D8.1 Initial Pipeline Definition of all Use Cases

DAPHNE

Integrated Data Analysis Pipelines for Large-Scale Data Management, HPC, and Machine Learning

Version 2.1

PUBLIC

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 957407.

Document Description

This document gives an overview of the initial pipeline definitions for all use cases.

D8.1 Initial use case definition all use cases			
WP8 – Use Case Studies			
Type of document	R	Version	2.1
Dissemination level	PU		
Lead partner	KAI		
Author(s)	Benjamin Steinwender (KAI), Marius Birkenbach (KAI), Marlies Hofer (AVL), Daniel Krems (AVL), Andreas Laber (IFAT), Nikolai Skuppin (DLR)		
Reviewer(s)	Matthias Boehm (KNOW), Ahmed Hamdy Mohamed Eleliemy (UNIBAS)		

Revision History

Version	Revisions and Comments	Author / Reviewer
V1.0	Initial document structure, merged description of all individual use cases	Benjamin Steinwender
V2.0	Incorporated reviewer comments	Benjamin Steinwender
V2.1	Final polishing and merge of use case refinements	Matthias Boehm

Executive Summary

Five use case pipelines are presented in this report. The details given in the individual sections should enable the reader of this document to implement a similar pipeline for evaluation purposes.

Although the scientific background of the different partners varies greatly, the data processing techniques and focus are largely identical:

- The **data pre-processing** part is a quite substantial effort in the design and definition of the pipelines.
- Machine learning processes are mostly described using the **Python** scripting language and well-known toolkits like scikit-learn [1].
- All pipeline descriptions mention **runtime** as being the most crucial measurable.

Table of Contents

List of Abbreviations

Abbreviation	Meaning
CFD	Computational Fluid Dynamics
CORE	Custom Output Range Exploration
CSV	Comma Separated Values
CV	Characteristic Value
DoE	Design of Experiments
DUT	Device Under Test
ELMN	Evolving Local Model Network
ER	Entrainment Ratio
GDAL	Geospatial Data Abstraction Library
IODP	Integrated and Open Development Platform
OEE	Overall Equipment Efficiency
P	Pressure
ΔP	Pressure Difference (pressure loss in stack or suction pressure in ejector)
PEM	Proton Exchange Membrane
ROI	Region Of Interest
SOFC	Solid Oxide Fuel Cell
SSF	Solver Steering File
T	Temperature
TCV	Target Characteristic Value
TDMS	Technical Data Management System

1 Introduction

In order to demonstrate impact on real-world applications, to ground our research activities with relevant use cases, and to allow for benchmarking and quantifying the research progress, we selected four – with use case 4 being described as two case studies in this document – complementary use case studies. In the following, we summarize the use cases and describe the related data types.

A **pipeline** describes a process which progresses towards a specific goal that involves an arbitrary number of lined up stages. In the case of a data pipeline, data is passed on between the stages in a directed way. One stage represents a processing step. Every stage has a defined task and processes the input data with a resulting outcome. Potentially these processing elements are executed in parallel. Machine learning pipelines consist of different steps to prepare the data, train a model or perform inference.

The main **objectives** of the use case studies are twofold. First, the variety of use cases represent real-world applications to quantify the productivity and performance improvements of the developed system infrastructure and provide feedback on pain points to the system-oriented work packages. Second, we aim to improve the accuracy and runtime of these important applications at algorithmic level. As most use cases leverage machine learning, we also aim to explore the accuracy-runtime trade-off which has huge potential but can only be done in close collaboration with the domain-specific applications.

A variety of different **use cases** is presented in the following sections. From climate zone classification of aerial images over manufacturing equipment optimization and material degradation studies to fuel cell development topics and virtual prototypes for vehicle development, these use cases all implement data processing techniques and potentially machine learning pipelines. Although the selected use cases are very diverse from both application point of view and initial setup, the requirement of a pipeline-based data processing system is ubiquitous.

2 Earth Observation Case Study: Local Climate Zone Classification (DLR)

Urbanization is the second largest mega-trend right after climate change. Accurate measurements of urban morphological and demographic figures are at the core of many international endeavors to address issues of urbanization. Local climate zones (LCZs) classification is a land-use, land-cover classification scheme originally designed for studying urban heat islands. But it also showed its potential in urban morphology mapping. The LCZs classification scheme has 17 classes that describe the building density and height of an area, which are shown in the left of Figure 2.1. On the middle of Figure 2.1, it shows the LCZs classification of Vancouver, Canada, as an example. The right of Figure 2.1 shows the Google image of the area marked by the yellow rectangle in the middle figure, which are the densest area in Vancouver. In order to provide a global monitoring, large amounts of satellite images need to be processed, including analysis-ready data preprocessing, machine learning model training, and global image inferencing. Therefore, this is per definition a big data problem. Efficient computation is required.

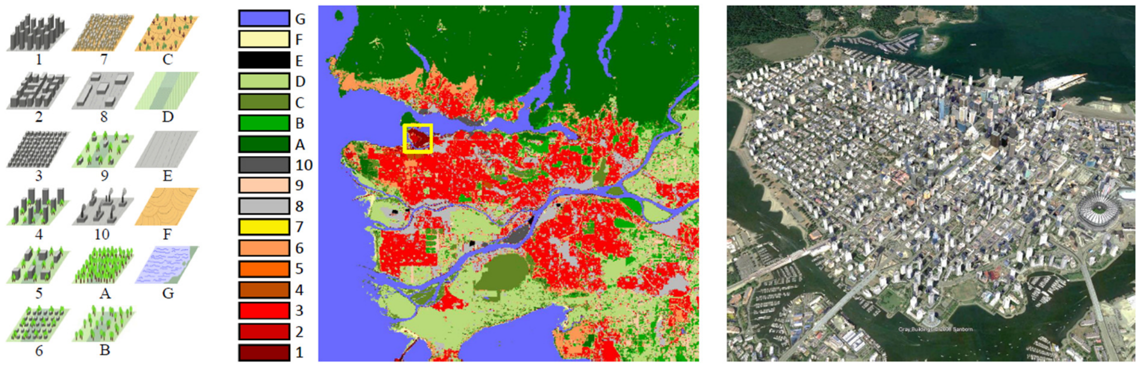


Figure 2.1: LCZ classes [2], example LCZ classification and a Google image of downtown area marked with the yellow rectangle. [3]

2.1 Overview of the Initial Pipeline

The processing from raw satellite images to the targeted LCZ maps can be divided into three stages:

2.1.1 Pre-processing

The target is to generate analysis ready data for the later stages. This consists of temporally and spatially merging of multiple satellite images of a given region of interest. Additionally, upsampling the different spatial bands to a target ground resolution (10m) and aggregating cloud free images. Currently, this process is implemented in Google Earth Engine (a cloud platform for processing Earth observation data).

The input is a Region Of Interest (ROI) and a time span. The region of interest is defined as a polygon in geographic coordinates (e.g., stored in a GeoJSON). The process will then merge all satellite images covering the ROI in the given time span to compose a cloud free image. For each image, a cloud and shadow probability mask are calculated and pixels are sorted by this value and pixels with least probability for cloud and shadow are selected. A detailed description of the pre-processing is given in [4]. The output is an analysis ready image of size A x B, with all thirteen Sentinel-2 bands (B1, **B2**, **B3**, **B4**, **B5**, **B6**, **B7**, **B8**, **B8A**, B9, B10, **B11**, **B12**) upsampled to 10 m, where only the bands marked in bold are later used for training and inference. The final image is stored to a GeoTIFF. Figure 2.2 illustrates the input and output dependencies of the pre-processing pipeline.

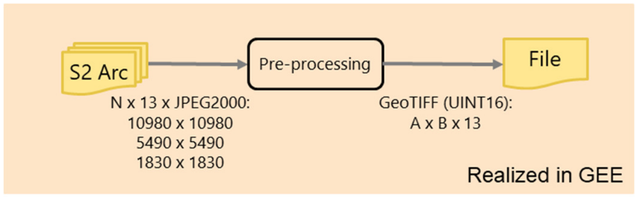


Figure 2.2: Pre-processing Pipeline

2.1.2 Training

The target is training a model for LCZ classification, using the available So2Sat-LCZ42 data set [3]. This consists of image patches of size 32 x 32 x 10 pixels labeled into one of the 17 LCZ classes and split into training, validation and test data. Please note that different versions of the data set have been published. We refer to version 2 to get the training, validation and testing split.

The output is a trained model (weights), which can be used for the inference of additional data. For the baseline, a ResNet20 model is used [5]. Figure 2.3 provides a high-level view on the input and output of the training process.

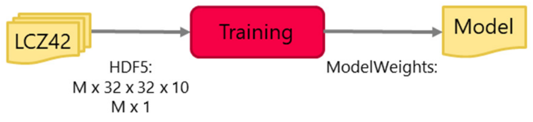


Figure 2.3: Training Pipeline

2.1.3 Inference

The target is to generate an LCZ map for a given region of interest. Inputs are the model obtained during training and the pre-processed images from the first step. The output is the LCZ map stored as an image. Figure 2.4 illustrates the different processing steps during inference.

