Similar part identification integrating machine learning approaches with a SDM workflow

Uwe Reuter[1], Marcelo Pintado[2], Marko Thiele[2], Akhil Pillai[2], Florian Moldering[3],

[1]TU Dresden
[2]SCALE GmbH
[3]Audi AG

1 Motivation and Objective

Machine learning (ML) approaches for geometric part recognition have been evaluated with 3D automotive data in [1], where only one vehicle was used (Toyota Yaris with around 200 parts) and the exact match was tested, which means that the model was able to identify only the particular part shown regardless of the other classes (one-to-one match).

Near-perfect accuracy was obtained as a result. However, a limiting approach was given. To be implemented in a productive environment, engineers are looking to increase the training data with all vehicles available in an Original Equipment Manufacturer (OEM), and for plausibility in terms of similar geometric shapes, to learn how specific shapes are constructed. This model should be able to gather all similar parts and not only the exact match.

By doing so, we enhance the CAE process for advanced vehicle development in early stages. This strategic approach can compare information from unlabeled vehicles (automobiles from other OEM companies) as benchmark prior a CAD design or filter similar designs (or production facilities, specifications, etc.) to assist the manufacturing process using an existing part instead of setting up a production line reducing costs (see Fig. 1). This design analysis is called Carry Over Part (COP) which ensures to minimize cost in the early design phase from previous projects.

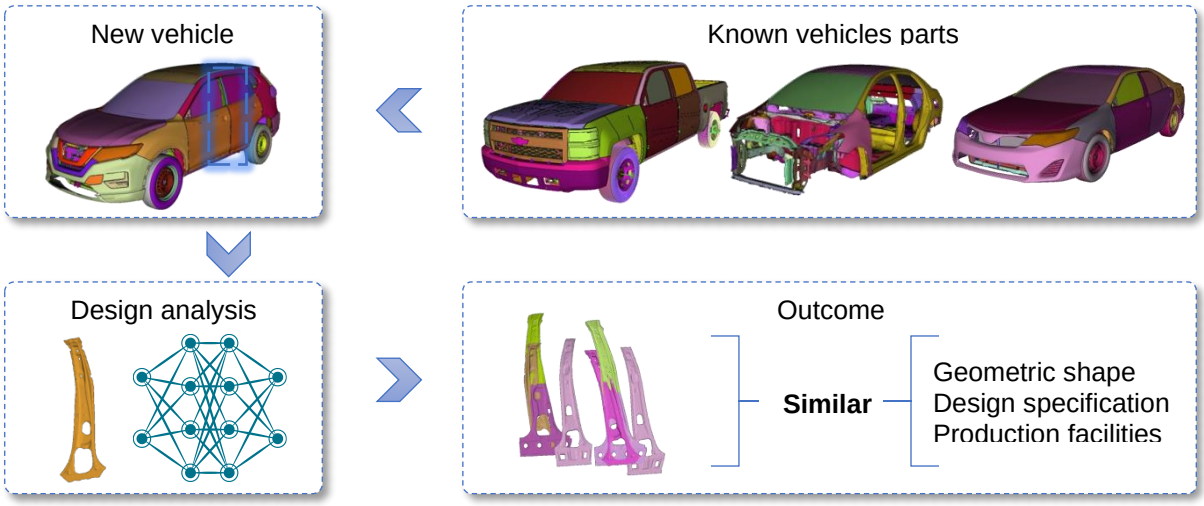

Fig.1: Motivation and Objective (Vehicles are courtesy of the CCSA team [2]).

2 Approach for classification of similar geometric parts

This work integrates a ML workflow with HPC and cloud resources, and automotive CAE data using *SCALE.model* [3], a powerful simulation data management (SDM) system for large CAE databases. Herewith, we investigate the feasibility of different techniques in three main sections. The first section is the input preparation, which starts with a 3D point cloud transformation and includes data augmentation. Some approaches were investigated, which led to the Iterative Closest Points (ICP) algorithm [4], which aligns all point-clouds similarly and generalizes the model.

The second section is architecture customization, which involves adjusting the ML algorithm for the specific use-case. The third section is the training customization, modifying the process with a proposed loss function exploiting the geometric similarities between automotive parts, with a non-supervised class clustering and smooth labeling (see Fig. 2).

