardware utilized in the experiments. Also, for visualizing and working with the examples, we recommend the installation of the Jupyter notebook library [25].

Table 6 – List of Python libraries required for utilizing our software package.

Library	Version	Application
Numpy	1.19.1	Scientific computing
Pandas	1.1.0	Data analysis toolkit
Tensorflow-GPU	1.14.0	Platform for machine learning applications
TFLearn	0.3.2	Deep learning library built upon Tensorflow
Matplotlib	3.2.2	Library for data visualization
Plotly	4.9.0	Standalone javascript data visualization library
Cuda toolkit	10.1.168	NVIDIA toolkit for GPU-accelerated applications
CudNN	7.6.5	NVIDIA CUDA® deep neural network library
Scikit Learn	0.23.2	Python module for machine learning
PyVista	0.29.1	Visualization toolkit for geometri data
Pycma	3.1.0	Implementation of the CMA-ES algorithm
Platypus	1.0.4	Framework for evolutionary computing

We developed our software package on machines with Linux OS (Ubuntu 18.04) with dedicated Conda® environments. In terms of hardware, we tested our scripts on machines with two CPUs Intel® Xeon® clocked at 2.10 GHz and on two different sets of GPUs: four GPUs Nvidia® GeForce® RTX2080 Ti with 12 GB each, and two GPUs Nvidia® Quadro® RTX8000 with 48GB each. The scripts for training the architectures are the most computationally demanding and, therefore, perform better in setups with GPUs. However, applying the networks as shape-generative models or for feature visualization requires less computational effort. Therefore, in these cases, we computed the shapes using CPUs.

The last requisite to utilize the scripts is the data set with the CAE models, which has to be generated by the users or downloaded from benchmark repositories, such as ShapeNet [2]. For our scripts, we assume that the data is stored in a single directory and that the files are in one of the following formats: *.csv, .xyz, .stl* or *.obj*. The maximum size and number of CAE models that can be loaded depends on the available hardware. In our experiments, we utilized data sets with up to 7000 shapes and a maximum point cloud size of 24578 points.

4.2. Software Modules

4.2.1. Pre-processing CAE Models

As a general case, we assume that the data set for training the architectures comprises CAE models without a common structure or topology. If the target task requires random uniform sampling (RUS), the user can directly proceed to the training scripts, since the sampling probability neglects

any information related to geometric features. In case the user decides to sample the shapes using high-pass filtering (HPF) or shrink-wrapping (SWM), one has to preprocess the data before running the scripts to train the architectures (Figure 29). For the HPF, our scripts apply an edge-detection filtering technique to the geometries and yield files with the sampling probabilities, which are utilized by the training scripts to load the data set. The procedure to sample the shapes with SWM is similar, but the corresponding script outputs the point cloud (XYZ) and mesh (STL) files, which are used by the training scripts.

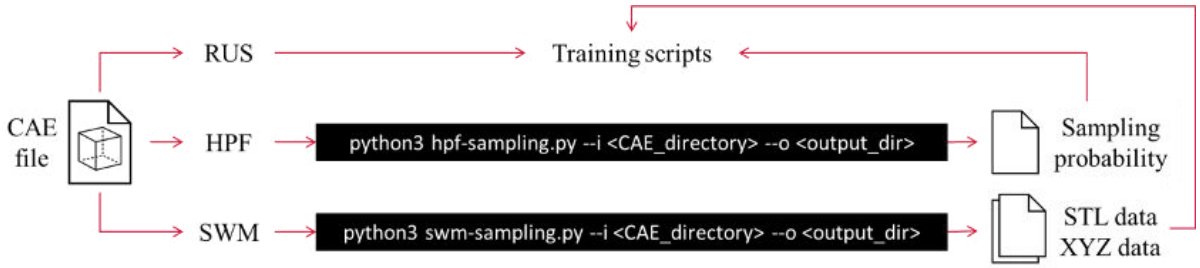

Figure 29 – Workflow for sampling CAE models before training the architectures with the commands for running the scripts.

