upscale

Upscaling **P**roduct development **S**imulation **C**apabilities exploiting **A**rtificial inte**L**ligence for **E**lectrified vehicles

D2.2 Reduced order models for aerodynamic performance prediction

Authors

Kyriazis Nikolaos, Eugene De Villiers, Jianbo Huang – Engys

Carsten Othmer, Markus Mrosek – VW

Luca Miretti – CRF

Enric Aramburu, Bhanu Prakash, Charalamps Tsimis, Albert Rodriguez de Liebana – IDIADA

Patrik Bangert – Algorithmica Technologies

Erik Sällström, Eysteinn Helgason – Volvo Cars

Month Year

04/2020

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement Nº 824306.

Project Details

PROJECT TITLE	Upscaling product development simulation capabilities exploiting artificial intelligence for electrified vehicles
PROJECT ACRONYM	Upscale
GRANT AGREEMENT NUMBER	824306
INSTRUMENT	RIA
CALL	LC-GV-2018
STARTING DATE OF THE PROJECT	November, 1ST 2018
PROJECT DURATION	42 Months

The UpScale Project

The UPSCALE (Upscaling Product development Simulation Capabilities exploiting Artificial inteLligence for Electrified vehicles) goal is demonstrating the feasibility of using AI enhanced CAE methods in EV development processes, such as vehicle aerodynamics, battery thermal modelling and crash simulation and leading the deployment of AI tools for other CAE applications. UPSCALE is the first EU-project that has the specific goal to integrate artificial intelligence (AI) methods directly into traditional physics-based Computer Aided Engineering (CAE)-software and –methods. These CAE-tools are currently being used to develop road transportation not only in Europe but worldwide. The current focus of the project is to apply AI-methods to reduce the development time and increase the performance of electric vehicles (EVs) which are required by the automotive industry to reduce global emission levels. High performance computing (HPC) and CAE-software and –methods play a decisive role in vehicle development process. In order to make a significant impact on the development process, the two most HPC intensive CAE-applications have been chosen as use cases for the project: vehicle aero/thermal- and crashmodelling. When considering total automotive HPC usage, approximately 20% is used for aero/thermal simulations and up to 50% of HPC resources are utilized for crash simulations. By improving the effectiveness of these two areas, great increases in efficiency will lead to a 20% reduction of product time to market. Other novel modelling approaches such as reduced order modelling will be coupled to the AI improved CAE-software and -methods to further reduce simulation time and ease the application of optimization tools needed to improve product quality. Through the combined effort of universities, research laboratories, European automotive OEMs, software companies and an AI-SME specialized in machine learning (ML), the UPSCALE project will provide a unique and effective environment to produce novel AI-based CAE-software solutions to improve European automotive competiveness.

The UpScale Consortium

PARTICIPANT Nº	PARTICIPANT ORGANISATION NAME	COUNTRY
1 (Coordinator)	**IDIADA AUTOMOTIVE TECHNOLOGY SA (IDIADA),**	Spain
2	VOLVO PERSONVAGNAR AB (Volvo Cars)	Sweden
3	VOLKSWAGEN AG (VW)	Germany
4	CENTRO RICERCHE FIAT SCPA (CRF)	Italy
5	ESI GROUP (ESI GROUP)	France
6	ENGYS LTD (ENGYS LTD)	United Kingdom
7	Kompetenzzentrum - Das Virtuelle Fahrzeug, Forschungsgesellschaft mbH (VIF)	Austria
8	VRIJE UNIVERSITEIT BRUSSEL (VUB)	Belgium
9	ECOLE NATIONALE SUPERIEURE D'ARTS ET METIERS (ENSAM PARISTECH)	France
10	ALGORITHMICA TECHNOLOGIES GMBH (ALGORITHMICA)	Germany
11	F INICIATIVAS I MAS D MAS I SLU (F-INICIATIVAS)	Spain

Document Details

DELIVERABLE TYPE	Report
DELIVERABLE Nº	2.2
DELIVERABLE TITLE	Reduced order models for aerodynamic performance prediction
NAME OF LEAD PARTNERS FOR THIS DELIVERABLE	Volvo
VERSION	1
CONTRACTUAL DELIVERY DATE	M18
ACTUAL DELIVERY DATE	M18
DISSEMINATION LEVEL	Public

Revision History

The following table describes the main changes done in the document since it was created

REVISION	DATE	DESCRIPTION	AUTHOR (ORGANIZATION)
V.0	13/04/2020	First complete version of deliverable	Eysteinn Helgason (Volvo Cars)
V.1	20/04/2020	Review by WP leader	Luca Miretti (CRF)
V.2	30/04/2020	Reviewed by Project Coordinator	Albert R. de Liébana (IDIADA)

Disclaimer

Table of contents

1. Executive Summary 6
2. Problem statement 7
2.1. Test case description 7
2.2. Generation of training data: Workflow 10
2.2.1. DOE 10
2.2.2. Automated workflow 11
2.3. Error Metrics 13
3. Methods 14
3.1. Deterministic Method: POD + Interpolation 14
3.2. Non-deterministic Method 15
3.2.1. Network details 16
3.2.2. Training 21
4. Results 21
4.1. Deterministic Method: POD + Interpolation 21
4.2. Non-deterministic Method 25
5. Conclusions and Future Work 30
ACKNOWLEDGEMENT 31
7. References 32

1. Executive Summary

This report is a part of work package 2, focusing on Reduced Order Models for aerodynamic prediction, and describes the work performed in Task 2.2, including subtasks 2.2.1 and 2.2.2. The results will be used as a basis for future work in work package 2 and 4.

The main goal of Task 2.2 is to build reduced order models that can be used to reduce optimization time as well as for real-time decision making. Two kinds of reduced order models are investigated in this report, a deterministic model is assessed in subtask 2.2.1 and a non-deterministic model in subtask 2.2.2. For the purpose of comparing the two types of models a 2 dimensional parameterized geometry based on the electrical drivAer model (1), presented in Deliverable 2.1, was created. The parametrized geometry was used to create approximately 1000 different geometries. The flow around each geometry was simulated using an automated workflow and the drag values, flow fields and pressure fields saved. A dataset consisting of approximately 1000 samples was generated in this way and used as an input to the reduced order models. Finally, the accuracy of the predicted drag values and the velocity and pressure fields is evaluated.

This deliverable doesn’t deviate from the plan in regard to its content or delivery date.

2. Problem statement

2.1. Test case description

For the investigation and the comparison of prediction capabilities by deterministic and non-deterministic models described in this report, a simplified aerodynamic test case was identified. The publicly available DrivAer CAD model was converted into an electrical “e-DrivAer” version by replacing the internal combustion engine with an e-Motor and modifying the underbody in order to install a simplified battery pack, according to the procedure described in deliverable D2.1. The notchback version was used for this study.

In this stage of the activity the main goal was to define a functional workflow. Considering that a significant number of tests and CFD simulations were planned, it was decided to work on a 2D version of the model in order to speed-up workflow turn-around times. The baseline shape was generated by extracting the centerline section of the full model, as presented in Figure 1.

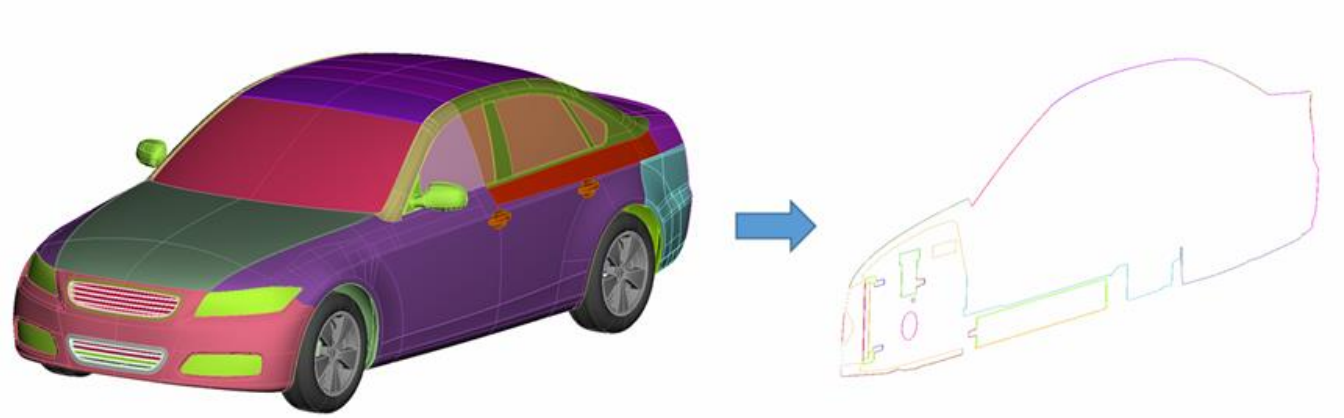


Figure 1: 2D centerline section of the e-DrivAer Notchback model

In order to generate a database of different shapes the morphing approach was used: a geometrical parametrization of the CAE model was built in BETA CAE ANSA pre-processor. This tool allows to re-design the shape of the vehicle by applying local modifications on surfaces and mesh. A qualitative overview of the involved parameters is available in Figure 2.

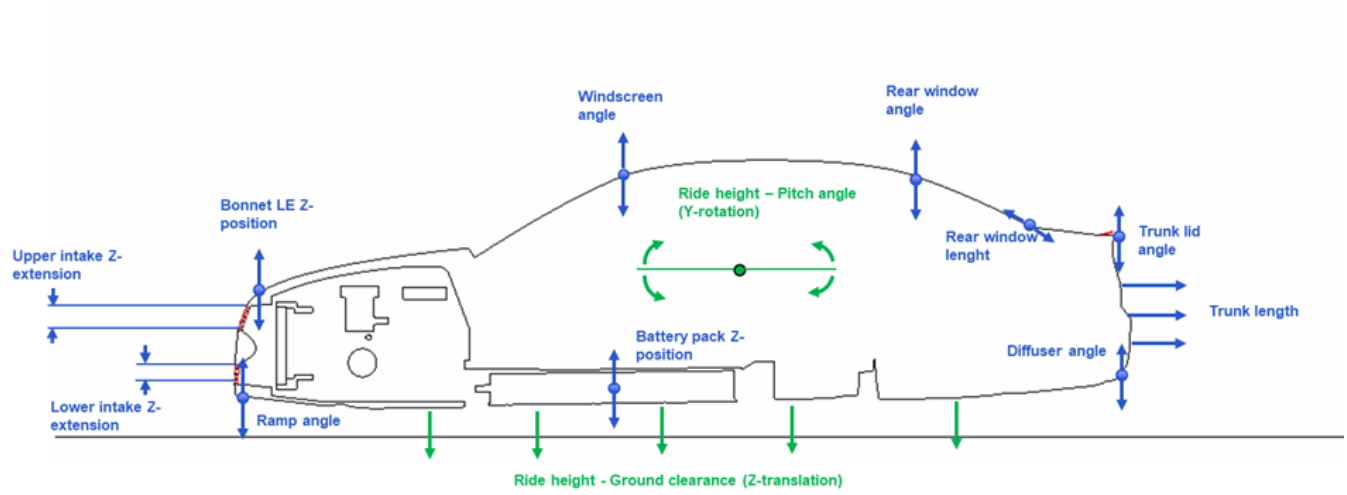


Figure 2: Qualitative description of model parameterization.