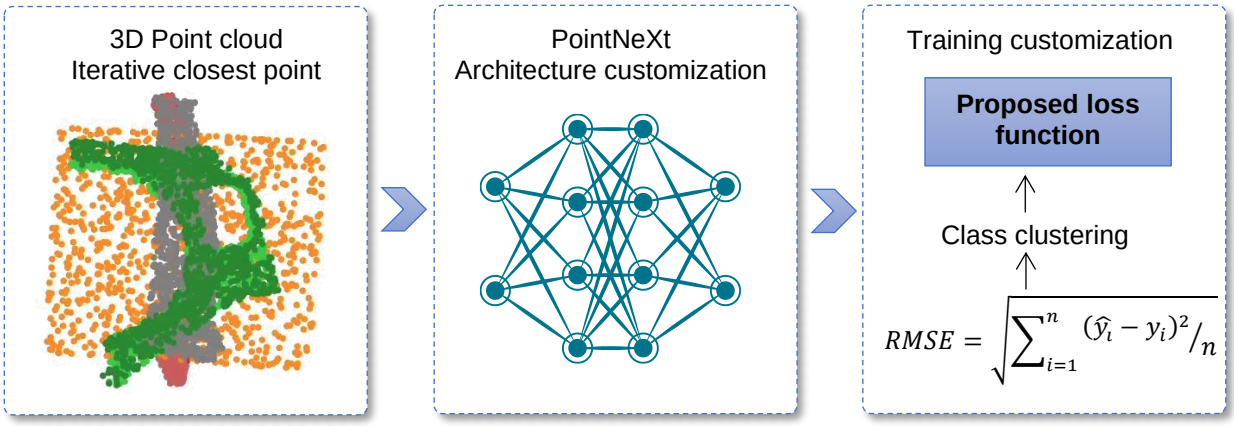

Fig.2: Machine learning workflow to find similar points.

2.1 Machine learning framework

This work uses the *PointNeXt* [5] framework based on *PointNet* [6] and *PointNet*++ [7]. *PointNet* is a pioneer architecture able to train a model with 3D point clouds with a desirable accuracy (see Table 1). *PointNet*++ is an extension with an added hierarchical structure, facing the limitation to capture local structures. It builds a hierarchical grouping composed of a set abstraction level, where in each level a set of points is processed and abstracted assembling a new set with fewer elements.

Three key layers exist in this set, the *Sampling layer* which takes the input points and defines the centroids of the local regions. The *Grouping layer* which constructs local regions finding neighboring points around that centroid, and a *PointNet* layer which is applied to each group separately and applies a transformation to later aggregate it through a max pooling, giving a multi-classification score.

PointNeXt is a fine tuned *PointNet*++ with improvements in its hyper-parameters. This influences the model performance and boosts its accuracy beyond the State-of-the-Art (see Table 1). Further enhancements such as model scaling with inverted residual MLP blocks were not used in this work.

Machine learning architecture	Overall accuracy
PointNet	89.2 [%]
PointNet++	90.7 [%]
PointNeXt	94.0 [%]

Table 1: Benchmark from machine learning architectures with ModelNet40 [5, 6, 7].

2.2 Dataset pre-processing

ML architectures are sensitive to the input data affecting directly their performance, which is why building the data is a crucial step from the entire training workflow. In general, this work starts with a 3D CAE model, which is the vehicle with all parts that are used for training. From each part, an optimal amount of data is needed to train an architecture. This is where data augmentation works.

Data augmentation (rotation, translation, etc.) and point cloud registration (ICP, principal axes, etc.) techniques were tested in [9], selecting a combination of Latin-hypercube sampling (LHS) with the ICP algorithm due to its performance (Table 3). This section is the start from the ML framework, which starts with a 3D CAE vehicle saved as ANSA database file.

From this file all parts are extracted and converted to a stereolithography (STL) file to convert all CAE files into a free research format, so the information can be further processed with python, and it has the advantage to achieve triangular elements from an already meshed file. A triangular mesh is required to create a 3D point cloud with mathematical approaches like LHS, barycentric coordinates, normalization and area weights to have consistent number of points along the entire surface.

A good practice [1] is to use number of points like 64, 128, 256, 512, 1024, 2048, etc. Results in [1] refer that 1024 points led to good performance without compromising time and computational efforts. Later, augmentation data and point cloud registration is applied, increasing the number of point clouds available and aligning them for better training. The type of approach chosen is not trivial due to the considerate influence from the data on the training process. This work chose LHS and ICP, a method to augment the data and register standardized point clouds.