4.2.2. Training the Deep-generative Models

The first step to use our PC-AE and PC-VAE includes the training of the parameters on a dataset of 3D point clouds. We created individual scripts to train each of the proposed models to address specific implementation details. However, all algorithms follow a similar structure (Figure 30) and command line to start the algorithms. To start the training scripts on a terminal window, the user has to execute the following command:

python *script_name* --config *config_file* --GPU *gpu_id*,

Where *script_name* is the name of the training script, *config_file* is a file with a dictionary containing the network hyperparameters, and *gpu_id* the identification of the GPU that should be used.

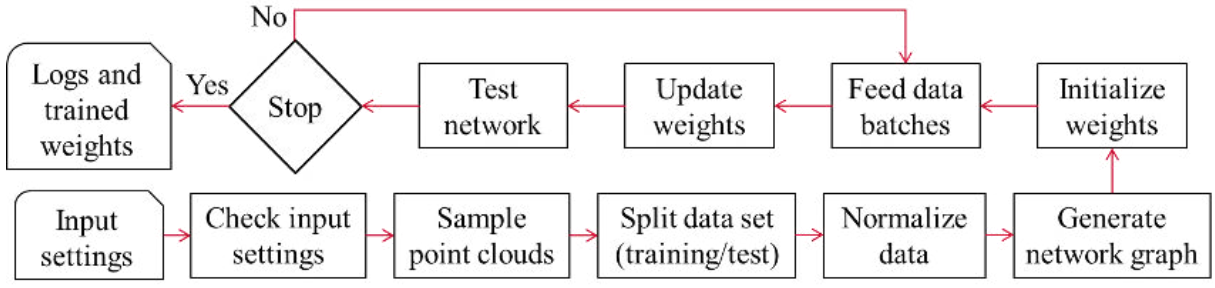

Figure 30 – Pseudocode of the basic algorithm utilized for training the architectures.

We use the configuration file for multiple scripts, such as training, testing, and visualizing the network features. Furthermore, it allows to easily log the network configuration in benchmark experiments and avoids errors by manually modifying multiple application scripts. The structure of the dictionary in the file (Figure 31) is fixed, as well as the names of the dictionary ("confignet") and keys (*e.g.*, "net_id"). The applications manage to utilize only the required keys and neglect

unnecessary information, *e.g.,* mesh templates for training PC-AE, and the dictionary is stored in a python file (*.py*).

```
confignet = {
        'net_id': 'name to identify the network',
        'dataset': [list of directories with CAE data],
        'proto_list': [list of files to be used as a mesh templates (Point2FFD)],
        'class_layers': [list with the layers of the MLP classifier (Point2FFD)],
        'ffd_lmn': [list with control planes in x-, y-, z-direction (Point2FFD)],
        'data_size': [list with number of shapes to be sampled from each data set],
        'out_data': 'directory to store the logs and trained weights',
        'training_batch_size': batch size for training the architecture,
        'test_batch_size': batch size for testing the architecture,
        'pc_size': point cloud size,
        'latent_layer': size of the latent layer,
        'noise_amp': standard deviation for the noise in the latent layer (Point2FFD),
        'encoder_layers': [list with the number of features the encoder layers],
        'decoder_layers': [list with the number of features the decoder layers],
        'l_rate': learning rate,
        'epochs_max': maximum number of epochs,
        'stop_training': termination criteria: minimum loss value,
        'frac_training': split of the data used for training the architecture (90% → 0.9),
        'autosave_rate': interval of epochs for saving the weights and log files
    }
```

Figure 31 – Structure of the configuration file used for training and applying the deep-generative models.

Finally, after training the architectures, the scripts store a set of files in the specified output directory. Besides a copy of the configuration file, the output data includes the lists of files used for training and testing the architecture, the history of losses calculated during training, the trained weights and graph of the architecture, which we need to load before utilizing the networks in downstream applications.

