# Whitepaper
# Process considerations: A reliable AI data labeling process

This paper was written as part of a spotlight project of the openGENESIS working group at the Eclipse Foundation in a cooperation between TÜV SÜD Auto Service GmbH and Incenda AI GmbH.

Matthis Eicher, TÜV SÜD Auto Service GmbH

Patrick Scharpfenecker, TÜV SÜD Auto Service GmbH

Dieter Ludwig, TÜV SÜD Auto Service GmbH

Felix Friedmann, Incenda AI GmbH

Florian Netter, Incenda AI GmbH

Marius Reuther, Incenda AI GmbH

## Abstract

Due to its wide success in recent years, Artificial Intelligence (AI) is being used in more and more systems. As established Software Engineering practices, including development processes, fail to capture the complexity and additional challenges of developing AI systems, many Software Developers struggle using AI, especially in safety critical areas like healthcare or automotive.

One of the AI methods with the highest impact so far is supervised Machine Learning (ML). The performance of supervised ML is determined to a large extent by the data used to train and evaluate the developed models and the application of established Software Engineering practices[33]. Common issues include data and label quality, immature frameworks and processes for supervised ML development, a lack of traceability of requirements to implementation and limited transparency of some models.

The contribution of this whitepaper is the discussion and establishment of a sound supervised ML lifecycle, with a focus on data quality, from the *intent* of developing a system using supervised ML to the *decommissioning* of the developed system. The different steps of the lifecycle are detailed and a deep-dive into the labeling steps is provided by defining a labeling process. The discussion includes activities that are recommended to be performed in order to create high quality labels and raises typical issues during labeling.

Version V1.3 – August 2020

# Contents

## 1 Introduction

Machine Learning – Deep Learning in particular – has enabled numerous commercial applications in recent years. Deep Learning algorithms have outperformed humans and have made entirely new technologies like autonomous driving, simultaneous translation and art synthesis possible.

With the rise of Deep Learning, concerns about its reliability and interpretability are raised in increasing frequency. Common concerns cover the topics **robustness against attacks**[i], **online confidence metrics**[ii], **interpretability**[iii] and **labeling quality**.

Since the performance of trained models highly depends on the data sets they are optimized against, the elicitation of high-quality ground truth data, particularly **labeling** (i.e. creating training targets to optimize the models against) is a key activity within machine learning development. Labeling is a substantial effort and a major challenge within machine learning development[12–14].

In contrast to the premise of fast training of highly capable models, the machine learning development **lifecycle** and corresponding **processes** are by far not as established as the development of conventional software and the dependency on training data requires enormous additional workloads[11].

A recent case study[33] evaluates the applied processes in Microsoft related to artificial intelligence. The authors collect information from management and developers using a questionnaire. The results are used to identify common practices and to extract a common workflow process for AI development. On the other hand, CRISP-DM[34] is an established high-level cross-industry standard for data-mining. This standard defines an iterative process for understanding, processing and applying insights based on data.

This whitepaper establishes a machine learning lifecycle and conducts a thorough assessment of the labeling process within machine learning development and derives crucial considerations for the elicitation of reliable ground truth data.

## 2 Approach to establish a lifecycle and labeling process

Within this work, a process and its necessary activities were developed to avoid quality weaknesses and errors as good as possible, with a focus on the phase of generating labels for training and test data for supervised machine learning. Several steps and analyses were carried out in order to achieve this objective, as follows.

In a first step in section 4 *Machine Learning Lifecycle*, the lifecycle of a typical development of a function based on supervised machine learning was drafted, while the focus was put on the

---

[i] Researchers have demonstrated that Deep Neural Networks (DNNs) can be fooled by various ("adversarial") attacks[1], including changes to single image pixels[2] and attacks in the physical world[3,4]. Consequently, counter measures against adversarial attacks were investigated[5,6].

[ii] Common DNN types, like Convolutional Neural Networks (CNNs), do not expose a reliable metric for their confidence. Uncertainty measures are crucial for designing reliable systems that react on low confidence e.g. by system performance degradation. Extensions to CNNs that overcome this fundamental shortcoming were investigated with promising results[7,8].

[iii] DNNs typically consist of millions of parameters that are not intuitively understandable for humans. Consequently, the failure of a DNN can usually not be traced back to a cause, which drastically decreases debuggability and possibilities for safety analysis. Research exploring this domain ("explainable AI") has yielded useful methods[9,10] that significantly improve interpretability, typically via visual representations.

specification phase. This phase is especially characterized by the specification, collection, preparation and provision of the required data. A very important part of this phase is the process of labeling, which is examined in more detail.

After the definition of the lifecycle, a first draft for the labeling process was derived from it and elaborated in the section 5 *Labeling Process*. For the graphical representation of this process it was decided to use an activity-based flowchart, which was drawn up according to a BPMN[28]notation. The BPMN notation was chosen because its clear definition allows an unambiguous graphical description.

Based on this input, a process analysis was performed using a Process FMEA according to VDA Volume 4[29]. During this analysis, the process definition and the process analysis were iterated repeatedly, thus the original draft of the process description was continuously improved. As soon as the process description was stable, it was detailed with regard to process specific descriptions such as the purpose and activities for each individual step.

This procedure ensures that all possible weaknesses in a labeling process are systematically considered and that appropriate improvement measures can be taken.

# 3 Background

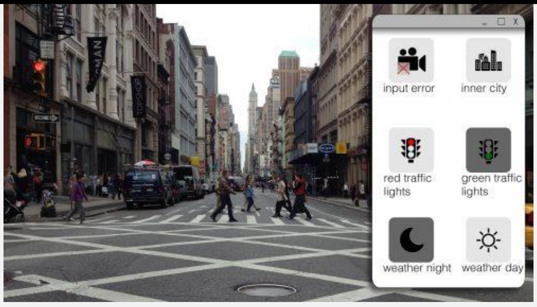
## 3.1 Labeling

Many commercial applications of Deep Learning are based on Supervised Learning, an approach that uses labels. **Labels** are annotations that provide a target to compare the output of trained models with, in order to derive an error value that can be used to update the model's parameters. These annotations are specific to the model's application (e.g. object detection, semantic segmentation) and are typically created manually or semi-automated, causing enormous cost and effort. The extent, representativity and quality of these annotations is decisive for the performance of the trained models. Additionally, these aspects are fundamental to validate the model's performance; a process for which a part of the labeled data ("validation split" and "test split") is used.

## 3.2 Common pattern recognition tasks on visual data

There is a variety of pattern recognition tasks requiring labels that power real-world applications. To illustrate the variety and to give an impression of the practical applications, some of the most common tasks are described below:

**Image classification**



Assignment of a set of non-exclusive labels to an image (i.e. Multi-Label classification). Example: for the image above the following conditions apply: red traffic light present, sunny, urban.

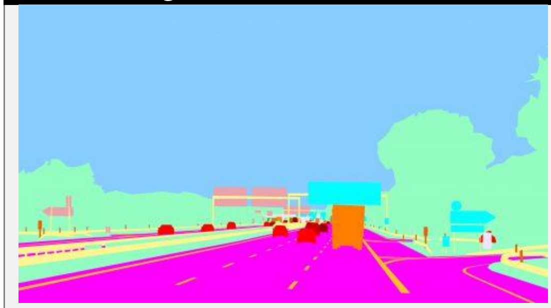
**Label:** List of applicable tags
**Dataset:** e.g. Tencent ML-Images[15]

**Object detection**

Detection of objects according to a pre-defined set of classes. E.g. detection of cars, trucks and pedestrians.

**Label:** 2d bounding boxes
**Dataset:** e.g. Kitti[16]

**Semantic segmentation**



Assignment of a class label to every pixel in an image.

**Label:** 2D class map
**Dataset:** e.g. Kitti[17]

**Pose estimation**

Estimation of the pose of a human body. This is typically achieved by predicting the position of the joints of the body.

**Label:** List of joints per person
**Dataset:** e.g. COCO Keypoint Detection[18]

The tasks outlined above are particularly common but are just an excerpt. Further tasks include: Instance-aware segmentation[19], panoptic segmentation[20], point cloud segmentation[21], lane detection[22], gaze estimation[23], pedestrian crossing intention[24] and many more.

