ECOLE
Experience-based Computation:
Learning to Optimise

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

Deliverable D1.2

Multi-Criteria Optimization Focusing On Learning For Adaptive Feature Selection And Constraints Prediction

(incl. Software Components)

Authors:
Sneha Saha, Thiago Rios, Stefan Menzel, Bernhard Sendhoff – HRI-EU
Jiawen Kong, Thomas Bäck – Leiden University

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

H2020 MSCA-ITN
Date of the report: 30.09.2020

Contents

1. Introduction 3
2. Geometric data manipulation 6
3.1. 3D point cloud autoencoder 8
3.2. 3D point cloud variational autoencoder (PC-VAE) 16
4. Surrogate assisted optimization and evaluation 23
4.1. Improving vehicle design classification 23
5. Summary and Outlook 26
Bibliography 28
Appendix A – Vanilla 3D point cloud autoencoder scripts 31
Appendix B – 3D point cloud variational autoencoder scripts 34
Appendix C – Improving imbalance classification 38

1. Introduction

In the Experience-based Computation: Learning to Optimise (ECOLE) project, we research novel methods and smart computational models for capturing the notion of experience embedded in (engineering) data, which are collected over the course of optimization runs, and exploiting said experience in similar, yet more challenging, optimization tasks (Figure 1). Our vision of experience is analogous to an engineer who also learns and transfers her/his knowledge from previous professional tasks to new and different types of applications. In automotive product design, this experience is usually shared between humans developing models of different type (sedan, convertible etc.) to handle efficiently the required criteria from different engineering domains as well as over the course of car model changes (minor/major model change) to apply existing knowledge from the past.

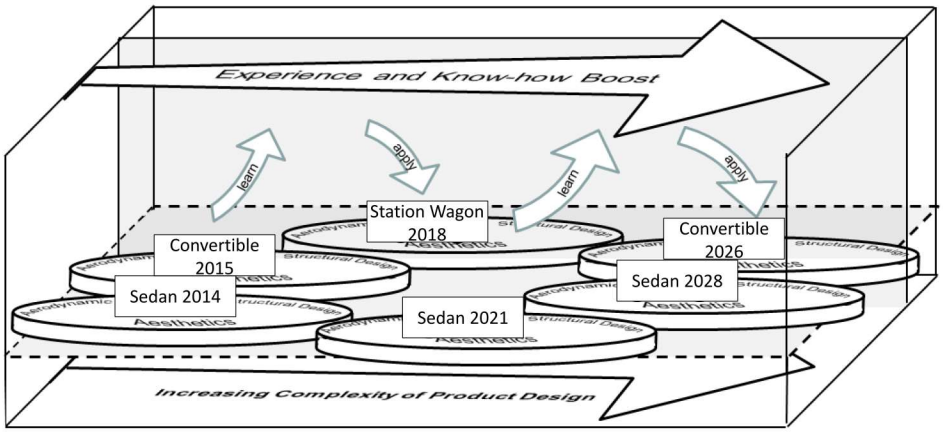


Figure 1– Vision for experience-based multi-criteria optimization in the automotive domain

In Work Package 1 (WP1) of the ECOLE project, we focus on the research on experience-guided optimization in automotive product design, linking scientific questions to industrial problems and adding a practical perspective to the research tasks. The utilization of experience in optimization problems comprises a variety of advantages. Thinking of the constantly increasing complexity during product design and of final products, experience allows the user to assess more accurate system responses by advanced simulation set-ups, e.g. based on improved feature selections. This leads to improved decision-making processes and to an efficient sampling among what-if scenarios. In addition, computational experience-based models enable the human user to work on multi-criteria optimization scenarios, where these models support the human user with complementary knowledge to assist in finding optimal trade-off solutions, e.g. by predicting if (or if not) solutions violate constraints and fulfill given specifications. More visionary, computational experience-based models may even coach novice human users to build up their reliable knowledge base.

Since a few decades, computer aided engineering and design (CAE/D) software plays an important role in the automotive development by providing an early assessment of the design performance based on virtual simulations, generating a huge amount of digital design data. Due to the recent advances in the field of artificial intelligence (AI), current engineering software development aims

at exploring such existing design data with statistical analyses and machine learning techniques, supporting the user to quickly generate a large variety of solutions. In optimization problems, learning the data collected from similar cases, which we see as embedded experience, potentially can capture the sensitivity of design parameters and regions in the design space that lead to optimality or novelty, which could be worth exploring. Nevertheless, mining engineering optimization data is challenging due to many factors: the dimensionality and type of design representations, the particularities of different types of numerical simulations, and the sparsity of data considering the design and output spaces defined in such problems. Furthermore, the data is distributed in different stages of the optimization and thus increases the complexity to extract and transfer knowledge between different problems.

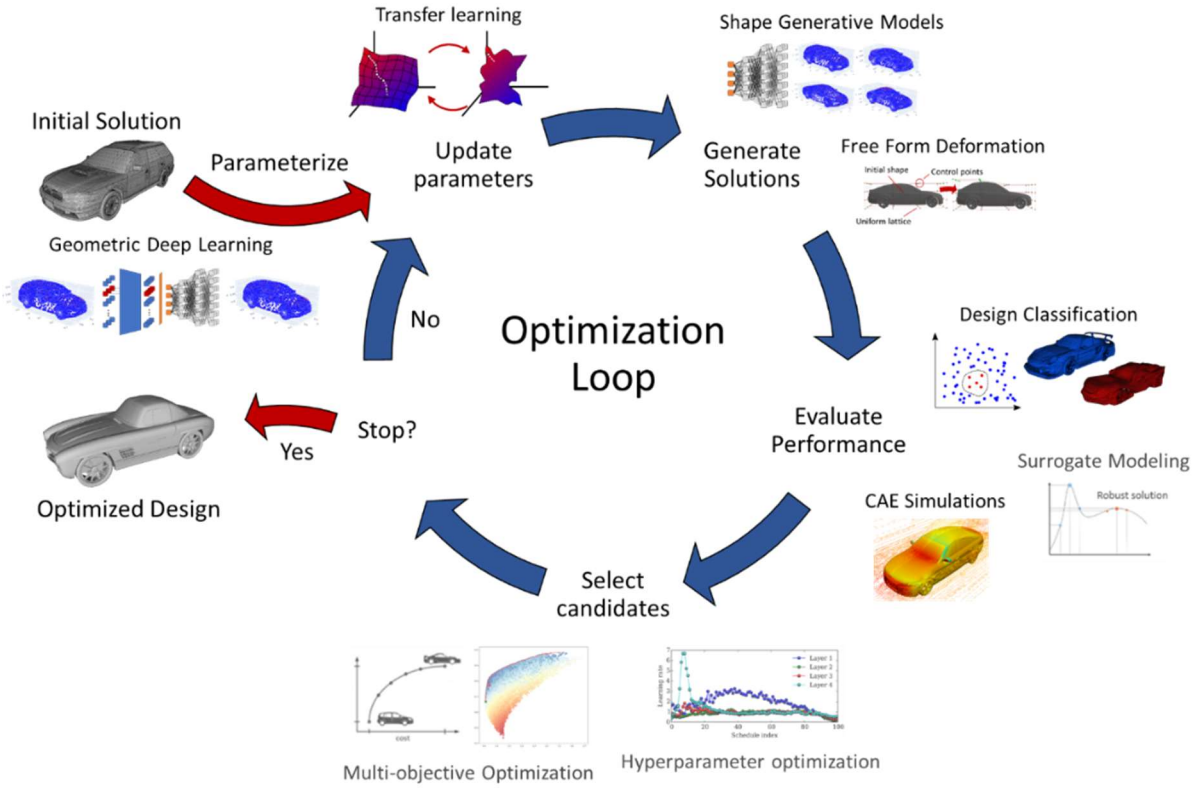


Figure 2 - Typical loop of an optimization algorithm split into tasks for which new algorithms have been developed in the ECOLE project.

A typical computational multi-criteria design optimization loop for engineering applications like automotive development is depicted in Figure 2. Usually a vehicle design is represented by a set of shape parameters. Design modifications are realized by parameter variations suggested by the chosen optimization algorithm, e.g. evolutionary optimization, gradient-based optimization among many others. These representations may either be designed by humans, e.g. CAD or shape morphing representations [1], or based on features extracted by machine learning algorithms in an unsupervised fashion. The latter technique is the focus of the present report.

Our approach utilizes geometric deep learning architectures that automatically extract features from existing CAE models, reducing the dimensionality of the geometric representation without

relying exclusively on the user expertise or particularities of the underlying physics where the shape is embedded (Section 3 and Appendix A, B). Of course, our methods rely on integrating the cumulative experience of prior vehicle designs using offline learning algorithms, while a purely human-made representation would be independent of existing data. Following the step of parameter variation, new shapes are generated using the selected shape representation. Each generated shape is then processed by computational simulations to assign a performance number to each design. These simulations typically evaluate multiple criteria, e.g. aerodynamic or structural performance, using state-of-the-art numerical solvers implemented in industrial tools. Since these computational simulations may be very time-consuming depending e.g. on applied numerical mesh resolutions and selected solvers, industrial optimization requires machine learning methods, which indicate potentially well-performing solutions rather than low-performance solutions. If low-performance solutions can be reliably detected with methods from machine learning, the simulation step can be skipped for these designs resulting in major time (=cost) savings. Thus, we also researched and implemented a classification algorithm for aerodynamic criteria that can handle imbalanced data for detecting potentially infeasible designs for computational fluid dynamics (CFD) simulations and which could be extended to other tasks/criteria within the optimization framework (Section 4 and Appendix C). For closing the optimization loop, in a final step designs and their parameters are selected from the candidate solutions. The performance of each design is either represented by a weighted sum of the performances from each criterion (single-objective optimization) or by a set of performance values that span a multi-dimensional so-called Pareto space (multi- or many-objective optimization).

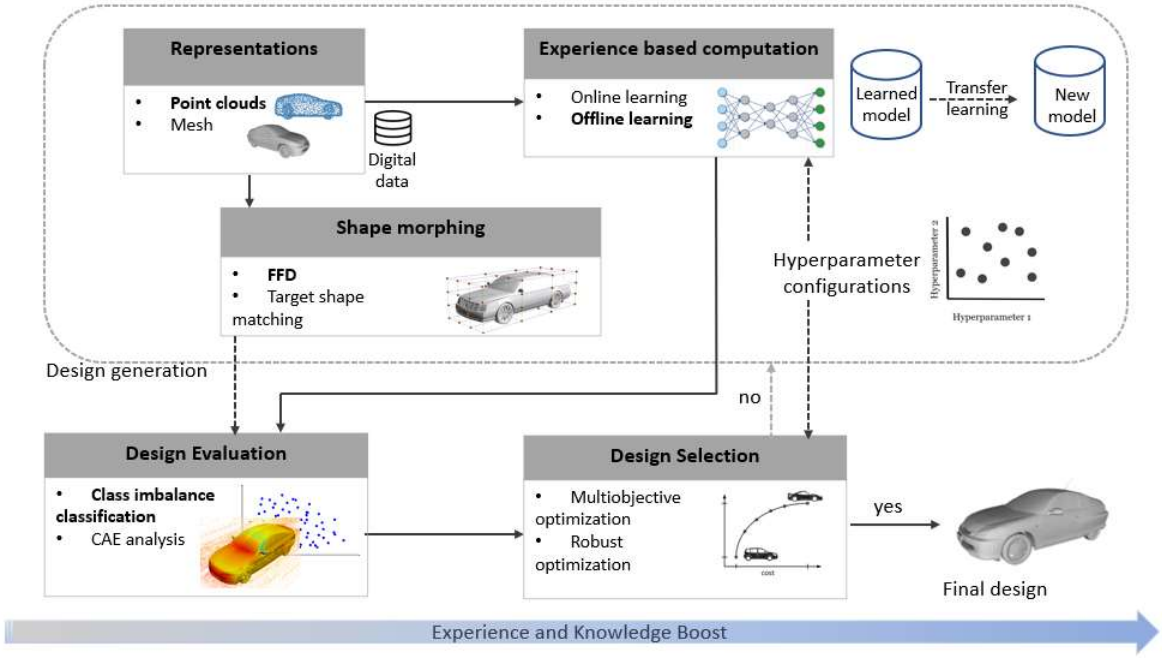


Figure 3 – Workflow for the generation of vehicle design prototypes split into tasks that were addressed with the algorithms developed in the ECOLE project.

Each of the steps in the optimization cycle comprises the implementation of state-of-the-art algorithms and the algorithms proposed within the ECOLE project, comparative tests on benchmark and (close to) real-world data, and finally publications at international conferences by the Early-Stage Researchers (ESRs). Figure 3 provides a more detailed overview on algorithms researched in ECOLE, which realize parts of the design optimization loop.

In this report, we present the current state of the multi-criteria design optimization framework focusing on the experience-based aspects of feature learning/adaptation and classification of imbalanced data. For each aspect we provide details on the scientific background of the developed methods and introduce the software components that were implemented. The remainder of the report is organized as follows: In section 2, we introduce 3D shape representations as well as benchmark and engineering data that we used for our research. In section 3, we describe the methodology for learning from cumulative existing data and experiences enabling the generation of solutions. In section 4, we demonstrate an efficient classification methodology for detecting infeasible designs in imbalanced data. Section 5 concludes the report. In the Appendix, details are provided on the developed software and on how to use the different components.

2. Geometric data manipulation

Generating and representing geometric data in a computational environment is central to many engineering processes in the automotive industry, from initial design through engineering development and analysis to manufacturing phases. Yet, the representation of 3D shapes is not canonical and often depends on the experience of the user as well as on the application. For example, surface and volume meshes predominate in 3D shape optimization problems that require computer aided engineering (CAE) simulations. In CAE, the physical domain is often discretized as meshes as a basis to solve the system of equations required to calculate e.g. flow fields or structural stiffness. However, for creating the geometries in computer aided design (CAD) software, engineers often prefer constructive solid geometry (CSG) as the representation, due to its intuitive applicability. For the research within ECOLE, we mainly explored two types of representations for computational processing: 3D polygonal meshes and 3D point clouds (Figure 4).