4.2.3. Network Evaluation and Feature Visualization

We also implemented the scripts to calculate the losses on test sets of geometric data, as well as to visualize the features calculated in the encoder layers. The main requisite for running these scripts are the files with the network configuration, trained weights, and network graph. The baseline algorithm (Figure 32) follows the initialization of the training algorithm. However, instead of the optimization of the network weights, the scripts load the trained network to calculate the network features and reconstructions based on the input data. Then, based on the obtained values, the scripts process the data to generate the output files.

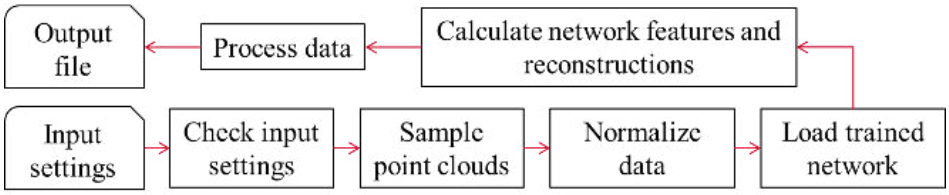

Figure 32 – Baseline algorithm for calculating the reconstruction losses and visualizing the network features based on a test set of geometric data.

In the scripts, we load the network information in different steps within a Tensorflow session (Figure 33). First, we load the graph and trained parameters (variable *graph*), which includes all operations and assigned variables of the deep neural network. In the following steps, we load specific network operators, such as the input neurons (x), latent vector (Z) and features for visualization (*feat*). We utilize the variables to feed or retrieve values processed by the network.

```
# Import the Tensorflow library
import tensorflow as tf

# Start a Tensorflow session
sess = tf.Session()

# Import Graph
new_saver = tf.train.import_meta_graph(TFmetaFile, clear_devices=True)
new_saver.restore(sess, tf.train.latest_checkpoint(TFDirectory))
graph = tf.get_default_graph()

## Import network layers' output as variables
# Input
x = graph.get_tensor_by_name("x:0")

# Latent Representation
Z = graph.get_tensor_by_name("latent_rep:0")

# Features for visualization
feat = graph.get_tensor_by_name("CLayer_4:0")

# Output point clouds
point_clouds = graph.get_tensor_by_name("PC:0")

# Calculate Z, feat, and point_clouds and assign to the variables lr, F and S, respectively
lr, F, S = sess.run([Z, feat, point_clouds], feed_dict={x: pc})
```

Figure 33 – Code snippet of the code used for loading data from the trained networks.

The network evaluation scripts output a text file with the names of the input geometries, the corresponding latent representation, reconstruction losses and a flag identifying if the shape was in the training or test data. For the feature visualization, the script outputs image files with the projected activations on the input point clouds as colormaps. The command lines for running the scripts follows the same pattern as for training the networks, but the name of the script is substituted by the corresponding scripts for network evaluation and feature visualization.

4.2.4. Shape-generative Tasks

We utilize the networks in shape-generative tasks within design and optimization problems using the same routine to load the networks (Figure 33). Although the optimization frameworks have different implementation details, the algorithms utilize similar sub-routines, such as to generate shapes from latent representations after performing operations using the latent variables.

A typical operation in the latent space is to combine shapes through interpolation, as we performed in [12] to initialize the populations in multi-objective design optimizations. In our scripts (Figure 34), after loading the trained PC-VAE network and assuming two point clouds $S_1$and S_2, we utilize the network operators to calculate the respective latent representations (Z_1, Z_2). To interpolate the designs, we linearly interpolate the latent variables (in our example with a ratio of 50%), and feed them back to the network using the operator Z as input to generate the output point cloud. This example can be extended to the combination and transfer of individual features, as performed in [7], and random sampling in the latent space, as shown in [13].

```
## Import network layers' output as variables
# Input
x = graph.get_tensor_by_name("x:0")

# Dropout rate (required only for the PC-VAE)
dout = graph.get_tensor_by_name("do_rate:0")

# Latent Representation
Z = graph.get_tensor_by_name("latent_rep:0")

# Output point clouds
point_clouds = graph.get_tensor_by_name("PC:0")

## Calculate the latent representations of two point clouds: S1 and S2
Z1 = sess.run(Z, feed_dict={x: S1})
Z2 = sess.run(Z, feed_dict={x: S2})

## Generate interpolated  latent representation (50% of Z1 + 50% of Z2)
Zm = 0.5*Z1 + 0.5*Z2

## Generate output point cloud
Sm = sess.run(point_clouds, feed_dict={Z: Zm, dout: 1.0})
```

Figure 34 – Code snippet to interpolate two shapes based on the respective latent representations and generate the corresponding 3D point cloud.