## 4 Machine Learning Lifecycle

For more complex use-cases like multi-view object detection and classification, the use of machine learning based approaches can be necessary, hence classical rule-based algorithms can often hardly be implemented, generates huge efforts in the implementation or perform poorly.

For developing machine learning models, a process is highly recommended. The reference lifecycle model as developed and proposed by this work can be seen in *Figure 1*.

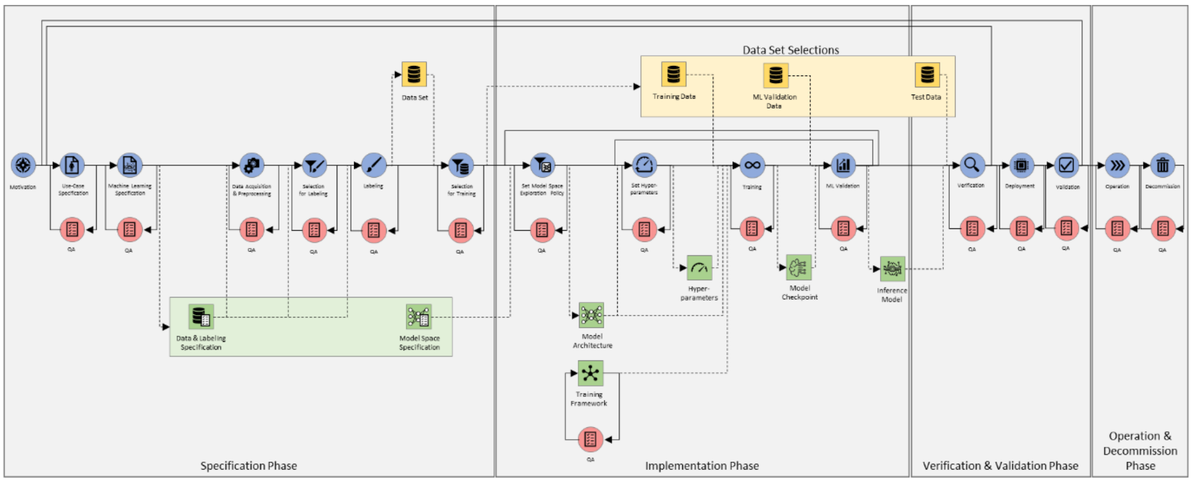


*Figure 1: Overview of a supervised machine learning lifecycle with focus on data (for better resolution see also Annex A ML Lifecycle Diagram)*

The diagram is centered around the lifecycle steps (in blue), from which general artifacts (like "Data & Labeling Specification", in green) or data sets (like "Training Data", in yellow) can be derived or inserted. In addition to this, the corresponding quality assurance measures are shown in red. The lifecycle is divided into four phases: specification, implementation, verification and validation, as well as operation and decommissioning.

In the following, the individual lifecycle steps will be described in detail.

## 4.1 Specification Phase

### 4.1.1 Motivation

Intention to provide a reliable, machine learning-based solution for a technical problem (e.g. driver assistance system).

### 4.1.2 Use-Case Specification

Description of the use-case for the machine learning-based solution to be developed. Should consider all environment conditions (lighting, environment variety, weather, etc.) that may have an impact on the performance of the solution to be developed[iv].

### 4.1.3 Machine Learning Specification

Specification of the machine learning approach to solve the described use-case, including:

- **Data & Labeling Specification**, considering:
    - Data of interest to be used for training and evaluation
        - should be representative for defined use-case, could contain e.g. class distribution.
        - should consider societal requirements[v] (ethics, laws, …) like being discrimination free.
    - Data format
        - Data should be acquired in the same format as it will be present for the deployed solution, e.g. sampling rate, resolution, sensor position, etc.

[iv] In the automotive industry, this includes the Operational Design Domain (ODD).

[v] This is a major, separate topic to consider[32] which will not be handled in this whitepaper. Note that these requirements also affect the training as the final model must, for example, be discrimination free. To support this, the data must be selected appropriately.

        - Labeling requirements
            - Description of labeling tasks to be carried out, e.g. semantic segmentation with defined set of classes.
            - Quality of labels, e.g. false positive rate, bounding box size precision, etc.
- **Model Space Specification**, considering
    - **Functional Requirements,** e.g. performance indications like confusion matrix
    - **Non-Functional Requirements**, considering
        - **Resource Constraints**, e.g. memory consumption, runtime
        - **Architectural Constraints**, e.g. recurrence, determinism, layer types/operations, hardware/accelerator

### 4.1.4 Data acquisition and preprocessing

The physical process of data acquisition in suitable environments and sources (e.g. sensors). Data acquisition should satisfy the requirements stated in the *Data & Labeling Specification*. Acquisition of new data may not be needed if suitable data already exists.

Includes preprocessing and cleansing of collected data to enable further usage and training. The preprocessing performed at this stage should be the same as is present for the deployed application. For example, if the data is normalized to zero mean and unit variance here, the same transformation must be applied for every datapoint the model is evaluated on.

### 4.1.5 Selection for Labeling

Selection of representative data for labeling according to *Data & Labeling Specification*. May contain multiple filtering steps, implementing:

- Subsampling, like spatial subsampling (limit number of samples per spatial region) and/or temporal subsampling (pick every n-th sample).
- Active Learning[25]: select data for labeling that is considered valuable for training according to an active learning algorithm, e.g. select data points containing class instances for which model performance is low.

### 4.1.6 Labeling

Creation of labels enabling training of supervised machine learning models according to *Data & Labeling Specification*.

Labeling can usually be automated to some degree (auto-labeling[26], weak supervision[27], data programming[31]), but the performance of current methods is not sufficient to replace manual effort entirely. Consequently, labeling typically requires a significant manual effort. E.g. the effort to perform instance segmentation for the COCO dataset was estimated with 10000s of man hours[14].

High quality labels are key factors for the performance and reliability of trained models. Despite the significant human effort, due to the enormous complexity and extent of the labeling activity, label quality is often a major challenge when creating high quality training datasets[13,14].

The labeling activity will be described in detail in *Labeling Process*.

### 4.1.7 Selection for Training

Filtering of all available labeled data towards a reduced training dataset. This is typically done by applying a series of filter conditions. The dataset is split into three parts:

- training subset used for learning the parameters of the model

- validation subset used for measuring the performance on data not used for training, so hyperparameter optimization (model selection, model parameters) or early stopping can be conducted
- test subset used for measuring the performance after training, to get an unbiased evaluation[vi] of a potential overfitting against the validation subset if the model and its parameters are chosen according to performance on the validation subset

The training data should be selected considering that it represents the intended use-case as well as that it contains important edge cases according to the *Data & Labeling Specification* and the increasing understanding of the occurring scenarios gained from data collection. It is vital for the performance of the machine learning model in the target environment that the data selected for training is representative. Consequently, this activity should be performed with the required rigor.

## 4.2 Implementation Phase

### 4.2.1 Set Model Space Exploration Policy

The model space is set by deriving hyperparameter constraints from the *Non-Functional Requirements* defined in the *Model Space Specification* and by setting the optimization approach.

Non-Functional Requirements may contain the following constraints:

- *Resource Constraints* define runtime and memory constraints that result in model and hyperparameter limits for training. For DNNs, this may affect the layer types, number of layers and number of parameters (weights).
- *Architectural Constraints* may limit model types and hyperparameters. For DNNs, this may affect layer types and quantization as they may not be supported by the target hardware. For example, use of recurrent DNNs and temporal memory may need to be avoided due to safety considerations.

The search space must be narrowed down to models and parameters satisfying the defined constraints.

The optimization approach should define how training is supposed to be conducted, considering the following aspects:

- Optimization method: reinforcement-learning based, grid search, or custom algorithm
- Sampling resolution for parameters: step size, granularity or sample limit
- **Model Space Exploration Stopping Criteria**: e.g. evaluation limit, performance threshold, time limit
- **Model Training Stopping Criteria**: e.g. epoch limit, performance threshold, time limit

### 4.2.2 Set Hyperparameters

While the *Set Model Space Exploration Policy* lifecycle step defines a generator for hyperparameter instances, the ***Set Hyperparameters*** step is the act of drawing hyperparameters from this generator. If parallelization is supported, multiple hyperparameter set instances can be used for parallel training, but this paper focusses on sequential trainings using single hyperparameter set instances for the sake of simplicity. This step includes the adaption of hyperparameters according to experience from previous iterations.