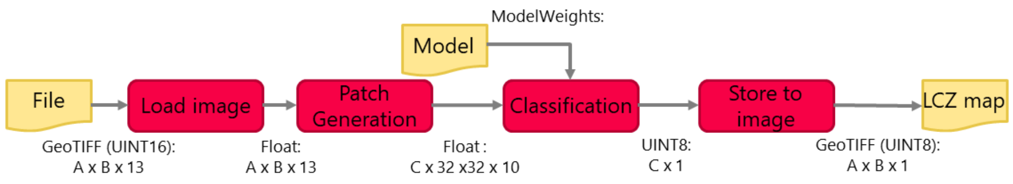


Figure 2.4: Inference Pipeline

2.2 Description

2.2.1 Pre-processing

Details of the pre-processing can be found in [4]. The results are stored in GeoTIFFs with the values coded in UINT16.

1.1.1 Training

A python script is used to train the model with the So2Sat-LCZ42 [3]. Uses version 2 of the data set, where *training.h5* is used for training and *validation.h5* for validation (*testing.h5* can be used for final testing and comparison of different models). The training data consists of image patches of 32 x 32 x 10 pixels (bands B2, B3, B4, B5, B6, B7, B8, B8A, B11 and B12). For the baseline a ResNet20 model is used [5]. The training uses a batch size of 16 and an initial learning rate of 2e-4. The learning rate is reduced by a factor of 5 when the validation loss did not improve for two epochs. The minimum learning rate is 1e-7 and the cool-down is three epochs. The maximum number of epochs is set to 100 and early stopping on the validation loss is used with a patience of 5 epochs. The optimizer is Adam with Nesterov momentum and default parameters (beta_1=0.9, beta_2=0.999, epsilon=1e-07). The loss is categorical cross entropy.

1.1.2 Inference

- **Load image**: A Python script to load bands B2, B3, B4, B5, B6, B7, B8, B8A, B11 and B12 from GeoTIFF and divide the values by 10,000 to obtain the reflectances. Uses GDAL to process the GeoTIFFs and returns a numpy array representing the image.
- **Patch generation**: A Python function to grid image into pixels of 100 x 100 square meters and cut patches of 32 x 32 x 10 pixels around each center coordinate from the original image (uses GDAL library). Uses symmetric padding of 16 pixels to capture border cases (cut patch at a border coordinate).
- **Classification**: A Python function to classify each 32 x 32 x 10 patch into one of the 17 LCZ classes. Uses model weights obtained during training to initialize the TensorFlow model. Classification of individual patches is independent from each other, which offers the possibility for parallel processing.
- **Store to image**: A Python function to store the LCZ map into a GeoTIFF. The pixel value at each image coordinate is set to the corresponding LCZ class, obtained in the previous step. Uses GDAL library to write GeoTIFF file.

2.2.2 Dependencies

In the following, the main dependencies are listed:

- Google Earth Engine (used for pre-processing)
- TensorFlow
- GDAL (used for processing GeoTIFFs)

2.3 Measurables

The most important measurable for this use case is the **runtime** for inference and training. Since the training time is expected to be negligible compared to inferencing, the primary concern shall be on the inference runtime. Both the training and inference are expected to benefit from improved and parallel processing of the image patches. There should also be no significant degradation of the **classification accuracy**. The pre-processing is highly dependent on the application and is run on a cloud platform with only high-level access, therefore, it is not the target for optimization.

Training on a single machine with an NVIDIA Tesla V100-SXM2 takes several hours and achieves a validation accuracy of about 61 %. Please note that these values are of stochastic nature and heavily depend on the underlying system architecture and settings. They only provide orientation for expected orders of magnitude.

3 Semiconductor Manufacturing Case Study: Optimizing the Equipment Stability and Utilization (IFAT)

There are many stages within semiconductor manufacturing, one of which is ion implantation. The objective of ion implantation is changing the physical, chemical or electrical properties of the target. The target in case of semiconductor manufacturing is in the majority of the cases a silicon wafer, which acts as a substrate for further processing. During ion implantation charged dopants are accelerated in an electric field and directed onto wafers. Special equipment is built for this processing step. A schematic of classical 'medium current' ion implantation equipment is shown in Figure 3.1.

The doping gas, e.g., boron trifluoride BF3, is injected in an ion source wherein a plasma is generated from this doping gas. The ions are then extracted from the surface plasma of the ion source with up to 80 kilovolts (kV). The charged particles are deflected by 90° by the magnetic field of the mass separator magnet. Particles that are too light/heavy are deflected more/less than the desired ions and intercepted by resolving apertures behind the mass separator. The ions are then accelerated to their final energy, 200 kiloelectron volt (keV) accelerate boron ions to approx. 2e-6 m/s. Lenses are distributed throughout the system to focus the ion beam. Mechanical wafer handling and electrostatic scanner plates can direct the ions to different positions on the wafer. Typically, the goal is to have a uniform dose from 1e-10 to 1e-17 at/cm^2 throughout the whole wafer.

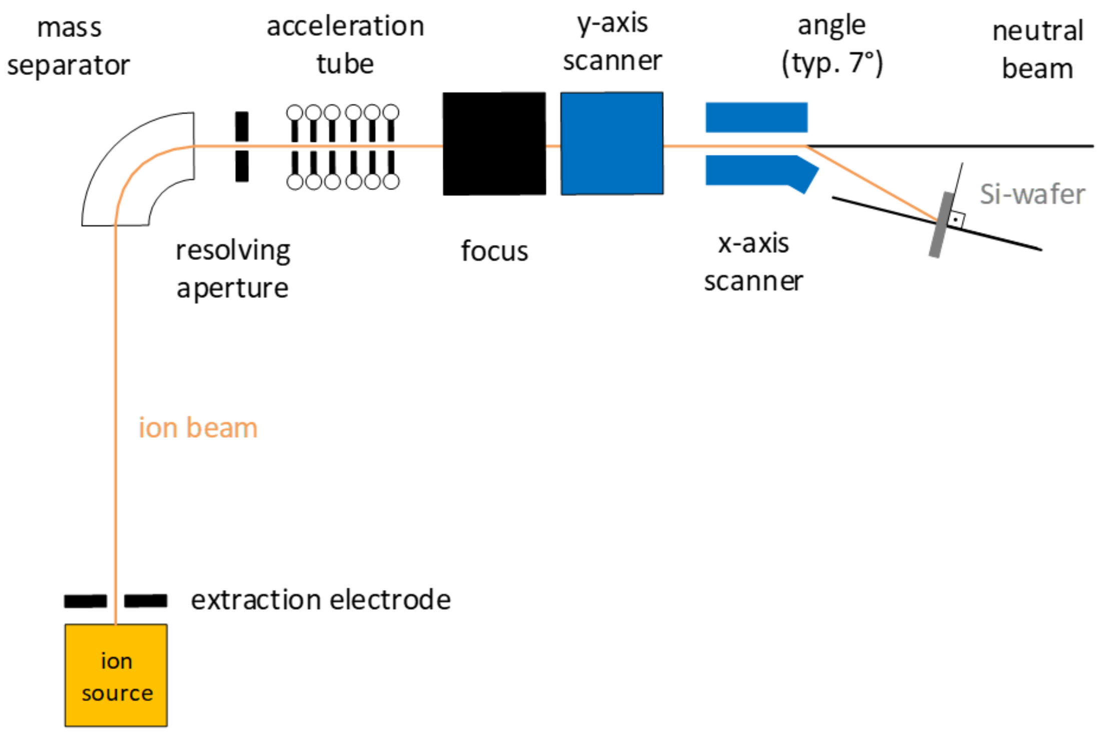


Figure 3.1: Schematic of Implantation Equipment

A recipe in the context of manufacturing states various parameter specifications to fulfill the process step's requirements. With every recipe change a setup is required, which includes ion beam tuning. Tuning ensures that recipe's specifications are met under varying equipment conditions. By manipulating the voltages and currents of specific components, the beam can be tuned to fit the specifications in the recipe. This usually is an automatic process, which is performed by the equipment on its own. If the tuning parameters only differ slightly from the previous ones, tuning can be rather quick. To name a bad scenario, if the specifications differ a lot and the ion source is near the end of its lifetime, it may take roughly 15 minutes, until the equipment stops without being able to tune.

The objective is to predict which recipes are tunable with high probability and which recipes are not, given the equipment's current state. A tuning fail decreases the overall equipment efficiency (OEE) and generates the need to schedule a lot, i.e., a batch of 25 wafers, with a different recipe. Not tunable recipes are blocked in order to avoid retries. These restrictions are lifted after performing a scheduled maintenance activity. After e.g., replacing the ion source, a wider variety of recipes are readily processable again. In order to reduce downtimes and increase throughput, it is key to predict tuning success before a decision is made on which lot to process next.

3.1 Overview of the Initial Pipeline

The use case in its current state, is a combination of batch and stream processing. It is a rather classical machine learning pipeline, where the model is trained on a batch of historical data and the model is queried via a stream of requests. The complete use case consists of data engineering, depicted in Figure 3.2, and the machine learning pipeline, depicted in Figure 3.3. The data engineering pipeline is a prototype for exploratory analyses. It will be reimplemented in a more stringent and production ready manner. The machine learning pipeline is implemented in Python and makes intensive use of the scikit-learn library [1]. The resulting model is deployed within a container orchestration platform and delivers predictions, whenever a new lot arrives at the implantation equipment. The lot is therefore listed in the corresponding dispatch list. The dispatch sequence is then optimized by considering the predicted probability of tuning success.