Figure 4 - Car representation in 3D polygonal mesh and in 3D point cloud

While the use of meshes is justified by the applicability in engineering tasks, point clouds emerged as a generic type for representing 3D shapes from the literature on geometric deep learning as being efficient for machine learning tasks [2], [3]. Furthermore, we can easily extract point clouds from engineering meshes by sampling the nodes, which allows for connecting multiple software components in the optimization process. In terms of data sets, we opted for benchmark data as available in ShapeNetCore [4] and the TUM DrivAer model [5], [6], which we already employed in a computer fluid dynamics (CFD) simulation framework.

For manipulating the geometric data, we implemented different software components, which are detailed in [7]. In the present report, we focus on the application of geometric data in optimization tasks and provide details on the learning and adaptation of 3D shape features. Furthermore, we outline classification methods for imbalanced data to support multi-criteria optimization. In addition, we provide information on the different software components to enable other research groups to utilize them in their work.

3. Learning 3D shape features from engineering data

Feature engineering is important for processing CAE data, but it is time-consuming and labor-intensive. As an alternative to manual feature extraction using shape descriptors, geometric deep learning architectures, such as (variational) autoencoders, learn in an unsupervised fashion a low-dimensional set of latent features. The 3D geometries are reconstructed from samples in the latent space, which can be explored in shape optimization problems for reduction of the dimensionality and as shape generative models.

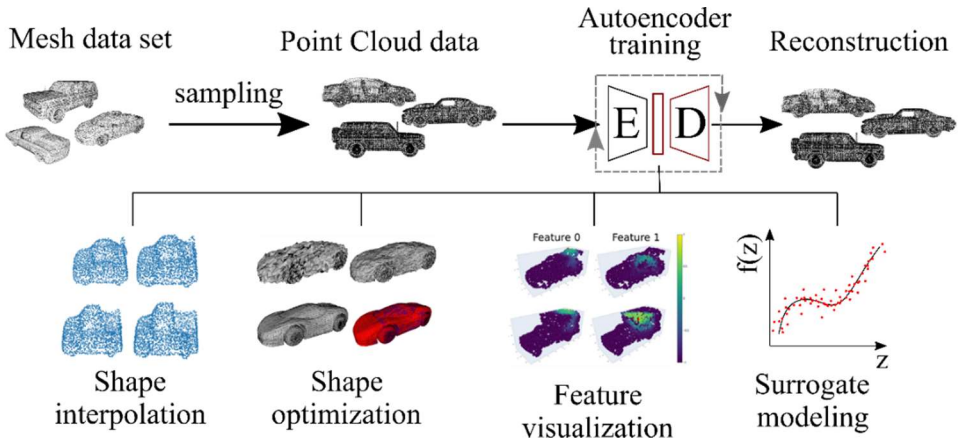


Figure 5 - Potential applications of geometric deep learning models for shape analysis and engineering optimization.

Geometric deep learning is an umbrella term for algorithms that learn on 3D geometric data. Differently to the 2D image domain, most of the geometric representations are non-Euclidean (unstructured) and invariant to the ordering of the elements. They require more sophisticated architectures and special machine learning algorithms, which distinguishes geometric deep learning from the *standard* set of deep learning methods. The literature on geometric deep learning provides architectures that handle different types of input data [2], [3], but as previously discussed, we opted for processing geometries as 3D point clouds, due to the flexibility of the representation. Thus, we aimed at researching and developing geometric deep learning models, in particular 3D autoencoders, for processing 3D point clouds.

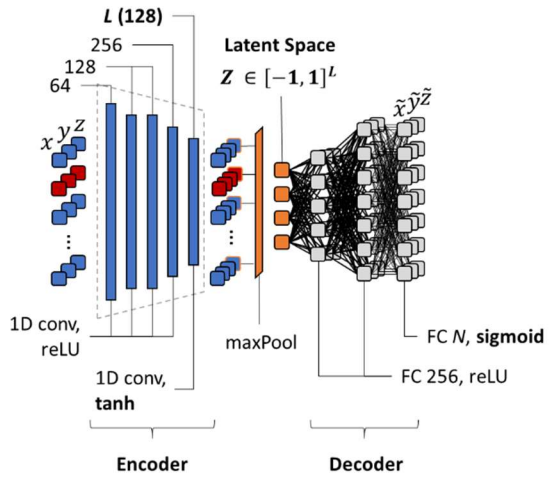
3D point cloud (variational) autoencoders were recently introduced as powerful shape generative and data compression techniques [8]. These architectures comprise an encoder, which reduces the dimensionality of the point clouds to a compact set of latent variables, and a decoder, which

recovers the cartesian coordinates of the points from the representations in latent space. The latent variables learned by the autoencoder are applicable to shape optimization problems as design parameters, performing similarly to state-of-the-art representations, such as free form deformation [9], [10]. Furthermore, these variables encode the distribution of the training data in the input space and potentially comprise enough information for fitting secondary surrogate models that map these variables to engineering performance metrics, such as aerodynamic forces. In the following subsections, we present the architectures and our research results of the standard and variational point cloud autoencoders that we propose for learning 3D shape features in an unsupervised fashion. Additionally, we provide first insights into their application to multi-criteria design optimization.

3.1. 3D point cloud autoencoder

3.1.1 Details on the 3D point cloud autoencoder architecture

As explained above, in the following we represent 3D car shapes as sets of 3D point clouds each given as Nx3 points (N: number of points), which we use to extract features provided through the latent parameters of our autoencoders. One of the main challenges for developing the architecture of a 3D point cloud autoencoder is the non-Euclidean nature of the point clouds. In order to learn features that are invariant with respect to the ordering of the points, we opted for an architecture with point-based operators [2], similar to the work presented in [11]. The encoder comprises five 1D convolutional layers, which operate pointwise on the Cartesian coordinates, followed by a maximum pooling operator, which processes the activation of the last convolutional layer in a feature-wise fashion and yields the latent representation. The decoder comprises three fully connected layers that recover the Cartesian coordinates from the latent variables, and also corresponds to the section of the network with the highest number of trainable parameters (Figure 6).



Layer	Input dimensions	Operation	Activation	Output dimensions	
0	Nx3	Input	None	Nx3	
1	Nx3	1D-C	ReLU	Nx64	Encoder
2	Nx64	1D-C	ReLU	Nx128	
3	Nx128	1D-C	ReLU	Nx128	
4	Nx128	1D-C	ReLU	Nx256	
5	Nx256	1D-C	tanh	Nx128	
6	Nx128	maxpool	None	1X128	
7	128x1	FC	ReLU	256x3	Decoder
8	256x3	FC	ReLU	256x3	
9	256x3	FC	sigmoid	Nx3	
10	Nx3	output	None	Nx3	

Figure 6 - Details of our 3D point cloud autoencoder architecture. In the table, N indicates the size of the point cloud.

Compared to the proposal in [11], we modified the activation function in the last convolutional layer from rectified linear units (ReLU) to hyperbolic tangents. Thus, it restricted the latent variables to the space $[-1, 1]^L$, thereby easing the formulation of constraints for performing shape operations in the latent space. Furthermore, we added a sigmoid function to the last fully connected layer, constraining the output space to $[0,1]^3$, which is the same as the normalization of our input data. We maintained the dimensionality of the remaining layers as proposed in the reference work, since we did not observe any issues during the experiments that could be caused by these parameters.

3.1.2 Validation of the 3D point cloud autoencoder architecture

Before applying the autoencoder to engineering problems, we validated the architecture by training the autoencoder on the car class of the ShapeNetCore data set [4], sampled uniformly to 2048 points. We opted for the standard Adam Optimizer [12] as training algorithm, with a learning rate $\eta = 5 \times 10^{-4}$, momentum terms $\beta = 0.90$ and $\beta_1 = 0.99$, and the epochs limited to 500. The shapes in the data set were randomly split into 90% and 10% partitions for training and testing the autoencoder, respectively, which were organized in batches of 50 shapes. We computed the point cloud reconstruction losses using the Chamfer Distance (CD) [13], which is invariant to the ordering of the points and thus can handle unorganized sets of point clouds. The hardware used for training and testing the model was a machine with 2 CPUs Intel Xeon Silver 4110, clocked at 2.10 GHz, with 4 GPUs NVIDIA GeForce RTX 2080 Ti. The architecture was implemented using Python scripts based on the TensorFlow 1.14 library for operating in graphic processing units (GPUs).

After training the model, we compared the mean reconstruction losses on the training and test sets to the value of the losses reported in [11], and we obtained a reduction from 3.34E-04 to 2.91E-04 and from 4.00E-04 to 3.03E-04, respectively. Additionally, we visually inspected the reconstruction of random samples in the data set (Figure 7) and interpolations between each of the samples, by which we confirmed that the autoencoder learned the car shapes in the data set.

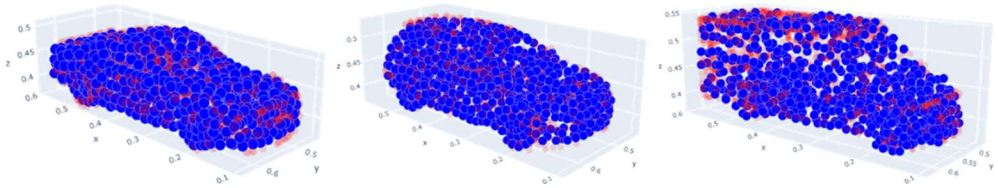


Figure 7 - Reconstruction of samples taken from the car class of ShapeNetCore [14]

3.1.3 Efficiency of our 3D point cloud autoencoders in shape optimization

Based on the successful validation of our architecture, we assessed the efficiency of using the latent features (variables) of our 3D point cloud autoencoder as design parameters for target shape matching optimization problems [15]. Therefore, we generated a synthetic data set of shapes by deforming a benchmark car surface mesh using free form deformation (FFD) [16], which is a state-of-the-art method for shape modifications used in engineering design optimization. 500 deformed car surface meshes have been converted to point clouds and used to train our autoencoder. We opted for the synthetic data for three reasons: First, we could compare the degrees of freedom

learned by the latent variables to the FFD parameterization, since we had control over the number of data set features. Second, we could control the difficulty to match the target shapes and explore the potential of the autoencoder to extrapolate the features learned in the training set. Third, since the data set comprised isomorphic meshes, we could enforce the ordering of the points during the training, which allowed us to quickly reconstruct the meshes from the point clouds generated by the autoencoder. The configuration of the FFD lattice used in the deformations contains six planes in x- and z-direction, and four in y-direction. To deform the shapes, we considered only the axial displacement of the planes and symmetry with respect to the geometric center of the car, simplifying the parameterization, which yielded 8 parameters (Figure 8).

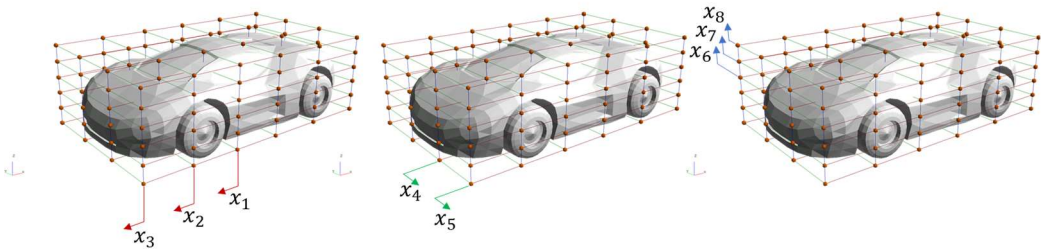


Figure 8 - FFD parametrization used for generating the data set and in the target shape matching optimization.

We divided the analysis into two optimization experiments, both performed for the FFD and autoencoder parameterization, and using the covariance matrix adaptation evolutionary strategy (CMA-ES) as optimization algorithm. In the first, we selected target shapes generated with the proposed FFD parameterization, such that the optimization with either the FFD or autoencoder representation could converge to the target shape. Hence, we could compare the convergence speed and quality of the final designs. In the second, we generated a new set of target shapes, by parameterizing the position of the control points in the y- and z-direction as harmonic functions of their position in the x-direction. Therefore, matching the target shapes would be challenging with any of the representations and we could evaluate the capability of the autoencoder to extrapolate the learned features.

In the first scenario, both representations converged to the target shapes, as expected, and the optimizations with the latent variables outperformed the FFD representation, achieving about 30% better fitness, even in the cases which showed the weakest performance of the autoencoder. Nevertheless, reconstructing intermediate results of the optimization, the autoencoder yielded meshes with a high level of noise (Figure 9), which is prohibitive in optimization problems with computational engineering simulations, such as CFD.

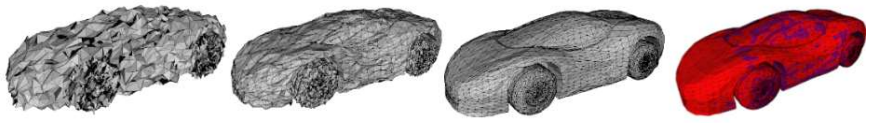


Figure 9 - Reconstruction of car shapes obtained with the autoencoder after 10, 20, 30 and 70 generations of the optimization (from the left to the right).

In the second scenario, the optimization achieved a local optimum with both representations, which was also in line with our expectations. Although numerically similar, the quality of the meshes obtained at the final optimization stage with each optimization was considerably different (Figure 10). Similar to the initial stages in the previous scenario, the autoencoder yielded a mesh with severe distortion and multiple intersections of elements. These results can be explained by the differences in the parameterizations: While the formulation of FFD ensures continuity and smoothness of the surfaces up to a certain degree, the reconstruction losses computed when training the autoencoder do not consider any global aspect of the geometry. Therefore, when the latent representation is driven to a region that is not represented in the training set, the reconstruction of the point clouds present a higher level of noise and collapse in more extreme cases.