4.3.Licensing and Deployment

We open-sourced our software package in a GitHub repository called *Geometric Deep Learning for Design Applications* (GDL4DesignApps), available in https://github.com/HRI-EU/GDL4DesignApps and forked to the project repository in https://github.com/ECOLE-ITN/GDL4DesignApps [26]. The scripts are provided under the terms of GNU General Public License version 3 (GPL 3.0) as published by the Free Software Foundation. In the repository, we provide instructions for downloading and usage of the scripts, as well as examples of application.

The repository is organized in three directories: *main*, *include* and *examples*. In *main*, we provide the software documentation and licensing information. The *include* directory contains the main scripts, such as for training the deep neural networks and optimization frameworks. In *examples*, we added more detailed instructions to utilize the software package in Jupyter notebook files.

5. Summary and Outlook

Engineers intuitively exploit experience learned by solving different problems to address new and more challenging tasks. The increasing availability of digital data generated during the design process of products delineate a potential scenario to learn design experience from engineering data using sophisticated deep learning architectures. Yet, the high-dimensionality and unstructured nature of the data poses many challenges for state-of-the-art machine learning algorithms, which we address with a new set of methods compiled in a single software package: GDL4DesignApps.

In this report, we present the background and implementation details of our software package. We divided the discussion of our methods into two main parts: pre-processing and optimization. In the pre-processing, we discuss the utilized techniques for sampling 3D point clouds from CAE models, the architecture of our proposed deep-generative models and a benchmark analysis, where we compare the performance of the proposed neural networks to a reference model from the literature. We show that our implementations learn geometric data more efficiently, since our networks provide comparable shape-generative performance and require lower computational effort. Furthermore, we have advanced the state-of-the-art on deep-generative models by proposing Point2FFD, a simulation-oriented generative model that generates shapes with higher state of readiness for engineering simulations than 3D point clouds, as used in previous research. We also utilize the technique that we have proposed for visualizing the network features to select degrees of freedom for optimization problems and feature transfer between shapes.

Following the pre-processing, we demonstrate the application of our proposed deep-generative models in four different real world-inspired optimization scenarios. In a surrogate modelling analysis, we show that organizing the point clouds enhances the information learned in the latent space of our autoencoders by providing an implicit notion of global geometric structure to the network. Furthermore, we explore the shape-generative capabilities of our proposed point cloud variational autoencoder (PC-VAE) for data augmentation in surrogate modelling, which is more efficient than direct manipulation of raw CAE data. Similarly, in multi-objective optimization scenarios, we explore smart sampling techniques in latent space learned by our PC-VAE to initialize the population with a “warm start”. Hence, we allow the optimization algorithms to converge faster to Pareto-optimal solutions and address better the user preferences.

In a multi-task vehicle aerodynamic optimization, we utilize our proposed point cloud autoencoder (PC-AE) as a method to map designs that address different tasks to a single design space. We show that by transferring latent features between shapes assigned to different tasks, the optimization algorithm implicitly transfers knowledge between different optimization problems. Furthermore, the proposed mechanism of feature transfer also fosters commonality in the optimized designs, which can be used to address manufacturing costs of multiple vehicles. In a last set of experiments,

we compare the performance of Point2FFD to the proposed PC-AE in vehicle aerodynamic optimizations. We show that Point2FFD generates shapes with more detail and is more robust than our previous generative models, which improves the quality of the optimization results, as well as accelerates the solution of the optimizations.