3.2 Data Engineering

Within data engineering, the setup and implant log files are scanned and their data is parsed and stored in a MySQL database. Another Python script queries this data from five different tables and manipulates it to prepare it for the machine learning pipeline. The resulting table is stored as a csv file, to make it available for further processing. The size variable A of the csv depends on the considered timeframe for model training. The extracted data consists of a set of 79 categorical and a set of 2468 numerical columns, which mainly represent process targets and sensor readings. Each row represents a finished setup with additional data, which is available before tuning is initiated. The setup result is used as a label for the supervised training.

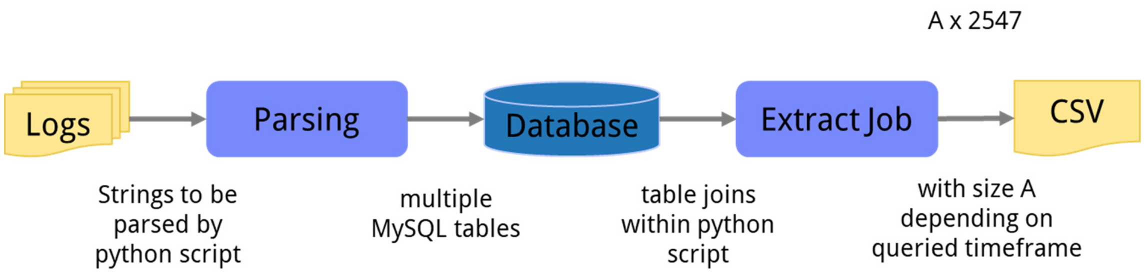

A x 2547
Logs
Parsing
Database
Extract Job
CSV
Strings to be parsed by python script
multiple MySQL tables
table joins within python script
with size A depending on queried timeframe

Figure 3.2: Data Engineering

A redesign decision could be to also include the "Extract Job" from Figure 3.2 into the machine learning pipeline and therefore eliminate the buffer csv file. This in turn would induce additional load on the database and a time delay for querying, while training a new model.

3.3 Machine Learning Pipeline

Given that data is loaded from the csv file to a pandas DataFrame, the existing machine learning pipeline at the time of writing manipulates the data in a few steps, which are elaborated on in the corresponding sections. An overview of the pipeline is shown in Figure 3.3.

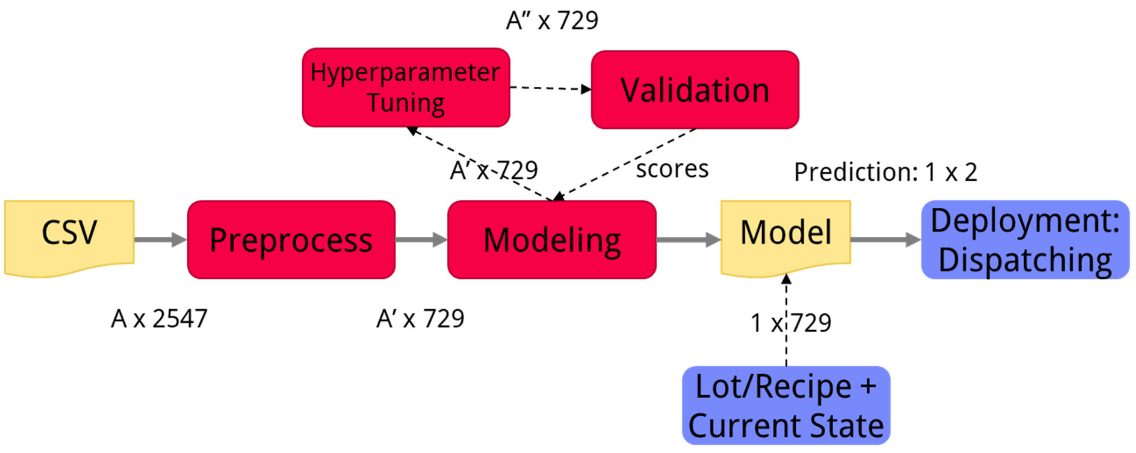


Figure 3.3: Machine Learning Pipeline

3.3.1 CSV

The dataset consists of 2547 columns. The content varies from columns containing recipe or equipment names as strings, over columns representing timestamps of e.g., the start and end of the setup process, to the majority of columns of datatype float, which represent sensor data. These sensor readings include, but are not limited to, gauge and vacuum pressures and various required settings and read-back values of components. One special column contains the label for ion beam tuning success or fail, in binary format.

3.3.2 Pre-processing

The data is split into a train (A') and test (A'') set by train_test_split(), resulting in a subset of observations for further processing. In regard to feature selection columns with static values or low variance are removed by sklearn's VarianceThreshold(). The model can also be improved by removing highly correlating features from the dataset. This is achieved by X.corr() and appropriate masking. The data is imputed by forward-filling missing entries with pandas X.fillna(). To fill up missing entries after forward-filling, SimpleImputer() replaces these values with the mean of the respective column of A′. The recipe column is one-hot encoded to provide processable features for classifiers and therefore extends the number of columns by the number of distinct recipes. After this selection process only 729 columns remain for the currently used evaluation dataset. With StandardScaler() data is scaled and centered on A' as not to leak information from A′′. A domain expert performed a one-time manual selection of the most promising features. This resulted in the selection of a subset of features for one specific equipment type. The goal is to let the model decide on its own. This allows for a more cost-efficient rollout to similar equipment.

3.3.3 Modeling

Random forest classifiers [6] are known to be well-performing and already produced promising results during the initial experiments for this use case. The classifier is created by sklearn's RandomForestClassifer(). On the evaluation dataset the following parameters proofed to perform well, max_depth=65 and n_estimators=450. These values were found iteratively by using RandomizedSearchCV() and GridSearchCV(). The output of the model on the training data are two percentage values stating the probability of tuning fail and tuning success respectively.

3.3.4 Validation

In most cases (> 80 %) tuning is successful. Therefore, accuracy is not a good validation method. With cross_validate() the F1 and ROC AUC scores are calculated. Moreover, the confusion matrix and the Matthews Correlation Coefficient (MCC) [7] are computed and in turn analyzed to monitor the validity of the model.

3.3.5 Deployment

During deployment the current equipment status is known, i.e., all feature columns are up to date within seconds. In rare cases, there is a possible delay of up to five minutes. The model is queried with live data from production. The dispatch list contains a list of lots, which are scheduled for process on implantation equipment. Whenever a new lot arrives at the equipment it is added to the list. For each lot, the corresponding recipe is known. For each recipe, a prediction is provided by the trained model in the form of probability percentages. According to the success probability and other production criteria, the lots are newly ranked within the dispatch list and the top one is dispatched. This decision is made lot by lot. Examples for additional ranking criteria are accumulated wait time, remaining time to due date, prioritization for development and customer demanded lots, lots needed for equipment checks and many more. With correct tuning predictions the OEE of the production equipment can be optimized, as it stays productive longer before the next maintenance activity is required and less time is spent on performing setups.

3.4 Measurables

Improvement by DAPHNE is expected to be in terms of computation time. This would allow for a more extensive search of the solution space, especially in the area of hyperparameter tuning while retraining. In turn, this should result in a positive impact on validation measures, F1, ROC AUC and MCC. The required retraining rate is not yet known, but it could be on a daily basis.

4 Material Degradation Case Study (KAI)

At KAI, we develop semiconductor test systems and methods for investigating degradation of power semiconductors. Thereby, large datasets are collected. A subset of selected data is shared within the consortium.

As a preparation step, we aim to streamline existing analysis pipelines for material degradation by automating data import/export, data preparation, and orchestration of analysis tools (DB, HPC FEM, analysis of simulation outputs) with a major focus on minimizing manual effort. Subsequently, we explore pipeline extensions by leveraging ML models and HPC for more accurate prediction of material degradation. Additionally, we aim to investigate temporal changes over waveform measurements and succinct representations of waveforms characteristics and their key parameters. Finally, we will explore accuracy-runtime tradeoffs with different data subsets, data representations, and analysis methods for different scenarios from offline analysis to device monitoring.

Our raw waveform data consists of two signals, the voltage across (V_{DS}) and the current through the DUT (I_D) during a test pulse. The pulses are recorded through our test systems. There is a series of pulses for every tested device.

4.1 Overview of the Initial Pipeline

The initial pipeline is composed of a data reduction and a subsequent simulation step. The input data consists of sampled waveforms which are stored in files. These raw waveforms require downsampling to enable performant storage in a database. Afterwards, several values from the database, including the decimated waveforms, serve as input parameters for the simulation step. The simulation results are stored in the same database.

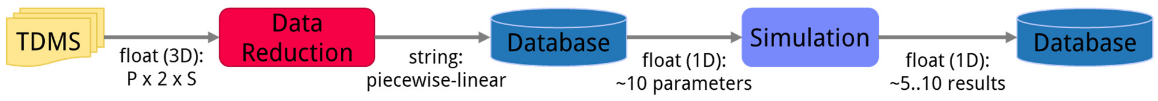


Figure 4.1: KAI pipeline overview.

As described in Figure 4.1, the initial data analysis pipeline comprises *P* number of pulses with 2 channels each stored in a TDMS file, *S* number of sample points per pulse and *R* number of sample points after data reduction.