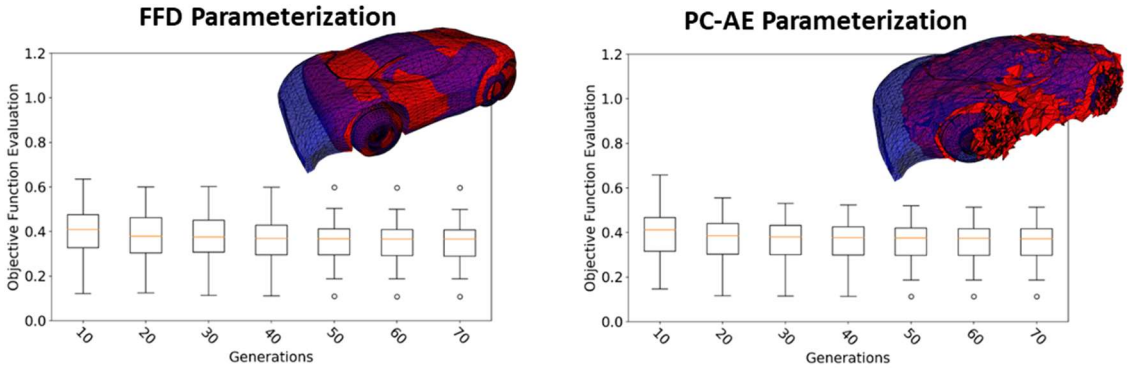


Figure 10 - Comparison between the normalized objective function at different generations and mesh reconstruction of the fittest individual obtained with the FFD and autoencoder representations

From our experiments, we concluded in general that the latent representation obtained with 3D point cloud autoencoders has the potential to increase the speed and quality of results in shape optimization problems. Nevertheless, the feasibility of the shapes generated during the process can be limited to the region in the latent space learned during the training phase, which hinders the generation of uncommon or novel shapes. It also impacts the selection of software components for the overall framework proposed for the project, which should also consider algorithms for detecting infeasible shapes for downstream engineering tasks. In addition, meshes should be recovered from autoencoder-based point clouds in the general case, when the point clouds in the data set do not share a common underlying structure, such as with isomorphic meshes.

3.1.4 Scalability of 3D point cloud autoencoders

Another aspect of the architecture and training algorithm that we analyzed was the scalability to higher-dimensional CAE models [15]. In the literature, the work on 3D point cloud autoencoders typically deals with point clouds that range between 1024 and 4096 points, while complex engineering meshes can easily comprise millions of nodes. Hence, in this second study, we formulated as objectives to define the maximum point cloud size that is supported for training the proposed architecture, and to propose a re-sampling strategy to reduce the dimensionality of the point clouds prior to the training, such that higher-dimensional models can be learned. The

hardware available for the experiments was a machine with two Intel Xeon CPUs, clocked at 2.10 GHz (16 cores, times 2 hyperthreaded), using a single Nvidia Quadro RTX 8000 GPUs (48 GB).

For determining the maximum point cloud size, we performed a run-to-crash experiment, where we iteratively started the training algorithm for 20 epochs while gradually increasing the point cloud size, until the script stopped working. With this setup, we achieved a maximum point cloud size of 200000 points, with an average GPU memory usage of (45.38 ± 19.91)% and elapsed time of 450s for 10 epochs. Regarding the data set, we also adopted synthetic data generated from a parameterized plate (Figure 11), such that we could control the geometric features contained in the data and refine the models up to 25,000 nodes. When the run-to-crash experiment exceeded the number of points in the models, the sampling algorithm automatically allowed the replication of sampled points, since we focused on the computational demand rather than the accuracy of the model.

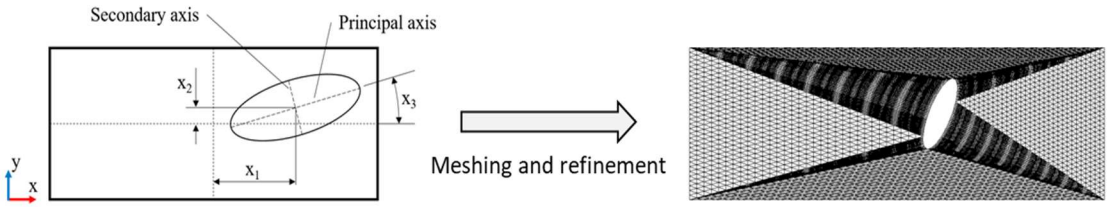
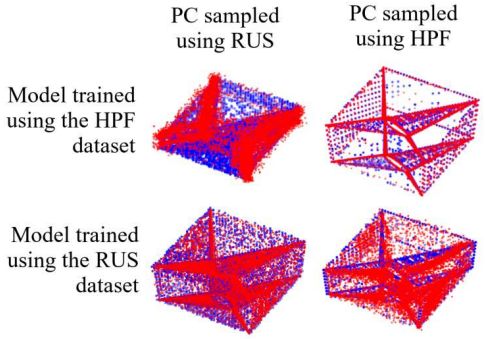


Figure 11 - Schematic of the parameterized plate used to generate the synthetic data set

For the re-sampling techniques, we considered two approaches: the random uniform sampling (RUS) and a high-pass filter (HPF) technique. In the RUS approach, all mesh vertices have the same sampling probability, therefore, it does not favor any geometric feature, but the regions with dense mesh refinement. The HPF technique was based on graph filtering proposed in [17], which increases the probability of sampling vertices close to severe shape transitions, such as corners, operating as an edge detector. The underlying motivation was that the HPF would yield a skeleton-like structure that could be used to differentiate the shapes and thus improve the training of the autoencoder. In order to account for the effects of the architecture and point cloud size, we considered three point cloud sizes (2048, 4096 and 8192) and three sizes of latent spaces (8, 16 and 32), and evaluated the mean values of the Chamfer Distance computed on a test set with 100 shapes, sampling with RUS and HPF techniques.

Analyzing the losses for training and testing data sets, we observed differences, both with respect to the sampling scheme and the results obtained with the previous data set. First, the models needed to be trained for more epochs and with data set augmentation to achieve the same range of reconstruction loss as in the previous study, even though the shapes in the data set had a lower degree of complexity and shared many geometric characteristics. Second, RUS led to better performance on the test set, but the advantage over HPF was reduced with increasing dimensionality of the point clouds, which is expected, since the point clouds sampled with HPF became more similar to the models sampled with RUS. However, when testing on the shapes sampled with alternative techniques, the model trained on point clouds sampled with RUS outperformed the HPF sampling in most of the cases, especially with lower dimensional latent spaces and point clouds (Figure 12). Our conclusion was that RUS yields point clouds with more

information about the global structure of the geometry, while HPF concentrates on specific features, which reduces the generalization capability of the model.



LR size	PC size	$CD_{RUS,HPF}$	$CD_{HPF,RUS}$
8	2048	**(4.28 ± 0.56)E-04**	(5.72 ± 1.50)E-04
	4096	**(3.55 ± 0.48)E-04**	(4.10 ± 1.30)E-04
	8192	**(3.98 ± 0.52)E-04**	(4.44 ± 1.00)E-04
16	2048	**(4.07 ± 0.92)E-04**	(5.87 ± 1.60)E-04
	4096	(3.92 ± 0.66)E-04	**(3.60 ± 0.96)E-04**
	8192	**(3.91 ± 0.49)E-04**	(11.7 ± 1.4)E-04
32	2048	(4.07 ± 0.96)E-04	**(4.00 ± 0.49)E-04**
	4096	(3.92 ± 0.66)E-04	**(3.46 ± 1.10)E-04**
	8192	(3.91 ± 0.49)E-04	**(3.45 ± 0.41)E-04**

Figure 12 - Reconstruction of geometries using the architectures trained on different data sets. The blue markers indicate the points of the input point cloud. In the table, CDi,j indicates the Chamfer distance of the model trained on data set i and tested on data [18]

Third and last, we analyzed the Pearson correlation coefficient between the parameters used to generate the data set and the latent variables of the model trained on point clouds with 8192 points and an 8-dimensional latent space (Figure 13). Although we expected the model trained on point clouds sampled with HPF to yield higher correlation values, there was no major difference between the results obtained for both sampling schemes. Furthermore, both cases indicate redundancies in the correlation with data set parameters, which could hinder the performance of the representation in shape optimization problems.

Figure 13 - Pearson correlation between the design variables x_i and latent variables (LV) obtained from the network trained on the RUS (left) and HPF dataset (right).

3.1.5 Feature visualization in 3D point cloud autoencoders

As a consequence of our results of the scalability study, and in order to evaluate and understand effects of changes in the point cloud architecture, we proposed a method for visualizing the features learned by the point cloud autoencoder [19]. We based our proposal on the properties of the 1D convolution, which operates in a point-wise fashion and thus allows for a straightforward mapping of the input points to the corresponding activations in the layers of the encoder (Figure 14). Here,

we defined a network feature as the set of activated values of neuron j at a layer ℓ for all points in the input point cloud, which corresponds to a column of the heatmap shown in Figure 14. In order to visualize a network feature, we projected the corresponding activations as colormaps onto the 3D scatter plot of the input point cloud that yielded the activations. Our expectation was that the activation would highlight geometric patterns in the point clouds, similarly to the features revealed in the layers of image processing networks. Then we could use these to interpret the geometric characteristics encoded in the latent layer.

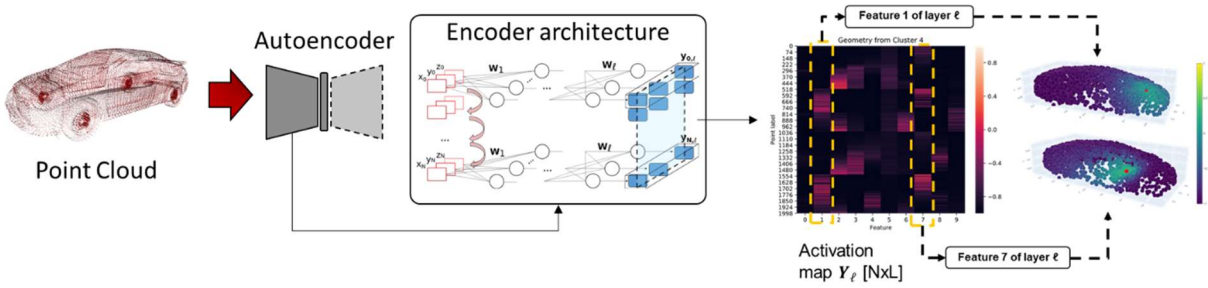


Figure 14 - Workflow of the proposed method for visualizing network features as colormaps projected onto the 3D representation input point cloud.

For the purposes of this study, we limited our analyses to the last convolutional layer of the encoder. Instead of learning higher-level features, the latent representations are extracted immediately after this layer with a maximum pooling operation. Therefore, the higher the activation value in a feature j, the more important is the corresponding input point for that feature. This makes the interpretation of the latent variables easier. For our experiments, we used the shapes in the car class of the ShapeNetCore repository [4], sampled with 2048 points, trained with the same hyperparameters of the aforementioned validation experiment (3.1.2) and for three sizes of the latent space: 2 (LR2), 10 (LR10) and 20 (LR20). We also only considered geometries with reconstruction losses close to the most frequent value obtained for the training set, avoiding over- and underfitted samples. We clustered the remaining geometries based on their latent representations, such that we could identify patterns of co-activations, easing the interpretation of the features.

At first, we visualized the features calculated with model LR2 for four shapes sampled from different clusters (Figure 15). We observed that the activations highlighted similar regions in the input space rather than revealing any complex structures, e.g. wheels or side mirrors. Our interpretation was that the features that yield the latent variables map the occupancy of the input space and not geometric patterns, which was what we initially expected.

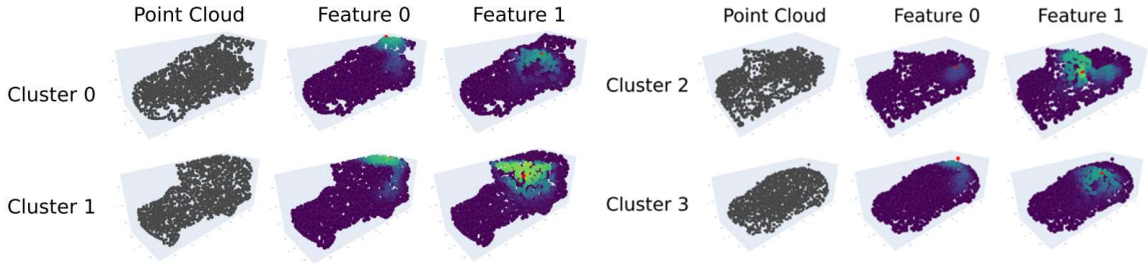


Figure 15 - Visualization of the features calculated with model LR2 for shapes sampled from different clusters in the latent space. The brightest colors indicate higher activation values.

The features visualized for model LR2 fit the characteristics of the 1D convolution, which processes the points individually and thus does not capture large-scale geometric properties. In order to confirm that the latent variables map the occupancy of the input space, we performed a second experiment, where we gradually shifted and rotated a single geometry in the input space and visualized the features for different positions (Figure 16). While the shape was moving, the activated region in the point clouds remained static with respect to the input space, confirming our hypothesis.

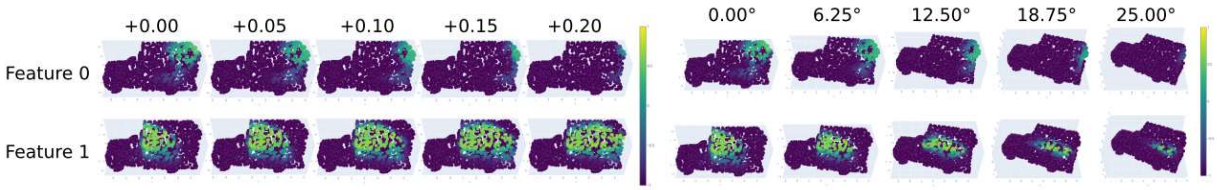


Figure 16 - Visualization of the features obtained for a car shape shifted in the x-direction (left) and rotated around the vertical axis (right) with different steps as input and using model LR2. The brightest colors indicate higher activation values.