[vi] While this is the ideal case, in practice experiments often get modified and repeated if the outcome is not good enough. This leads to re-using the test data and therefore a potential optimization to this data.

### 4.2.3 Training

The automated process of learning the parameters (e.g. weights) of a trainable model (e.g. neural network) according to labeled training data to maximize the performance of the model on a given machine learning task (e.g. image classification). This activity is performed in accordance to the *Model Space Specification*.

Training of machine learning models typically requires large amounts of data, ranging from hundreds of ideally independent samples for well-constrained tasks in lab environments to millions of samples for complex tasks in the wild (e.g. autonomous driving).

This step also contains optimization of the trained model for the target hardware as it may affect compliance with the *Functional Requirements* (e.g. performance loss due to weight quantization) and *Non-Functional Requirements* (e.g. less memory used due to model compression) specified in the *Model Space Specification*.

### 4.2.4 ML Validation

Measurement of the performance of a trainable model on labeled **ML validation** data as part of the training in order to get a metric for its performance that can be used to measure progress during training, to adjust training hyperparameters (learning rate etc.) and to select the best model for the **Verification** activity. This metric also allows to detect overfitting[vii] by comparing the performance measured on training data vs. the performance measured on the **ML validation** data – in case the latter gets worse while the former further increases, the model is likely overfitting.

The ***Training*** and ***ML Validation*** steps can be referred to as the **Inner Training Loop**. Before entering the ***Verification* step** this **Inner Training Loop** is executed multiple times, until the *Model Training Stopping Criteria* are reached.

## 4.3 Verification- and Validation-Phase

### 4.3.1 Verification

Verification: “confirmation, through the provision of objective evidence, that specified requirements have been fulfilled” [SOURCE: ISO 9000:2005].

In the context of the development of ML-based applications, the **Verification** activity includes measuring the functional performance of trained models on “test data”, which was never used for **Training** or **ML validation**, to confirm that the functional performance satisfies the requirements expressed in the **Model space specification**.

This activity happens after **ML validation** to verify the model did not overfit against the **ML** validation data, which may happen if the hyperparameters of the model were chosen according to the maximum performance on the **ML validation** data, or if many models were trained and the one with best performance on the validation data was selected. A check with the “test data” can confirm whether overfitting occurred and provides a final performance metric that can be used to confirm that the performance threshold defined in the **Model space specification** is reached.

In case the training and target hardware environments differ, **Verification** results should be confirmed in the target environment to provide representative evidence for the deployment in the field. This is

[vii] During training, models may start to store (parts of) the input data instead of learning to generalize over (potential) input samples. This may cause a very low performance of the model when confronted with data only slightly different from the training data.

necessary because of potential differences in performance due to e.g. weight quantization, target precision, etc.

The **Inner Training** *Loop* together with the *Verification* step can be referred to as the **Outer Training Loop** and may also be executed multiple times, while every pass through the *Outer Training Loop* typically involves multiple passes through the *Inner Training Loop*. The *Model Space Exploration Stopping Criteria* define how often the *Outer Training Loop* is passed through and under which conditions it finishes by returning to the ***Set Hyperparameters*** step.

### 4.3.2 Deployment

Deployment of the ML-based application in the target real world environment or a representative simulated environment. This step includes system integration that connects all system modules (e.g. from sensors to actuators) and enables the observation of dependencies and resulting failures.

A practical example for deployment would be flashing of the software containing the trained model to the ECUs of a vehicle planned to be used for the *Validation* step.

### 4.3.3 Validation

Validation: "confirmation, through the provision of objective evidence, that the requirements for a specific intended use or application have been fulfilled" [SOURCE: ISO 9000:2005].

In the context of the development of ML-based applications, the **Validation** activity includes measuring the performance of a trained model under real-world conditions to confirm that the initial assumptions expressed in the **Use-Case Specification, Data & Labeling Specification** and the used **data** are valid.

If the observed performance is significantly lower compared to the results in **Verification**, the **Verification** setup may not have been representative for the application in the real world, e.g. because the labeled data used for verification was not collected from representative environments. In this case it may be required to go back to the **Use-Case Specification** step, update the **Data & Labeling Specification** accordingly and re-execute the required, subsequent steps.

## 4.4 Operation- and Decommissioning-Phase

### 4.4.1 Operation

While the application is operated in the field (e.g. robotaxi fleet driving in a major city), the following activities can be pursued:

- **Field observation**: monitoring of the behavior of ML-powered systems in the field. Tracking of the location, usage, and ownership in order to enable mitigation in case of severe issues (see below).
- **Maintenance/Updates**: Provisioning of software updates, particularly updates for the trained models. Updates enable ML-powered systems in the field to further increase in functional performance and to adapt to environment changes.
- **Online Learning**: Continuous training of the model in its real-world environment. Like weight updates, this enables performance increases in the field, but with the difference that the individual ML-powered systems can directly learn from their own experiences without waiting for training, release and updates from a backend. Consequently, the training process in the

individual ML-powered system should satisfy the same requirements[viii] imposed on the **Training** activity that happens in the **Implementation** phase.

In case systems in the field show critical performance issues, measures must be taken to prevent causing harm:

- **Notification**: Operators (e.g. remote control centers) should be informed about known issues.
- **Degradation**: The ML-powered systems should either degrade in performance themselves (e.g. autonomous cars reducing speed or capabilities), or limits should be imposed externally (e.g. disabling certain routes) in order to ensure safe operation.
- **Grounding**: In case of a severe potential for harm of people or business, the ML-powered applications should be grounded, by calling them back to their depot (in case of mobile robots), switching them off remotely, or by notifying their operators with grounding instructions.

### 4.4.2 Decommissioning

Permanent retirement of ML-powered systems. Owners of the systems should be informed sufficiently early, deactivation should be tracked (and enforced if necessary) and finally all observation activities can be stopped.

# 5 Labeling Process

The previously discussed *Machine Learning Lifecycle* covers the entire lifecycle of a supervised machine learning based system and focuses especially on the usage of data.

In the following, a reference process for the **Labeling** lifecycle step with integrated quality assurance will be described. This process is visualized by an activity-based flowchart shown in *Figure 2*.

In general, traceability should be established for reproducibility and trackability across the entire process. As soon as requirements, configurations or information are derived from or build on each other, this should be documented.

For the individual processes and process steps within the labeling lifecycle step, this paper will focus on the creation of the labels and will describe the purpose and activities of each process step. Due to the extent of this paper, further specifications of the process such as those of SIPOC[30] (Supplier, Input, Processing, Output, Customer), RASI[30] (Responsible, Accountable, Consulted, Informed) or similar are included only partly.

[viii] This might depend on the required integrity of the system.