Allowed variation range for each modification was defined as a compromise between maximum achievable shape change and quality of the resulting mesh. Morphing was applied

on the surface mesh. The parametrization globally consisted of thirteen different continuous shape parameters. Combined together they allowed to generate shapes significantly different, in terms of aerodynamic performance, one from another. An example of explored parametric shape is given in Figure 3**,** where some random combinations of variables are compared with the original configuration.

Figure 3: Example of randomly generated shapes obtained by surface morphing .

A full list of involved parameters, including maximum range of variation and a brief description is available in Table 1. Surface morphing was also forced to use quantized levels, indicated in the column “step”, in order to ensure that appreciable differences exist between different levels. All reported values have to be considered as variations with respect to the original position.

Table 1: List of morphing parameters included in the parameterization.

ID	Parameter	Type	Min Value	Max Value	Step	Description
1	D_upperGrilleZ	DFM EDGE FIT	-28 mm	20 mm	4 mm	z-extension of the front upper air intake
2	E_lowerGrilleZ	DFM EDGE FIT	-30 mm	30 mm	4 mm	z-extension of the front lower air intake
3	F_bonnetLEz	DFM EDGE FIT	-32 mm	48 mm	8 mm	z-translation of bonnet leading edge, in order to modify the hood angle
4	G_windscreenAngle	DFM EDGE FIT	-25 mm	30 mm	5 mm	z-translation of the windscreen trailing edge, in order to modify the glass angle
5	H_rampAngle	DFM EDGE FIT	-50 mm	50 mm	10 mm	z-translation of front bumper leading edge, in order to modify ramp angle
6	I_batteryPackZ	DFM EDGE FIT	-10 mm	30 mm	4 mm	z-translation of the underbody battery pack
7	L_rearWindowAngle	DFM EDGE FIT	-25 mm	25 mm	5 mm	z-translation of the windscreen leading edge, in order to modify the glass angle
8	M_rearWindowLength	DFM EDGE FIT	-45 mm	45 mm	6 mm	extension of rear window length

9	O_trunkLidAngle	DFM	EDGE FIT	-30 mm	48 mm	6 mm	z-translation of trunk trailing edge to modify the lid angle
10	P_diffuserAngle	DFM	EDGE FIT	-50 mm	50 mm	10 mm	z-translation of rear bumper lower edge, in order to modify diffuser angle
11	Q_trunkLength	DFM	EDGE FIT	-55 mm	55 mm	11 mm	x-extension of vehicle trunk
12	R_rideHeightZ	DFM	TRANSLATE	-25 mm	50 mm	5 mm	Rigid z-translation of the vehicle in order to modify ground clearance
13	S_rideHeightAngle	DFM	ROTATE	-1 °	1 °	0.2 °	y-rotation of the vehicle in order to change pitch angle

All morphing parameters were defined as “Direct Fitting Movement” type (2), a system able to move parts of the model, both geometry or mesh, as not deformable body, while the surrounding elements can absorb the deformation without damaging the continuity of the model. As an example, definition and application of parameter 9, trunk lid angle, is provided in Figure 4, showing original and min/max positions as described in Table 1.

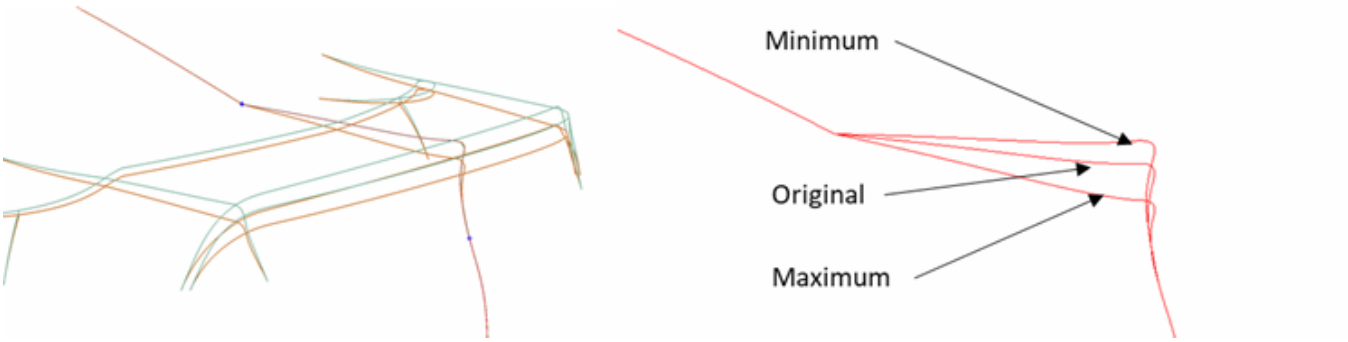


Figure 4: Morphing of trunk lid angle with the O_trunkLidAngle parameter.

The parametric model was then used to generate different versions of the DrivAer model to be evaluated by CFD simulations, according to the workflow defined in Figure 5, from which drag coefficient, pressure and velocity fields would be extracted to feed the reduced order models, according to the procedure described in the next chapter. The CFD simulation setup is described in Section 2.2.

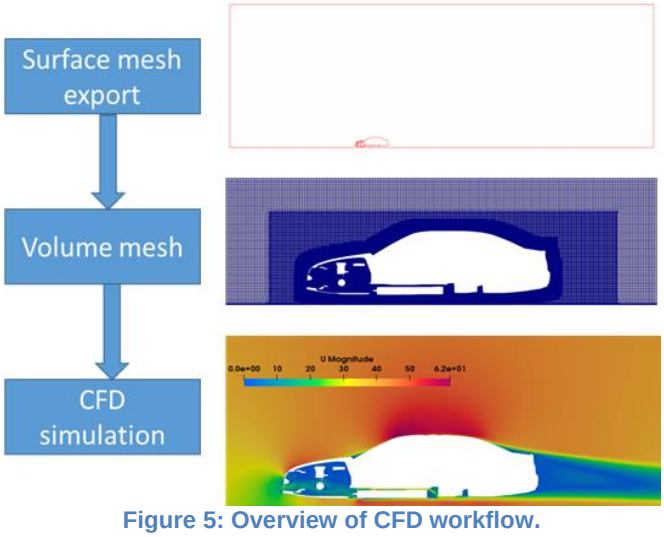


Figure 5: Overview of CFD workflow.

2.2. Generation of training data: Workflow

2.2.1. DOE

In the original database (DrivAer 2D), which consists of approximately 1000 samples, apart from the several control points which are float numbers, there are also Boolean features (e.g. upper and lower grille shutters) for describing the geometry of each sample.

A SOBOL sequence (3) was used to generate the dataset of geometries, for a total of one thousand different shapes. A SOBOL sequence is a quasi-random sequence and its aim is to fill the sampling space in a uniform way, without recognizable patterns. An example of generated points distribution is provided in Figure 6. For the sake of simplicity, only the first three parameters are shown. Fifty additional geometries were generated as spare data, in case of unfeasible designs in the original dataset.

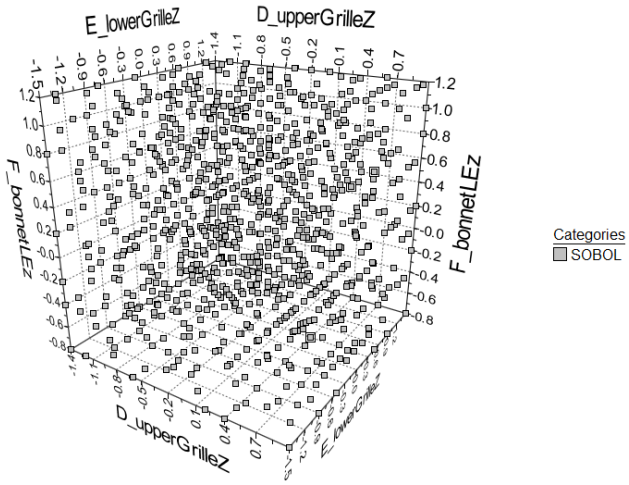


Figure 6: Distribution of points generated as a SOBOL sequence for paratemers D_upperGrilleZ, E_lowerGrilleZ, F_bonnetLEz.

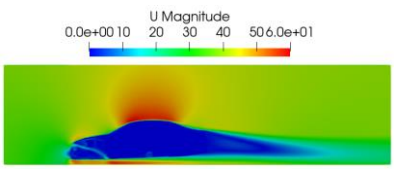
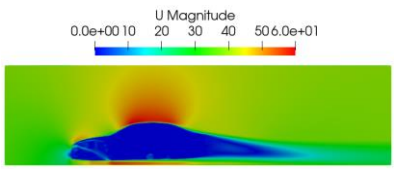
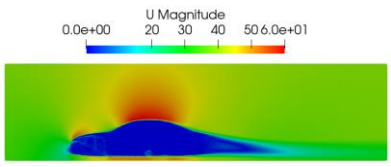
In the interest of estimating the influence of such Boolean features in the accuracy of the NN, two different datasets were generated, consisting of the same combination of parameters, differentiating each other just with an open or closed lower air intake. The two datasets are referred respectively as “OpenAGS” and “ClosedAGS”. A visual example is provided in Figure 7.

Figure 7: Example of shapes included in the datasets, both from "ClosedAGS" and "OpenAGS" configuration.

Additional datasets, mixing shapes from the two original datasets, were then created to be tested with the neural network workflow, as described in Section 3.2.2.

An example of aerodynamic results obtained within the database is presented in Table 2, where drag coefficients and velocity magnitude distribution are shown for the baseline, the worst and the best shape from the "ClosedAGS" dataset. As can be seen, the variability within the considered set of data is of the order of 120 drag counts, indicating that the prescribed parametrization was able to produce shapes significantly different from an aerodynamic point of view. Velocity magnitude visualization confirms this statement also in terms of flow fields.

Table 2: Overview of aerodynamic variability within the "ClosedAGS" dataset.

ID	sample 765 (Worst)	Baseline	sample 586 (Best)
C_D	0.279	0.214	0.159
Shape			
Magnitude of velocity			

2.2.2. Automated workflow

In this section, the workflow prior to the training procedure and the framework for automating it is described. This framework is robust and capable of handling different types of datasets, and either 2D or 3D geometries. Due to the variety of the database types, it is important to have a consistent framework for both 2D and 3D cases which will require minimal user input, such as the sampling resolution. The workflow in brief includes 1) geometry generation (see Section 2.1), 2) meshing, 3) setup CFD configuration, 4) running the CFD solver, 5) resampling and additional post-processing of the results, 6) transformation of the data obtained by CFD to a format compatible with the ML library, 7) training of the Neural Network (NN), 8) testing of the generated model and 9) prediction of the output variables, comparison with the targets, which were derived by CFD and statistical analysis.

Given the variety of the case dimensions and the different machine types the user can have at his disposal, the present workflow prior to training can be summarized into two different levels of abstraction. The first one refers to the case dimensions (2D or 3D), whereas the second one corresponds to the type of the machine and can be either a local workstation or a cluster with a job scheduler. Different pre- and post-processing utilities run for 2D and 3D cases respectively, such as grid generation, which is solved in the framework with the creation of a parent class that contains the basic interface and two inherited classes with different implementations depending on the case dimensions. In case the user is using a local workstation, the workflow is rather straightforward for several applications to run in parallel. However, when using HPC facilities, a task manager has to be incorporated into the workflow to limit the number of the submitted jobs to a certain amount and not submit a new one, unless a previous job is finalized. For that purpose, the two different classes regarding the machine type have been implemented and they have as an argument the case dimensions class in their constructor (see Figure 8). (4)

Figure 8: Logic diagram of the main classes called. The blue arrows indicate inheritance, whereas the green arrows show the path of the workflow. The most important classes are shown in red and conventional-type arguments are in black. In case3D class, createGeometry variable is an additional argument as the geometry for the 3D rectangular boxes is generated using pyDOE (4), rather than following the methodology described in Section 2.1.