Additionally, we performed transfer of network features obtained for different shapes considering the model with a latent size of 10 (LR10). Our goal was to verify the correspondence between the transfer of geometric and network features, and we observed that this assumption is only valid for cases where the shapes share similar occupied regions in the input space. Otherwise, the decoder fails to reconstruct smooth car-like shapes, which indicated that the decoder recovers the Cartesian coordinates of the points based on the combination of occupied regions in the input space. When the transferred network features yielded the representation of an uncommon distribution of points, compared to the training data, it forced the decoder to extrapolate the learned features, which led to fuzzy point cloud reconstructions.

Finally, we analyzed the visualization of combined network features, calculated on a dense uniform lattice, and compared them to the most frequently occupied regions in the training data (Figure 17) calculated using a Kernel Density Estimator (KDE) model [20]. Our expectation was that the combination of features would reveal similar results to the KDE, since the features map the occupancy of the input space. However, the analysis revealed that the features defined nearly the boundary of the most frequent car shape with the model LR2, and gradually highlighted other regions of the input space with increasing number of latent variables. Our interpretation was that for a more intense reduction of dimensionality, as for model LR2, the autoencoder learns

preferably how to differentiate the shapes, which yields the observed boundary in Figure 17. When the dimensionality of the latent space increases, it allows the encoder to describe the shapes as a combination of finer regions and thus other regions of the input space become activated.

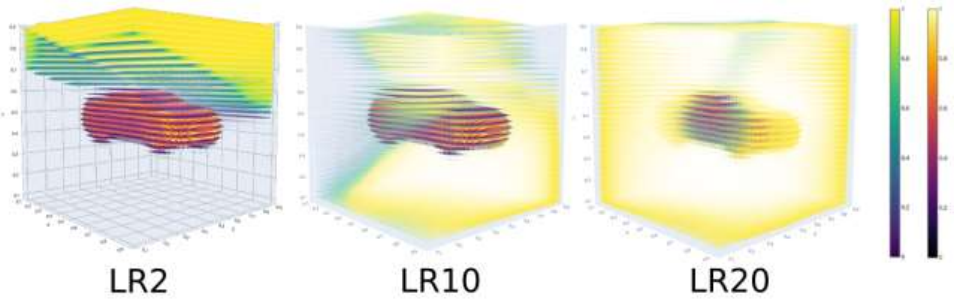


Figure 17 - Visualization of the KDE score (central car shape) and combined features of a dense uniform lattice in the input space. For visualization purposes, the size of the markers varied with the value of the metrics, redundant to the colors. In the representations, the lightest color indicates the highest values for both metrics.

3.1.6 3D point cloud autoencoders software components

Besides the scientific publications, the work on the 3D point cloud autoencoder yielded two important software modules for the ECOLE project. First, we developed a reliable algorithm for training and testing a 3D point cloud autoencoder, which can be implemented in engineering analyses for learning features for dimensionality reduction or as a shape generative model. Second, we provided a method for visualizing the features learned by the encoder, which provides an interpretation of the information abstracted by the autoencoder and can guide future modifications in the architecture. The current work focuses on two aspects that are central to the implementation of the autoencoder in engineering optimization pipelines. On the one hand, we aim at recovering meshes from autoencoder-based point clouds, when the data set does not comprise isomorphic meshes, as in the ShapeNetCore repository, since most of the CAE simulations require meshed representations of the shapes. On the other hand, our objective is to quantify the information contained in the latent variables on engineering performance metrics, such as aerodynamic forces, so that we can tailor the latent space for specific engineering optimization problems. Our implementation of the point cloud autoencoder and how to use it is detailed in the Appendix A.

3.2. 3D point cloud variational autoencoder (PC-VAE)

3.2.1. Details on the 3D point cloud variational autoencoder architecture

The VAE [21], [22] is a generative model that, opposed to a standard AE, aims at learning "disentangled, semantically meaningful, statistically independent and causal factors of variation in data" [23]. The VAE may be seen as a regularized version of the AE that forces the learned latent space towards following an a-priori specified distribution. Prior research on VAEs which are closest to our goal of generating novel and diverse shapes is CompoNet [24], a generative neural network for 2D or 3D shapes based on a part-based prior, which relies on a VAE for 2D

shape synthesis and the above mentioned AE+GMM for 3D objects. Zamorski et al. [25] proposed an end-to-end solution to generate 3D shapes with an adversarial autoencoder (AAE) for 3D point clouds using a binary representation in the latent space. The AAE differs from a VAE in the loss computed on the latent space, where the AAE uses the adversarial loss similar to a generative adversarial network (GAN) while the VAE uses the Kullback-Leibler (KL) divergence to enforce regularization of the latent space. In our work, we target on the one hand the extraction of features from 3D shape data sets in an unsupervised way and on the other hand the generation of diverse solutions for further simulation and analysis in a vehicle design framework. To fulfill both targets, we propose a variational autoencoder for 3D point clouds (PC-VAE).

The PC-VAE used in our experiment is implemented based on an architecture presented in [8], [13], which extends a proposal in [10], [15] (Figure 18). The encoder part of the PC-VAE follows [11], [26], who propose to use 1D-convolutional layers together with permutation-invariant global operators (e.g., max-pooling at a deeper layer of the network) in order to make the architecture invariant against permutations in the input point clouds. The encoder-decoder structure used here is similar to the architecture proposed in [11], only the last layer of the decoder is replaced with sigmoid activation functions [10] to normalize the coordinates of all points to the range [0.1,0.9]. In our encoder, we use five 1D-convolutional layers, each followed by a ReLU [27] and a batch normalization layer. The decoder consists of three fully connected layers.

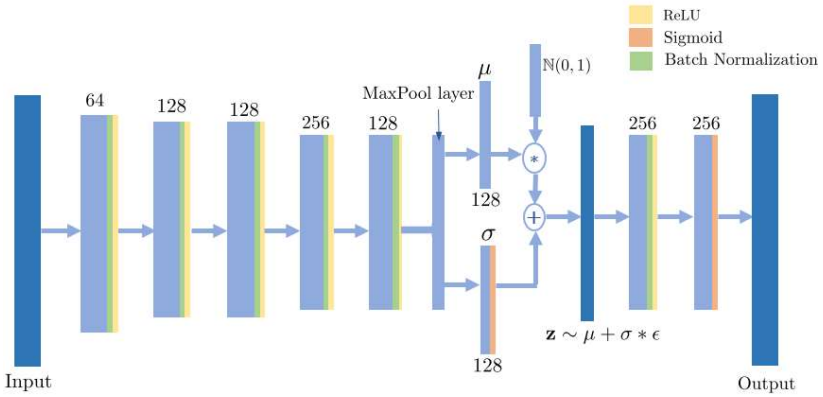


Figure 18 - Overview of the generative PC-VAE for generating designs, shape analysis and engineering optimization.

To modify the autoencoder architecture in [10], [11], [15] to become variational autoencoders, the output of the last convolution layer in the encoder is passed to a max-pool layer that produces a k-dimensional vector that forms the bottleneck for two separate k-dimensional vectors: a mean vector μ and a standard deviation vector. The mean vector has no activation function, while the deviation vector uses a sigmoid activation function. A vector sampled from the latent distribution is fed into the decoder network for reconstruction of a point cloud and is represented as a vector $z \sim \mu + \sigma * \varepsilon$, where $\epsilon \sim N(0,1)$. The PC-VAE minimizes the loss function between the input point cloud, S_1, and the reconstructed point cloud, S_2, as

$$\mathcal{L}_{\mathcal{VAE}}(S_1, S_2) = \alpha\, d_{CD} + \beta \mathcal{D}_{KL}\,(q(z|S_1)\,||p(z)) \qquad (1)$$

where the first term on the right-hand side denotes the reconstruction loss, measured by the Chamfer distance (CD).

$$d_{CD}(S_1, S_2) = \sum_{x \in S_1} \min_{y \in S_2} |x - y|_2^2 + \sum_{y \in S_2} \min_{x \in S_1} |x - y|_2^2 \qquad (2)$$

and the second term is the Kullback-Leibler (KL)-divergence that quantifies the distance between the learned latent representation and the prior. Both terms of the loss function differ by several orders of magnitude. To bridge this gap, two parameters α and β are introduced to scale the reconstruction loss and KL-divergence, respectively.

We tested our approach on an autoencoder trained on the car class from ShapeNetCore [14], varying the dimensionality of the latent space. We concluded that the layers of the encoder map the occupancy of the input space rather than complex geometric features [15], [19]. Tuning the parameters of the loss functions is an important aspect to overcome mode collapse in VAE and also to improve reconstruction of the geometries. We used a hyperparameter optimization technique (OPTUNA) [28] to tune α and β in equation (1). We also used a grid search and arrived at parameter values of α = 250 and β = 0.001, which resulted in an acceptable trade-off between the reconstruction accuracy and divergence in the latent space. The workflow to train a PC-VAE and python scripts are explained in Appendix B.

3.2.2 Validation of the 3D point cloud variational autoencoder architecture

To validate the implementation and function of our proposed PC-VAE model, we compared its generative performance to that of existing models as reported in the literature, specifically a regular AE [11] and a 3D-Adversarial AE (3dAAE-G) [25]. As no prior work was based on 3D car shapes, we re-trained our model on the chair class of the ShapeNet data set, split into 85 % training, 5 % validation, and 10 % test-split to match the settings to the reference models from the literature. We calculated the MMD-CD between the reconstructed point clouds and their corresponding ground truth in the test data set of the chair object class (Table 1). The resulting MMD-CD values indicate that our model is comparable to existing models regarding the ability to encode and reconstruct the data set, so that in a next step, we can evaluate the model's generative capabilities with respect to realism and novelty of generated shapes.

Table 1: Comparison of the reconstruction capability between our PC-VAE and reference models.

Methods	**MMD-CD**
PC-VAE	.0008
AE [10]	.0011
3dAAE [20]	.0008

3.2.3 3D point cloud variational autoencoder for generating realistic and novel shapes

Once we have our trained model, we aim at generating both realistic and novel shapes, as both aspects are central to application in the engineering domain. In other words, a model should generate plausible shapes, i.e., shapes that in some aspects closely resemble the shapes in the dataset, but also novel or diverse shapes that are sufficiently distinct from the shapes used for training. The trained generative PC-VAE (defined in section 3.2.1) can be used to generate a large amount of design samples either by sampling from the latent space or by interpolating in the latent space (as shown in Figure 19). To use a generative model in the engineering design process, the model has to be able to generate realistic and novel shapes. In our research, we therefore explored methods to enforce the generation of such realistic and novel shapes. From the trained PC-VAE we generate a set of new shapes ($\boldsymbol{G}$), either by interpolating in the latent space or by random sampling of the learned distribution in the latent space. Further, we explained methods to evaluate realism and novelty of this generated shapes.

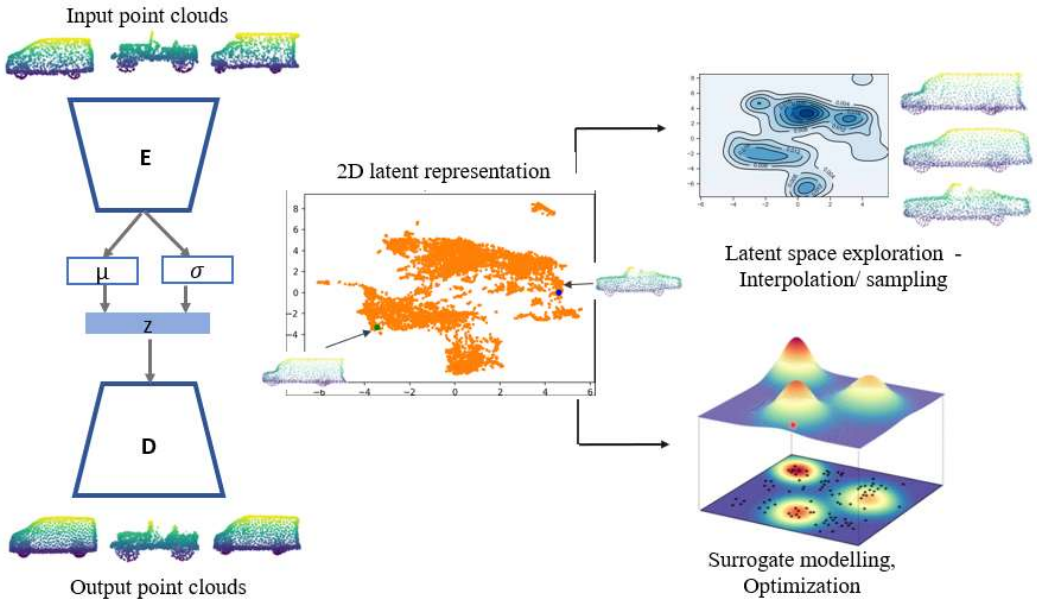


Figure 19 - Application of PC-VAE for design generations and evaluations.

Evaluating the realism of the shapes - To quantify the "realism" of a generated shape, Berthelot et al. [29] proposed a mean distance (MD) metric in 2D that compares the minimum cosine distance of the interpolated data-points with the original data-points by finding the nearest neighbor of each interpolation step in the training set. Similar to this approach, Achlioptas et al. [11] proposed a minimum matching distance (MMD) metric to measure the closeness of two-point cloud sets using the Chamfer distance (CD) [13], where closeness between sets is assumed to indicate a higher realism. Besides following the same idea as the MD metric, closeness measures the distance of each point in a point cloud set to its nearest neighbor in another set using the Chamfer distance (CD). This measure is similar to the mean distance (MD) measure but uses Chamfer distance (CD) to measure closeness between two-point cloud sets. So, we used this approach to measure the realism of the generated shapes using PC-VAE. To also quantify realism,

we propose to employ the method introduced in [11], [29], who measure the feasibility of shapes by the mean distance between the shapes generated through interpolation in the latent space (refer to as set $\boldsymbol{G}$) and the shapes in dataset $\boldsymbol{S}$.