This approach allows translation, scaling, rotation and mirroring with the purpose of transforming point clouds to have similar alignment. The procedure is divided into two steps. The first one is *Data association*, where all points from two point clouds are associated to each other to the closest point. The second step is *Transformation*, where a transformation matrix is built, and one point cloud is transformed by a series of iterations to align both point clouds similarly.

On the left side of Fig. 3, the visual representation of a 3D model, its parts and point clouds. And on the right side ICP is applied to the point cloud. There is a normalized ruby-shaped point cloud as reference and all point clouds already aligned as outcome. The process is to create all point cloud samples and then perform the transformation for each sample.

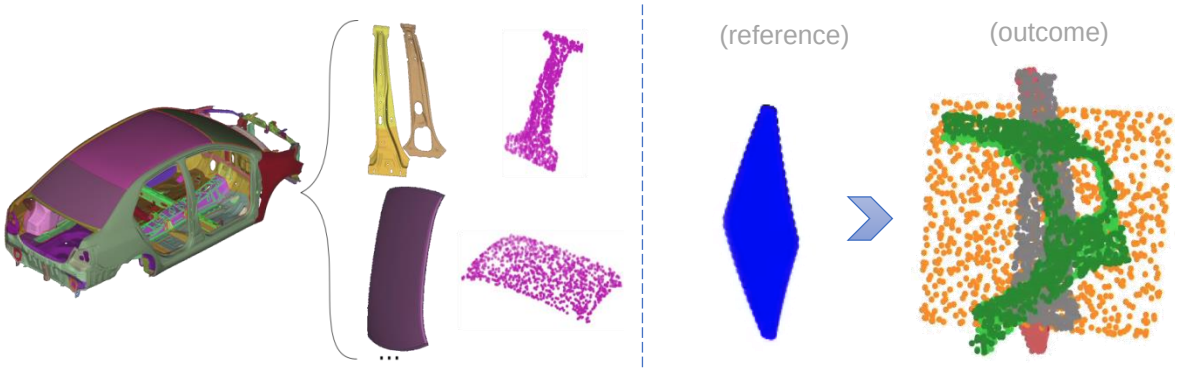

Fig.3: Data pre-processing (left). Iterative closest point (right).

2.3 Architecture customization

The training process starts with *PointNeXt* architecture, built with ModelNet40 benchmark and shaped with 40 different classes each representing one part and 1024 points per point cloud. Its structure starts with an input layer, then many Convolutional Neural Networks (CNNs), the classifier, and finally the output layer with the number of classes (see Fig. 4).

The feature extraction starts with a one-dimensional (1D) CNN, extracting higher features. Later, five groups of two two-dimensional (2D) CNN appear reshaping the data structure from (3 - number of coordinates, 1024 - number of points) to (512, 64), finalizing with a global feature of 512. The classifier has two hidden layers of 512 and 256 neurons and an output layer with the number of classes.

In practice, a batch of point clouds is used at the same time. Hence, the last layer has a final shape [B - batch number (32, 64, etc.), C - number of classes]. This last one is adapted to a new use-case, where the number of classes changes from the initial 40 (benchmark) to 912 (our work) leading to an architecture adjustment with important factors. First, the number of hidden layers required for better training. For multi-label classification tasks in deep learning, two would be sufficient. Second, the number of neurons these hidden layers should have.

Here only guidelines [10] were found. The input is the global feature layer (512), and the output is the number of classes (912). The guidelines imply that the number must be between the input and output

classes. It also recommends that the hidden neurons must not be greater than twice the number of the input, giving a threshold of 1024 neurons. It also stands that the hidden neurons should not increase and decrease between layers without a pattern. Eq. 2.1 and 2.2 show further guidelines. Lastly, for better efficiency purposes is to choose hidden neurons among numbers 512, 768, 1024, etc.