An OpenFOAM case directory is created for each geometry provided in the dataset, based on the predefined dictionaries and then, an unstructured computational grid is automatically generated, which is refined around the solid surfaces. The case setup follows, where apart from setting the boundary conditions and the required options for each solver, the computational domain is decomposed to the corresponding number of subdomains for running the CFD solver in parallel. A segregated pressure correction (SIMPLE) solver in the finite volume framework was utilized (simpleFoam). In order to model turbulence, RANS simulations have been performed by employing the k-omega SST turbulence model, which behaves reasonably well, considering the purpose of the activity, for external aerodynamics. Once the solution is converged to the required residual, post-processing of the results takes place.

Due to current memory limitations in modern HPC facilities, the NN training was performed for cases of lower resolution compared to the original solution acquired by CFD analysis and therefore, a solution resampling algorithm had to be developed and incorporated into HELYX

CFD package (5) provided by ENGYS. For instance, most of the ML training were performed in Cirrus HPC system housed at EPCC, where each GPU compute node contains four NVIDIA Tesla V100-PCIE-16GB (Volta) GPU accelerators. Even by making use of such state of the art GPU accelerators, preliminary investigations show that the memory requirements for training 3D datasets exceed the available GPU accelerator memory (*16 GB*) even for spatial sampling resolution of 128^3 and a batch size of *12* cases. Having in mind that the smaller the batch size, the more inaccurate the gradient estimation will be, a compromise between accuracy and memory resources must be done. Further investigations on shared memory applications will be performed in the next steps of the work package, in order to verify if the limitation could be overcome by multiplying the hardware. The purpose of the resampling algorithm is to approximate the flow fields needed as input (solid mask) and output channels (pressure and velocity vectors) for the NN, based on the flow fields solution at the cell centers of the finer unstructured CFD grid. In essence, the above-mentioned channels are in a structured, equally spaced grid-like format, but not necessarily equally spaced among different dimensions. Although a similar utility already exists in OpenFOAM (mapFields), the current feature is much more computationally efficient, by using the K-nearest Neighbours algorithm (6), (7) instead of the Octree methodology.

Apart from resampling, post-processing the CFD results also includes file-folder manipulation and obtaining the aerodynamic coefficients for each case, as well as exception handling of diverging solutions to disregard such cases from the training database. The remaining dataset is randomly split into two sets, the train and the test datasets. The size of the former is approximately *80%* of the whole dataset and the remaining *20%* is the size of the latter. Finally, the data are transformed and compressed into a numpy (8) format of multiple dimensional arrays (.npz files) named *npOutput*. The first dimension of the array consists of the inputs (n_{inputs}) and the outputs ($n_{outputs}$) used for the training of the NN (see Section 3.2), while the following ones denote the sampling resolution in the x, y and z dimensions (for 3-D cases): $npOutput[n_{total}][n_x][n_y][n_z]$; $n_{total}=n_{inputs}+n_{outputs}$.

There is also the post-processing stage after the prediction of the flow field and the drag coefficient, where statistical analysis of the errors is conducted and the predicted flow field is visualized. For the latter, a post-processing utility in HELYX CFD has been developed in order to transform the predicted results which are in numpy format to OpenFOAM format.

2.3. Error Metrics

In order to compare the solution obtained by the CFD analysis with the predicted flow fields, either derived by the deterministic method or by the ML approach, the following error metrics are being defined and have been used throughout this report for consistency. It has to be clarified here that the CFD solutions used for comparison are the sampled fields, ensuring that way one-to-one comparison between corresponding points. If the CFD solution at the cell-center had been selected, an additional error would have been introduced due to the approximation of the cell-centered solution to the sampling points. The errors for the pressure and the velocity magnitude are calculated by finding the spatially averaged error of the sampled grid points for each case and then, the mean value among all cases in the test dataset is calculated. The final error expressions are as follows:

$$p_{error} = \sum_{s=1}^{N} \left(\sum_{i=1}^{P} \frac{| p_{i,s}^{AI} - p_{i,s}^{CFD} |}{| \mathbf{U}_{\infty} |^2} \right) \Big/ NP \qquad \text{(Eq. 1)}$$

$$U_{error} = \sum_{s=1}^{N} \left(\sum_{i=1}^{P} \frac{| \mathbf{u}_{i,s}^{AI} - \mathbf{u}_{i,s}^{CFD} |}{| \mathbf{U}_{\infty} |} \right) \Big/ NP, \qquad \text{(Eq. 2)}$$

where in the above equations, $\boldsymbol{u}_{i,s}$, $p_{i,s}$ are the velocity vector and the pressure at point *i* of case *s*. *N* stands for the number of samples in the test database and *P* is the number of the total grid points for each case:

$$P = n_x \cdot n_y \cdot n_z,$$

where n_x, n_y, n_z are the sampling points in the *x, y* and *z* directions respectively. The AI and CFD superscript denote the output of the NN and the target respectively. Regarding the error estimation for the Lift and Drag prediction, it is defined as the case accumulated relative error for each aerodynamic force; for instance, the error on aerodynamic drag is estimated as follows:

$$D_{error} = \sum_{s=1}^{N} \frac{| D_s^{AI} - D_s^{CFD} |}{| D_s^{CFD} |} \qquad \text{(Eq. 3)}$$

Based on the above error expressions, the standard deviation of the spatial error, the minimum, and the maximum spatial errors have been recorded to have an estimation of how much the spatial error can vary from case to case.

3. Methods

3.1. Deterministic Method: POD + Interpolation

The deterministic method investigated in this report follows the well-known “POD+I” paradigm (see Figure 9):

1. Decomposing the entirety of training snapshots into its main variational modes (“base modes”) via Proper Orthogonal Decomposition (POD (9)),
2. Truncating the thus obtained POD basis at a suitable number of modes determined usually by their relative information content (RIC), and
3. Using an interpolation method for the base mode coefficients to construct the flow field for any unseen geometry.

This procedure was shown to deliver robust field predictions with controllable accuracy for aerospace applications (9; 10), and first attempts were made to introduce it in automotive contexts (11; 12; 13). Those studies were restricted to rather low-dimensional parameter spaces with dimensionalities well below 10 and comparatively small sample numbers. Since the interpolation part of POD+I suffers from the “curse of dimensionality” (14), it is an open question, if this method delivers results of satisfactory accuracy also for the 14-dimensional test case under investigation here. POD+I is therefore the natural choice as a deterministic method to be applied in UPSCALE: both in order to verify its performance in higher-dimensional parameter spaces and as a reference for the non-deterministic approaches pursued within this project.

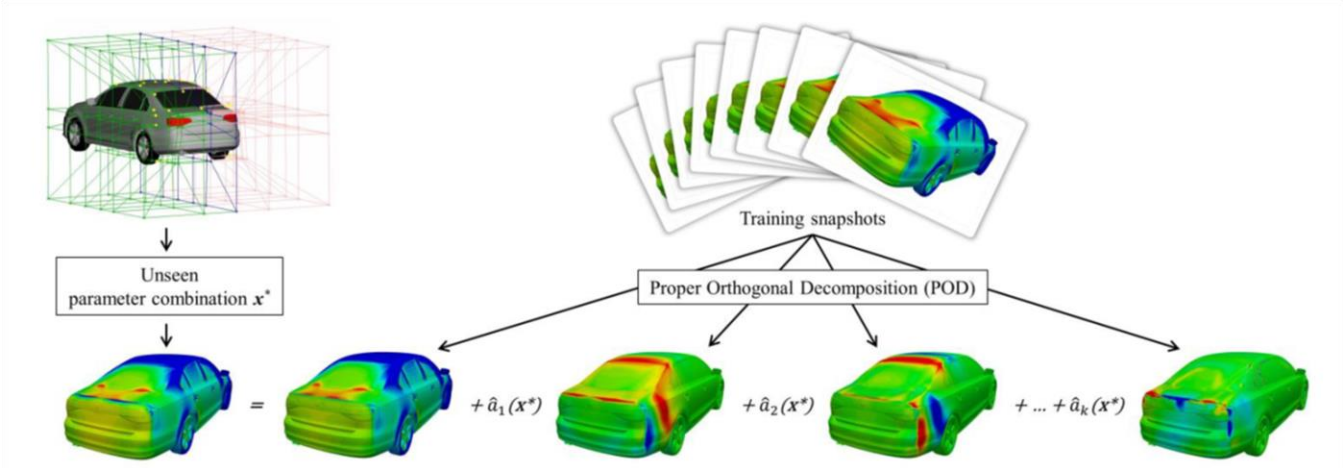


Figure 9: The workflow of the POD+I method: Shown are the pressure POD modes. Their linear combination with suitably interpolated POD base mode coefficients (a_j)ˆserves as the pressure prediction for an unseen geometry represented by the parameter set x^* (reproduced from (12)).

3.2. Non-deterministic Method

In this section, the machine learning methodology used for predicting the flow characteristics is described. A NN can, in general, represent a non-linear function of the form of a function. The main concept is to model the original function with a network of connected nodes which form layers. For instance, for a layer *l* of the network, the output of the *k-th* node $\alpha_{k,l}$ is:

$$\alpha_{k,l} = g\left(\sum_{j=0}^{n_l-1} w_{kj,l-1}\alpha_{j,i-1}\right), \qquad \text{(Eq. 4)}$$

where *g* is the activation function and n_l is the number of the nodes in the *l* layer. The unknown is the weight vector ***w*** which approximates the original function.

The fields of the velocity vector, the pressure, and the turbulent quantities, accompanied by the main aerodynamic coefficients can sufficiently describe the vehicle aerodynamics and can, therefore, lead towards preliminary EV design. For proof of concept purposes, the vehicle drag coefficient, the pressure and the velocity fields around the vehicle have been chosen as targets/outputs of the NN, without restricting the applicability of the present methodology to these specific variables. Turbulence quantities or other aerodynamic coefficients, such as moment coefficients can potentially be selected as additional targets aiming to a more complete description of the flow field and vehicle aerodynamics. In the current work, the drag coefficient has been chosen as a separate output (a single number for each case in the database), rather than estimating it from the predicted pressure and velocity fields which would have accumulated the error of both. Furthermore, in that approach the viscous stress tensor would have been roughly approximated by the velocity vector at the coarse sampling grid, significantly increasing the error in the prediction that way. In order to predict the pressure and the velocity vector fields, a regression type NN has been employed, whereas for the drag prediction, a separate NN of classification type architecture has been implemented. Although the drag prediction algorithm doesn't output the pressure and the velocity fields, and hence, they are not needed as targets, a single database as described above (see Section 2.2) is used for both NN architectures. The main reason behind that is consistency, avoiding

generation and storage of additional data that already exist. Therefore, these outputs are not used as targets for the training process in the classification algorithm.

The velocity vector and the pressure fields are non-dimensionalized by the freestream velocity:

$$\hat{\mathbf{u}} = \mathbf{u} / \mid \mathbf{U}_{\infty} \mid, \qquad \text{(Eq. 5)}$$

$$\hat{p} = \frac{p - \bar{p}}{\mid \mathbf{U}_{\infty} \mid^2}, \qquad \text{(Eq. 6)}$$

removing that way the quadratic influence of velocity from the target data (15). Besides, the pressure is offset by the mean pressure for each pressure sample, as the pressure offsets in the targets are not correlated with the inputs (15). Finally, the targets are normalized to [-1, 1] to minimize errors due to numerical precision during the training stage (15). It has to be clarified here that once the outputs are predicted, they are transformed back to their original form and therefore, the error is measured for the actual pressure and velocity fields.