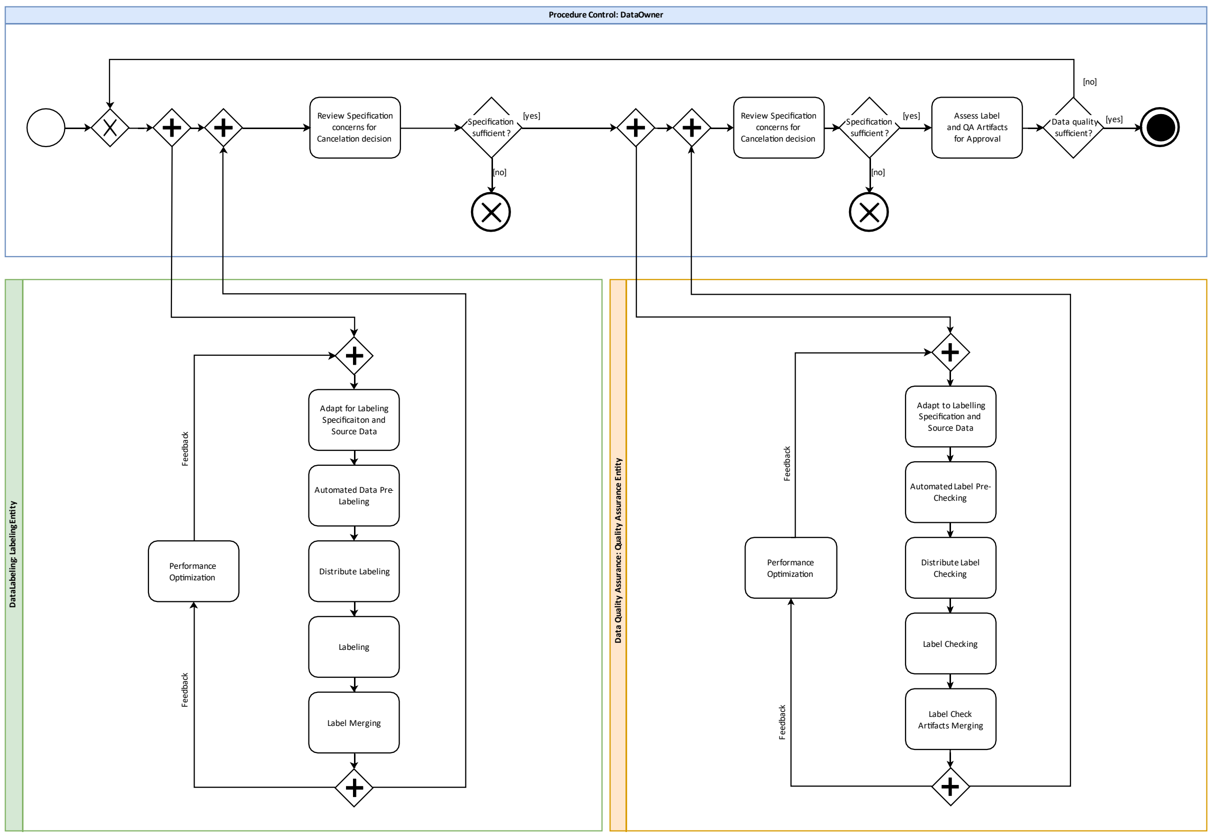

Procedure Control: DataOwner
Review Specification concerns for Cancelation decision
Specification sufficient ?
[yes]
[no]
Review Specification concerns for Cancelation decision
Specification sufficient ?
[yes]
[no]
Assess Label and QA Artifacts for Approval
Data quality sufficient?
[yes]
[no]
DataLabeling: Labeling Entity
Adapt for Labeling Specificaiton and Source Data
Automated Data Pre-Labeling
Distribute Labeling
Labeling
Label Merging
Performance Optimization
Feedback
Feedback
Data Quality Assurance: Quality Assurance Entity
Adapt to Labelling Specification and Source Data
Automated Label Pre-Checking
Distribute Label Checking
Label Checking
Label Check Artifacts Merging
Performance Optimization
Feedback
Feedback

*Figure 2: Labeling Process diagram*

## 5.1 Overview

### 5.1.1 Main Entities

The following roles are involved in or responsible for the different steps of the labeling process.

#### *Data Owner Entity*

- Owns the data and is responsible for the complete labeling procedure.
- Performs **Procedure Control**.
- Initiates and approves the artifacts from the label generation as well as the label quality check.
- Checks the progress and decides at given points about proceeding to the next process-step.

#### *Labeling Entity*

- Performs **Labeling**.
- Reports its progress and potential issues continuously as well as provides the generated labels to the Data Owner.

#### *Quality Assurance Entity*

- Performs **Data Quality Assurance**.
- Executes the task of label quality check.
- Reports progress and potential issues continuously and provides the identified label issues to the Data Owner.

The mentioned roles used here are generic and can be understood both as a person or an organizational entity.

### 5.1.2 Major Artifacts

The following artifacts are used in several process steps or are exchanged between different entities.

### *Labeling Specification*

The Labeling Specification specifies the requirements to the labeling task, e.g.

- labeling task and scope (classes, text, numerical values, …),
- format for the labels to be produced (boxes, points, … paired with class, text, …),
- accuracy targets and tolerances.

Potential Issues:

- **Imprecision** or **Ambiguity**, e.g. it has not been specified if pick-up trucks belong to the class of trucks or passenger cars.
- **Incompleteness**, e.g. a class of objects has been forgotten
- **Inconsistency**, e.g. including pick-up trucks in two classes, trucks and passenger cars.
- **Format**, e.g. objects are requested to be labeled as points while the learnt system is required produce bounding boxes.
- **Lack of precise quality requirements**, e.g. it should be specified how much a bounding box size is allowed to deviate from the actual object in the data.
- **Specification version mismatch,** e.g. data owner entity, labeling entity and quality assurance entity work on different versions of the labeling specification or the source data does not match to the labeling specification version.

### *Analysis Instructions*

The Analysis Instructions describe how the labeled data will be analyzed and which conditions should be satisfied, for example

- overlap of data between different Labelers,
- metrics for quality evaluation and
- thresholds for acceptable labeling quality.

### *Source Data*

The unlabeled data in a specified format (images with a given resolution, tabular data, text corpus, …) that should be labeled.

Potential Issues:

- **Corruption**, e.g. having random bit-flips due to some transmission process or by a faulty sensor
- **Swapping**, e.g. files are accidently swapped resulting in the wrong data being annotated, potentially biasing the dataset.
- **Inaptitude**, e.g. data being defective due to sensor or environment artifacts, e.g. overexposed images, insufficient camera resolution, raindrops/dirt/insects on the camera lens, etc.
- **Inappropriate**, e.g. for a product that should be deployed in an urban environment, but data shows only highway scenarios.

### *Labels*

The labels assigned to the Source Data, created by the Labeling Entity.

Potential Issues:

- **Misclassification**, e.g. a vehicle is incorrectly classified as a bicycle.
- **False Positives**, e.g. a bounding box was provided for an object that doesn’t actually exist in the scene.

- **False Negatives**, e.g. no bounding box was provided for a cyclist.
- **Shape Mismatch**, e.g. a segment or bounding box doesn't precisely enclose a cyclist.
- **Incorrect Re-identification**, e.g. if it is required to provide consistent IDs over time, but the labeled IDs are incorrect.
- **Incorrect Format**, e.g. labels don't comply with the specified format.

#### *Issue List*

A list of all issues occurring in the course of labeling (e.g. requirements being ambiguous). All entities can raise issues that are discussed with the data owner to improve the specifications or labeling process, if applicable.

Potential Issues:

- **Incompleteness**, i.e. no organized issue collection
- **Inaccuracy**, i.e. vague, inaccurate statements

## 5.2 Process Description

In the following an overview of the tasks performed by the three involved entities is given. A detailed description of the individual process steps can be found in *Annex B*.

### 5.2.1 Procedure Control

**Responsible:** Data Owner Entity

The general duties of the Procedure Control are:

- Be the interface to other lifecycle steps.
- Delegation of tasks within the Labeling lifecycle step.
- Coordination of the execution of the process steps by the involved parties.
- Monitor quality to detect potential issues and react to them.
- Decide if the generated data labels reach the specified quality for approval.
- Tracking and resolution of items in the issue list.

### 5.2.2 Labeling

**Responsible:** Labeling Entity

The general duties of the Labeling are:

- Perform the labeling and hand over resulting labels and related working products.
- Adapt the labeling setup according to incoming specifications and requirements.
- Raise issues immediately, e.g. specification ambiguities.

### 5.2.3 Data Quality Assurance

**Responsible:** Quality Assurance Entity

The general duties of the Data Quality Assurance are:

- Perform the data quality assurance and hand over the identified deviations.
- Adapt the data quality assurance setup according to incoming specifications and requirements.
- Provide detailed feedback on found labeling issues to the labeling entity and/or the data owner.
- Raise issues immediately.

# 6 Conclusion and Outlook

The presented **Machine Learning Lifecycle** and **Labeling Process** establish the foundations for the quality and reliability of supervised ML systems:

The ***Machine Learning Lifecycle*** provides a concise view on the development of machine learning-based systems, from the initial intent to their decommission. The lifecycle has a special focus on data quality – a crucial factor for successful development of machine learning-based systems that is not always considered sufficiently. The lifecycle shows how specifications that are generated early in the development process are being picked up by activities downstream, enabling a clear view on the propagation of errors via the specifications. It was also shown how data is produced and consumed as part of the lifecycle, derived from the specifications.

The ***Labeling Process*** shows how the key stakeholders should interact with each other to facilitate production of high-quality labeled data: The *Data Owner Entity* provides specifications, coordinates the process and approves the labels, the *Labeling Entity* produces labels according to the specification, and the *Quality Assurance Entity* checks them. Additionally, a rigorous monitoring of the process activities is highly recommended; issues should be collected and reviewed, correcting measures should be applied. To enable a detailed understanding of the process, frequent issues with key artifacts and a detailed description of the individual process steps (see *Annex B*) are provided.