Finally, we introduce our software package, which we open-sourced as a Python library on the GitHub page of the ECOLE project. We provide the main requisites for installing and utilizing the algorithms, which includes the specification of the hardware utilized to test the algorithms. Also, we present an overview of the implemented modules and provide short examples of usage through code snippets and block diagrams of the algorithms, which are also available in the repository together with other examples as part of the software documentation.

References

[1] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam and P. Vandergheynst, "Geometric Deep learning: Going Beyond Euclidean Data," *IEEE signal Processing Magazine,* vol. July, pp. 18-42, 2017.

[2] A. X. Chang, A. Funkhouser, L. J. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi and F. Yu, "ShapeNet: An Information-rich 3D Model Repository," *ArXiv,* vol. arXiv preprint 1512.03012v1 [cs.GR], 2015.

[3] T. Rios, P. Wollstadt, B. van Stein, T. Bäck, Z. Xu, B. Sendhoff and S. Menzel, "Scalability of Learning Tasks on 3D CAE Models Using Point Cloud Autoencoders," in *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*, Xiamen, 2019.

[4] S. Saha, T. Rios, J. Kong, T. Bäck, S. Menzel and B. Sendhoff, "Multi-Criteria Optimization Focusing On Learning For Adaptive Feature Selection And Constraints Prediction," Honda Research Institute Europe GmbH, Offenbach am Main, 2020.

[5] T. Rios, B. van Stein, P. Wollstadt, T. Bäck, B. Sendhoff and S. Menzel, "Exploiting Local Geometric Features in Vehicle Design Optimization with 3D Point Cloud Autoencoders," in *2021 IEEE Congress on Evolutionary Computation (CEC)*, Krákow, 2021.

[6] J. Vollmer, R. Mencl and H. Müller, "Improved Laplacian Smoothing of Noisy Surface Meshes," *Computer Graphics Forum,* vol. 18, pp. 131-138, 1999.

[7] T. Rios, B. van Stein, T. Bäck, B. Sendhoff and S. Menzel, "Multi-Task Shape Optimization Using a 3D Point Cloud Autoencoder as Unified Representation," *IEEE Transactions on Evolutionary Computation,* 2021.

[8] P. Achlioptas, O. Diamanti, I. Mitliagkas and L. Guibas, "Learning Representations and Generative Models for 3D Point Clouds," *Proceedings of the 35th International Conference on Machine Learning (ICML),* vol. 80, pp. 40-49, 2018.

[9] T. Rios, B. van Stein, T. Bäck, B. Sendhoff and S. Menzel, "On the Efficiency of a Point Cloud Autoencoder as a Geometric Representation for Shape Optimization," in *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*, Xiamen, 2019.

[10] T. Rios, B. van Stein, S. Menzel, T. Bäck, B. Sendhoff and P. Wollstadt, "Feature Visualization for 3D Point Cloud Autoencoders," in *International Joint Conference on Neural Networks (IJCNN), 2020.*, Glasgow, Scotland, 2020.

[11] H. Fan, H. Su and L. Guibas, "A Point Set Generation Network for 3D Object Reconstruction from a Single Image," *30th IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2017),* Vols. 2017-January, pp. 2463-2471, 2017.

[12] S. Saha, L. L. Minky, X. Yao, B. Sendhoff and S. Menzel, "Exploiting Linear Interpolation of Variational Autoencoders for Satisfying Preferences in Evolutionary Design Optimization," in *iEEE Congress on Evolutionary Computation (CEC)*, 2021.

[13] S. Saha, T. Rios, L. L. Minku, B. v. Stein, P. Wollstadt, X. Yao, T. Baeck, B. Sendhoff and S. Menzel, "Exploiting Generative Models for Performance Predictions of 3D Car Designs," in *IEEE Symposium Series on Computational Intelligence (SSCI) (submitted)*, 2021.

[14] N. Sharp and M. Ovsjanikov, "PointTriNet: Learned Triangulation of 3D Point Sets," in *Computer Vision - ECCV 2020*, Springer International Publishing, 2020, pp. 762-778.