The parallelization capabilities of machine learning libraries are limited by CPU and GPU parallel processing, prohibiting training of higher resolution 3D samples due to CPU computational inefficiency and GPU memory limitations respectively. Based on recent performance studies conducted at ENGYS, the clock time when using thread parallelism is increased by approximately *20* times, for training NN's relevant to this study. However, it is feasible to train samples of higher resolution, due to the significantly larger memory available at CPU nodes. To give an estimation of the CPU and GPU memory specifications, the RAM capacity in each CPU node at Cirrus HPC is 256 GB, compared to a GPU node which has four Tesla GPU, with 16 GB memory each. PyTorch (16) machine learning framework was selected for consistency reasons, as both PyTorch and HELYX CFD are being built with CMake.

3.2.1. Network details

Drag prediction algorithm

Based on previous work conducted at ENGYS, classification methods have been proved to predict the drag coefficient with higher accuracy compared to the regression family of methods, even though the drag coefficient is a continuous variable and a regression type NN would seem more appropriate. The input of the NN is the geometry mask on the sampling grid as described above, whereas the output is the probability for the drag of the predicted case to belong in each class.

Firstly the maximum drag coefficient C_D, max among all samples in the database is found. The possible range of the drag coefficient is, where $C_{D,min}=0$, and α is an amplification factor, currently set to 1.2, to account for the event of a predicted case to have drag coefficient higher that $C_{D,max}$. The next step is to classify the different cases from the dataset to N equally spaced classes based on their drag coefficient. $N=600$ was selected as a default value for the number of classes. Therefore, a case with drag coefficient C_D has label i:

$$i = int\left[(C_D - C_{D,min}) \cdot N / (a \cdot C_{D,max} - C_{D,min})\right], \qquad \text{(Eq. 7)}$$

which means it belongs to the *i+1* class.

The NN created here consists of multiple convolution blocks, similar to image classification networks. The NN algorithm has been implemented in a generic way offering the option to the user to select the number of the layers (convolution levels). However, it has to be noted that the number of levels cannot exceed the minimum sampling resolution of the physical dimensions:

$$Layers_{max}=log_2 min(n_x, n_y, n_z), \quad \text{(Eq. 8)}$$

where n_x, n_y, n_z are the initial sampling resolutions in the *x*, *y*, and *z* directions respectively. The logarithm of 2 was calculated since, after each convolution block which will be later explained, the spatial resolution in each dimension is reduced to half. Depending on the physical dimensions of the problem, 2D or 3D convolution blocks are created. From an implementation point of view, the 2D NN is the base class and the 3D NN is the derived class that inherits the necessary common features from the parent class. After performing parametric studies, the number of the layers of the constructed NN is usually set as the number of the maximum layers reduced by 1 or 2, in order to avoid overfitting of the model.

Each convolution block consists of a 2D or 3D convolution operator, depending on the samples' physical dimensions, a rectified linear unit (ReLU) and a down-sampling process (max pooling) which acts as a generalizer while at the same time it reduces the number of the unknown parameters.

The number of the output channels after each convolution block (each convolution block has one convolution layer) can be determined by the following empirical equation:

$$channels_{out} = 2^{layer+4}; if \quad layer > 1; else \quad 2^4. \quad \text{(Eq. 9)}$$

This equation was derived to automatically create *M* layers. The number of the sampling grid points in the x-direction (W_{out}) after each convolution or pooling operation is given by:

$$W_{out} = int\left(\frac{W_{in} - D \cdot (K-1) - 1 + 2 \cdot P}{S} + 1\right), \quad \text{(Eq. 10)}$$

where *K* stands for the convolving kernel size, *D* is defined as dilation and it is the spacing between kernel elements, *P* is padding and *S* is the stride of convolution. Since each convolution block consists of a convolution and a max-pooling unit, the above equation is applied twice within the convolution block. For the 2-D or 3-D convolution $K_c=5$, $S_c=1$, $P_c=2$, $D_c=1$ were chosen to keep the number of the sampling points the same, whereas for the 2-D or 3-D max-pooling $K_p=2$, $S_p=2$, $P_p=0$, $D_p=1$ were chosen aiming to reduce each dimension's sampling points by half. Overall, after each convolution block, the number of sampling points is reduced by a factor of 2 (see Figure 10).

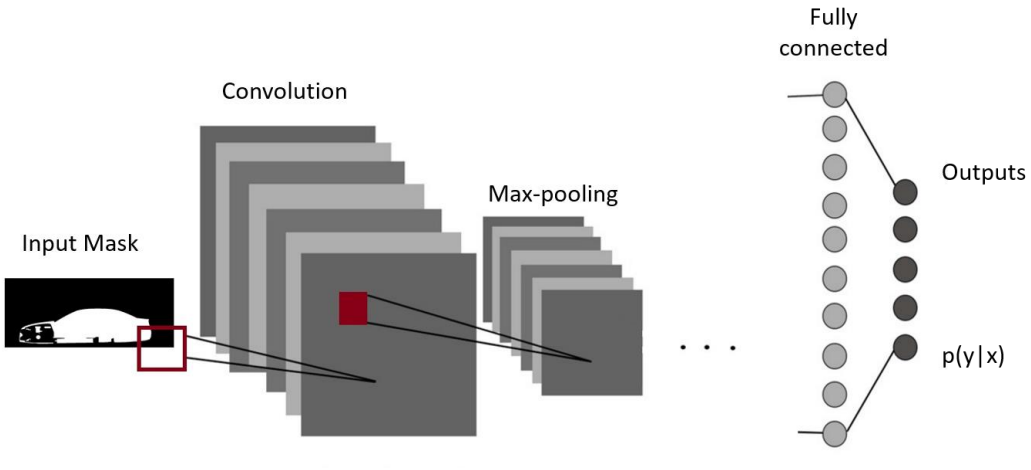


Figure 10: Diagram of a generic convolution NN for classification problems. Only 1 convolution block is shown consisting of a convolution layer and a max-pooling operator. The spatial resolution is reduced to half after each pooling operator. For demonstration purposes, only 7 channels were created after the first convolution layer instead of 16

The same equation can be applied for each direction (*W, H, L*) by replacing the output (out subscript) and the input (in subscript) channels accordingly. If the Kernel, Dilation, Padding or Stride size is not the same in each direction, these parameters are also modified accordingly. In the drag prediction training that has been performed so far, the same parameters were used for each spatial dimension. Initially, for the first convolution block calculations, the spatial resolution is given by $[W_{in}, H_{in}, L_{in}]=[n_x, n_y, n_z]$ and based on that, the output size is calculated from Eq. 9. The next convolution block has as an input the output of the previous one and the output size is calculated likewise. After the last convolution block, the linear unit follows, which applies a linear transformation of the form $y=xAT+b$. The output tensor of the last convolution block is flattened to a single M_{x1} tensor, where $M=channels_{out}W_{in}H_{in}L_{in}$. A second linear transformation follows, which has as output the number of the classes *N*. In order to change the output of the network to the probability distribution over classes instead, a softmax activation function is placed at the output layer:

$$\sigma(x)_j = \frac{e^{x_j}}{\sum_i e^{x_i}}. \qquad \text{(Eq. 11)}$$

In classification problems, in the forward pass the softmax function is usually used alongside the negative log likelihood loss (NLLoss), which is the negative natural logarithm of the probability distribution of the prediction. Next, the weights are adjusted in the backward pass by computing the gradient of the loss function with respect to the learning parameters. Several different methods (optimizers) are used for updating the trainable parameters, such as the stochastic gradient descent (SGD) (17) and AdaGrad (18).

The purpose of the above described NN algorithm is to predict the drag coefficient of a given geometry (input) by providing a probability distribution for the drag to belong in each class. Once this is accomplished, for each class *i*, the drag coefficient which is within the class' range is approximated by:

$$C_{D,i} = i\frac{a \cdot C_{D,max} - C_{D,min}}{N} + C_{D,min}. \qquad \text{(Eq. 12)}$$

In practice, the above 2 linear units can be combined resulting in the same outcome, they have been split for implementation purposes. For instance, by using a sampling resolution of 256x256, the 256x256x1 initial tensor after 5 convolution blocks becomes of size 8x8x512. After the first linear unit, it becomes a vector-like tensor of dimensions 32768x1, whereas the final output features of the second linear unit are 600 (number of classes). The parameter values for setting up the NN are shown in Table 3**.**

Table 3: Tuning of the parameters needed in the classification problem. The beta1 and beta2 parameters are needed by Adam optimizer, the finalLearningRate is needed by adaBound, weight_decay is used in adaGrad, whereas momentum is used for achieving faster convergence in the stochastic gradient descent algorithm.

Classification NN parameters	
[nx, ny, nz]	*[256, 256, 1]*
amplifyRate	*1.2*
nclass	*600*
batch_size	*16*
epochs	*400*
levels	*5*
optimizer	*adaGrad*
learningRate	*0.01*
weightDecay	*0*
finalLearningRate	*0.1*
beta1	*0.5*
beta2	*0.999*
momentum	*0.5*

Flow prediction algorithm

A regression type architecture has been employed for predicting the flow fields. More specifically, a U-Net architecture, which has an encoder-decoder architecture has been implemented (19). The algorithm is generalized for 2D and 3D physical problems and the number of the down-sampling levels, which is the same as the number of the up-sampling levels, is being defined by the user, similar to the classification algorithm, having as only restriction the initial spatial resolution from Eq. 8.

The U-Net algorithm starts with a single convolution layer, the down-sampling convolution blocks come next. The latter consists of an activation function, a leaky version of a ReLU (rectified linear unit), followed by a convolution operator which is also followed by batch normalization. For the down-sampling convolution blocks, K_c=4, S_c=2, P_c=1, D_c=1 were chosen to reduce each dimension's sampling points by a factor of 2. Due to the increasing number of channels created, large scale abstract information is extracted. After reaching the maximum depth layers, the opposite process takes place by increasing the sampling points in each dimension and by reducing the number of the feature channels. The decoding starts with an

equal number of up-sampling convolution blocks as the down-sampling ones. The former consists of a ReLU activation function, an up-sampling operator, a convolution and a batch normalization operation. For these blocks, $K_d=3$, $S_d=1$, $P_d=1$, $D_d=1$ were selected to keep the size of the tensor constant. However, after each block, a concatenation operation is performed doubling the number of sampling points in each physical direction. Then, one transposed convolution block is introduced which consists of a ReLU in conjunction with a transposed convolution layer. For the transposed convolution, $K_t=4$, $S_t=2$, $P_t=1$, $D_t=1$ were selected increasing the size of the tensor by a factor of *2* (see Figure 11 for a generated NN of *7* depth levels). The number of the output channels after each down-sampling convolution block can be determined by the following empirical equation, which is slightly modified compared to the one used for the drag prediction to apply a cut off upper limit:

$$channels_{out} = 2^{layer+4}; if \quad layer \leq 4; else \quad 2^8 \qquad \textit{(Eq. 13)}$$

The spatial resolution in the output of each convolution block can be calculated by applying Eq. 9 twice, as both convolution and pooling operations exist, whereas for the transposed convolution block the grid points in each dimension are calculated by:

$$W_{out} = int\Big((W_{in} - 1) \cdot S - 2 \cdot P + D \cdot (K - 1) + 1\Big) \qquad \textit{(Eq. 14)}$$

A simple L_1 loss function is used here in the forward pass to measure the divergence from the target:

$$L_1 = mean\Big(\{l_1, ..., l_N\}^T\Big), \quad l_n = \mid output_n - target_n \mid \qquad \textit{(Eq. 15)}$$

Here, *N* is the batch size, as outputs are considered to be the pressure and velocity fields predicted by the NN after denormalization, and target are the fields computed by CFD. Finally, the trainable parameters are updated in the backward pass. The main methods for training, testing and prediction processes are organized into 3 classes, having as a base class the training. The test class is derived from the training, and similarly, the prediction class is derived by the testing class. The parameter values for setting up the NN are shown in Table 4.