$$\#\ Hidden\ neurons = \frac{input + output}{2} = 712 \qquad 2.1$$

$$\#\ Hidden\ neurons = \sqrt{input \times output} \approx 684 \qquad 2.2$$

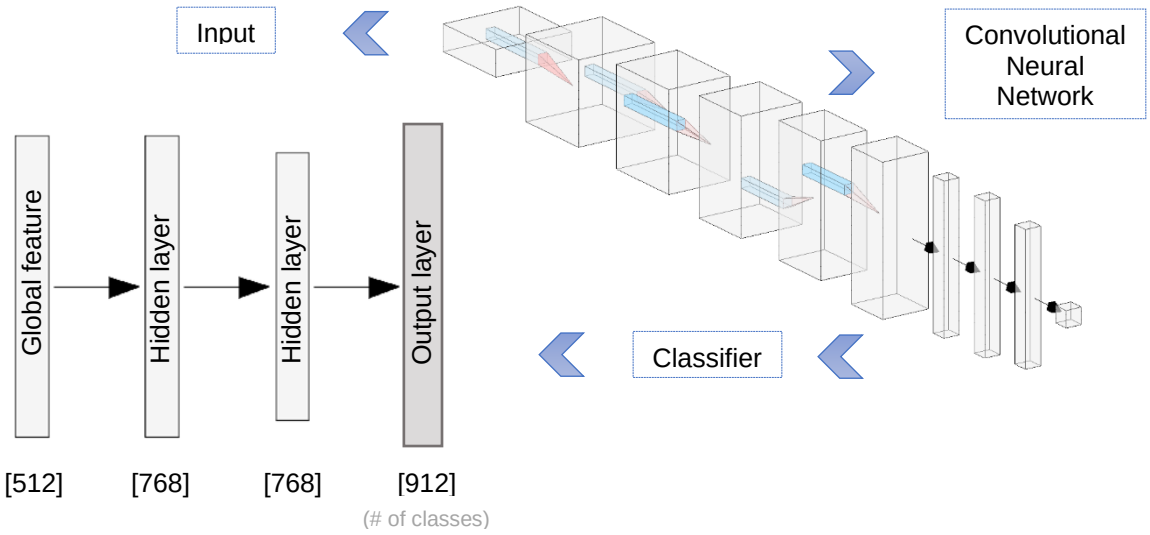

Fig.4: Machine learning architecture: PointNeXt.

2.4 Training customization

The Loss function is an important feature for training, helping the non-linear training algorithm to find the optimal convergence range according to the use-case. The learning algorithm takes the error value and back-propagates it along all derivatives calculating new weights and biases for each neuron. The *cross entropy* loss function gives good accuracy results for multi-label classification. Moreover, new implementations have been made to this function in this work to increase accuracy.

The loss function starts from the output layer formed by C classes. For C classes there are C output neurons with an x value from each neuron. A Softmax function S_n is applied to the class n (see Eq. 2.3), which is dependent on the x value of the same class n, and the total number of classes in the output layer C. As a result, there is an always positive normalized value and the sum from the output layer is equal to one. Which is then applied in the output layer (Eq. 2.4) (as an example, only two neurons are shown).

$$S_n = \frac{e^{(x_n)}}{\sum_{i=1}^{C} e^{(x_i)}} \qquad 2.3$$

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \xrightarrow{yields} \begin{bmatrix} -\ln(S_1) \\ -\ln(S_2) \end{bmatrix} \qquad 2.4$$

The result vector is multiplied by the target vector, giving an error value that needs to be minimized to give feedback to the network. The *cross-entropy* loss function takes the label vector and places one in the correct class and zero in all the other classes. This means, it uses the negative ln of the function S_n from the correct class. The *smooth labeling function* gives rather a high value (i.e. 0.8) and the rest (0.2) spreads it evenly to the other classes. From all functions, one batch is given to the model, which means that the mean is calculated and used for the back-propagation procedure.

PointNeXt is 'un-calibrated' to perform best at the first match. The use-case is to extract all similar parts in the top predicted values. Hence, this work made an adjustment resulting in an innovative training approach. First, a non-supervised clustering between classes where each class is compared and filtered together. Later, *smooth labeling* was chosen. The correct class would be 0.8. Let us assume that for this specific correct class, 10 classes were clustered from the previous step.

Then, those classes would have 0.02 value (which is 0.2 spread equally between the cluster), and the rest would have zero value, strengthening the relationship between clustered classes. This work uses Root Mean Square Error (RMSE) as criteria to cluster. It starts with the data association step between the closest points. The error value is the RMSE that represents how similar point clouds are. Later, *variable smooth labeling approach* was selected, a type of cross-entropy function dependent on the number of clustered classes.