We hypothesized that the PC-VAE due to the applied regularization learns a smoother latent space compared to the AE model, which should lead to the generation of more realistic car shapes when reconstructing random or interpolated samples from the latent space of the PC-VAE compared to the AE [30]. To test this hypothesis, we compared reconstructions from a 10-step linear interpolation between 50 randomly selected pairs of car geometries from the data set S using both our PC-VAE and the AE (Figure 20). Each interpolation pair consisted of an initial shape A and a target shape C, as well as an intermediate shapes B, which were reconstructed from the interpolation in the latent space.

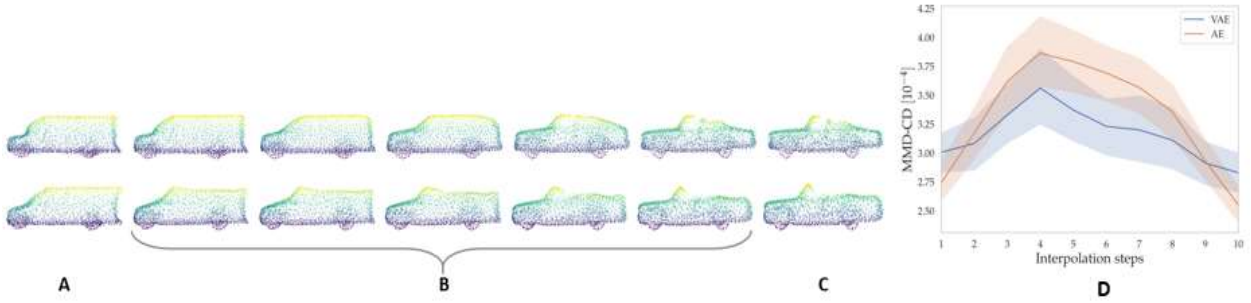


Figure 20 – (A): Initial shape. (B): Interpolation between shapes (A) and (C) in 10 steps—reconstruction of the interpolation at steps 2, 4, 6, 8, and 10. (C): Target shape. Top row: Reconstruction of the interpolation using the proposed PC-VAE. Bottom row: Reconstruction of the interpolation using the proposed PC-VAE. Bottom row: Reconstruction of the interpolation using the AE. (D): MMD-CD measure for 50 randomly selected car pairs (each with 10 interpolation steps) using AE and PC-VAE (shaded areas indicate the standard deviation).

We first evaluated the generated shapes qualitatively by visual inspection. We show one example (Figure 20 (A), (B), (C)), where we observed a major difference in the roof region. A first visual inspection shows that the intermediate interpolations performed in the AE latent space lead to a slanted roof as well as to dents in the roof region of the car shape. The interpolation with the PC-VAE induces a more gradual change in the roof region and mostly maintained the curvature of the roof, in total leading to more realistic interpolated car shapes.

To also quantitatively assess the difference of the generative capability between AE and PC-VAE, we calculated the closeness (MMD-CD) between the samples generated for each interpolation and the samples in the complete dataset $\boldsymbol{S}$. Figure 20 D shows the MMD-CD over all 50 interpolations for both, AE and PC-VAE. A statistical analysis verified our results that interpolated shapes with PC-VAE generate lower MMD-CD compare to baseline AE, see further details in [30]. Overall, the PC-VAE showed a lower MMD-CD compared to the AE, indicating that the interpolation led to shapes that were more similar to the training set shapes and hence produced more realistic cars.

Identifying novel shapes using a classifier - Building a surrogate model on the latent representation defines a supervised learning problem to extract the relation between the input and output. Surrogate modelling yields an easier mechanism for predicting the system's response for a new previously unobserved input, which can be subsequently used to accelerate downstream tasks such as optimization loops and quantitative evaluations. Therefore, after testing the capability

of the PC-VAE to generate realistic shapes, we evaluated the diversity/novelty of the generated shapes using a classifier based surrogate model (as shown in Figure 21 (A)).

To quantify the diversity/novelty of generated shapes, $\boldsymbol{G}$, we used the diversity measure presented in [24]. Similar to the approach outlined there, we trained a binary classifier on the latent representation of the whole dataset $\boldsymbol{S}$, where the information whether a sample belonged to either the training set, $\boldsymbol{D_{train}}$, or test set, $\boldsymbol{D_{test}}$, was used as the label. The diversity is then calculated as the percentage of generated samples, $\boldsymbol{G}$, that are classified as belonging to the test set. In other words, the classifier quantifies the percentage of samples generated by the model that are closer to the unseen test set than to the training set. The higher the number of generated samples classified into the unseen test set, the higher the diversity of the model. Here, we used a multilayer perceptron (MLP) with two hidden layers [100, 60] and a *tanh* activation function for the hidden layers.

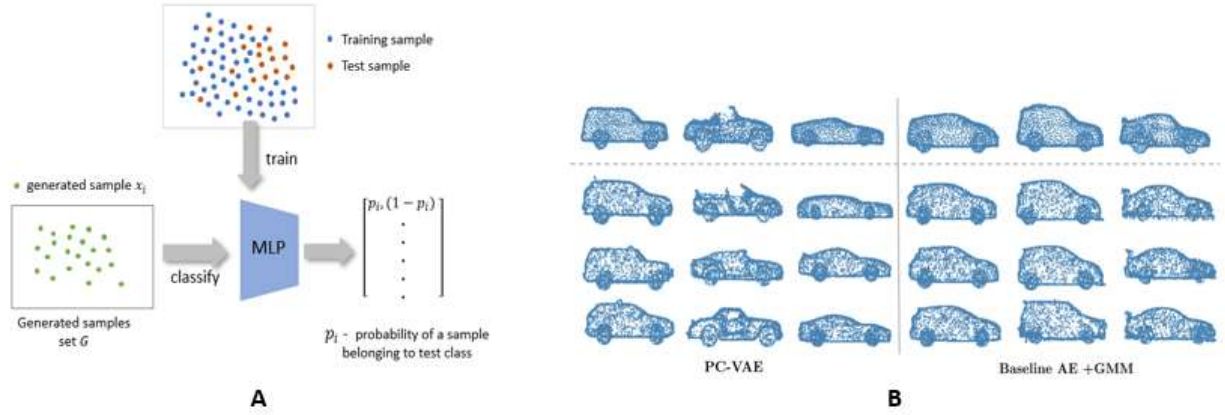


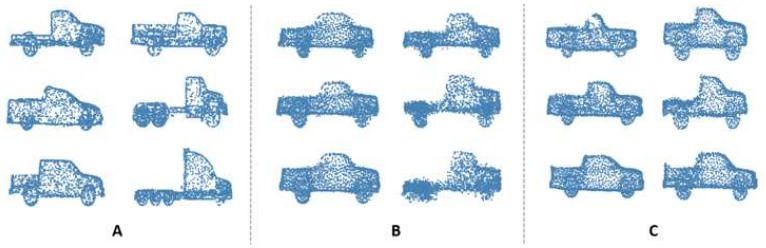
Figure 21 – (A): Schematic overview of the generative diversity evaluation. (B): Qualitative comparison of generative diversity of PC-VAE and baseline AE+GMM. Row 1: Three randomly selected shapes from the generated samples. Row 2-4: Nearest neighbors of the generated samples. Row 2-4: Nearest neighbors of the generated shapes in the training set (measured by CD).

Table 2: Comparing generative diversity of the PC-VAE and the AE+GMM (baseline). Best generative diversity for each train-test-split shown in bold.

Train- test split	AE+GMM	PC-VAE Encoder
50/50	40.3 ± 0.80	**58.19 ± 0.19**
70/30	21.7 ± 0.5	**26.96 ± 0.8**
90/10	4.0 ± 0.3	**6.5 ± 0.08**

We repeated the experiment ten times. The averaged percentage of generated samples classified as belonging to the test set are shown in Table 2. For all train-test-splits, the PC-VAE generated shapes with a higher diversity than the baseline model. For an additional, qualitative evaluation of the generative diversity of the models, we randomly selected three generated samples from each model and searched for the three nearest neighbors of each shape in the training set, $\boldsymbol{D_{train}}$ (Figure 21 (B)). Both models generated realistic shapes, however the PC-VAE generated shapes were more dissimilar to its nearest neighbors in $\boldsymbol{D_{train}}$ compared to the baseline model.

In [30], we also tested our architecture for its ability to generate a novel type of car shapes using the classification technique. This methodology helps to determine the model's capability to generate unseen novel shapes. We re-trained the PC-VAE and the baseline AE on the car class after manually excluding all pickup truck designs from the data. The pickup truck shapes present in the data set are a combination of convertibles and coupe-like car designs (see Figure 22 (A) for examples). We used the separated pickup truck shapes as the test set, $\boldsymbol{D_{test}}$, (630 shapes in total). We generated 1800 samples (three times the size of $\boldsymbol{D_{test}}$), $\boldsymbol{G}$, from the trained PC-VAE and baseline AE+GMM. We then retrained the MLP classifier on the new $\boldsymbol{D_{train}}$ and $\boldsymbol{D_{test}}$ to calculate the diversity of the generated shapes. Furthermore, we calculated two other quantitative measures: coverage and MMD-CD (for further details see [30]). All three measures showed that the PC-VAE was able to generate more truck-like and hence more novel shapes. Examples of generated shapes that were classified as belonging to $\boldsymbol{D_{test}}$, i.e., pickup trucks are shown in Figure 22 (B, C). Note that the trucks generated by the PC-VAE show a slightly higher quality.



D

Models	Generative diversity
AE+GMM	0.3 ± 0.13
PC-VAE Encoder	**3.1 ± 2.2**

Figure 22 – (A): Randomly sampled geometries from the test set consisting of pickup truck shapes only. (B): Shapes generated by the AE+GMM baseline model that were classified as belonging to the test set. (C): Shapes generated by the PC-VAE classified as belonging to the test set. (D): Comparing generative diversity, coverage and minimum matching distance (MMD-CD) for a subclass of car shapes on test split (best performance for each measure shown in bold.)

We want to emphasize that this experiment poses a significant challenge for the generative model, since we are testing for the generation of a shape that is significantly different from the training set, $\boldsymbol{D_{train}}$. This difficulty is also reflected in the small number of generated shapes that are classified as truck-like (3% of 1800 generated samples, Figure 22 (D)). Nevertheless, both models surprisingly showed the ability to generate such novel shapes, not seen in the training set, where the PC-VAE generated shapes of higher quality.

Enforcing generation of novel shapes - To use a generative model in the engineering design process, the model has to be able to generate realistic and novel shapes. In our research, we therefore explored methods to enforce the generation of such realistic and novel shapes. After training the classifier (as mentioned in Section 3.2.3) we performed an optimization on the input to the classifier that was aimed at generating novel samples that are classified as belonging to $\boldsymbol{D_{test}}$, i.e., are considered novel or diverse. Reconstructing the latent representations returned by the optimization process leads then to the generation of novel shapes in a targeted fashion. However, for the optimization, we trained an additional Gaussian process model (GP) on the latent space representation and the corresponding output probabilities of the MLP classifier for each shape. We trained an additional model to make use of gradient-based methods that are typically faster. Hence, we used a differentiable model to map from the latent representation to the classification results.

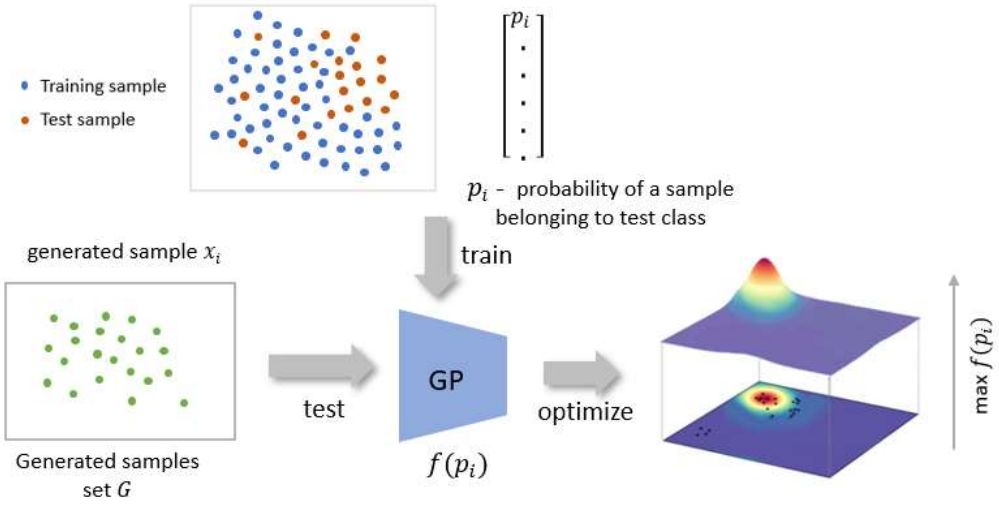


Figure 23 - Schematic overview of using gradient based optimization for novel shapes generations.

We trained the GP on top of the classification results to apply the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm to generate shapes that maximize the GP output, i.e., the probability of classifying a latent sample as belonging to $\boldsymbol{D_{test}}$. BFGS is an iterative method for solving unconstrained non-linear optimization problems efficiently. Figure 23 shows a schematic overview of our proposed optimization approach. In sum, we here propose an approach that is a combination of the exploratory ability of the PC-VAE, and a classifier and optimization applied to the learned latent space. While we make use of the explorative and generative capabilities of the PC-VAE, we utilize an additional model and optimization to identify and guide the generation of shapes towards more novel designs. Our implementation of the point cloud variational autoencoder and how to use it is detailed in Appendix B.