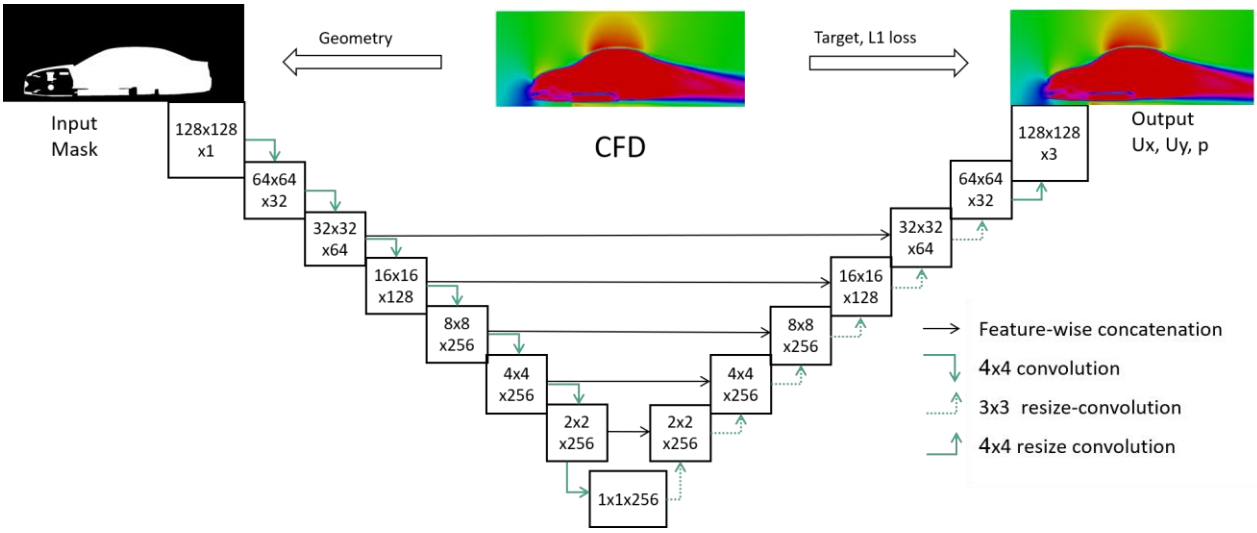


Figure 11: Diagram of the U-Net with 7 layers of convolution for spatial sampling resolution of 128x128, resulting in 7.567M trainable parameters.

Table 4: Tuning of the parameters needed in the regression problem. The beta1 and beta2 parameters are needed by Adam optimizer, whereas the finalLearningRate is needed by adaBound and weight_decay by adaGrad.

Regression NN parameters	
[nx, ny, nz]	*[256, 256, 1]*
batch_size	*12*
iterations	*50000*
levels	*7*
optimizer	*Adam*
learningRate	*0.0006*
weightDecay	*0*
finalLearningRate	*0.1*
beta1	*0.5*
beta2	*0.999*
momentum	*0.5*

3.2.2. Training

It is a good practice in deep learning to select the training and the testing subsets of each database by randomly splitting the database so that 80% of the total number of the samples are used for training and the remaining 20% are used for testing. This provides a sufficient number of cases for training and a representative sample database for testing. In this report, as the testing set, the training set will be used as a means of preliminary validation of the model.

Apart from the OpenAGS and the ClosedAGS, a mixed dataset (mixed-small) was created by combining the above two databases (see Section 2.2). In order to make the numerical experiment independent of the database size, the mixed dataset was created by randomly selecting half of the OpenAGS cases and half of the ClosedAGS grilled shutter cases. An additional mixed (mixed-all) database was created by simply combining the OpenAGS and ClosedAGS databases, resulting in a database of double size compared to the previously mixed dataset. This will be useful to evaluate how the sensitivity of the results depends on the database size.

4. Results

4.1. Deterministic Method: POD + Interpolation

The results in this section were obtained with the Reduced-Order-Modeling (ROM) toolbox NAVPACK from NAVASTO®. We restrict ourselves to applying NAVPACK to the OpenAGS and ClosedAGS data sets, since mixed datasets would require two separate ROMs for each value of the Boolean parameter and make a comparison with the non-deterministic methods more difficult.

For the sake of a valid comparison between both methods (to be reported in D2.3), we made sure that the input data is exactly the same for both approaches: the POD+I method takes the same snapshots as the NN, resampled in the same way to the same Cartesian mesh. The geometry representation is, however, not the binary mask of the NN, but the ANSA morphing parameters introduced above.

The POD base modes are calculated separately for velocity U and pressure p by performing a Thin Singular Value Decomposition (TSVD (20)) of the snapshot matrix. They constitute a low-dimensional linear subspace of the entirety of CFD training solutions within the design space. The squared singular values of the TSVD equal the eigenvalues λ_j of the training snapshot correlation matrix (21) and are each associated to a POD base mode. They can be seen as a measure of its information content. The relative information content (RIC) of the first k eigenmodes can therefore be computed as

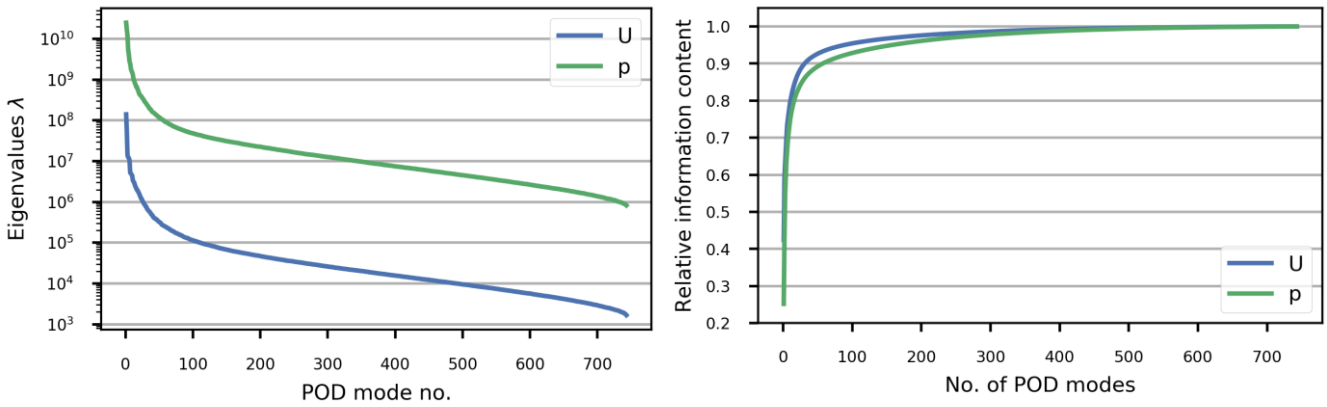
$$I(k) = \frac{\sum_{j=1}^{k} \lambda_j}{\sum_{j=1}^{n} \lambda_j},$$

with n being the total number of training snapshots.

Figure 12a and b show, respectively, the eigenvalues λ_j in descending order and the corresponding RIC $I(k)$ for U and p of the OpenAGS data set. The eigenvalue spectrum exhibits the typical strong drop within the first few modes, allowing to reach a RIC of more than 90% already with less than 100 modes. In particular, to reach the commonly target RIC of at least 95%, the following number of modes are required for the respective data sets:

- OpenAGS: 88 modes for velocity U, 157 modes for pressure p,
- ClosedAGS: 81 modes for U, 148 modes for p.

In order to have a minimum RIC of 95% for both data sets, we therefore truncated the POD base after 88 modes for U and after 157 modes for p for the further analysis.



a) Eigenvalues b) Relative Information Content

Figure 12: Eigenvalues (a) and relative information content RIC (b) for velocity and pressure of the OpenAGS dataset.

After computing the POD base modes and truncating them suitably for the purpose of dimensionality reduction, the only missing step to complete the “POD+I” model reduction procedure is the interpolation of the POD base coefficients. Common choices for interpolation are Kriging (22) and Thin-Plate-Spline (23). According to our previous experience with these two methods for moderate parameter space dimensionalities, their achievable accuracy is of

similar order. As TPS does not require hyperparameter tuning and is significantly faster for the high number of geometric parameters of this test case, we opted for TPS in the current investigation.

Table 5: Errors for pressure and velocity when reconstructing the training data using 157 and 88 POD modes, respectively. The errors are specified as mean+/-standard deviation over all the training snapshots.

Dataset	Error pressure [%]	Error velocity [%]
OpenAGS	*0.19 +/- 0.02*	*0.35 +/- 0.05*
ClosedAGS	*0.16 +/- 0.02*	*0.33 +/- 0.05*

Table 5 summarizes the errors when reconstructing the training data via TPS based on the truncated POD basis as detailed above with 157 pressure and 88 velocity modes, respectively, for both the OpenAGS and the ClosedAGD data sets. While the pressure error is below 0.2% for both data sets, the velocity error is slightly higher, but still at a satisfactory level of less than 0.4%. In addition, Figure 13 provides some error statistics in the form of histograms for pressure and velocity of all 800 training samples of the ClosedAGS set. The histograms approximately follow a Gaussian distribution with a standard deviation of 0.02 and 0.05%, respectively, which *quantitatively* underpins the validity of the chosen ROM approach.