The proposed approach should be applied in practice to observe and evaluate its effectiveness. The results can be used as feedback to refine the proposed approach in case weaknesses are found.

Analogous to the analysis of the labeling step performed here, the other lifecycle steps such as specifications, data acquisition & preprocessing, selection for labeling as well as for training can now be described and analyzed in detail. This will establish a complete understanding of all aspects of the machine learning lifecycle and the corresponding quality assurance.

A concise approach for the quality assurance of AI data is a fundamental necessity towards the maturity of the handling of AI and a precondition for safety-relevant systems, such as automated vehicles.

## 7 References

1. Nguyen, A., Yosinski, J. & Clune, J. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (2015). doi:10.1109/CVPR.2015.7298640

2. Su, J., Vargas, D. V. & Sakurai, K. One pixel attack for fooling deep neural networks. (2017).

3. Sitawarin, C., Bhagoji, A. N., Mosenia, A., Chiang, M. & Mittal, P. DARTS: Deceiving Autonomous Cars with Toxic Signs. (2018).

4. Kurakin, A., Goodfellow, I. J. & Bengio, S. Adversarial examples in the physical world. (2016).

5. Madry, A., Makelov, A., Schmidt, L., Tsipras, D. & Vladu, A. Towards Deep Learning Models Resistant to Adversarial Attacks. (2017).

6. Yuan, X., He, P., Zhu, Q. & Li, X. Adversarial Examples: Attacks and Defenses for Deep Learning. (2017).

7. Gal, Y. & Ghahramani, Z. Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. (2015).

8. Postels, J., Ferroni, F., Coskun, H., Navab, N. & Tombari, F. Sampling-free Epistemic Uncertainty Estimation Using Approximated Variance Propagation. (2019).

9. Yosinski, J., Clune, J., Nguyen, A., Fuchs, T. & Lipson, H. Understanding Neural Networks Through Deep Visualization. (2015).

10. Samek, W., Wiegand, T. & Müller, K.-R. Explainable Artificial Intelligence: Understanding, Visualizing and Interpreting Deep Learning Models. (2017).

11. Sculley, D. *et al.* Hidden technical debt in machine learning systems. in *Advances in Neural Information Processing Systems* (2015).

12. Roh, Y., Heo, G. & Whang, S. E. A Survey on Data Collection for Machine Learning: a Big Data -- AI Integration Perspective. (2018).

13. Everingham, M. *et al.* The Pascal Visual Object Classes Challenge: A Retrospective. *Int. J. Comput. Vis.* (2014). doi:10.1007/s11263-014-0733-5

14. Lin, T. Y. *et al.* Microsoft COCO: Common objects in context. in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (2014). doi:10.1007/978-3-319-10602-1_48

15. Wu, B. *et al.* Tencent ML-Images: A Large-Scale Multi-Label Image Database for Visual Representation Learning. (2019).

16. The KITTI Vision Benchmark Suite: Object Detection Evaluation 2012. Available at: http://www.cvlibs.net/datasets/kitti/eval_object.php?obj_benchmark=2d. (Accessed: 1st December 2019)

17. The KITTI Vision Benchmark Suite: Semantic Segmentation Evaluation. Available at: http://www.cvlibs.net/datasets/kitti/eval_semseg.php?benchmark=semantics2015. (Accessed: 1st December 2019)

18. COCO - Common Objects in Context - COCO 2019 Keypoint Detection Task. Available at: http://cocodataset.org/#keypoints-2019. (Accessed: 1st December 2019)

19. Li, Y., Qi, H., Dai, J., Ji, X. & Wei, Y. Fully convolutional instance-aware semantic segmentation.

in *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017* (2017). doi:10.1109/CVPR.2017.472

20. Li, Q., Arnab, A. & Torr, P. H. S. Weakly- and semi-supervised panoptic segmentation. in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (2018). doi:10.1007/978-3-030-01267-0_7

21. Qi, C. R., Su, H., Mo, K. & Guibas, L. J. PointNet: Deep learning on point sets for 3D classification and segmentation. in *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017* (2017). doi:10.1109/CVPR.2017.16

22. Neven, D., De Brabandere, B., Georgoulis, S., Proesmans, M. & Van Gool, L. Towards End-to-End Lane Detection: An Instance Segmentation Approach. in *IEEE Intelligent Vehicles Symposium, Proceedings* (2018). doi:10.1109/IVS.2018.8500547

23. Zhang, X., Sugano, Y., Fritz, M. & Bulling, A. Appearance-based gaze estimation in the wild. in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (2015). doi:10.1109/CVPR.2015.7299081

24. Evans, D. & Norman, P. Understanding pedestrians' road crossing decisions: An application of the theory of planned behaviour. *Health Educ. Res.* (1998). doi:10.1093/her/13.4.481-a

25. Active learning (machine learning) - Wikipedia. Available at: https://en.wikipedia.org/wiki/Active_learning_(machine_learning). (Accessed: 14th December 2019)

26. Zhuang, B., Liu, L., Li, Y., Shen, C. & Reid, I. Attend in groups: A weakly-supervised deep learning framework for learning from web data. in *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017* (2017). doi:10.1109/CVPR.2017.311

27. Zhou, Z. H. A brief introduction to weakly supervised learning. *National Science Review* (2018). doi:10.1093/nsr/nwx106

28. International Organization for Standardization. ISO/IEC 19510:2013: Information technology - Object Management Group Business Process Model and Notation. (2013).

29. Verband der Automobilindustrie. Volume 4 - Quality Assurance in the Process Landscape. (2009).

30. International Organization for Standardization. ISO 13053-2:2011: Quantitative methods in process improvement - Six Sigma - Part 2: Tools and techinques. (2011).

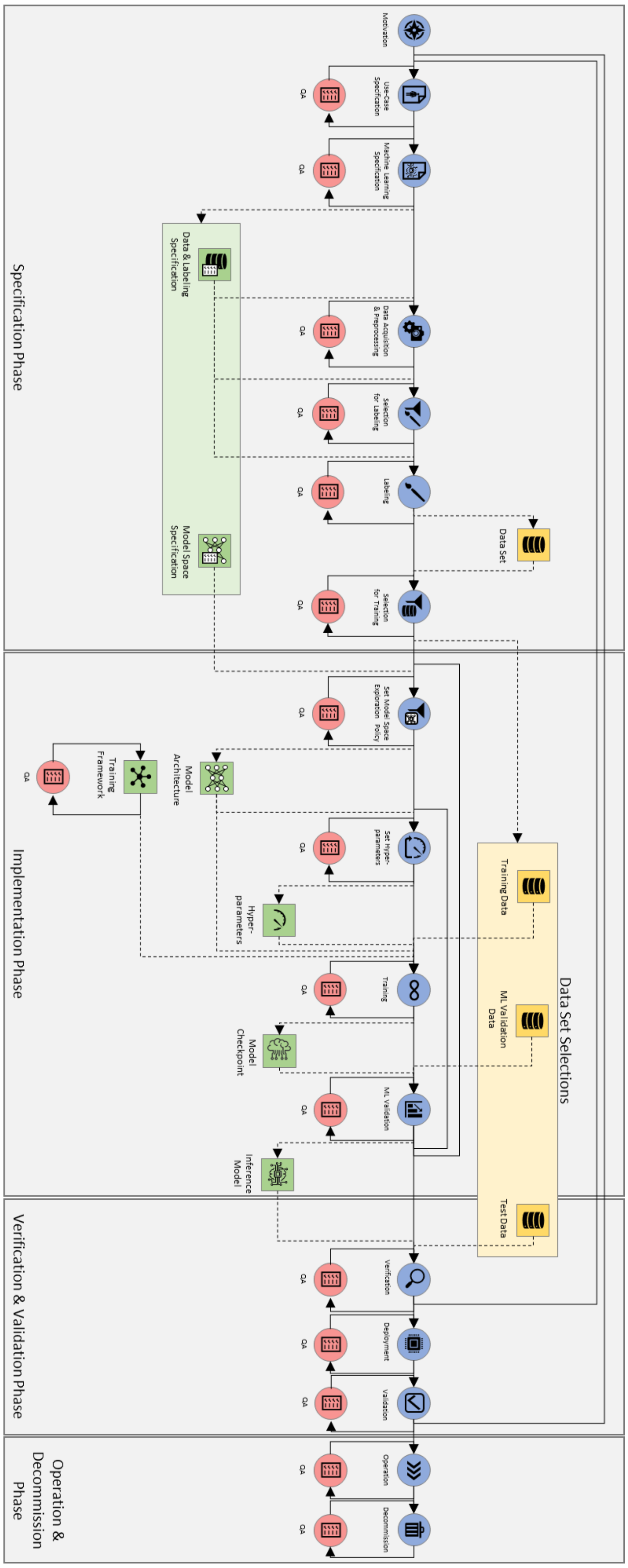
31. Ratner, A. J., De Sa, C. M., Wu, S., Selsam, D., & Ré, C. Data programming: Creating large training sets, quickly. In Advances in neural information processing systems (2016).