[15] R. Hanocka, G. Metzer, R. Giryes and D. Cohenor, "Point2Mesh: A Self-Prior for Deformable Meshes," *ACM Transaction on Graphics,* vol. 39, no. 4, pp. 1-12, 2020.

[16] D. Jack, J. K. Pontes, S. Sridharan, C. Fookes, S. Shirazi, F. Maire and A. Erikson, "Learning Free-Form Deformations for 3D Object Reconstruction," in *Computer Vision - ACCV 2018*, Springer International Publisher, 2019, pp. 317-333.

[17] L. Gao, J. Yang, T. Wu, Y.-J. Yuan, H. Fu, Y.-K. Lai and H. Zhang, "SDM-NET: Deep Generative Network for Structured Deformable Mesh," *ACM Transactions on Graphics,* vol. 38, no. 6, pp. 1-15, 2019.

[18] T. W. Sederberg and S. R. Parry, "Free-form Deformation of solid geometric models," *Proceedings of the 13th Annual Conference ib Computer Graphics and Interactive Techniques (SIGGRAPH '86),* pp. 151-160, 1986.

[19] T. Rios, J. Kong, B. van Stein, T. Bäck, P. Wollstadt, B. Sendhoff and S. Menzel, "Back To Meshes: Optimal Simulation-ready Mesh Prototypes For Autoencoder-based 3D Car Point Clouds," in *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, Canberra, Australia, 2020.

[20] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *ArXiv,* vol. arXiv:1412.6980 [cs.LG], 2015.

[21] R. Gómez-Bombarelli, J. N. Wei, D. Duvenaud, J. M. Hernández-Lobato, B. Sánchez-Lengeling, D. Sheberla, J. Aguilera-Iparraguirre , T. D. Hirzel, R. P. Adams and A. Aspuru-Guzik, "Automatic Chemical Design Using a Data-Driven Continuous Representation of Molecules," *ACS Central Science,* vol. 4, no. 2, pp. 268-276, 2018.

[22] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies.," *Evolutionary computation,* vol. 9, no. 2, pp. 159-195, 2001.

[23] A. Gupta, Y.-S. Ong and L. Feng, "Multifactorial Evolution: Toward Evolutionary Multitasking," *IEEE Transactions on Evolutionary Computation,* vol. 20, no. 3, pp. 343-357, 2016.

[24] J. Ding, C. Yang, Y. Jin and T. Chai, "Generalized Multitasking for Evolutionary Optimization of Expensive Problems," *IEEE Transactions on Evolutionary Computation,* vol. 23, no. 1, pp. 44-58, 2019.

[25] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla and C. Willing, "Jupyter Notebooks – A Publishing Format for Reproducible Computational Workflows," in *Proceedings of the 20th International Conference on Electronic Publishing*, Göttingen, Germany, 2016.

[26] T. Rios, S. Saha, S. Bernhard and S. Menzel, "GDL4DesignApps: Geometric Deep Learning for Design Applications," GitHub, 20 08 2021. [Online]. Available: https://github.com/ECOLE-ITN. [Accessed 20 08 2021].

Appendix A

Table A 1 - Architectures and corresponding hyperparameters utilized in the benchmark analysis.