4. Surrogate assisted optimization and evaluation

4.1. Improving vehicle design classification

As stated in the introduction, simulations of vehicle bodies, like computational fluid dynamics (CFD) for aerodynamic performance, are typically very time-consuming and costly. Therefore, it is important in any multi-criteria optimization to reduce the number of function evaluations. Machine learning methods allow based on given data sets to identify solution proposals, which are likely to have a low performance and may not be considered for costly simulations. However, it is often the case that the data is imbalanced, which means that the ratio between the sample number belonging to different classes is very large. The classification problem under class imbalance has caught growing attention from both the academic and industrial field. Recent advances in data storage and management as well as in data science enables practitioners from industry and engineering to collect a large amount of data with the purpose of extracting knowledge and acquire hidden insights. An example may be illustrated from the field of computational design optimization, where product parameters are modified to generate digital prototypes, which performances are evaluated by numerical simulations or based on equations expressing human

heuristics and preferences. Since the ratio between valid and invalid designs is likely to favour valid designs, we will typically have an imbalanced data set. Under this circumstance, a database would contain a large number of designs which are valid (even if some may be of low performance) and a smaller number of designs which violate pre-defined product requirements. In our study, we set up several experiments on our real-world inspired vehicle dataset to study whether performing oversampling techniques can improve vehicle design classification. Detailed information on the dataset is available in [31] and in the ECOLE deliverable report D1.1, Section 2.

We briefly review six powerful oversampling approaches, including both “classical" ones (SMOTE, ADASYN, MWMOTE) and new ones (RACOG, wRACOG, RWO-Sampling) [32], [33], [34], [35], [36]. The six reviewed oversampling techniques can be divided into two groups according to whether they consider the overall minority class distribution. Among the six approaches, RACOG, wRACOG, and RWO-Sampling consider the overall minority class distribution while the other three do not. Due to the limited space, we only introduce the working procedure of the SMOTE, ADASYN, ROCOG and wRACOG here, the working procedure of the other two oversampling techniques are described in [31].

SMOTE and ADASYN - The synthetic minority oversampling technique (SMOTE) is the most famous resampling technique [32]. SMOTE produces synthetic minority samples based on the randomly chosen minority samples and their *K*-nearest neighbors. The new synthetic sample can be generated by using the randomized interpolation scheme above for minority samples. The main improvement in the adaptive synthetic (ADASYN) sampling technique is that the samples which are harder to learn are given higher importance and will be oversampled more often in ADASYN [33].

RACOG and wRACOG - The oversampling approaches can effectively increase the number of minority class samples and achieve a balanced training dataset for classifiers. However, the oversampling approaches introduced above heavily reply on *local* information of the minority class samples and do not take the overall distribution of the minority class into account. Hence, the *global* information of the minority samples cannot be guaranteed. In order to tackle this problem, Das et al. [35] proposed RACOG (RApidy COnverging Gibbs) and wRACOG (Wrapper-based RApidy COnverging Gibbs).

In these two algorithms, the *n*-dimensional probability distribution of the minority class is optimally approximated by Chow-Liu's dependence tree algorithm and the synthetic samples are generated from the approximated distribution using Gibbs sampling. Instead of running an “exhausting” long Markov chain, the two algorithms produce multiple relatively short Markov chains, each starting with a different minority class sample. RACOG selects the new minority samples from the Gibbs sampler using a predefined *lag* and this selection procedure does not take the usefulness of the generated samples into account. On the other hand, wRACOG considers the usefulness of the generated samples and selects those samples which have the highest probability of being misclassified by the existing learning model [35], [37].

The original idea of the geometric data for engineering applications has been provided in D1.1 Section 2. Hence, we only emphasize the synthetic dataset generation procedure here. For the experiments we adopted the computational fluid dynamics (CFD) simulation of a configuration of the TUM DrivAer model [5]. The simulation model is deformed using the FFD algorithm with a lattice with 7 planes in x- and z-directions, and 10 in y-direction (Figure 24). The planes closer to the boundaries of the control volume are not displaced in order to enable a smooth transition from the region affected by the deformations to the original domain. Assuming symmetry of the shape with respect to the vertical plane (xz) and assuming deformations caused by displacement of entire control planes only in the direction of their normal vectors, we define a design space with nine parameters. To generate the data set, the displacements x_i were sampled from a random uniform distribution and constrained to the volume of the lattice, allowing the overlap of planes.

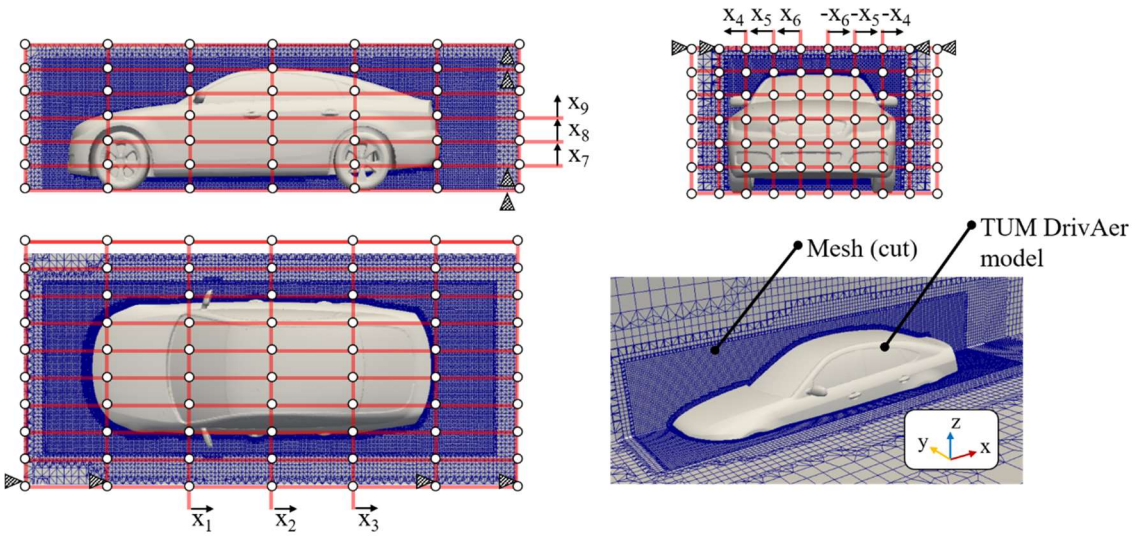


Figure 24 - Free form deformation lattice used to generate the data set for the experiments.

The initial mesh was generated using the algorithms *blockMesh* and *snappyHexMesh* of **OpenFOAM**. We automatically generated 300 meshes based on the FFD algorithm implemented in Python and evaluated them using the **OpenFOAM** *checkMesh* routine. In the process, six meshes were discarded due to errors in the meshing process. The metrics used to define the quality of the meshes were the number of warnings raised by the *checkMesh* algorithm, the maximum skewness and maximum aspect ratio. We manually labelled the feasible meshes according to the rules shown in Table 3. The imbalance ratios after manual labelling are also given in Table 3. Please note that the input attributes are exactly the same for all three sets of datasets, only the "class" labels are different.

Table 3: Feasible meshes labelling rule.

Dataset	#Attribute	#Sample	#Warnings	Max skewness	Max aspect ratio	IR
Set1	9	294	<4	<6	<10	6.35
Set2	9	294	<4	<6.2	<10.5	2.54
Set3	9	294	<2	<5.8	<10.3	12.36

The experimental results on the digital vehicle dataset are given in Table 4. We find that applying the oversampling techniques can improve the performance by around 10% for our digital vehicle datasets.

Table 4: Experimental Results (AUC) on Digital Vehicle Dataset (with Decision Tree).

Dataset	Baseline	SMOTE	ADASYN	MWMOTE	RACOG	wRACOG	RWO
Set1	0.7786	0.8412	0.8315	0.8354	**0.8543**	0.8406	0.8502
Set2	0.6952	0.7575	0.7560	**0.7651**	0.7614	0.7421	0.7452
Set3	0.6708	0.7780	0.7792	0.7660	**0.7823**	0.7534	0.7743

In this work, we reviewed six powerful oversampling techniques, including "classical" ones (SMOTE, ADASYN and MWMOTE) and new ones (RACOG, wRACOG and RWO-Sampling), in which the new ones consider the minority class distribution while the "classical" ones do not. The six reviewed oversampling approaches were tested on 19 benchmark imbalanced datasets (not described in this report, see [31]) and an imbalanced real-world inspired vehicle dataset to investigate their efficiency.

In our experiment, in most cases, oversampling approaches which consider the minority class distribution (RACOG, wRACOG and RWO-Sampling) perform better. For both the benchmark data sets and our real-world inspired data set, RACOG performs best and MWMOTE second best. The application of the imbalanced learning techniques, i.e., oversampling approaches, to the vehicle design classification problem has shown to be a success with an improvement of classification performance by around 10%. The scripts and workflow for vehicle design classification are explained in more detail in Appendix C.

5. Summary and Outlook

This report introduces some essential components for multi-criteria design optimization, namely unsupervised learning methods of features representing 3D designs and a classification technique to minimize computational costs of design simulations. We provided details on our developed point cloud (variational) autoencoder architectures and our research results based on both implementations. We reported on validation results and introduced potential applications in multi-criteria design optimization. Along with the software details given in the following Appendices, other researchers are enabled to utilize our code to compare their own methods with our results or to extend our frameworks for solving additional research questions. We have also shown how classification of imbalanced data supports the optimization and reduces (computational) costs to fulfill constraints. Future work will cover the application of this method to the features, i.e., latent parameters of car models, which are extracted by our autoencoders in an unsupervised fashion.

In the following Appendix, the details on how to use our software are provided. According to the ECOLE data management plan document (D5.3), the software for our (variational) autoencoder is classified as restricted, i.e., the software can be accessed via the Bear Data sharing repository of the University of Birmingham (https://beardatashare.bham.ac.uk/login) after registration has been requested and completed, or by contact through the form provided on the ECOLE webpage: https://ecole-itn.eu/contact/.

The software for the classification of imbalanced data is available on Zenodo. The link is given in Appendix C.

Bibliography

[1] S. Menzel and B. Sendhoff, "Representing the change - Free form deformation for evolutionary design optimization," *Studies in Computational Intelligence,* p. 63–86, 2008.

[2] M. M. Bronstein, . J. Bruna, Y. Lecun, A. Szlam, . P. Vandergheynst and C. V. May, "Geometric deep learning : going beyond Euclidean data," *IEEE Signal Processing Magazine,* vol. July, pp. 1-22, 2017.

[3] T. Friedrich, N. Aulig and S. Menzel, "On the potential and challenges of neural style transfer for three-dimensional shape data," in *International Conference on Engineering Optimization*, Springer International Publishing, 2019, p. 581–592.

[4] 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.

[5] M. Berger, A. Tagliasacchi, L. M. Seversky, P. Alliez, J. a. Levine, A. Sharf, A. Tagliasacchi, L. M. Seversky and J. a. Levine, Joshua a., "State of the Art in Surface Reconstruction from Point Clouds," in *Proceedings of the Eurographics 2014*, 2014.

[6] A. I. Heft, T. Indinger and N. A. Adams, "Experimental and Numerical Investigation of the DrivAer Model," in *American Society of Mechanical Engineers, Fluids Engineering Division (Publication) FEDSM*, 2012.

[7] T. Rios, S. Menzel and B. Sendhoff, "Engineering Data and Descriptors," Experience-based COmputation: Learning to optimisE (ECOLE) - Deliverable D1.1, 2020.

[8] Y. Guo, H. Wang, Q. Hu, H. Liu and M. Bennamoun, "Deep learning for 3D Point Clouds: A Survey," *arXiv preprint,* vol. arXiv:1912.12033[cs.CV], 2019.

[9] S. Saha, T. Rios, S. Menzel, L. L. Minku, X. Yao and B. Sendhoff, "Optimal Evolutionary Optimization Hyper-parameters to Mimic Human User Behavior," in *IEEE Symposium Series on Computational Intelligence (SSCI), Xiamen, China, 2019,*.

[10] T. Rios, B. Sendhoff, S. Menzel, T. Back and B. V. Stein, "On The Efficiency of a Point Cloud Autoencoder as a Geometric Representation for Shape Optimization," in *IEEE Symposium Series on Computational Intelligence, SSCI*, 2019.

[11] 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.

[12] D. P. Kingma and J. Ba, "Adam: A Method for Stohastic Optimization," in *3rd International Conference on Learning Representations (ICLR 2015)*, San Diego, 2015.

[13] 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.

[14] 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.

[15] T. Rios, P. Wollstadt, B. V. Stein, T. Back, 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 2019*, Xiamen, China, 2019.

[16] 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.

[17] S. Chen, D. Tian, C. Feng, A. Vetro and J. Kovačević, "Fast Resampling of Three-Dimensional Point Clouds via Graphs," *IEEE Transactions on Signal Processing,* vol. 66, no. 3, pp. 666-681, 2018.

[18] T. Rios, P. Wollstadt, B. V. Stein, T. Back, 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,China, 2019.

[19] T. Rios, B. V. 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.

[20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and É. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research,* 2011.

[21] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *2nd International Conference on Learning Representations, ICLR*, 2014.

[22] D. J. Rezende, S. Mohamed and D. Wierstra, "Stochastic Backpropagation and Approximate Inference," in *31st International Conference on International Conference on Machine Learning*, 2014.

[23] D. P. Kingma and M. Welling, "An introduction to variational autoencoders," *An introduction to variational autoencoders,* vol. 12 , no. 4, 2019.

[24] N. Schor, O. Katzir, H. Zhang and D. Cohen-Or, "CompoNet: Learning to generate the unseen by part synthesis and composition," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019.

[25] M. Zamorski, M. Zięba, P. Klukowski, R. Nowak, K. Kurach, W. Stokowiec and T. Trzciński, "Adversarial autoencoders for compact representations of 3D point clouds," *Computer Vision and Image Understanding,* p. 193, 2020.