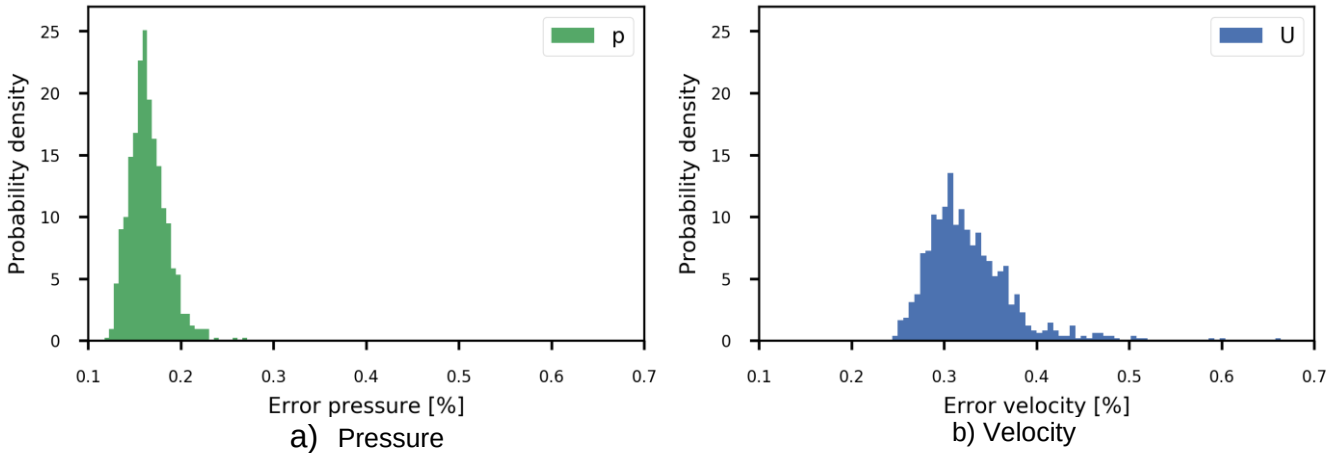
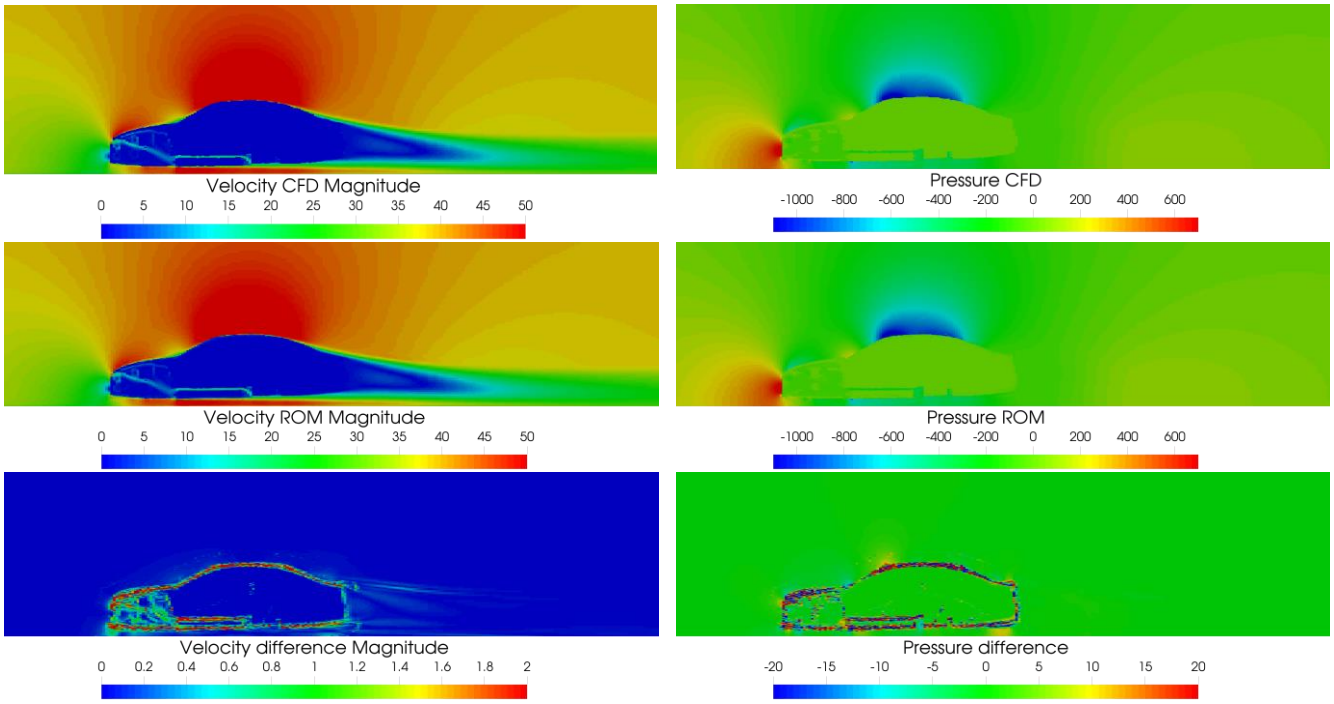


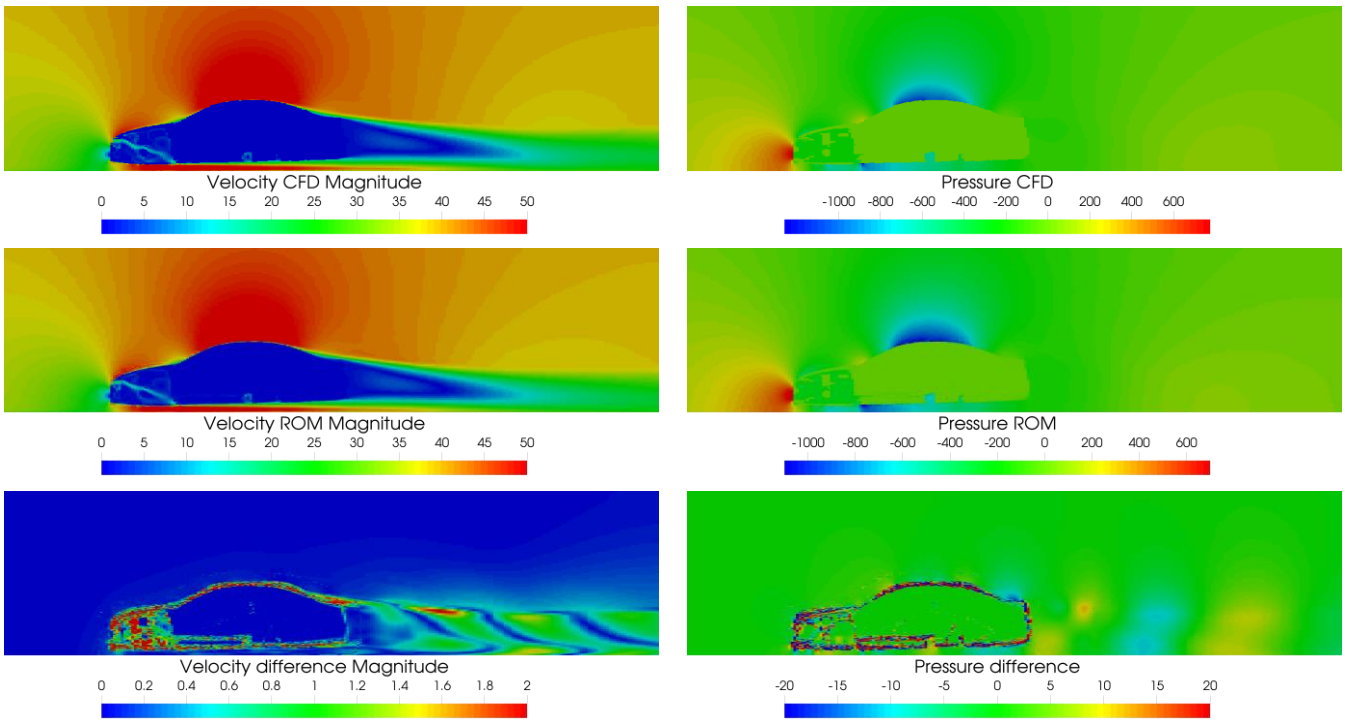
Figure 13: Histogram of the pressure (a) and velocity (b) errors for the 800 training samples of the ClosedAGS dataset. 157 and 88 POD modes were used for pressure and velocity, respectively.

In order to *qualitatively* illustrate how accurately the training data is represented by the constructed ROM, we show a few representative contour plots of the actual fields in the following. We thereby restrict ourselves to the ClosedAGS data set, which has the smaller combined (pressure + velocity) error of 0.16% + 0.33%. Within this data set, sample #344 was found to have the lowest combined reconstruction error (0.38%), whereas sample #626 was the hardest to reconstruct with a combined error of 0.94%. The true flow field from the CFD training simulations, the ROM-predicted flow field and their difference are depicted in Figure 14 and Figure 15.



a) Velocity (in m/s) b) Pressure (in Pa)

Figure 14: Comparison of fields from CFD (top row) with ROM predictions (middle row) for the training sample with lowest combined error (#344). The difference between ROM prediction and CFD is shown in the third row.



a) Velocity (in m/s) b) Pressure (in Pa)

Figure 15: Same as Figure 14, but for the training sample with highest combined error (#626).

Note that for the well-predicted sample #344, the prediction error is practically restricted to the surface of the car, whereas the bulk field – of both pressure and velocity – do not exhibit any significant difference. The poor reconstruction on the surface is probably due to an insufficient spatial resolution of the car boundary and does not have anything to do with the model order reduction of the fields themselves. In contrast, Figure 15 clearly exhibits deficiencies in the

prediction of sample #626 in the whole wake region: a chain of artificial pressure minima and maxima and unphysical structures in the velocity field. This is a direct consequence of the information loss introduced via basis truncation. However, even for this worst sample, the combined error of 0.94% is acceptably small and is hardly visible at all in the originally scaled images of pressure and velocity (top two rows of Figure 15).

Finally, Table 6 presents some performance measures for the ROM-based field predictions. The computational effort for both training and field prediction scales with the number of modes and is therefore higher for pressure (157 modes) than for velocity (88 modes). Still, the roughly 100s for training occur only once and are negligible in comparison to the total computational effort for the generation of the CFD training data. The time consumption of approximately 100ms for a single field prediction is small enough to allow for the intended interactivity of predicting fields for modified geometries.

Table 6: Summary of performance measures for field predictions. The wall clock time for training and prediction is reported as mean+/-standard deviation over 10 trials measured on a Linux workstation with 16 CPU-cores (2 × Intel® Xeon® CPU E5-2667 v4 @ 3.20 GHz)

Predicted fields	#Modes	#Predicted values	Training time [s]	Prediction time [ms]
Pressure	157	66560	104+/-10	106+/-4
Velocity	88	199680	75+/-4	44+/-14

4.2. Non-deterministic Method

Drag prediction

In this section, a numerical investigation on the effect of the different databases on the accuracy of the drag coefficient prediction is performed. The different databases are the OpenAGS, ClosedAGS, mixed-small and mixed-all sets that have been described in Section 3.2.2. In the numerical experiments that follow for all databases, a sampling spatial resolution of 256x256x1 was chosen, Adagrad was the optimizer selected and 5 convolution blocks were used for the NN. Here, for validation purposes, the training set of each database is used as the testing set.

For the sake of a valid comparison between both methods (to be reported in D2.3), we made sure that the input data is exactly the same for both approaches: the POD+I method takes the same snapshots as the NN, resampled in the same way to the same Cartesian mesh. The geometry representation is, however, not the binary mask of the NN, but the ANSA morphing parameters introduced above.

The error in the drag prediction and its standard deviation are shown in Table 7. It can be concluded that the most accurate models regarding drag prediction are the OpenAGS and ClosedAGS datasets, where the error in the prediction, remembering that a discretization of 600 classes was used, is 0%. Although this would support the initial argument of having a separate training process for the existence or not of Boolean features, the model is overfitting and further investigation is needed. The training using the mixed-all database results in much more accurate predictions (1 order of magnitude) compared to the mixed-small database. This

is reasonable in the sense that more samples are classified in each class, as the number of samples in this database is double the number of the rest databases. In Figure 16 the histogram of the drag coefficient error for the mixed-all database is shown. As can be seen, almost for 75% of the test dataset the accuracy in the drag coefficient prediction is around 99.8% and the error distribution, has in general, a monotonically decreasing function pattern (monotonic in subdomains, not in the whole domain). Besides, the wall clock time is similar for the OpenAGS, ClosedAGS and mixed-small databases, but it is almost double for the mixed-all database, as it consists of 2000 cases, whereas the rest databases consist of 1000. The simulations were performed in Cirrus HPC, by using one NVIDIA Tesla V-100-SXM2-16GB (Volta) GPU accelerator out of four which exist in each GPU node.

Table 7: Effect of the absence or presence of the Boolean feature on the drag prediction using 5 convolution layers and the adaGrad optimizer. The error, the standard deviation of the error, as well as the wall clock time are shown.