- The waveform data is stored in the NI TDMS file format [8]
    - Binary structured file format (lightweight version similar to HDF5)
    - Our dataset shared with the consortium consists of ~13,760 TDMS files
    - One file contains the test pulses of one DUT (varying amount, typically ~3000)
- Data Reduction
    - A polyline simplification algorithm compresses the waveform data [9], [10]
    - The goal is to reduce the massive amount of waveform data without losing too much information
- Our database is an in-house developed web application based on PHP and MySQL
- Simulation

- Our physics-based simulations are carried out using ANSYS APDL. ANSYS allows the integration of black-box models and algorithms in larger workflows. In future, we might actually see if we can improve key components of the simulation too.
- Text files serve as input / output

4.2 Data Reduction

The focus lies on the data reduction component. It contains a sub-pipeline itself, which is elaborated in Figure 4.2. There are *P* number of pulses stored in a TDMS file, *S* number of sample points per pulse and *R* number of sample points after data reduction.

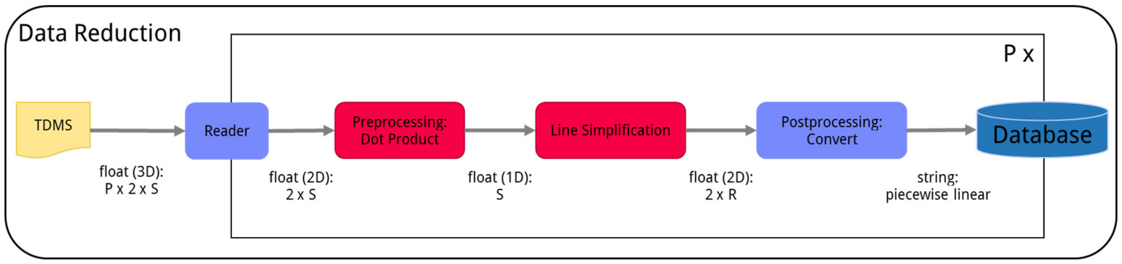


Figure 4.2: Data reduction pipeline.

4.2.1 TDMS File (Waveform Data)

The TDMS file format consists of a hierarchical structure, namely groups and channels. One file contains an arbitrary number of groups. A group consists of an arbitrary number of channels.

At KAI, one file contains all waveforms of one specific DUT. The waveforms are acquired sequentially during a degradation test. One group represents one test pulse (about 3000 pulses are recorded per test). One test pulse consists of two time-series data channels, the drain-source voltage V_{DS} across the DUT and the drain current I_D. One pulse is sampled with around 750 sample points.

- TDMS file (One DUT)
  - Group 1: Recorded Test pulse #1
    - Channel 1: drain-source voltage (V_{DS})
    - Channel 2: drain current (I_D)
  - Group 2: Recorded Test pulse #2
  - …
  - Group N: Recorded Test pulse #N

Mapped as a matrix: [~3000 pulses] X [V_{DS}, I_D] X [~750 sample points]

4.2.2 Reader

To read data from the TDMS files, various tools can be used, e.g., NI LabVIEW, or the Python package *npTDMS* [11].

The reader passes the waveforms (V_{DS} and I_D) group-wise to the preprocessing component. Currently, the reader is implemented in Python 3.

Input: [~3000 pulses] X [V_{DS}, I_D] X [768 sample points]

Output: [V_{DS}, I_D] X [768 sample points]

4.2.3 Pre-processing: Dot Product

The total power dissipation is the decisive factor for the thermo-mechanical simulation. Therefore, the voltage and the current are multiplied.

$$P_{tot} = V_{DS} \cdot I_D$$

The dot product of the voltage vector and the current vector is calculated.

$$\begin{pmatrix} V_{DS,1} \\ \vdots \\ V_{DS,S} \end{pmatrix} \cdot \begin{pmatrix} I_{D,1} \\ \vdots \\ I_{D,S} \end{pmatrix} = \begin{pmatrix} P_{tot,1} \\ \vdots \\ P_{tot,S} \end{pmatrix}$$

The resulting power waveform is forwarded to the line simplification algorithm.

Input: [V_{DS}, I_D] X [~750 sample points]

Output: [P_{tot}] X [~750 sample points]

4.2.4 Line Simplification

This component is the actual lossy data compression part. So far, we discovered two suitable algorithms, the Visvalingam-Whyatt algorithm (VW) [10] and the Douglas-Peucker algorithm (DP) [9]. At the moment, the former is used, but the latter still is an option.

Visvalingam-Whyatt is an iterative algorithm which step by step discards points of the line with least information loss. The point with the smallest area of the triangle between itself and its neighboring points is discarded next. Since we feed a power curve to the algorithm, it minimizes the energy deviation during downsampling.

The algorithm has the following parameters:

1. *max_points*
    - Maximum number of points after simplification
    - Set to 18 (it has been shown that our data can be reduced very well to this amount)
2. *min_points*
    - Minimum number of points after simplification
    - Set to 2 (a line needs at least two points)
    - Latest abort criterion
3. *tolerance*
    - Upper barrier for the area of the triangle between neighboring points when discarding

If the *tolerance* already is exceeded while the number of points still is higher than the limit *max_points*, the algorithm continues discarding until the condition is reached.

If the limit *max_points* has already been undershot while the *tolerance* still is bigger than some triangles, the algorithm will continue until the *tolerance* is exceeded.

The algorithm terminates at the latest when *min_points* is reached.

Input: [P_{tot}] X [~750 sample points]

Output: [t_i, P_{tot}] X [~18 sample points]

4.2.5 Post-processing: Convert

The reduced waveform needs to be in the piecewise linear format. Linear regression is used to connect these points and form the line segments.

The result is stored in the database as a string. The post processing step therefore converts the reduced line into a list of numbers separated by space in string format: "t_0 $P_{tot,0}$ t_1 $P_{tot,1}$ … t_R $P_{tot,R}$".

4.2.6 Database

To access the database, various REST endpoints can be queried. The proprietary client is implemented with Python and can be imported as a package. The client accesses the server via a REST API. Basically, the database client sends 2D csv files via POST request to insert data.

4.3 Measurables

Within the DAPHNE project, we identified the following measurables for our initial pipeline:

- Runtime
- Memory requirements
- Compression-information-loss-ratio

The above mentioned measurables are evaluated for single pipeline components or jointly for successive components. For instance, the pre-processing step together with the line simplification step of the data reduction sub-pipeline. Parts of the pipeline whose implementation is addressed by the DAPHNE project are of special interest.

For our future and more developed pipeline, we can further evaluate with the number of detected features and the prediction accuracy for various sub-use cases.

5 Automotive Vehicle Development Case Study: Ejector Geometry Optimization (AVL 1)

AVL is developing gas ejectors for PEM and SOFC fuel cell systems. The ejector should compensate pressure losses of the stack and other components in the anode recirculation line (see Figure 5.1). The primary domain of the ejector is the fuel supply. A certain hydrogen (H2) flow rate is supplied from the fuel tank through an injector at a specific temperature and pressure and gets accelerated via the nozzle. Downstream the nozzle, in the mixing chamber of the ejector, the primary stream and the recirculated secondary stream are mixed. The secondary stream is accelerated by the high flow momentum of the primary stream. In the diffuser of the ejector, the mixed flow is decelerated while increasing the pressure to p2. The outlet gas of the ejector is led to the anode side of the fuel cell stack where a pressure loss occurs (Δp=p2-p3). The stack off-gas is led into the secondary domain of the ejector (=recirculation line).

Depending on the geometric variables of the ejector and the operating conditions like the supply pressure of the fuel, a certain entrainment ratio ER (ratio of recirculated flow and fresh fuel flow) and suction pressure can be achieved. The optimized ejector generates high suction pressures / recirculation flow rates.

AVL established a DOE workflow combining AVL software components (AVL FIRE™for Multiphysics CFD and AVL CAMEO™for parameter behavior model and optimizer) and elaborated a continuously growing data set which is getting denser for standard fuel cell system applications.

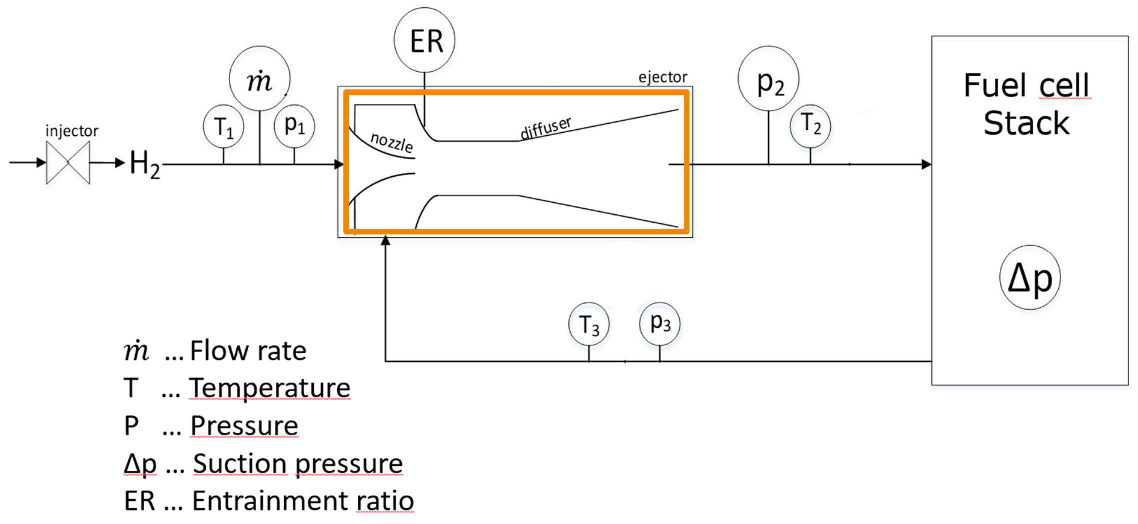


Figure 5.1: Schematic of anode recirculation line containing ejector (marked in orange) and the fuel cell stack. Aim of the ejector is to recover the pressure loss of the stack.