32. HEG-KI. Ethics guidelines for trustworthy AI. https://ec.europa.eu/digital-single-market/en/news/ethics-guidelines-trustworthy-ai (2019). (Accessed: 20.03.2020)

33. Amershi, S., Begel, A., Bird, C., DeLine, R., Gall, H., Kamar, E., Nagappan, N., Nushi, B., Zimmermann, T.. Software engineering for machine learning: a case study. In Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP '19). IEEE Press, 291–300 (2019).

34. Shearer, C. The CRISP-DM Model: The New Blueprint for Data Mining. Journal of Data Warehousing, 5 (2000).

# Annex A ML Lifecycle Diagram



# Annex B Detailed Process Description

This section contains a detailed description of all process steps for the process steps associated with Procedure Control, Labeling and Data Quality Assurance. A graphical representation of this process can be found in *Figure 2: Labeling Process diagram*.

The process steps will be described according to the following schema:

- Process Step Title (as headline)
    - Purpose
    - Inputs
    - Activities, each containing
        - Description
        - Outputs
        - Potential Issues (if applicable)
        - Examples (if applicable)
    - Resources (if applicable)
    - Tools (if applicable)

## Annex B.1 Procedure Control Process Steps

### 1 Review Specification concerns for Cancelation (after Labeling)

#### 1.1 Purpose

To evaluate if the Labeling Specification had any issues and to decide if the Labeling Specification needs a refinement before proceeding further.

#### 1.2 Inputs

##### 1.2.1 Labeling Specification

See *Labeling Specification*.

##### 1.2.2 Labeling Artifacts

All artifacts created by the Labeling Entity. Includes, for example, the Labels and the Issue List.

#### 1.3 Activities

##### 1.3.1 Review Concerns

**Description:** The Issue Analyst must analyze the source of the raised issues, which were documented in the Issue List during labeling. Based on this identification, suitable measures need to be taken.

If the Labeling Specification has weaknesses, the current process should be aborted, and the Labeling Specification should be fixed. After the correction of the Labeling Specification, the labeling can be repeated.

The earlier potential problems are found here, the more efficient and cost effective labeling can be performed. Therefore, an issue assessment should take place continuously.

If there are no issues left, the next tasks of label quality assurance can be entered.

**Outputs:** The decision whether the Labeling Specification must be updated or if the next process step can be entered.

### 1.4 Resources

#### 1.4.1 Issue Analyst

Performs the activity *Review Concerns*.

## 2 Review Specification concerns for Cancelation (after Label Checking)

### 2.1 Purpose

To evaluate if the Labeling Specification had any issues and to decide if the Labeling Specification needs a refinement before further proceed.

### 2.2 Inputs

#### 2.2.1 Labeling Specification

See *Labeling Specification*.

#### 2.2.2 QA Artifacts

All artifacts created by the Quality Assurance Entity. Includes, for example, the Issue List.

### 2.3 Activities

#### 2.3.1 Review Concerns

**Description:** The Issue Analyst must analyze the source of the raised issues, which were documented in the Issue List during the label checking. Based on this identification, suitable measures need to be taken.

If the Labeling Specification has weaknesses, the current process should be aborted, and the Labeling Specification should be fixed first. After the correction of the Labeling Specification, the labeling can be repeated.

The earlier potential problems are found here, the more efficient and cost effective labeling can be performed. Therefore, an issue assessment should take place continuously.

If there are no issues left, the next step of approval can be performed.

**Outputs:** The decision, if the Labeling Specification must be updated, or if the next process step can be entered.

### 2.4 Resources

#### 2.4.1 Issue Analyst

Performs the activity *Review Concerns*.

## 3 Assess Labels and QA Artifacts for Approval

### 3.1 Purpose

To implement a final quality gate for the activities performed in this lifecycle step.

### 3.2 Inputs

#### 3.2.1 Labeling Specification

See *Labeling Specification*.

#### 3.2.2 Labels

See *Labels*.

#### 3.2.3 QA Artifacts
The artifacts created during the process steps performed by the Quality Assurance Entity.

### 3.3 Activities

#### 3.3.1 Assess artifacts
**Description:** The Issue Analyst must analyze the source of the raised issues, which were documented in the Issue List during label checking. Based on this identification, suitable measures need to be taken.

If the issues are based on insufficient label quality, the labeling tasks need to be repeated with adoptions to the corresponding issues.

The earlier potential problems are found here, the more efficient and cost effective labeling can be performed. Therefore, an issue assessment should take place continuously.

If there are no or only acceptable (minor) issues left, the data and its labels can be approved and used in the next lifecycle steps.

**Outputs:** The decision, if either the data quality is sufficient, or this lifecycle steps has to start from the beginning to improve the label quality.

### 3.4 Resources

#### 3.4.1 Issue Analyst
Performs the activity *Assess artifacts*.

# Annex B.2 Labeling Process Steps

## 1 Adapt for Labeling Specification and Source Data (for Labeling)

### 1.1 Purpose
Prepares the setup for the further steps of this process. Implements the required modifications to adapt to the current labeling task.

### 1.2 Inputs

#### 1.2.1 Labeling Specification
See *Labeling Specification*.

#### 1.2.2 Source Data
See *Source Data*.

#### 1.2.3 Analysis Instructions
See *Analysis Instructions*.

#### 1.2.4 Labeler Availability
A list of Labelers with their availability and experience.

### 1.3 Activities

#### 1.3.1 Labeling Instruction Derivation
**Description:** The Analyst assesses the Labeling Specification and Source Data according to the Analysis Instructions and creates Labeling Instructions which will later be used by the Labelers to label the

Source Data. These instructions must specify tolerances, be non-contradictory, complete and unambiguous.

**Output:** Labeling Instructions

**Potential Issues:**

- The analyst could derive incorrect labeling instructions which lead to e.g.
    - Imprecise labels, for example, if the instructions specify “the bounding box should not be too big” instead of “the bounding box should be as small as possible while still containing the full object, with a tolerance of at most one pixel on each side of the box”.
    - Ambiguous labels, for example, if the instructions specify “label the classes vehicle and passenger car” instead of “label the classes vehicle without passenger car and passenger car”.
    - Incomplete labels, for example, if the instructions specify “label all vehicles into the classes passenger car and truck” while not specifying all possible vehicle classes or a class “other”.
    - Inconsistent labels, for example, if tolerances are not the same for different classes of the same target.

**Example:** Consider the following input:

- a dataset of 1000 images
- a Labeling Specification asking for minimal bounding boxes around every passenger car

The created Labeling Instructions could be:

- Provide a list of all passenger car (stationary and non-stationary, given a well-defined set of characteristics of passenger cars considering, for example, if trailers belong to the passenger car or not) in every image with a bounding box that is
    - as small as possible,
    - does not have one or more vehicle pixels outside the bounding box and is not more than one pixel per side larger than required to fulfill the second condition.

### 1.3.2 Pre-Labeling Configuration Parameters Derivation

**Description:** The Analyst assesses the Labeling Specification and Source Data according to the Analysis Instructions and creates the Pre-Labeling Configuration Parameters. These parameters instantiate the Pre-Labeling Tool to automatically perform an initial labeling of the data. The configuration must consider

- general parameters of the Pre-Labeling Tool,
- the concrete Labeling Specifications, e.g. parameters describing what to label and how to label it,
- the Analysis Instructions, e.g. tolerances,
- how well the labeling must be, and
- other criteria.