PC-AE-Achlioptas				**PC-AE**			
Loss function: Chamfer Distance [11]				**Loss function:** Mean-squared distance (MSD)			
Layer	**Type**	**Activation**	**Output**	**Layer**	**Type**	**Activation**	**Output**
1	1D-C	ReLU	[N_p,64]	1	1D-C	ReLU	[N_p,64]
2	1D-C	ReLU	[N_p,128]	2	1D-C	ReLU	[N_p,128]
3	1D-C	ReLU	[N_p,128]	3	1D-C	ReLU	[N_p,128]
4	1D-C	ReLU	[N_p,256]	4	1D-C	ReLU	[N_p,256]
5	1D-C	ReLU	[N_p,128]	5	1D-C	tanh	[N_p,128]
6	max-pool	None	[1,128]	6	max-pool	None	[1,128]
7	FC	ReLU	[256,3]	7	FC	ReLU	[256,3]
8	FC	ReLU	[256,3]	8	FC	ReLU	[256,3]
9	FC	None	[N_p,3]	9	FC	sigmoid	[N_p,3]
PC-VAE				**Point2FFD**			
Loss function: $\alpha_1 CD + \alpha_2 MSD$				**Loss function:** $MSD(S_i, \tilde{S}_i) + MSD_{templt}$			
1	1D-C	ReLU, BN	[N_p,64]	1	1D-C	ReLU	[N_p,64]
2	1D-C	ReLU, BN	[N_p,128]	2	1D-C	ReLU	[N_p,128]
3	1D-C	ReLU, BN	[N_p,128]	3	1D-C	ReLU	[N_p,128]
4	1D-C	ReLU, BN	[N_p,256]	4	1D-C	ReLU	[N_p,256]
5	1D-C	tanh	[N_p,128]	5	1D-C	tanh	[N_p,128]
6	max-pool	None	[1,128]	6	max-pool	None	[1,128]
7,μ	none	None	[1,128]	7,d	FC	ReLU	[256,3]
7,σ	1D-C	sigmoid	[1,128]	8,d	FC	ReLU	[256,3]
8	FC	ReLU, BN	[256,3]	9,d	FC	None	[lmn,3]
9	FC	ReLU, BN	[256,3]	7,m	FC	ReLU	[25,1]
10	FC	sigmoid	[N_p,3]	8,m	FC	softmax	[K,1]

Acronyms and variables: 1D convolution (1D-C), fully connected (FC), point cloud size (N_p), number of templates (K) and batch normalization (BN). The indices *m* and *d* in the layers of Point2FFD indicate *MLP* and *decoder*, respectively.

Appendix B

Algorithm 2 – MFEA algorithm adapted from [23] for the multi-task optimization framework proposed in [7].

ALGORITHM: Adapted multi-factorial evolutionary algorithm

REQUIRE: Trained autoencoder $(\mathcal{E}, \mathcal{D})$, rmp, λ

1: INITIALIZATION

2: **FOR** each task T_j , $j = (1, \ldots, K)$ **DO**

3: Calculate the latent representation of the initial design S_j using the trained encoder: $Z_j^0 = \mathcal{E}(S_j)$

4: Mutate λ/K times the initial solution Z_j^0 and append to the population P

5: Assign the skill factor of Z_j^0 to each offspring $Z_i : \tau_i = j$

6: **END FOR**

7: **FOR** Each individual in p_i, $i = (1, \ldots, \lambda)$ **DO**

8: Transfer the features in Z_j', from Z_i to Z_j^0, where $j = \tau_i$, which generates a temporary individual Z_j^{0*}

9: Reconstruct the shape S_i in the Cartesian space using the trained decoder: $S_i = \mathcal{D}(Z_j^{0*})$

10: Evaluate the design S_i for the task τ_i and calculate Ψ_i

11: **END FOR**

12: ITERATIONS

13: **WHILE** (termination criteria are not achieved) **DO**

14: Calculate the factorial rank r_i and the scalar fitness φ_i I of the individuals

15: Select individuals to generate the population in the next iteration

16: Perform assortative mating and generate offspring C

17: Assign τ_i through vertical cultural transmission algorithm

18: **FOR** Each individual in $c_{i,j} = (1, \ldots, \lambda)$ **DO**

19: Transfer the features in Z_j', from Z_i to Z_j^0, where $j = \tau_i$, which generates a temporary individual Z_j^{0*}

20: Reconstruct the shape S_i in the Cartesian space using the trained decoder: $S_i = \mathcal{D}(Z_j^{0*})$

21: Evaluate the design S_i for the task τ_i and calculate Ψ_i

22: **END FOR**

23: **END WHILE**

24: **RETURN** S_j @ $\max(\varphi_j)$, $j = (1, \ldots, K)$

Variables: Encoder ($\mathcal{E}$), decoder ($\mathcal{D}$), latent representation (Z), number of tasks (K), population size (λ), factorial cost (Ψ), skill factor (τ), scalar fitness (φ)