5.1 Overview of the Initial Pipeline

The current ejector geometry optimization pipeline consists of an initial lookup in the data set and further looping of the DoE (prediction of geometry and CFD verification) till target is achieved. The biggest and most complex elements are for sure the numerically expensive CFD simulation and the DoE model using machine learning techniques. Besides them, there are various scripts for data and simulation preprocessing and result post-processing, the outcome is collected in the data set that trains the behavior model with every new simulation.

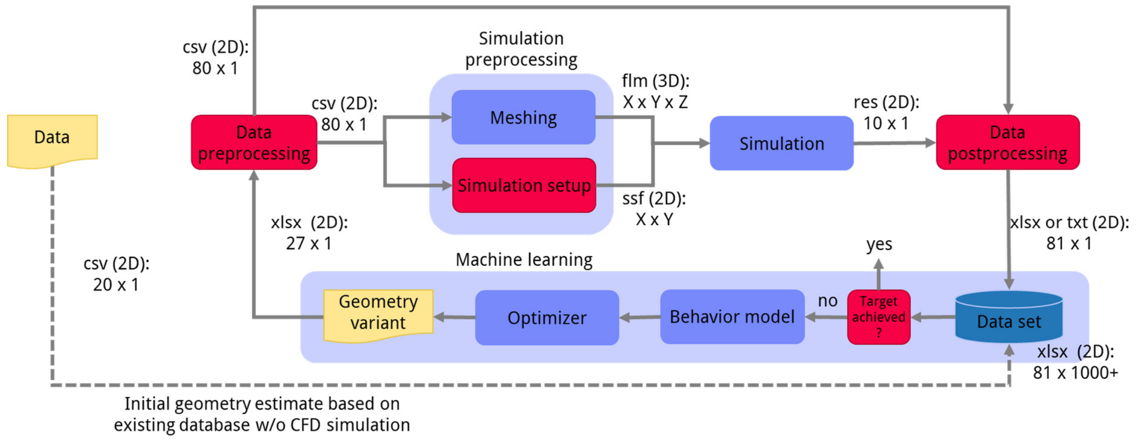


Figure 5.2: Initial Pipeline

5.2 Getting Started

5.2.1 Input Data

The goal of the task ejector geometry optimization is to find the most suitable combination of geometry parameters to meet a certain entrainment ratio and suction pressure for a specific operation condition. This operation condition is provided by the customer and consists at least of following information:

- Mass flow, temperature, gas composition and supply pressure of the fuel supply which is the ejector inlet primary side (Nr.1 in Figure 5.1)
- Mass flow, temperature and gas composition of secondary inlet side (Nr.3 in Figure 5.1)
- Pressure at stack inlet (=ejector outlet – Nr.2 in Figure 5.1)
- Target suction pressure
- Packaging constraints

This data mentioned above is collected and manually brought to a specific format with physical units which is the input **data**.

5.2.2 Initial Main Geometry Parameter Estimation

Before starting the first CFD simulation to get a result for the achievable suction pressure at the operation condition, the main design parameters of the ejector need to be defined. To receive the missing initial geometry parameters of the ejector, the first step is to look into the **already existing data set**, enter the operation conditions and constraints and run the optimizer, which is based on the **already existing generic behavior model** by maximizing the suction pressure. The optimizer provides a first geometry proposal consisting of 7 key design parameters (geometry variant) and a corresponding suction pressure expectation. Together with the operation condition parameters, the data is forwarded to the data pre-processing.

5.3 Pre-processing

5.3.1 Data Pre-processing

With the operation condition data, and the provided first suggestion of ejector design main parameters, all necessary parameters such as mole fractions, densities and side design parameters of the ejector are calculated with a Python script. The result is a csv file used as input for in meshing and simulation scripts.

5.3.2 CFD Simulation Pre-processing

Now, since all necessary parameters are defined and collected in one row of a CSV file, the **CFD simulation pre-processing** is started with a Python script. While the numerical mesh for the CFD simulation is created, mainly based on the main and side design parameters, the simulation setup file is fed by the operation condition parameters. The geometric variables of the nozzle are calculated using physical correlations of isentropic nozzle flow. When the CFD simulation pre-processing is completed, a folder with the computational mesh in FLM (AVL FIRE™ geometry file) format, simulation setup with all data and parameters to perform the numerical simulation in SSF (Solver Steering File, an AVL FIRE™ input file) format and a file which couples both in DAT format is available. The CFD simulation can be started executing a CSH (C Shell) script (copying several scripts for result post-processing and a run script from a template directory to the calculation directory).

5.4 CFD Simulation

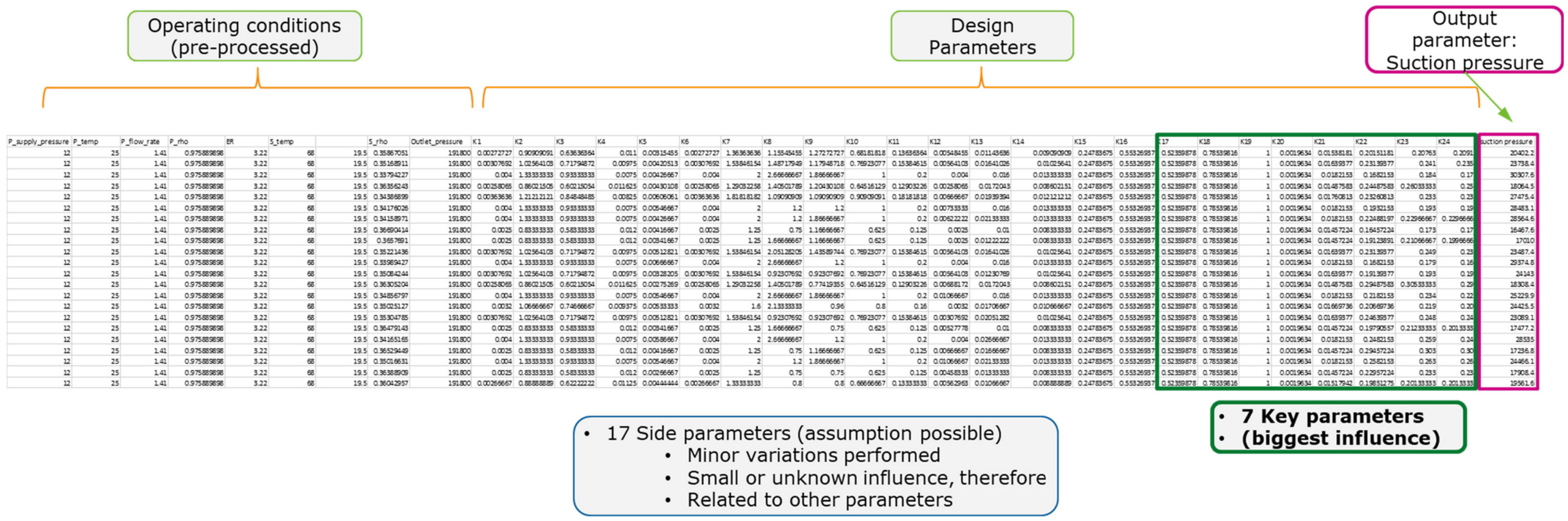
The CFD simulation is an important part to confirm the suction pressure prediction from the optimizer using physical relations. Compared to testing the ejector geometry on a test bed, a CFD simulation is much cheaper. The CFD **simulation** of the geometry variant is solving non-linear partial differential equations (Navier Stokes and Euler) for pressure, velocity and enthalpy in each cell of the ejector mesh. A high flow momentum can be detected in the nozzle cross section. The transient simulation takes around 14 hours to reach a stationary converged condition. CFD simulation software is an AVL in-house program called AVL FIRE™. The CFD simulation is an important part to confirm the suction pressure prediction.

5.5 Post-processing

After the simulation is finished, a **post-processing** script gathers the suction pressure (pressure at the outlet ejector – pressure at the secondary inlet of ejector) out of the simulation raw result. This information is combined with the geometric and the operation condition input parameters from data pre-processing (arrow from data preprocessing to data post-processing in Figure 5.2). After that, the generated data row (input and result) is added to the existing data set.

5.6 Data Set

The **data set** (see an extract in Figure 5.3) consists of 1000+ data rows in one text file. Each data row is characterized by the operating condition (i.e., pressure, temperature, gas composition, mass flow), 24 design parameters - with 7 key parameters (where we experienced the biggest influence on suction pressure) and 17 side parameters (where we experienced less influence on suction pressure and which are constant or derived from correlation to other parameters) - and the resulting suction pressure (Δp). Each row is a different geometry variant with all input variables and the resulting suction pressure.