In the case the Pre-Labeling Tool uses randomness for example for training, to ensure reproducibility, the Pre-Labeling Configuration Parameters Derivation must produce the appropriate seeds.

**Output:** Pre-Labeling Configuration Parameters

**Potential Issues:**

- The analyst could derive incorrect configuration parameters which configure the Pre-Labeling Tool to mislabel the data, e.g. imprecise, ambiguous, incomplete, inconsistent labels (see *Labeling Instruction Derivation* for more details).

### 1.3.3 Distribution Instruction Derivation

**Description:** Based on the Source Data, Labeling Specification, Analysis Instructions and Labeler Availability, the Analyst derives a Distribution Instruction which states in how many subsets the data and/or labels will be sliced. If the Analysis Instructions specify some overlap between data subsets (to be used, for example, for validating the Labeler accuracy), this must be accounted for in the derived Distribution Instructions.

**Output:** Distribution Instructions

**Potential Issues:**

- The Analyst could derive inappropriate or incorrect Distribution Instructions, for example, by not including an overlap between the different labelers for cross-checking their performance (see following example).

**Example** Consider the following input:

- a dataset of 1000 images
- a Labeling Specification asking for minimal bounding boxes around every person
- instructions to cross-check 5% of the labeled images
- 5 available Labelers

The created Distribution Instructions could be:

- Create 5 random, non-overlapping subsets of 200 images each.
- Select another 50 random images.
- Assign every Labeler one of the 5 subsets of 200 images plus the 50 shared images.

The Distribution Instructions could also use a different method for creating an overlap between the data for every Labeler. If there are multiple labels to create, the distribution could additionally (or instead) distribute the target labels between the Labelers. The chosen distribution must be appropriate for the concrete instantiation of the label generation task.

## 1.4 Resources

### 1.4.1 Analyst

**Description:** The Analyst performs the activities *Labeling Instruction Derivation*, *Pre-Labeling Configuration Parameters Derivation* and *Distribution Instruction Derivation*.

**Potential Issues:**

- While assessing the Source Data and Labeling Specification according to Analysis Instructions, the Analyst could
    - Assess the wrong Source Data or wrong Labeling Specification.
    - Assess Source Data and Labeling Specification not according to the Analysis Instructions by being
        - imprecise (for example by not maintaining the required level of accuracy/quality).

        - incomplete (for example by not considering the given requirements).
- While producing the labeling instructions for the Annotator, the Analyst could produce incorrect labeling instructions.
- While producing the configuration parameters for the automated Pre-Labeling Tool, the Analyst could produce incorrect configuration parameters.

# 2 Automated Data Pre-Labeling

## 2.1 Purpose

Automatically prelabel data to get an initial labeling and improve/speed-up the following labeling task.

## 2.2 Inputs

### 2.2.1 Pre-Labeling Configuration Parameters

Configuration parameters of the Pre-Labeling Tool. This configuration might contain (hyper-) parameters like, for example,

- memory/runtime parameters (tolerances for tuning accuracy vs. runtime, …),
- algorithm parameters (“number of clusters”, thresholds, hyperparameters, models, …), and
- seeds (to ensure reproducibility in case the Pre-Labeling Tool uses randomness for retraining or prelabeling).

### 2.2.2 Source Data

See *Source Data*.

### 2.2.3 Data for Retraining

Labeled data to be used for retraining and improving the Pre-Labeling Tool. Usually, this input is empty in the first iteration of the labeling process, subsequent iterations can provide labeled data to improve the Pre-Labeling Tool and reduce further effort of the Labelers.

## 2.3 Activities

### 2.3.1 Configuration Parameter Setup

**Description:** The Tool Operator uses the Pre-Labeling Configuration Parameters and modifies the Configuration State of the Pre-Labeling Tool.

**Example:** Based on the Pre-Labeling Configuration Parameters, the Tool Operator sets the Pre-Labeling Tool to

- use a certain amount of memory for caching,
- use a defined classification threshold,
- apply non-maximum suppression,
- use a given number as randomness seed (usually only used for retraining).

### 2.3.2 Pre-Labeling Algorithm Retraining

**Description:** The Tool Operator uses labeled data from previous iterations of the Labeling process step to retrain the Pre-Labeling Tool and can improve its performance. This feedback-loop can be executed several times during the label generation phase. This activity is optional and might not be executed during the first iteration of the label generation phase. If it is executed, this will affect the quality of the *Pre-Labeling* activity.

**Potential Issues:**

- Known as Catastrophic Forgetting, neural networks can forget previously learnt information when re-training with new data. If the Pre-Labeling Algorithm uses a neural network, special care must be taken that the newly labeled data does not reduce performance on previous data.
- Like the problems arising while training the ML model under development, training the Pre-Labeling Algorithm must take care of over- and underfitting.
- If this activity is provided data or labels with reduced quality, this could lead to a negative feedback-loop by reducing the quality of the created pre-labels which could lower the quality of the labels. The resulting performance decrease could reinforce itself.

### 2.3.3 Pre-Labeling

**Description:** The Tool Operator executes the Pre-Labeling Tool, using the input Configuration State and a (potentially trained) State (see *Pre-Labeling Algorithm Retraining*), on the Source Data to get the Prelabels. The Prelabes represent an initial labeling of the data according to the Labeling Specification.

**Potential Issues:**

- The created Prelabels could be misleading the labelers. For example, if the algorithm, as opposed to the requirements, systematically creates separate bounding boxes around passenger cars and their connected trailers, the labelers could interpret this behavior as intended and thereby also mislabel trailers as separate entities.
- Too many issues may overwhelm the labeler, setting a low quality expectation for the labeler that leads to overlooking of small issues.
- The trained model could take decisions that are not covered by the requirements and may motivate human labelers to follow its decisions. Human labelers should not follow this example but should raise requirements issues.

**Output:** Prelabels

**Example:** Consider the task of labeling all vehicles in a set of images with bounding boxes and assigning the manufacturer as class. Initially, the Pre-Labeling Tool might only be able to identify (most) vehicles and create approximate bounding boxes around them without knowing the manufacturer. This preparation reduces the effort for manual labeling significantly. After some data has been manually labeled (bounding boxes are corrected and annotated with the correct manufacturer), the Pre-Labeling Tool (or the AI system under development which is supposed to perform this task in the end) can be trained using these labels to create better bounding boxes around vehicles and to assign an initial guess of the vehicle manufacturer. This further reduces the effort of consecutive iterations of labeling.

## 2.4 Resources

### 2.4.1 Tool Operator

The Tool Operator performs the activities *Configuration Parameter Setup*, *Pre-Labeling Algorithm Retraining* and *Pre-Labeling*.

## 2.5 Tools

### 2.5.1 Pre-Labeling Tool

The Pre-Labeling Tool performs an initial labeling of the source data. It might use a classical algorithm,

- the AI algorithm under development (in an iterative approach), or
- a different (more or less powerful) AI algorithm

to perform Pre-Labeling. The tool must be configured using a

- configuration state from some configuration space (e.g. hyperparameters) and
- a (potentially trained or empty) state (depending on if the Pre-Labeling Tool uses a classical or AI algorithm).

Potential Issues:

- The Pre-Labeling Tool might have issues common to software tools like reading, writing or processing data erroneously.
- It must be expected the Pre-Labeling Tool will not produce perfect labels.

# 3 Distribute Labeling

## 3.1 Purpose

Distribute the tasks to parallelize and speed-up the labeling or to allow cross-checks between the labels of different labelers.

## 3.2 Inputs

### 3.2.1 Prelabels

The initial labels for the Source Data created by the Pre-Labeling Tool.

### 3.2.2 Source Data

See *Source Data*.

### 3.2.3 Labeling Instructions

Instructions on what and how to label the given Source Data.

### 3.2.4 Distribution Instructions

Instructions on how the Source Data or label targets should be distributed to different labelers, e.g.

- are the subsets on data, on label targets (for example, one labels cars and the other pedestrians), or both,
- how the Source Data has to be sliced (consecutive data subsets, random subsets, based on scenarios, …)
- how many subsets should be created,
- how large should the subsets be,
- is an overlap between subsets required?

### 3.2.5 Labeler Qualification and Availability

A list of available labelers with additional information, e.g.