These smooth label number changes so the correct class has the highest number, and the other clustered classes have a close high number. As an example, see Table 2, the first column is the number of clustered classes, the second column is the value for the correct class, always higher than the third column, the value of those clustered classes. The last column is the value for the rest.

No. similar classes	Label → Correct	Label → Similar	Label → Wrong
1	$[0.6]$	$[0.4]$	$[0 \quad \cdots \quad 0]$
2	$[0.4]$	$[0.3 \quad 0.3]$	$[0 \quad \cdots \quad 0]$
3	$[0.3]$	$[0.23 \quad 0.23 \quad 0.23]$	$[0 \quad \cdots \quad 0]$
…	…	…	…
10	$[0.1]$	$[{}^{0.9}/_{10} \quad \dots \quad {}^{0.9}/_{10}]$	$[0 \quad \cdots \quad 0]$
11	$[0.1]$	$[{}^{0.9}/_{11} \quad \dots \quad {}^{0.9}/_{11}]$	$[0 \quad \cdots \quad 0]$
12	$[0.1]$	$[{}^{0.9}/_{12} \quad \dots \quad {}^{0.9}/_{12}]$	$[0 \quad \cdots \quad 0]$
…	…	…	…

Table 2: Example of the variable smooth labeling approach.

2.5 Evaluation parameter

This work decided to use the top 20 predicted classes as a pre-defined parameter to evaluate the accuracy and have a first insight. An architecture is normally evaluated by the overall accuracy, which indicates how accurate the model is to predict the exact match from the whole dataset. However, this use-case needs to extract all information from the similar geometry matches (see Eq. 2.5).

$$Accuracy\ [\%] = \frac{Similar\ parts\ found\ in\ top\ 20}{All\ posible\ similar\ parts} \times 100\% \qquad 2.5$$

3 Implementation and evaluation

The implementation was done with SCALE GmbH and Audi AG. The starting point was 3D parts from 3 different vehicles provided by Audi AG. Results are from these vehicles and due to data protection, they are not named and instead free data is used for illustrations (see Fig. 1). These 3 models together have a total of 912 automotive parts from all the training classes.

First, LHS was implemented to create and augment the point cloud data. Later, the ICP algorithm was used to align all point clouds and the architecture customization was made increasing the length from the output layer for all classes (from 40 to 912). Moreover, hidden layers from the classifier were changed through guidelines (subsection 2.3), having a combination of 2 hidden layers with 768 neurons to extract information without over-fitting the model.

Additionally, many data augmentation algorithms (rotation, translation, scaling, etc.) were tested increasing exponentially the time, in which the combination of LHS with ICP had the best accuracy and time ratio to build the dataset using 4 CPUs (see Table 3).

Description	Time	Accuracy
Initial	6 hours	46 %
Conventional data augmentation	14 hours	69 %
LHS + ICP	6.5 hours	73 %

Table 3: Example of the variable smooth labeling approach.

The next step consists in a clustering method of similar classes and a variable smooth labeling approach dependent on the number of clustered parts, having always the highest value for the correct class and a close high value to the clustered classes. The rest classes have a zero value, and the sum of all classes is equal to one.

The clustering method should be an efficient non-supervised cluster approach which would give all similar-shaped parts clustered together. This method uses RMSE value criteria, extracted after comparing two aligned point clouds by the ICP. A threshold of 0.002 was used to filter classes. The RMSE was doubled obtained, which means that both point clouds were used as reference to the other respectively, having two RMSE values to cluster them together. Finally, the variable smooth labeling approach was implemented (subsection 2.4).

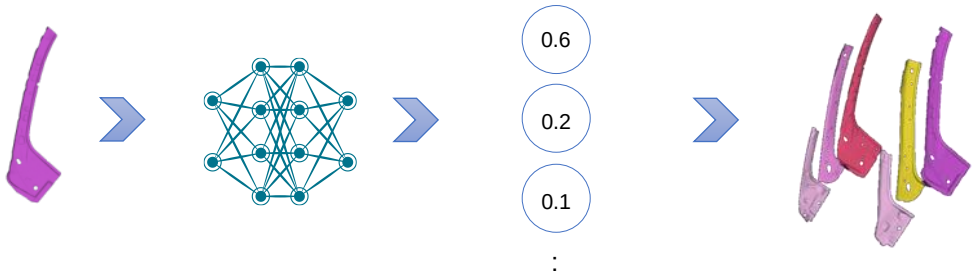

Fig.5: Testing pipeline

The main purpose is to ensure that all similar classes appear in the top-ranking values. Once the model is trained with all the data available, one class is given to the trained model. This model gives the results from the output layer. It is then arranged from the highest to lowest number, from the first to the last prediction. To reframe it, the ML model expresses its prediction likelihood by giving a maximum value to the classes that are similar. For this work, the position from all similar classes is extracted to see whether they are close to the top positions (maximum values) or not (see Fig. 5).