Operating conditions (pre-processed)
Design Parameters
Output parameter: Suction pressure
• 17 Side parameters (assumption possible)
• Minor variations performed
• Small or unknown influence, therefore
• Related to other parameters
• 7 Key parameters
• (biggest influence)

P_supply_pressure	P_temp	P_flow_rate	P_rho	ER	S_temp		S_rho	Outlet_pressure	K1	K2	K3	K4	K5	K6	K7	K8
12	25	1.41	0.975889898	3.22	68	19.5	0.35867051	191800	0.00272727	0.90909091	0.63636364	0.011	0.00315455	0.00272727	1.36363636	1.15545455
12	25	1.41	0.975889898	3.22	68	19.5	0.35168911	191800	0.00307692	1.02564103	0.71794872	0.00975	0.00420513	0.00307692	1.53846154	1.48717949
12	25	1.41	0.975889898	3.22	68	19.5	0.33794227	191800	0.004	1.33333333	0.93333333	0.0075	0.00426667	0.004	2	2.66666667
12	25	1.41	0.975889898	3.22	68	19.5	0.36356243	191800	0.00258065	0.86021505	0.60215054	0.011625	0.00430108	0.00258065	1.29032258	1.40501789
12	25	1.41	0.975889898	3.22	68	19.5	0.34386899	191800	0.00363636	1.21212121	0.84848485	0.00825	0.00606061	0.00363636	1.81818182	1.09090909
12	25	1.41	0.975889898	3.22	68	19.5	0.34176026	191800	0.004	1.33333333	0.93333333	0.0075	0.00546667	0.004	2	1.2
12	25	1.41	0.975889898	3.22	68	19.5	0.34158971	191800	0.004	1.33333333	0.93333333	0.0075	0.00426667	0.004	2	1.2
12	25	1.41	0.975889898	3.22	68	19.5	0.36690414	191800	0.0025	0.83333333	0.58333333	0.012	0.00416667	0.0025	1.25	0.75
12	25	1.41	0.975889898	3.22	68	19.5	0.3657691	191800	0.0025	0.83333333	0.58333333	0.012	0.00341667	0.0025	1.25	1.66666667
12	25	1.41	0.975889898	3.22	68	19.5	0.35221436	191800	0.00307692	1.02564103	0.71794872	0.00975	0.00512821	0.00307692	1.53846154	2.05128205
12	25	1.41	0.975889898	3.22	68	19.5	0.33989427	191800	0.004	1.33333333	0.93333333	0.0075	0.00666667	0.004	2	2.66666667
12	25	1.41	0.975889898	3.22	68	19.5	0.35084244	191800	0.00307692	1.02564103	0.71794872	0.00975	0.00328205	0.00307692	1.53846154	0.92307692
12	25	1.41	0.975889898	3.22	68	19.5	0.36305204	191800	0.00258065	0.86021505	0.60215054	0.011625	0.00275269	0.00258065	1.29032258	1.40501789
12	25	1.41	0.975889898	3.22	68	19.5	0.34856797	191800	0.004	1.33333333	0.93333333	0.0075	0.00546667	0.004	2	2.66666667
12	25	1.41	0.975889898	3.22	68	19.5	0.35025127	191800	0.0032	1.06666667	0.74666667	0.009375	0.00533333	0.0032	1.6	2.13333333
12	25	1.41	0.975889898	3.22	68	19.5	0.35304785	191800	0.00307692	1.02564103	0.71794872	0.00975	0.00512821	0.00307692	1.53846154	0.92307692
12	25	1.41	0.975889898	3.22	68	19.5	0.36479143	191800	0.0025	0.83333333	0.58333333	0.012	0.00341667	0.0025	1.25	1.66666667
12	25	1.41	0.975889898	3.22	68	19.5	0.34165165	191800	0.004	1.33333333	0.93333333	0.0075	0.00586667	0.004	2	2.66666667
12	25	1.41	0.975889898	3.22	68	19.5	0.36529449	191800	0.0025	0.83333333	0.58333333	0.012	0.00416667	0.0025	1.25	0.75
12	25	1.41	0.975889898	3.22	68	19.5	0.35016631	191800	0.004	1.33333333	0.93333333	0.0075	0.00546667	0.004	2	1.2
12	25	1.41	0.975889898	3.22	68	19.5	0.36388909	191800	0.0025	0.83333333	0.58333333	0.012	0.00266667	0.0025	1.25	0.75
12	25	1.41	0.975889898	3.22	68	19.5	0.36042957	191800	0.00266667	0.88888889	0.62222222	0.01125	0.00444444	0.00266667	1.33333333	0.8

K9	K10	K11	K12	K13	K14	K15	K16	K17	K18	K19	K20	K21	K22	K23	K24	suction pressure
1.27272727	0.68181818	0.13636364	0.00548455	0.01143636	0.009090909	0.24783675	0.55326937	0.52359878	0.78539816	1	0.0019634	0.01538181	0.20151181	0.20763	0.2091	20402.2
1.17948718	0.76923077	0.15384615	0.00564103	0.01641026	0.01025641	0.24783675	0.55326937	0.52359878	0.78539816	1	0.0019634	0.01639377	0.23139377	0.241	0.235	23738.4
1.86666667	1	0.2	0.004	0.016	0.013333333	0.24783675	0.55326937	0.52359878	0.78539816	1	0.0019634	0.0182153	0.1682153	0.184	0.17	30307.6
1.20430108	0.64516129	0.12903226	0.00258065	0.0172043	0.008602151	0.24783675	0.55326937	0.52359878	0.78539816	1	0.0019634	0.01487583	0.24487583	0.26033333	0.25	18064.5
1.09090909	0.90909091	0.18181818	0.00666667	0.01939394	0.012121212	0.24783675	0.55326937	0.52359878	0.78539816	1	0.0019634	0.01760813	0.23260813	0.233	0.23	27475.4
1.2	1	0.2	0.00733333	0.016	0.013333333	0.24783675	0.55326937	0.52359878	0.78539816	1	0.0019634	0.0182153	0.1932153	0.193	0.19	28483.1
1.86666667	1	0.2	0.00622222	0.02133333	0.013333333	0.24783675	0.55326937	0.52359878	0.78539816	1	0.0019634	0.0182153	0.22488197	0.22966667	0.2296666	28564.6
1.16666667	0.625	0.125	0.0025	0.01	0.008333333	0.24783675	0.55326937	0.52359878	0.78539816	1	0.0019634	0.01457224	0.16457224	0.173	0.17	16467.6
1.16666667	0.625	0.125	0.0025	0.01222222	0.008333333	0.24783675	0.55326937	0.52359878	0.78539816	1	0.0019634	0.01457224	0.19123891	0.21066667	0.1996666	17010
1.43589744	0.76923077	0.15384615	0.00564103	0.01641026	0.01025641	0.24783675	0.55326937	0.52359878	0.78539816	1	0.0019634	0.01639377	0.23139377	0.249	0.23	23487.4
1.2	1	0.2	0.004	0.016	0.013333333	0.24783675	0.55326937	0.52359878	0.78539816	1	0.0019634	0.0182153	0.1682153	0.179	0.16	29374.8
0.92307692	0.76923077	0.15384615	0.00564103	0.01230769	0.01025641	0.24783675	0.55326937	0.52359878	0.78539816	1	0.0019634	0.01639377	0.19139377	0.193	0.19	24143
0.77419355	0.64516129	0.12903226	0.00688172	0.0172043	0.008602151	0.24783675	0.55326937	0.52359878	0.78539816	1	0.0019634	0.01487583	0.29487583	0.30533333	0.29	18308.4
1.86666667	1	0.2	0.01066667	0.016	0.013333333	0.24783675	0.55326937	0.52359878	0.78539816	1	0.0019634	0.0182153	0.2182153	0.234	0.22	25229.9
0.96	0.8	0.16	0.0032	0.01706667	0.010666667	0.24783675	0.55326937	0.52359878	0.78539816	1	0.0019634	0.01669736	0.20669736	0.219	0.20	24425.5
0.92307692	0.76923077	0.15384615	0.00307692	0.02051282	0.01025641	0.24783675	0.55326937	0.52359878	0.78539816	1	0.0019634	0.01639377	0.24639377	0.248	0.24	23089.1
0.75	0.625	0.125	0.00527778	0.01	0.008333333	0.24783675	0.55326937	0.52359878	0.78539816	1	0.0019634	0.01457224	0.19790557	0.21233333	0.2013333	17477.2
1.2	1	0.2	0.004	0.02666667	0.013333333	0.24783675	0.55326937	0.52359878	0.78539816	1	0.0019634	0.0182153	0.2482153	0.259	0.24	28535
1.16666667	0.625	0.125	0.00666667	0.01666667	0.008333333	0.24783675	0.55326937	0.52359878	0.78539816	1	0.0019634	0.01457224	0.29457224	0.303	0.30	17236.8
1.86666667	1	0.2	0.01066667	0.02133333	0.013333333	0.24783675	0.55326937	0.52359878	0.78539816	1	0.0019634	0.0182153	0.2582153	0.263	0.26	24466.1
0.75	0.625	0.125	0.00458333	0.01333333	0.008333333	0.24783675	0.55326937	0.52359878	0.78539816	1	0.0019634	0.01457224	0.22957224	0.233	0.23	17908.4
0.8	0.66666667	0.13333333	0.00562963	0.01066667	0.008888889	0.24783675	0.55326937	0.52359878	0.78539816	1	0.0019634	0.01517942	0.19851275	0.20133333	0.2013333	19561.6

Figure 5.3: Section of constantly growing database

5.7 Behavior Model and Optimizer

The **machine learning component** of the pipeline is the experiment design and data-driven model. Currently, this is an in-house Software called AVL CAMEO™ which is used for modelling the component behavior based on the behavior model. The optimizer varies specific parameters in the behavior model in a way to reach the highest suction pressure within defined constraints.