[26] C. R. Qi, H. Su, K. Mo and L. J. Guibas, "PointNet: Deep learning on point sets for 3D classification and segmentation," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2017.

[27] A. L. Maas, A. Y. Hannun and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, 2013.

[28] T. Akiba, S. Sano, T. Yanase, T. Ohta and M. Koyama, "Optuna: A Next-generation Hyperparameter Optimization Framework," in *Proceedings of the 25th International Conference on Knowledge Discovery and Data Mining*, 2019.

[29] D. Berthelot, I. Goodfellow, C. Raffel and A. Roy, "Understanding and improving interpolation in autoencoders via an adversarial regularizer," *7th International Conference on Learning Representations, ICLR,* pp. 1-20, 2019.

[30] S. Saha, S. Menzel, L. L. Minku, X. Yao, B. Sendhoff and P. Wollstadt, "Quantifying The Generative Capabilities Of Variational Autoencoders For 3D Car Point Clouds," in *IEEE Symposium Series on Computational Intelligence, SSCI 2020*, 2020.

[31] J. Kong, T. Rios, W. Kowalczyk, S. Menzel and T. Bäck, "On the Performance of Oversampling Techniques for Class Imbalance Problems," in *In Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 2020.

[32] N. V. Chawla, K. W. Bowyer, . L. O. Hall and W. . P. Kegelmeyer, , "SMOTE: Synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research,* vol. 16, pp. 321-257, 2002.

[33] H. He, Y. Bai, E. A. Garcia and S. Li, "ADASYN: Adaptive synthetic sampling approach for imbalanced learning," in *Proceedings of the International Joint Conference on Neural Networks*, 2008.

[34] S. Barua, M. M. Islam, X. Yao and K. Murase, "MWMOTE - Majority weighted minority oversampling technique for imbalanced data set learning," in *IEEE Transactions on Knowledge and Data Engineering*, 2014.

[35] B. Das, N. C. Krishnan and D. J. Cook, "RACOG and wRACOG: Two probabilistic oversampling techniques," in *IEEE Transactions on Knowledge and Data Engineering*, 2015.

[36] H. Zhang and . M. Li, "RWO-Sampling: A random walk over-sampling approach to imbalanced data classification," *Information Fusion,* pp. 99-116, 2014.

[37] A. I. Heft, T. Indinger and N. A. Adams, "Experimental and numerical investigation of the drivaer model," in *American Society of Mechanical Engineers, Fluids Engineering Division (Publication) FEDSM*, 2012.

[38] 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.

[39] Alcalá-Fdez, A. Fernández, J. Luengo, J. Derrac, . S. García, L. Sánchez and F. Herrera, "KEEL data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework," *Journal of Multiple-Valued Logic and Soft Computing,* 2011.

Appendix

The software modules have been developed using Python (Appendices A and B) and R (Appendix C) and thus need prior Python/R installations. The scripts in the present repository are provided under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or any later version.

The scripts described in the Appendices A and B were tested with Linux OS (Ubuntu 18.04, Conda environment) and Python 3.6.10 and require the installation of the following Python libraries:

- numpy, version 1.19.1
- TensorFlow, version 1.14.0
- TFLearn, version 0.3.2
- cudatoolkit, version 10.1.168
- cuDNN, version 7.6.5
- pandas, version 1.1.0
- implementation of the Chamfer distance used in [38], available at https://github.com/optas/latent_3d_points. If the scripts are used in an environment with Python 2.7+, CUDA 8.0+ or cuDNN 6.0+, the scripts in [38] from Achlioptas et al. must be adjusted and compiled to the current versions in the environment.

Furthermore, the data set for training the point cloud autoencoders have to be stored in a single directory, which comprises files in one of the following formats: *.csv*, *.xyz*, *.stl* or *.obj*. The algorithms in the data pre-processing script were developed to load files from a directory with mixed formats, however, it is not recommended. If the user would like to train or test the models on the ShapeNetCore data, for example, the data have to be pre-processed and organized in a single directory prior to the training, otherwise the scripts cannot load the point cloud data.

For the software in Appendix C, the R software needs to be installed along with R packages for imbalance classification. The training data from the KEEL data set [39] needs to be preprocessed before being loaded into the scripts, by changing the file format from .dat to .csv and the class labels from {positive, negative} to {1, 0}.

Appendix A – Vanilla 3D point cloud autoencoder scripts

Step 1: Training the autoencoder – the first step to use the autoencoder is to train the parameters on a data set of 3D point clouds. The training algorithm is performed by using the following command in the terminal:

```
python pcae_training.py --N [point_cloud_size] --LR [lr_size] --GPU [gpu_id]
--i [data_set_directory] --o [output_directory]
```

The script generates a directory named *Network_pcae_N{i}_LR{j}* in the output directory, as defined for the parameter *–o*, with values of *i* and *j* as assigned to *--N* and *--LR*, respectively. If the parameter is not specified, the directory is created in the path, where the user starts the training

algorithm. Further parameters of the architecture were hard coded in the training script and, if necessary, need to be changed in the *pcae_training.py* file (Figure 25).

The script outputs the following files:

- checkpoint, pcae.data-00000-of-00001, pcae.index and pcae.meta: these files store information about the tensorflow graph and are used to recover the trained parameters.
- geometries_training.csv and geometries_testing.csv: text files containing the path to the geometries assigned to the training and testing partitions.
- pcae_N{}_LR{}_losses_training.csv and pcae_N{}_LR{}_losses_test.csv: text files containing the values of the mean and standard deviation calculated for each epoch on the training and test partitions.
- normvalues.csv: contains the maximum and minimum values observed in the data set, used for normalizing the data before training
- plot_losses_pcae_N{}_LR{}.png: scatter plot of the losses computed on the training and test sets.
- log_dictionary.py: python dictionary containing the parameters assigned for training the autoencoder.

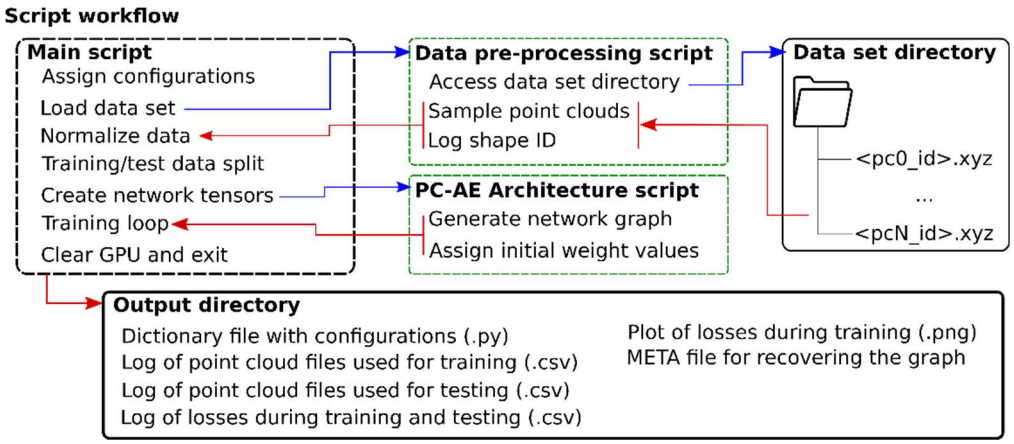


Figure 25 - Configurations and information workflow of the autoencoder training algorithm.

Step 2: Testing the trained architecture – The trained architecture can be tested using two scripts: one for computing the reconstruction loss on a set of point clouds and the other for generating point clouds from samples in the latent space. For computing the reconstruction losses, the following command should be used in the terminal:

```
python evaluate_loss.py --N [point_cloud_size] --LR [lr_size] --GPU [gpu_id]
--i [list_with_pointclouds_path] --VAE False
```

where the parameter *--i* specifies the path to a text file, which contains the paths of the point clouds that should be used as input. For this script, the point cloud data have to be stored as .xyz files, otherwise they cannot be loaded for testing the architecture. An example of potential list files that could be used are the geometries_training.csv and geometries_testing.csv, generated by the autoencoder training algorithm. The output of the script is a file named *reconstruction_losses.dat*

in the directory *pcae_test*, generated inside the training folder, containing the path to the tested geometries, corresponding latent representations and the computed reconstruction losses.

Since both architectures and algorithms are very similar, a single script was developed for testing both architectures. The parameter --*VAE,* is a flag to indicate whether the script is applied to a Vanilla (False) or Variational (True) autoencoder, such that the algorithm can assign different paths and architectures. If not assigned, it is assumed that the algorithm is applied to a vanilla autoencoder.

For generating point clouds from samples in the latent space, the following command should be input to the terminal:

```
python pointcloud_generator.py --N [point_cloud_size] --LR [lr_size]
--GPU [gpu_id] --i [list_with_latentvariables_path] --VAE False
```

The file assigned to the parameter --i must comprise the values of the latent variables to reconstruct the point clouds, formatted as comma-separated values: each line describes the latent representation of a shape with latent variables separated by commas. The remaining parameters enable the same functionalities as described for the previous scripts. The outputs of the algorithm are point cloud data (.xyz) and plots (.html), stored in a folder point_cloud_generation inside the training directory of the autoencoder.

Step 3: Extracting the network parameters – in order to use the trained weights and biases in further application without requiring the tensorflow graph files, one can extract and save the parameters as text files, running the following command in the terminal:

```
python ae_parameters_extraction.py --N [point_cloud_size]
--LR [latent_representation_size] --GPU [gpu_id] --VAE [True
```

The output of the script are .*dat files with the corresponding name of the layer, as assigned in the file *architecture_autoencoders.py*, comprising the values of the trained parameters.

Step 4: Example of application – in order to illustrate a potential application of the autoencoder, we provide a script that performs interpolation between shapes in the training and test sets, using the representations in the latent space. The algorithm runs by calling the following command in the terminal window:

```
python autoencoder_lr_interpolation.py --N [pc_size]
--LR [latent_space_dimension] --VAE [if variational, True]
--GPU [gpu_to_be_assigned] --VAE [True/False
```

where the parameters assigned in the command line have the same functionality as in the previous scripts. The algorithm assigns five shapes from the training set and five from the test set to the interpolation batch, and generates five intermediate representations between each shape, which are reconstructed using the decoder and plotted as *.png* and *.html* files (Figure 26). After plotting the point clouds, the algorithm generates a .gif animation of the interpolation, stored in the directory

point_cloud_itnerpolation, created inside the autoencoder training folder, together with the other files.

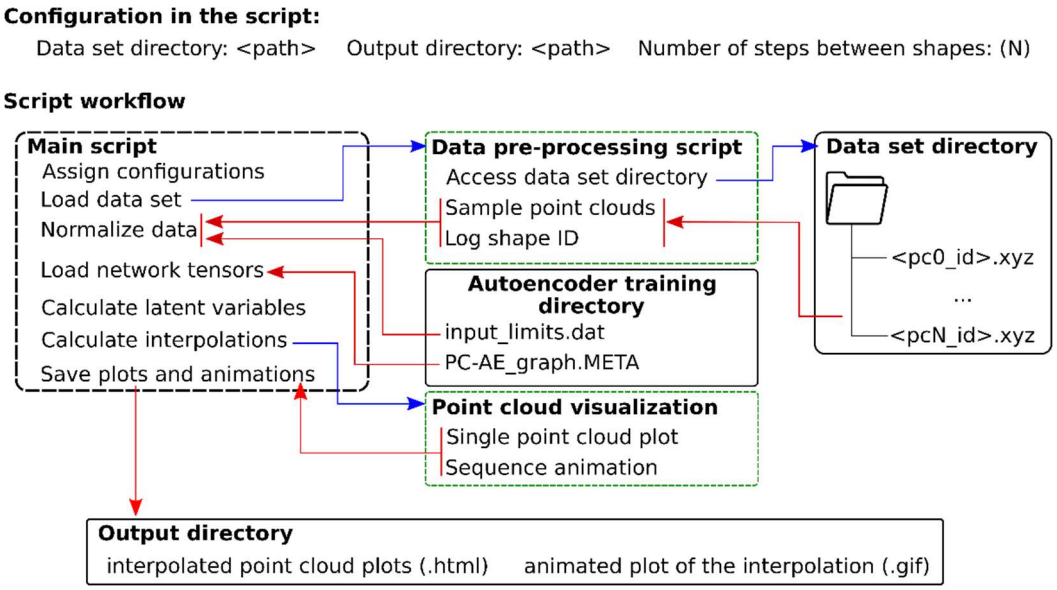


Figure 26 - Data workflow for a shape interpolation task performed in the latent space.

According to the ECOLE data management plan document (D5.3) the software in Appendix A is classified as restricted, i.e., the software can be accessed via the Bear Data sharing repository of the University of Birmingham (https://beardatashare.bham.ac.uk/login) after registration has been requested and completed, or by contact through the form provided on the ECOLE webpage: https://ecole-itn.eu/contact/.
The filename is: D1_PointCloudAutoencoder-200921-RESTRICTED.tgz

Appendix B – 3D point cloud variational autoencoder scripts

The implementation of 3D point cloud variational autoencoder (PC-VAE) *vpcae_training.py* - is tuned for the car class of ShapeNet dataset, i.e. the optimal value of *alpha* and *beta* for training PC-VAE with car class shapes. The script for training the autoencoder has a rather fixed structure, where one assigns the architecture and training hyperparameters and performs the optimization of the network weights. After training the model, the script also saves the architecture with trained weights, which can be used in downstream tasks, the history of losses during training and the log of the shapes sampled from the data set. The script can run using the inline command. A schematic overview of the data flow for training the point cloud autoencoder is shown in Figure 27. The steps for training and applying the variational autoencoder are as follows:

Step 1: Training the autoencoder – the first step to use the autoencoder is to train the parameters on a data set of 3D point clouds. The training algorithm is performed by using the following command in the terminal:

```
python vpcae_training.py – i [path to dataset] --o [path to output directory --N [pc_size] --LR [latent_space_dimension] --GPU [GPU id]
```

The script generates a directory named ***Network_v_pcae_N{i}_LR{j}*** in the output directory, as defined by the parameter *–o*, with values of *i* and *j* as assigned to *--N* and *--LR*, respectively. If the parameter is not specified, the directory is created in the path where the user starts the training algorithm. Further parameters of the architecture were hard coded in the training script and, if necessary, need to be changed in the v*pcae_training.py* file (Figure 27).