Scope of analysis		Training data	
Part for analysis	*Value*	*Training parameter*	*Value*
Pillar A	6	Vehicles, Parts	3, 912
Pillar B	6	Training, testing samples	200, 40
Wheelhouse	6	Points per sample	200
Roof	2	ML model	PointNeXt
Floorboard	3	Number of epochs	300
Total	23	GPU	NVIDIA GeForce RTX 3060 Ti

Table 4: Scope of analysis and Training data parameters.

We have pre-defined 5 groups (23 classes) for testing purposes, pillar A, pillar B, wheelhouses, roof, and floorboards (see Table 4). Though we understand the selection is subjective, it was chosen internally to have a first insight of the behavior from pillars, flat and round shapes. All methods were tested among the scope of analysis.

Visualization is shown in Fig. 6 with the initial state (left side) and the final state (right side). An x-y graph is shown, the x-axis shows each point cloud used from Table 4. The y-axis is the top 20 (maximum) rank values obtained after a prediction was made using the classes from x-axis. Inside each chart, all similar predicted classes are represented with a rectangle.

Each rectangle matches the color with the clustered classes seen in Table 4 (yellow: floorboard, green: pillar A, red: pillar B, black: roof, and blue: wheelhouse). Fig. 6 shows a 27 % increase in accuracy. All approaches were able to converge in 300 epochs. 4 CPUs were used for the pre-processing tasks and a GPU NVIDIA GeForce for the training process. SCALE.model was implemented as SDM for all tests (section 4).

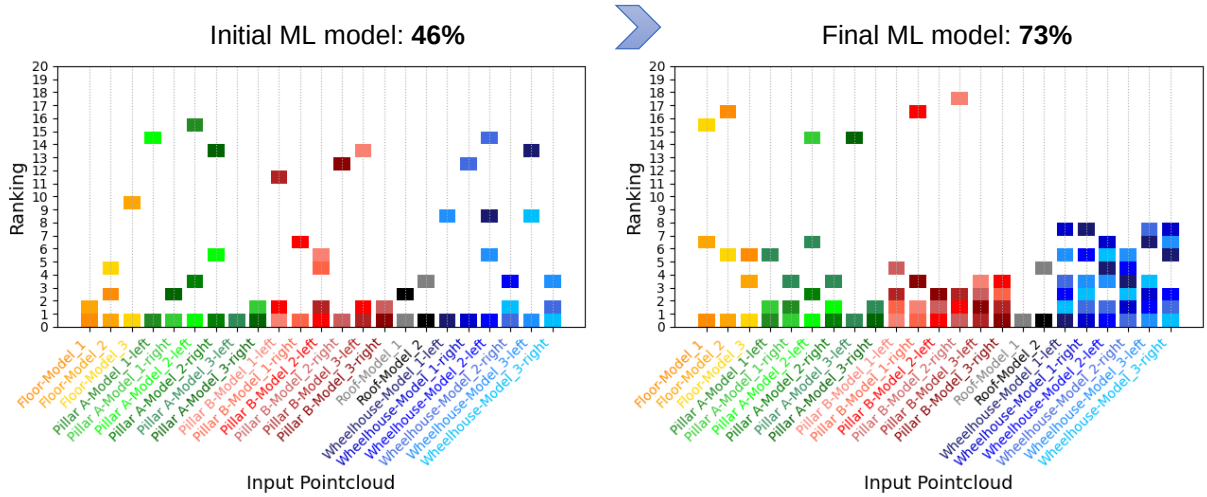

Fig.6: Initial and final state from a machine learning model within the scope of the research.

4 SDM implementation

Many tests in different stages (pre-, post-processing and training) were employed during this work. We integrate the whole process in an SDM software (SCALE.model [3]) to speed up the process, make it reproducible in terms of evaluation and achieve tracking for each one of the tests. Furthermore, the source data, e.g. the CAE models containing the parts that are needed to create the dataset for the training are maintained within the SDM system and on the other hand the SDM system can then apply the trained model to provide functionality for the user to search for similar parts.