Database	C_D (error %)	std($C_{D,error}$)	Wall clock time (min)
OpenAGS	0.00	0.000	7.71
ClosedAGS	0.00	0.000	8.41
mixed-small	2.940	0.156	8.18
mixed-all	0.115	0.017	15.76

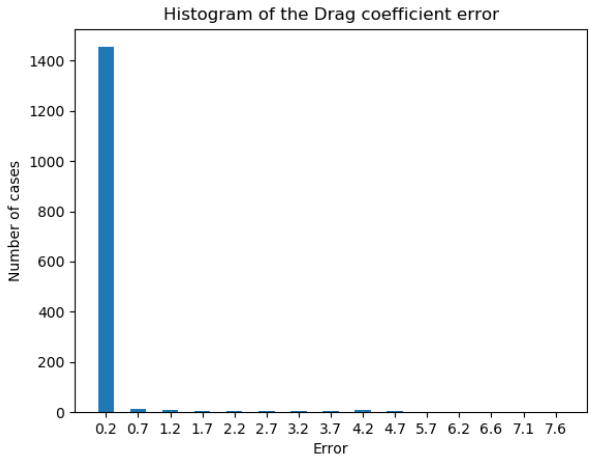


Figure 16: Histogram of the Drag coefficient error for the mixed-all database.

Flow prediction

The effect of the presence or absence of Boolean features is investigated next by using different databases. The different databases are the OpenAGS, ClosedAGS, mixed-small and mixed-all sets that have been described in Section 3.2.2. In the numerical experiments that follow for all databases, a sampling spatial resolution of 256x256x1 was chosen, Adam was the optimizer selected and 7 convolution layers were used for the NN. Here, for validation purposes, the training set of each database is used as the testing set.

As can be seen from Table 8, due to small error values it is unclear whether the training based on the ClosedAGS and OpenAGS databases result in a more accurate prediction of the flow field. Using more samples in the mixed-all database doesn't actually improve the accuracy which means that having a database of the order of 1000 samples is sufficient for AI predictions. However, the standard deviation of the error for the pressure and velocity magnitude predictions is significantly smaller in the databases with the Boolean feature on or off, as it is below 30% of the corresponding error value. This is not the case for the mixed databases, where the standard deviation can be around 50% of the error or even above. In Figure 17 the L_1 norm for all databases is plotted. The training models generated by the OpenAGS and ClosedAGS databases demonstrate better convergence compared to the mixed-small and DrivAer databases. Furthermore, it takes the half number of epochs for the mixed-all database to converge to the same residual as the other databases. However, the wall clock time of each epoch is double, compared to the other databases, as the number of the samples is double. The total number of the epochs in the algorithm is defined as the number of the iterations divided by the number of the batches, resulting in around 500 epochs for the mixed-all database and around 1000 epochs for the other databases. Therefore, the wall clock time is from 48 to 50 minutes for all databases using the computational resources mentioned in the above section. In Figure 18 the histograms of the pressure and velocity magnitude errors for the mixed-all database are shown. It is elucidated that for 60% of the dataset the accuracy in the prediction is 99.9% for both pressure and velocity magnitude and the distribution error is a monotonic function.

Table 8: Effect of Boolean features (different training databases) on the accuracy of the flow prediction variables. Error in the pressure and velocity magnitude prediction, as well as the standard deviation of these errors, are shown in the corresponding columns.

Database	**p** (error %)	**U** (error %)	std(p_{error})	std(U_{error})	Wall clock time (min)
OpenAGS	0.552	0.555	0.102	0.134	49.98
ClosedAGS	0.346	0.471	0.073	0.073	49.95
mixed-small AGS	0.523	0.551	0.295	0.221	49.01
mixed-all AGS	0.598	0.707	0.350	0.226	48.51

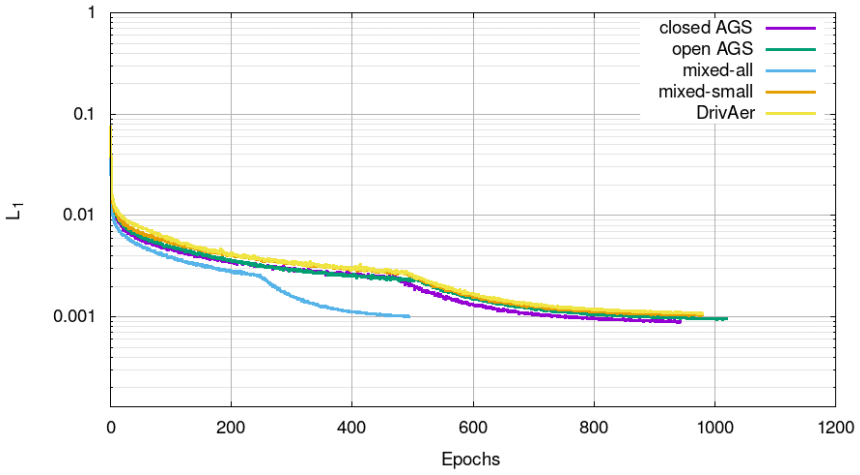


Figure 17: L_1 norms with respect to the number of epochs for the different datasets.

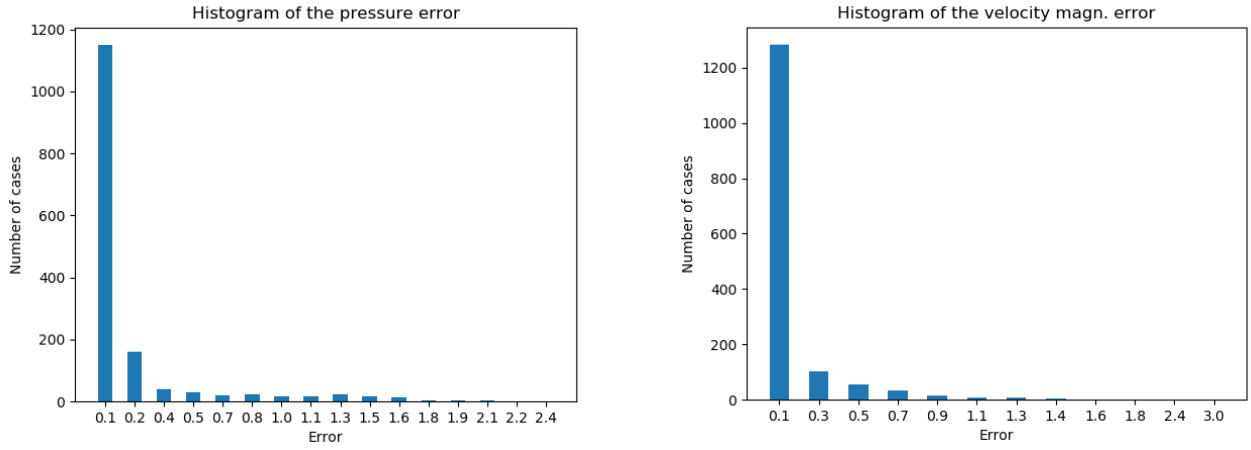


Figure 18: Histogram for the pressure (left) and the velocity magnitude (right) errors for the mixed-all database.

Once the prediction model is generated, which is the most time-consuming process, the flow field prediction of the requested cases is generated on the fly. The wall clock time for predicting the flow field of such a case, comparing it with the CFD sampled solution and exporting it to file is negligible and it was measured to be 1.25 sec when using a Tesla GPU accelerator in Cirrus and 0.51 sec when using a conventional CPU (Intel(R) Core(TM) i7-9700K CPU @ 3.60GHz). The wall clock time for the np2foam utility was measured to be approximately 3 seconds, for converting the predicted, target and error pressure and velocity vector fields to OF format, as it was measured in the above-mentioned CPU machine. It is obvious that the wall clock times measured in this section are negligible compared to the training time and can lead towards the development of a real time prediction tool for the aerodynamic coefficients and flow fields.

In an effort to visualize the predicted flow field and validate the developed post-processing tool (np2foam), which has been developed in HELYX CFD, indicative contour plots are demonstrated for some of the predicted cases.

The cases with the minimum and the maximum average error from the most accurate database were selected for visualization. The most accurate database was found to be the OpenAGS database by taking the average of the pressure and velocity errors. The average error is simply the sum of the pressure and the velocity error of each case, divided by 2. The case with the minimum average error value is number 562 with error 0.275% (Figure 19) and the one with the maximum average error is case 679 with error 0.892% (Figure 20).

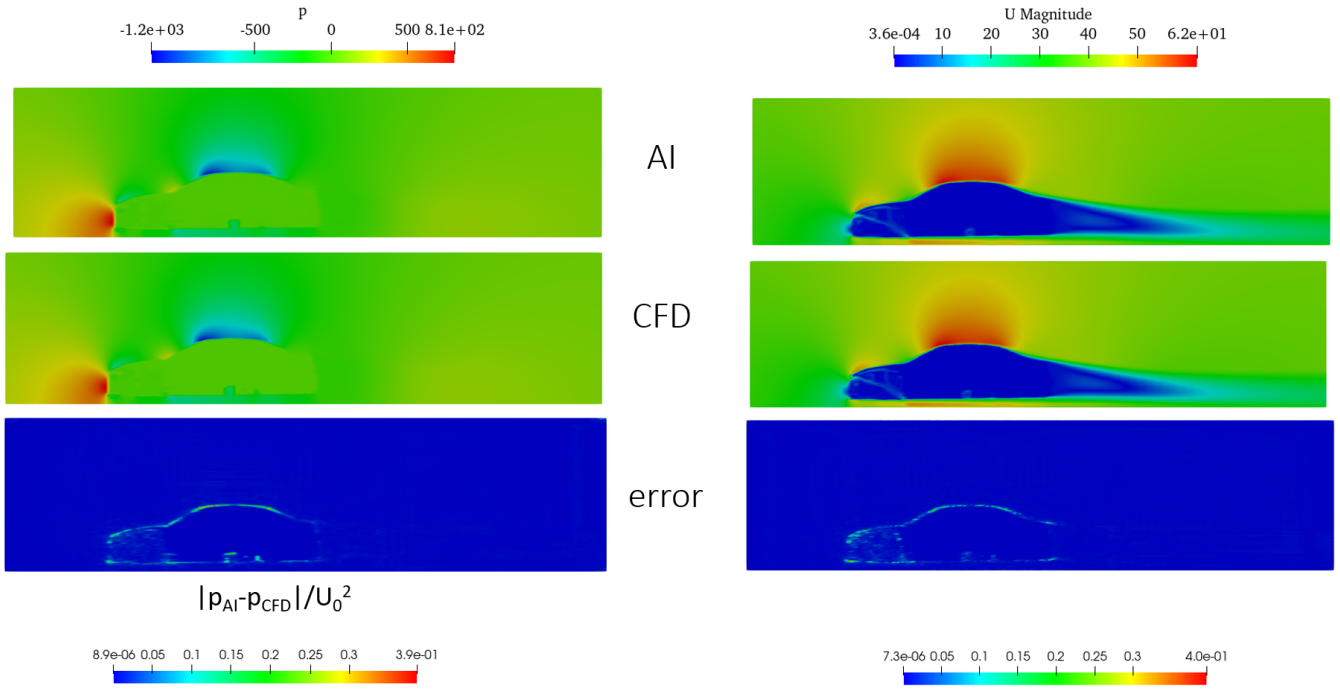


Figure 19: Pressure and velocity vector contour fields for case 562 of the ClosedAGS database. On the left side, the predicted (AI, 1st line) and the target (CFD, 2nd line) pressure fields are demonstrated first and then the error is shown (error, 3rd line). Similarly, on the right side, the predicted (AI, 1st line) and the target (CFD, 2nd line) velocity magnitude fields are demonstrated first and then the error is shown (error, 3rd line).