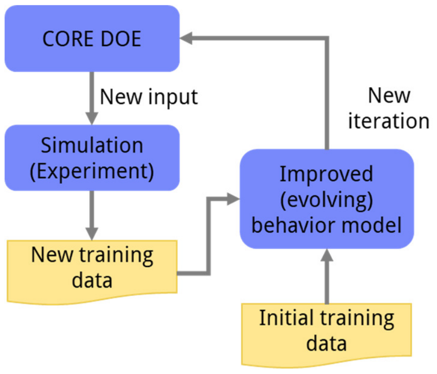


Figure 5.4: Training of DOE model [12]

The iteration process starts with an initial design of experiments (DoE), which is used to parameterize an initial model. In our case, a generic behavior model, which was trained on the data of 800+ simulations, is already existing. The target suction pressure can be modelled in different ways. The evolving model is trained with the ejector variants in the DOE database according to a non-linear or weighted least square algorithm (ELMN (evolving local model networks) training algorithm). The evolving local model network is extending the data-driven model based on the new incoming simulation results. Both the DoE and the data-driven model are subsequently extended in a way that relevant domains of the design space are explored. This process is termed custom output range exploration (CORE – relies on automatic adaptation technique and evolving scheme for data-driven models).

5.8 Measurables

Computational time: During several studies, the computational domain and mesh size was already improved in several steps regarding small cell number to reduce the numeric effort of the CFD simulation and to preserve physical accuracy of the result. Another already implemented step to minimize the computational time is the parallelization of the CFD simulation on a multiple core cluster.

Runtime: Comparing the runtime of all processes in the pipeline, the CFD simulation has the longest one with around 14 hours per simulation on 10 cores. Far behind is the behavior model and the optimizer. Depending on the conditions, building the behavior model(s) can take max. 30 seconds, running the optimizer can take up to a few minutes. All other process runtimes (pre-, post-processing, data management) are negligible small.

With our current approach, several loops (initial geometry estimation, CFD simulation to verify estimation, adding new data point to data set, rerun optimizer, CFD simulation…) are required to get a satisfying ejector geometry from the optimizer that meets the target suction pressure and packaging constraints. The goal is to reduce these loops to a minimum. In the context of the Daphne project, a sophisticated optimizer component, trained on the available dataset, is expected with enhanced prediction quality to wider range of application considering a bigger number of design parameters. The desired benefit for our workflow is a smaller number of numerically expensive CFD Simulations for prediction verification.

6 Automotive Vehicle Development Case Study: Virtual Prototype Development (AVL 2)

AVL List proposes the so-called *Integrated and Open Development Platform* (IODP) for the continuous verification and validation of a product (vehicle) under development. A key concept is to systematically link vehicle simulation and tests to establish so-called *Virtual Prototypes*, which are functional representations of the vehicle under development. The systematic linking and management of involved information entities ensures traceability. Using this IODP approach (e.g., at an automotive OEM or supplier), highly structured data emerges that accurately represents the vehicle under development and its evolution along the vehicle development process (VDP). This approach is described in more detail in [13].

The schematic graph in Figure 6.1 shows the evolution of the exemplary *key performance indicator* KPI (also referred to as *characteristic value* CV) "fuel consumption". The KPI curve consists of many discrete data points. The KPI eventually must meet the KPI target (also referred to as *target characteristic value* TCV), which is defined by requirements. The IODP vision is to provide system maturity assessment competence at any point in time, i.e., to be able to check whether the KPI value of the current state of development meets the KPI target. Each data point *(Time/KPI)* on the curve is linked to the entities that were involved in its determination, e.g., the used simulation models and simulation model parameters. Each data point is also associated with a maturity, i.e., a quantitative measure of confidence of the KPI value. Maturity increases with time.

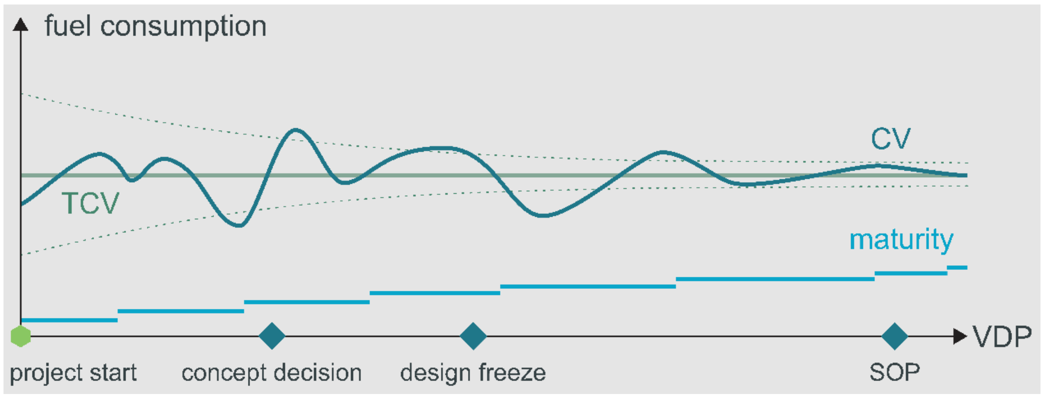


Figure 6.1: The exemplary characteristic value (CV) "fuel consumption" and its maturity evolve along the vehicle development process (VDP). The graph also shows the target characteristic value (TCV). [13]

AVL works with customers to implement this IODP approach in their organizations. This well-structured vehicle development process data is not yet available but will become available during the DAPHNE project with high probability. Once available, organizations will want to make use of it, apply data analytics and make inferences in order to optimize their vehicle development process. Of particular interest are the prediction of maturity for a given data point and of KPI evolution in ongoing development projects based on previous development projects. From AVL's perspective, it is sensible to already now start working on such process mining solutions. Development of these solutions involves both creating realistic artificial data and appropriate mining approaches. Real vehicle development processes consist of hundreds to thousands of different KPIs. Each KPI is evaluated many times during the development process. Each specific KPI evaluation is associated with a multitude of other entities (simulation models, parameters, bill of materials etc.). Thus, both creating artificial data as well as mining solutions need to be able to handle large amounts of data, so that questions of efficiency, computational performance, scalability etc. arise.

6.1 Overview of the Initial Pipeline

The initial pipeline for developing process mining approaches, specifically to predict KPI evolution and KPI maturity, is a first proof-of-concept. It creates artificial data for 7 exemplary KPIs (float values) based on 7 different simulation model parameters (float values / arrays), i.e., significantly fewer than in real-world processes. At the time of writing, no computational bottlenecks in terms of runtime, memory etc. have been encountered yet. However, bottlenecks are expected to appear when the data is scaled up to realistic dimensions in the future.

The initial pipeline can be divided into three sub-pipelines:

6.1.1 Data generation

The first step is to manually define the overall development process structure. A csv file (2D file containing float and string values) defines the stages of the mimicked process, the simulation model parameters (e.g., vehicle mass, wheel base, coefficient of aerodynamic drag) to be changed in each stage, the range of parameter variation and the number of different parameter values to be used. With this csv the simulation runs are initialized in the simulation environment Model.CONNECT^{TM} from AVL. Approx. 10,000 to 15,000 different simulation runs (i.e., tests of the virtual prototype) are created and executed. The simulation tool produces results for each run, e.g., the transient fuel consumption over time for a standard driving cycle. These intermediate results (csv) and the parameter values of each simulation run (xml) are input for the post-processing step (Python script). For each simulation run, it computes the KPI values (e.g., fuel consumption in L / 100 km) from the intermediate results. All simulation runs' KPI values, associated parameter values and process stage are stored in a JSON format file. The JSON is hierarchically structured and consists of nested JSON objects: Process stage objects contain simulation run objects contain KPI objects and parameter objects. Each simulation run may correspond to a data point in the KPI-over-time curve (see section 6.2.1). The time-series-creation step (Python script) breaks each process stage in the JSON down into individual parameter-KPI-tuples, puts these tuples into a temporal order (equidistant time steps) and stores the ordered tuples as a JSON file. This sub-pipeline is executed multiple times. Each resulting JSON file mimics an individual development project. All JSON files have the same hierarchically nested structure, but different numbers of objects on the process stage and simulation run levels.

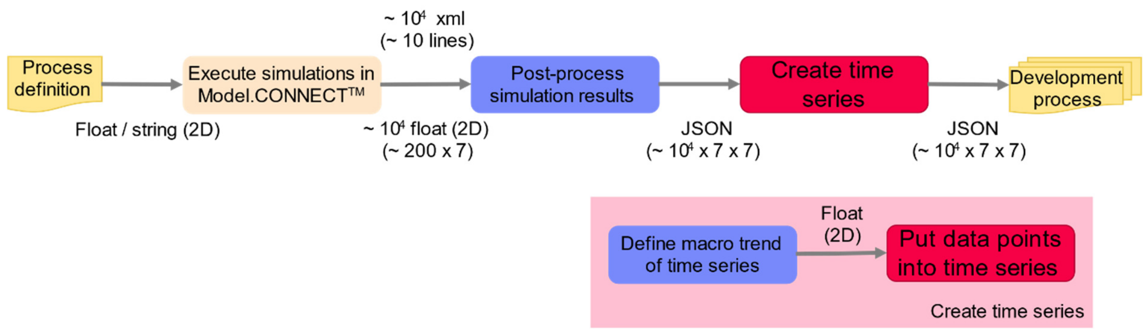


Figure 6.2: Sub-pipeline for generating the data used by subsequent pipeline steps.