Figure 7 indicates in detail the SDM implementation. The overall concept was made so it has 3 main stages, each of them are included in a separate place to manage their data, this place is called Pools. Moreover, one Plug-In was built to enable the user to make predictions from any environment. The stages are named Part Library, Machine Learning Dataset and Geometry Identification. Each of them implements a specific stage from the overall ML training process.

First, the Part Library. This pool stores all 3D CAE data needed as a source for a specific training. It is separated from the others stages and kept in a separate pool so the user can decide how many vehicles to use for training, and in each vehicle, decide which parts should be included. The input data is the vehicle. The process generates ANSA database and STL files for each part of the provided CAE file, including support information for next steps.

The next stage is to generate the Machine Learning Dataset. This stage is organized in a separate pool and has the previous pool mounted to provide all data needed as input to generate the dataset for the ML. The input is all STL files, the scripts used for creating the dataset collect these STL files

and generate the point clouds. Here the user can test all types of pre-processing, interact with different libraries, and prepare tests for different classes.

The LHS and ICP algorithm was used as final step in the data preparation (subsection 2.2) to augment and align all point clouds. Later, the RMSE criteria is implemented to cluster similar classes. User can test different criteria by adjusting the scripts to see which one works better for the use-case. Finally, a ML package (scripts related to the structure) is available in the Pool. The user can test multiple architectures and adjustments. In this work, we used PointNeXt architecture and the number of classes to adjust it (sub-section 2.3).

Subsequently, the Geometry Identification Pool is the one to implement the last stage. This pool has all previous pools mounted so it has access to all the output information (point clouds and ML architecture). The training workflow starts in this pool. The user can test hyper-parameters and adjust the loss functions for the training process. The output is the trained network, a file with all data about the structure and saved values where the best accuracy results were found. This file is stored as a result in form of a “Run-Output-Component” in that same pool.

Finally, a Plug-In was implemented to test the trained network inferring a part and evaluating its accuracy. A plugin is the user interface where the user can bring any STL file with any shape, choose the ML algorithm and use it to find similar parts from the trained vehicle data. This process can also be used with 3D data from any benchmark (other OEM) and make correlations between unknown data and their own data (production facilities, design specification, etc.).

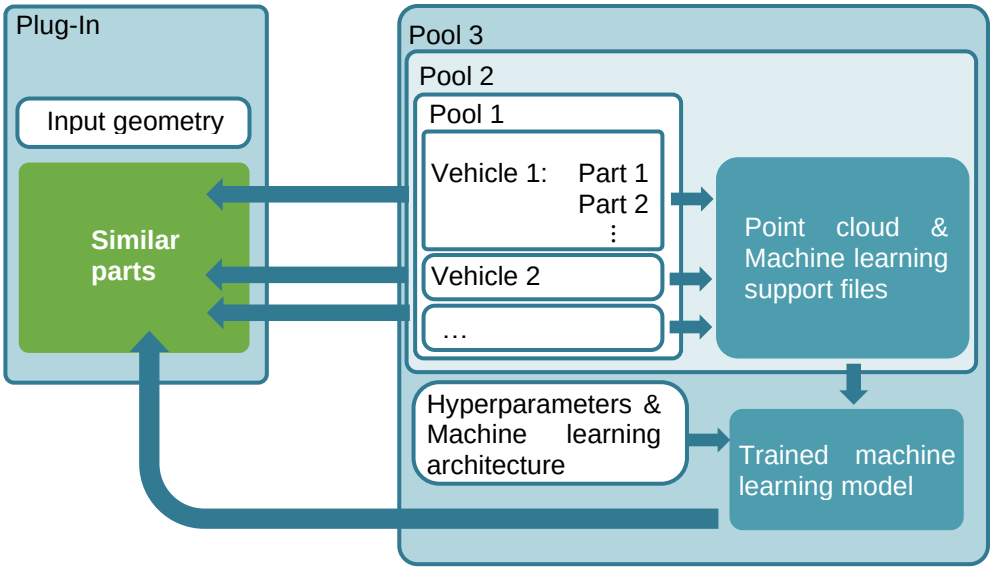

Fig.7: Simulation Data Management pipeline.