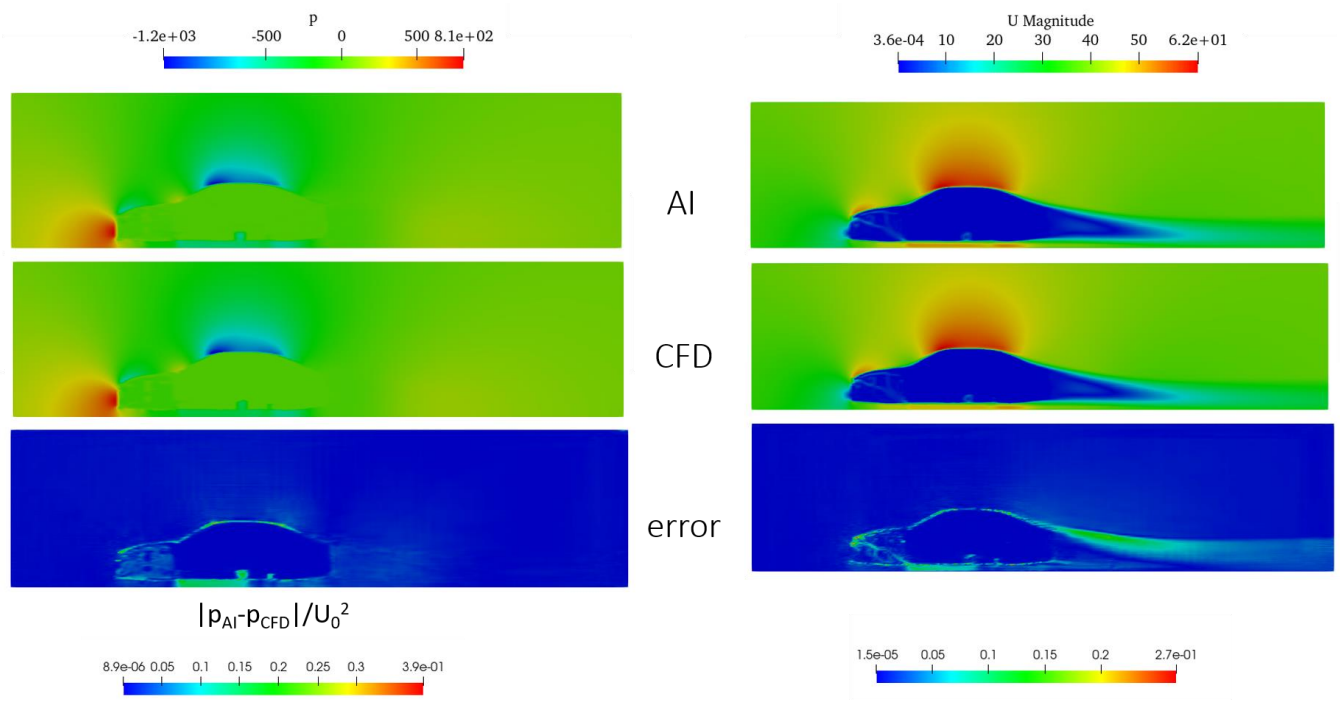


Figure 20: Pressure and velocity vector contour fields for case 679 of the closed AGS database. On the left side, the predicted (AI, 1st line) and the target (CFD, 2nd line) pressure fields are demonstrated first and then the error is shown (error, 3rd line). Similarly, on the right side, the predicted (AI, 1st line) and the target (CFD, 2nd line) velocity magnitude fields are demonstrated first and then the error is shown (error, 3rd line).

It can be concluded that the predicted and target flow fields look visually similar, if not the same and that the areas with the higher error values are at the solid-fluid interface, especially in the engine regime, where many geometrical details exist.

5. Conclusions and Future Work

The above results give us confidence to try predicting the drag coefficient and the flow fields in cases that have not been used for training (test set). It is unclear however, if separate training for the existence or not of each Boolean feature results in improved accuracy in the drag coefficient and the flow fields prediction. A decision tree training, depending on the existence or not of each feature can be followed. However, for n Boolean features, 2^n training processes have to be performed, increasing that way the computational cost, as well as the storage requirements for the different databases that have to be generated. It is unclear though if such a decision tree training affects the accuracy in the pressure and the velocity magnitude prediction. Therefore, cases that have not been used during the training will be used as test set is the next task in order to conclude if the OpenAGS and ClosedAGS databases provide more accurate predictions compared to the mixed ones.

From the pressure and velocity magnitude contour fields it can be concluded that although the error is generally low, its maximum values are usually close to the solid boundary. This is more evident when the boundary has many geometrical details, which cannot be captured with the existing sampling resolution. Similarly, unphysical artefacts are noticed in the sampled flow field and hence, even if the AI predicted solution is physically correct, the error is increased as the latter is compared to the sampled solution. In an attempt to reduce the existing error and to investigate its source, higher resolution sampling which will potentially diminish the aliasing error and will be developed in future work. Once the current methodology has been validated for 2D cases, the next step is to apply it to 3D cases and to predict the flow field in realistic vehicle geometries. The size of such datasets will unavoidably increase, especially if a decision tree approach is followed, where a different database and training for each non-linear feature has to be stored. Therefore, switching to HDF5 file format is indicated, as they ensure high-performance I/O and significantly smaller database size. Furthermore, Metadata related to Boolean features or different flow conditions, e.g. Reynolds number can also be stored in HDF5 file format.

ACKNOWLEDGEMENT

The author(s) would like to thank the partners in the project for their input, valuable comments on previous drafts and for performing the review.

Project partners:

PARTICIPANT Nº	PARTICIPANT ORGANISATION NAME	COUNTRY
1 (Coordinator)	IDIADA AUTOMOTIVE TECHNOLOGY SA (IDIADA),	Spain
2	VOLVO PERSONVAGNAR AB (Volvo Cars)	Sweden
3	VOLKSWAGEN AG (VW)	Germany
4	CENTRO RICERCHE FIAT SCPA (CRF)	Italy
5	ESI GROUP (ESI GROUP)	France
6	ENGYS LTD (ENGYS LTD)	United Kingdom
7	Kompetenzzentrum - Das Virtuelle Fahrzeug, Forschungsgesellschaft mbH (VIF)	Austria
8	VRIJE UNIVERSITEIT BRUSSEL (VUB)	Belgium
9	ECOLE NATIONALE SUPERIEURE D'ARTS ET METIERS (ENSAM PARISTECH)	France
10	ALGORITHMICA TECHNOLOGIES GMBH (ALGORITHMICA)	Germany
11	F INICIATIVAS I MAS D MAS I SLU (F-INICIATIVAS)	Spain

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement no. 824306

7. References

1. *Introduction of a New Realistic Generic Car Model for Aerodynamic Investigations.* A. Heft, T. Indinger, N. Adams. Detroit, Michigan, USA. s.n., April 23-26, 2012.
2. Systems, BETA CAE. ANSA version 18.0.x User's Guide. 2017.
3. *On the distribution of points in a cube and the approximate evaluation of integrals.* Sobol, I.M. 4. Elsevier BV, 1967, USSR Computational Mathematics and Mathematical Physics, Vol. 7, pp. 86-112. 0041-5553.
4. pyDOE. *https://pythonhosted.org/pyDOE/ .* [Online] 2020.
5. HELYX Open-source CFD for Enterprise. [Online] ENGYS, April 2020. https://engys.com/products/helyx.
6. *Nearest Neighbor Pattern Classification.* Cover, T. M. and Hart, P. E. 1, s.l. : Institute of Electrical and Electronics Engineers (IEEE), 1967, IEEE Transactions on Information Theory, Vol. 13, pp. 21-27. 10.1109/TIT.1967.1053964.
7. *An Introduction to Kernel and Nearest-Neighbor Nonparametric Regression.* Altman, N. S. 3. JSTOR, 1992, The American Statistician, Vol. 46, p. 175. 10.2307/2685209.
8. *SciPy 1.0: fundamental algorithms for scientific computing in Python.* Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, et al. 3. Nature Publishing Group US, 3 2020, Nature Methods, Vol. 17, pp. 261-272. 1907.10121.
9. Braconnier, T., Ferrier, M., Jouhaud, J.-C., Montagnac, M., Sagaut, P. Towards an Adaptive POD/SVD Surrogate Model for Aeronautic Design. *Computers & Fluids.* 1 40 2011, Vol. 40, 1, pp. 195-209.
10. Zimmermann, R., Görtz, S. Improved extrapolation of steady turbulent aerodynamics using a non-linear POD-based reduced order model. *The Aeronautical Journal.* 2012, Vol. 116, 1184.
11. *Towards Real-time Vehicle Aerodynamic Design via Multi-fidelity Data-driven Reduced Order Modeling.* Bertram, A., Othmer, C., Zimmermann, R. Florida, USA: AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, 2018.
12. Mrosek, M., Othmer, C., Radespiel, R. Reduced Order Modeling for Vehicle Aerodynamics via Proper Orthogonal Decomposition. *SAE Int. J. Passeng. Cars – Mech. Syst.* 12, 2019, Vol. 3.
13. *POD Based Reduced Order Model for CFD Optimization of Vehicle Aerodynamics.* Miretti, L., Ribaldone, E., Lorefice, L., Scantamburlo, G. Madrid: Eurogen, 2017.
14. Bellman, R.E. *Adaptive Control Processes.* Princeton, NJ: Princeton University Press, 1961.
15. *Deep Learning Methods for Reynolds-Averaged Navier–Stokes Simulations of Airfoil Flows.* Thuerey, N., Weißenow, K., Prantl, L., Hu, X. 1: American Institute of Aeronautics and Astronautics (AIAA), 2020, AIAA Journal, Vol. 58, pp. 25-36. 1810.08217.
16. Paszke, A., Gross, S., Massa, F., et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. [book auth.] H. Wallach and H. Larochelle and A. Beygelzimer and F. d\textquotesingle Alch\'{e}-Buc and E. Fox and R. Garnett. *Advances in Neural Information Processing Systems 32.* Curran Associates, Inc., 2019, pp. 8024--8035.
17. Taddy, M. Stochastic Gradient Descent. Business Data Science: Combining Machine Learning and Economics to Optimize, Automate, and Accelerate Business Decisions. s.l. : McGraw Hill Professional, 2019.
18. J. Duchi, E. Hazan, Y. Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. Journal of Machine Learning Research, 2011. Vol. 12.
19. Ronneberger, O., Fischer, P., Brox, T. U-net: Convolutional networks for biomedical image segmentation. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics).* s.l. : Springer International Publishing, 2015. Vol. 9351, pp. 234-241. 1505.04597.

20. Golub, G.H., van Loan, C.F. *Matrix Computations.* Baltimore, MD : Johns Hopkins Univ. Press, 2013. ISBN: 978-0801854149.
21. Pinnau, R. Model Reduction via Proper Orthogonal Decomposition. *Mathematic in Industry: Model Order Reduction: Theory, Research Aspects and Applications.* Berlin, Heidelberg: Springer, 2008.
22. Forrester, A., Sóbester, A., Keane, A. *Engineering Design via Surrogate Modelling: A Practical Guide.* Chichester: Wiley, 2008.
23. Santner, T. J., Williams, B. J., and Notz, W. I. *The Design and Analysis of Computer Experiments.* New York Berlin Heidelberg : Springer, 2003.