The script outputs the following files:

- checkpoint, vpcae.data-00000-of-00001, vpcae.index and vpcae.meta: these files store information about the tensorflow graph and are used to recover the trained parameters.
- geometries_training.csv and geometries_testing.csv: text files containing the path to the geometries assigned to the training and testing partitions.
- v_pcae_N{}_LR{}_losses_training.csv and v_pcae_N{}_LR{}_losses_test.csv: text files containing the values of the mean and standard deviation calculated for each epoch on the training and test partitions.
- v_pcae_N{}_LR{}_recon_losses_training.csv and v_pcae_N{}_LR{}_recon_losses_test.csv: text files containing the values of the mean and standard deviation calculated for each epoch on the training and test partitions for the reconstruction loss.
- v_pcae_N{}_LR{}_KL_losses_training.csv and v_pcae_N{}_LR{}_KL_losses_test.csv: text files containing the values of the mean and standard deviation calculated for each epoch on the training and test partitions for the KL divergence loss.
- normvalues.csv: contains the maximum and minimum values observed in the data set, used for normalizing the data before training
- plot_losses_v_pcae_N{}_LR{}.png: scatter plot of the losses computed on the training and test sets.
- log_dictionary.py: Python dictionary containing the parameters assigned for training the autoencoder.

- Training.pkl, test.pkl, validation.pkl : pickle file containing the training,test and validation shapes.

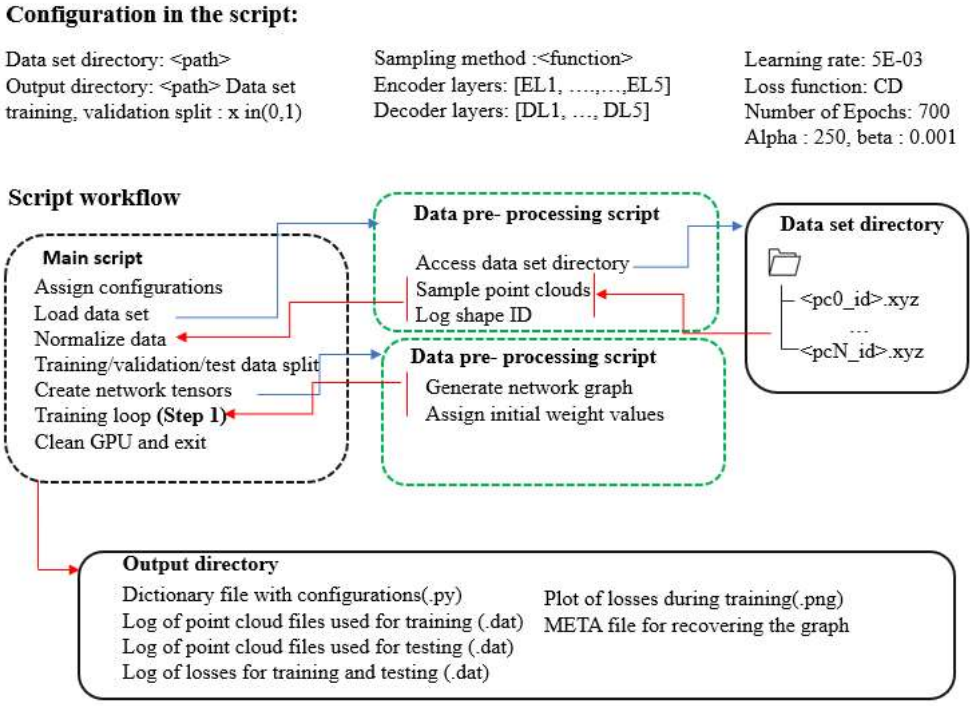


Figure 27 - Data workflow of variational autoencoder training in Step 1.

Step 2: Testing the trained architecture – The trained architecture can be tested using two scripts: one for computing the reconstruction loss on a set of point clouds and the other for generating point clouds from samples in the latent space. For computing the reconstruction losses, the following command should be used in the terminal:

```
python evaluate_loss.py --N [point_cloud_size] --LR [lr_size] --GPU [gpu_id]
--i [list_with_pointclouds_path] --VAE True
```

where the parameter *--i* specifies the path to a text file, which contains the paths of the point clouds that should be used as input. For this script, the point cloud data have to be stored as .xyz files, otherwise they cannot be loaded for testing the architecture. An example of potential list files that could be used are the geometries_training.csv and geometries_testing.csv, generated by the autoencoder training algorithm. The output of the script is a file named *reconstruction_losses.dat* in the directory ***pcae_test***, generated inside the training folder, containing the path to the tested geometries, corresponding latent representations and the computed reconstruction losses. The workflow of the script is described in Figure 28.

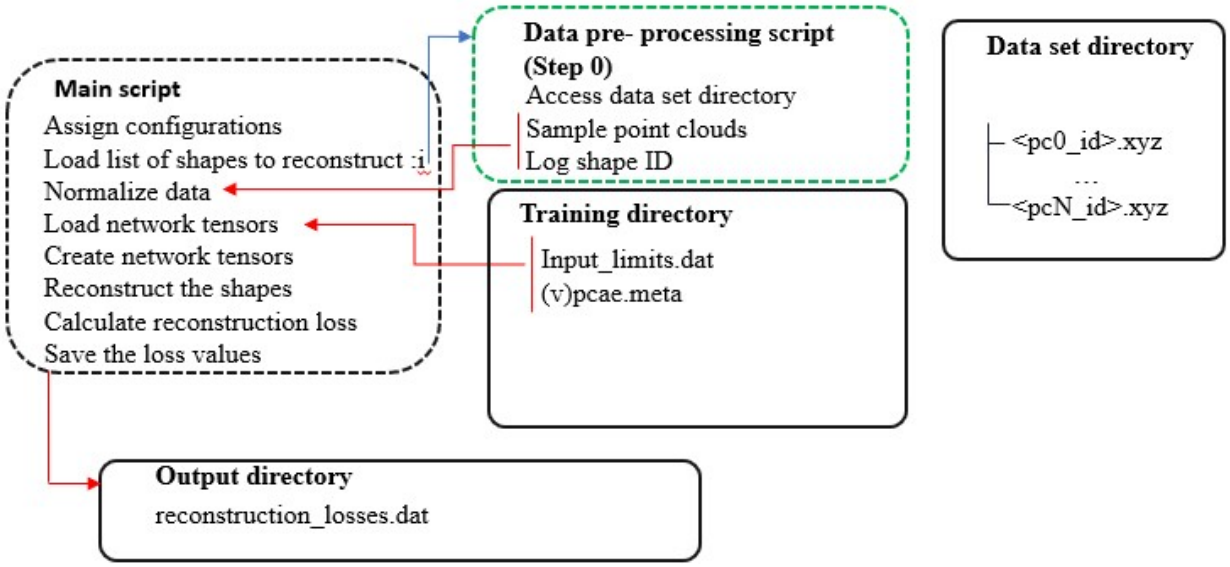


Figure 28 - Data workflow for evaluating the reconstruction loss using the trained model

For generating point clouds from samples in the latent space, the following command should be used in the terminal:

```
python pointcloud_generator.py --N [point_cloud_size] --LR [lr_size]
--GPU [gpu_id] --i [list_with_latentvariables_path] --VAE True
```

The file assigned to the parameter --i must comprise the values of the latent variables to reconstruct the point clouds, formatted as comma-separated values: each line describes the latent representation of a shape with latent variables separated by commas. The remaining parameters enable the same functionalities as described for the previous scripts. The outputs of the algorithm are point cloud data (.xyz) and plots (.html), stored in a folder ***point_cloud_generation*** inside the training directory of the autoencoder.

Step 3: Extracting the network parameters – in order to use the trained weights and biases in further applications without requiring the tensorflow graph files, one can extract and save the parameters as text files, running the following command in the terminal:

```
python ae_parameters_extraction.py --N [point_cloud_size]
--LR [latent_representation_size] --GPU [gpu_id] --VAE True
```

The output of the script are .*dat files with the corresponding name of the layer, as assigned in the file *architecture_autoencoders.py*, comprising the values of the trained parameters.

Step 4: Example of application – in order to illustrate a potential application of the variational autoencoder, we provide a script that performs interpolation between shapes in the training and test sets, using the representations in the latent space. The algorithm runs by calling the following command in the terminal window:

```
python autoencoder_lr_interpolation.py --N [pc_size]
--LR [latent_space_dimension] -V --GPU [gpu_to_be_assigned] --VAE True
```

Where the parameters assigned in the command line have the same functionality as in the previous scripts. The algorithm assigns five shapes from the training set and five from the test set to the interpolation batch, and generates five intermediate representations between each shape, which are reconstructed using the decoder and plotted as *.png* and *.html* files (Figure 26). After plotting the point clouds, the algorithm generates a .gif animation of the interpolation, stored in the directory *point_cloud_interpolation,* created inside the autoencoder training folder, together with the other files.

According to the ECOLE data management plan document (D5.3) the software in Appendix A is classified as restricted, i.e., the software can be accessed via the Bear Data sharing repository of the University of Birmingham (https://beardatashare.bham.ac.uk/login) after registration has been requested and completed, or by contact through the form provided on the ECOLE webpage: https://ecole-itn.eu/contact/.
The filename is: D2_PointCloudVariationalAutoencoder-200921-RESTRICTED.tgz

Appendix C – Improving imbalance classification

We aim to improve vehicle design classification using this software component. As we emphasized, in the field of computational design optimization, a minority number of design parameter variations will result in invalid geometries that violate some given constraints. Since the ratio between valid and invalid designs is likely to favor valid designs, we will typically have an imbalanced data set. Hence, performing proper imbalanced classification algorithms on the design parameters could save computation time and improve accuracy. The code is available at

https://zenodo.org/record/3855094#.XzpRvuj7QuV

and the steps for selecting a proper imbalance classification algorithm on the design parameters using this script are as follows:

Step 1: processing the script - The path to the data set and output directories were hardcoded in the scripts and need to be adapted before running the scripts. The script can be processed using the run command of the R software. The script calculates the performance matrix for SMOTE, ADASYN, MWMOTE, RACOG, wRACOG, RWO oversampling techniques and save it in the output directory.

Step 2: Evaluation: The final performance matrices are saved in the output directory in .csv format.

Configuration in the script:

Data set directory: <path>
Data set training split: stratified folds
Classifiers: C5.0 or SVM
Output directory: <path>
Random seed: <set.seed>
Oversampling methods: <function>
After-sampled IR (imbalance ratio): x in (0.8, 1]

Script workflow

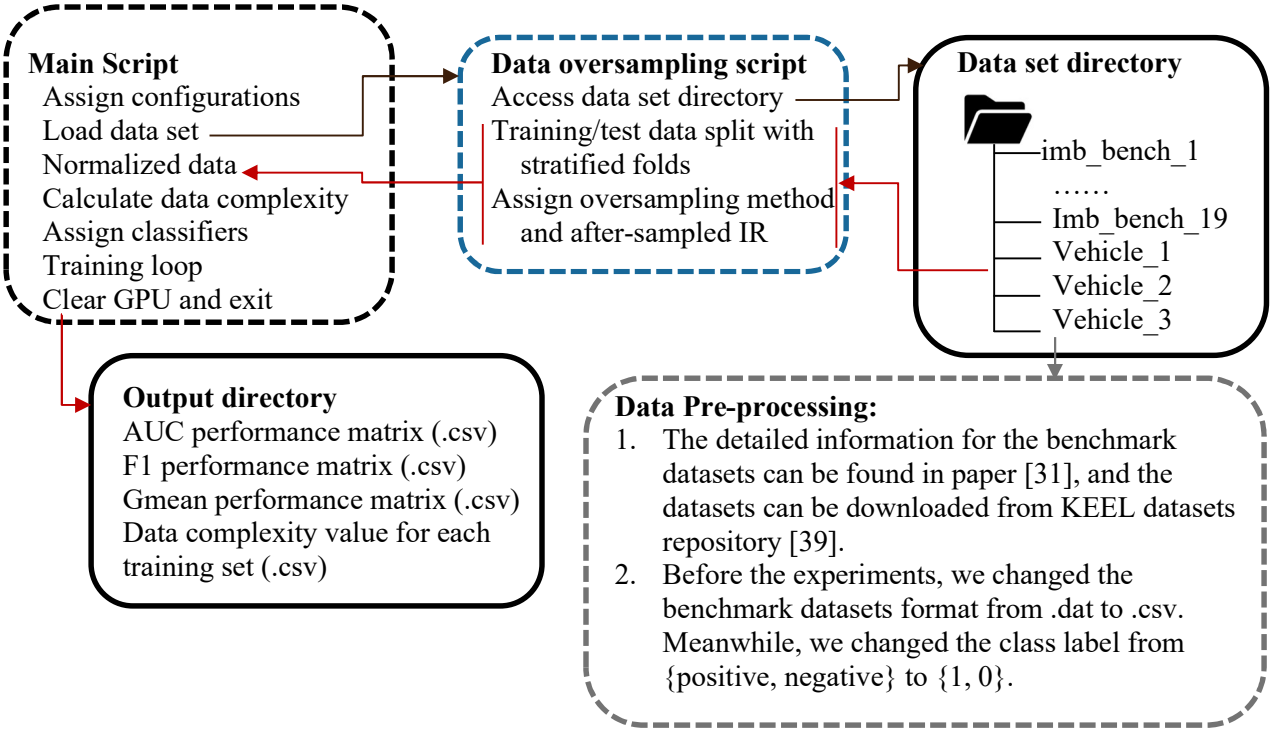


Figure 29 - Data workflow of class imbalance classification