- capacity (potentially influencing the amount of data for a labeler),
- qualification/experience (can influence the cross-checking).

## 3.3 Activities

### 3.3.1 Slice Subsets

**Description:** Based on the Distribution Instructions, the Data Distributor slices the Source Data, Prelabels and Distribution Instructions into different subsets.

**Output:** Prelabel Subsets, Source Data Subsets, Labeling Instruction Subsets, Distribution Instructions (refined, includes the documentation of the performed assignment)

**Example:** Consider an input of 1000 images with bounding-boxes of passenger vehicles and trucks as pre-labels, four available labelers and the instruction to divide the labeling tasks based on the targets (passenger vehicles and trucks) and split the data in two slices. Then the data can be split in 500 images each, gets paired with one of the two labeling targets. The result are four subsets, each containing 500 images and the labeling instructions and pre-labels for one of the two targets

#### 3.3.2 Assign Subsets to Labeler

**Description:** Based on the Distribution Instructions and the Labeler Qualification and Availability, the Data Distributor derives the number of subsets (based on data and labeling target) and assigns these subsets to the labelers. This activity may consider criteria like,

- qualification, experience and previous performance (for example for cross-checking),
- labeling volume capacity and
- availability.

**Output:** Subset Mapping

#### 3.3.3 Distribute Data

**Description:** The Data Distributor distributes the subsets for Prelabels, Source Data and Labeling Instructions to the labelers defined by the Subset Mapping.

**Output:** For every Labeler, a Prelabel Subset, a Source Data Subset and a Labeling Instruction Subset.

### 3.4 Resources

#### 3.4.1 Data Distributor

The Data Distributor is responsible to distribute the data and/or labeling targets to different labelers.

# 4 Labeling

### 4.1 Purpose

Label the data to get labels for the source data which can be used for training the ML model.

### 4.2 Inputs

#### 4.2.1 Source Data Subset

The subset of data to be labeled by the Labeler.

#### 4.2.2 Labeling Instruction Subset

Instructions for the current labeler on what and how to label in the given Source Data Subset.

#### 4.2.3 Prelabels Subset

The Prelabels for the Source Data Subset and the Labeling Instruction Subset. Only needs to contain the label targets relevant for this labeler.

### 4.3 Activities

#### 4.3.1 Label Data

**Description:** Here, the labeler refines the Prelabels by labeling the provided Source Data Subset according to the Labeling Instruction Subset. This activity is highly dependent on

- the data to label (images, audio, tabular data, …),

- the target labels (boxes, classes, numerical values, text, …, or a combination thereof),
- the given prelabels (precision/recall of prelabels, accuracy, classes, …) and
- the labeling instructions (tolerances, …).

The created labels should also establish traceability of their creation. For example, information that should be collected (ideally in an automated way):

- current state of requirements and instructions
- versions of data, Prelabeling and Labeling Tool
- Labeler

**Output:** Labels (including traceability)

**Potential Issues:**

- The Labeler could
  - miss fixing pre-label errors (for example, misleading labels, see *Pre-Labeling*)
  - introduce new errors by either
    - breaking correct pre-labels or
    - by introducing new, wrong labels.
  - miss adding labels (which have also not been added by the pre-labeling).

**Example:** Given a set of audio files as data and automatically generated transcriptions of the spoken words in those files (prelabels), the labeler listens to every audio file and modifies the prelabels to exactly represent the spoken words. A challenge for the labeler might be that the prelabeling could not distinguish different speakers in the same audio file. The labeler would then have to, in correspondence to the labeling instructions, correct the transcribed text and assign the sentences to different speakers.

### 4.4 Resources

#### 4.4.1 Labeler

The Labeler executes the manual task of labeling a subset of the Source Data with a subset of the label targets, given the prelabels of this data and label target subset.

## 5 Label Merging

### 5.1 Purpose

Merge the labels of different labelers to get a consistent set of labels on the full source data, to possibly cross-check the labeler performance and to identify possible issues of the previous steps.

### 5.2 Inputs

#### 5.2.1 Label Subsets

The labels for the Source Data Subsets.

#### 5.2.2 Source Data Subsets

The subsets of data that have been labeled.

#### 5.2.3 Analysis Instructions

See *Analysis Instructions*.

#### 5.2.4 Distribution Instructions

Instructions on how the Source Data or label targets should be distributed to different labelers. For details, see *Distribution Instruction Derivation* and *Distribution Instructions*.

### 5.3 Activities

#### 5.3.1 Combine Label Subsets

**Description:** The Data Merger combines the Label Subsets based on the Distribution Instructions. The Distribution Instructions specify on how the labels must be merged (data subsets and/or label target subsets).

**Output:** Merged Labels (with possible inconsistencies)

#### 5.3.2 Identify Issues between Subsets

**Description:** The Data Merger analyzes the Merged Labels (with possible inconsistencies) based on the Analysis Instructions. The Analysis Instructions specify if and how the labels can be cross-checked between different subsets

- due to a possible overlap between labelers, or
- due to inconsistencies between target labels (for example, if one labeler classified the same area in an image as passenger car while a different labeler labeled it as truck).

**Output:** Merged Labels, Issue List

### 5.4 Resources

#### 5.4.1 Data Merger

The Data Merger is responsible to merge the data, labels and issues from different labelers.

## 6 Performance Optimization (for Labeling)

This step cannot only be started after the whole labeling process is finished but anywhere during its execution, whenever issues (in the generated prelabels, labels, specification, instructions, …) have been identified.

### 6.1 Purpose

To improve the whole labeling process and to identify/resolve issues found in previous steps.

### 6.2 Inputs

#### 6.2.1 Source Data

See *Source Data*.

#### 6.2.2 Labels

See *Labels*.

#### 6.2.3 Issue List

See *Issue List*.

### 6.3 Activities

#### 6.3.1 Provide labeled data to Pre-Labeling Tool

**Description** After the Labeling successfully generated labels, these labels can be used to re-train the Prelabeling Tool and improve its performance. This should reduce the further labeling effort and speed-up the whole labeling process.

**Output:** Data for Retraining

**Example** In a first loop the Automated Data Prelabeling did not have enough information to apply the Prelabeling Tool. Therefore, no prelabels have been created and the labeling process has been executed only on a relatively small subset of all examples (for example 1% of the data). Labeling this data required a lot of time as the labelers could not use any prelabels. After this small subset of data has been labeled, the Prelabeling Tool can be trained and used for the remaining 99% of the data.

To further improve the Prelabeling Tool and speed up the labeling process, the next iteration of the labeling process might only handle another 5% of the data, further improving the Prelabeling Tool. This can go on until all data is labeled and might continue even later, if new data is added.

#### 6.3.2 Management of Quality Anomalies

**Description:** This activity is applied for every issue identified in the Label Merging (or any other step of the labeling process). The Performance Optimizer has to consider/analyze the given issue and take the necessary steps to resolve this issue. This can include

- updating some specification, instruction, …
- fixing some problem of the data acquisition or some involved tool (prelabeling, labeling, …), or
- repeating some earlier process step (like a faulty labeling step, …).

**Output:** No dedicated output. Improves the quality of the label generation phase.

**Example:** Consider the speech-to-text problem detailed in the *Label Data* activity and assume the Analysis Instructions specified to add some overlap between different labels to cross-check their performance. While merging the labels, an issue with one target label was identified: while some labelers only transcribed the spoken text, some also identified the speakers (with a list like SpeakerA, SpeakerB, …). The Performance Optimizer traces this difference back to its origin. The result states that while the Labeling Tool supports this identification of the speaker, the Labeling Specification did not specify to do so. But as the overall task of the ML system under development requires to identify different speakers, the Labeling Specification must be updated, and all steps influenced by this must be repeated.

### 6.4 Resources

#### 6.4.1 Performance Optimizer

The Performance Optimizer is responsible to trigger the improvements of either the Prelabeling or any other step of the labeling process.

## Annex B.3 Data Quality Assurance Steps

The process phase is structured similar as the *Labeling* and includes the following:

- Adapt to Labeling Specification and Source Data (for Label Checking)
- Automated Label Pre-Checking
- Distribute Label Checking
- Label Checking
- Label Check Artifacts Merging
- Performance Optimization (for Label Checking)

Due to the similarity to *Labeling*, no detailed description of the individual steps of *Data Quality Assurance* is provided.