6.1.2 Model Training

The created data sets / time series are split into two groups: Group 1 contains finished development projects and is used for model training. Group 2 contains test data sets representing ongoing projects, which are used for prediction (cf. 6.1.3). Gaussian process regression (GPR) [15] and its implementation in scikit-learn [1] is used to build models. Three separate models are trained (implemented as a Python script / Jupyter notebook):

- *Model A* learns the patterns in the time series of each parameter. It is used to predict parameter values over time.
- *Model B* learns the patterns among parameter values and KPI values, irrespective of temporal order.
- *Model C* is trained to predict the maturity of a given data point. The data points in the last process stage have the highest maturity, i.e., the highest confidence. Contrary to model B, model C is not trained with all data points in the training data sets, but only with the data points in the last process stage.

The target model KPI=f(time) required for predicting KPI evolution is intentionally split up into the models A and B. Having two separate models is more representative for realistic processes: Engineers decide what to do next and which parameters to change in order to improve the vehicle under development. This process is captured by model A. Based on these parameters, KPIs can be determined (e.g., using simulation), which is captured by model B.

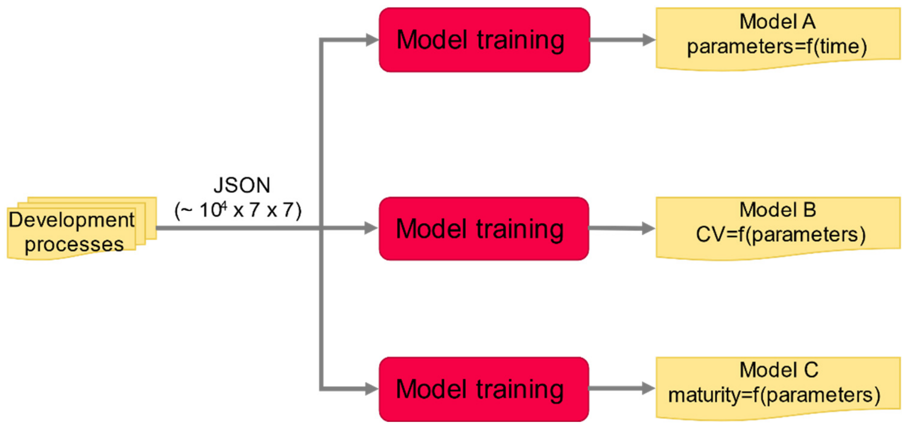


Figure 6.3: Sub-pipeline for the creation and training of models.

6.1.3 Prediction

Model A is used to predict the values of each parameter over time for the remainder of a data set from the test data sets. Using these predicted parameter values, model B is used to predict the KPI values for the remainder of data set. Model C is used to predict the standard deviation (i.e., a measure for the data point's maturity) of the data points in the data set. As for model training (cf. 6.1.2), the prediction uses the implementation of GPR in scikit-learn [1].

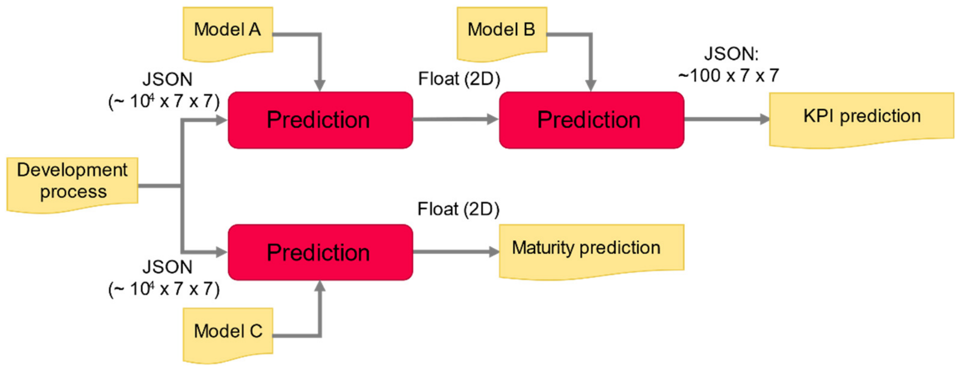


Figure 6.4: Sub-pipeline for predicting the development process.

6.2 Description of Elements

6.2.1 Training Data Generation

The *Create time series* element (cf. Figure 6.2) is implemented as a Python script, without using any special Python machine learning library. It chooses one (parameters/KPI) tuple randomly out of the JSON data set as the first data point of the time series. The trend of the time series shall be oscillatory and converging to a target KPI value (cf. Figure 6.1). The script computes the target KPI value from the JSON data set (random value close to data set mean). For each process stage object in the data set, the min and max KPI value are retrieved. The interpolation between these KPI values (first point, stage 1 min, stage 1 max, ..., stage n min, stage n max, target KPI value) defines the trend curve. The script then builds the time series from start to end by arranging a user-defined portion of the data points in the JSON data set (e.g., 80%) into a temporal structure, i.e., picking the best data point from the data set for a given position in the time series. "Best" is defined here by 2 criteria: (1) proximity of a data point's KPI value to the trend curve, and (2) proximity of a data point's parameters to the parameters of the previous data point in the time series.

6.2.2 Model Training

For training the GPR models, the algorithm for automatic kernel search proposed by D. Duvenaud [16] is used: Starting with an initial set of kernel functions (constant, linear, periodic, Matérn, squared exponential, rational quadratic, noise kernel), combinations (sums and products) of different kernel functions are iteratively created. (Combinations of) Kernels are evaluated using the Bayesian information criterion (BIC) score. The search for an optimal kernel stops after a predefined number of iterations, or if the BIC score no longer improves.

6.2.3 Prediction

A Python script (embedded into Jupyter notebook) using scikit-learn [1] (particularly its *predict* method) is used to evaluate the models A, B and C.

6.3 Measurables

The most important measurables for the data creation are (1) how realistically the KPI and parameter evolution are mimicked (assessed qualitatively by AVL experts) and (2) scalability, i.e., how runtime scales as more KPIs and parameters are added to the data generation process. For the models, the most important measurables are (1) accuracy, i.e., how precisely KPIs, parameters and maturity can be predicted, and (2) scalability, i.e., how model training and prediction runtime scale as more KPIs and parameters are added to the data sets.

7 References

[1] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and E. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research,* vol. 12, pp. 2825-2830, 2011.

[2] I. D. Stewart and T. R. Oke, "Local Climate Zones for Urban Temperature Studies," *Bulletin of the American Meteorological Society,* vol. 93, pp. 1879-1900, 12 2012.

[3] X. X. Zhu, J. Hu, C. Qiu, Y. Shi, J. Kang, L. Mou, H. Bagheri, M. Haberle, Y. Hua, R. Huang, L. Hughes, H. Li, Y. Sun, G. Zhang, S. Han, M. Schmitt and Y. Wang, "So2Sat LCZ42: A Benchmark Data Set for the Classification of Global Local Climate Zones [Software and Data Sets]," *IEEE Geoscience and Remote Sensing Magazine,* vol. 8, pp. 76-89, 9 2020.

[4] M. Schmitt, L. H. Hughes, C. Qiu and X. X. Zhu, "Aggregating cloud-free Sentinel-2 images with google earth engine," *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences,* Vols. IV-2/W7, pp. 145-152, 9 2019.

[5] B. Ko, "ResNet v2," 2018. [Online]. Available: https://github.com/kobiso/CBAM-keras/blob/master/models/resnet_v2.py. [Accessed 4 August 2021].

[6] L. Breiman, "Random forests," *Machine learning,* vol. 45, pp. 5-32, 2001.

[7] B. W. Matthews, "Comparison of the predicted and observed secondary structure of T4 phage lysozyme," *Biochimica et Biophysica Acta (BBA)-Protein Structure,* vol. 405, pp. 442-451, 1975.

[8] National Instruments, "The NI TDMS File format," 3 June 2021. [Online]. Available: https://www.ni.com/en-us/support/documentation/supplemental/06/the-ni-tdms-file-format.html. [Accessed 6 August 2021].

[9] D. H. Douglas and T. K. Peucker, "Algorithms for the Reduction of the Number of Points Required to Represent a Digitised Line or its Caricature," *The Canadian Cartographer,* vol. 10, pp. 112-122, 12 1973.

[10] M. Visvalingam and J. D. Whyatt, "Line generalisation by repeated elimination of the smallest area," 1992.

[11] A. Reeve, "npTDMS," 2012. [Online]. Available: https://pypi.org/project/npTDMS. [Accessed 6 August 2021].

[12] M. Deregnaucourt, M. Stadlbauer, C. Hametner, S. Jakubek and H.-M. Koegeler, "Evolving model architecture for custom output range exploration," *Mathematical and Computer Modelling of Dynamical Systems,* vol. 21, pp. 1-22, 2 2014.

[13] W. Puntigam, J. Zehetner, E. Lappano and D. Krems, "Integrated and Open Development Platform for the Automotive Industry," in *Systems Engineering for Automotive Powertrain Development*, Springer International Publishing, 2020.

[14] C. E. Rasmussen and C. K. I. Williams, Gaussian Processes for Machine Learning, The MIT Press, 2006.

[15] D. Duvenaud, J. Lloyd, R. Grosse, J. Tenenbaum and G. Zoubin, "Structure Discovery in Nonparametric Regression through Compositional Kernel Search," in *Proceedings of the 30th International Conference on Machine Learning*, Atlanta, 2013.