5 Conclusions and Outlooks

3D point clouds are canonical which makes this work widely applicable. Moreover, PointNeXt was the ML algorithm applied based on PointNet++ and PointNet. It is heavily influenced by data transformation which makes the data augmentation important. Further ML applications can be beneficial. For instance, supervised learning trains through user feedback, a strategy that could lead to a better accuracy after conventional learning processes reach their best performance.

This work implemented LHS with ICP as data pre-processing step because it helps generating data sets that when used to train with the PointNeXt architecture it provides a good balance between accuracy (~73%) and computational efforts needed for the generation of the data sets (~6.5 hours). The ICP method aligns all point clouds together standardizing the input data set.

Furthermore, the architecture was modified by arranging a new classifier with 2 hidden layers with 768 neurons using guidelines for a better correlation between the global feature and the output layer. Further research is advised to find which reference point cloud is the optimal one for automotive parts. A more precise alignment is expected to improve the outcomes.

Current Artificial Neural Networks (ANNs) are unbalanced, which helps to increase their one-to-one match accuracy performance, but their accuracy for similar parts is low. Therefore, an original loss function was proposed. It begins with a non-supervised cluster between classes using the RMSE value criteria below 0.002. Similar-shaped classes have low RMSE.

Later, a smooth labeling approach with variable labels was implemented and targeted to the clustered classes, giving the highest label to the correct class and the remaining is spread equally into the cluster. All labels are bordering the highest one, and add up to one. The remaining classes have a label of zero. A 73% accuracy performance w.r.t. all similar parts were achieved for the classes analyzed and the first predicted class was still an accurate one-to-one match. The smooth labeling strengthens the relation between classes.

Finding similar parts in an early design stage can help to gain knowledge about previous designs, reduce costs in production if a COP can be identified and avoid the need to create CAE models for previously used parts, i.e. if it was created for other simulation disciplines. SDM Systems can be used not only to setup simulation processes but also to implement a ML workflow on top of CAE data that is already that originates from the SDM System. Also, the functionality of an SDM-System allows a standardized integration, speeding the process of future research.

The proposed training algorithm gave adequate results. 100% accuracy was reached with manual clustering validating the approach if the clustering was ideal. Hence, the non-supervised clustering approach must be improved. For instance, ideas like unlabeled data, noisy label data, or other supervised approaches can be investigated to compare classes and extract feature correlations to improve the clustering performance. Finally, scalability of more vehicles or parts must be investigated. Current topics like continual and transfer learning should be implemented.

6 Literature

[1] Pillai A., Reuter U., Thiele M.: "Estimation of spot weld design parameters using deep learning". *12th European LS-DYNA Conference 2019*. No. 1, 2019, pp. 1-16.

[2] George Mason University. Center for Collision Safety and Analysis (CCSA). Washington DC.

[3] SCALE GmbH. 2023. SCALE.model (1.182.0). [Software]. [Accessed 01 May 2023].

[4] Zhang J., Yao Y., Deng B.: "Fast and robust iterative closest point". *IEEE Transactions on Pattern Analysis and Machine Intelligence*. No. 2, 2022, pp. 1-19.

[5] Qian G., Li Y., Peng H., Mai J., Hammoud H., Elhoseiny M., Ghanem B.: "Pointnext: Revisting pointnet++ with improved training and scaling strategies". *NeurIPS 2022*. No. 2, 2022, pp. 1-18.

[6] Qi C., Su H., Mo K., Guibas L.:"Pointnet: Deep learning on point sets for 3d classification and segmentation". *CVPR 2017*. No. 2, 2017, pp. 1-19.

[7] Qi C., Yi L., Su H., Guibas L.:"Pointnet++: Deep hierarchical feature learning on point sets in a metric space.". *CVPR 2018*. No. 1, 2017, pp. 1-14

[8] Gare G., Galeotti J.: "Exploiting class similarity for machine learning with confidence labels and projective loss functions". *CoRR*. No. 1, 2021, pp. 1-9.

[9] Pintado M.: "Enhancing the concept of geometry identification to predict similar parts using 3D point clouds and machine learning approaches". *Master Thesis*. Faculty of Civil Engineering, Technische Universität Dresden, 2023.

[10] Singh, H.: "Choosing number of hidden layers and number of hidden neurons in neural networks". *LinkedIn*, https://www.linkedin.com/pulse/choosing-number-hidden-layers-neurons-neural-networks-sachdev/ [Accessed 01 May 2023].